pythonwhat Documentation

Release 2.23.0

DataCamp

Jan 30, 2020

Glossary

For an introduction to SCTs and how they use pythonwhat, visit the README.

This documentation features:

- A glossary with typical use-cases and corresponding SCT constructs.
- Reference documentation of all actively maintained pythonwhat functions.
- A set of basic and advanced articles that gradually expose you to all of pythonwhat's functionality and best practices.

If you are new to writing SCTs for Python exercises, start with the tutorial and work your way through the other basic articles. The glossary is good to get a quick overview of how all functions play together after you have a basic understanding. The reference docs become useful when you grasp all concepts and want to look up details on how to call certain functions and specify custom feedback messages.

CHAPTER 1

Glossary

This article lists some example solutions. For each of these solutions, an SCT is included, as well as some example student submissions that would pass and fail. In all of these, a submission that is identical to the solution will pass.

Note: These SCT examples are not golden bullets that are perfect for your situation. Depending on the exercise, you may want to focus on certain parts of a statement, or be more accepting for different alternative answers.

1.1 Check object

```
# solution
x = 10

# sct
Ex().check_object('x').has_equal_value()

# passing submissions
x = 5 + 5
x = 6 + 4
y = 4; x = y + 6
```

1.2 Check function call

```
# solution
import pandas as pd
pd.DataFrame([1, 2, 3], columns=['a'])

# sct
Ex().check_function('pandas.DataFrame')\
```

(continues on next page)

(continued from previous page)

```
    .multi(
        check_args('data').has_equal_value(),
        check_args('columns').has_equal_value()
    )

# passing submissions
pd.DataFrame([1, 1+1, 3], columns=['a'])
pd.DataFrame(data=[1, 2, 3], columns=['a'])
pd.DataFrame(columns=['a'], data=[1, 2, 3])
```

1.3 Check pandas chain (1)

```
# solution
import pandas as pd
df = pd.DataFrame([1, 2, 3], columns=['a'])
df.a.sum()

# sct
Ex().check_function("df.a.sum").has_equal_value()
```

1.4 Check pandas chain (2)

```
# pec
import pandas as pd
df = pd.DataFrame({'a': [1, 2, 3], 'b': ['x', 'x', 'y']})

# solution
df.groupby('b').sum()

# sct
sig = sig_from_obj("df.groupby('b').sum")
Ex().check_correct(
    # check if group by works
    check_function("df.groupby.sum", signature = sig).has_equal_value(),
    # check if group_by called correctly
    check_function("df.groupby").check_correct(
        has_equal_value(func = lambda x,y: x.keys == y.keys),
        check_args(0).has_equal_value()
    )
)

# passing submissions
df.groupby('b').sum()
df.groupby(['b']).sum()

# failing submissions
df               # Did you call df.groupby()?
df.groupby('a')  # arg of groupby is incorrect
df.groupby('b')  # did you call df.groupby.sum()?
```

1.5 Check pandas plotting

```
# pec
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
np.random.seed(42)
df = pd.DataFrame({'val': np.random.rand(300) })

# solution
df.val.plot(kind='hist')
plt.title('my plot')
plt.show()
plt.clf()

# sct
Ex().check_or(
    multi(
        check_function('df.val.plot').check_args('kind').has_equal_value(),
        check_function('matplotlib.pyplot.title').check_args(0).has_equal_value()
    ),
    override("df.val.plot(kind='hist', title='my plot')").check_function('df.val.plot
↪').multi(
        check_args('kind').has_equal_value(),
        check_args('title').has_equal_value()
    ),
    override("df['val'].plot(kind = 'hist'); plt.title('my plot')").multi(
        check_function('df.plot').check_args('kind').has_equal_value(),
        check_function('matplotlib.pyplot.title').check_args(0).has_equal_value()
    ),
    override("df['val'].plot(kind='hist', title='my plot')").check_function('df.plot
↪').multi(
        check_args('kind').has_equal_value(),
        check_args('title').has_equal_value()
    )
)
Ex().check_function('matplotlib.pyplot.show')
Ex().check_function('matplotlib.pyplot.clf')
```

1.6 Check object created through function call

```
# pec
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6])

# solution
result = np.mean(arr)

# sct
Ex().check_correct(
    check_object("result").has_equal_value(),
    check_function("numpy.mean").check_args("a").has_equal_value()
)
```

(continues on next page)

(continued from previous page)

```
# passing submissions
result = np.mean(arr)
result = np.sum(arr) / arr.size
```

1.7 Check DataFrame

```
# solution
import pandas as pd
my_df = pd.DataFrame({"a": [1, 2, 3], "b": [4, 5, 6]})

# sct
Ex().check_df("my_df").check_keys("a").has_equal_value()

# passing submissions
my_df = pd.DataFrame({"a": [1, 1 + 1, 3], "b": [4, 5, 6]})
my_df = pd.DataFrame({"b": [4, 5,  6], "a": [1, 2, 3]})
```

1.8 Check printout

```
# solution
x = 3
print(x)

# sct
Ex().has_printout(0)

# passing submissions
print(3)
print(1 + 1)
x = 4; print(x - 1)
```

1.9 Check output

```
# solution
print("This is weird stuff")

# sct
Ex().has_output(r"This is \w* stuff")

# passing submissions
print("This is weird stuff")
print("This is fancy stuff")
print("This is cool stuff")

# failing submissions
print("this is weird stuff")
print("Thisis weird stuff")
```

1.10 Check Multiple Choice

```
# solution (implicit)
# 3 is the correct answer

# sct
Ex().has_chosen(correct = 3, # 1-base indexed
                msgs = ["That's someone who makes soups.",
                        "That's a clown who likes burgers.",
                        "Correct! Head over to the next exercise!"])
```

1.11 Check import

See has_import doc

1.12 Check if statement

See check_if_else doc

1.13 Check function definition

See check_function_def doc

1.14 Check list comprehensions

See check_list_comp doc

CHAPTER 2

Reference

Note:

- `check_` functions typically 'dive' deeper into a part of the state it was passed. They are typically chained for further checking.
- `has_` functions always return the state that they were intially passed and are used at the 'end' of a chain.

2.1 Objects

check_object (*state*, *index*, *missing_msg=None*, *expand_msg=None*, *typestr='variable'*)

Check object existence (and equality)

Check whether an object is defined in the student's process, and zoom in on its value in both student and solution process to inspect quality (with has_equal_value().

In `pythonbackend`, both the student's submission as well as the solution code are executed, in separate processes. `check_object()` looks at these processes and checks if the referenced object is available in the student process. Next, you can use `has_equal_value()` to check whether the objects in the student and solution process correspond.

Parameters

- **index** (*str*) – the name of the object which value has to be checked.
- **missing_msg** (*str*) – feedback message when the object is not defined in the student process.
- **expand_msg** (*str*) – If specified, this overrides any messages that are prepended by previous SCT chains.

Example Suppose you want the student to create a variable `x`, equal to 15:

```
x = 15
```

The following SCT will verify this:

```
Ex().check_object("x").has_equal_value()
```

- `check_object()` will check if the variable `x` is defined in the student process.
- `has_equal_value()` will check whether the value of `x` in the solution process is the same as in the student process.

Note that `has_equal_value()` only looks at **end result** of a variable in the student process. In the example, how the object `x` came about in the student's submission, does not matter. This means that all of the following submission will also pass the above SCT:

```
x = 15
x = 12 + 3
x = 3; x += 12
```

Example As the previous example mentioned, `has_equal_value()` only looks at the **end result**. If your exercise is first initializing and object and further down the script is updating the object, you can only look at the final value!

Suppose you want the student to initialize and populate a list *my_list* as follows:

```
my_list = []
for i in range(20):
    if i % 3 == 0:
        my_list.append(i)
```

There is no robust way to verify whether *my_list = [0]* was coded correctly in a separate way. The best SCT would look something like this:

```
msg = "Have you correctly initialized `my_list`?"
Ex().check_correct(
    check_object('my_list').has_equal_value(),
    multi(
        # check initialization: [] or list()
        check_or(
            has_equal_ast(code = "[]", incorrect_msg = msg),
            check_function('list')
        ),
        check_for_loop().multi(
            check_iter().has_equal_value(),
            check_body().check_if_else().multi(
                check_test().multi(
                    set_context(2).has_equal_value(),
                    set_context(3).has_equal_value()
                ),
                check_body().set_context(3).\
                    set_env(my_list = [0]).\
                    has_equal_value(name = 'my_list')
            )
        )
    )
)
```

- `check_correct()` is used to robustly check whether `my_list` was built correctly.
- If `my_list` is not correct, **both** the initialization and the population code are checked.

Example Because checking object correctness incorrectly is such a common misconception, we're adding another example:

```
import pandas as pd
df = pd.DataFrame({'a': [1, 2, 3], 'b': [4, 5, 6]})
df['c'] = [7, 8, 9]
```

The following SCT would be **wrong**, as it does not factor in the possibility that the 'add column `c`' step could've been wrong:

```
Ex().check_correct(
    check_object('df').has_equal_value(),
    check_function('pandas.DataFrame').check_args(0).has_equal_value()
)
```

The following SCT would be better, as it is specific to the steps:

```
# verify the df = pd.DataFrame(...) step
Ex().check_correct(
    check_df('df').multi(
        check_keys('a').has_equal_value(),
        check_keys('b').has_equal_value()
    ),
    check_function('pandas.DataFrame').check_args(0).has_equal_value()
)

# verify the df['c'] = [...] step
Ex().check_df('df').check_keys('c').has_equal_value()
```

Example pythonwhat compares the objects in the student and solution process with the == operator. For basic objects, this == is operator is properly implemented, so that the objects can be effectively compared. For more complex objects that are produced by third-party packages, however, it's possible that this equality operator is not implemented in a way you'd expect. Often, for these object types the == will compare the actual object instances:

```
# pre exercise code
class Number():
    def __init__(self, n):
        self.n = n

# solution
x = Number(1)

# sct that won't work
Ex().check_object().has_equal_value()

# sct
Ex().check_object().has_equal_value(expr_code = 'x.n')

# submissions that will pass this sct
x = Number(1)
x = Number(2 - 1)
```

The basic SCT like in the previous example will notwork here. Notice how we used the

`expr_code` argument to _override_ which value *has_equal_value()* is checking. Instead of checking whether *x* corresponds between student and solution process, it's now executing the expression `x.n` and seeing if the result of running this expression in both student and solution process match.

is_instance(*state*, *inst*, *not_instance_msg=None*)

Check whether an object is an instance of a certain class.

`is_instance()` can currently only be used when chained from `check_object()`, the function that is used to 'zoom in' on the object of interest.

Parameters

- **inst** (*class*) – The class that the object should have.
- **not_instance_msg** (*str*) – When specified, this overrides the automatically generated message in case the object does not have the expected class.
- **state** (*State*) – The state that is passed in through the SCT chain (don't specify this).

Example Student code and solution code:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
```

SCT:

```
# Verify the class of arr
import numpy
Ex().check_object('arr').is_instance(numpy.ndarray)
```

check_df(*state*, *index*, *missing_msg=None*, *not_instance_msg=None*, *expand_msg=None*)

Check whether a DataFrame was defined and it is the right type

`check_df()` is a combo of `check_object()` and `is_instance()` that checks whether the specified object exists and whether the specified object is pandas DataFrame.

You can continue checking the data frame with `check_keys()` function to 'zoom in' on a particular column in the pandas DataFrame:

Parameters

- **index** (*str*) – Name of the data frame to zoom in on.
- **missing_msg** (*str*) – See `check_object()`.
- **not_instance_msg** (*str*) – See `is_instance()`.
- **expand_msg** (*str*) – If specified, this overrides any messages that are prepended by previous SCT chains.

Example Suppose you want the student to create a DataFrame `my_df` with two columns. The column `a` should contain the numbers 1 to 3, while the contents of column `b` can be anything:

```
import pandas as pd
my_df = pd.DataFrame({"a": [1, 2, 3], "b": ["a", "n", "y"]})
```

The following SCT would robustly check that:

```
Ex().check_df("my_df").multi(
    check_keys("a").has_equal_value(),
    check_keys("b")
)
```

- `check_df()` checks if `my_df` exists (`check_object()` behind the scenes) and is a DataFrame (`is_instance()`)
- `check_keys("a")` zooms in on the column `a` of the data frame, and `has_equal_value()` checks if the columns correspond between student and solution process.
- `check_keys("b")` zooms in on hte column `b` of the data frame, but there's no 'equality checking' happening

The following submissions would pass the SCT above:

```
my_df = pd.DataFrame({"a": [1, 1 + 1, 3], "b": ["a", "1", "1"]})
my_df = pd.DataFrame({"a": [1, 2, 3], "b": [4, 5, 6], "c": [7, 8, 9]})
```

check_keys(*state*, *key*, *missing_msg=None*, *expand_msg=None*)

Check whether an object (dict, DataFrame, etc) has a key.

`check_keys()` can currently only be used when chained from `check_object()`, the function that is used to 'zoom in' on the object of interest.

Parameters

- **key** (*str*) – Name of the key that the object should have.
- **missing_msg** (*str*) – When specified, this overrides the automatically generated message in case the key does not exist.
- **expand_msg** (*str*) – If specified, this overrides any messages that are prepended by previous SCT chains.
- **state** (*State*) – The state that is passed in through the SCT chain (don't specify this).

Example Student code and solution code:

```
x = {'a': 2}
```

SCT:

```
# Verify that x contains a key a
Ex().check_object('x').check_keys('a')

# Verify that x contains a key a and a is correct.
Ex().check_object('x').check_keys('a').has_equal_value()
```

2.2 Function calls

check_function(*state*, *name*, *index=0*, *missing_msg=None*, *params_not_matched_msg=None*, *expand_msg=None*, *signature=True*)

Check whether a particular function is called.

`check_function()` is typically followed by:

- `check_args()` to check whether the arguments were specified. In turn, `check_args()` can be followed by `has_equal_value()` or `has_equal_ast()` to assert that the arguments were correctly specified.
- `has_equal_value()` to check whether rerunning the function call coded by the student gives the same result as calling the function call as in the solution.

Checking function calls is a tricky topic. Please visit the dedicated article for more explanation, edge cases and best practices.

Parameters

- **name** (*str*) – the name of the function to be tested. When checking functions in packages, always use the 'full path' of the function.
- **index** (*int*) – index of the function call to be checked. Defaults to 0.
- **missing_msg** (*str*) – If specified, this overrides an automatically generated feedback message in case the student did not call the function correctly.
- **params_not_matched_msg** (*str*) – If specified, this overrides an automatically generated feedback message in case the function parameters were not successfully matched.
- **expand_msg** (*str*) – If specified, this overrides any messages that are prepended by previous SCT chains.
- **signature** (*Signature*) – Normally, check_function() can figure out what the function signature is, but it might be necessary to use `sig_from_params()` to manually build a signature and pass this along.
- **state** (*State*) – State object that is passed from the SCT Chain (don't specify this).

Examples Student code and solution code:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
np.mean(arr)
```

SCT:

```
# Verify whether arr was correctly set in np.mean
Ex().check_function('numpy.mean').check_args('a').has_equal_value()

# Verify whether np.mean(arr) produced the same result
Ex().check_function('numpy.mean').has_equal_value()
```

check_args(*state*, *name*, *missing_msg=None*)

Check whether a function argument is specified.

This function can follow `check_function()` in an SCT chain and verifies whether an argument is specified. If you want to go on and check whether the argument was correctly specified, you can can continue chaining with `has_equal_value()` (value-based check) or `has_equal_ast()` (AST-based check)

This function can also follow `check_function_def()` or `check_lambda_function()` to see if arguments have been specified.

Parameters

- **name** (*str*) – the name of the argument for which you want to check if it is specified. This can also be a number, in which case it refers to the positional arguments. Named arguments take precedence.
- **missing_msg** (*str*) – If specified, this overrides the automatically generated feedback message in case the student did specify the argument.
- **state** (*State*) – State object that is passed from the SCT Chain (don't specify this).

Examples Student and solution code:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
np.mean(arr)
```

SCT:

```
# Verify whether arr was correctly set in np.mean
# has_equal_value() checks the value of arr, used to set argument a
Ex().check_function('numpy.mean').check_args('a').has_equal_value()

# Verify whether arr was correctly set in np.mean
# has_equal_ast() checks the expression used to set argument a
Ex().check_function('numpy.mean').check_args('a').has_equal_ast()
```

Student and solution code:

```
def my_power(x):
    print("calculating sqrt...")
    return(x * x)
```

SCT:

```
Ex().check_function_def('my_power').multi(
    check_args('x') # will fail if student used y as arg
    check_args(0)   # will still pass if student used y as arg
)
```

2.3 Output

has_output(*state*, *text*, *pattern=True*, *no_output_msg=None*)

Search student output for a pattern.

Among the student and solution process, the student submission and solution code as a string, the `Ex()` state also contains the output that a student generated with his or her submission.

With `has_output()`, you can access this output and match it against a regular or fixed expression.

Parameters

- **text** (*str*) – the text that is searched for
- **pattern** (*bool*) – if True (default), the text is treated as a pattern. If False, it is treated as plain text.
- **no_output_msg** (*str*) – feedback message to be displayed if the output is not found.

Example As an example, suppose we want a student to print out a sentence:

```
# Print the "This is some ... stuff"
print("This is some weird stuff")
```

The following SCT tests whether the student prints out `This is some weird stuff`:

```
# Using exact string matching
Ex().has_output("This is some weird stuff", pattern = False)

# Using a regular expression (more robust)
```

(continues on next page)

(continued from previous page)

```
# pattern = True is the default
msg = "Print out ``This is some ... stuff`` to the output, " + \
      "fill in ``...`` with a word you like."
Ex().has_output(r"This is some \w* stuff", no_output_msg = msg)
```

has_printout(*state*, *index*, *not_printed_msg=None*, *pre_code=None*, *name=None*, *copy=False*)

Check if the right printouts happened.

`has_printout()` will look for the printout in the solution code that you specified with `index` (0 in this case), rerun the `print()` call in the solution process, capture its output, and verify whether the output is present in the output of the student.

This is more robust as `Ex().check_function('print')` initiated chains as students can use as many printouts as they want, as long as they do the correct one somewhere.

Parameters

- **index** (*int*) – index of the `print()` call in the solution whose output you want to search for in the student output.
- **not_printed_msg** (*str*) – if specified, this overrides the default message that is generated when the output is not found in the student output.
- **pre_code** (*str*) – Python code as a string that is executed before running the targeted student call. This is the ideal place to set a random seed, for example.
- **copy** (*bool*) – whether to try to deep copy objects in the environment, such as lists, that could accidentally be mutated. Disabled by default, which speeds up SCTs.
- **state** (*State*) – state as passed by the SCT chain. Don't specify this explicitly.

Example Suppose you want somebody to print out 4:

```
print(1, 2, 3, 4)
```

The following SCT would check that:

```
Ex().has_printout(0)
```

All of the following SCTs would pass:

```
print(1, 2, 3, 4)
print('1 2 3 4')
print(1, 2, '3 4')
print("random"); print(1, 2, 3, 4)
```

Example Watch out: `has_printout()` will effectively **rerun** the `print()` call in the solution process after the entire solution script was executed. If your solution script updates the value of *x* after executing it, `has_printout()` will not work.

Suppose you have the following solution:

```
x = 4
print(x)
x = 6
```

The following SCT will not work:

```
Ex().has_printout(0)
```

Why? When the `print(x)` call is executed, the value of `x` will be 6, and pythonwhat will look for the output *'6'* in the output the student generated. In cases like these, `has_printout()` cannot be used.

Example Inside a for loop `has_printout()`

Suppose you have the following solution:

```
for i in range(5):
    print(i)
```

The following SCT will not work:

```
Ex().check_for_loop().check_body().has_printout(0)
```

The reason is that `has_printout()` can only be called from the root state. `Ex()`. If you want to check printouts done in e.g. a for loop, you have to use a *check_function('print')* chain instead:

```
Ex().check_for_loop().check_body().\
    set_context(0).check_function('print').\
    check_args(0).has_equal_value()
```

has_no_error(*state*, *incorrect_msg='Have a look at the console: your code contains an error. Fix it and try again!'*)

Check whether the submission did not generate a runtime error.

If all SCTs for an exercise pass, before marking the submission as correct pythonwhat will automatically check whether the student submission generated an error. This means it is not needed to use `has_no_error()` explicitly.

However, in some cases, using `has_no_error()` explicitly somewhere throughout your SCT execution can be helpful:

- If you want to make sure people didn't write typos when writing a long function name.
- If you want to first verify whether a function actually runs, before checking whether the arguments were specified correctly.
- More generally, if, because of the content, it's instrumental that the script runs without errors before doing any other verifications.

Parameters **incorrect_msg** – if specified, this overrides the default message if the student code generated an error.

Example Suppose you're verifying an exercise about model training and validation:

```
# pre exercise code
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn import datasets
from sklearn import svm

iris = datasets.load_iris()
iris.data.shape, iris.target.shape

# solution
X_train, X_test, y_train, y_test = train_test_split(
    iris.data, iris.target, test_size=0.4, random_state=0)
```

If you want to make sure that `train_test_split()` ran without errors, which would check if the student typed the function without typos and used sensical arguments, you could use the following SCT:

```
Ex().has_no_error()
Ex().check_function('sklearn.model_selection.train_test_split').multi(
    check_args(['arrays', 0]).has_equal_value(),
    check_args(['arrays', 0]).has_equal_value(),
    check_args(['options', 'test_size']).has_equal_value(),
    check_args(['options', 'random_state']).has_equal_value()
)
```

If, on the other hand, you want to fall back onto pythonwhat's built in behavior, that checks for an error before marking the exercise as correct, you can simply leave of the `has_no_error()` step.

2.4 Code

has_code(*state*, *text*, *pattern=True*, *not_typed_msg=None*)

Test the student code.

Tests if the student typed a (pattern of) text. It is advised to use `has_equal_ast()` instead of `has_code()`, as it is more robust to small syntactical differences that don't change the code's behavior.

Parameters

- **text** (*str*) – the text that is searched for
- **pattern** (*bool*) – if True (the default), the text is treated as a pattern. If False, it is treated as plain text.
- **not_typed_msg** (*str*) – feedback message to be displayed if the student did not type the text.

Example Student code and solution code:

```
y = 1 + 2 + 3
```

SCT:

```
# Verify that student code contains pattern (not robust!!):
Ex().has_code(r"1\s*\+2\s*\+3")
```

has_import(*state*, *name*, *same_as=False*, *not_imported_msg='Did you import `{{pkg}}`?'*, *incorrect_as_msg='Did you import `{{pkg}}` as `{{alias}}`?'*)

Checks whether student imported a package or function correctly.

Python features many ways to import packages. All of these different methods revolve around the `import`, `from` and `as` keywords. `has_import()` provides a robust way to check whether a student correctly imported a certain package.

By default, `has_import()` allows for different ways of aliasing the imported package or function. If you want to make sure the correct alias was used to refer to the package or function that was imported, set `same_as=True`.

Parameters

- **name** (*str*) – the name of the package that has to be checked.

- **same_as** (*bool*) – if True, the alias of the package or function has to be the same. Defaults to False.
- **not_imported_msg** (*str*) – feedback message when the package is not imported.
- **incorrect_as_msg** (*str*) – feedback message if the alias is wrong.

Example Example 1, where aliases don't matter (defaut):

```
# solution
import matplotlib.pyplot as plt

# sct
Ex().has_import("matplotlib.pyplot")

# passing submissions
import matplotlib.pyplot as plt
from matplotlib import pyplot as plt
import matplotlib.pyplot as pltttt

# failing submissions
import matplotlib as mpl
```

Example 2, where the SCT is coded so aliases do matter:

```
# solution
import matplotlib.pyplot as plt

# sct
Ex().has_import("matplotlib.pyplot", same_as=True)

# passing submissions
import matplotlib.pyplot as plt
from matplotlib import pyplot as plt

# failing submissions
import matplotlib.pyplot as pltttt
```

2.5 has_equal_x

has_equal_value(*state*, *incorrect_msg=None*, *error_msg=None*, *undefined_msg=None*, *append=None*, *extra_env=None*, *context_vals=None*, *pre_code=None*, *expr_code=None*, *name=None*, *copy=True*, *func=None*, *override=None*, ***, *test='value'*)

Run targeted student and solution code, and compare returned value.

When called on an SCT chain, `has_equal_value()` will execute the student and solution code that is 'zoomed in on' and compare the returned values.

Parameters

- **incorrect_msg** (*str*) – feedback message if the returned value of the expression in the solution doesn't match the one of the student. This feedback message will be expanded if it is used in the context of another check function, like `check_if_else`.
- **error_msg** (*str*) – feedback message if there was an error when running the targeted student code. Note that when testing for an error, this message is displayed when none is raised.

- **undefined_msg** (*str*) – feedback message if the `name` argument is defined, but a variable with that name doesn't exist after running the targeted student code.
- **extra_env** (*dict*) – set variables to the extra environment. They will update the student and solution environment in the active state before the student/solution code in the active state is ran. This argument should contain a dictionary with the keys the names of the variables you want to set, and the values are the values of these variables. You can also use `set_env()` for this.
- **context_vals** (*list*) – set variables which are bound in a `for` loop to certain values. This argument is only useful when checking a for loop (or list comprehensions). It contains a list with the values of the bound variables. You can also use `set_context()` for this.
- **pre_code** (*str*) – the code in string form that should be executed before the expression is executed. This is the ideal place to set a random seed, for example.
- **expr_code** (*str*) – If this argument is set, the expression in the student/solution code will not be ran. Instead, the given piece of code will be ran in the student as well as the solution environment and the result will be compared. However if the string contains one or more placeholders `__focus__`, they will be substituted by the currently focused code.
- **name** (*str*) – If this is specified, the returned value of running this expression after running the focused expression is returned, instead of the returned value of the focused expression in itself. This is typically used to inspect the returned value of an object after executing the body of e.g. a `for` loop.
- **copy** (*bool*) – whether to try to deep copy objects in the environment, such as lists, that could accidentally be mutated. Disable to speed up SCTs. Disabling may lead to cryptic mutation issues.
- **func** (*function*) – custom binary function of form f(stu_result, sol_result), for equality testing.
- **override** – If specified, this avoids the execution of the targeted code in the solution process. Instead, it will compare the returned value of the expression in the student process with the value specified in `override`. Typically used in a `SingleProcessExercise` or if you want to allow for different solutions other than the one coded up in the solution.

Example Student code and solution code:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
np.mean(arr)
```

SCT:

```
# Verify equality of arr:
Ex().check_object('arr').has_equal_value()

# Verify whether arr was correctly set in np.mean
Ex().check_function('numpy.mean').check_args('a').has_equal_value()

# Verify whether np.mean(arr) produced the same result
Ex().check_function('numpy.mean').has_equal_value()
```

has_equal_output(*state*, *incorrect_msg=None*, *error_msg=None*, *undefined_msg=None*, *append=None*, *extra_env=None*, *context_vals=None*, *pre_code=None*, *expr_code=None*, *name=None*, *copy=True*, *func=None*, *override=None*, ***, *test='output'*)

Run targeted student and solution code, and compare output.

When called on an SCT chain, `has_equal_output()` will execute the student and solution code that is 'zoomed in on' and compare the output.

Parameters

- **incorrect_msg** (*str*) – feedback message if the output of the expression in the solution doesn't match the one of the student. This feedback message will be expanded if it is used in the context of another check function, like `check_if_else`.
- **error_msg** (*str*) – feedback message if there was an error when running the targeted student code. Note that when testing for an error, this message is displayed when none is raised.
- **undefined_msg** (*str*) – feedback message if the `name` argument is defined, but a variable with that name doesn't exist after running the targeted student code.
- **extra_env** (*dict*) – set variables to the extra environment. They will update the student and solution environment in the active state before the student/solution code in the active state is ran. This argument should contain a dictionary with the keys the names of the variables you want to set, and the values are the values of these variables. You can also use `set_env()` for this.
- **context_vals** (*list*) – set variables which are bound in a `for` loop to certain values. This argument is only useful when checking a for loop (or list comprehensions). It contains a list with the values of the bound variables. You can also use `set_context()` for this.
- **pre_code** (*str*) – the code in string form that should be executed before the expression is executed. This is the ideal place to set a random seed, for example.
- **expr_code** (*str*) – If this argument is set, the expression in the student/solution code will not be ran. Instead, the given piece of code will be ran in the student as well as the solution environment and the result will be compared. However if the string contains one or more placeholders `__focus__`, they will be substituted by the currently focused code.
- **name** (*str*) – If this is specified, the output of running this expression after running the focused expression is returned, instead of the output of the focused expression in itself. This is typically used to inspect the output of an object after executing the body of e.g. a `for` loop.
- **copy** (*bool*) – whether to try to deep copy objects in the environment, such as lists, that could accidentally be mutated. Disable to speed up SCTs. Disabling may lead to cryptic mutation issues.
- **func** (*function*) – custom binary function of form f(stu_result, sol_result), for equality testing.
- **override** – If specified, this avoids the execution of the targeted code in the solution process. Instead, it will compare the output of the expression in the student process with the value specified in `override`. Typically used in a `SingleProcessExercise` or if you want to allow for different solutions other than the one coded up in the solution.

has_equal_error(*state*, *incorrect_msg=None*, *error_msg=None*, *undefined_msg=None*, *append=None*, *extra_env=None*, *context_vals=None*, *pre_code=None*, *expr_code=None*, *name=None*, *copy=True*, *func=None*, *override=None*, ***, *test='error'*)

Run targeted student and solution code, and compare generated errors.

When called on an SCT chain, `has_equal_error()` will execute the student and solution code that is 'zoomed in on' and compare the errors that they generate.

Parameters

- **incorrect_msg** (*str*) – feedback message if the error of the expression in the solution doesn't match the one of the student. This feedback message will be expanded if it is used in the context of another check function, like `check_if_else`.
- **error_msg** (*str*) – feedback message if there was an error when running the targeted student code. Note that when testing for an error, this message is displayed when none is raised.
- **undefined_msg** (*str*) – feedback message if the `name` argument is defined, but a variable with that name doesn't exist after running the targeted student code.
- **extra_env** (*dict*) – set variables to the extra environment. They will update the student and solution environment in the active state before the student/solution code in the active state is ran. This argument should contain a dictionary with the keys the names of the variables you want to set, and the values are the values of these variables. You can also use `set_env()` for this.
- **context_vals** (*list*) – set variables which are bound in a `for` loop to certain values. This argument is only useful when checking a for loop (or list comprehensions). It contains a list with the values of the bound variables. You can also use `set_context()` for this.
- **pre_code** (*str*) – the code in string form that should be executed before the expression is executed. This is the ideal place to set a random seed, for example.
- **expr_code** (*str*) – If this argument is set, the expression in the student/solution code will not be ran. Instead, the given piece of code will be ran in the student as well as the solution environment and the result will be compared. However if the string contains one or more placeholders `__focus__`, they will be substituted by the currently focused code.
- **name** (*str*) – If this is specified, the error of running this expression after running the focused expression is returned, instead of the error of the focused expression in itself. This is typically used to inspect the error of an object after executing the body of e.g. a `for` loop.
- **copy** (*bool*) – whether to try to deep copy objects in the environment, such as lists, that could accidentally be mutated. Disable to speed up SCTs. Disabling may lead to cryptic mutation issues.
- **func** (*function*) – custom binary function of form f(stu_result, sol_result), for equality testing.
- **override** – If specified, this avoids the execution of the targeted code in the solution process. Instead, it will compare the error of the expression in the student process with the value specified in `override`. Typically used in a `SingleProcessExercise` or if you want to allow for different solutions other than the one coded up in the solution.

has_equal_ast(*state*, *incorrect_msg=None*, *code=None*, *exact=True*, *append=None*)

Test whether abstract syntax trees match between the student and solution code.

`has_equal_ast()` can be used in two ways:

- As a robust version of `has_code()`. By setting `code`, you can look for the AST representation of `code` in the student's submission. But be aware that `a` and `a = 1` won't match, as reading and assigning are not the same in an AST. Use `ast.dump(ast.parse(code))` to see an AST representation of `code`.
- As an expression-based check when using more advanced SCT chain, e.g. to compare the equality of expressions to set function arguments.

Parameters

- **incorrect_msg** – message displayed when ASTs mismatch. When you specify `code` yourself, you have to specify this.

- **code** – optional code to use instead of the solution AST.
- **exact** – whether the representations must match exactly. If false, the solution AST only needs to be contained within the student AST (similar to using test student typed). Defaults to `True`, unless the `code` argument has been specified.

Example Student and Solution Code:

```
dict(a = 'value').keys()
```

SCT:

```
# all pass
Ex().has_equal_ast()
Ex().has_equal_ast(code = "dict(a = 'value').keys()")
Ex().has_equal_ast(code = "dict(a = 'value')", exact = False)
```

Student and Solution Code:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
np.mean(arr)
```

SCT:

```
# Check underlying value of arugment a of np.mean:
Ex().check_function('numpy.mean').check_args('a').has_equal_ast()

# Only check AST equality of expression used to specify argument a:
Ex().check_function('numpy.mean').check_args('a').has_equal_ast()
```

2.6 Combining SCTs

multi (*state*, **tests*)

Run multiple subtests. Return original state (for chaining).

This function could be thought as an AND statement, since all tests it runs must pass

Parameters

- **state** – State instance describing student and solution code, can be omitted if used with Ex()
- **tests** – one or more sub-SCTs to run.

Example The SCT below checks two has_code cases..

```
Ex().multi(has_code('SELECT'), has_code('WHERE'))
```

The SCT below uses `multi` to 'branch out' to check that the SELECT statement has both a WHERE and LIMIT clause..

```
Ex().check_node('SelectStmt', 0).multi(
    check_edge('where_clause'),
    check_edge('limit_clause')
)
```

Example Suppose we want to verify the following function call:

```
round(1.2345, ndigits=2)
```

The following SCT would verify this, using `multi` to 'branch out' the state to two sub-SCTs:

```
Ex().check_function('round').multi(
    check_args(0).has_equal_value(),
    check_args('ndigits').has_equal_value()
)
```

check_correct (*state*, *check*, *diagnose*)
Allows feedback from a diagnostic SCT, only if a check SCT fails.

Parameters

- **state** – State instance describing student and solution code. Can be omitted if used with Ex().
- **check** – An sct chain that must succeed.
- **diagnose** – An sct chain to run if the check fails.

Example The SCT below tests whether students query result is correct, before running diagnostic SCTs..

```
Ex().check_correct(
    check_result(),
    check_node('SelectStmt')
)
```

Example The SCT below tests whether an object is correct. Only if the object is not correct, will the function calling checks be executed

```
Ex().check_correct(
    check_object('x').has_equal_value(),
    check_function('round').check_args(0).has_equal_value()
)
```

check_or (*state*, **tests*)
Test whether at least one SCT passes.

Parameters

- **state** – State instance describing student and solution code, can be omitted if used with Ex()
- **tests** – one or more sub-SCTs to run

Example The SCT below tests that the student typed either 'SELECT' or 'WHERE' (or both)..

```
Ex().check_or(
    has_code('SELECT'),
    has_code('WHERE')
)
```

The SCT below checks that a SELECT statement has at least a WHERE c or LIMIT clause..

```
Ex().check_node('SelectStmt', 0).check_or(
    check_edge('where_clause'),
    check_edge('limit_clause')
)
```

Example The SCT below tests that the student typed either 'mean' or 'median':

```
Ex().check_or(
    has_code('mean'),
    has_code('median')
)
```

If the student didn't type either, the feedback message generated by `has_code(mean)`, the first SCT, will be presented to the student.

check_not(*state*, **tests*, *msg*)

Run multiple subtests that should fail. If all subtests fail, returns original state (for chaining)

- This function is currently only tested in working with `has_code()` in the subtests.
- This function can be thought as a `NOT(x OR y OR ...)` statement, since all tests it runs must fail
- This function can be considered a direct counterpart of multi.

Parameters

- **state** – State instance describing student and solution code, can be omitted if used with Ex()
- ***tests** – one or more sub-SCTs to run
- **msg** – feedback message that is shown in case not all tests specified in `*tests` fail.

Example Thh SCT below runs two has_code cases..

```
Ex().check_not(
    has_code('INNER'),
    has_code('OUTER'),
    incorrect_msg="Don't use `INNER` or `OUTER`!"
)
```

If students use `INNER (JOIN)` or `OUTER (JOIN)` in their code, this test will fail.

Example The SCT fails with feedback for a specific incorrect value, defined using an override:

```
Ex().check_object('result').multi(
    check_not(
        has_equal_value(override=100),
        msg='100 is incorrect for reason xyz.'
    ),
    has_equal_value()
)
```

Notice that `check_not` comes before the `has_equal_value` test that checks if the student value is equal to the solution value.

Example The SCT below runs two `has_code` cases:

```
Ex().check_not(
    has_code('mean'),
    has_code('median'),
    msg='Check your code'
)
```

If students use `mean` or `median` anywhere in their code, this SCT will fail.

Note:

- This function is not yet tested with all checks, please report unexpected behaviour.
- This function can be thought as a NOT(x OR y OR . . .) statement, since all tests it runs must fail
- This function can be considered a direct counterpart of multi.

2.7 Function/Class/Lambda definitions

check_function_def(*state*, *index=0*, *typestr='{{ordinal}} node'*, *missing_msg=None*, *expand_msg=None*)

Check whether a function was defined and zoom in on it.

Can be chained with `check_call()`, `check_args()` and `check_body()`.

Parameters

- **index** – the name of the function definition.
- **typestr** – If specified, this overrides the standard way of referring to the construct you're zooming in on.
- **missing_msg** – If specified, this overrides the automatically generated feedback message in case the construct could not be found.
- **expand_msg** – If specified, this overrides the automatically generated feedback message that is prepended to feedback messages that are thrown further in the SCT chain.

Example Suppose you want a student to create a function `shout_echo()`:

```
def shout_echo(word1, echo=1):
    echo_word = word1 * echo
    shout_words = echo_word + '!!!'
    return shout_words
```

The following SCT robustly checks this:

```
Ex().check_function_def('shout_echo').check_correct(
    multi(
        check_call("f('hey', 3)").has_equal_value(),
        check_call("f('hi', 2)").has_equal_value(),
        check_call("f('hi')").has_equal_value()
    ),
    check_body().set_context('test', 1).multi(
        has_equal_value(name = 'echo_word'),
        has_equal_value(name = 'shout_words')
    )
)
```

Here:

- `check_function_def()` zooms in on the function definition of `shout_echo` in both student and solution code (and process).
- `check_correct()` is used to
  - First check whether the function gives the correct result when called in different ways (through `check_call()`).

- Only if these 'function unit tests' don't pass, `check_correct()` will run the *check_body()* chain that dives deeper into the function definition body. This chain sets the context variables - `word1` and `echo`, the arguments of the function - to the values `'test'` and `1` respectively, again while being agnostic to the actual name of these context variables.

Notice how `check_correct()` is used to great effect here: why check the function definition internals if the I/O of the function works fine? Because of this construct, all the following submissions will pass the SCT:

```
# passing submission 1
def shout_echo(w, e=1):
    ew = w * e
    return ew + '!!!'

# passing submission 2
def shout_echo(a, b=1):
    return a * b + '!!!'
```

Example `check_args()` is most commonly used in combination with `check_function()` to verify the arguments of function **calls**, but it can also be used to verify the arguments specified in the signature of a function definition.

We can extend the SCT for the previous example to explicitly verify the signature:

```
msg1 = "Make sure to specify 2 arguments!"
msg2 = "don't specify default arg!"
msg3 = "specify a default arg!"
Ex().check_function_def('shout_echo').check_correct(
    multi(
        check_call("f('hey', 3)").has_equal_value(),
        check_call("f('hi', 2)").has_equal_value(),
        check_call("f('hi')").has_equal_value()
    ),
    multi(
        has_equal_part_len("args", unequal_msg=1),
        check_args(0).has_equal_part('is_default', msg=msg2),
        check_args('word1').has_equal_part('is_default', msg=msg2),
        check_args(1).\
            has_equal_part('is_default', msg=msg3).has_equal_value(),
        check_args('echo').\
            has_equal_part('is_default', msg=msg3).has_equal_value(),
        check_body().set_context('test', 1).multi(
            has_equal_value(name = 'echo_word'),
            has_equal_value(name = 'shout_words')
        )
    )
)
```

- `has_equal_part_len("args")` verifies whether student and solution function definition have the same number of arguments.
- `check_args(0)` refers to the first argument in the signature by position, and the chain checks whether the student did not specify a default as in the solution.
- An alternative for the `check_args(0)` chain is to use `check_args('word1')` to refer to the first argument. This is more restrictive, as the requires the student to use the exact same name.

- `check_args(1)` refers to the second argument in the signature by position, and the chain checks whether the student specified a default, as in the solution, and whether the value of this default corresponds to the one in the solution.
- The `check_args('echo')` chain is a more restrictive alternative for the `check_args(1)` chain.

Notice that support for verifying arguments is not great yet:

- A lot of work is needed to verify the number of arguments and whether or not defaults are set.
- You have to specify custom messages because pythonwhat doesn't automatically generate messages.

We are working on it!

has_equal_part_len(*state*, *name*, *unequal_msg*)

Verify that a part that is zoomed in on has equal length.

Typically used in the context of `check_function_def()`

Parameters

- **name** (*str*) – name of the part for which to check the length to the corresponding part in the solution.
- **unequal_msg** (*str*) – Message in case the lengths do not match.
- **state** (*State*) – state as passed by the SCT chain. Don't specify this explicitly.

Examples Student and solution code:

```
def shout(word):
    return word + '!!!'
```

SCT that checks number of arguments:

```
Ex().check_function_def('shout').has_equal_part_len('args', 'not␣
↪enough args!')
```

check_call(*state*, *callstr*, *argstr=None*, *expand_msg=None*)

When checking a function definition of lambda function, prepare has_equal_x for checking the call of a user-defined function.

Parameters

- **callstr** (*str*) – call string that specifies how the function should be called, e.g. *f(1, a = 2)*. `check_call()` will replace `f` with the function/lambda you're targeting.
- **argstr** (*str*) – If specified, this overrides the way the function call is refered to in the expand message.
- **expand_msg** (*str*) – If specified, this overrides any messages that are prepended by previous SCT chains.
- **state** (*State*) – state object that is chained from.

Example Student and solution code:

```
def my_power(x):
    print("calculating sqrt...")
    return(x * x)
```

SCT:

```
Ex().check_function_def('my_power').multi(
    check_call("f(3)").has_equal_value()
    check_call("f(3)").has_equal_output()
)
```

check_class_def(*state*, *index=0*, *typestr='{{ordinal}} node'*, *missing_msg=None*, *expand_msg=None*)

Check whether a class was defined and zoom in on its definition

Can be chained with `check_bases()` and `check_body()`.

Parameters

- **index** – the name of the function definition.
- **typestr** – If specified, this overrides the standard way of referring to the construct you're zooming in on.
- **missing_msg** – If specified, this overrides the automatically generated feedback message in case the construct could not be found.
- **expand_msg** – If specified, this overrides the automatically generated feedback message that is prepended to feedback messages that are thrown further in the SCT chain.

Example Suppose you want to check whether a class was defined correctly:

```
class MyInt(int):
    def __init__(self, i):
        super().__init__(i + 1)
```

The following SCT would verify this:

```
Ex().check_class_def('MyInt').multi(
    check_bases(0).has_equal_ast(),
    check_body().check_function_def('__init__').multi(
        check_args('self'),
        check_args('i'),
        check_body().set_context(i = 2).multi(
            check_function('super', signature=False),
            check_function('super.__init__').check_args(0).has_equal_
↪value()
        )
    )
)
```

- `check_class_def()` looks for the class definition itself.
- With `check_bases()`, you can zoom in on the different basse classes that the class definition inherits from.
- With `check_body()`, you zoom in on the class body, after which you can use other functions such as `check_function_def()` to look for class methods.
- Of course, just like for other examples, you can use `check_correct()` where necessary, e.g. to verify whether class methods give the right behavior with `check_call()` before diving into the body of the method itself.

check_lambda_function(*state*, *index=0*, *typestr='{{ordinal}} node'*, *missing_msg=None*, *expand_msg=None*)

Check whether a lambda function was coded zoom in on it.

Can be chained with `check_call()`, `check_args()` and `check_body()`.

Parameters

- **index** – the index of the lambda function (0-based).
- **typestr** – If specified, this overrides the standard way of referring to the construct you're zooming in on.
- **missing_msg** – If specified, this overrides the automatically generated feedback message in case the construct could not be found.
- **expand_msg** – If specified, this overrides the automatically generated feedback message that is prepended to feedback messages that are thrown further in the SCT chain.

Example Suppose you want a student to create a lambda function that returns the length of an array times two:

```
lambda x: len(x)*2
```

The following SCT robustly checks this:

```
Ex().check_lambda_function().check_correct(
    multi(
        check_call("f([1])").has_equal_value(),
        check_call("f([1, 2])").has_equal_value()
    ),
    check_body().set_context([1, 2, 3]).has_equal_value()
)
```

Here:

- `check_lambda_function()` zooms in on the first lambda function in both student and solution code.
- `check_correct()` is used to
  - First check whether the lambda function gives the correct result when called in different ways (through `check_call()`).
  - Only if these 'function unit tests' don't pass, `check_correct()` will run the *check_body()* chain that dives deeper into the lambda function's body. This chain sets the context variable *x*, the argument of the function, to the values `[1, 2, 3]`, while being agnostic to the actual name the student used for this context variable.

Notice how `check_correct()` is used to great effect here: why check the function definition internals if the I/O of the function works fine? Because of this construct, all the following submissions will pass the SCT:

```
# passing submission 1
lambda x: len(x) + len(x)

# passing submission 2
lambda y, times=2: len(y) * times
```

2.8 Control flow

check_if_else(*state*, *index=0*, *typestr='{{ordinal}} node'*, *missing_msg=None*, *expand_msg=None*)

Check whether an if statement was coded zoom in on it.

Parameters

- **index** – the index of the if statement to look for (0 based)
- **typestr** – If specified, this overrides the standard way of referring to the construct you're zooming in on.
- **missing_msg** – If specified, this overrides the automatically generated feedback message in case the construct could not be found.
- **expand_msg** – If specified, this overrides the automatically generated feedback message that is prepended to feedback messages that are thrown further in the SCT chain.

Example Suppose you want students to print out a message if `x` is larger than 0:

```
x = 4
if x > 0:
    print("x is strictly positive")
```

The following SCT would verify that:

```
Ex().check_if_else().multi(
    check_test().multi(
        set_env(x = -1).has_equal_value(),
        set_env(x =  1).has_equal_value(),
        set_env(x =  0).has_equal_value()
    ),
    check_body().check_function('print', 0).\
        check_args('value').has_equal_value()
)
```

- `check_if_else()` zooms in on the first if statement in the student and solution submission.
- `check_test()` zooms in on the 'test' portion of the if statement, `x > 0` in case of the solution. `has_equal_value()` reruns this expression and the corresponding expression in the student code for different values of `x` (set with `set_env()`) and compare there results. This way, you can robustly verify whether the if test was coded up correctly. If the student codes up the condition as `0 < x`, this would also be accepted.
- `check_body()` zooms in on the 'body' portion of the if statement, `print("...")` in case of the solution. With a classical `check_function()` chain, it is verified whether the if statement contains a function `print()` and whether its argument is set correctly.

Example In Python, when an if-else statement has an `elif` clause, it is held in the *orelse* part. In this sense, an if-elif-else statement is represented by python as nested if-elses. More specifically, this if-else statement:

```
if x > 0:
    print(x)
elif y > 0:
    print(y)
else:
    print('none')
```

Is syntactically equivalent to:

```
if x > 0:
    print(x)
else:
    if y > 0:
        print(y)
    else:
        print('none')
```

The second representation has to be followed when writing the corresponding SCT:

```
Ex().check_if_else().multi(
    check_test(),           # zoom in on x > 0
    check_body(),           # zoom in on print(x)
    check_orelse().check_if_else().multi(
        check_test(),       # zoom in on y > 0
        check_body(),       # zoom in on print(y)
        check_orelse()      # zoom in on print('none')
    )
)
```

check_try_except(*state*, *index=0*, *typestr='{{ordinal}} node'*, *missing_msg=None*, *expand_msg=None*)

Check whether a try except statement was coded zoom in on it.

Can be chained with `check_body()`, `check_handlers()`, `check_orelse()` and `check_finalbody()`.

Parameters

- **index** – the index of the try except statement (0-based).
- **typestr** – If specified, this overrides the standard way of referring to the construct you're zooming in on.
- **missing_msg** – If specified, this overrides the automatically generated feedback message in case the construct could not be found.
- **expand_msg** – If specified, this overrides the automatically generated feedback message that is prepended to feedback messages that are thrown further in the SCT chain.

Example Suppose you want to verify whether the student did a *try-except* statement properly:

```
do_dangerous_thing = lambda n: n

try:
    x = do_dangerous_thing(n = 4)
except ValueError as e:
    x = 'something wrong with inputs'
except:
    x = 'something went wrong'
finally:
    print('ciao!')
```

The following SCT can be used to verify this:

```
Ex().check_try_except().multi(
    check_body().\
        check_function('do_dangerous_thing').\
        check_args('n').has_equal_value(),
    check_handlers('ValueError').\
```

(continues on next page)

(continued from previous page)

```
        has_equal_value(name = 'x'),
    check_handlers('all').\
        has_equal_value(name = 'x'),
    check_finalbody().\
        check_function('print').check_args(0).has_equal_value()
)
```

check_if_exp(*state*, *index=0*, *typestr='{{ordinal}} node'*, *missing_msg=None*, *expand_msg=None*)
Check whether an if expression was coded zoom in on it.

This function works the exact same way as `check_if_else()`.

check_with(*state*, *index=0*, *typestr='{{ordinal}} node'*, *missing_msg=None*, *expand_msg=None*)
Check whether a with statement was coded zoom in on it.

Parameters

- **index** – the index of the‘‘with‘‘ statement to verify (0-based)
- **typestr** – If specified, this overrides the standard way of referring to the construct you're zooming in on.
- **missing_msg** – If specified, this overrides the automatically generated feedback message in case the construct could not be found.
- **expand_msg** – If specified, this overrides the automatically generated feedback message that is prepended to feedback messages that are thrown further in the SCT chain.

2.9 Loops

check_for_loop(*state*, *index=0*, *typestr='{{ordinal}} node'*, *missing_msg=None*, *expand_msg=None*)
Check whether a for loop was coded and zoom in on it.

Can be chained with `check_iter()` and `check_body()`.

Parameters

- **index** – Index of the for loop (0-based).
- **typestr** – If specified, this overrides the standard way of referring to the construct you're zooming in on.
- **missing_msg** – If specified, this overrides the automatically generated feedback message in case the construct could not be found.
- **expand_msg** – If specified, this overrides the automatically generated feedback message that is prepended to feedback messages that are thrown further in the SCT chain.

Example Suppose you want a student to iterate over a predefined dictionary `my_dict` and do the appropriate printouts:

```
for key, value in my_dict.items():
    print(key + " - " + str(value))
```

The following SCT would verify this:

```
Ex().check_for_loop().multi(
    check_iter().has_equal_value(),
    check_body().multi(
        set_context('a', 1).has_equal_output(),
        set_context('b', 2).has_equal_output()
    )
)
```

- `check_for_loop()` zooms in on the `for` loop, and makes its parts available for further checking.
- `check_iter()` zooms in on the iterator part of the for loop, `my_dict.items()` in the solution. `has_equal_value()` re-executes the expressions specified by student and solution and compares their results.
- `check_body()` zooms in on the body part of the for loop, `print(key + " - " + str(value))`. For different values of `key` and `value`, the student's body and solution's body are executed again and the printouts are captured and compared to see if they are equal.

Notice how you do not need to specify the variables by name in `set_context()`. pythonwhat can figure out the variable names used in both student and solution code, and can do the verification independent of that. That way, we can make the SCT robust against submissions that code the correct logic, but use different names for the context values. In other words, the following student submissions that would also pass the SCT:

```
# passing submission 1
my_dict = {'a': 1, 'b': 2}
for k, v in my_dict.items():
    print(k + " - " + str(v))

# passing submission 2
my_dict = {'a': 1, 'b': 2}
for first, second in my_dict.items():
    mess = first + " - " + str(second)
    print(mess)
```

Example As another example, suppose you want the student to build a list of doubles as follows:

```
even = []
for i in range(10):
    even.append(2*i)
```

The following SCT would robustly verify this:

```
Ex().check_correct(
    check_object('even').has_equal_value(),
    check_for_loop().multi(
        check_iter().has_equal_value(),
        check_body().set_context(2).set_env(even = []).\
            has_equal_value(name = 'even')
    )
)
```

- `check_correct()` makes sure that we do not dive into the `for` loop if the array `even` is correctly populated in the end.
- If `even` was not correctly populated, `check_for_loop()` will zoom in on the for loop.

- The `check_iter()` chain verifies whether *range(10)* (or something equivalent) was used to iterate over.
- `check_body()` zooms in on the body, and reruns the body (`even.append(2*i)` in the solution) for `i` equal to 2, and even temporarily set to an empty array. Notice how we use `set_context()` to robustly set the context value (the student can use a different variable name), while we have to explicitly set `even` with `set_env()`. Also notice how we use `has_equal_value(name = 'even')` instead of the usual `check_object()`; `check_object()` can only be called from the root state `Ex()`.

Example As a follow-up example, suppose you want the student to build a list of doubles of the even numbers only:

```
even = []
for i in range(10):
    if i % 2 == 0:
        even.append(2*i)
```

The following SCT would robustly verify this:

```
Ex().check_correct(
    check_object('even').has_equal_value(),
    check_for_loop().multi(
        check_iter().has_equal_value(),
        check_body().check_if_else().multi(
            check_test().multi(
                set_context(1).has_equal_value(),
                set_context(2).has_equal_value()
            ),
            check_body().set_context(2).\
                set_env(even = []).has_equal_value(name = 'even')
        )
    )
)
```

check_while(*state*, *index=0*, *typestr='{{ordinal}} node'*, *missing_msg=None*, *expand_msg=None*)

Check whether a while loop was coded and zoom in on it.

Can be chained with `check_test()`, `check_body()` and `check_orelse()`.

Parameters

- **index** – the index of the while loop to verify (0-based).
- **typestr** – If specified, this overrides the standard way of referring to the construct you're zooming in on.
- **missing_msg** – If specified, this overrides the automatically generated feedback message in case the construct could not be found.
- **expand_msg** – If specified, this overrides the automatically generated feedback message that is prepended to feedback messages that are thrown further in the SCT chain.

Example Suppose you want a student to code a while loop that counts down a counter from 50 until a multilpe of 11 is found. If it is found, the value should be printed out.

```
i = 50
while i % 11 != 0:
    i -= 1
```

The following SCT robustly verifies this:

```
Ex().check_correct(
    check_object('i').has_equal_value(),
    check_while().multi(
            check_test().multi(
                set_env(i = 45).has_equal_value(),
                set_env(i = 44).has_equal_value()
            ),
            check_body().set_env(i = 3).has_equal_value(name = 'i')
    )
)
```

- `check_correct()` first checks whether the end result of `i` is correct. If it is, the entire chain that checks the `while` loop is skipped.
- If `i` is not correctly calculated, `check_while_loop()` zooms in on the while loop.
- `check_test()` zooms in on the condition of the `while` loop, `i % 11 != 0` in the solution, and verifies whether the expression gives the same results for different values of `i`, set through `set_env()`, when comparing student and solution.
- `check_body()` zooms in on the body of the `while` loop, and `has_equal_value()` checks whether rerunning this body updates `i` as expected when `i` is temporarily set to 3 with `set_env()`.

check_list_comp(*state*, *index=0*, *typestr='{{ordinal}} node'*, *missing_msg=None*, *expand_msg=None*)

Check whether a list comprehension was coded and zoom in on it.

Can be chained with `check_iter()`, `check_body()`, and `check_ifs()`.

Parameters

- **index** – Index of the list comprehension (0-based)
- **typestr** – If specified, this overrides the standard way of referring to the construct you're zooming in on.
- **missing_msg** – If specified, this overrides the automatically generated feedback message in case the construct could not be found.
- **expand_msg** – If specified, this overrides the automatically generated feedback message that is prepended to feedback messages that are thrown further in the SCT chain.

Example Suppose you expect students to create a list `my_list` as follows:

```
my_list = [ i*2 for i in range(0,10) if i>2 ]
```

The following SCT would robustly verify this:

```
Ex().check_correct(
    check_object('my_list').has_equal_value(),
    check_list_comp().multi(
        check_iter().has_equal_value(),
        check_body().set_context(4).has_equal_value(),
        check_ifs(0).multi(
            set_context(0).has_equal_value(),
            set_context(3).has_equal_value(),
            set_context(5).has_equal_value()
        )
```

(continues on next page)

(continued from previous page)

```
    )
)
```

- With `check_correct()`, we're making sure that the list comprehension checking is not executed if `my_list` was calculated properly.
- If `my_list` is not correct, the 'diagnose' chain will run: `check_list_comp()` looks for the first list comprehension in the student's submission.
- Next, `check_iter()` zooms in on the iterator, `range(0, 10)` in the case of the solution. `has_equal_value()` verifies whether the expression that the student used evaluates to the same value as the expression that the solution used.
- `check_body()` zooms in on the body, `i*2` in the case of the solution. `set_context()` sets the iterator to 4, allowing for the fact that the student used another name instead of `i` for this iterator. `has_equal_value()` reruns the body in the student and solution code with the iterator set to 4, and checks if the results are the same.
- `check_ifs(0)` zooms in on the first `if` of the list comprehension, `i>2` in case of the solution. With a series of `set_context()` and `has_equal_value()`, it is verifies whether this condition evaluates to the same value in student and solution code for different values of the iterator (*i* in the case of the solution, whatever in the case of the student).

check_dict_comp(*state*, *index=0*, *typestr='{{ordinal}} node'*, *missing_msg=None*, *expand_msg=None*)

Check whether a dictionary comprehension was coded and zoom in on it.

Can be chained with `check_key()`, `check_value()`, and `check_ifs()`.

Parameters

- **index** – Index of the dictionary comprehension (0-based)
- **typestr** – If specified, this overrides the standard way of referring to the construct you're zooming in on.
- **missing_msg** – If specified, this overrides the automatically generated feedback message in case the construct could not be found.
- **expand_msg** – If specified, this overrides the automatically generated feedback message that is prepended to feedback messages that are thrown further in the SCT chain.

Example Suppose you expect students to create a dictionary `my_dict` as follows:

```
my_dict = { m:len(m) for m in ['a', 'ab', 'abc'] }
```

The following SCT would robustly verify this:

```
Ex().check_correct(
    check_object('my_dict').has_equal_value(),
    check_dict_comp().multi(
        check_iter().has_equal_value(),
        check_key().set_context('ab').has_equal_value(),
        check_value().set_context('ab').has_equal_value()
    )
)
```

- With `check_correct()`, we're making sure that the dictionary comprehension checking is not executed if `my_dict` was created properly.

- If `my_dict` is not correct, the 'diagnose' chain will run: `check_dict_comp()` looks for the first dictionary comprehension in the student's submission.
- Next, `check_iter()` zooms in on the iterator, `['a', 'ab', 'abc']` in the case of the solution. `has_equal_value()` verifies whether the expression that the student used evaluates to the same value as the expression that the solution used.
- `check_key()` zooms in on the key of the comprehension, `m` in the case of the solution. `set_context()` temporaritly sets the iterator to `'ab'`, allowing for the fact that the student used another name instead of `m` for this iterator. `has_equal_value()` reruns the key expression in the student and solution code with the iterator set to `'ab'`, and checks if the results are the same.
- `check_value()` zooms in on the value of the comprehension, `len(m)` in the case of the solution. `has_equal_value()` reruns the value expression in the student and solution code with the iterator set to `'ab'`, and checks if the results are the same.

check_generator_exp(*state*, *index=0*, *typestr='{{ordinal}} node'*, *missing_msg=None*, *expand_msg=None*)

Check whether a generator expression was coded and zoom in on it.

Can be chained with `check_iter()`, `check_body()`, and `check_ifs()`.

Parameters

- **index** – Index of the generator expression (0-based)
- **typestr** – If specified, this overrides the standard way of referring to the construct you're zooming in on.
- **missing_msg** – If specified, this overrides the automatically generated feedback message in case the construct could not be found.
- **expand_msg** – If specified, this overrides the automatically generated feedback message that is prepended to feedback messages that are thrown further in the SCT chain.

Example Suppose you expect students to create a generator `my_gen` as follows:

```
my_gen = ( i*2 for i in range(0,10) )
```

The following SCT would robustly verify this:

```
Ex().check_correct(
    check_object('my_gen').has_equal_value(),
    check_generator_exp().multi(
        check_iter().has_equal_value(),
        check_body().set_context(4).has_equal_value()
    )
)
```

Have a look at `check_list_comp` to understand what's going on; it is very similar.

2.10 State management

override(*state*, *solution*)

Override the solution code with something arbitrary.

There might be cases in which you want to temporarily override the solution code so you can allow for alternative ways of solving an exercise. When you use `override()` in an SCT chain, the remainder of that SCT chain will run as if the solution code you specified is the only code that was in the solution.

Check the glossary for an example (pandas plotting)

Parameters

- **`solution`** – solution code as a string that overrides the original solution code.
- **`state`** – State instance describing student and solution code. Can be omitted if used with Ex().

`disable_highlighting`(*state*)

Disable highlighting in the remainder of the SCT chain.

Include this function if you want to avoid that pythonwhat marks which part of the student submission is incorrect.

Examples SCT that will mark the 'number' portion if it is incorrect:

```
Ex().check_function('round').check_args(0).has_equal_ast()
```

SCT chains that will not mark certain mistakes. The earlier you put the function, the more types of mistakes will no longer be highlighted:

```
Ex().disable_highlighting().check_function('round').check_args(0).has_
↪equal_ast()
Ex().check_function('round').disable_highlighting().check_args(0).has_
↪equal_ast()
Ex().check_function('round').check_args(0).disable_highlighting().has_
↪equal_ast()
```

`set_context`(*state*, **args*, ***kwargs*)

Update context values for student and solution environments.

When `has_equal_x()` is used after this, the context values (in `for` loops and function definitions, for example) will have the values specified through his function. It is the function equivalent of the `context_vals` argument of the `has_equal_x()` functions.

- Note 1: excess args and unmatched kwargs will be unused in the student environment.
- Note 2: When you try to set context values that don't match any target variables in the solution code, `set_context()` raises an exception that lists the ones available.
- Note 3: positional arguments are more robust to the student using different names for context values.
- Note 4: You have to specify arguments either by position, either by name. A combination is not possible.

Example Solution code:

```
total = 0
for i in range(10):
    print(i ** 2)
```

Student submission that will pass (different iterator, different calculation):

```
total = 0
for j in range(10):
    print(j * j)
```

SCT:

```
# set_context is robust against different names of context values.
Ex().check_for_loop().check_body().multi(
    set_context(1).has_equal_output(),
    set_context(2).has_equal_output(),
    set_context(3).has_equal_output()
)

# equivalent SCT, by setting context_vals in has_equal_output()
Ex().check_for_loop().check_body().\
    multi([s.has_equal_output(context_vals=[i]) for i in range(1, 4)])
```

set_env(*state*, ***kwargs*)

Update/set environemnt variables for student and solution environments.

When `has_equal_x()` is used after this, the variables specified through this function will be available in the student and solution process. Note that you will not see these variables in the student process of the state produced by this function: the values are saved on the state and are only added to the student and solution processes when `has_equal_ast()` is called.

Example Student and Solution Code:

```
a = 1
if a > 4:
    print('pretty large')
```

SCT:

```
# check if condition works with different values of a
Ex().check_if_else().check_test().multi(
    set_env(a = 3).has_equal_value(),
    set_env(a = 4).has_equal_value(),
    set_env(a = 5).has_equal_value()
)

# equivalent SCT, by setting extra_env in has_equal_value()
Ex().check_if_else().check_test().\
    multi([has_equal_value(extra_env={'a': i}) for i in range(3, 6)])
```

2.11 Checking files

check_file(*state: protowhat.State.State*, *path*, *missing_msg='Did you create the file '{}'?'*, *is_dir_msg='Want to check the file '{}', but found a directory.'*, *parse=True*, *solution_code=None*)

Test whether file exists, and make its contents the student code.

Parameters

- **state** – State instance describing student and solution code. Can be omitted if used with Ex().
- **path** – expected location of the file
- **missing_msg** – feedback message if no file is found in the expected location
- **is_dir_msg** – feedback message if the path is a directory instead of a file

- **parse** – If `True` (the default) the content of the file is interpreted as code in the main exercise technology. This enables more checks on the content of the file.
- **solution_code** – this argument can be used to pass the expected code for the file so it can be used by subsequent checks.

Note: This SCT fails if the file is a directory.

Example To check if a user created the file `my_output.txt` in the subdirectory `resources` of the directory where the exercise is run, use this SCT:

```
Ex().check_file("resources/my_output.txt", parse=False)
```

has_dir(*state: protowhat.State.State*, *path*, *msg='Did you create a directory '{}'?'*)
Test whether a directory exists.

Parameters

- **state** – State instance describing student and solution code. Can be omitted if used with Ex().
- **path** – expected location of the directory
- **msg** – feedback message if no directory is found in the expected location

Example To check if a user created the subdirectory `resources` in the directory where the exercise is run, use this SCT:

```
Ex().has_dir("resources")
```

run(*state*, *relative_working_dir=None*, *solution_dir='../solution'*, *run_solution=True*)
Run the focused student and solution code in the specified location

This function can be used after `check_file` to execute student and solution code. The arguments allow configuring the correct context for execution.

SCT functions chained after this one that execute pieces of code (custom expressions or the focused part of a file) execute in the same student and solution locations as the file.

Note: This function does not execute the file itself, but code in memory. This can have an impact when:

- the solution code imports from a different file in the expected solution (code that is not installed)
- using functionality depending on e.g. `__file__` and `inspect`

When the expected code has imports from a different file that is part of the exercise, it can only work if the solution code provided earlier does not have these imports but instead has all that functionality inlined.

Parameters

- **relative_working_dir** (*str*) – if specified, this relative path is the subdirectory inside the student and solution context in which the code is executed
- **solution_dir** (*str*) – a relative path, `solution` by default, that sets the root of the solution context, relative to that of the student execution context
- **state** (*State*) – state as passed by the SCT chain. Don't specify this explicitly.

If `relative_working_dir` is not set, it will be the directory the file was loaded from by `check_file` and fall back to the root of the student execution context (the working directory pythonwhat runs in).

The `solution_dir` helps to prevent solution side effects from conflicting with those of the student. If the set or derived value of `relative_working_dir` is an absolute path, `relative_working_dir` will not be used to form the solution execution working directory: the solution code will be executed in the root of the solution execution context.

Example Suppose the student and solution have a file `script.py` in `/home/repl/`:

```
if True:
    a = 1

print("Hi!")
```

We can check it with this SCT (with `file_content` containing the expected file content):

```
Ex().check_file(
    "script.py",
    solution_code=file_content
).run().multi(
    check_object("a").has_equal_value(),
    has_printout(0)
)
```

2.12 Bash history checks

get_bash_history(*full_history=False*, *bash_history_path=None*)

Get the commands in the bash history

Parameters

- **full_history** (*bool*) – if true, returns all commands in the bash history, else only return the commands executed after the last bash history info update
- **bash_history_path** (*str | Path*) – path to the bash history file

Returns a list of commands (empty if the file is not found)

Import from `from protowhat.checks import get_bash_history`.

has_command(*state*, *pattern*, *msg*, *fixed=False*, *commands=None*)

Test whether the bash history has a command matching the pattern

Parameters

- **state** – State instance describing student and solution code. Can be omitted if used with Ex().
- **pattern** – text that command must contain (can be a regex pattern or a simple string)
- **msg** – feedback message if no matching command is found
- **fixed** – whether to match text exactly, rather than using regular expressions
- **commands** – the bash history commands to check against. By default this will be all commands since the last bash history info update. Otherwise pass a list of commands to search through, created by calling the helper function `get_bash_history()`.

Note: The helper function `update_bash_history_info(bash_history_path=None)` needs to be called in the pre-exercise code in exercise types that don't have built-in support for bash history features.

Note: If the bash history info is updated every time code is submitted (by using `update_bash_history_info()` in the pre-exercise code), it's advised to only use this function as the second part of a `check_correct()` to help students debug the command they haven't correctly run yet. Look at the examples to see what could go wrong.

If bash history info is only updated at the start of an exercise, this can be used everywhere as the (cumulative) commands from all submissions are known.

Example The goal of an exercise is to use `man`.

If the exercise doesn't have built-in support for bash history SCTs, update the bash history info in the pre-exercise code:

```
update_bash_history_info()
```

In the SCT, check whether a command with `man` was used:

```
Ex().has_command("$man\s", "Your command should start with ``man ...
↪``.")
```

Example The goal of an exercise is to use `touch` to create two files.

In the pre-exercise code, put:

```
update_bash_history_info()
```

This SCT can cause problems:

```
Ex().has_command("touch.*file1", "Use `touch` to create `file1`")
Ex().has_command("touch.*file2", "Use `touch` to create `file2`")
```

If a student submits after running `touch file0 && touch file1` in the console, they will get feedback to create `file2`. If they submit again after running `touch file2` in the console, they will get feedback to create `file1`, since the SCT only has access to commands after the last bash history info update (only the second command in this case). Only if they execute all required commands in a single submission the SCT will pass.

A better SCT in this situation checks the outcome first and checks the command to help the student achieve it:

```
Ex().check_correct(
    check_file('file1', parse=False),
    has_command("touch.*file1", "Use `touch` to create `file1`")
)
Ex().check_correct(
    check_file('file2', parse=False),
    has_command("touch.*file2", "Use `touch` to create `file2`")
)
```

prepare_validation(*state: protowhat.State.State, commands: List[str], bash_history_path: Optional[str] = None*) → protowhat.State.State

Let the exercise validation know what shell commands are required to complete the exercise

Import using `from protowhat.checks import prepare_validation`.

Parameters

- **state** – State instance describing student and solution code. Can be omitted if used with Ex().
- **commands** – List of strings that a student is expected to execute
- **bash_history_path** (*str | Path*) – path to the bash history file

Example The goal of an exercise is to run a build and check the output.

At the start of the SCT, put:

```
Ex().prepare_validation(["make", "cd build", "ls"])
```

Further down you can now use `has_command`.

update_bash_history_info(*bash_history_path=None*)

Store the current number of commands in the bash history

`get_bash_history` can use this info later to get only newer commands.

Depending on the wanted behaviour this function should be called at the start of the exercise or every time the exercise is submitted.

Import using `from protowhat.checks import update_bash_history_info`.

2.13 Electives

has_chosen(*state*, *correct*, *msgs*)

Test multiple choice exercise.

Test for a MultipleChoiceExercise. The correct answer (as an integer) and feedback messages are passed to this function.

Parameters

- **correct** (*int*) – the index of the correct answer (should be an instruction). Starts at 1.
- **msgs** (*list(str)*) – a list containing all feedback messages belonging to each choice of the student. The list should have the same length as the number of options.

success_msg(*message*)

Set the succes message of the sct. This message will be the feedback if all tests pass. :param message: A string containing the feedback message. :type message: str

allow_errors(*state*)

Allow running the student code to generate errors.

This has to be used only once for every time code is executed or a different xwhat library is used. In most exercises that means it should be used just once.

Example The following SCT allows the student code to generate errors:

```
Ex().allow_errors()
```

fail(*state*, *msg='fail'*)

Always fails the SCT, with an optional msg.

This function takes a single argument, `msg`, that is the feedback given to the student. Note that this would be a terrible idea for grading submissions, but may be useful while writing SCTs. For example, failing a test will highlight the code as if the previous test/check had failed.

Example As a trivial SCT example,

```
Ex().check_for_loop().check_body().fail()
```

This can also be helpful for debugging SCTs, as it can be used to stop testing as a given point.

CHAPTER 3

Tutorial

pythonwhat uses the `.` to 'chain together' SCT functions. Every chain starts with the `Ex()` function call, which holds the exercise state. This exercise state contains all the information that is required to check if an exercise is correct, which are:

- the student submission and the solution as text, and their corresponding parse trees.
- a reference to the student process and the solution process.
- the output and errors that were generated when executing the student code.

As SCT functions are chained together with `.`, the `Ex()` exercise state is copied and adapted into 'sub states' to zoom in on particular parts of the state. Before this terminology blows your brains out, let's have a look at some basic examples.

3.1 Example 1: output

Assume we want to robustly check whether a student correctly printed out a sentence:

```
print('hi, my name is DataCamp')
```

The following SCT would do that:

```
Ex().has_output(r'[H|h]i,\s+my name is \w+')
```

Let's see what happens when the SCT runs:

- `Ex()` returns the 'root state', which considers the entire student submission and solution code, a reference to the student and solution process, and the output and errors generated.
- `has_output(r'<regex>')` fetches the output the student generated from the root state and checks whether it can match the specified regular expression against it.
  - If the student had submitted `print('Hi, my name is Filip')`, the regex will match, the SCT will pass, and the student is presented with a congratulatory message.

  - If the student had submitted `print('Hi,mynameis'_`, the regex will not have found a match, the SCT will fail, and pythonwhat will automatically generate a feedback message.

3.2 Example 2: function call

Assume we want to check whether a student correctly called the `DataFrame` function of the `pandas` package.

```
import pandas as pd
pd.DataFrame([1, 2, 3])
```

The following SCT would do that:

```
Ex().check_function('pandas.DataFrame').check_args('data').has_equal_value()
```

Assume the student submits the following (incorrect) script:

```
import pandas as pd
pd.DataFrame([1, 2, 3, 4])
```

Let's see what happens when the SCT runs:

- `Ex()` returns the 'root state', which considers the entire student submission and solution code:

```
# solution
import pandas as pd
pd.DataFrame([1, 2, 3])

# student
import pandas as pd
pd.DataFrame([1, 2, 3, 4])
```

- `check_function('pandas.DataFrame')` continues from the root state (considering the entire student submission and solution), and looks for a call of `pd.DataFrame` in both. It finds them, and 'zooms in' on the arguments. In simplified terms, this is the state that `check_function()` produces:

```
# solution args
{ "data": [1, 2, 3] }

# student arg
{ "data": [1, 2, 3, 4] }
```

- `check_args('data')` continues from the state produced by `check_function()` and looks for the `"data"` argument in both the student and solution arguments. It finds it in both and produces a state that zooms in on the expression used to specify this argument:

```
# solution expression for data arg
[1, 2, 3]

# student expression for data arg
[1, 2, 3, 4]
```

- Finally, `has_equal_value()` takes the state produced by `check_args()`, executes the student and solution expression in their respective processes, and verifies if they give the same result. In this example, the results of the expressions don't match: a 3-element array vs a 4-element array. Hence, the SCT fails and automatically generates a meaningful feedback message.

3.3 Example 3: if statement

As a more advanced example, assume we want to check that the student coded up an *if* statement correctly:

```
x = 4
if x > 0:
    print("x is strictly positive")
```

The following SCT would do that:

```
Ex().check_if_else().multi(
    check_test().has_code(r'x\s+>\s+0'), # chain A
    check_body().check_function('print').check_args(0).has_equal_value() # chain B
    )
```

Notice how this time, `multi()` is used to have the SCT chains 'branch out'; both `check_body()` and `check_test()` continue from the state produced by `check_if_else()`.

3.3.1 Case 1

In the first case, assume the following incorrect student submission:

```
x = 4
if x < 0:
    print("x is negative")
```

In chain A, this is what happens:

- `check_if_else()` considers the entire submission received from `Ex()`, looks for the first if-else statement in both student and solution code, and produces a child state that zooms in on onlty these `if` statements:

  ```
  # solution
  if x > 0:
      print("x is strictly positive")

  # student
  if x < 0:
      print("x is negative")
  ```

- `check_test()` considers the state above produced by `check_if_else()` and produces a child state that zooms in on the condition parts of the `if` statements:

  ```
  # solution
  x > 0

  # student
  x < 0
  ```

- `has_code()` considers the state above produced by `check_test()` and tries to match the regexes to the `x < 0` student snippet. The regex does not match, so the test fails.

3.3.2 Case 2

Assume now that the student corrects the mistake and submits the following (which is still not correct):

```
x = 4
if x > 0:
    print("x is negative")
```

Chain A will go through the same steps and will pass this time as `x > 0` in the student submission now matches the regex. In Chain B:

- `check_body()` considers the state produced by `check_if_else()`, and produces a child state that zooms in on the body parts of the `if` statements:

```
# solution
print("x is strictly positive")

# student
print("x is negative")
```

- `check_function()` considers the state above produced by `check_if_else()`, and tries to find the function `print()`. Next, it produces a state that refers to the different function arguments and the expressions used to specify them:

```
# solution
{ "value": "x is strictly positive" }

# student
{ "value": "x is negative" }
```

- `check_args(0)` looks for the first argument in the state produced by `check_function()` and produces a child state that zooms in on the expressions for the `value` argument:

```
# solution
"x is strictly positive"

# student
"x is negative"
```

- Finally, `has_equal_value()` takes the state produced by `check_args()`, executes the student and solution expression in their respective processes, and verifies if they give the same result. The result of executing `"x is strictly positive"` and `"x is negative"` don't match so the SCT fails.

Caution: We strongly advise against using `has_code()` to verify the correctness of excerpts of a student submission. Visit the 'checking compount statements' article to take a deeper dive.

3.4 What is good feedback?

For larger exercises, you'll often want to be flexible: if students get the end result right, you don't want to be picky about how they got there. However, when they do make a mistake, you want to be specific about the mistake they are making. In other words, a good SCT is robust against different ways of solving a problem, but specific when something's wrong.

These seemingly conflicting requirements can be satisfied with `check_correct()`. It is an **extremely powerful function** that should be used whenever it makes sense. The Make your SCT robust article is highly recommended reading.

For other guidelines on writing good SCTs, check out the 'How to write good SCTs' section on DataCamp's general SCT documentation page.

CHAPTER 4

Checking function calls

4.1 Basic functionality

Take the following example that checks whether a student used the `round()` function correctly:

```
# solution
round(2.718282, ndigits = 3)

# sct
Ex().check_function("round").multi(
    check_args("number").has_equal_value(),
    check_args("ndigits").has_equal_value()
)

# submissions that pass:
round(2.718282, 3)
round(2.718282, ndigits = 3
round(number=2.718282, ndigits=3)
round(ndigits=3, number=2.718282)
val=2.718282; dig=3; round(val, dig)
val=2.718282; dig=3; round(number=val, dig)
int_part = 2; dec_part = 0.718282; round(int_part + dec_part, 3)
```

- *check_function()* checks whether `round()` is called by the student, and parses all the arguments.
- `check_args()` checks whether a certain argument was specified, and zooms in on the expression used to specify that argument.
- `has_equal_value()` will rerun the expressions used to specify the arguments in both student and solution process, and compare the results.

Note: In `check_args()` you can refer to the argument of a function call both by argument name and by position.

4.1.1 Customizations

If you only want to check the `number` parameter, just don't include a second chain with `check_args("ndigits")`:

```
Ex().check_function("round").check_args("number").has_equal_value()
```

If you only want to check whether the `number` parameter was specified, but not that it was specified correctly, drop `has_equal_value()`:

```
Ex().check_function("round").check_args("number")
```

If you just want to check whether the function was called, drop `check_args()`:

```
Ex().check_function("round")
```

If you want to compare the 'string versions' of the expressions used to set the arguments instead of the evaluated result of these expressions, you can use `has_equal_ast()` instead of `has_equal_value()`:

Now, the following submissions would fail:

- `val=2.718282; dig=3; round(val, dig)` – the string representation of `val` in the student code is compared to `2.718282` in the solution code.
- `val=2.718282; dig=3; round(number=val, dig)` – same
- `int_part = 2; dec_part = 0.718282; round(int_part + dec_part, 3)` – the string representation of `int_part + dec_part` in the student code is compered to `2.718282` in the solution code.

As you can see, doing exact string comparison of arguments is not a good idea here, as it is very inflexible. There are cases, however, where it makes sense to use this, e.g. when there are very big objects passed to functions, and you don't want to spend the processing power to fetch these objects from the student and solution processes.

4.2 Functions in packages

If you're testing whether function calls of particular packages are used correctly, you should always refer to these functions with their 'full name'. Suppose you want to test whether the function `show` of `matplotlib.pyplot` was called, use this SCT:

```
Ex().check_function("matplotlib.pyplot.show")
```

`check_function()` can handle it when a student used aliases for the python packages (all `import` and `import * from *` calls are supported). If the student did not properly call the function, `check_function()` will automatically generate a feedback message that corresponds to how the student imported the modules/functions.

4.3 has_equal_value? has_equal_ast?

In the customizations section above, you could already notice the difference between `has_equal_value()` and `has_equal_ast()` for checking whether arguments are correct. The former **reruns** the expression used to specify the argument in both student and solution process and compares their results, while the latter simply compares the expression's AST representations. Clearly, the former is more robust, but there are some cases in which `has_equal_ast()` can be useful:

- For better feedback. When using `has_equal_ast()`, the 'expected x got y' message that is automatically generated when the arguments don't match up will use the actual expressions used. `has_equal_value()` will use string representations of the evaluations of the expressions, if they make sense, and this is typically less useful.
- To avoid very expensive object comparisons. If you are 100% sure that the object people have to pass as an argument is already correct (because you checked it earlier in the SCT or because it was already specified in the pre exercise code) and doing an equality check on this object between student and solution project is likely going to be expensive, then you can safely use `has_equal_ast()` to speed things up.
- If you want to save yourself the trouble of building exotic contexts. You'll often find yourself checking function calls in e.g. a for loop. Typically, these function calls will use objects that were generated inside the loop. To easily unit test the body of a for loop, you'll typically have to use `set_context()` and `set_env()`. For exotic for loops, this can become tricky, and it might be a quick fix to be a little more specific about the object names people should use, and just use `has_equal_ast()` for the argument comparison. That way, you're bypassing the need to build up a context in the student/solution process and do object comparisions.

4.4 Signatures

The `round()` example earlier in this article showed that a student can call the function in a multitude of ways, specifying arguments by position, by keyword or a mix of those. To be robust against this, pythonwhat uses the concept of argument binding.

More specifically, each function has a function signature. Given this signature and the way the function was called, argument binding can map each parameter you specified to an argument. This small demo fetches the signature of the `open` function and tries to bind arguments that have been specified in two different ways. Notice how the resulting bound arguments are the same:

```
>>> import inspect

>>> sig = inspect.signature(open)

>>> sig
<Signature (file, mode='r', buffering=-1, encoding=None, errors=None, newline=None,␣
↪closefd=True, opener=None)>

>>> sig.bind('my_file.txt', mode = 'r')
<BoundArguments (file='my_file.txt', mode='r')>

>>> sig.bind(file = 'my_file.txt', mode = 'r')
<BoundArguments (file='my_file.txt', mode='r')>
```

When you're using `check_args()` you are actually selecting these bound arguments. This works fine for functions like `round()` and `open()` that have a list of named arguments, but things get tricky when dealing with functions that take `*args` and `*kwargs`.

4.4.1 `*args` example

Python allows functions to take a variable number of unnamed arguments through `*args`, like this function:

```
def multiply(*args):
    res = 1
    for num in args:
```

(continues on next page)

(continued from previous page)

```
        res *= num
    return res
```

Let's see what happens when different calls are bound to their arguments:

```
>>> import inspect

>>> inspect.signature(multiply)
<Signature (*args)>

>>> sig = inspect.signature(multiply)

>>> sig
<Signature (*args)>

>>> sig.bind(1, 2)
<BoundArguments (args=(1, 2))>

>>> sig.bind(3, 4, 5)
<BoundArguments (args=(3, 4, 5))>
```

Notice how now the list of arguments is grouped under a tuple with the name `args` in the bound arguments. To be able to check each of these arguments individually, pythonwhat allows you to do repeated indexing in `check_args()`. Instead of specifying the name of an argument, you can specify a list of indices:

```
# solution to check against
multiply(2, 3, 4)

# corresponding SCT
Ex().check_function("multiply").multi(
    check_args(["args", 0]).has_equal_value(),
    check_args(["args", 1]).has_equal_value(),
    check_args(["args", 2]).has_equal_value()
)
```

The `check_args()` subchains each zoom in on a particular tuple element of the bound `args` argument.

4.4.2 `**kwargs` example

Python allows functions to take a variable number of named arguments through `**kwargs`, like this function:

```
def my_dict(**kwargs):
    return dict(**kwargs)
```

Let's see what happens when different calls are bound to their arguments:

```
>>> import inspect

>>> sig = inspect.signature(my_dict)

>>> sig.bind(a = 1, b = 2)
<BoundArguments (kwargs={'b': 2, 'a': 1})>

>>> sig.bind(c = 2, b = 3)
<BoundArguments (kwargs={'b': 3, 'c': 2})>
```

Notice how now the list of arguments is grouped under a dictionary name `kwargs` in the bound arguments. To be able to check each of these arguments individually, pythonwhat allows you to do repeated indexing in `check_args()`. Instead of specifying the name of an argument, you can specify a list of indices:

```
# solution to check against
my_dict(a = 1, b = 2)

# corresponding SCT
Ex().check_function("my_dict").multi(
    check_args(["kwargs", "a"]).has_equal_value(),
    check_args(["kwargs", "b"]).has_equal_value()
)
```

The `check_args()` subchains each zoom in on a particular dictionary element of the bound `kwargs` argument.

4.4.3 Manual signatures

Unfortunately for a lot of Python's built-in functions no function signature is readily available because the function has been implemented in C code. To work around this, pythonwhat already includes manually specified signatures for functions such as `print()`, `str()`, `hasattr()`, etc, but it's still possible that some signatures are missing.

That's why `check_function()` features a `signature` parameter, that is `True` by default. If pythonwhat can't retrieve a signature for the function you want to test, you can pass an object of the class `inspect.Signature` to the `signature` parameter.

Suppose, for the sake of example, that `check_function()` can't find a signature for the `round()` function. In a real situation, you will be informed about a missing signature through a backend error. To be able to implement this SCT, you can use the `sig_from_params()` function:

```
sig = sig_from_params(param("number", param.POSITIONAL_OR_KEYWORD),
                      param("ndigits", param.POSITIONAL_OR_KEYWORD, default=0))
Ex().check_function("round", signature=sig).multi(
    check_args("number").has_equal_value(),
    check_args("ndigits").has_equal_value()
)
```

You can pass `sig_from_params()` as many parameters as you want.

`param` is an alias of the `Parameter` class that's inside the `inspect` module. - The first argument of `param()` should be the name of the parameter, - The second argument should be the 'kind' of parameter. `param.POSITIONAL_OR_KEYWORD` tells `check_function` that the parameter can be specified either through a positional argument or through a keyword argument. Other common possibilities are `param.POSITIONAL_ONLY` and `param.KEYWORD_ONLY` (for a full list, refer to the docs). - The third optional argument allows you to specify a default value for the parameter.

Note: If you find vital Python functions that are used very often and that are not included in pythonwhat by default, you can let us know and we'll add the function to our list of manual signatures.

4.5 Multiple function calls

Inside `check_function()` the `index` argument (`0` by default), becomes important when there are several calls of the same function. Suppose that your exercise requires the student to call the `round()` function twice: once on `pi` and once on Euler's number:

```
# Call round on pi
round(3.14159, 3)

# Call round on e
round(2.71828, 3)
```

To test both these function calls, you'll need the following SCT:

```
Ex().check_function("round", 0).multi(
    check_args("number").has_equal_value()
    check_args("ndigits").has_equal_value()
)
Ex().check_function("round", 1).multi(
    check_args("number").has_equal_value()
    check_args("ndigits").has_equal_value()
)
```

The first `check_function()` chain, where `index=0`, looks for the first call of `round()` in both student solution code, while `check_funtion()` with `index=1` will look for the second function call. After this, the rest of the SCT chain behaves as before.

4.6 Methods

Methods are Python functions that are called on objects. For testing this, you can also use `check_function()`. Consider the following examples, that calculates the `mean()` of the column `a` in the pandas data frame `df`:

```
# pec
import pandas as pd
df = pd.DataFrame({ 'a': [1, 2, 3, 4] })

# solution
df.a.mean()

# sct
Ex().check_function('df.a.mean').has_equal_value()
```

The SCT is checking whether the method `df.a.mean` was called in the student code, and whether rerunning the call in both student and solution process is returning the same result.

As a more advanced example, consider this example of chained method calls:

```
# pec
import pandas as pd
df = pd.DataFrame({ 'type': ['a', 'b', 'a', 'b'], 'val': [1, 2, 3, 4] })

# solution
df.groupby('type').mean()

# sct
Ex().check_function('df.groupby').check_args(0).has_equal_value()
Ex().check_function('df.groupby.mean', signature=sig_from_obj('df.mean')).has_equal_
↪value()
```

Here:

- The first SCT is checking whether `df.groupby()` was called and whether the argument for `df.groupby()` was specified correctly to be `'type'`.
- The second SCT is first checking whether `df.groupby.mean()` was called and whether calling it gives the right result. Notice several things:
  - We describe the entire chain of method calls, leaving out the parentheses and arguments used for method calls in between.
  - We use `sig_from_obj()` to manually specify a Python expression that pythonwhat can use to derive the signature from. If the string you use to describe the function to check evaluates to a method or function in the solution process, like for `'df.groupby'`, pythonwhat can figure out the signature. However, for `'df.groupby.mean'` will *not* evaluate to a method object in the solution process, so we need to manually specify a valid expression that *will* evaluate to a valid signature with `sig_from_obj()`.

In this example, you are only checking whether the function is called and whether rerunning it gives the correct result. You are not checking the actual arguments, so there's actually no point in trying to match the function call to its signature. In cases like this, you can set `signature=False`, which skips the fetching of a signature and the binding or arguments altogether:

```
# pec
import pandas as pd
df = pd.DataFrame({ 'type': ['a', 'b', 'a', 'b'], 'val': [1, 2, 3, 4] })

# solution
df.groupby('type').mean()

# sct
Ex().check_function('df.groupby').check_args(0).has_equal_value()
Ex().check_function('df.groupby.mean', signature=False).has_equal_value()
```

Warning: Watch out with disabling signature binding as a one-stop solution to make your SCT run without errors. If there are arguments to check, argument binding makes sure that various ways of calling the function can all work. Setting `signature=False` will skip this binding, which can cause your SCT to mark perfectly valid student submissions as incorrect!

Note: You can also use the `sig_from_params()` function to manually build the signature from scratch, but this this more work than simply specifying the function object as a string from which to extract the signature.

CHAPTER 5

Make your SCT robust

For larger exercises, you'll often want to be flexible: if students get the end result right, you don't want to be picky about how they got there. However, when they do make a mistake, you want to be specific about the mistake they are making. These seemingly conflicting requirements can be satisfied with `check_correct()` and `check_or()`.

5.1 `check_correct()`

To explain the concept of `check_correct()`, consider this example:

```
# setup
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6])

# calculate result
result = np.mean(arr)
```

You want the SCT to pass when the student manages to store the correct value in the object `result`. How `result` was calculated, does not matter to you: as long as `result` is correct, the SCT should accept the submission. If something about `result` is not correct, you want to dig a little deeper and see if the student used the `np.mean()` function correctly. The following SCT will do just that:

```
Ex().check_correct(
    check_object("result").has_equal_value(),
    check_function("numpy.mean").check_args("a").has_equal_value()
)
```

Inside `check_correct()`, two SCT chains are specified, separated by a comma:

- A `check` chain, that has to pass in all cases, but when it fails, it doesn't immediately stop the SCT execution and fail the exercise.
- A `diagnose` chain, that is only execute if the `check` chain failed silently.

In the example, we're checking the end value of `result` first. Only if this is not correct, will the `check_function()` chain be run, to verify if the student used `numpy.mean`. If the `diagnose` chain does not fail, the `check` chain is executed again 'loudly'.

Let's see what happens in case of different student submissions:

- The student submits `result = np.mean(arr)`
  - `check_correct()` runs the `check_object()` chain.
  - This test passes, so `check_correct()` stops.
  - The SCT passes.
- The student submits `result = np.sum(arr) / arr.size`
  - `check_correct()` runs the `check_object()` chain.
  - This test passes, so `check_correct()` stops before running `check_function()`.
  - The entire SCT passes even though `np.mean()` was not used.
- The student submits `result = np.mean(arr + 1)`
  - `check_correct()` runs the `check_object()` chain.
  - This test fails, so `check_correct()` continues with the `diagnose` part, running the `check_function()` chain.
  - This chain fails, since the argument passed to `numpy.mean()` in the student submission does not correspond to the argument passed in the solution.
  - A meaningful, specific feedback message is presented to the student: you did not correctly specify the arguments inside `np.mean()`.
- The student submits `result = np.mean(arr) + 1`
  - `check_correct()` runs the `check_object()` chain.
  - This test fails, so `check_correct()` continues with the `diagnose` part, running the `check_function()` chain.
  - This function passes, because `np.mean()` is called in exactly the same way in the student code as in the solution.
  - Because there is something wrong - `result` is not correct - the `check` chain is executed again, and this time its feedback on failure is presented to the student.
  - The student gets the message that `result` does not contain the correct value.

5.1.1 Multiple functions in `diagnose` and *check*

It is perfectly possible for your `check` and `diagnose` SCT chains to branch out into different sub-branches with `multi()`:

```
Ex().check_correct(
    multi(
      check_object('a').has_equal_value(), # multiple check SCTs
      check_object('b').has_equal_value()
    ),
    check_function("numpy.mean").check_args("a").has_equal_value()
)
```

5.1.2 Why use *check_correct()*

You will find that `check_correct()` is an **extremely powerful function&** to allow for different ways of solving the same problem. You can use `check_correct()` to check the end result of a calculation. If the end result is correct, you can go ahead and accept the entire exercise. If the end result is incorrect, you can use the `diagnose` part of `check_correct()` to dig a little deeper.

It is also perfectly possible to use `check_correct()` inside another `check_correct()`.

5.2 `check_or()`

`check_or()` tests whether one of the SCTs you specify inside it passes. Suppose you want to check whether people correctly printed out any integer between 3 and 7. A solution could be:

```
print(4)
```

To test this in a robust way, you could use `has_code()` with a suitable regular expression that covers everything, or you can use `check_or()` with three separate `has_code()` functions:

```
Ex().check_or(has_code('4'),
              has_code('5'),
              has_code('6'))
```

You can consider `check_or()` a logic-inducing function. The different calls to pythonwhat functions that are in your SCT are actually all tests that _have_ to pass: they are `AND` tests. With `check_or()` you can add chunks of `OR` tests in there.

CHAPTER 6

Checking through string matching

6.1 has_code

With `has_code()`, you can look through the student's submission to find a match with a search pattern you have specified.

- If `pattern = True`, the default, the `text` is used as a regular expression to match against.
- If `pattern = False`, `has_code()` will consider the text you pass as an actual string that has to be found exactly.

Caution: It is often tempting to use `has_code()` as it's straightforward to use, but **you should avoid using this function**, as it imposes severe restrictions on how a student can solve an exercise. Often, there are many different ways to solve an exercise. Unless you have a very advanced regular expression, `has_code()` will not be able to accept all these different approaches. Always think about better ways to test a student submission before you resort to `has_code()`.

Take the following example:

```
# solution
s = sum(range(10))

# sct that checks whether sum(range( is in the code
Ex().has_code("sum\s*\(\s*range\s*\(", not_typed_msg="You didn't use ``range()``␣
↪inside ``sum()``.")
```

We also used `not_typed_msg` here to specify the feedback message shown to the student if `has_code()` doesn't pass.

6.2 has_equal_ast

AST stands for *abstract syntax tree*; it is a way of representing the high-level structure of python code. As the name suggests, `has_equal_ast()` verifies whether the code portion under consideration has the same AST representation in student and solution. Compared to `has_code()`, it is more robust to small syntactical details that are equivalent.

- Quotes: the AST for `x = "1"` or `x = '1'` will be the same.
- Parentheses: Grouping by parentheses produces the same AST, when the same statement would work the same without them. `(True or False) and True`, and `True or False and True`, are the same due to operator precedence.
- Spacing: `x = 1` or `x = 1` have the same AST.

The AST does **not** represent is values that are found through evaluation. For example, the first item in the list in

```
x = 1
[x, 2, 3]
```

and

```
[1, 2, 3]
```

Is not the same. In the first case, the AST represents that a variable `x` needs to be evaluated in order to find out what its value is. In the second case, it just represents the value `1`.

Caution: Note that it is *not* a good idea to use `Ex().has_equal_ast()`, effectively comparing the entire solution with the entire student submission. It *is* a good idea, however, to use *has_equal_ast* for checking small excerpts of code when checking compound statements, for example to inspect the test part of an `if` statement.

As an example, consider this example that checks whether a student correclty coded a condition in a for loop (note that there are better ways to check this with `has_equal_value()`!):

```
# solution
x = 3
if x % 2 == 0:
    print('x is even')

# sct
Ex().check_if_else().multi(
    check_test().has_equal_ast(),
    check_body().has_equal_output()
)

# passing submission 1
x = 3
if (x % 2 == 0):
    print('x is even')

# passing submission 2
x = 3
if x%2==0:
    print('x is even')

# failing submission
```

(continues on next page)

(continued from previous page)

```
x = 3
if 0 == x % 2:
    print('x is even')
```

Here, the `Ex().check_if_else().check_test()` chain zooms in on the test part of the if statement. With `has_equal_ast()` you are checking whether the AST representation of the test in the solution, `x % 2 == 0` is also found in the test specified by the student. Notice that `has_equal_ast()` is not robust against a simple switching the order of the operands of the == operator. A better SCT here would not use string (or AST) matching in the first place and rerun the test for different values of `x:

```
Ex().check_if_else().multi(
    check_test().multi(
        set_env(x = 3).has_equal_value(),
        set_env(x = 4).has_equal_value(),
        set_env(x = 4).has_equal_value()
    ),
    check_body().has_equal_output()
)
```

CHAPTER 7

Checking compound statements

As described in the official Python documentation, *compound statements contain (groups of) other statements; they affect or control the execution of those other statements in some way. In general, compound statements span multiple lines, although in simple incarnations a whole compound statement may be contained in one line.*

`if`, `while`, `for`, `try`, and `with` statements are all examples of compounds statements, and pythonwhat contains functionality to check all of these, - as well as function definitions, list and dictionary comprehensions, generator expressions and lambda functions - in a consistent fashion.

7.1 Inner workings

The `if` statement example in the tutorial describes how different `check_` functions zoom into a specific part of a submission and solution, producing a child state, to which additional SCT functions can be chained. The `check_if_else()` function scanned the code for an `if` statement, and broke it into three parts: a `test`, the `body` and the `orelse` part; the former two were dived into with the SCT functions `check_test` and `check_body`. Notice that the naming is consistent: the `test` part that `check_if_else()` surfaces can be inspected with `check_test()`. The `body` part that `check_if_else()` unearths can be inspected with `check_body`.

Similar to how `if` statements has a `check_if_else` associated with it, all other compound statements have corresponding `check_` functions to perform this action of looking up a statement, and chopping it up into its constituents that can be inspected with `check_<part>()`:

- `check_for_loop()` will look for a `for` loop, an break it up into a `iter`, `body` and `orelse` part, that can be zoomed in on using `check_iter()`, `check_body()` and `check_orelse()` respectively.
- `check_list_comp()` will look for a list comprehension and break it up into a `iter`, `ifs` and `body` part, that can be zoomed in on using `check_iter()`, `check_ifs()` and `check_body()` respectively.
- etc.

For specific examples on checking for loops, list comprehensions, function definitions etc., visit the reference. Every function is documented with a full example and corresponding explanation. All of these examples are specific to a single construct, but of course you can combine things up to crazy levels.

7.2 Crazy combo, example 1

Suppose you want to check whether a function definition containing a for loop was coded correctly as follows:

```
def counter(lst, key):
    count = 0
    for l in lst:
        count += l[key]
    return count
```

The following SCT would robustly verify this:

```
Ex().check_function_def('counter').check_correct(
    multi(
        check_call("f([{'a': 1}], 'a')").has_equal_value(),
        check_call("f([{'b': 1}, {'b': 2}], 'b')").has_equal_value()
    ),
    check_body().set_context([{'a': 1}, {'a': 2}], 'a').set_env(count = 0).check_for_
↪loop().multi(
        check_iter().has_equal_value(),
        check_body().set_context({'a': 1}).has_equal_value(name = 'count')
    )
)
```

Some notes about this SCT:

- `check_correct()` is used so the body is not further checked if calling the function in different ways produces the same value in both student and solution process.
- `set_context()` is used twice. Once to set the context variables introduced by the function definition, and once to set the context variable introducted by the for loop.
- `set_env()` had to be used to initialize `count` to a variable that was scoped only to the function definition.

7.3 Overview of all supported compound statements

The table below summarizes all checks that pythonwhat supports to test compound statements.

- Code in all caps indicates the name of a piece of code that may be inspected using `check_{part}`, where `{part}` is replaced by the name in caps (e.g. `check_if_else().check_test()`).
- If the statement produces context variables, these are referred to in the parts column and listed in the context variables column. The names used are just to refer to which context variable comes from where; you are totally free in naming your context variables.

check	parts	context variables

check_if_else()	`if TEST:`
	`  BODY`
	`else:`
	`  ORELSE`	

check_while()	`while TEST:`
	`  BODY`
	`else:`
	`  ORELSE`	

check_list_comp()	`[BODY for i in ITER if IFS[0] if IFS[1]]`	i
check_generator_exp()	`(BODY for i in ITER if IFS[0] if IFS[1])`	i
check_dict_comp()	`{KEY : VALUE for k, v in ITER if IFS[0]}`	k, v

check_for_loop()	`for i, j in ITER:`
	`  BODY`
	`else:`
	`  ORELSE`
		i, j

check_try_except()	`try:`
	`  BODY`
	`except BaseException as e:`
	`  HANDLERS['BaseException']`
	`except:`
	`  HANDLERS['all']`
	`else:`
	`  ORELSE`
	`finally:`
	`  FINALBODY`
		e

check_with()	`with CONTEXT[0] as f1, CONTEXT[1] as f2:` `  BODY`	f
check_function_def('f')	`def f(ARGS[0], ARGS[1]):` `  BODY`	argument names
check_lambda_function()	`lambda ARGS[0], ARGS[1]: BODY`	argument names
check_class_def('f')	`class KLS(BASES[0], BASES[1]):` `  BODY`	

CHAPTER 8

Expression tests

Expression tests run pieces of the student and solution code, and then check the resulting value, printed output, or errors they produce.

8.1 `has_equal` syntax

Once student/submission code has been selected using a check function, we can run it using one of three functions. They all take the same arguments, and run the student and submission code in the same way. However, they differ in how they compare the outcome:

- `has_equal_value()` - compares the value returned by the code.
- `has_equal_output()` - compares printed output.
- `has_equal_error()` - compares any errors raised.

8.2 Basic Usage

8.2.1 Running the whole code submission

In the example below, we re-run the entire student and submission code, and check that they print out the same output.

```
# solution
x = [1,2,3]
print(x)

# sct
Ex().has_equal_output()
```

Note that while we could have used `has_output()` to verify that the student printed `"[1, 2, 3]"`, using `has_equal_output` simply requires that the student output matches the solution output.

8.2.2 Running part of the code

Combining an expression test with part checks will run only a piece of the submitted code. The example below first uses `has_equal_value` to run an entire if expression, and then to run only its body.

```
# solution
x = [1,2,3]
sum(x) if x else None

# sct to test body of if expression
(Ex().check_if_exp()        # focus on if expression
     .has_equal_value()    # run entire if expression, check value
     .check_body()         # focus on body "sum(x)"
     .has_equal_value()    # run body, check value
     )
```

Note: Because `has_equal_value()` returns the exact same state as it was passed, commands chaining off of `has_equal_value` behave as they would have if `has_equal_value` weren't used.

8.3 Context Values

Suppose we want the student to define a function, that loops over the elements in a dictionary, and prints out each key and value, as follows:

```
# solution
def print_dict(my_dict):
    for key, value in my_dict.items():
        print(key + " - " + str(value))
```

An appropriate SCT for this exercise could be the following (for clarity, we're not using any default messages):

```
# get for loop code, set context for my_dict argument
for_loop = (Ex()
     .check_function_def('print_dict')           # ensure 'print_dict' is defined
     .check_body()                               # get student/solution code in body
     .set_context(my_dict = {'a': 2, 'b': 3})    # set print_dict's my_dict arg
     .check_for_loop()                          # ensure for loop is defined
     )

# test for loop iterator
for_loop.check_iter().has_equal_value()          # run iterator (my_dict.items())
# test for loop body
for_loop.check_body().set_context(key = 'c', value = 3).has_equal_value()
```

Assuming the student coded the function in the exact same way as the solution, the following things happen:

- checks whether `print_dict` is defined, then gets the code for the function definition body.
- because `print_dict` takes an argument `my_dict`, which would be undefined if we ran the body code, `set_context` defines what `my_dict` should be when running the code. Note that its okay if the submitted code named the argument `my_dict` something else, since set_context matches submission / solution arguments up by position.

When running the bottom two SCTs for the for_loop

- `for_loop.check_iter().has_equal_value()` - runs the code for the iterator, `my_dict.items()` in the solution and its corresponding code in the submission, and compares the values they return.
- `for_loop.check_body().set_context(key = 'c', value = 3).has_equal_value()` - runs the code in the for loop body, `print(key + " - " + str(value))` in the solution, and compares outputs. Since this code may use variables the for loop defined, `key` and `value`, we need to define them using `set_context`.

8.3.1 How are context values matched?

Context values are matched by position. For example, the submission and solution codes...

```
# solution
for ii, x in enumerate(range(3)): print(ii)

# student submission
for jj, y in enumerate(range(3)): print(jj)
```

Using `Ex().check_for_loop().check_body().set_context(...)` will do the following...

statement	solution (ii, x)	submission (jj, y)
set_context(ii=1, x=2)	ii = 1, x = 2	jj = 1, y = 2
set_context(ii=1)	ii = 1, x is undefined	jj = 1, y is undefined
set_context(x=2)	ii is undefined, x = 2	jj is undefined, y = 2

Note: If `set_context` does not define a variable, nothing is done with it. This means that in the code examples above, running the body of the for loop would call print with ::ii:: or ::jj:: left at 2 (the values they have in the solution/submission environments).

8.3.2 Context values for nested parts

Context values may now be defined for nested parts. For example, the print statement below,

```
for i in range(2):              # outer for loop part
    for j in range(3):          # inner for loop part
        print(i + j)
```

may be tested by setting context values at each level,

```
(Ex()
    .check_for_loop().check_body().set_context(i = 1)     # outer for
    .check_for_loop().check_body().set_context(j = 2)     # inner for
        .has_equal_output()
    )
```

8.4 `pre_code`: fixing mutations

Python code commonly mutates, or changes values within an object. For example, the variable `x` points to an object that is mutated every time a function is called.

```
x = {'a': 1}

def f(d): d['a'] += 1

f(x)      # x['a'] == 2 now
f(x)      # x['a'] == 3 now
```

In this case, when `f` is run, it changes the contents of `x` as a side-effect and returns None. When using SCTs that run expressions, mutations in either the solution or submission environment can cause very confusing results. For example, calling `np.random.random()` will advance numpy's random number generator. Consider the markdown source for an exercise that illustrates this.

````
`@pre_exercise_code`
```{python}
import numpy as np
np.random.seed(42)               # set random generator seed to 42
```

`@solution`
```{python}
if True: np.random.random()      # 1st random call: .37

np.random.random()               # 2nd random call: .95
```

`@sct`
```{python}
# Should pass but fails, because random generator has advanced
# twice in solution, but only once in submission
Ex().check_if_else().check_body().has_equal_value()
```
````

Assume this student submission:

```
if True: np.random.random()      # 1st random call: .37

# forgot 2nd call to np.random.random()
```

In this situation the random seed is set to 42, but the solution code advances the random generator further than the submission code. As a result the SCT will fail. In order to test random code, the random generator needs to be at the same state between submission and solution environments. Since their generators can be thrown out of sync, the most reliable way to do this is to set the seed using the `pre_code` argument to `has_equal_value`. In the case above, the SCT may be fixed as follows

```
Ex().check_if_else().check_body().has_equal_value(pre_code = "np.random.seed(42)")
```

More generally, it can be helpful to define a pre_code variable to use before expression tests...

```
pre_code = """
np.random.seed(42)
"""

Ex().has_equal_output(pre_code=pre_code)
Ex().check_if_else().check_body().has_equal_value(pre_code = pre_code)
```

8.5 extra_env

As illustrated in the Advanced part checking section of the Checking compound statements article, `set_env()` (as a function) or `extra_env` (as an arugment) can be used to temporarily override the student and solution process to run an expression in multiple situations.

Setting extra environment variables is similar to `pre_code`, in that you can (re)define objects in the student and submission environment before running an expression. The difference is that, rather than passing a string that is executed in each environment, `extra_env` lets you pass objects directly. For example, the three SCT chains below are equivalent...

```
Ex().has_equal_value(pre_code="x = 10")
Ex().set_env(x = 10).has_equal_value()
Ex().has_equal_value(extra_env = {'x': 10})
```

In practice they can often be used interchangably. However, one area where `extra_env` may shine is in mocking up data objects before running tests. For example, if the SCT below didn't use `extra_env`, then it would take a long time to run.

````
`@pre_exercise_code`
```{python}
a_list = list(range(10000000))
```

`@solution`
```{python}
print(a_list[1])
```

`@sct`
```{python}
Ex().set_env(a_list = list(range(10))).has_equal_output()
```
````

The reason extra_env is important here, is that pythonwhat tries to make a deepcopy of lists, so that course developers don't get bit by unexpected mutations. However, the larger the list, the longer it takes to make a deepcopy. If an SCT is running slowly, there's a good chance it uses a very large object that is being copied for every expression test.

8.6 expr_code: change expression

The `expr_code` argument takes a string, and uses it to replace the code that would be run by an expression test. For example, the markdown source for the following exercise simply runs `len(x)` in the solution and student environments.

````
`@solution`
```{python}
# keep x the same length
x = [1,2,3]
```

`@sct`
```{python}
Ex().check_object('x').has_equal_value(expr_code="len(x)")
```
````

Note: Using `expr_code` does not change how expression tests perform highlighting. This means that `Ex().for_loop().has_equal_value(expr_code="x[0]")` would highlight the body of the checked for loop.

8.7 `func`: Override the equality function

After running the expression in question, the `has_equal_x` function will compare the result/output/error of the expression using a built-in equality function. This equality function is geared towards the types of objects you are trying to compare and does its job just fine in 99% of the cases. However, there are cases where you want to customize the equality operation. To do this, you can set `func` to be function that takes two arguments and returns a boolean.

Reiterating over the example from the `expr_code` section above, you can write an equivalent SCT with `func` instead of `expr_code`:

````
`@solution`
```{python}
# keep x the same length
x = [1,2,3]
```

`@sct`
```{python}
Ex().check_object('x').has_equal_value(func = lambda x, y: len(x) == len(y))
```
````

CHAPTER 9

Processes

As explained on the SCT authoring homepage, DataCamp's Python coding backends use uses two separate processes: one process to run the solution code, and one process to run the student's submission. As such, pythonwhat has access to the 'ideal ending scenario' of an exercises, which in turn makes it easier to write SCTs. Instead of having to specify which value an object should be, we can have pythonwhat look into the solution process and compare the object in that process with the object in the student process.

9.1 Problem

Fetching Python objects or the results of running expressions inside a process is not straightforward. To be able to pull data from a process, Python needs to 'dill' and 'undill' files: it converts the Python objects to a byte representation (dilling) that can be passed between processes, and then, inside the process that you want to work with the object, builds up the object from the byte representation again (undilling).

For the majority of Python objects, this conversion to and from a byte representation works fine, but for some more complex objects, it doesn't.

If you're writing an SCT with functions that require work in the solution process, such as `has_equal_value()`, and you try it out in an exercise, it is possible that you'll get the following backend error:

```
... dilling inside process failed - write manual converter
... undilling of bytestream failed - write manual converter
```

The first error tells you that 'dilling' - converting the object to a bytestream representation - failed. The second error tells you that 'undilling' - converting the byte representation back to a Python object - failed. These errors will typically occur if you're dealing with exotic objects, such as objects that interface to files, connections to databases, etc.

9.2 Solution

To be able to handle these errors, pythonwhat allows you to write your own converters for Python objects. Say, for example, that you're writing an exercise to import Excel data into Python, and you're using the `pandas` package:

```
import pandas as pd
xl = pd.ExcelFile('battledeath.xlsx')
```

This is the corresponding SCT:

```
Ex().check_object('xl').has_equal_value()
```

Suppose now that objects such as `xl`, which are of the type `pandas.io.excel.ExcelFile`, can't be properly dilled and undilled. (Because of hardcoded converters inside pythonwhat, they can, see below). To make sure that you can still use `check_object('xl')` to test the equality of the `xl` object between student and solution process, you can manually define a converter with the `set_converter()` function. You can extend the SCT as follows:

```
def my_converter(x):
    return(x.sheet_names)
set_converter(key = "pandas.io.excel.ExcelFile", fundef = my_converter)
Ex().check_object('xl').has_equal_value()
```

With a lambda function, it's even easier:

```
set_converter(key = "pandas.io.excel.ExcelFile", fundef = lambda x: x.sheet_names)
Ex().check_object('xl').has_equal_value()
```

The first arguemnt of `set_converter()`, the `key` takes the type of the object you want to add a manual converter for as a string. The second argument, `fundef`, takes a function definition, taking one argument and returning a single object. This function definition converts the exotic object into something more standard. In this case, the function converts the object of type `pandas.io.excel.ExcelFile` into a simple list of strings. A list of strings is something that can easily be converted into a bytestream and back into a Python object again, hence solving the problem.

If you want to reuse the same manual converter over different exercises, you'll have to use `set_converter()` in every SCT.

9.3 Hardcoded converters

Next to primitive classes like `str`, `int`, `list`, `dict`, ... and objects with a semantically correct implemenation of ==, there are also a bunch of often-used complex objects that don't have a proper implementation of ==. For example, the result of calling `.keys()` and `.items()` on dictionaries can't be dilled and undilled without extra work. To handle these common yet problematic situations, pythonwhat features a list of hardcoded converters, so that you don't have to manually specify them each time. This list is available in the source code. Feel free to do a pull request if you want to add more converts to this list, which will reduce the amount of code duplication you have to do if you want to reuse the same converter in different exercises.

9.4 Customize equality

The `set_converter()` function opens up possibilities for objects that can actually be dilled and undilled perfectly fine. Say you want to test a `numpy` array, but you only want to check only if the dimensions of the array the student codes up match those in the solution process. You can easily write a manual converter that overrides the typical dilling and undilling of Numpy arrays, implementing your custom equality behavior:

```
# solution
import numpy as np
my_array = np.array([[1,2], [3,4], [5,6]])
```

(continues on next page)

(continued from previous page)

```
# sct
set_converter(key = "numpy.ndarray", fundef = lambda x: x.shape)
Ex().check_object('my_array').has_equal_value()

# both these submissions will pass
my_array = np.array([[1,2], [3,4], [5,6]])
my_array = np.array([[0,0], [0,0], [5,6]])
```

CHAPTER 10

SingleProcessExercise

10.1 Introduction

Typical interactive exercises on DataCamp will be of the type `NormalExercise` or something similar.

For these normal exercises, the pythonbackend (the Python package responsible for running Python code that the student submitted) will execute:

- the solution code in a solution process (once, at exercise initialization),
- the student's submission in a student process (every time the student hits submit, after which the process is restarted from scratch)
- the student's experimentation commands in the console in a console process (every time the user executes a command, without restarting afterwards)

These completely separate processes make sure that:

- the different commands do not interfere with one another; if you import a package in one process, the package will not become available in the other process.
- pythonwhat has access to a 'target solution process' to easily do comparisons; to compare an object `x`, you simply have to use `Ex().check_object('x').has_equal_value()` and pythonwhat will figure out the value `x` should have from the solution process.

To learn more about how the backend works, you can visit this wiki article.

10.2 Why does this exercise type exist?

There are Python courses that make extensive use of programs running outside of Python. Sometimes, these programs cannot handle it well when different Python process are trying to interface with it. An example of this is PySpark, where on container startup, a Spark cluster is started up, that you can then interface with. Things go horribly wrong if you try to access this PySpark cluster from different Python processes.

To solve for this, a new exercise type was built, that does not create three separate Python processes (solution, student, console). Instead, only one process is created:

- the solution code is not executed in this process.
- the student's sumission is executed in this process, but the process is not restarted afterwards
- the student's experimentation commands in the console are executed in the same process.

From a user perspective, this shouldn't pose too much difficulties, with the exception that the code execution is now stateful.

10.3 So, what's the problem then?

As mentioned earlier, pythonwhat depends heavily on the existence of two separate process: a 'target' solution process, and a student process. Functions such as `has_equal_value()` compare values and the results of expression in these processes. In the `SingleProcessExercise`, the student process and the solution process are identical, it's one and the same process, so these comparisons don't make any sense.

Therefore, the 'process-based checks' in pythonwhat have to be used with care when writing SCTs for a `SingleProcessExercise`. More specifically:

- `check_object()` should work okay, as there is some magic happening behind the scenes.
- `has_equal_value()` and `has_equal_output()` should be used with the `override` argument. When this argument is specified, the expression that is 'zoomed in on' in the solution code will not be executed in the solution proces. Instead, it will just take the value you pass to `override` to compare the result/output of the expression that is zoomed in on in the student code to.

10.4 Example

As an example, suppose we want to check whether a student correctly created a list `x`:

```
# solution
x = [1, 2, 3, 4, 5]
```

If this solution were part of a traditional `NormalExercise`, your SCT would be simple:

```
# SCT
Ex().check_object('x').has_equal_value()
```

However, if this solution were part of a `SingleProcessExercise`, the above SCT would not work. Instead, you'll want to do the following:

```
# SCT
Ex().check_object('x').has_equal_value(override = [1, 2, 3, 4, 5])
```

Here, we use `override` to tell pythonwhat not to go look for the value of `x` in the solution process. Instead, it uses the manually specified value in `override` to compare to.

You can use `override` in combination with other arguments in `has_equal_x()`, such as `expr_code`. Suppose you're only interested in the element at index 2 of the list `x`:

```
# SCT
Ex().check_object('x').has_equal_value(expr_code = 'x[2]', override = 3)
```

Tricky stuff, but it works!

CHAPTER 11

Electives

11.1 Success message

When all tests in an SCT pass, pythonwhat will automatically generate a congratulatory message to present to the student. If you want to override this 'success message', you can use the `success_msg()` function.

```
Ex().check_object("x").has_equal_value()
success_msg("You are a hero when it comes to variable assignment!")
```

This article on the authoring docs describes how to write good success messages.

11.2 Multiple choice exercises

Multiple choice exercises are straightforward to test. Use `has_chosen()` to provide tailored feedback for both the incorrect options, as the correct option. Below is the markdown source for a multiple choice exercise example, with an SCT that uses `has_chosen`:

````
## The author of Python

```yaml
type: MultipleChoiceExercise
```

Who is the author of the Python programming language?

`@instructions`

- Roy Co
- Ronald McDonald
- Guido van Rossum

`@sct`
````

(continues on next page)

(continued from previous page)

```
```{python}
Ex().has_chosen(correct = 3,
                msgs = ["That's someone who makes soups.",
                        "That's a clown who likes burgers.",
                        "Correct! Head over to the next exercise!"])
```
```

- `correct` specifies the number of the correct answer in this list (1-base indexed).
- `msgs` argument should be a list of strings with a length equal to the number of options. We encourage you to provide feedback messages that are informative and tailored to the (incorrect) option that people selected.

Notice that there's no need for `success_msg()` in multiple choice exercises, as you have to specify the success message inside `has_chosen()`, along with the feedback for incorrect options.

11.3 Capabilities of multi

`multi()` is always used to 'branch' different chains of SCT functions, so that the same state is passed to two sub-chains. There are different ways

11.3.1 Comma separated arguments

Most commonly, `multi()` is used to convert this code

```
Ex().check_if_exp().check_body().has_equal_value()
Ex().check_if_exp().check_test().has_equal_value()
```

into this equivalent (and more performant) SCT:

```
Ex().check_if_exp().multi(
        check_body().has_equal_value(),
        check_test().has_equal_value()
        )
```

11.3.2 List or generator of subtests

Rather than one or more subtest args, multi can take a single list or generator of subtests. For example, the code below checks that the body of a list comprehension has equal value for 10 possible values of the iterator variable, `i`.

```
Ex().check_list_comp()
    .check_body()
    .multi(set_context(i=x).has_equal_value() for x in range(10))
```

11.3.3 Chaining off multi

Multi returns the same state, or focus, it was given, so whatever comes after multi will run the same as if multi wasn't used. For example, the code below tests a list comprehension's body, followed by its iterator.

```
Ex().check_list_comp() \
    .multi(check_body().has_equal_value()) \
    .check_iter().has_equal_value()
```

11.4 has_context

Tests whether context variables defined by the student match the solution, for a selected block of code. A context variable is one that is defined in a looping or block statement. For example, `ii` in the code below.

```
[ii + 1 for ii in range(3)]
```

By default, the test fails if the submission code does not have the same number of context variables. This is illustrated below.

```
# solution
# ii and ltr are context variables
for ii, ltr in enumerate(['a']): pass

# sct
Ex().check_for_loop().check_body().has_context()

# passing submission
# still 2 variables, just different names
for jj, Ltr in enumerate(['a']): pass

# failing submission
# only 1 variable
for ii in enumerate(['a']): pass
```

Note: If you use `has_context(exact_names = True)`, then the submission must use the same names for the context variables, which would cause the passing submission above to fail.

11.5 set_context

Sets the value of a temporary variable, such as `ii` in the list comprehension below.

```
[ii + 1 for ii in range(3)]
```

Variable names may be specified using positional or keyword arguments.

11.5.1 Example

```
# solution
ltrs = ['a', 'b']
for ii, ltr in enumerate(ltrs):
    print(ii)

# sct
```

(continues on next page)

(continued from previous page)

```
Ex().check_for_loop().check_body() \
    .set_context(ii=0, ltr='a').has_equal_output() \
    .set_context(ii=1, ltr='b').has_equal_output()
```

Note that if a student replaced `ii` with `jj` in their submission, `set_context` would still work. It uses the solution code as a reference. While we specified the target variables `ii` and `ltr` by name in the SCT above, they may also be given by position..

```
Ex().check_for_loop().check_body().set_context(0, 'a').has_equal_output()
```

11.6 with_context

with_context (*state*, **args*, *child=None*)

Runs subtests after setting the context for a `with` statement.

This function takes arguments in the same form as `multi`.

11.6.1 Context Managers Explained

With statements are special in python in that they enter objects called a context manager at the beginning of the block, and exit them at the end. For example, the object returned by `open('fname.txt')` below is a context manager.

```
with open('fname.txt') as f:
    print(f.read())
```

This code runs by

1. assigning `f` to the context manager returned by `open('fname.txt')`
2. calling `f.__enter__()`
3. running the block
4. calling `f.__exit__()`

`with_context` was designed to emulate this sequence of events, by setting up context values as in step (1), and replacing step (3) with any sub-tests given as arguments.

CHAPTER 12

Test to Check

If you are looking at the SCTs of old DataCamp courses, you'll notice they use `test_x()` functions instead of `check_x()` functions, and there is no usage of `Ex()`. The `test_x()` way of doing things has now been phased out in favor of the more transparent and composable `check_x()` functions that start with `Ex()` and are chained together with the `.` operator.

12.1 Common cases

Whenever you come across an SCT that uses `test_x()` functions, you'll make everybody's life easier by converting it to a `check_x()`-based SCT. Below are the most common cases you will encounter, together with instructions on how to translate from one to the other.

Something you came across that you didn't find in this list? Just create an issue on GitHub. Content Engineering will explain how to translate the SCT and update this article.

12.1.1 `test_student_typed`

```
# Solution
y = 1 + 2 + 3

# old SCT
test_student_typed(r'1\s*\+2\s*\+3')

# new SCT
Ex().has_code(r'1\s*\+2\s*\+3')
```

12.1.2 `test_object`

```
# Solution
x = 4

# old SCT (checks equality by default)
test_object('x')

# new SCT
Ex().check_object('x').has_equal_value()
```

```
# Solution
x = 4

# old SCT
test_object('x', do_eval=False)

# new SCT
Ex().check_object('x')
```

12.1.3 test_function

```
# Solution
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
np.mean(arr)

# old SCT (checks all args specified in solution)
test_function('numpy.array')

# new SCT
Ex().check_function('numpy.array').check_args('a').has_equal_value()
```

```
# Solution
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
np.mean(arr)
np.mean(arr + arr)

# old SCT (1-based indexed)
test_function('numpy.array', index=1)
test_function('numpy.array', index=2)

# new SCT (0-based indexed)
Ex().check_function('numpy.array', index=0).check_args('a').has_equal_value()
Ex().check_function('numpy.array', index=1).check_args('a').has_equal_value()
```

12.1.4 test_function_v2

```
# Solution
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
np.mean(arr)
```

(continues on next page)

(continued from previous page)

```
# old SCT (explicitly specify args)
test_function_v2('numpy.array', params=['a'], index=1)

# new SCT
Ex().check_function('numpy.array', index=0).check_args('a').has_equal_value()
```

12.1.5 test_correct

```
# Solution
import numpy as np
arr = np.array([1, 2, 3, 4, 5])

# old SCT (use lambdas to defer execution)
test_correct(lambda: test_object('arr'),
             labmda: test_function('numpy.array'))

# new SCT (no need for lambdas)
Ex().check_correct(check_object('arr').has_equal_value(),
                   check_function('numpy.array').check_args('a').has_equal_value())
```

12.2 Enforcing check functions

For newer courses an updated version of the base Docker image is used that sets the `PYTHONWHAT_V2_ONLY` environment variable. When this variable is set, pythonwhat will no longer allow sct authors to use the old `test_x()` functions. If you are updating the Docker image for courses that have old skool SCTs, this would mean you have to rewrite all the SCTs to their check equivalents to make the build pass. If you want to work around this, thus being able to use test functions even with the latest base image, you can include the following line of code in your `requirements.sh` file:

```
echo "import os; os.environ['PYTHONWHAT_V2_ONLY'] = '0'" > /home/repl/.startup.py
```

CHAPTER 13

Tests

Note: The examples are numbered and linkable, but numbers (and links) can change between builds of the documentation.

13.1 test_check_function_def.py

13.1.1 Example 1

No PEC

Solution code

```
def shout(word): print(word + '!!')
```

Student code

```
def shout(word): print(word + '!!')
```

No output

SCT

```
Ex().check_function_def('shout').check_body().set_context('test').has_equal_output()
```

Result

```
Great work!
```

No error

13.1.2 Example 2

No PEC

Solution code

```
def shout(word): print(word + '!!!')
```

Student code

```
def shout(word): print(word + '!!!')
```

No output

SCT

```
test_function_definition('shout', body = lambda: test_expression_output(context_vals
↪= ['help']))
```

Result

```
Great work!
```

No error

13.1.3 Example 3

No PEC

Solution code

```
def shout(word): print(word + '!!!')
```

Student code

```
def shout(word): print(word + '!!!')
```

No output

SCT

```
test_function_definition('shout', body = test_expression_output(context_vals = ['help
↪']))
```

Result

```
Great work!
```

No error

13.1.4 Example 4

No PEC

Solution code

```
def my_fun(*x, **y): pass
```

Student code

```
def my_fun(*x, **y): pass
```

No output

SCT

```
Ex().check_function_def('my_fun').multi(
    check_args('*args').has_equal_part('name', msg='x'),
    check_args('**kwargs').has_equal_part('name', msg='x')
)
```

Result

```
Great work!
```

No error

13.1.5 Example 5

No PEC

Solution code

```
def my_fun(*x, **y): pass
```

Student code

```
def my_fun(*x, **y): pass
```

No output

SCT

```
Ex().test_function_definition('my_fun')
```

Result

```
Great work!
```

No error

13.1.6 Example 6

No PEC

Solution code

```
def f(a, b): pass
```

Student code

```
def f(): pass
```

No output

SCT

```
Ex().check_function_def('f').has_equal_part_len('args', unequal_msg='wrong')
```

Result

```
Check the definition of <code>f()</code>. wrong
```

No error

13.1.7 Example 7

No PEC

Solution code

```
def f(a, b): pass
```

Student code

```
def f(): pass
```

No output

SCT

```
Ex().test_function_definition('f')
```

Result

```
Check the definition of <code>f()</code>. You should define  with 2 arguments,␣
↪instead got 0.
```

No error

13.1.8 Example 8

No PEC

Solution code

```
def f(a, b): pass
```

Student code

```
def f(a): pass
```

No output

SCT

```
Ex().check_function_def('f').has_equal_part_len('args', unequal_msg='wrong')
```

Result

```
Check the definition of <code>f()</code>. wrong
```

No error

13.1.9 Example 9

No PEC

Solution code

```
def f(a, b): pass
```

Student code

```
def f(a): pass
```

No output

SCT

```
Ex().test_function_definition('f')
```

Result

```
Check the definition of <code>f()</code>. You should define  with 2 arguments,␣
↪instead got 1.
```

No error

13.1.10 Example 10

No PEC

Solution code

```
def f(a, b): pass
```

Student code

```
def f(a, b): pass
```

No output

SCT

```
Ex().check_function_def('f').has_equal_part_len('args', unequal_msg='wrong')
```

Result

```
Great work!
```

No error

13.1.11 Example 11

No PEC

Solution code

```
def f(a, b): pass
```

Student code

```
def f(a, b): pass
```

No output

SCT

```
Ex().test_function_definition('f')
```

Result

```
Great work!
```

No error

13.1.12 Example 12

No PEC

Solution code

```
def my_fun(a):
  print(a + 2)
  return a + 2
```

Student code

```
def my_fun(a):
  print(a + 2)
  return a + 2
```

No output

SCT

```
Ex().test_function_definition('my_fun', results=[[1]])
```

Result

```
Great work!
```

No error

13.1.13 Example 13

No PEC

Solution code

```
def my_fun(a):
  print(a + 2)
  return a + 2
```

Student code

```
def my_fun(a):
  print(a + 2)
  return a + 2
```

No output

SCT

```
Ex().test_function_definition('my_fun', results=[(1,)])
```

Result

```
Great work!
```

No error

13.1.14 Example 14

No PEC

Solution code

```
def my_fun(a):
  print(a + 2)
  return a + 2
```

Student code

```
def my_fun(a):
  print(a + 2)
  return a + 2
```

No output

SCT

```
Ex().test_function_definition('my_fun', outputs=[[1]])
```

Result

```
Great work!
```

No error

13.1.15 Example 15

No PEC

Solution code

```
def my_fun(a):
  print(a + 2)
  return a + 2
```

Student code

```
def my_fun(a):
  print(a + 2)
  return a + 2
```

No output

SCT

```
Ex().test_function_definition('my_fun', outputs=[(1,)])
```

Result

```
Great work!
```

No error

13.1.16 Example 16

No PEC

Solution code

```
def my_fun(a):
  print(a + 2)
  return a + 2
```

Student code

```
def my_fun(a):
  print(a + 2)
  return a + 2
```

No output

SCT

```
Ex().test_function_definition('my_fun', errors=[['1']])
```

Result

```
Great work!
```

No error

13.1.17 Example 17

No PEC

Solution code

```
def my_fun(a):
  print(a + 2)
  return a + 2
```

Student code

```
def my_fun(a):
  print(a + 2)
  return a + 2
```

No output

SCT

```
Ex().test_function_definition('my_fun', errors=[('1',)])
```

Result

```
Great work!
```

No error

13.1.18 Example 18

No PEC

Solution code

```
def my_fun(a):
  print(a + 2)
  return a + 2
```

Student code

```
def my_fun(a):
  print(a + 2)
  return a + 2
```

No output

SCT

```
Ex().test_function_definition('my_fun', errors=['1'])
```

Result

```
Great work!
```

No error

13.2 test_check_if_else.py

13.2.1 Example 1

PEC

```
offset = 8
```

Solution code

```
if offset > 8: x = 5
else: x = round(2.123)
```

No student code

No output

SCT

```
def condition_test():
    test_expression_result({"offset": 7})
    test_expression_result({"offset": 8})
    test_expression_result({"offset": 9})
test_if_else(index=1,
             test = condition_test,
             body = lambda: test_student_typed(r'x\s*=\s*5'),
             orelse = lambda: test_function('round'))
```

Result

```
The system wants to check the first if expression but hasn't found it.
```

No error

13.2.2 Example 2

PEC

```
offset = 8
```

Solution code

```
if offset > 8: x = 5
else: x = round(2.123)
```

No student code

No output

SCT

```
condition_test = [
    test_expression_result({"offset": 7}),
    test_expression_result({"offset": 8}),
    test_expression_result({"offset": 9})
]

test_if_else(index=1,
             test = condition_test,
             body = test_student_typed(r'x\s*=\s*5'),
             orelse = test_function('round'))
```

Result

```
The system wants to check the first if expression but hasn't found it.
```

No error

13.2.3 Example 3

PEC

```
offset = 8
```

Solution code

```
if offset > 8: x = 5
else: x = round(2.123)
```

No student code

No output

SCT

```
Ex().check_if_else().multi(
    check_test().multi([ set_env(offset = i).has_equal_value() for i in range(7,10)␣
↪]),
    check_body().has_code(r'x\s*=\s*5'),
    check_orelse().check_function('round').check_args(0).has_equal_value()
)
```

Result

```
The system wants to check the first if statement but hasn't found it.
```

No error

13.2.4 Example 4

PEC

```
offset = 8
```

Solution code

```
if offset > 8: x = 5
else: x = round(2.123)
```

Student code

```
if offset > 10: x = 5
else: x = round(2.123)
```

No output

SCT

```
def condition_test():
    test_expression_result({"offset": 7})
    test_expression_result({"offset": 8})
    test_expression_result({"offset": 9})
test_if_else(index=1,
             test = condition_test,
             body = lambda: test_student_typed(r'x\s*=\s*5'),
             orelse = lambda: test_function('round'))
```

Result

```
Check the first if expression. Did you correctly specify the condition? Expected
↪<code>True</code>, but got <code>False</code>.
```

No error

13.2.5 Example 5

PEC

```
offset = 8
```

Solution code

```
if offset > 8: x = 5
else: x = round(2.123)
```

Student code

```
if offset > 10: x = 5
else: x = round(2.123)
```

No output

SCT

```
condition_test = [
    test_expression_result({"offset": 7}),
    test_expression_result({"offset": 8}),
    test_expression_result({"offset": 9})
]

test_if_else(index=1,
             test = condition_test,
             body = test_student_typed(r'x\s*=\s*5'),
             orelse = test_function('round'))
```

Result

```
Check the first if expression. Did you correctly specify the condition? Expected
↪<code>True</code>, but got <code>False</code>.
```

No error

13.2.6 Example 6

PEC

```
offset = 8
```

Solution code

```
if offset > 8: x = 5
else: x = round(2.123)
```

Student code

```
if offset > 10: x = 5
else: x = round(2.123)
```

No output

SCT

```
Ex().check_if_else().multi(
    check_test().multi([ set_env(offset = i).has_equal_value() for i in range(7,10)␣
↪]),
    check_body().has_code(r'x\s*=\s*5'),
    check_orelse().check_function('round').check_args(0).has_equal_value()
)
```

Result

```
Check the first if statement. Did you correctly specify the condition? Expected <code>
↪True</code>, but got <code>False</code>.
```

No error

13.2.7 Example 7

PEC

```
offset = 8
```

Solution code

```
if offset > 8: x = 5
else: x = round(2.123)
```

Student code

```
if offset > 8: x = 7
else: x = round(2.123)
```

No output

SCT

```
def condition_test():
    test_expression_result({"offset": 7})
    test_expression_result({"offset": 8})
```

(continues on next page)

(continued from previous page)

```
    test_expression_result({"offset": 9})
test_if_else(index=1,
             test = condition_test,
             body = lambda: test_student_typed(r'x\s*=\s*5'),
             orelse = lambda: test_function('round'))
```

Result

```
Check the first if expression. Did you correctly specify the body? Could not find the␣
↪correct pattern in your code.
```

No error

13.2.8 Example 8

PEC

```
offset = 8
```

Solution code

```
if offset > 8: x = 5
else: x = round(2.123)
```

Student code

```
if offset > 8: x = 7
else: x = round(2.123)
```

No output

SCT

```
condition_test = [
    test_expression_result({"offset": 7}),
    test_expression_result({"offset": 8}),
    test_expression_result({"offset": 9})
]

test_if_else(index=1,
             test = condition_test,
             body = test_student_typed(r'x\s*=\s*5'),
             orelse = test_function('round'))
```

Result

```
Check the first if expression. Did you correctly specify the body? Could not find the␣
↪correct pattern in your code.
```

No error

13.2.9 Example 9

PEC

```
offset = 8
```

Solution code

```
if offset > 8: x = 5
else: x = round(2.123)
```

Student code

```
if offset > 8: x = 7
else: x = round(2.123)
```

No output

SCT

```
Ex().check_if_else().multi(
    check_test().multi([ set_env(offset = i).has_equal_value() for i in range(7,10)␣
↪]),
    check_body().has_code(r'x\s*=\s*5'),
    check_orelse().check_function('round').check_args(0).has_equal_value()
)
```

Result

```
Check the first if statement. Did you correctly specify the body? Could not find the␣
↪correct pattern in your code.
```

No error

13.2.10 Example 10

PEC

```
offset = 8
```

Solution code

```
if offset > 8: x = 5
else: x = round(2.123)
```

Student code

```
if offset > 8: x,y = 7,12
else: x = round(2.123)
```

No output

SCT

```
def condition_test():
    test_expression_result({"offset": 7})
    test_expression_result({"offset": 8})
    test_expression_result({"offset": 9})
test_if_else(index=1,
             test = condition_test,
```

(continues on next page)

(continued from previous page)

```
            body = lambda: test_student_typed(r'x\s*=\s*5'),
            orelse = lambda: test_function('round'))
```

Result

```
Check the first if expression. Did you correctly specify the body? Could not find the␣
↪correct pattern in your code.
```

No error

13.2.11 Example 11

PEC

```
offset = 8
```

Solution code

```
if offset > 8: x = 5
else: x = round(2.123)
```

Student code

```
if offset > 8: x,y = 7,12
else: x = round(2.123)
```

No output

SCT

```
condition_test = [
    test_expression_result({"offset": 7}),
    test_expression_result({"offset": 8}),
    test_expression_result({"offset": 9})
]

test_if_else(index=1,
             test = condition_test,
             body = test_student_typed(r'x\s*=\s*5'),
             orelse = test_function('round'))
```

Result

```
Check the first if expression. Did you correctly specify the body? Could not find the␣
↪correct pattern in your code.
```

No error

13.2.12 Example 12

PEC

```
offset = 8
```

Solution code

```
if offset > 8: x = 5
else: x = round(2.123)
```

Student code

```
if offset > 8: x,y = 7,12
else: x = round(2.123)
```

No output

SCT

```
Ex().check_if_else().multi(
    check_test().multi([ set_env(offset = i).has_equal_value() for i in range(7,10)␣
↪]),
    check_body().has_code(r'x\s*=\s*5'),
    check_orelse().check_function('round').check_args(0).has_equal_value()
)
```

Result

```
Check the first if statement. Did you correctly specify the body? Could not find the␣
↪correct pattern in your code.
```

No error

13.2.13 Example 13

PEC

```
offset = 8
```

Solution code

```
if offset > 8: x = 5
else: x = round(2.123)
```

Student code

```
if offset > 8: x = 5
else: x = 8
```

No output

SCT

```
def condition_test():
    test_expression_result({"offset": 7})
    test_expression_result({"offset": 8})
    test_expression_result({"offset": 9})
test_if_else(index=1,
             test = condition_test,
             body = lambda: test_student_typed(r'x\s*=\s*5'),
             orelse = lambda: test_function('round'))
```

Result

```
Check the first if expression. Did you correctly specify the else part? Did you call
↪<code>round()</code>?
```

No error

13.2.14 Example 14

PEC

```
offset = 8
```

Solution code

```
if offset > 8: x = 5
else: x = round(2.123)
```

Student code

```
if offset > 8: x = 5
else: x = 8
```

No output

SCT

```
condition_test = [
    test_expression_result({"offset": 7}),
    test_expression_result({"offset": 8}),
    test_expression_result({"offset": 9})
]

test_if_else(index=1,
             test = condition_test,
             body = test_student_typed(r'x\s*=\s*5'),
             orelse = test_function('round'))
```

Result

```
Check the first if expression. Did you correctly specify the else part? Did you call
↪<code>round()</code>?
```

No error

13.2.15 Example 15

PEC

```
offset = 8
```

Solution code

```
if offset > 8: x = 5
else: x = round(2.123)
```

Student code

```
if offset > 8: x = 5
else: x = 8
```

No output

SCT

```
Ex().check_if_else().multi(
    check_test().multi([ set_env(offset = i).has_equal_value() for i in range(7,10)␣
↪]),
    check_body().has_code(r'x\s*=\s*5'),
    check_orelse().check_function('round').check_args(0).has_equal_value()
)
```

Result

```
Check the first if statement. Did you correctly specify the else part? Did you call
↪<code>round()</code>?
```

No error

13.2.16 Example 16

PEC

```
offset = 8
```

Solution code

```
if offset > 8: x = 5
else: x = round(2.123)
```

Student code

```
if offset > 8: x = 5
else: x = round(2.2121314)
```

No output

SCT

```
def condition_test():
    test_expression_result({"offset": 7})
    test_expression_result({"offset": 8})
    test_expression_result({"offset": 9})
test_if_else(index=1,
             test = condition_test,
             body = lambda: test_student_typed(r'x\s*=\s*5'),
             orelse = lambda: test_function('round'))
```

Result

```
Check your call of <code>round()</code>. Did you correctly specify the first argument?
↪ Expected <code>2.123</code>, but got <code>2.2121314</code>.
```

No error

13.2.17 Example 17

PEC

```
offset = 8
```

Solution code

```
if offset > 8: x = 5
else: x = round(2.123)
```

Student code

```
if offset > 8: x = 5
else: x = round(2.2121314)
```

No output

SCT

```
condition_test = [
    test_expression_result({"offset": 7}),
    test_expression_result({"offset": 8}),
    test_expression_result({"offset": 9})
]

test_if_else(index=1,
             test = condition_test,
             body = test_student_typed(r'x\s*=\s*5'),
             orelse = test_function('round'))
```

Result

```
Check your call of <code>round()</code>. Did you correctly specify the first argument?
↪ Expected <code>2.123</code>, but got <code>2.2121314</code>.
```

No error

13.2.18 Example 18

PEC

```
offset = 8
```

Solution code

```
if offset > 8: x = 5
else: x = round(2.123)
```

Student code

```
if offset > 8: x = 5
else: x = round(2.2121314)
```

No output

SCT

```
Ex().check_if_else().multi(
    check_test().multi([ set_env(offset = i).has_equal_value() for i in range(7,10)␣
↪]),
    check_body().has_code(r'x\s*=\s*5'),
    check_orelse().check_function('round').check_args(0).has_equal_value()
)
```

Result

```
Check your call of <code>round()</code>. Did you correctly specify the first argument?
↪ Expected <code>2.123</code>, but got <code>2.2121314</code>.
```

No error

13.2.19 Example 19

PEC

```
offset = 8
```

Solution code

```
if offset > 8: x = 5
else: x = round(2.123)
```

Student code

```
if offset > 8: x = 5
else: x = round(2.123)
```

No output

SCT

```
def condition_test():
    test_expression_result({"offset": 7})
    test_expression_result({"offset": 8})
    test_expression_result({"offset": 9})
test_if_else(index=1,
             test = condition_test,
             body = lambda: test_student_typed(r'x\s*=\s*5'),
             orelse = lambda: test_function('round'))
```

Result

```
Great work!
```

No error

13.2.20 Example 20

PEC

```
offset = 8
```

Solution code

```
if offset > 8: x = 5
else: x = round(2.123)
```

Student code

```
if offset > 8: x = 5
else: x = round(2.123)
```

No output

SCT

```
condition_test = [
    test_expression_result({"offset": 7}),
    test_expression_result({"offset": 8}),
    test_expression_result({"offset": 9})
]

test_if_else(index=1,
             test = condition_test,
             body = test_student_typed(r'x\s*=\s*5'),
             orelse = test_function('round'))
```

Result

```
Great work!
```

No error

13.2.21 Example 21

PEC

```
offset = 8
```

Solution code

```
if offset > 8: x = 5
else: x = round(2.123)
```

Student code

```
if offset > 8: x = 5
else: x = round(2.123)
```

No output

SCT

```
Ex().check_if_else().multi(
    check_test().multi([ set_env(offset = i).has_equal_value() for i in range(7,10)␣
↪]),
    check_body().has_code(r'x\s*=\s*5'),
    check_orelse().check_function('round').check_args(0).has_equal_value()
)
```

Result

```
Great work!
```

No error

13.2.22 Example 22

PEC

```
offset = 8
```

Solution code

```
if offset > 8: x = 5
elif offset > 5: x = 7
else: x = round(9)
```

No student code

No output

SCT

```
def test_test():
    test_expression_result({"offset": 7})
    test_expression_result({"offset": 8})
    test_expression_result({"offset": 9})

def body_test():
    test_student_typed('5')

def orelse_test():
    def test_test2():
        test_expression_result({"offset": 4})
        test_expression_result({"offset": 5})
        test_expression_result({"offset": 6})
    def body_test2():
        test_student_typed('7')
    def orelse_test2():
        test_function('round')
    test_if_else(index = 1,
                 test = test_test2,
                 body = body_test2,
                 orelse = orelse_test2)

test_if_else(index=1,
             test=test_test,
             body=body_test,
             orelse=orelse_test)
```

Result

```
The system wants to check the first if expression but hasn't found it.
```

No error

13.2.23 Example 23

PEC

```
offset = 8
```

Solution code

```
if offset > 8: x = 5
elif offset > 5: x = 7
else: x = round(9)
```

No student code

No output

SCT

```
test_if_else(index=1,
    test=[
        test_expression_result({"offset": 7}),
        test_expression_result({"offset": 8}),
        test_expression_result({"offset": 9})
    ],
    body=test_student_typed('5'),
    orelse=test_if_else(index = 1,
        test=[
            test_expression_result({"offset": 4}),
            test_expression_result({"offset": 5}),
            test_expression_result({"offset": 6})
        ],
        body = test_student_typed('7'),
        orelse = test_function('round')
    )
)
```

Result

```
The system wants to check the first if expression but hasn't found it.
```

No error

13.2.24 Example 24

PEC

```
offset = 8
```

Solution code

```
if offset > 8: x = 5
elif offset > 5: x = 7
else: x = round(9)
```

No student code

No output

SCT

```
Ex().check_if_else().multi(
    check_test().multi([ set_env(offset = i).has_equal_value() for i in range(7, 10)␣
↪]),
    check_body().has_code(r'x\s*=\s*5'),
    check_orelse().check_if_else().multi(
        check_test().multi([ set_env(offset = i).has_equal_value() for i in range(4,␣
↪7) ]),
        check_body().has_code('7'),
        check_orelse().check_function('round').check_args(0).has_equal_value()
    )
)
```

Result

```
The system wants to check the first if statement but hasn't found it.
```

No error

13.2.25 Example 25

PEC

```
offset = 8
```

Solution code

```
if offset > 8: x = 5
elif offset > 5: x = 7
else: x = round(9)
```

Student code

```
if offset > 9: x = 5
elif offset > 5: x = 7
else: x = round(9)
```

No output

SCT

```
def test_test():
    test_expression_result({"offset": 7})
    test_expression_result({"offset": 8})
    test_expression_result({"offset": 9})

def body_test():
    test_student_typed('5')

def orelse_test():
    def test_test2():
        test_expression_result({"offset": 4})
        test_expression_result({"offset": 5})
        test_expression_result({"offset": 6})
    def body_test2():
        test_student_typed('7')
    def orelse_test2():
```

(continues on next page)

(continued from previous page)

```
        test_function('round')
    test_if_else(index = 1,
                 test = test_test2,
                 body = body_test2,
                 orelse = orelse_test2)

test_if_else(index=1,
             test=test_test,
             body=body_test,
             orelse=orelse_test)
```

Result

```
Check the first if expression. Did you correctly specify the condition? Expected
↪<code>True</code>, but got <code>False</code>.
```

No error

13.2.26 Example 26

PEC

```
offset = 8
```

Solution code

```
if offset > 8: x = 5
elif offset > 5: x = 7
else: x = round(9)
```

Student code

```
if offset > 9: x = 5
elif offset > 5: x = 7
else: x = round(9)
```

No output

SCT

```
test_if_else(index=1,
    test=[
        test_expression_result({"offset": 7}),
        test_expression_result({"offset": 8}),
        test_expression_result({"offset": 9})
    ],
    body=test_student_typed('5'),
    orelse=test_if_else(index = 1,
        test=[
            test_expression_result({"offset": 4}),
            test_expression_result({"offset": 5}),
            test_expression_result({"offset": 6})
        ],
        body = test_student_typed('7'),
        orelse = test_function('round')
```

(continues on next page)

(continued from previous page)

```
    )
)
```

Result

```
Check the first if expression. Did you correctly specify the condition? Expected
↪<code>True</code>, but got <code>False</code>.
```

No error

13.2.27 Example 27

PEC

```
offset = 8
```

Solution code

```
if offset > 8: x = 5
elif offset > 5: x = 7
else: x = round(9)
```

Student code

```
if offset > 9: x = 5
elif offset > 5: x = 7
else: x = round(9)
```

No output

SCT

```
Ex().check_if_else().multi(
    check_test().multi([ set_env(offset = i).has_equal_value() for i in range(7, 10)␣
↪]),
    check_body().has_code(r'x\s*=\s*5'),
    check_orelse().check_if_else().multi(
        check_test().multi([ set_env(offset = i).has_equal_value() for i in range(4,␣
↪7) ]),
        check_body().has_code('7'),
        check_orelse().check_function('round').check_args(0).has_equal_value()
    )
)
```

Result

```
Check the first if statement. Did you correctly specify the condition? Expected <code>
↪True</code>, but got <code>False</code>.
```

No error

13.2.28 Example 28

PEC

```
offset = 8
```

Solution code

```
if offset > 8: x = 5
elif offset > 5: x = 7
else: x = round(9)
```

Student code

```
if offset > 8: x = 6
elif offset > 5: x = 7
else: x = round(9)
```

No output

SCT

```
def test_test():
    test_expression_result({"offset": 7})
    test_expression_result({"offset": 8})
    test_expression_result({"offset": 9})

def body_test():
    test_student_typed('5')

def orelse_test():
    def test_test2():
        test_expression_result({"offset": 4})
        test_expression_result({"offset": 5})
        test_expression_result({"offset": 6})
    def body_test2():
        test_student_typed('7')
    def orelse_test2():
        test_function('round')
    test_if_else(index = 1,
                 test = test_test2,
                 body = body_test2,
                 orelse = orelse_test2)

test_if_else(index=1,
             test=test_test,
             body=body_test,
             orelse=orelse_test)
```

Result

```
Check the first if expression. Did you correctly specify the body? Could not find the␣
↪correct pattern in your code.
```

No error

13.2.29 Example 29

PEC

```
offset = 8
```

Solution code

```
if offset > 8: x = 5
elif offset > 5: x = 7
else: x = round(9)
```

Student code

```
if offset > 8: x = 6
elif offset > 5: x = 7
else: x = round(9)
```

No output

SCT

```
test_if_else(index=1,
    test=[
        test_expression_result({"offset": 7}),
        test_expression_result({"offset": 8}),
        test_expression_result({"offset": 9})
    ],
    body=test_student_typed('5'),
    orelse=test_if_else(index = 1,
        test=[
            test_expression_result({"offset": 4}),
            test_expression_result({"offset": 5}),
            test_expression_result({"offset": 6})
        ],
        body = test_student_typed('7'),
        orelse = test_function('round')
    )
)
```

Result

```
Check the first if expression. Did you correctly specify the body? Could not find the␣
↪correct pattern in your code.
```

No error

13.2.30 Example 30

PEC

```
offset = 8
```

Solution code

```
if offset > 8: x = 5
elif offset > 5: x = 7
else: x = round(9)
```

Student code

```
if offset > 8: x = 6
elif offset > 5: x = 7
else: x = round(9)
```

No output

SCT

```
Ex().check_if_else().multi(
    check_test().multi([ set_env(offset = i).has_equal_value() for i in range(7, 10)␣
↪]),
    check_body().has_code(r'x\s*=\s*5'),
    check_orelse().check_if_else().multi(
        check_test().multi([ set_env(offset = i).has_equal_value() for i in range(4,␣
↪7) ]),
        check_body().has_code('7'),
        check_orelse().check_function('round').check_args(0).has_equal_value()
    )
)
```

Result

```
Check the first if statement. Did you correctly specify the body? Could not find the␣
↪correct pattern in your code.
```

No error

13.2.31 Example 31

PEC

```
offset = 8
```

Solution code

```
if offset > 8: x = 5
elif offset > 5: x = 7
else: x = round(9)
```

Student code

```
if offset > 8: x = 5
elif offset > 6: x = 7
else: x = round(9)
```

No output

SCT

```
def test_test():
    test_expression_result({"offset": 7})
    test_expression_result({"offset": 8})
    test_expression_result({"offset": 9})

def body_test():
    test_student_typed('5')
```

(continues on next page)

(continued from previous page)

```
def orelse_test():
    def test_test2():
        test_expression_result({"offset": 4})
        test_expression_result({"offset": 5})
        test_expression_result({"offset": 6})
    def body_test2():
        test_student_typed('7')
    def orelse_test2():
        test_function('round')
    test_if_else(index = 1,
                 test = test_test2,
                 body = body_test2,
                 orelse = orelse_test2)

test_if_else(index=1,
             test=test_test,
             body=body_test,
             orelse=orelse_test)
```

Result

```
Check the first if expression. Did you correctly specify the condition? Expected
↪<code>True</code>, but got <code>False</code>.
```

No error

13.2.32 Example 32

PEC

```
offset = 8
```

Solution code

```
if offset > 8: x = 5
elif offset > 5: x = 7
else: x = round(9)
```

Student code

```
if offset > 8: x = 5
elif offset > 6: x = 7
else: x = round(9)
```

No output

SCT

```
test_if_else(index=1,
    test=[
        test_expression_result({"offset": 7}),
        test_expression_result({"offset": 8}),
        test_expression_result({"offset": 9})
    ],
```

(continues on next page)

(continued from previous page)

```
    body=test_student_typed('5'),
    orelse=test_if_else(index = 1,
        test=[
            test_expression_result({"offset": 4}),
            test_expression_result({"offset": 5}),
            test_expression_result({"offset": 6})
        ],
        body = test_student_typed('7'),
        orelse = test_function('round')
    )
)
```

Result

```
Check the first if expression. Did you correctly specify the condition? Expected
↪<code>True</code>, but got <code>False</code>.
```

No error

13.2.33 Example 33

PEC

```
offset = 8
```

Solution code

```
if offset > 8: x = 5
elif offset > 5: x = 7
else: x = round(9)
```

Student code

```
if offset > 8: x = 5
elif offset > 6: x = 7
else: x = round(9)
```

No output

SCT

```
Ex().check_if_else().multi(
    check_test().multi([ set_env(offset = i).has_equal_value() for i in range(7, 10)␣
↪]),
    check_body().has_code(r'x\s*=\s*5'),
    check_orelse().check_if_else().multi(
        check_test().multi([ set_env(offset = i).has_equal_value() for i in range(4,␣
↪7) ]),
        check_body().has_code('7'),
        check_orelse().check_function('round').check_args(0).has_equal_value()
    )
)
```

Result

```
Check the first if statement. Did you correctly specify the condition? Expected <code>
↪True</code>, but got <code>False</code>.
```

No error

13.2.34 Example 34

PEC

```
offset = 8
```

Solution code

```
if offset > 8: x = 5
elif offset > 5: x = 7
else: x = round(9)
```

Student code

```
if offset > 8: x = 5
elif offset > 5: x = 8
else: x = round(9)
```

No output

SCT

```
def test_test():
    test_expression_result({"offset": 7})
    test_expression_result({"offset": 8})
    test_expression_result({"offset": 9})

def body_test():
    test_student_typed('5')

def orelse_test():
    def test_test2():
        test_expression_result({"offset": 4})
        test_expression_result({"offset": 5})
        test_expression_result({"offset": 6})
    def body_test2():
        test_student_typed('7')
    def orelse_test2():
        test_function('round')
    test_if_else(index = 1,
                 test = test_test2,
                 body = body_test2,
                 orelse = orelse_test2)

test_if_else(index=1,
             test=test_test,
             body=body_test,
             orelse=orelse_test)
```

Result

```
Check the first if expression. Did you correctly specify the body? Could not find the␣
↪correct pattern in your code.
```

No error

13.2.35 Example 35

PEC

```
offset = 8
```

Solution code

```
if offset > 8: x = 5
elif offset > 5: x = 7
else: x = round(9)
```

Student code

```
if offset > 8: x = 5
elif offset > 5: x = 8
else: x = round(9)
```

No output

SCT

```
test_if_else(index=1,
    test=[
        test_expression_result({"offset": 7}),
        test_expression_result({"offset": 8}),
        test_expression_result({"offset": 9})
    ],
    body=test_student_typed('5'),
    orelse=test_if_else(index = 1,
        test=[
            test_expression_result({"offset": 4}),
            test_expression_result({"offset": 5}),
            test_expression_result({"offset": 6})
        ],
        body = test_student_typed('7'),
        orelse = test_function('round')
    )
)
```

Result

```
Check the first if expression. Did you correctly specify the body? Could not find the␣
↪correct pattern in your code.
```

No error

13.2.36 Example 36

PEC

```
offset = 8
```

Solution code

```
if offset > 8: x = 5
elif offset > 5: x = 7
else: x = round(9)
```

Student code

```
if offset > 8: x = 5
elif offset > 5: x = 8
else: x = round(9)
```

No output

SCT

```
Ex().check_if_else().multi(
    check_test().multi([ set_env(offset = i).has_equal_value() for i in range(7, 10)␣
↪]),
    check_body().has_code(r'x\s*=\s*5'),
    check_orelse().check_if_else().multi(
        check_test().multi([ set_env(offset = i).has_equal_value() for i in range(4,␣
↪7) ]),
        check_body().has_code('7'),
        check_orelse().check_function('round').check_args(0).has_equal_value()
    )
)
```

Result

```
Check the first if statement. Did you correctly specify the body? Could not find the␣
↪correct pattern in your code.
```

No error

13.2.37 Example 37

PEC

```
offset = 8
```

Solution code

```
if offset > 8: x = 5
elif offset > 5: x = 7
else: x = round(9)
```

Student code

```
if offset > 8: x = 5
elif offset > 5: x = 7
else: x = round(10)
```

No output

SCT

```
def test_test():
    test_expression_result({"offset": 7})
    test_expression_result({"offset": 8})
    test_expression_result({"offset": 9})

def body_test():
    test_student_typed('5')

def orelse_test():
    def test_test2():
        test_expression_result({"offset": 4})
        test_expression_result({"offset": 5})
        test_expression_result({"offset": 6})
    def body_test2():
        test_student_typed('7')
    def orelse_test2():
        test_function('round')
    test_if_else(index = 1,
                 test = test_test2,
                 body = body_test2,
                 orelse = orelse_test2)

test_if_else(index=1,
             test=test_test,
             body=body_test,
             orelse=orelse_test)
```

Result

```
Check your call of <code>round()</code>. Did you correctly specify the first argument?
↪ Expected <code>9</code>, but got <code>10</code>.
```

No error

13.2.38 Example 38

PEC

```
offset = 8
```

Solution code

```
if offset > 8: x = 5
elif offset > 5: x = 7
else: x = round(9)
```

Student code

```
if offset > 8: x = 5
elif offset > 5: x = 7
else: x = round(10)
```

No output

SCT

```
test_if_else(index=1,
    test=[
        test_expression_result({"offset": 7}),
        test_expression_result({"offset": 8}),
        test_expression_result({"offset": 9})
    ],
    body=test_student_typed('5'),
    orelse=test_if_else(index = 1,
        test=[
            test_expression_result({"offset": 4}),
            test_expression_result({"offset": 5}),
            test_expression_result({"offset": 6})
        ],
        body = test_student_typed('7'),
        orelse = test_function('round')
    )
)
```

Result

```
Check your call of <code>round()</code>. Did you correctly specify the first argument?
↪ Expected <code>9</code>, but got <code>10</code>.
```

No error

13.2.39 Example 39

PEC

```
offset = 8
```

Solution code

```
if offset > 8: x = 5
elif offset > 5: x = 7
else: x = round(9)
```

Student code

```
if offset > 8: x = 5
elif offset > 5: x = 7
else: x = round(10)
```

No output

SCT

```
Ex().check_if_else().multi(
    check_test().multi([ set_env(offset = i).has_equal_value() for i in range(7, 10)␣
↪]),
    check_body().has_code(r'x\s*=\s*5'),
    check_orelse().check_if_else().multi(
        check_test().multi([ set_env(offset = i).has_equal_value() for i in range(4,␣
↪7) ]),
        check_body().has_code('7'),
        check_orelse().check_function('round').check_args(0).has_equal_value()
```

(continues on next page)

(continued from previous page)

```
    )
)
```

Result

```
Check your call of <code>round()</code>. Did you correctly specify the first argument?
↪ Expected <code>9</code>, but got <code>10</code>.
```

No error

13.2.40 Example 40

PEC

```
offset = 8
```

Solution code

```
if offset > 8: x = 5
elif offset > 5: x = 7
else: x = round(9)
```

Student code

```
if offset > 8: x = 5
elif offset > 5: x = 7
else: x = round(9)
```

No output

SCT

```
def test_test():
    test_expression_result({"offset": 7})
    test_expression_result({"offset": 8})
    test_expression_result({"offset": 9})

def body_test():
    test_student_typed('5')

def orelse_test():
    def test_test2():
        test_expression_result({"offset": 4})
        test_expression_result({"offset": 5})
        test_expression_result({"offset": 6})
    def body_test2():
        test_student_typed('7')
    def orelse_test2():
        test_function('round')
    test_if_else(index = 1,
                 test = test_test2,
                 body = body_test2,
                 orelse = orelse_test2)

test_if_else(index=1,
```

(continues on next page)

(continued from previous page)

```
            test=test_test,
            body=body_test,
            orelse=orelse_test)
```

Result

```
Great work!
```

No error

13.2.41 Example 41

PEC

```
offset = 8
```

Solution code

```
if offset > 8: x = 5
elif offset > 5: x = 7
else: x = round(9)
```

Student code

```
if offset > 8: x = 5
elif offset > 5: x = 7
else: x = round(9)
```

No output

SCT

```
test_if_else(index=1,
    test=[
        test_expression_result({"offset": 7}),
        test_expression_result({"offset": 8}),
        test_expression_result({"offset": 9})
    ],
    body=test_student_typed('5'),
    orelse=test_if_else(index = 1,
        test=[
            test_expression_result({"offset": 4}),
            test_expression_result({"offset": 5}),
            test_expression_result({"offset": 6})
        ],
        body = test_student_typed('7'),
        orelse = test_function('round')
    )
)
```

Result

```
Great work!
```

No error

13.2.42 Example 42

PEC

```
offset = 8
```

Solution code

```
if offset > 8: x = 5
elif offset > 5: x = 7
else: x = round(9)
```

Student code

```
if offset > 8: x = 5
elif offset > 5: x = 7
else: x = round(9)
```

No output

SCT

```
Ex().check_if_else().multi(
    check_test().multi([ set_env(offset = i).has_equal_value() for i in range(7, 10)␣
↪]),
    check_body().has_code(r'x\s*=\s*5'),
    check_orelse().check_if_else().multi(
        check_test().multi([ set_env(offset = i).has_equal_value() for i in range(4,␣
↪7) ]),
        check_body().has_code('7'),
        check_orelse().check_function('round').check_args(0).has_equal_value()
    )
)
```

Result

```
Great work!
```

No error

13.2.43 Example 43

PEC

```
offset = 8
```

Solution code

```
x = 5 if offset > 8 else 7 if offset > 5 else round(9)
```

No student code

No output

SCT

```
Ex().check_if_exp().multi(
    check_test().multi([ set_env(offset = i).has_equal_value() for i in range(7, 10)␣
↪]),
    check_body().has_code('5'),
    check_orelse().check_if_exp().multi(
        check_test().multi([ set_env(offset = i).has_equal_value() for i in range(4,␣
↪7) ]),
        check_body().has_code('7'),
        check_orelse().check_function('round').check_args(0).has_equal_value()
    )
)
```

Result

```
The system wants to check the first if expression but hasn't found it.
```

No error

13.2.44 Example 44

PEC

```
offset = 8
```

Solution code

```
x = 5 if offset > 8 else 7 if offset > 5 else round(9)
```

Student code

```
x = 5 if offset > 9 else 7 if offset > 5 else round(9)
```

No output

SCT

```
Ex().check_if_exp().multi(
    check_test().multi([ set_env(offset = i).has_equal_value() for i in range(7, 10)␣
↪]),
    check_body().has_code('5'),
    check_orelse().check_if_exp().multi(
        check_test().multi([ set_env(offset = i).has_equal_value() for i in range(4,␣
↪7) ]),
        check_body().has_code('7'),
        check_orelse().check_function('round').check_args(0).has_equal_value()
    )
)
```

Result

```
Check the first if expression. Did you correctly specify the condition? Expected
↪<code>True</code>, but got <code>False</code>.
```

No error

13.2.45 Example 45

PEC

```
offset = 8
```

Solution code

```
x = 5 if offset > 8 else 7 if offset > 5 else round(9)
```

Student code

```
x = 6 if offset > 8 else 7 if offset > 5 else round(9)
```

No output

SCT

```
Ex().check_if_exp().multi(
    check_test().multi([ set_env(offset = i).has_equal_value() for i in range(7, 10)␣
↪]),
    check_body().has_code('5'),
    check_orelse().check_if_exp().multi(
        check_test().multi([ set_env(offset = i).has_equal_value() for i in range(4,␣
↪7) ]),
        check_body().has_code('7'),
        check_orelse().check_function('round').check_args(0).has_equal_value()
    )
)
```

Result

```
Check the first if expression. Did you correctly specify the body? Could not find the␣
↪correct pattern in your code.
```

No error

13.2.46 Example 46

PEC

```
offset = 8
```

Solution code

```
x = 5 if offset > 8 else 7 if offset > 5 else round(9)
```

Student code

```
x = 5 if offset > 8 else 7 if offset > 6 else round(9)
```

No output

SCT

```
Ex().check_if_exp().multi(
    check_test().multi([ set_env(offset = i).has_equal_value() for i in range(7, 10)␣
↪]),
    check_body().has_code('5'),
    check_orelse().check_if_exp().multi(
        check_test().multi([ set_env(offset = i).has_equal_value() for i in range(4,␣
↪7) ]),
        check_body().has_code('7'),
        check_orelse().check_function('round').check_args(0).has_equal_value()
    )
)
```

Result

```
Check the first if expression. Did you correctly specify the condition? Expected
↪<code>True</code>, but got <code>False</code>.
```

No error

13.2.47 Example 47

PEC

```
offset = 8
```

Solution code

```
x = 5 if offset > 8 else 7 if offset > 5 else round(9)
```

Student code

```
x = 5 if offset > 8 else 8 if offset > 5 else round(9)
```

No output

SCT

```
Ex().check_if_exp().multi(
    check_test().multi([ set_env(offset = i).has_equal_value() for i in range(7, 10)␣
↪]),
    check_body().has_code('5'),
    check_orelse().check_if_exp().multi(
        check_test().multi([ set_env(offset = i).has_equal_value() for i in range(4,␣
↪7) ]),
        check_body().has_code('7'),
        check_orelse().check_function('round').check_args(0).has_equal_value()
    )
)
```

Result

```
Check the first if expression. Did you correctly specify the body? Could not find the␣
↪correct pattern in your code.
```

No error

13.2.48 Example 48

PEC

```
offset = 8
```

Solution code

```
x = 5 if offset > 8 else 7 if offset > 5 else round(9)
```

Student code

```
x = 5 if offset > 8 else 7 if offset > 5 else round(10)
```

No output

SCT

```
Ex().check_if_exp().multi(
    check_test().multi([ set_env(offset = i).has_equal_value() for i in range(7, 10)␣
↪]),
    check_body().has_code('5'),
    check_orelse().check_if_exp().multi(
        check_test().multi([ set_env(offset = i).has_equal_value() for i in range(4,␣
↪7) ]),
        check_body().has_code('7'),
        check_orelse().check_function('round').check_args(0).has_equal_value()
    )
)
```

Result

```
Check your call of <code>round()</code>. Did you correctly specify the first argument?
↪ Expected <code>9</code>, but got <code>10</code>.
```

No error

13.2.49 Example 49

PEC

```
offset = 8
```

Solution code

```
x = 5 if offset > 8 else 7 if offset > 5 else round(9)
```

Student code

```
x = 5 if offset > 8 else 7 if offset > 5 else round(9)
```

No output

SCT

```
Ex().check_if_exp().multi(
    check_test().multi([ set_env(offset = i).has_equal_value() for i in range(7, 10)␣
↪]),
    check_body().has_code('5'),
    check_orelse().check_if_exp().multi(
        check_test().multi([ set_env(offset = i).has_equal_value() for i in range(4,␣
↪7) ]),
        check_body().has_code('7'),
        check_orelse().check_function('round').check_args(0).has_equal_value()
    )
)
```

Result

```
Great work!
```

No error

13.3 test_spec.py

13.3.1 Example 1

No PEC

Solution code

```
[[ii+1 for ii in range(aa)] for aa in range(2)]
[[ii*2 for ii in range(bb)] for bb in range(1,3)]
```

Student code

```
[[ii+1 for ii in range(aa)] for aa in range(2)]
[[ii*2 for ii in range(bb)] for bb in range(1,3)]
```

No output

SCT

```
list_comp = F().check_list_comp(0).check_body().set_context(ii=2).has_equal_value(
↪'unequal')
Ex().check_list_comp(0).check_body().set_context(aa=2).multi(list_comp)
```

Result

```
Great work!
```

No error

13.3.2 Example 2

No PEC

Solution code

```
[[ii+1 for ii in range(aa)] for aa in range(2)]
[[ii*2 for ii in range(bb)] for bb in range(1,3)]
```

Student code

```
[[ii+1 for ii in range(aa)] for aa in range(2)]
[[ii*2 for ii in range(bb)] for bb in range(1,3)]
```

No output

SCT

```
list_comp = check_list_comp(0).check_body().set_context(ii=2).has_equal_value('unequal
↪')
Ex().check_list_comp(0).check_body().set_context(aa=2).multi(list_comp)
```

Result

```
Great work!
```

No error

13.3.3 Example 3

No PEC

Solution code

```
[[ii+1 for ii in range(aa)] for aa in range(2)]
[[ii*2 for ii in range(bb)] for bb in range(1,3)]
```

Student code

```
[[ii+1 for ii in range(aa)] for aa in range(2)]
[[ii*2 for ii in range(bb)] for bb in range(1,3)]
```

No output

SCT

```
# funky, but we're testing nested check functions!
multi_test = multi(check_list_comp(0).check_body().set_context(aa=2).has_equal_value(
↪'badbody'))
Ex().multi(multi_test)
```

Result

```
Great work!
```

No error

13.3.4 Example 4

No PEC

Solution code

```
[[ii+1 for ii in range(aa)] for aa in range(2)]
[[ii*2 for ii in range(bb)] for bb in range(1,3)]
```

Student code

```
[[ii+1 for ii in range(aa)] for aa in range(2)]
[[ii*2 for ii in range(bb)] for bb in range(1,3)]
```

No output

SCT

```
list_comp = F().check_list_comp(0).check_body().set_context(ii=2).has_equal_value(
↪'unequal')
Ex().check_list_comp(0).check_body().set_context(aa=2).multi(list_comp)
Ex().check_list_comp(1).check_body().set_context(bb=4).multi(list_comp)
```

Result

```
Great work!
```

No error

13.3.5 Example 5

No PEC

Solution code

```
[[ii+1 for ii in range(aa)] for aa in range(2)]
[[ii*2 for ii in range(bb)] for bb in range(1,3)]
```

Student code

```
[[ii+1 for ii in range(aa)] for aa in range(2)]
[[ii*2 for ii in range(bb)] for bb in range(1,3)]
```

No output

SCT

```
eq_test = F().check_list_comp(0).check_body().set_context(1).has_equal_value
Ex().multi(eq_test('unequal'))
```

Result

```
Great work!
```

No error

13.3.6 Example 6

No PEC

Solution code

```
[aa+1 for aa in range(2)]
```

Student code

```
[aa+1 for aa in range(2)]
```

No output

SCT

```
te = test_expression_result(extra_env={'aa':2}, incorrect_msg='unequal')
Ex().multi(test_list_comp(body=te))
↪     # spec 1 inside multi
Ex().check_list_comp(0).check_body().multi(te)
↪     # half of each spec
Ex().check_list_comp(0).check_body().set_context(aa=2).has_equal_value('unequal')
↪     # full spec 2
test_list_comp(body=te)
↪     # full spec 1
```

Result

```
Great work!
```

No error

13.3.7 Example 7

No PEC

Solution code

```
[aa+1 for aa in range(2)]
```

Student code

```
for aa in range(3): aa
```

No output

SCT

```
test_list_comp(body=test_expression_result(expr_code = 'aa', incorrect_msg='unequal'))
Ex().check_list_comp(0).check_body().multi(test_expression_result(incorrect_msg=
↪'unequal'))
Ex().check_list_comp(0).check_body().has_equal_value('unequal')
```

Result

```
The system wants to check the first list comprehension but hasn't found it.
```

No error

13.3.8 Example 8

No PEC

Solution code

```
[aa+1 for aa in range(2)]
```

Student code

```
[aa for aa in range(2)]
```

No output

SCT

```
Ex().test_list_comp(body=test_expression_result(extra_env={'aa': 2}, incorrect_msg =
↪'spec1'))
Ex().check_list_comp(0).check_body().set_context(aa=2).has_equal_value('spec2')
```

Result

```
spec1
```

No error

13.3.9 Example 9

No PEC

Solution code

```
[aa+1 for aa in range(2)]
```

Student code

```
[aa+1 for aa in range(2)]
```

No output

SCT

```
test_body = F().check_body().set_context(aa=2).has_equal_value('wrong')
Ex().check_list_comp(0).multi(F().multi(test_body))
```

Result

```
Great work!
```

No error

13.3.10 Example 10

No PEC

Solution code

```
[aa+1 for aa in range(2)]
```

Student code

```
[aa+1 for aa in range(2)]
```

No output

SCT

```
test_body = F().check_list_comp(0).check_body().set_context(aa=2).has_equal_value(
↪'wrong')
Ex().check_list_comp(0).multi(F().check_body().set_context(aa=2).has_equal_value(
↪'wrong'))
```

Result

```
Great work!
```

No error

13.3.11 Example 11

No PEC

Solution code

```
[aa+1 for aa in range(2)]
```

Student code

```
[aa+1 for aa in range(2)]
```

No output

SCT

```
Ex().check_list_comp(0).check_body()         .multi(set_context(aa=i).has_equal_value(
↪'wrong') for i in range(2))
```

Result

```
Great work!
```

No error

13.3.12 Example 12

No PEC

No solution code

No student code

No output

SCT

```
Ex().fail()
```

Result

```
fail
```

No error

13.3.13 Example 13

No PEC

Solution code

```
dict(a = "a").keys()
```

Student code

```
dict(a = 'a')     .keys()
```

No output

SCT

```
Ex().has_equal_ast()
```

Result

```
Great work!
```

No error

13.3.14 Example 14

No PEC

Solution code

```
dict(a = "a").keys()
```

Student code

```
dict(A = 'a').keys(somearg = 2)
```

No output

SCT

```
Ex().has_equal_ast()
```

Result

```
Expected <code>dict(a = "a").keys()</code>, but got <code>dict(A = 'a').keys(somearg␣
↪= 2)</code>.
```

No error

13.3.15 Example 15

No PEC

Solution code

```
dict(a = "a").keys()
```

Student code

```
dict(a = 'a')     .keys()
```

No output

SCT

```
Ex().check_function('dict', 0, signature=False).has_equal_ast()
```

Result

```
Great work!
```

No error

13.3.16 Example 16

No PEC

Solution code

```
dict(a = "a").keys()
```

Student code

```
dict(A = 'a').keys(somearg = 2)
```

No output

SCT

```
Ex().check_function('dict', 0, signature=False).has_equal_ast()
```

Result

```
Check your call of <code>dict()</code>. Expected <code>dict(a = "a")</code>, but got
↪<code>dict(A = 'a')</code>.
```

No error

13.3.17 Example 17

No PEC

Solution code

```
dict(a = "a").keys()
```

Student code

```
dict(a = 'a')     .keys()
```

No output

SCT

```
Ex().has_equal_ast(code = 'dict(a = "a").keys()', incorrect_msg = 'icr')
```

Result

```
Great work!
```

No error

13.3.18 Example 18

No PEC

Solution code

```
dict(a = "a").keys()
```

Student code

```
dict(A = 'a').keys(somearg = 2)
```

No output

SCT

```
Ex().has_equal_ast(code = 'dict(a = "a").keys()', incorrect_msg = 'icr')
```

Result

```
icr
```

No error

13.3.19 Example 19

No PEC

Solution code

```
dict(a = "a").keys()
```

Student code

```
dict(a = 'a').keys()
print('extra')
```

Student output

```
extra
```

SCT

```
Ex().has_equal_ast(exact=False)
```

Result

```
Great work!
```

No error

13.3.20 Example 20

No PEC

Solution code

```
dict(a = "a").keys()
```

Student code

```
dict(A = 'a').keys(somearg = 2)
```

No output

SCT

```
Ex().has_equal_ast(exact=False)
```

Result

```
Expected <code>dict(a = "a").keys()</code>, but got <code>dict(A = 'a').keys(somearg␣
↪= 2)</code>.
```

No error

13.3.21 Example 21

No PEC

Solution code

```
dict(a = "a").keys()
```

Student code

```
dict(a = 'a')     .keys()
```

No output

SCT

```
Ex().has_equal_ast(code = 'dict(a = "a")', exact=False, incorrect_msg = 'icr')
```

Result

```
Great work!
```

No error

13.3.22 Example 22

No PEC

Solution code

```
dict(a = "a").keys()
```

Student code

```
dict(A = 'a').keys(somearg = 2)
```

No output

SCT

```
Ex().has_equal_ast(code = 'dict(a = "a")', exact=False, incorrect_msg = 'icr')
```

Result

```
icr
```

No error

13.3.23 Example 23

No PEC

Solution code

```
1 if False else 2
```

Student code

```
1 if False else 2
```

No output

SCT

```
Ex().check_if_exp(0).override("""1 if False else 2""").has_equal_ast()
```

Result

```
Great work!
```

No error

13.3.24 Example 24

No PEC

Solution code

```
1 if False else 2
```

Student code

```
1 if False else 2
```

No output

SCT

```
Ex().check_if_exp(0).override("""'WRONG ANSWER'""").has_equal_ast()
```

Result

```
Check the first if expression. Expected `<code>, but got</code>1 if False else 2`.
```

No error

13.3.25 Example 25

No PEC

Solution code

```
1 if False else 2
```

Student code

```
1 if False else 2
```

No output

SCT

```
Ex().check_if_exp(0).check_body().override("""1""").has_equal_ast()
```

Result

```
Great work!
```

No error

13.3.26 Example 26

No PEC

Solution code

```
1 if False else 2
```

Student code

```
1 if False else 2
```

No output

SCT

```
Ex().check_if_exp(0).check_body().override("""1 if False else 2""").has_equal_ast()
```

Result

```
Did you correctly specify the body? Expected `<code>, but got</code>1`.
```

No error

13.3.27 Example 27

No PEC

Solution code

```
1 if False else 2
```

Student code

```
1 if False else 2
```

No output

SCT

```
Ex().check_if_exp(0).check_test().override("""False""").has_equal_ast()
```

Result

```
Great work!
```

No error

13.3.28 Example 28

No PEC

Solution code

```
1 if False else 2
```

Student code

```
1 if False else 2
```

No output

SCT

```
Ex().check_if_exp(0).check_test().override("""1 if False else 2""").has_equal_ast()
```

Result

```
Did you correctly specify the condition? Expected `<code>, but got</code>False`.
```

No error

13.3.29 Example 29

No PEC

Solution code

```
1 if False else 2
```

Student code

```
1 if False else 2
```

No output

SCT

```
Ex().check_if_exp(0).check_orelse().override("""2""").has_equal_ast()
```

Result

```
Great work!
```

No error

13.3.30 Example 30

No PEC

Solution code

```
1 if False else 2
```

Student code

```
1 if False else 2
```

No output

SCT

```
Ex().check_if_exp(0).check_orelse().override("""1 if False else 2""").has_equal_ast()
```

Result

```
Did you correctly specify the else part? Expected `<code>, but got</code>2`.
```

No error

13.3.31 Example 31

No PEC

Solution code

```
[1 for i in range(3)]
```

Student code

```
[1 for i in range(3)]
```

No output

SCT

```
Ex().check_list_comp(0).override("""[1 for i in range(3)]""").has_equal_ast()
```

Result

```
Great work!
```

No error

13.3.32 Example 32

No PEC

Solution code

```
[1 for i in range(3)]
```

Student code

```
[1 for i in range(3)]
```

No output

SCT

```
Ex().check_list_comp(0).override("""'WRONG ANSWER'""").has_equal_ast()
```

Result

```
Check the first list comprehension. Expected `<code>, but got</code>[1 for i in␣
↪range(3)]`.
```

No error

13.3.33 Example 33

No PEC

Solution code

```
[1 for i in range(3)]
```

Student code

```
[1 for i in range(3)]
```

No output

SCT

```
Ex().check_list_comp(0).check_body().override("""1""").has_equal_ast()
```

Result

```
Great work!
```

No error

13.3.34 Example 34

No PEC

Solution code

```
[1 for i in range(3)]
```

Student code

```
[1 for i in range(3)]
```

No output

SCT

```
Ex().check_list_comp(0).check_body().override("""[1 for i in range(3)]""").has_equal_
↪ast()
```

Result

```
Did you correctly specify the body? Expected `<code>, but got</code>1`.
```

No error

13.3.35 Example 35

No PEC

Solution code

```
[1 for i in range(3)]
```

Student code

```
[1 for i in range(3)]
```

No output

SCT

```
Ex().check_list_comp(0).check_iter().override("""range(3)""").has_equal_ast()
```

Result

```
Great work!
```

No error

13.3.36 Example 36

No PEC

Solution code

```
[1 for i in range(3)]
```

Student code

```
[1 for i in range(3)]
```

No output

SCT

```
Ex().check_list_comp(0).check_iter().override("""[1 for i in range(3)]""").has_equal_
↪ast()
```

Result

```
Did you correctly specify the iterable part? Expected `<code>, but got</code>
↪range(3)`.
```

No error

13.3.37 Example 37

No PEC

Solution code

```
{ 3: 4  for i in range(3) }
```

Student code

```
{ 3: 4  for i in range(3) }
```

No output

SCT

```
Ex().check_dict_comp(0).override("""{ 3: 4  for i in range(3) }""").has_equal_ast()
```

Result

```
Great work!
```

No error

13.3.38 Example 38

No PEC

Solution code

```
{ 3: 4  for i in range(3) }
```

Student code

```
{ 3: 4  for i in range(3) }
```

No output

SCT

```
Ex().check_dict_comp(0).override("""'WRONG ANSWER'""").has_equal_ast()
```

Result

```
Check the first dictionary comprehension. Expected `<code>, but got</code>{ 3: 4  for␣
↪i in range(3) }`.
```

No error

13.3.39 Example 39

No PEC

Solution code

```
{ 3: 4  for i in range(3) }
```

Student code

```
{ 3: 4  for i in range(3) }
```

No output

SCT

```
Ex().check_dict_comp(0).check_key().override("""3""").has_equal_ast()
```

Result

```
Great work!
```

No error

13.3.40 Example 40

No PEC

Solution code

```
{ 3: 4  for i in range(3) }
```

Student code

```
{ 3: 4  for i in range(3) }
```

No output

SCT

```
Ex().check_dict_comp(0).check_key().override("""{ 3: 4  for i in range(3) }""").has_
↪equal_ast()
```

Result

```
Did you correctly specify the key part? Expected `<code>, but got</code>3`.
```

No error

13.3.41 Example 41

No PEC

Solution code

```
{ 3: 4  for i in range(3) }
```

Student code

```
{ 3: 4  for i in range(3) }
```

No output

SCT

```
Ex().check_dict_comp(0).check_value().override("""4""").has_equal_ast()
```

Result

```
Great work!
```

No error

13.3.42 Example 42

No PEC

Solution code

```
{ 3: 4  for i in range(3) }
```

Student code

```
{ 3: 4  for i in range(3) }
```

No output

SCT

```
Ex().check_dict_comp(0).check_value().override("""{ 3: 4  for i in range(3) }""").has_
↪equal_ast()
```

Result

```
Did you correctly specify the value part? Expected `<code>, but got</code>4`.
```

No error

13.3.43 Example 43

No PEC

Solution code

```
{ 3: 4  for i in range(3) }
```

Student code

```
{ 3: 4  for i in range(3) }
```

No output

SCT

```
Ex().check_dict_comp(0).check_iter().override("""range(3)""").has_equal_ast()
```

Result

```
Great work!
```

No error

13.3.44 Example 44

No PEC

Solution code

```
{ 3: 4  for i in range(3) }
```

Student code

```
{ 3: 4  for i in range(3) }
```

No output

SCT

```
Ex().check_dict_comp(0).check_iter().override("""{ 3: 4  for i in range(3) }""").has_
↪equal_ast()
```

Result

```
Did you correctly specify the iterable part? Expected `<code>, but got</code>
↪range(3)`.
```

No error

13.3.45 Example 45

No PEC

Solution code

```
for i in range(3): 1
```

Student code

```
for i in range(3): 1
```

No output

SCT

```
Ex().check_for_loop(0).override("""for i in range(3): 1""").has_equal_ast()
```

Result

```
Great work!
```

No error

13.3.46 Example 46

No PEC

Solution code

```
for i in range(3): 1
```

Student code

```
for i in range(3): 1
```

No output

SCT

```
Ex().check_for_loop(0).override("""'WRONG ANSWER'""").has_equal_ast()
```

Result

```
Check the first for loop. Expected `<code>, but got</code>for i in range(3): 1`.
```

No error

13.3.47 Example 47

No PEC

Solution code

```
for i in range(3): 1
```

Student code

```
for i in range(3): 1
```

No output

SCT

```
Ex().check_for_loop(0).check_iter().override("""range(3)""").has_equal_ast()
```

Result

```
Great work!
```

No error

13.3.48 Example 48

No PEC

Solution code

```
for i in range(3): 1
```

Student code

```
for i in range(3): 1
```

No output

SCT

```
Ex().check_for_loop(0).check_iter().override("""for i in range(3): 1""").has_equal_
↪ast()
```

Result

```
Did you correctly specify the iterable part? Expected `<code>, but got</code>
↪range(3)`.
```

No error

13.3.49 Example 49

No PEC

Solution code

```
for i in range(3): 1
```

Student code

```
for i in range(3): 1
```

No output

SCT

```
Ex().check_for_loop(0).check_body().override("""1""").has_equal_ast()
```

Result

```
Great work!
```

No error

13.3.50 Example 50

No PEC

Solution code

```
for i in range(3): 1
```

Student code

```
for i in range(3): 1
```

No output

SCT

```
Ex().check_for_loop(0).check_body().override("""for i in range(3): 1""").has_equal_
↪ast()
```

Result

```
Did you correctly specify the body? Expected `<code>, but got</code>1`.
```

No error

13.3.51 Example 51

No PEC

Solution code

```
while False: 1
```

Student code

```
while False: 1
```

No output

SCT

```
Ex().check_while(0).override("""while False: 1""").has_equal_ast()
```

Result

```
Great work!
```

No error

13.3.52 Example 52

No PEC

Solution code

```
while False: 1
```

Student code

```
while False: 1
```

No output

SCT

```
Ex().check_while(0).override("""'WRONG ANSWER'""").has_equal_ast()
```

Result

```
Check the first <code>while</code> loop. Expected `<code>, but got</code>while False:␣
↪1`.
```

No error

13.3.53 Example 53

No PEC

Solution code

```
while False: 1
```

Student code

```
while False: 1
```

No output

SCT

```
Ex().check_while(0).check_test().override("""False""").has_equal_ast()
```

Result

```
Great work!
```

No error

13.3.54 Example 54

No PEC

Solution code

```
while False: 1
```

Student code

```
while False: 1
```

No output

SCT

```
Ex().check_while(0).check_test().override("""while False: 1""").has_equal_ast()
```

Result

```
Did you correctly specify the condition? Expected `<code>, but got</code>False`.
```

No error

13.3.55 Example 55

No PEC

Solution code

```
while False: 1
```

Student code

```
while False: 1
```

No output

SCT

```
Ex().check_while(0).check_body().override("""1""").has_equal_ast()
```

Result

```
Great work!
```

No error

13.3.56 Example 56

No PEC

Solution code

```
while False: 1
```

Student code

```
while False: 1
```

No output

SCT

```
Ex().check_while(0).check_body().override("""while False: 1""").has_equal_ast()
```

Result

```
Did you correctly specify the body? Expected `<code>, but got</code>1`.
```

No error

13.3.57 Example 57

No PEC

Solution code

```
try: 1
except: pass
else: 2
```

Student code

```
try: 1
except: pass
else: 2
```

No output

SCT

```
Ex().check_try_except(0).override("""try: 1
except: pass
else: 2""").has_equal_ast()
```

Result

```
Great work!
```

No error

13.3.58 Example 58

No PEC

Solution code

```
try: 1
except: pass
else: 2
```

Student code

```
try: 1
except: pass
else: 2
```

No output

SCT

```
Ex().check_try_except(0).override("""'WRONG ANSWER'""").has_equal_ast()
```

Result

```
<p>Check the first try statement. Expected `<code>, but got</code>try: 1
except: pass
else: 2`.</p>
```

No error

13.3.59 Example 59

No PEC

Solution code

```
try: 1
except: pass
else: 2
```

Student code

```
try: 1
except: pass
else: 2
```

No output

SCT

```
Ex().check_try_except(0).check_body().override("""1""").has_equal_ast()
```

Result

```
Great work!
```

No error

13.3.60 Example 60

No PEC

Solution code

```
try: 1
except: pass
else: 2
```

Student code

```
try: 1
except: pass
else: 2
```

No output

SCT

```
Ex().check_try_except(0).check_body().override("""try: 1
except: pass
else: 2""").has_equal_ast()
```

Result

```
Did you correctly specify the body? Expected `<code>, but got</code>1`.
```

No error

13.3.61 Example 61

No PEC

Solution code

```
try: 1
except: pass
else: 2
```

Student code

```
try: 1
except: pass
else: 2
```

No output

SCT

```
Ex().check_try_except(0).check_orelse().override("""2""").has_equal_ast()
```

Result

```
Great work!
```

No error

13.3.62 Example 62

No PEC

Solution code

```
try: 1
except: pass
else: 2
```

Student code

```
try: 1
except: pass
else: 2
```

No output

SCT

```
Ex().check_try_except(0).check_orelse().override("""try: 1
except: pass
else: 2""").has_equal_ast()
```

Result

```
Did you correctly specify the else part? Expected `<code>, but got</code>2`.
```

No error

13.3.63 Example 63

No PEC

Solution code

```
lambda a=(1,2,3): 1
```

Student code

```
lambda a=(1,2,3): 1
```

No output

SCT

```
Ex().check_lambda_function(0).override("""lambda a=(1,2,3): 1""").has_equal_ast()
```

Result

```
Great work!
```

No error

13.3.64 Example 64

No PEC

Solution code

```
lambda a=(1,2,3): 1
```

Student code

```
lambda a=(1,2,3): 1
```

No output

SCT

```
Ex().check_lambda_function(0).override("""'WRONG ANSWER'""").has_equal_ast()
```

Result

```
Check the first lambda function. Expected `<code>, but got</code>lambda a=(1,2,3): 1`.
```

No error

13.3.65 Example 65

No PEC

Solution code

```
lambda a=(1,2,3): 1
```

Student code

```
lambda a=(1,2,3): 1
```

No output

SCT

```
Ex().check_lambda_function(0).check_args(0).override("""(1,2,3)""").has_equal_ast()
```

Result

```
Great work!
```

No error

13.3.66 Example 66

No PEC

Solution code

```
lambda a=(1,2,3): 1
```

Student code

```
lambda a=(1,2,3): 1
```

No output

SCT

```
Ex().check_lambda_function(0).check_args(0).override("""lambda a=(1,2,3): 1""").has_
↪equal_ast()
```

Result

```
Did you correctly specify the first argument? Expected `<code>, but got</code>1,2,3`.
```

No error

13.3.67 Example 67

No PEC

Solution code

```
lambda a=(1,2,3): 1
```

Student code

```
lambda a=(1,2,3): 1
```

No output

SCT

```
Ex().check_lambda_function(0).check_body().override("""1""").has_equal_ast()
```

Result

```
Great work!
```

No error

13.3.68 Example 68

No PEC

Solution code

```
lambda a=(1,2,3): 1
```

Student code

```
lambda a=(1,2,3): 1
```

No output

SCT

```
Ex().check_lambda_function(0).check_body().override("""lambda a=(1,2,3): 1""").has_
↪equal_ast()
```

Result

```
Did you correctly specify the body? Expected `<code>, but got</code>1`.
```

No error

13.3.69 Example 69

No PEC

Solution code

```
def sum(a=(1,2,3)): 1
```

Student code

```
def sum(a=(1,2,3)): 1
```

No output

SCT

```
Ex().check_function_def('sum').override("""def sum(a=(1,2,3)): 1""").has_equal_ast()
```

Result

```
Great work!
```

No error

13.3.70 Example 70

No PEC

Solution code

```
def sum(a=(1,2,3)): 1
```

Student code

```
def sum(a=(1,2,3)): 1
```

No output

SCT

```
Ex().check_function_def('sum').override("""'WRONG ANSWER'""").has_equal_ast()
```

Result

```
Check the definition of <code>sum()</code>. Expected `<code>, but got</code>def␣
↪sum(a=(1,2,3)): 1`.
```

No error

13.3.71 Example 71

No PEC

Solution code

```
def sum(a=(1,2,3)): 1
```

Student code

```
def sum(a=(1,2,3)): 1
```

No output

SCT

```
Ex().check_function_def('sum').check_args(0).override("""(1,2,3)""").has_equal_ast()
```

Result

```
Great work!
```

No error

13.3.72 Example 72

No PEC

Solution code

```
def sum(a=(1,2,3)): 1
```

Student code

```
def sum(a=(1,2,3)): 1
```

No output

SCT

```
Ex().check_function_def('sum').check_args(0).override("""def sum(a=(1,2,3)): 1""").
↪has_equal_ast()
```

Result

```
Did you correctly specify the first argument? Expected `<code>, but got</code>1,2,3`.
```

No error

13.3.73 Example 73

No PEC

Solution code

```
def sum(a=(1,2,3)): 1
```

Student code

```
def sum(a=(1,2,3)): 1
```

No output

SCT

```
Ex().check_function_def('sum').check_body().override("""1""").has_equal_ast()
```

Result

```
Great work!
```

No error

13.3.74 Example 74

No PEC

Solution code

```
def sum(a=(1,2,3)): 1
```

Student code

```
def sum(a=(1,2,3)): 1
```

No output

SCT

```
Ex().check_function_def('sum').check_body().override("""def sum(a=(1,2,3)): 1""").has_
↪equal_ast()
```

Result

```
Did you correctly specify the body? Expected `<code>, but got</code>1`.
```

No error

13.3.75 Example 75

No PEC

Solution code

```
sum((1,2,3))
```

Student code

```
sum((1,2,3))
```

No output

SCT

```
Ex().check_function('sum', 0).override("""sum((1,2,3))""").has_equal_ast()
```

Result

```
Great work!
```

No error

13.3.76 Example 76

No PEC

Solution code

```
sum((1,2,3))
```

Student code

```
sum((1,2,3))
```

No output

SCT

```
Ex().check_function('sum', 0).override("""'WRONG ANSWER'""").has_equal_ast()
```

Result

```
Check your call of <code>sum()</code>. Expected `<code>, but got</code>sum((1,2,3))`.
```

No error

13.3.77 Example 77

No PEC

Solution code

```
sum((1,2,3))
```

Student code

```
sum((1,2,3))
```

No output

SCT

```
Ex().check_function('sum', 0).check_args(0).override("""(1,2,3)""").has_equal_ast()
```

Result

```
Great work!
```

No error

13.3.78 Example 78

No PEC

Solution code

```
sum((1,2,3))
```

Student code

```
sum((1,2,3))
```

No output

SCT

```
Ex().check_function('sum', 0).check_args(0).override("""sum((1,2,3))""").has_equal_
↪ast()
```

Result

```
Did you correctly specify the first argument? Expected `<code>, but got</code>1,2,3`.
```

No error

13.4 test_check_object.py

13.4.1 Example 1

No PEC

Solution code

```
x = 100
```

No student code

No output

SCT

```
test_object('x', undefined_msg='udm', incorrect_msg='icm')
```

Result

```
udm
```

No error

13.4.2 Example 2

No PEC

Solution code

```
x = 100
```

No student code

No output

SCT

```
Ex().check_object('x', missing_msg='udm').has_equal_value(incorrect_msg='icm')
```

Result

```
udm
```

No error

13.4.3 Example 3

No PEC

Solution code

```
x = 100
```

Student code

```
x = 1
```

No output

SCT

```
test_object('x', undefined_msg='udm', incorrect_msg='icm')
```

Result

```
icm
```

No error

13.4.4 Example 4

No PEC

Solution code

```
x = 100
```

Student code

```
x = 1
```

No output

SCT

```
Ex().check_object('x', missing_msg='udm').has_equal_value(incorrect_msg='icm')
```

Result

```
icm
```

No error

13.4.5 Example 5

No PEC

Solution code

```
x = 100
```

Student code

```
x = 100
```

No output

SCT

```
test_object('x', undefined_msg='udm', incorrect_msg='icm')
```

Result

```
Great work!
```

No error

13.4.6 Example 6

No PEC

Solution code

```
x = 100
```

Student code

```
x = 100
```

No output

SCT

```
Ex().check_object('x', missing_msg='udm').has_equal_value(incorrect_msg='icm')
```

Result

```
Great work!
```

No error

13.4.7 Example 7

No PEC

Solution code

```
x = filter(lambda x: x > 0, [1, 1])
```

Student code

```
x = filter(lambda x: x > 0, [0, 1])
```

No output

SCT

```
Ex().check_object('x').has_equal_value()
```

Result

```
Did you correctly define the variable <code>x</code>? Expected <code>&lt;filter␣
↪object at 0x7f792c4b5198&gt;</code>, but got <code>&lt;filter object at␣
↪0x7f792c4b5b00&gt;</code>.
```

No error

13.4.8 Example 8

No PEC

Solution code

```
x = filter(lambda x: x > 0, [1, 1])
```

Student code

```
x = filter(lambda x: x > 0, [1, 1])
```

No output

SCT

```
Ex().check_object('x').has_equal_value()
```

Result

```
Great work!
```

No error

13.4.9 Example 9

PEC

```
import numpy as np
```

Solution code

```
x = (np.array([1, 2]), np.array([3, 4]))
```

Student code

```
x = (np.array([1, 2]), np.array([1, 2]))
```

No output

SCT

```
Ex().check_object('x').has_equal_value()
```

Result

```
Did you correctly define the variable <code>x</code>? Expected <code>(array([1, 2]),␣
↪array([3, 4]))</code>, but got <code>(array([1, 2]), array([1, 2]))</code>.
```

No error

13.4.10 Example 10

PEC

```
import numpy as np
```

Solution code

```
x = (np.array([1, 2]), np.array([3, 4]))
```

Student code

```
x = (np.array([1, 2]), np.array([3, 4]))
```

No output

SCT

```
Ex().check_object('x').has_equal_value()
```

Result

```
Great work!
```

No error

13.4.11 Example 11

No PEC

Solution code

```
x = [4, 5, 6]
```

Student code

```
x = [1, 2, 3]
```

No output

SCT

```
Ex().check_object("x").has_equal_value(func = lambda x,y: len(x) == len(y))
```

Result

```
Great work!
```

No error

13.4.12 Example 12

No PEC

Solution code

```
x = [4, 5, 6]
```

Student code

```
x = [1, 2, 3, 4]
```

No output

SCT

```
Ex().check_object("x").has_equal_value(func = lambda x,y: len(x) == len(y))
```

Result

```
Did you correctly define the variable <code>x</code>? Expected <code>[4, 5, 6]</code>,
↪ but got <code>[1, 2, 3, 4]</code>.
```

No error

13.4.13 Example 13

PEC

```
import numpy as np
```

Solution code

```
arr = np.array([1, 2, 3, 4])
```

Student code

```
arr = 4
```

No output

SCT

```
import numpy; Ex().check_object('arr').is_instance(numpy.ndarray)
```

Result

```
Did you correctly define the variable <code>arr</code>? Is it a ndarray?
```

No error

13.4.14 Example 14

PEC

```
import numpy as np
```

Solution code

```
arr = np.array([1, 2, 3, 4])
```

Student code

```
arr = np.array([1])
```

No output

SCT

```
import numpy; Ex().check_object('arr').is_instance(numpy.ndarray)
```

Result

```
Great work!
```

No error

13.4.15 Example 15

PEC

```
import pandas as pd
```

Solution code

```
df = pd.DataFrame({"a": [1, 2, 3]})
```

No student code

No output

SCT

```
test_data_frame('df', columns=['a'], undefined_msg='udm', not_data_frame_msg='ndfm',␣
↪undefined_cols_msg='ucm', incorrect_msg='icm')
```

Result

```
udm
```

No error

13.4.16 Example 16

PEC

```
import pandas as pd
```

Solution code

```
df = pd.DataFrame({"a": [1, 2, 3]})
```

No student code

No output

SCT

```
test_data_frame('df', columns=None, undefined_msg='udm', not_data_frame_msg='ndfm',␣
↪undefined_cols_msg='ucm', incorrect_msg='icm')
```

Result

```
udm
```

No error

13.4.17 Example 17

PEC

```
import pandas as pd
```

Solution code

```
df = pd.DataFrame({"a": [1, 2, 3]})
```

No student code

No output

SCT

```
import pandas as pd
Ex().check_object('df', missing_msg='udm', expand_msg='').     is_instance(pd.
↪DataFrame, not_instance_msg='ndfm').     check_keys('a', missing_msg='ucm').has_
↪equal_value(incorrect_msg='icm')
```

Result

```
udm
```

No error

13.4.18 Example 18

PEC

```
import pandas as pd
```

Solution code

```
df = pd.DataFrame({"a": [1, 2, 3]})
```

No student code

No output

SCT

```
import pandas as pd
Ex().check_df('df', missing_msg='udm', expand_msg='', not_instance_msg='ndfm').
↪check_keys('a', missing_msg='ucm').has_equal_value(incorrect_msg='icm')
```

Result

```
udm
```

No error

13.4.19 Example 19

PEC

```
import pandas as pd
```

Solution code

```
df = pd.DataFrame({"a": [1, 2, 3]})
```

Student code

```
df = 3
```

No output

SCT

```
test_data_frame('df', columns=['a'], undefined_msg='udm', not_data_frame_msg='ndfm',␣
↪undefined_cols_msg='ucm', incorrect_msg='icm')
```

Result

```
ndfm
```

No error

13.4.20 Example 20

PEC

```
import pandas as pd
```

Solution code

```
df = pd.DataFrame({"a": [1, 2, 3]})
```

Student code

```
df = 3
```

No output

SCT

```
test_data_frame('df', columns=None, undefined_msg='udm', not_data_frame_msg='ndfm',␣
↪undefined_cols_msg='ucm', incorrect_msg='icm')
```

Result

```
ndfm
```

No error

13.4.21 Example 21

PEC

```
import pandas as pd
```

Solution code

```
df = pd.DataFrame({"a": [1, 2, 3]})
```

Student code

```
df = 3
```

No output

SCT

```
import pandas as pd
Ex().check_object('df', missing_msg='udm', expand_msg='').      is_instance(pd.
↪DataFrame, not_instance_msg='ndfm').      check_keys('a', missing_msg='ucm').has_
↪equal_value(incorrect_msg='icm')
```

Result

```
ndfm
```

No error

13.4.22 Example 22

PEC

```
import pandas as pd
```

Solution code

```
df = pd.DataFrame({"a": [1, 2, 3]})
```

Student code

```
df = 3
```

No output

SCT

```
import pandas as pd
Ex().check_df('df', missing_msg='udm', expand_msg='', not_instance_msg='ndfm').
↪check_keys('a', missing_msg='ucm').has_equal_value(incorrect_msg='icm')
```

Result

```
ndfm
```

No error

13.4.23 Example 23

PEC

```
import pandas as pd
```

Solution code

```
df = pd.DataFrame({"a": [1, 2, 3]})
```

Student code

```
df = pd.DataFrame({ "b": [1]})
```

No output

SCT

```
test_data_frame('df', columns=['a'], undefined_msg='udm', not_data_frame_msg='ndfm',␣
↪undefined_cols_msg='ucm', incorrect_msg='icm')
```

Result

```
ucm
```

No error

13.4.24 Example 24

PEC

```
import pandas as pd
```

Solution code

```
df = pd.DataFrame({"a": [1, 2, 3]})
```

Student code

```
df = pd.DataFrame({ "b": [1]})
```

No output

SCT

```
test_data_frame('df', columns=None, undefined_msg='udm', not_data_frame_msg='ndfm',␣
↪undefined_cols_msg='ucm', incorrect_msg='icm')
```

Result

```
ucm
```

No error

13.4.25 Example 25

PEC

```
import pandas as pd
```

Solution code

```
df = pd.DataFrame({"a": [1, 2, 3]})
```

Student code

```
df = pd.DataFrame({ "b": [1]})
```

No output

SCT

```
import pandas as pd
Ex().check_object('df', missing_msg='udm', expand_msg='').      is_instance(pd.
↪DataFrame, not_instance_msg='ndfm').      check_keys('a', missing_msg='ucm').has_
↪equal_value(incorrect_msg='icm')
```

Result

```
ucm
```

No error

13.4.26 Example 26

PEC

```
import pandas as pd
```

Solution code

```
df = pd.DataFrame({"a": [1, 2, 3]})
```

Student code

```
df = pd.DataFrame({ "b": [1]})
```

No output

SCT

```
import pandas as pd
Ex().check_df('df', missing_msg='udm', expand_msg='', not_instance_msg='ndfm').
↪check_keys('a', missing_msg='ucm').has_equal_value(incorrect_msg='icm')
```

Result

```
ucm
```

No error

13.4.27 Example 27

PEC

```
import pandas as pd
```

Solution code

```
df = pd.DataFrame({"a": [1, 2, 3]})
```

Student code

```
df = pd.DataFrame({ "a": [1]})
```

No output

SCT

```
test_data_frame('df', columns=['a'], undefined_msg='udm', not_data_frame_msg='ndfm',␣
↪undefined_cols_msg='ucm', incorrect_msg='icm')
```

Result

```
icm
```

No error

13.4.28 Example 28

PEC

```
import pandas as pd
```

Solution code

```
df = pd.DataFrame({"a": [1, 2, 3]})
```

Student code

```
df = pd.DataFrame({ "a": [1]})
```

No output

SCT

```
test_data_frame('df', columns=None, undefined_msg='udm', not_data_frame_msg='ndfm',␣
↪undefined_cols_msg='ucm', incorrect_msg='icm')
```

Result

```
icm
```

No error

13.4.29 Example 29

PEC

```
import pandas as pd
```

Solution code

```
df = pd.DataFrame({"a": [1, 2, 3]})
```

Student code

```
df = pd.DataFrame({ "a": [1]})
```

No output

SCT

```
import pandas as pd
Ex().check_object('df', missing_msg='udm', expand_msg='').      is_instance(pd.
↪DataFrame, not_instance_msg='ndfm').      check_keys('a', missing_msg='ucm').has_
↪equal_value(incorrect_msg='icm')
```

Result

```
icm
```

No error

13.4.30 Example 30

PEC

```
import pandas as pd
```

Solution code

```
df = pd.DataFrame({"a": [1, 2, 3]})
```

Student code

```
df = pd.DataFrame({ "a": [1]})
```

No output

SCT

```
import pandas as pd
Ex().check_df('df', missing_msg='udm', expand_msg='', not_instance_msg='ndfm').
↪check_keys('a', missing_msg='ucm').has_equal_value(incorrect_msg='icm')
```

Result

```
icm
```

No error

13.4.31 Example 31

PEC

```
import pandas as pd
```

Solution code

```
df = pd.DataFrame({"a": [1, 2, 3]})
```

Student code

```
df = pd.DataFrame({ "a": [1, 2, 3] })
```

No output

SCT

```
test_data_frame('df', columns=['a'], undefined_msg='udm', not_data_frame_msg='ndfm',␣
↪undefined_cols_msg='ucm', incorrect_msg='icm')
```

Result

```
Great work!
```

No error

13.4.32 Example 32

PEC

```
import pandas as pd
```

Solution code

```
df = pd.DataFrame({"a": [1, 2, 3]})
```

Student code

```
df = pd.DataFrame({ "a": [1, 2, 3] })
```

No output

SCT

```
test_data_frame('df', columns=None, undefined_msg='udm', not_data_frame_msg='ndfm',␣
↪undefined_cols_msg='ucm', incorrect_msg='icm')
```

Result

```
Great work!
```

No error

13.4.33 Example 33

PEC

```
import pandas as pd
```

Solution code

```
df = pd.DataFrame({"a": [1, 2, 3]})
```

Student code

```
df = pd.DataFrame({ "a": [1, 2, 3] })
```

No output

SCT

```
import pandas as pd
Ex().check_object('df', missing_msg='udm', expand_msg='').      is_instance(pd.
↪DataFrame, not_instance_msg='ndfm').      check_keys('a', missing_msg='ucm').has_
↪equal_value(incorrect_msg='icm')
```

Result

```
Great work!
```

No error

13.4.34 Example 34

PEC

```
import pandas as pd
```

Solution code

```
df = pd.DataFrame({"a": [1, 2, 3]})
```

Student code

```
df = pd.DataFrame({ "a": [1, 2, 3] })
```

No output

SCT

```
import pandas as pd
Ex().check_df('df', missing_msg='udm', expand_msg='', not_instance_msg='ndfm').
↪check_keys('a', missing_msg='ucm').has_equal_value(incorrect_msg='icm')
```

Result

```
Great work!
```

No error

13.4.35 Example 35

PEC

```
import pandas as pd
```

Solution code

```
df = pd.DataFrame({"a": [1, 2, 3]})
```

Student code

```
df = pd.DataFrame({ "a": [1, 2, 3], "b": [3, 4, 5] })
```

No output

SCT

```
test_data_frame('df', columns=['a'], undefined_msg='udm', not_data_frame_msg='ndfm',␣
↪undefined_cols_msg='ucm', incorrect_msg='icm')
```

Result

```
Great work!
```

No error

13.4.36 Example 36

PEC

```
import pandas as pd
```

Solution code

```
df = pd.DataFrame({"a": [1, 2, 3]})
```

Student code

```
df = pd.DataFrame({ "a": [1, 2, 3], "b": [3, 4, 5] })
```

No output

SCT

```
test_data_frame('df', columns=None, undefined_msg='udm', not_data_frame_msg='ndfm',␣
↪undefined_cols_msg='ucm', incorrect_msg='icm')
```

Result

```
Great work!
```

No error

13.4.37 Example 37

PEC

```
import pandas as pd
```

Solution code

```
df = pd.DataFrame({"a": [1, 2, 3]})
```

Student code

```
df = pd.DataFrame({ "a": [1, 2, 3], "b": [3, 4, 5] })
```

No output

SCT

```
import pandas as pd
Ex().check_object('df', missing_msg='udm', expand_msg='').      is_instance(pd.
↪DataFrame, not_instance_msg='ndfm').      check_keys('a', missing_msg='ucm').has_
↪equal_value(incorrect_msg='icm')
```

Result

```
Great work!
```

No error

13.4.38 Example 38

PEC

```
import pandas as pd
```

Solution code

```
df = pd.DataFrame({"a": [1, 2, 3]})
```

Student code

```
df = pd.DataFrame({ "a": [1, 2, 3], "b": [3, 4, 5] })
```

No output

SCT

```
import pandas as pd
Ex().check_df('df', missing_msg='udm', expand_msg='', not_instance_msg='ndfm').
↪check_keys('a', missing_msg='ucm').has_equal_value(incorrect_msg='icm')
```

Result

```
Great work!
```

No error

13.4.39 Example 39

No PEC

Solution code

```
x = {"a": 2}
```

Student code

```
x = {}
```

No output

SCT

```
Ex().check_object("x").check_keys("a").has_equal_value()
```

Result

```
Did you correctly define the variable <code>x</code>? There is no key <code>'a'</code>
↪.
```

No error

13.4.40 Example 40

No PEC

Solution code

```
x = {"a": 2}
```

Student code

```
x = {"b": 3}
```

No output

SCT

```
Ex().check_object("x").check_keys("a").has_equal_value()
```

Result

```
Did you correctly define the variable <code>x</code>? There is no key <code>'a'</code>
↪.
```

No error

13.4.41 Example 41

No PEC

Solution code

```
x = {"a": 2}
```

Student code

```
x = {"a": 3}
```

No output

SCT

```
Ex().check_object("x").check_keys("a").has_equal_value()
```

Result

```
Did you correctly define the variable <code>x</code>? Did you correctly set the key
↪<code>'a'</code>? Expected <code>2</code>, but got <code>3</code>.
```

No error

13.4.42 Example 42

No PEC

Solution code

```
x = {"a": 2}
```

Student code

```
x = {"a": 2}
```

No output

SCT

```
Ex().check_object("x").check_keys("a").has_equal_value()
```

Result

```
Great work!
```

No error

13.4.43 Example 43

No PEC

Solution code

```
x = {"a": 2}
```

Student code

```
x = {"a": 2, "b": 3}
```

No output

SCT

```
Ex().check_object("x").check_keys("a").has_equal_value()
```

Result

```
Great work!
```

No error

13.4.44 Example 44

PEC

```
import pandas as pd
users = pd.read_csv('https://s3.amazonaws.com/assets.datacamp.com/production/course_
↪1650/datasets/users.csv')
```

Solution code

```
pivot = users.pivot(index='weekday', columns='city')
```

Student code

```
pivot = users.pivot(index='weekday', columns='city')
```

No output

SCT

```
Ex().test_data_frame('pivot')
```

Result

```
Great work!
```

No error

13.4.45 Example 45

PEC

```
import pandas as pd
users = pd.read_csv('https://s3.amazonaws.com/assets.datacamp.com/production/course_
↪1650/datasets/users.csv')
```

Solution code

```
pivot = users.pivot(index='weekday', columns='city')
```

Student code

```
pivot = users.pivot(index='weekday', columns='city')
```

No output

SCT

```
Ex().check_df('pivot').check_keys(('visitors', 'Austin')).has_equal_value()
```

Result

```
Great work!
```

No error

13.4.46 Example 46

PEC

```
import scipy.io; from urllib.request import urlretrieve; urlretrieve('https://s3.
↪amazonaws.com/assets.datacamp.com/production/course_998/datasets/ja_data2.mat',
↪'albeck_gene_expression.mat')
```

Solution code

```
mat = scipy.io.loadmat('albeck_gene_expression.mat')
```

Student code

```
mat = scipy.io.loadmat('albeck_gene_expression.mat')
```

No output

SCT

```
Ex().check_object('mat').has_equal_value()
```

Result

```
Great work!
```

No error

13.4.47 Example 47

No PEC

Solution code

```
if True:
    a = 10

if False:
    b = 20
else:
    c = 30

for i in range(2):
    d = 40
```

(continues on next page)

(continued from previous page)

```
x = 2
while x > 0:
    e = 50
    x -= 1

try:
    f = 60
except:
    pass

try:
    g = 70
except:
    pass
finally:
    h = 80

# 2 assignments
i = 90
if True:
    i = 90
```

Student code

```
if True:
    a = 1

if False:
    b = 2
else:
    c = 3

for i in range(2):
    d = 4

x = 2
while x > 0:
    e = 5
    x -= 1

try:
    f = 6
except:
    pass

try:
    g = 7
except:
    pass
finally:
    h = 8

# 2 assignments
i = 9
if True:
    i = 9
```

No output

SCT

```
Ex().check_object("a").has_equal_value()
```

Result

```
Did you correctly define the variable <code>a</code>? Expected <code>10</code>, but␣
↪got <code>1</code>.
```

No error

13.4.48 Example 48

No PEC

Solution code

```
if True:
    a = 10

if False:
    b = 20
else:
    c = 30

for i in range(2):
    d = 40

x = 2
while x > 0:
    e = 50
    x -= 1

try:
    f = 60
except:
    pass

try:
    g = 70
except:
    pass
finally:
    h = 80

# 2 assignments
i = 90
if True:
    i = 90
```

Student code

```
if True:
    a = 1
```

(continues on next page)

(continued from previous page)

```
if False:
    b = 2
else:
    c = 3

for i in range(2):
    d = 4

x = 2
while x > 0:
    e = 5
    x -= 1

try:
    f = 6
except:
    pass

try:
    g = 7
except:
    pass
finally:
    h = 8

# 2 assignments
i = 9
if True:
    i = 9
```

No output

SCT

```
Ex().check_object("c").has_equal_value()
```

Result

```
Did you correctly define the variable <code>c</code>? Expected <code>30</code>, but␣
↪got <code>3</code>.
```

No error

13.4.49 Example 49

No PEC

Solution code

```
if True:
    a = 10

if False:
    b = 20
else:
    c = 30
```

(continues on next page)

(continued from previous page)

```
for i in range(2):
    d = 40

x = 2
while x > 0:
    e = 50
    x -= 1

try:
    f = 60
except:
    pass

try:
    g = 70
except:
    pass
finally:
    h = 80

# 2 assignments
i = 90
if True:
    i = 90
```

Student code

```
if True:
    a = 1

if False:
    b = 2
else:
    c = 3

for i in range(2):
    d = 4

x = 2
while x > 0:
    e = 5
    x -= 1

try:
    f = 6
except:
    pass

try:
    g = 7
except:
    pass
finally:
    h = 8

# 2 assignments
```

(continues on next page)

(continued from previous page)

```
i = 9
if True:
    i = 9
```

No output

SCT

```
Ex().check_object("d").has_equal_value()
```

Result

```
Did you correctly define the variable <code>d</code>? Expected <code>40</code>, but
↪got <code>4</code>.
```

No error

13.4.50 Example 50

No PEC

Solution code

```
if True:
    a = 10

if False:
    b = 20
else:
    c = 30

for i in range(2):
    d = 40

x = 2
while x > 0:
    e = 50
    x -= 1

try:
    f = 60
except:
    pass

try:
    g = 70
except:
    pass
finally:
    h = 80

# 2 assignments
i = 90
if True:
    i = 90
```

Student code

```
if True:
    a = 1

if False:
    b = 2
else:
    c = 3

for i in range(2):
    d = 4

x = 2
while x > 0:
    e = 5
    x -= 1

try:
    f = 6
except:
    pass

try:
    g = 7
except:
    pass
finally:
    h = 8

# 2 assignments
i = 9
if True:
    i = 9
```

No output

SCT

```
Ex().check_object("e").has_equal_value()
```

Result

```
Did you correctly define the variable <code>e</code>? Expected <code>50</code>, but␣
↪got <code>5</code>.
```

No error

13.4.51 Example 51

No PEC

Solution code

```
if True:
    a = 10

if False:
    b = 20
```

(continues on next page)

(continued from previous page)

```
else:
    c = 30

for i in range(2):
    d = 40

x = 2
while x > 0:
    e = 50
    x -= 1

try:
    f = 60
except:
    pass

try:
    g = 70
except:
    pass
finally:
    h = 80

# 2 assignments
i = 90
if True:
    i = 90
```

Student code

```
if True:
    a = 1

if False:
    b = 2
else:
    c = 3

for i in range(2):
    d = 4

x = 2
while x > 0:
    e = 5
    x -= 1

try:
    f = 6
except:
    pass

try:
    g = 7
except:
    pass
finally:
    h = 8
```

(continues on next page)

(continued from previous page)

```
# 2 assignments
i = 9
if True:
    i = 9
```

No output

SCT

```
Ex().check_object("f").has_equal_value()
```

Result

```
Did you correctly define the variable <code>f</code>? Expected <code>60</code>, but
↪got <code>6</code>.
```

No error

13.4.52 Example 52

No PEC

Solution code

```
if True:
    a = 10

if False:
    b = 20
else:
    c = 30

for i in range(2):
    d = 40

x = 2
while x > 0:
    e = 50
    x -= 1

try:
    f = 60
except:
    pass

try:
    g = 70
except:
    pass
finally:
    h = 80

# 2 assignments
i = 90
```

(continues on next page)

(continued from previous page)

```
if True:
    i = 90
```

Student code

```
if True:
    a = 1

if False:
    b = 2
else:
    c = 3

for i in range(2):
    d = 4

x = 2
while x > 0:
    e = 5
    x -= 1

try:
    f = 6
except:
    pass

try:
    g = 7
except:
    pass
finally:
    h = 8

# 2 assignments
i = 9
if True:
    i = 9
```

No output

SCT

```
Ex().check_object("g").has_equal_value()
```

Result

```
Did you correctly define the variable <code>g</code>? Expected <code>70</code>, but␣
↪got <code>7</code>.
```

No error

13.4.53 Example 53

No PEC

Solution code

```
if True:
    a = 10

if False:
    b = 20
else:
    c = 30

for i in range(2):
    d = 40

x = 2
while x > 0:
    e = 50
    x -= 1

try:
    f = 60
except:
    pass

try:
    g = 70
except:
    pass
finally:
    h = 80

# 2 assignments
i = 90
if True:
    i = 90
```

Student code

```
if True:
    a = 1

if False:
    b = 2
else:
    c = 3

for i in range(2):
    d = 4

x = 2
while x > 0:
    e = 5
    x -= 1

try:
    f = 6
except:
    pass

try:
```

(continues on next page)

(continued from previous page)

```
    g = 7
except:
    pass
finally:
    h = 8

# 2 assignments
i = 9
if True:
    i = 9
```

No output

SCT

```
Ex().check_object("h").has_equal_value()
```

Result

```
Did you correctly define the variable <code>h</code>? Expected <code>80</code>, but␣
↪got <code>8</code>.
```

No error

13.4.54 Example 54

No PEC

Solution code

```
if True:
    a = 10

if False:
    b = 20
else:
    c = 30

for i in range(2):
    d = 40

x = 2
while x > 0:
    e = 50
    x -= 1

try:
    f = 60
except:
    pass

try:
    g = 70
except:
    pass
finally:
```

(continues on next page)

(continued from previous page)

```
    h = 80

# 2 assignments
i = 90
if True:
    i = 90
```

Student code

```
if True:
    a = 1

if False:
    b = 2
else:
    c = 3

for i in range(2):
    d = 4

x = 2
while x > 0:
    e = 5
    x -= 1

try:
    f = 6
except:
    pass

try:
    g = 7
except:
    pass
finally:
    h = 8

# 2 assignments
i = 9
if True:
    i = 9
```

No output

SCT

```
Ex().check_object("i").has_equal_value()
```

Result

```
Did you correctly define the variable <code>i</code>? Expected <code>90</code>, but␣
↪got <code>9</code>.
```

No error

13.4.55 Example 55

No PEC

Solution code

```
import pandas as pd
df = pd.DataFrame({"a": [1, 2, 3], "b": [4, 5, 6]})
df.columns = ["c", "d"]

df2 = pd.DataFrame({"a": [1, 2, 3], "b": [4, 5, 6]})
df2.columns = ["c", "d"]
```

Student code

```
import pandas as pd
df = pd.DataFrame({"a": [1, 2, 3], "b": [4, 5, 6]})
df.columns = ["c", "d"]

df2 = pd.DataFrame({"a": [1, 2, 3], "b": [4, 5, 6]})
df2.columns = ["e", "f"]
```

No output

SCT

```
Ex().check_object('df').has_equal_value()
```

Result

```
Great work!
```

No error

13.4.56 Example 56

No PEC

Solution code

```
import pandas as pd
df = pd.DataFrame({"a": [1, 2, 3], "b": [4, 5, 6]})
df.columns = ["c", "d"]

df2 = pd.DataFrame({"a": [1, 2, 3], "b": [4, 5, 6]})
df2.columns = ["c", "d"]
```

Student code

```
import pandas as pd
df = pd.DataFrame({"a": [1, 2, 3], "b": [4, 5, 6]})
df.columns = ["c", "d"]

df2 = pd.DataFrame({"a": [1, 2, 3], "b": [4, 5, 6]})
df2.columns = ["e", "f"]
```

No output

SCT

```
Ex().check_object('df2').has_equal_value()
```

Result

```
Did you correctly define the variable <code>df2</code>? Expected something different.
```

No error

13.5 test_test_object_accessed.py

13.5.1 Example 1

No PEC

Solution code

```
# not used
```

Student code

```
import numpy as np
import math as m
arr = np.array([1, 2, 3])
x = arr.shape
print(arr.data)
print(m.e)
```

Student output

```
<memory at 0x7f792c4c8288>
2.718281828459045
```

SCT

```
test_object_accessed("arr")
```

Result

```
Great work!
```

No error

13.5.2 Example 2

No PEC

Solution code

```
# not used
```

Student code

```
import numpy as np
import math as m
arr = np.array([1, 2, 3])
x = arr.shape
print(arr.data)
print(m.e)
```

Student output

```
<memory at 0x7f792c4c8288>
2.718281828459045
```

SCT

```
test_object_accessed("ar")
```

Result

```
Have you accessed <code>ar</code>?
```

No error

13.5.3 Example 3

No PEC

Solution code

```
# not used
```

Student code

```
import numpy as np
import math as m
arr = np.array([1, 2, 3])
x = arr.shape
print(arr.data)
print(m.e)
```

Student output

```
<memory at 0x7f793e8d9e88>
2.718281828459045
```

SCT

```
test_object_accessed("arr", times=2)
```

Result

```
Great work!
```

No error

13.5.4 Example 4

No PEC

Solution code

```
# not used
```

Student code

```
import numpy as np
import math as m
arr = np.array([1, 2, 3])
x = arr.shape
print(arr.data)
print(m.e)
```

Student output

```
<memory at 0x7f793e8d9e88>
2.718281828459045
```

SCT

```
test_object_accessed("arr", times=3)
```

Result

```
Have you accessed <code>arr</code> at least three times?
```

No error

13.5.5 Example 5

No PEC

Solution code

```
# not used
```

Student code

```
import numpy as np
import math as m
arr = np.array([1, 2, 3])
x = arr.shape
print(arr.data)
print(m.e)
```

Student output

```
<memory at 0x7f793e8d9f48>
2.718281828459045
```

SCT

```
test_object_accessed("arr", times=3, not_accessed_msg="silly")
```

Result

```
silly
```

No error

13.5.6 Example 6

No PEC

Solution code

```
# not used
```

Student code

```
import numpy as np
import math as m
arr = np.array([1, 2, 3])
x = arr.shape
print(arr.data)
print(m.e)
```

Student output

```
<memory at 0x7f793e8d9f48>
2.718281828459045
```

SCT

```
test_object_accessed("arr.shape")
```

Result

```
Great work!
```

No error

13.5.7 Example 7

No PEC

Solution code

```
# not used
```

Student code

```
import numpy as np
import math as m
arr = np.array([1, 2, 3])
x = arr.shape
print(arr.data)
print(m.e)
```

Student output

```
<memory at 0x7f792c4c84c8>
2.718281828459045
```

SCT

```
test_object_accessed("arr.shape", times=2)
```

Result

```
Have you accessed <code>arr.shape</code> at least twice?
```

No error

13.5.8 Example 8

No PEC

Solution code

```
# not used
```

Student code

```
import numpy as np
import math as m
arr = np.array([1, 2, 3])
x = arr.shape
print(arr.data)
print(m.e)
```

Student output

```
<memory at 0x7f792c4c81c8>
2.718281828459045
```

SCT

```
test_object_accessed("arr.shape", times=2, not_accessed_msg="silly")
```

Result

```
silly
```

No error

13.5.9 Example 9

No PEC

Solution code

```
# not used
```

Student code

```
import numpy as np
import math as m
arr = np.array([1, 2, 3])
x = arr.shape
print(arr.data)
print(m.e)
```

Student output

```
<memory at 0x7f792cefa1c8>
2.718281828459045
```

SCT

```
test_object_accessed("arr.dtype")
```

Result

```
Have you accessed <code>arr.dtype</code>?
```

No error

13.5.10 Example 10

No PEC

Solution code

```
# not used
```

Student code

```
import numpy as np
import math as m
arr = np.array([1, 2, 3])
x = arr.shape
print(arr.data)
print(m.e)
```

Student output

```
<memory at 0x7f792c4c8288>
2.718281828459045
```

SCT

```
test_object_accessed("arr.dtype", not_accessed_msg="silly")
```

Result

```
silly
```

No error

13.5.11 Example 11

No PEC

Solution code

```
# not used
```

Student code

```
import numpy as np
import math as m
arr = np.array([1, 2, 3])
x = arr.shape
print(arr.data)
print(m.e)
```

Student output

```
<memory at 0x7f792cefa1c8>
2.718281828459045
```

SCT

```
test_object_accessed("math.e")
```

Result

```
Great work!
```

No error

13.5.12 Example 12

No PEC

Solution code

```
# not used
```

Student code

```
import numpy as np
import math as m
arr = np.array([1, 2, 3])
x = arr.shape
print(arr.data)
print(m.e)
```

Student output

```
<memory at 0x7f792cefa1c8>
2.718281828459045
```

SCT

```
test_object_accessed("math.pi")
```

Result

```
Have you accessed <code>m.pi</code>?
```

No error

13.5.13 Example 13

No PEC

Solution code

```
# not used
```

Student code

```
import numpy as np
import math as m
arr = np.array([1, 2, 3])
x = arr.shape
print(arr.data)
print(m.e)
```

Student output

```
<memory at 0x7f792cefaf48>
2.718281828459045
```

SCT

```
test_object_accessed("math.pi", not_accessed_msg="silly")
```

Result

```
silly
```

No error

13.6 test_check_logic.py

13.6.1 Example 1

No PEC

Solution code

```
a = 1
```

Student code

```
a = 1
```

No output

SCT

```
Ex().check_not(has_code('a'), msg = 'x')
```

Result

```
x
```

No error

13.6.2 Example 2

No PEC

Solution code

```
a = 1
```

Student code

```
a = 1
```

No output

SCT

```
Ex().check_not(has_code('a'), has_code('b'), msg = 'x')
```

Result

```
x
```

No error

13.6.3 Example 3

No PEC

Solution code

```
a = 1
```

Student code

```
a = 1
```

No output

SCT

```
Ex().check_not(has_code('b'), msg = 'x')
```

Result

```
Great work!
```

No error

13.6.4 Example 4

No PEC

Solution code

```
a = 1
```

Student code

```
a = 1
```

No output

SCT

```
Ex().check_not(has_code('b'), has_code('c'), msg = 'x')
```

Result

```
Great work!
```

No error

13.6.5 Example 5

No PEC

Solution code

```
a = 1
```

Student code

```
a = 1
```

No output

SCT

```
Ex().check_not(check_object('a').has_equal_value(override=1), msg = 'x')
```

Result

```
x
```

No error

13.6.6 Example 6

No PEC

Solution code

```
a = 1
```

Student code

```
a = 1
```

No output

SCT

```
Ex().check_not(check_object('a').has_equal_value(override=2), msg = 'x')
```

Result

```
Great work!
```

No error

13.6.7 Example 7

No PEC

No solution code

Student code

```
'a'
```

No output

SCT

```
Ex().check_or(has_code('a'))
```

Result

```
Great work!
```

No error

13.6.8 Example 8

No PEC

No solution code

Student code

```
'a'
```

No output

SCT

```
Ex().check_or(has_code('a'), has_code('b'))
```

Result

```
Great work!
```

No error

13.6.9 Example 9

No PEC

No solution code

Student code

```
'a'
```

No output

SCT

```
Ex().check_or(has_code('b'))
```

Result

```
Could not find the correct pattern in your code.
```

No error

13.6.10 Example 10

No PEC

No solution code

Student code

```
'a'
```

No output

SCT

```
Ex().check_or(has_code('b'), has_code('c'))
```

Result

```
Could not find the correct pattern in your code.
```

No error

13.6.11 Example 11

No PEC

No solution code

Student code

```
'a'
```

No output

SCT

```
test_or(lambda: test_student_typed('a'))
```

Result

```
Great work!
```

No error

13.6.12 Example 12

No PEC

No solution code

Student code

```
'a'
```

No output

SCT

```
test_or(test_student_typed('a'))
```

Result

```
Great work!
```

No error

13.6.13 Example 13

No PEC

No solution code

Student code

```
'a'
```

No output

SCT

```
Ex().test_or(has_code('a'))
```

Result

```
Great work!
```

No error

13.6.14 Example 14

No PEC

No solution code

Student code

```
'a'
```

No output

SCT

```
Ex().test_or(test_student_typed('a'))
```

Result

```
Great work!
```

No error

13.6.15 Example 15

No PEC

No solution code

Student code

```
'a'
```

No output

SCT

```
test_or(lambda: test_student_typed('a'), lambda: test_student_typed('b'))
```

Result

```
Great work!
```

No error

13.6.16 Example 16

No PEC

No solution code

Student code

```
'a'
```

No output

SCT

```
test_or(test_student_typed('a'), test_student_typed('b'))
```

Result

```
Great work!
```

No error

13.6.17 Example 17

No PEC

No solution code

Student code

```
'a'
```

No output

SCT

```
Ex().test_or(has_code('a'), has_code('b'))
```

Result

```
Great work!
```

No error

13.6.18 Example 18

No PEC

No solution code

Student code

```
'a'
```

No output

SCT

```
Ex().test_or(test_student_typed('a'), test_student_typed('b'))
```

Result

```
Great work!
```

No error

13.6.19 Example 19

No PEC

No solution code

Student code

```
'a'
```

No output

SCT

```
test_or(lambda: test_student_typed('a'), lambda: test_student_typed('b'))
```

Result

```
Great work!
```

No error

13.6.20 Example 20

No PEC

No solution code

Student code

```
'a'
```

No output

SCT

```
test_or(test_student_typed('a'), test_student_typed('b'))
```

Result

```
Great work!
```

No error

13.6.21 Example 21

No PEC

No solution code

Student code

```
'a'
```

No output

SCT

```
Ex().test_or(has_code('a'), has_code('b'))
```

Result

```
Great work!
```

No error

13.6.22 Example 22

No PEC

No solution code

Student code

```
'a'
```

No output

SCT

```
Ex().test_or(test_student_typed('a'), test_student_typed('b'))
```

Result

```
Great work!
```

No error

13.6.23 Example 23

No PEC

No solution code

Student code

```
'a'
```

No output

SCT

```
test_or(lambda: test_student_typed('b'))
```

Result

```
Could not find the correct pattern in your code.
```

No error

13.6.24 Example 24

No PEC

No solution code

Student code

```
'a'
```

No output

SCT

```
test_or(test_student_typed('b'))
```

Result

```
Could not find the correct pattern in your code.
```

No error

13.6.25 Example 25

No PEC

No solution code

Student code

```
'a'
```

No output

SCT

```
Ex().test_or(has_code('b'))
```

Result

```
Could not find the correct pattern in your code.
```

No error

13.6.26 Example 26

No PEC

No solution code

Student code

```
'a'
```

No output

SCT

```
Ex().test_or(test_student_typed('b'))
```

Result

```
Could not find the correct pattern in your code.
```

No error

13.6.27 Example 27

No PEC

No solution code

Student code

```
'a'
```

No output

SCT

```
test_or(lambda: test_student_typed('b'), lambda: test_student_typed('c'))
```

Result

```
Could not find the correct pattern in your code.
```

No error

13.6.28 Example 28

No PEC

No solution code

Student code

```
'a'
```

No output

SCT

```
test_or(test_student_typed('b'), test_student_typed('c'))
```

Result

```
Could not find the correct pattern in your code.
```

No error

13.6.29 Example 29

No PEC

No solution code

Student code

```
'a'
```

No output

SCT

```
Ex().test_or(has_code('b'), has_code('c'))
```

Result

```
Could not find the correct pattern in your code.
```

No error

13.6.30 Example 30

No PEC

No solution code

Student code

```
'a'
```

No output

SCT

```
Ex().test_or(test_student_typed('b'), test_student_typed('c'))
```

Result

```
Could not find the correct pattern in your code.
```

No error

13.6.31 Example 31

No PEC

No solution code

Student code

```
'def'
```

No output

SCT

```
Ex().check_or(has_code('a', not_typed_msg = 'a'), check_or(has_code('b'), has_code('c
↪')))
```

Result

```
a
```

No error

13.6.32 Example 32

No PEC

No solution code

Student code

```
'def'
```

No output

SCT

```
Ex().test_or(has_code('a', not_typed_msg = 'a'), F().test_or(has_code('b'), has_code(
↪'c')))
```

Result

```
a
```

No error

13.6.33 Example 33

No PEC

No solution code

Student code

```
'def'
```

No output

SCT

```
test_or(test_student_typed('a', not_typed_msg = 'a'), test_or(test_student_typed('b'),
↪ test_student_typed('c')))
```

Result

```
a
```

No error

13.6.34 Example 34

No PEC

No solution code

Student code

```
'a'
```

No output

SCT

```
Ex().check_correct(has_code('a'), has_code('b'))
```

Result

```
Great work!
```

No error

13.6.35 Example 35

No PEC

No solution code

Student code

```
'a'
```

No output

SCT

```
Ex().check_correct(has_code('a'), has_code('c'))
```

Result

```
Great work!
```

No error

13.6.36 Example 36

No PEC

No solution code

Student code

```
'a'
```

No output

SCT

```
Ex().check_correct(has_code('b'), has_code('c', not_typed_msg='x'))
```

Result

```
x
```

No error

13.6.37 Example 37

No PEC

No solution code

Student code

```
'a'
```

No output

SCT

```
Ex().check_correct(has_code('b', not_typed_msg='x'), has_code('a'))
```

Result

```
x
```

No error

13.6.38 Example 38

No PEC

No solution code

Student code

```
'a'
```

No output

SCT

```
Ex().check_correct(has_code('a'), has_code('b'))
```

Result

```
Could not find the correct pattern in your code.
```

No error

13.6.39 Example 39

No PEC

No solution code

Student code

```
'a'
```

No output

SCT

```
Ex().check_correct(has_code('a'), has_code('c'))
```

Result

```
Could not find the correct pattern in your code.
```

No error

13.6.40 Example 40

No PEC

No solution code

Student code

```
'a'
```

No output

SCT

```
Ex().check_correct(has_code('b'), has_code('c', not_typed_msg='x'))
```

Result

```
x
```

No error

13.6.41 Example 41

No PEC

No solution code

Student code

```
'a'
```

No output

SCT

```
Ex().check_correct(has_code('b', not_typed_msg='x'), has_code('a'))
```

Result

```
x
```

No error

13.6.42 Example 42

No PEC

No solution code

Student code

```
'a'
```

No output

SCT

```
test_correct(lambda: test_student_typed('a'), lambda: test_student_typed('b'))
```

Result

```
Great work!
```

No error

13.6.43 Example 43

No PEC

No solution code

Student code

```
'a'
```

No output

SCT

```
test_correct(test_student_typed('a'), test_student_typed('b'))
```

Result

```
Great work!
```

No error

13.6.44 Example 44

No PEC

No solution code

Student code

```
'a'
```

No output

SCT

```
Ex().test_correct(has_code('a'), has_code('b'))
```

Result

```
Great work!
```

No error

13.6.45 Example 45

No PEC

No solution code

Student code

```
'a'
```

No output

SCT

```
Ex().test_correct(test_student_typed('a'), test_student_typed('b'))
```

Result

```
Great work!
```

No error

13.6.46 Example 46

No PEC

No solution code

Student code

```
'a'
```

No output

SCT

```
test_correct(lambda: test_student_typed('a'), lambda: test_student_typed('c'))
```

Result

```
Great work!
```

No error

13.6.47 Example 47

No PEC

No solution code

Student code

```
'a'
```

No output

SCT

```
test_correct(test_student_typed('a'), test_student_typed('c'))
```

Result

```
Great work!
```

No error

13.6.48 Example 48

No PEC

No solution code

Student code

```
'a'
```

No output

SCT

```
Ex().test_correct(has_code('a'), has_code('c'))
```

Result

```
Great work!
```

No error

13.6.49 Example 49

No PEC

No solution code

Student code

```
'a'
```

No output

SCT

```
Ex().test_correct(test_student_typed('a'), test_student_typed('c'))
```

Result

```
Great work!
```

No error

13.6.50 Example 50

No PEC

No solution code

Student code

```
'a'
```

No output

SCT

```
test_correct(lambda: test_student_typed('b'), lambda: test_student_typed('c', not_
↪typed_msg='x'))
```

Result

```
x
```

No error

13.6.51 Example 51

No PEC

No solution code

Student code

```
'a'
```

No output

SCT

```
test_correct(test_student_typed('b'), test_student_typed('c', not_typed_msg='x'))
```

Result

```
x
```

No error

13.6.52 Example 52

No PEC

No solution code

Student code

```
'a'
```

No output

SCT

```
Ex().test_correct(has_code('b'), has_code('c', not_typed_msg='x'))
```

Result

```
x
```

No error

13.6.53 Example 53

No PEC

No solution code

Student code

```
'a'
```

No output

SCT

```
Ex().test_correct(test_student_typed('b'), test_student_typed('c', not_typed_msg='x'))
```

Result

```
x
```

No error

13.6.54 Example 54

No PEC

No solution code

Student code

```
'a'
```

No output

SCT

```
test_correct(lambda: test_student_typed('b', not_typed_msg='x'), lambda: test_student_
↪typed('a'))
```

Result

```
x
```

No error

13.6.55 Example 55

No PEC

No solution code

Student code

```
'a'
```

No output

SCT

```
test_correct(test_student_typed('b', not_typed_msg='x'), test_student_typed('a'))
```

Result

```
x
```

No error

13.6.56 Example 56

No PEC

No solution code

Student code

```
'a'
```

No output

SCT

```
Ex().test_correct(has_code('b', not_typed_msg='x'), has_code('a'))
```

Result

```
x
```

No error

13.6.57 Example 57

No PEC

No solution code

Student code

```
'a'
```

No output

SCT

```
Ex().test_correct(test_student_typed('b', not_typed_msg='x'), test_student_typed('a'))
```

Result

```
x
```

No error

13.6.58 Example 58

No PEC

No solution code

Student code

```
'def'
```

No output

SCT

```
Ex().check_correct(has_code('a'), check_correct(has_code('b'), has_code('c', not_
↪typed_msg = 'c')))
```

Result

```
c
```

No error

13.6.59 Example 59

No PEC

No solution code

Student code

```
'def'
```

No output

SCT

```
Ex().test_correct(has_code('a'), F().test_correct(has_code('b'), has_code('c', not_
↪typed_msg = 'c')))
```

Result

```
c
```

No error

13.6.60 Example 60

No PEC

No solution code

Student code

```
'def'
```

No output

SCT

```
test_correct(test_student_typed('a'), test_correct(test_student_typed('b'), test_
↪student_typed('c', not_typed_msg = 'c')))
```

Result

```
c
```

No error

13.7 test_test_exercise.py

13.7.1 Example 1

PEC

```
#no pec
```

Solution code

```
x = 4
```

Student code

```
x = 4
```

No output

SCT

```
test_object("x")
success_msg("nice")
```

Result

```
nice
```

No error

13.7.2 Example 2

PEC

```
#no pec
```

Solution code

```
x = 6
```

Student code

```
x = 4
```

No output

SCT

```
test_object("x")
success_msg("nice")
```

Result

```
Did you correctly define the variable <code>x</code>? Expected <code>6</code>, but␣
↪got <code>4</code>.
```

No error

13.7.3 Example 3

PEC

```
#no pec
```

Solution code

```
x = 6
```

Student code

```
x = y
```

No output

SCT

```
# no sct
```

Result

```
Your code generated an error. Fix it and try again!
```

Error

```
name 'y' is not defined
```

13.7.4 Example 4

PEC

```
# no pec
```

Solution code

```
x = 6
```

Student code

```
print "yolo"
```

No output

SCT

```
test_object("x")
```

Result

```
Your code can not be executed due to a syntax error:<br><code>Missing parentheses in␣
↪call to 'print' (script.py, line 1).</code>
```

Error

```
Missing parentheses in call to 'print' (<string>, line 1)
```

13.7.5 Example 5

PEC

```
# no pec
```

Solution code

```
x = 6
```

Student code

```
print("yolo")
```

No output

SCT

```
test_object("x")
```

Result

```
Your code could not be parsed due to an error in the indentation:<br><code>unexpected␣
↪indent (script.py, line 1).</code>
```

Error

```
unexpected indent (<string>, line 1)
```

13.7.6 Example 6

PEC

```
# no pec
```

Solution code

```
x = 6
```

No student code

No output

SCT

```
test_object("x")
```

Result

```
Did you define the variable <code>x</code> without errors?
```

No error

13.8 test_test_with.py

13.8.1 Example 1

PEC

```
from urllib.request import urlretrieve; urlretrieve('http://s3.amazonaws.com/assets.
↪datacamp.com/production/course_998/datasets/moby_opens.txt', 'moby_dick.txt')
```

Solution code

```
# Read & print the first 3 lines
with open('moby_dick.txt') as file:
    print(file.readline())
    print(file.readline())
    print(file.readline())

# The rows that you wish to print
I = [0,1,3,5,6,7,8,9]

# Print out these rows
with open('moby_dick.txt') as file:
    for i, row in enumerate(file):
        if i in I:
            print(row)
```

Student code

```
# Read & print the first 3 lines
with open('moby_dick.txt') as file:
    print(file.readline())
    print(file.readline())
    print('test')

# The rows that you wish to print
I = [0,1,3,5,6,7,8,9]

# Print out these rows
with open('moby_dick.txt') as file:
    for i, row in enumerate(file):
        if i in I:
            print(row)
```

Student output

```
CHAPTER 1. Loomings.



test
CHAPTER 1. Loomings.



little or no money in my purse, and nothing particular to interest me on

the world. It is a way I have of driving off the spleen and regulating
```

(continues on next page)

(continued from previous page)

```
the circulation. Whenever I find myself growing grim about the mouth;

whenever it is a damp, drizzly November in my soul; whenever I find

myself involuntarily pausing before coffin warehouses, and bringing up

the rear of every funeral I meet; and especially whenever my hypos get
```

SCT

```
test_with(1, body = lambda: [test_function('print', index = i + 1) for i in range(3)])
```

Result

```
Check your third call of <code>print()</code>. Did you correctly specify the first␣
↪argument? Expected something different.
```

No error

13.8.2 Example 2

PEC

```
from urllib.request import urlretrieve; urlretrieve('http://s3.amazonaws.com/assets.
↪datacamp.com/production/course_998/datasets/moby_opens.txt', 'moby_dick.txt')
```

Solution code

```
# Read & print the first 3 lines
with open('moby_dick.txt') as file:
    print(file.readline())
    print(file.readline())
    print(file.readline())

# The rows that you wish to print
I = [0,1,3,5,6,7,8,9]

# Print out these rows
with open('moby_dick.txt') as file:
    for i, row in enumerate(file):
        if i in I:
            print(row)
```

Student code

```
# Read & print the first 3 lines
with open('moby_dick.txt') as file:
    print(file.readline())
    print(file.readline())
    print('test')

# The rows that you wish to print
I = [0,1,3,5,6,7,8,9]
```

(continues on next page)

(continued from previous page)

```
# Print out these rows
with open('moby_dick.txt') as file:
    for i, row in enumerate(file):
        if i in I:
            print(row)
```

Student output

```
CHAPTER 1. Loomings.



test
CHAPTER 1. Loomings.



little or no money in my purse, and nothing particular to interest me on

the world. It is a way I have of driving off the spleen and regulating

the circulation. Whenever I find myself growing grim about the mouth;

whenever it is a damp, drizzly November in my soul; whenever I find

myself involuntarily pausing before coffin warehouses, and bringing up

the rear of every funeral I meet; and especially whenever my hypos get
```

SCT

```
test_with(2, body = lambda: test_for_loop(1, body = lambda: test_if_else(1, body =␣
↪lambda: test_function('print'))))
```

Result

```
Great work!
```

No error

13.8.3 Example 3

PEC

```
from urllib.request import urlretrieve; urlretrieve('http://s3.amazonaws.com/assets.
↪datacamp.com/production/course_998/datasets/moby_opens.txt', 'moby_dick.txt')
```

Solution code

```
# Read & print the first 3 lines
with open('moby_dick.txt') as file:
    print(file.readline())
    print(file.readline())
    print(file.readline())
```

(continues on next page)

(continued from previous page)

```
# The rows that you wish to print
I = [0,1,3,5,6,7,8,9]

# Print out these rows
with open('moby_dick.txt') as file:
    for i, row in enumerate(file):
        if i in I:
            print(row)
```

Student code

```
# Read & print the first 3 lines
with open('moby_dick.txt') as file:
    print(file.readline())
    print(file.readline())
    print('test')

# The rows that you wish to print
I = [0,1,3,5,6,7,8,9]

# Print out these rows
with open('moby_dick.txt') as file:
    for i, row in enumerate(file):
        if i in I:
            print(row)
```

Student output

```
CHAPTER 1. Loomings.



test
CHAPTER 1. Loomings.



little or no money in my purse, and nothing particular to interest me on

the world. It is a way I have of driving off the spleen and regulating

the circulation. Whenever I find myself growing grim about the mouth;

whenever it is a damp, drizzly November in my soul; whenever I find

myself involuntarily pausing before coffin warehouses, and bringing up

the rear of every funeral I meet; and especially whenever my hypos get
```

SCT

```
test_with(1, body = [test_function('print', index = i + 1) for i in range(3)])
```

Result

```
Check your third call of <code>print()</code>. Did you correctly specify the first␣
↪argument? Expected something different.
```

No error

13.8.4 Example 4

PEC

```
from urllib.request import urlretrieve; urlretrieve('http://s3.amazonaws.com/assets.
↪datacamp.com/production/course_998/datasets/moby_opens.txt', 'moby_dick.txt')
```

Solution code

```
# Read & print the first 3 lines
with open('moby_dick.txt') as file:
    print(file.readline())
    print(file.readline())
    print(file.readline())

# The rows that you wish to print
I = [0,1,3,5,6,7,8,9]

# Print out these rows
with open('moby_dick.txt') as file:
    for i, row in enumerate(file):
        if i in I:
            print(row)
```

Student code

```
# Read & print the first 3 lines
with open('moby_dick.txt') as file:
    print(file.readline())
    print(file.readline())
    print('test')

# The rows that you wish to print
I = [0,1,3,5,6,7,8,9]

# Print out these rows
with open('moby_dick.txt') as file:
    for i, row in enumerate(file):
        if i in I:
            print(row)
```

Student output

```
CHAPTER 1. Loomings.



test
CHAPTER 1. Loomings.



little or no money in my purse, and nothing particular to interest me on

the world. It is a way I have of driving off the spleen and regulating
```

(continues on next page)

(continued from previous page)

```
the circulation. Whenever I find myself growing grim about the mouth;

whenever it is a damp, drizzly November in my soul; whenever I find

myself involuntarily pausing before coffin warehouses, and bringing up

the rear of every funeral I meet; and especially whenever my hypos get
```

SCT

```
test_with(2, body = test_for_loop(1, body = test_if_else(1, body = test_function(
↪'print'))))
```

Result

```
Great work!
```

No error

13.8.5 Example 5

PEC

```
from urllib.request import urlretrieve; urlretrieve('http://s3.amazonaws.com/assets.
↪datacamp.com/production/course_998/datasets/moby_opens.txt', 'moby_dick.txt')
```

Solution code

```
# Read & print the first 3 lines
with open('moby_dick.txt') as file:
    print(file.readline())
    print(file.readline())
    print(file.readline())

# The rows that you wish to print
I = [0,1,3,5,6,7,8,9]

# Print out these rows
with open('moby_dick.txt') as file:
    for i, row in enumerate(file):
        if i in I:
            print(row)
```

Student code

```
# Read & print the first 3 lines
with open('moby_dick.txt') as file:
    print(file.readline())
    print(file.readline())
    print('test')

# The rows that you wish to print
I = [0,1,3,5,6,7,8,9]
```

(continues on next page)

(continued from previous page)

```
# Print out these rows
with open('moby_dick.txt') as file:
    for i, row in enumerate(file):
        if i in I:
            print(row)
```

Student output

```
CHAPTER 1. Loomings.



test
CHAPTER 1. Loomings.



little or no money in my purse, and nothing particular to interest me on

the world. It is a way I have of driving off the spleen and regulating

the circulation. Whenever I find myself growing grim about the mouth;

whenever it is a damp, drizzly November in my soul; whenever I find

myself involuntarily pausing before coffin warehouses, and bringing up

the rear of every funeral I meet; and especially whenever my hypos get
```

SCT

```
for_test = test_for_loop(1, body = test_if_else(1, body = test_function('print')))
Ex().check_with(1).check_body().with_context(for_test)
```

Result

```
Great work!
```

No error

13.8.6 Example 6

PEC

```
from urllib.request import urlretrieve; urlretrieve('http://s3.amazonaws.com/assets.
↪datacamp.com/production/course_998/datasets/moby_opens.txt', 'moby_dick.txt')
```

Solution code

```
# Read & print the first 3 lines
with open('moby_dick.txt') as file:
    print(file.readline())
    print(file.readline())
    print(file.readline())
```

(continues on next page)

(continued from previous page)

```
# The rows that you wish to print
I = [0,1,3,5,6,7,8,9]

# Print out these rows
with open('moby_dick.txt') as file:
    for i, row in enumerate(file):
        if i in I:
            print(row)
```

Student code

```
# Read & print the first 3 lines
with open('moby_dick.txt') as file:
    print(file.readline())
    print(file.readline())
    print('test')

# The rows that you wish to print
I = [0,1,3,5,6,7,8,9]

# Print out these rows
with open('moby_dick.txt') as file:
    for i, row in enumerate(file):
        if i in I:
            print(row)
```

Student output

```
CHAPTER 1. Loomings.



test
CHAPTER 1. Loomings.



little or no money in my purse, and nothing particular to interest me on

the world. It is a way I have of driving off the spleen and regulating

the circulation. Whenever I find myself growing grim about the mouth;

whenever it is a damp, drizzly November in my soul; whenever I find

myself involuntarily pausing before coffin warehouses, and bringing up

the rear of every funeral I meet; and especially whenever my hypos get
```

SCT

```
Ex().check_with(0).check_body().with_context([test_function('print', index = i+1) for␣
↪i in range(3)])
```

Result

```
Check your third call of <code>print()</code>. Did you correctly specify the first␣
↪argument? Expected something different.
```

No error

13.8.7 Example 7

PEC

```
from urllib.request import urlretrieve; urlretrieve('http://s3.amazonaws.com/assets.
↪datacamp.com/production/course_998/datasets/moby_opens.txt', 'moby_dick.txt')
```

Solution code

```
# Read & print the first 3 lines
with open('moby_dick.txt') as file:
    print(file.readline())
    print(file.readline())
    print(file.readline())

# The rows that you wish to print
I = [0,1,3,5,6,7,8,9]

# Print out these rows
with open('moby_dick.txt') as file:
    for i, row in enumerate(file):
        if i in I:
            print(row)
```

Student code

```
# Read & print the first 3 lines
with open('moby_dick.txt') as file:
    print(file.readline())
    print(file.readline())
    print('test')

# The rows that you wish to print
I = [0,1,3,5,6,7,8,9]

# Print out these rows
with open('moby_dick.txt') as file:
    for i, row in enumerate(file):
        if i in I:
            print(row)
```

Student output

```
CHAPTER 1. Loomings.



test
CHAPTER 1. Loomings.
```

(continues on next page)

(continued from previous page)

```
little or no money in my purse, and nothing particular to interest me on

the world. It is a way I have of driving off the spleen and regulating

the circulation. Whenever I find myself growing grim about the mouth;

whenever it is a damp, drizzly November in my soul; whenever I find

myself involuntarily pausing before coffin warehouses, and bringing up

the rear of every funeral I meet; and especially whenever my hypos get
```

SCT

```
# since the print func is being tested w/o SCTs setting any variables, don't need␣
↪with_context
for_test = test_for_loop(1, body = test_if_else(1, body = test_function('print')))
Ex().check_with(1).check_body().multi(for_test)
```

Result

```
Great work!
```

No error

13.8.8 Example 8

PEC

```
from urllib.request import urlretrieve; urlretrieve('http://s3.amazonaws.com/assets.
↪datacamp.com/production/course_998/datasets/moby_opens.txt', 'moby_dick.txt')
```

Solution code

```
# Read & print the first 3 lines
with open('moby_dick.txt') as file:
    print(file.readline())
    print(file.readline())
    print(file.readline())

# The rows that you wish to print
I = [0,1,3,5,6,7,8,9]

# Print out these rows
with open('moby_dick.txt') as file:
    for i, row in enumerate(file):
        if i in I:
            print(row)
```

Student code

```
# Read & print the first 3 lines
with open('moby_dick.txt') as file, open('moby_dick.txt'):
    print(file.readline())
```

(continues on next page)

(continued from previous page)

```
    print(file.readline())
    print('test')

# The rows that you wish to print
I = [0,1,3,5,6,7,8,9]

# Print out these rows
with open('moby_dick.txt') as not_file:
    for i, row in enumerate(not_file):
        if i in I:
            print(row)
```

Student output

```
CHAPTER 1. Loomings.



test
CHAPTER 1. Loomings.



little or no money in my purse, and nothing particular to interest me on

the world. It is a way I have of driving off the spleen and regulating

the circulation. Whenever I find myself growing grim about the mouth;

whenever it is a damp, drizzly November in my soul; whenever I find

myself involuntarily pausing before coffin warehouses, and bringing up

the rear of every funeral I meet; and especially whenever my hypos get
```

SCT

```
test_with(1, context_vals=True)
```

Result

```
Check the first <code>with</code> statement. Make sure to use the correct number of␣
↪context variables. It seems you defined too many.
```

No error

13.8.9 Example 9

PEC

```
from urllib.request import urlretrieve; urlretrieve('http://s3.amazonaws.com/assets.
↪datacamp.com/production/course_998/datasets/moby_opens.txt', 'moby_dick.txt')
```

Solution code

```
# Read & print the first 3 lines
with open('moby_dick.txt') as file:
    print(file.readline())
    print(file.readline())
    print(file.readline())

# The rows that you wish to print
I = [0,1,3,5,6,7,8,9]

# Print out these rows
with open('moby_dick.txt') as file:
    for i, row in enumerate(file):
        if i in I:
            print(row)
```

Student code

```
# Read & print the first 3 lines
with open('moby_dick.txt') as file, open('moby_dick.txt'):
    print(file.readline())
    print(file.readline())
    print('test')

# The rows that you wish to print
I = [0,1,3,5,6,7,8,9]

# Print out these rows
with open('moby_dick.txt') as not_file:
    for i, row in enumerate(not_file):
        if i in I:
            print(row)
```

Student output

```
CHAPTER 1. Loomings.



test
CHAPTER 1. Loomings.



little or no money in my purse, and nothing particular to interest me on

the world. It is a way I have of driving off the spleen and regulating

the circulation. Whenever I find myself growing grim about the mouth;

whenever it is a damp, drizzly November in my soul; whenever I find

myself involuntarily pausing before coffin warehouses, and bringing up

the rear of every funeral I meet; and especially whenever my hypos get
```

SCT

```
test_with(2, context_vals=True)
```

Result

```
Check the second <code>with</code> statement. Did you correctly specify the first␣
↪context? Make sure to use the correct context variable names. Was expecting <code>
↪file</code> but got <code>not_file</code>.
```

No error

13.8.10 Example 10

PEC

```
from urllib.request import urlretrieve; urlretrieve('http://s3.amazonaws.com/assets.
↪datacamp.com/production/course_998/datasets/moby_opens.txt', 'moby_dick.txt')
from urllib.request import urlretrieve; urlretrieve('https://s3.amazonaws.com/assets.
↪datacamp.com/production/course_998/datasets/moby_opens.txt', 'not_moby_dick.txt')
```

Solution code

```
# Read & print the first 3 lines
with open('moby_dick.txt') as file, open('moby_dick.txt'):
    print(file.readline())
    print(file.readline())
    print(file.readline())

# The rows that you wish to print
I = [0,1,3,5,6,7,8,9]

# Print out these rows
with open('moby_dick.txt') as file, open('not_moby_dick.txt') as not_file:
    for i, row in enumerate(file):
        if i in I:
            print(row)
```

Student code

```
# Read & print the first 3 lines
with open('moby_dick.txt') as file:
    print(file.readline())
    print(file.readline())
    print('test')

# The rows that you wish to print
I = [0,1,3,5,6,7,8,9]

# Print out these rows
with open('moby_dick.txt') as not_file, open('moby_dick.txt') as file:
    for i, row in enumerate(not_file):
        if i in I:
            print(row)
```

Student output

```
CHAPTER 1. Loomings.



test
CHAPTER 1. Loomings.



little or no money in my purse, and nothing particular to interest me on

the world. It is a way I have of driving off the spleen and regulating

the circulation. Whenever I find myself growing grim about the mouth;

whenever it is a damp, drizzly November in my soul; whenever I find

myself involuntarily pausing before coffin warehouses, and bringing up

the rear of every funeral I meet; and especially whenever my hypos get
```

SCT

```
test_with(1, context_tests=lambda: test_function('open'))
```

Result

```
Great work!
```

No error

13.8.11 Example 11

PEC

```
from urllib.request import urlretrieve; urlretrieve('http://s3.amazonaws.com/assets.
↪datacamp.com/production/course_998/datasets/moby_opens.txt', 'moby_dick.txt')
from urllib.request import urlretrieve; urlretrieve('https://s3.amazonaws.com/assets.
↪datacamp.com/production/course_998/datasets/moby_opens.txt', 'not_moby_dick.txt')
```

Solution code

```
# Read & print the first 3 lines
with open('moby_dick.txt') as file, open('moby_dick.txt'):
    print(file.readline())
    print(file.readline())
    print(file.readline())

# The rows that you wish to print
I = [0,1,3,5,6,7,8,9]

# Print out these rows
with open('moby_dick.txt') as file, open('not_moby_dick.txt') as not_file:
    for i, row in enumerate(file):
        if i in I:
            print(row)
```

Student code

```
# Read & print the first 3 lines
with open('moby_dick.txt') as file:
    print(file.readline())
    print(file.readline())
    print('test')

# The rows that you wish to print
I = [0,1,3,5,6,7,8,9]

# Print out these rows
with open('moby_dick.txt') as not_file, open('moby_dick.txt') as file:
    for i, row in enumerate(not_file):
        if i in I:
            print(row)
```

Student output

```
CHAPTER 1. Loomings.



test
CHAPTER 1. Loomings.



little or no money in my purse, and nothing particular to interest me on

the world. It is a way I have of driving off the spleen and regulating

the circulation. Whenever I find myself growing grim about the mouth;

whenever it is a damp, drizzly November in my soul; whenever I find

myself involuntarily pausing before coffin warehouses, and bringing up

the rear of every funeral I meet; and especially whenever my hypos get
```

SCT

```
test_with(1, context_tests=[
    lambda: test_function('open'),
    lambda: test_function('open')])
```

Result

```
Check the first <code>with</code> statement. Make sure to use the correct number of␣
↪context variables. It seems you defined too little.
```

No error

13.8.12 Example 12

PEC

```
from urllib.request import urlretrieve; urlretrieve('http://s3.amazonaws.com/assets.
↪datacamp.com/production/course_998/datasets/moby_opens.txt', 'moby_dick.txt')
from urllib.request import urlretrieve; urlretrieve('https://s3.amazonaws.com/assets.
↪datacamp.com/production/course_998/datasets/moby_opens.txt', 'not_moby_dick.txt')
```

Solution code

```
# Read & print the first 3 lines
with open('moby_dick.txt') as file, open('moby_dick.txt'):
    print(file.readline())
    print(file.readline())
    print(file.readline())

# The rows that you wish to print
I = [0,1,3,5,6,7,8,9]

# Print out these rows
with open('moby_dick.txt') as file, open('not_moby_dick.txt') as not_file:
    for i, row in enumerate(file):
        if i in I:
            print(row)
```

Student code

```
# Read & print the first 3 lines
with open('moby_dick.txt') as file:
    print(file.readline())
    print(file.readline())
    print('test')

# The rows that you wish to print
I = [0,1,3,5,6,7,8,9]

# Print out these rows
with open('moby_dick.txt') as not_file, open('moby_dick.txt') as file:
    for i, row in enumerate(not_file):
        if i in I:
            print(row)
```

Student output

```
CHAPTER 1. Loomings.



test
CHAPTER 1. Loomings.



little or no money in my purse, and nothing particular to interest me on

the world. It is a way I have of driving off the spleen and regulating

the circulation. Whenever I find myself growing grim about the mouth;

whenever it is a damp, drizzly November in my soul; whenever I find
```

(continues on next page)

(continued from previous page)

```
myself involuntarily pausing before coffin warehouses, and bringing up

the rear of every funeral I meet; and especially whenever my hypos get
```

SCT

```
test_with(2, context_tests=[
    lambda: test_function('open'),
    lambda: test_function('open')])
```

Result

```
Check your call of <code>open()</code>. Did you correctly specify the first argument? ↪Expected <code>not_moby_dick.txt</code>, but got <code>moby_dick.txt</code>.
```

No error

13.8.13 Example 13

PEC

```
class A:
    def __enter__(self): return [1,2, 3]
    def __exit__(self, *args, **kwargs): return
```

Solution code

```
with A() as (one, *others):
    print(one)
    print(others)
```

Student code

```
with A() as (one, *others):
    print(one)
    print(others)
```

Student output

```
1
[2, 3]
```

SCT

```
test_with(1, body=[test_function('print'), test_function('print')])
```

Result

```
Great work!
```

No error

13.8.14 Example 14

PEC

```
from io import StringIO
```

Solution code

```
with StringIO() as f1, StringIO() as f2: pass
```

Student code

```
with StringIO() as f1, StringIO() as f2: pass
```

No output

SCT

```
Ex().check_with(0).has_context()
```

Result

```
Great work!
```

No error

13.8.15 Example 15

PEC

```
from io import StringIO
```

Solution code

```
with StringIO() as f1, StringIO() as f2: pass
```

Student code

```
with StringIO() as f1: pass
```

No output

SCT

```
Ex().check_with(0).has_context()
```

Result

```
Check the first <code>with</code> statement. Are you sure you defined the second␣
↪context?
```

No error

13.8.16 Example 16

PEC

```
from io import StringIO
```

Solution code

```
with StringIO() as f1, StringIO() as f2: pass
```

Student code

```
with StringIO() as f3, StringIO() as f4: pass
```

No output

SCT

```
Ex().check_with(0).has_context(exact_names=True)
```

Result

```
Check the first <code>with</code> statement. Did you correctly specify the first␣
↪context? Make sure to use the correct context variable names. Was expecting <code>f1
↪</code> but got <code>f3</code>.
```

No error

13.8.17 Example 17

PEC

```
from io import StringIO
```

Solution code

```
with StringIO() as f1, StringIO() as f2: pass
```

Student code

```
with StringIO() as f1, StringIO() as f2: pass
```

No output

SCT

```
Ex().check_with(0).check_context(0).has_context()
```

Result

```
Great work!
```

No error

13.8.18 Example 18

PEC

```
from io import StringIO
```

Solution code

```
with StringIO() as f1, StringIO() as f2: pass
```

Student code

```
with StringIO() as f3: pass
```

No output

SCT

```
Ex().check_with(0).check_context(0).has_context(exact_names=True)
```

Result

```
Check the first <code>with</code> statement. Did you correctly specify the first␣
↪context?
```

No error

13.9 test_check_try_except.py

13.9.1 Example 1

PEC

```
import numpy as np
```

Solution code

```
try:
    x = max([1, 2, 'a'])
except TypeError as e:
    x = 'typeerror'
except __builtin__.ValueError:
    x = 'valueerror'
except (ZeroDivisionError, IOError) as e:
    x = e
except pd.io.common.CParserError:
    x = 'CParserError'
except :
    x = 'someerror'
else :
    passed = True
finally:
    print('done')
```

No student code

No output

SCT

```
Ex().check_try_except().multi(
    check_body().check_function('max', signature = False).check_args(0).has_equal_
↪value(),
    check_handlers('TypeError').has_equal_value(name = 'x'),
    check_handlers('ValueError').has_equal_value(name = 'x'),
    check_handlers('ZeroDivisionError').set_context(e = 'anerror').has_equal_
↪value(name = 'x'),
    check_handlers('IOError').set_context(e = 'anerror').has_equal_value(name = 'x'),
    check_handlers('CParserError').has_equal_value(name = 'x'),
    check_handlers('all').has_equal_value(name = 'x'),
    check_orelse().has_equal_value(name = 'passed'),
    check_finalbody().check_function('print').check_args(0).has_equal_value()
)
```

Result

```
The system wants to check the first try statement but hasn't found it.
```

No error

13.9.2 Example 2

PEC

```
import numpy as np
```

Solution code

```
try:
    x = max([1, 2, 'a'])
except TypeError as e:
    x = 'typeerror'
except __builtin__.ValueError:
    x = 'valueerror'
except (ZeroDivisionError, IOError) as e:
    x = e
except pd.io.common.CParserError:
    x = 'CParserError'
except :
    x = 'someerror'
else :
    passed = True
finally:
    print('done')
```

Student code

```
try:
    x = max([1, 2, 'a'])
except TypeError as e:
    x = 'typeerrors'
```

No output

SCT

```
Ex().check_try_except().multi(
    check_body().check_function('max', signature = False).check_args(0).has_equal_
↪value(),
    check_handlers('TypeError').has_equal_value(name = 'x'),
    check_handlers('ValueError').has_equal_value(name = 'x'),
    check_handlers('ZeroDivisionError').set_context(e = 'anerror').has_equal_
↪value(name = 'x'),
    check_handlers('IOError').set_context(e = 'anerror').has_equal_value(name = 'x'),
    check_handlers('CParserError').has_equal_value(name = 'x'),
    check_handlers('all').has_equal_value(name = 'x'),
    check_orelse().has_equal_value(name = 'passed'),
    check_finalbody().check_function('print').check_args(0).has_equal_value()
)
```

Result

```
Check the first try statement. Did you correctly specify the <code>TypeError</code>
↪<code>except</code> block? Are you sure you assigned the correct value to <code>x</
↪code>?
```

No error

13.9.3 Example 3

PEC

```
import numpy as np
```

Solution code

```
try:
    x = max([1, 2, 'a'])
except TypeError as e:
    x = 'typeerror'
except __builtin__.ValueError:
    x = 'valueerror'
except (ZeroDivisionError, IOError) as e:
    x = e
except pd.io.common.CParserError:
    x = 'CParserError'
except :
    x = 'someerror'
else :
    passed = True
finally:
    print('done')
```

Student code

```
try:
    x = max([1, 2, 'a'])
except TypeError as e:
    x = 'typeerror'
```

No output

SCT

```
Ex().check_try_except().multi(
    check_body().check_function('max', signature = False).check_args(0).has_equal_
↪value(),
    check_handlers('TypeError').has_equal_value(name = 'x'),
    check_handlers('ValueError').has_equal_value(name = 'x'),
    check_handlers('ZeroDivisionError').set_context(e = 'anerror').has_equal_
↪value(name = 'x'),
    check_handlers('IOError').set_context(e = 'anerror').has_equal_value(name = 'x'),
    check_handlers('CParserError').has_equal_value(name = 'x'),
    check_handlers('all').has_equal_value(name = 'x'),
    check_orelse().has_equal_value(name = 'passed'),
    check_finalbody().check_function('print').check_args(0).has_equal_value()
)
```

Result

```
Check the first try statement. Are you sure you defined the <code>ValueError</code>
↪<code>except</code> block?
```

No error

13.9.4 Example 4

PEC

```
import numpy as np
```

Solution code

```
try:
    x = max([1, 2, 'a'])
except TypeError as e:
    x = 'typeerror'
except __builtin__.ValueError:
    x = 'valueerror'
except (ZeroDivisionError, IOError) as e:
    x = e
except pd.io.common.CParserError:
    x = 'CParserError'
except :
    x = 'someerror'
else :
    passed = True
finally:
    print('done')
```

Student code

```
try:
    x = max([1, 2, 'a'])
except TypeError as e:
    x = 'typeerror'
except ValueError:
    x = 'valueerrors'
```

No output

SCT

```
Ex().check_try_except().multi(
    check_body().check_function('max', signature = False).check_args(0).has_equal_
↪value(),
    check_handlers('TypeError').has_equal_value(name = 'x'),
    check_handlers('ValueError').has_equal_value(name = 'x'),
    check_handlers('ZeroDivisionError').set_context(e = 'anerror').has_equal_
↪value(name = 'x'),
    check_handlers('IOError').set_context(e = 'anerror').has_equal_value(name = 'x'),
    check_handlers('CParserError').has_equal_value(name = 'x'),
    check_handlers('all').has_equal_value(name = 'x'),
    check_orelse().has_equal_value(name = 'passed'),
    check_finalbody().check_function('print').check_args(0).has_equal_value()
)
```

Result

```
Check the first try statement. Did you correctly specify the <code>ValueError</code>
↪<code>except</code> block? Are you sure you assigned the correct value to <code>x</
↪code>?
```

No error

13.9.5 Example 5

PEC

```
import numpy as np
```

Solution code

```
try:
    x = max([1, 2, 'a'])
except TypeError as e:
    x = 'typeerror'
except __builtin__.ValueError:
    x = 'valueerror'
except (ZeroDivisionError, IOError) as e:
    x = e
except pd.io.common.CParserError:
    x = 'CParserError'
except :
    x = 'someerror'
else :
    passed = True
finally:
    print('done')
```

Student code

```
try:
    x = max([1, 2, 'a'])
except TypeError as e:
    x = 'typeerror'
except ValueError:
    x = 'valueerror'
```

No output

SCT

```
Ex().check_try_except().multi(
    check_body().check_function('max', signature = False).check_args(0).has_equal_
↪value(),
    check_handlers('TypeError').has_equal_value(name = 'x'),
    check_handlers('ValueError').has_equal_value(name = 'x'),
    check_handlers('ZeroDivisionError').set_context(e = 'anerror').has_equal_
↪value(name = 'x'),
    check_handlers('IOError').set_context(e = 'anerror').has_equal_value(name = 'x'),
    check_handlers('CParserError').has_equal_value(name = 'x'),
    check_handlers('all').has_equal_value(name = 'x'),
    check_orelse().has_equal_value(name = 'passed'),
    check_finalbody().check_function('print').check_args(0).has_equal_value()
)
```

Result

```
Check the first try statement. Are you sure you defined the <code>ZeroDivisionError</
↪code> <code>except</code> block?
```

No error

13.9.6 Example 6

PEC

```
import numpy as np
```

Solution code

```
try:
    x = max([1, 2, 'a'])
except TypeError as e:
    x = 'typeerror'
except __builtin__.ValueError:
    x = 'valueerror'
except (ZeroDivisionError, IOError) as e:
    x = e
except pd.io.common.CParserError:
    x = 'CParserError'
except :
    x = 'someerror'
else :
    passed = True
finally:
    print('done')
```

Student code

```
try:
    x = max([1, 2, 'a'])
except TypeError as e:
    x = 'typeerror'
except ValueError:
```

(continues on next page)

(continued from previous page)

```
    x = 'valueerror'
except ZeroDivisionError as e:
    x = 'test'
```

No output

SCT

```
Ex().check_try_except().multi(
    check_body().check_function('max', signature = False).check_args(0).has_equal_
↪value(),
    check_handlers('TypeError').has_equal_value(name = 'x'),
    check_handlers('ValueError').has_equal_value(name = 'x'),
    check_handlers('ZeroDivisionError').set_context(e = 'anerror').has_equal_
↪value(name = 'x'),
    check_handlers('IOError').set_context(e = 'anerror').has_equal_value(name = 'x'),
    check_handlers('CParserError').has_equal_value(name = 'x'),
    check_handlers('all').has_equal_value(name = 'x'),
    check_orelse().has_equal_value(name = 'passed'),
    check_finalbody().check_function('print').check_args(0).has_equal_value()
)
```

Result

```
Check the first try statement. Did you correctly specify the <code>ZeroDivisionError</
↪code> <code>except</code> block? Are you sure you assigned the correct value to
↪<code>x</code>?
```

No error

13.9.7 Example 7

PEC

```
import numpy as np
```

Solution code

```
try:
    x = max([1, 2, 'a'])
except TypeError as e:
    x = 'typeerror'
except __builtin__.ValueError:
    x = 'valueerror'
except (ZeroDivisionError, IOError) as e:
    x = e
except pd.io.common.CParserError:
    x = 'CParserError'
except :
    x = 'someerror'
else :
    passed = True
finally:
    print('done')
```

Student code

```
try:
    x = max([1, 2, 'a'])
except TypeError as e:
    x = 'typeerror'
except ValueError:
    x = 'valueerror'
except ZeroDivisionError as e:
    x = e
```

No output

SCT

```
Ex().check_try_except().multi(
    check_body().check_function('max', signature = False).check_args(0).has_equal_
↪value(),
    check_handlers('TypeError').has_equal_value(name = 'x'),
    check_handlers('ValueError').has_equal_value(name = 'x'),
    check_handlers('ZeroDivisionError').set_context(e = 'anerror').has_equal_
↪value(name = 'x'),
    check_handlers('IOError').set_context(e = 'anerror').has_equal_value(name = 'x'),
    check_handlers('CParserError').has_equal_value(name = 'x'),
    check_handlers('all').has_equal_value(name = 'x'),
    check_orelse().has_equal_value(name = 'passed'),
    check_finalbody().check_function('print').check_args(0).has_equal_value()
)
```

Result

```
Check the first try statement. Are you sure you defined the <code>IOError</code>
↪<code>except</code> block?
```

No error

13.9.8 Example 8

PEC

```
import numpy as np
```

Solution code

```
try:
    x = max([1, 2, 'a'])
except TypeError as e:
    x = 'typeerror'
except __builtin__.ValueError:
    x = 'valueerror'
except (ZeroDivisionError, IOError) as e:
    x = e
except pd.io.common.CParserError:
    x = 'CParserError'
except :
    x = 'someerror'
else :
    passed = True
```

(continues on next page)

(continued from previous page)

```
finally:
    print('done')
```

Student code

```
try:
    x = max([1, 2, 'a'])
except TypeError as e:
    x = 'typeerror'
except ValueError:
    x = 'valueerror'
except (ZeroDivisionError, IOError) as e:
    x = 'test'
```

No output

SCT

```
Ex().check_try_except().multi(
    check_body().check_function('max', signature = False).check_args(0).has_equal_
↪value(),
    check_handlers('TypeError').has_equal_value(name = 'x'),
    check_handlers('ValueError').has_equal_value(name = 'x'),
    check_handlers('ZeroDivisionError').set_context(e = 'anerror').has_equal_
↪value(name = 'x'),
    check_handlers('IOError').set_context(e = 'anerror').has_equal_value(name = 'x'),
    check_handlers('CParserError').has_equal_value(name = 'x'),
    check_handlers('all').has_equal_value(name = 'x'),
    check_orelse().has_equal_value(name = 'passed'),
    check_finalbody().check_function('print').check_args(0).has_equal_value()
)
```

Result

```
Check the first try statement. Did you correctly specify the <code>ZeroDivisionError</
↪code> <code>except</code> block? Are you sure you assigned the correct value to
↪<code>x</code>?
```

No error

13.9.9 Example 9

PEC

```
import numpy as np
```

Solution code

```
try:
    x = max([1, 2, 'a'])
except TypeError as e:
    x = 'typeerror'
except __builtin__.ValueError:
    x = 'valueerror'
except (ZeroDivisionError, IOError) as e:
```

(continues on next page)

(continued from previous page)

```
    x = e
except pd.io.common.CParserError:
    x = 'CParserError'
except :
    x = 'someerror'
else :
    passed = True
finally:
    print('done')
```

Student code

```
try:
    x = max([1, 2, 'a'])
except TypeError as e:
    x = 'typeerror'
except ValueError:
    x = 'valueerror'
except (ZeroDivisionError, IOError) as e:
    x = e
```

No output

SCT

```
Ex().check_try_except().multi(
    check_body().check_function('max', signature = False).check_args(0).has_equal_
↪value(),
    check_handlers('TypeError').has_equal_value(name = 'x'),
    check_handlers('ValueError').has_equal_value(name = 'x'),
    check_handlers('ZeroDivisionError').set_context(e = 'anerror').has_equal_
↪value(name = 'x'),
    check_handlers('IOError').set_context(e = 'anerror').has_equal_value(name = 'x'),
    check_handlers('CParserError').has_equal_value(name = 'x'),
    check_handlers('all').has_equal_value(name = 'x'),
    check_orelse().has_equal_value(name = 'passed'),
    check_finalbody().check_function('print').check_args(0).has_equal_value()
)
```

Result

```
Check the first try statement. Are you sure you defined the <code>CParserError</code>
↪<code>except</code> block?
```

No error

13.9.10 Example 10

PEC

```
import numpy as np
```

Solution code

```
try:
    x = max([1, 2, 'a'])
except TypeError as e:
    x = 'typeerror'
except __builtin__.ValueError:
    x = 'valueerror'
except (ZeroDivisionError, IOError) as e:
    x = e
except pd.io.common.CParserError:
    x = 'CParserError'
except :
    x = 'someerror'
else :
    passed = True
finally:
    print('done')
```

Student code

```
try:
    x = max([1, 2, 'a'])
except TypeError as e:
    x = 'typeerror'
except ValueError:
    x = 'valueerror'
except (ZeroDivisionError, IOError) as e:
    x = e
except pd.io.common.CParserError as e:
    x = 'CParserError'
```

No output

SCT

```
Ex().check_try_except().multi(
    check_body().check_function('max', signature = False).check_args(0).has_equal_
↪value(),
    check_handlers('TypeError').has_equal_value(name = 'x'),
    check_handlers('ValueError').has_equal_value(name = 'x'),
    check_handlers('ZeroDivisionError').set_context(e = 'anerror').has_equal_
↪value(name = 'x'),
    check_handlers('IOError').set_context(e = 'anerror').has_equal_value(name = 'x'),
    check_handlers('CParserError').has_equal_value(name = 'x'),
    check_handlers('all').has_equal_value(name = 'x'),
    check_orelse().has_equal_value(name = 'passed'),
    check_finalbody().check_function('print').check_args(0).has_equal_value()
)
```

Result

```
Check the first try statement. Are you sure you defined the <code>all</code> <code>
↪except</code> block?
```

No error

13.9.11 Example 11

PEC

```
import numpy as np
```

Solution code

```
try:
    x = max([1, 2, 'a'])
except TypeError as e:
    x = 'typeerror'
except __builtin__.ValueError:
    x = 'valueerror'
except (ZeroDivisionError, IOError) as e:
    x = e
except pd.io.common.CParserError:
    x = 'CParserError'
except :
    x = 'someerror'
else :
    passed = True
finally:
    print('done')
```

Student code

```
try:
    x = max([1, 2, 'a'])
except TypeError as e:
    x = 'typeerror'
except ValueError:
    x = 'valueerror'
except (ZeroDivisionError, IOError) as e:
    x = e
except pd.io.common.CParserError as e:
    x = 'CParserError'
except :
    x = 'someerrors'
```

No output

SCT

```
Ex().check_try_except().multi(
    check_body().check_function('max', signature = False).check_args(0).has_equal_
↪value(),
    check_handlers('TypeError').has_equal_value(name = 'x'),
    check_handlers('ValueError').has_equal_value(name = 'x'),
    check_handlers('ZeroDivisionError').set_context(e = 'anerror').has_equal_
↪value(name = 'x'),
    check_handlers('IOError').set_context(e = 'anerror').has_equal_value(name = 'x'),
    check_handlers('CParserError').has_equal_value(name = 'x'),
    check_handlers('all').has_equal_value(name = 'x'),
    check_orelse().has_equal_value(name = 'passed'),
    check_finalbody().check_function('print').check_args(0).has_equal_value()
)
```

Result

```
Check the first try statement. Did you correctly specify the <code>all</code> <code>
↪except</code> block? Are you sure you assigned the correct value to <code>x</code>?
```

No error

13.9.12 Example 12

PEC

```
import numpy as np
```

Solution code

```
try:
    x = max([1, 2, 'a'])
except TypeError as e:
    x = 'typeerror'
except __builtin__.ValueError:
    x = 'valueerror'
except (ZeroDivisionError, IOError) as e:
    x = e
except pd.io.common.CParserError:
    x = 'CParserError'
except :
    x = 'someerror'
else :
    passed = True
finally:
    print('done')
```

Student code

```
try:
    x = max([1, 2, 'a'])
except TypeError as e:
    x = 'typeerror'
except ValueError:
    x = 'valueerror'
except (ZeroDivisionError, IOError) as e:
    x = e
except pd.io.common.CParserError as e:
    x = 'CParserError'
except :
    x = 'someerror'
```

No output

SCT

```
Ex().check_try_except().multi(
    check_body().check_function('max', signature = False).check_args(0).has_equal_
↪value(),
    check_handlers('TypeError').has_equal_value(name = 'x'),
    check_handlers('ValueError').has_equal_value(name = 'x'),
    check_handlers('ZeroDivisionError').set_context(e = 'anerror').has_equal_
↪value(name = 'x'),
```

(continues on next page)

(continued from previous page)

```
    check_handlers('IOError').set_context(e = 'anerror').has_equal_value(name = 'x'),
    check_handlers('CParserError').has_equal_value(name = 'x'),
    check_handlers('all').has_equal_value(name = 'x'),
    check_orelse().has_equal_value(name = 'passed'),
    check_finalbody().check_function('print').check_args(0).has_equal_value()
)
```

Result

```
Check the first try statement. Are you sure you defined the else part?
```

No error

13.9.13 Example 13

PEC

```
import numpy as np
```

Solution code

```
try:
    x = max([1, 2, 'a'])
except TypeError as e:
    x = 'typeerror'
except __builtin__.ValueError:
    x = 'valueerror'
except (ZeroDivisionError, IOError) as e:
    x = e
except pd.io.common.CParserError:
    x = 'CParserError'
except :
    x = 'someerror'
else :
    passed = True
finally:
    print('done')
```

Student code

```
try:
    x = max([1, 2, 'a'])
except TypeError as e:
    x = 'typeerror'
except ValueError:
    x = 'valueerror'
except (ZeroDivisionError, IOError) as e:
    x = e
except pd.io.common.CParserError as e:
    x = 'CParserError'
except :
    x = 'someerror'
else :
    passed = False
```

No output

SCT

```
Ex().check_try_except().multi(
    check_body().check_function('max', signature = False).check_args(0).has_equal_
↪value(),
    check_handlers('TypeError').has_equal_value(name = 'x'),
    check_handlers('ValueError').has_equal_value(name = 'x'),
    check_handlers('ZeroDivisionError').set_context(e = 'anerror').has_equal_
↪value(name = 'x'),
    check_handlers('IOError').set_context(e = 'anerror').has_equal_value(name = 'x'),
    check_handlers('CParserError').has_equal_value(name = 'x'),
    check_handlers('all').has_equal_value(name = 'x'),
    check_orelse().has_equal_value(name = 'passed'),
    check_finalbody().check_function('print').check_args(0).has_equal_value()
)
```

Result

```
Check the first try statement. Did you correctly specify the else part? Are you sure␣
↪you assigned the correct value to <code>passed</code>?
```

No error

13.9.14 Example 14

PEC

```
import numpy as np
```

Solution code

```
try:
    x = max([1, 2, 'a'])
except TypeError as e:
    x = 'typeerror'
except __builtin__.ValueError:
    x = 'valueerror'
except (ZeroDivisionError, IOError) as e:
    x = e
except pd.io.common.CParserError:
    x = 'CParserError'
except :
    x = 'someerror'
else :
    passed = True
finally:
    print('done')
```

Student code

```
try:
    x = max([1, 2, 'a'])
except TypeError as e:
    x = 'typeerror'
except ValueError:
    x = 'valueerror'
except (ZeroDivisionError, IOError) as e:
```

(continues on next page)

(continued from previous page)

```
    x = e
except pd.io.common.CParserError as e:
    x = 'CParserError'
except :
    x = 'someerror'
else :
    passed = True
```

No output

SCT

```
Ex().check_try_except().multi(
    check_body().check_function('max', signature = False).check_args(0).has_equal_
↪value(),
    check_handlers('TypeError').has_equal_value(name = 'x'),
    check_handlers('ValueError').has_equal_value(name = 'x'),
    check_handlers('ZeroDivisionError').set_context(e = 'anerror').has_equal_
↪value(name = 'x'),
    check_handlers('IOError').set_context(e = 'anerror').has_equal_value(name = 'x'),
    check_handlers('CParserError').has_equal_value(name = 'x'),
    check_handlers('all').has_equal_value(name = 'x'),
    check_orelse().has_equal_value(name = 'passed'),
    check_finalbody().check_function('print').check_args(0).has_equal_value()
)
```

Result

```
Check the first try statement. Are you sure you defined the finally part?
```

No error

13.9.15 Example 15

PEC

```
import numpy as np
```

Solution code

```
try:
    x = max([1, 2, 'a'])
except TypeError as e:
    x = 'typeerror'
except __builtin__.ValueError:
    x = 'valueerror'
except (ZeroDivisionError, IOError) as e:
    x = e
except pd.io.common.CParserError:
    x = 'CParserError'
except :
    x = 'someerror'
else :
    passed = True
finally:
    print('done')
```

Student code

```
try:
    x = max([1, 2, 'a'])
except TypeError as e:
    x = 'typeerror'
except ValueError:
    x = 'valueerror'
except (ZeroDivisionError, IOError) as e:
    x = e
except pd.io.common.CParserError as e:
    x = 'CParserError'
except :
    x = 'someerror'
else :
    passed = True
finally:
    print('donessss')
```

Student output

```
donessss
```

SCT

```
Ex().check_try_except().multi(
    check_body().check_function('max', signature = False).check_args(0).has_equal_
↪value(),
    check_handlers('TypeError').has_equal_value(name = 'x'),
    check_handlers('ValueError').has_equal_value(name = 'x'),
    check_handlers('ZeroDivisionError').set_context(e = 'anerror').has_equal_
↪value(name = 'x'),
    check_handlers('IOError').set_context(e = 'anerror').has_equal_value(name = 'x'),
    check_handlers('CParserError').has_equal_value(name = 'x'),
    check_handlers('all').has_equal_value(name = 'x'),
    check_orelse().has_equal_value(name = 'passed'),
    check_finalbody().check_function('print').check_args(0).has_equal_value()
)
```

Result

```
Check your call of <code>print()</code>. Did you correctly specify the first argument?
↪ Expected <code>done</code>, but got <code>donessss</code>.
```

No error

13.9.16 Example 16

PEC

```
import numpy as np
```

Solution code

```
try:
    x = max([1, 2, 'a'])
```

(continues on next page)

(continued from previous page)

```
except TypeError as e:
    x = 'typeerror'
except __builtin__.ValueError:
    x = 'valueerror'
except (ZeroDivisionError, IOError) as e:
    x = e
except pd.io.common.CParserError:
    x = 'CParserError'
except :
    x = 'someerror'
else :
    passed = True
finally:
    print('done')
```

Student code

```
try:
    x = max([1, 2, 'a'])
except TypeError as e:
    x = 'typeerror'
except __builtin__.ValueError:
    x = 'valueerror'
except (ZeroDivisionError, IOError) as e:
    x = e
except pd.io.common.CParserError:
    x = 'CParserError'
except :
    x = 'someerror'
else :
    passed = True
finally:
    print('done')
```

Student output

```
done
```

SCT

```
Ex().check_try_except().multi(
    check_body().check_function('max', signature = False).check_args(0).has_equal_
↪value(),
    check_handlers('TypeError').has_equal_value(name = 'x'),
    check_handlers('ValueError').has_equal_value(name = 'x'),
    check_handlers('ZeroDivisionError').set_context(e = 'anerror').has_equal_
↪value(name = 'x'),
    check_handlers('IOError').set_context(e = 'anerror').has_equal_value(name = 'x'),
    check_handlers('CParserError').has_equal_value(name = 'x'),
    check_handlers('all').has_equal_value(name = 'x'),
    check_orelse().has_equal_value(name = 'passed'),
    check_finalbody().check_function('print').check_args(0).has_equal_value()
)
```

Result

```
Great work!
```

No error

13.10 test_test_compound_statement.py

13.10.1 Example 1

No PEC

Solution code

```
for i in range(3): print(i)
```

No student code

No output

SCT

```
test_for_loop(for_iter=lambda: test_expression_result(), body=lambda: test_expression_
↪output())
```

Result

```
The system wants to check the first for loop but hasn't found it.
```

No error

13.10.2 Example 2

No PEC

Solution code

```
for i in range(3): print(i)
```

No student code

No output

SCT

```
Ex().test_for_loop(for_iter=lambda: test_expression_result(), body=lambda: test_
↪expression_output())
```

Result

```
The system wants to check the first for loop but hasn't found it.
```

No error

13.10.3 Example 3

No PEC

Solution code

```
for i in range(3): print(i)
```

No student code

No output

SCT

```
Ex().check_for_loop().multi(check_iter().has_equal_value(), check_body().has_equal_
↪output())
```

Result

```
The system wants to check the first for loop but hasn't found it.
```

No error

13.10.4 Example 4

No PEC

Solution code

```
for i in range(3): print(i)
```

Student code

```
for i in range(4): pass
```

No output

SCT

```
test_for_loop(for_iter=lambda: test_expression_result(), body=lambda: test_expression_
↪output())
```

Result

```
Check the first for loop. Did you correctly specify the sequence part? Expected <code>
↪range(0, 3)</code>, but got <code>range(0, 4)</code>.
```

No error

13.10.5 Example 5

No PEC

Solution code

```
for i in range(3): print(i)
```

Student code

```
for i in range(4): pass
```

No output

SCT

```
Ex().test_for_loop(for_iter=lambda: test_expression_result(), body=lambda: test_
↪expression_output())
```

Result

```
Expected something different.
```

No error

13.10.6 Example 6

No PEC

Solution code

```
for i in range(3): print(i)
```

Student code

```
for i in range(4): pass
```

No output

SCT

```
Ex().check_for_loop().multi(check_iter().has_equal_value(), check_body().has_equal_
↪output())
```

Result

```
Check the first for loop. Did you correctly specify the iterable part? Expected <code>
↪range(0, 3)</code>, but got <code>range(0, 4)</code>.
```

No error

13.10.7 Example 7

No PEC

Solution code

```
for i in range(3): print(i)
```

Student code

```
for i in range(3): pass
```

No output

SCT

```
test_for_loop(for_iter=lambda: test_expression_result(), body=lambda: test_expression_
↪output())
```

Result

```
Check the first for loop. Did you correctly specify the body? Expected the output
↪<code>2</code>, but got <code>no printouts</code>.
```

No error

13.10.8 Example 8

No PEC

Solution code

```
for i in range(3): print(i)
```

Student code

```
for i in range(3): pass
```

No output

SCT

```
Ex().test_for_loop(for_iter=lambda: test_expression_result(), body=lambda: test_
↪expression_output())
```

Result

```
Expected something different.
```

No error

13.10.9 Example 9

No PEC

Solution code

```
for i in range(3): print(i)
```

Student code

```
for i in range(3): pass
```

No output

SCT

```
Ex().check_for_loop().multi(check_iter().has_equal_value(), check_body().has_equal_
↪output())
```

Result

```
Check the first for loop. Did you correctly specify the body? Expected the output
↪<code>2</code>, but got <code>no printouts</code>.
```

No error

13.10.10 Example 10

No PEC

Solution code

```
for i in range(3): print(i)
```

Student code

```
for i in range(3): print(i)
```

Student output

```
0
1
2
```

SCT

```
test_for_loop(for_iter=lambda: test_expression_result(), body=lambda: test_expression_
↪output())
```

Result

```
Great work!
```

No error

13.10.11 Example 11

No PEC

Solution code

```
for i in range(3): print(i)
```

Student code

```
for i in range(3): print(i)
```

Student output

```
0
1
2
```

SCT

```
Ex().test_for_loop(for_iter=lambda: test_expression_result(), body=lambda: test_
↪expression_output())
```

Result

```
Great work!
```

No error

13.10.12 Example 12

No PEC

Solution code

```
for i in range(3): print(i)
```

Student code

```
for i in range(3): print(i)
```

Student output

```
0
1
2
```

SCT

```
Ex().check_for_loop().multi(check_iter().has_equal_value(), check_body().has_equal_
↪output())
```

Result

```
Great work!
```

No error

13.10.13 Example 13

No PEC

Solution code

```
for i in range(3): print(i)
```

Student code

```
for j in range(3): print(j)
```

Student output

```
0
1
2
```

SCT

```
test_for_loop(for_iter=lambda: test_expression_result(), body=lambda: test_expression_
↪output())
```

Result

```
Great work!
```

No error

13.10.14 Example 14

No PEC

Solution code

```
for i in range(3): print(i)
```

Student code

```
for j in range(3): print(j)
```

Student output

```
0
1
2
```

SCT

```
Ex().test_for_loop(for_iter=lambda: test_expression_result(), body=lambda: test_
↪expression_output())
```

Result

```
Great work!
```

No error

13.10.15 Example 15

No PEC

Solution code

```
for i in range(3): print(i)
```

Student code

```
for j in range(3): print(j)
```

Student output

```
0
1
2
```

SCT

```
Ex().check_for_loop().multi(check_iter().has_equal_value(), check_body().has_equal_
↪output())
```

Result

```
Great work!
```

No error

13.10.16 Example 16

No PEC

Solution code

```
while 3 > 4: print(2)
```

No student code

No output

SCT

```
test_while_loop(test = lambda: test_student_typed('3'), body = lambda: test_student_
↪typed('print'))
```

Result

```
The system wants to check the first while loop but hasn't found it.
```

No error

13.10.17 Example 17

No PEC

Solution code

```
while 3 > 4: print(2)
```

No student code

No output

SCT

```
Ex().test_while_loop(test = test_student_typed('3'), body = test_student_typed('print
↪'))
```

Result

```
The system wants to check the first while loop but hasn't found it.
```

No error

13.10.18 Example 18

No PEC

Solution code

```
while 3 > 4: print(2)
```

No student code

No output

SCT

```
Ex().check_while().multi(check_test().has_code('3'), check_body().has_code('print'))
```

Result

```
The system wants to check the first <code>while</code> loop but hasn't found it.
```

No error

13.10.19 Example 19

No PEC

Solution code

```
while 3 > 4: print(2)
```

Student code

```
while False: pass
```

No output

SCT

```
test_while_loop(test = lambda: test_student_typed('3'), body = lambda: test_student_
↪typed('print'))
```

Result

```
Check the first while loop. Did you correctly specify the condition? Could not find␣
↪the correct pattern in your code.
```

No error

13.10.20 Example 20

No PEC

Solution code

```
while 3 > 4: print(2)
```

Student code

```
while False: pass
```

No output

SCT

```
Ex().test_while_loop(test = test_student_typed('3'), body = test_student_typed('print
↪'))
```

Result

```
Check the first while loop. Did you correctly specify the condition? Could not find␣
↪the correct pattern in your code.
```

No error

13.10.21 Example 21

No PEC

Solution code

```
while 3 > 4: print(2)
```

Student code

```
while False: pass
```

No output

SCT

```
Ex().check_while().multi(check_test().has_code('3'), check_body().has_code('print'))
```

Result

```
Check the first <code>while</code> loop. Did you correctly specify the condition?␣
↪Could not find the correct pattern in your code.
```

No error

13.10.22 Example 22

No PEC

Solution code

```
while 3 > 4: print(2)
```

Student code

```
while 3 > 4: pass
```

No output

SCT

```
test_while_loop(test = lambda: test_student_typed('3'), body = lambda: test_student_
↪typed('print'))
```

Result

```
Check the first while loop. Did you correctly specify the body? Could not find the␣
↪correct pattern in your code.
```

No error

13.10.23 Example 23

No PEC

Solution code

```
while 3 > 4: print(2)
```

Student code

```
while 3 > 4: pass
```

No output

SCT

```
Ex().test_while_loop(test = test_student_typed('3'), body = test_student_typed('print
↪'))
```

Result

```
Check the first while loop. Did you correctly specify the body? Could not find the␣
↪correct pattern in your code.
```

No error

13.10.24 Example 24

No PEC

Solution code

```
while 3 > 4: print(2)
```

Student code

```
while 3 > 4: pass
```

No output

SCT

```
Ex().check_while().multi(check_test().has_code('3'), check_body().has_code('print'))
```

Result

```
Check the first <code>while</code> loop. Did you correctly specify the body? Could␣
↪not find the correct pattern in your code.
```

No error

13.10.25 Example 25

No PEC

Solution code

```
while 3 > 4: print(2)
```

Student code

```
while 3 > 4: print(2)
```

No output

SCT

```
test_while_loop(test = lambda: test_student_typed('3'), body = lambda: test_student_
↪typed('print'))
```

Result

```
Great work!
```

No error

13.10.26 Example 26

No PEC

Solution code

```
while 3 > 4: print(2)
```

Student code

```
while 3 > 4: print(2)
```

No output

SCT

```
Ex().test_while_loop(test = test_student_typed('3'), body = test_student_typed('print
↪'))
```

Result

```
Great work!
```

No error

13.10.27 Example 27

No PEC

Solution code

```
while 3 > 4: print(2)
```

Student code

```
while 3 > 4: print(2)
```

No output

SCT

```
Ex().check_while().multi(check_test().has_code('3'), check_body().has_code('print'))
```

Result

```
Great work!
```

No error

13.11 test_highlighting.py

13.11.1 Example 1

No PEC

Solution code

```
round(2.345, 2)
```

Student code

```
round(1.234, 2)
```

No output

SCT

```
Ex().check_function("round").check_args("number").has_equal_value()
```

Result

```
Check your call of <code>round()</code>. Did you correctly specify the argument <code>
↪number</code>? Expected <code>2.345</code>, but got <code>1.234</code>.
```

No error

13.11.2 Example 2

No PEC

Solution code

```
round(2.345, 2)
```

Student code

```
round(1.234, 2)
```

No output

SCT

```
Ex().disable_highlighting().check_function("round").check_args("number").has_equal_
↪value()
```

Result

```
Check your call of <code>round()</code>. Did you correctly specify the argument <code>
↪number</code>? Expected <code>2.345</code>, but got <code>1.234</code>.
```

No error

13.11.3 Example 3

No PEC

Solution code

```
round(2.345, 2)
```

Student code

```
round(1.234, 2)
```

No output

SCT

```
Ex().check_function("round").disable_highlighting().check_args("number").has_equal_
↪value()
```

Result

```
Check your call of <code>round()</code>. Did you correctly specify the argument <code>
↪number</code>? Expected <code>2.345</code>, but got <code>1.234</code>.
```

No error

13.11.4 Example 4

No PEC

Solution code

```
round(2.345, 2)
```

Student code

```
round(1.234, 2)
```

No output

SCT

```
Ex().check_function("round").check_args("number").disable_highlighting().has_equal_
↪value()
```

Result

```
Check your call of <code>round()</code>. Did you correctly specify the argument <code>
↪number</code>? Expected <code>2.345</code>, but got <code>1.234</code>.
```

No error

13.11.5 Example 5

No PEC

Solution code

```
round(1)
round(2)
```

Student code

```
round(1)
round(3)
```

No output

SCT

```
Ex().check_function("round", index=0).check_args("number").has_equal_value()
Ex().check_function("round", index=1).check_args("number").has_equal_value()
```

Result

```
Check your second call of <code>round()</code>. Did you correctly specify the␣
↪argument <code>number</code>? Expected <code>2</code>, but got <code>3</code>.
```

No error

13.11.6 Example 6

No PEC

Solution code

```
round(1)
round(2)
```

Student code

```
round(3)
round(2)
```

No output

SCT

```
Ex().check_function("round", index=0).check_args("number").has_equal_value()
Ex().check_function("round", index=1).check_args("number").has_equal_value()
```

Result

```
Check your call of <code>round()</code>. Did you correctly specify the argument <code>
↪number</code>? Expected <code>1</code>, but got <code>3</code>.
```

No error

13.11.7 Example 7

No PEC

Solution code

```
round(1)
round(2)
```

Student code

```
round(3)
round(1)
```

No output

SCT

```
Ex().check_function("round", index=0).check_args("number").has_equal_value()
Ex().check_function("round", index=1).check_args("number").has_equal_value()
```

Result

```
Check your call of <code>round()</code>. Did you correctly specify the argument <code>
↪number</code>? Expected <code>1</code>, but got <code>3</code>.
```

No error

13.11.8 Example 8

No PEC

Solution code

```
round(1)
round(2)
```

Student code

```
round(2)
round(3)
```

No output

SCT

```
Ex().check_function("round", index=0).check_args("number").has_equal_value()
Ex().check_function("round", index=1).check_args("number").has_equal_value()
```

Result

```
Check your call of <code>round()</code>. Did you correctly specify the argument <code>
↪number</code>? Expected <code>1</code>, but got <code>2</code>.
```

No error

13.11.9 Example 9

No PEC

Solution code

```
round(1)
round(2)
```

Student code

```
round(1)
round(3)
```

No output

SCT

```
Ex().check_function("round", index=1).check_args("number").has_equal_value()
```

Result

```
Check your second call of <code>round()</code>. Did you correctly specify the␣
↪argument <code>number</code>? Expected <code>2</code>, but got <code>3</code>.
```

No error

13.12 test_docs.py

13.12.1 Example 1

No PEC

Solution code

```
x = 4
if x > 0:
    print("x is strictly positive")
```

Student code

```
x = 4
if x > 0:
    print("x is strictly positive")
```

Student output

```
x is strictly positive
```

SCT

```
Ex().check_if_else().multi(
    check_test().multi(
        set_env(x = -1).has_equal_value(), # chain A1
        set_env(x =  1).has_equal_value(), # chain A2
        set_env(x =  0).has_equal_value()  # chain A3
    ),
    check_body().check_function('print', 0).check_args('value').has_equal_value() #␣
↪chain B
)
```

Result

```
Great work!
```

No error

13.12.2 Example 2

No PEC

Solution code

```
x = 4
if x > 0:
    print("x is strictly positive")
```

Student code

```
x = 4
if 0 < x:
    print("x is strictly positive")
```

Student output

```
x is strictly positive
```

SCT

```
Ex().check_if_else().multi(
    check_test().multi(
        set_env(x = -1).has_equal_value(), # chain A1
        set_env(x =  1).has_equal_value(), # chain A2
        set_env(x =  0).has_equal_value()  # chain A3
    ),
```

(continues on next page)

(continued from previous page)

```
    check_body().check_function('print', 0).check_args('value').has_equal_value() #␣
↪chain B
)
```

Result

```
Great work!
```

No error

13.12.3 Example 3

No PEC

Solution code

```
x = 4
if x > 0:
    print("x is strictly positive")
```

Student code

```
x = 4
if x >= 0:
    print("x is strictly positive")
```

Student output

```
x is strictly positive
```

SCT

```
Ex().check_if_else().multi(
    check_test().multi(
        set_env(x = -1).has_equal_value(), # chain A1
        set_env(x =  1).has_equal_value(), # chain A2
        set_env(x =  0).has_equal_value()  # chain A3
    ),
    check_body().check_function('print', 0).check_args('value').has_equal_value() #␣
↪chain B
)
```

Result

```
Check the first if statement. Did you correctly specify the condition? Expected <code>
↪False</code>, but got <code>True</code>.
```

No error

13.12.4 Example 4

No PEC

Solution code

```
my_dict = {'a': 1, 'b': 2}
for key, value in my_dict.items():
    print(key + " - " + str(value))
```

Student code

```
my_dict = {'a': 1, 'b': 2}
for key, value in my_dict.items():
    print(key + " - " + str(value))
```

Student output

```
b - 2
a - 1
```

SCT

```
Ex().check_object('my_dict').has_equal_value()
Ex().check_for_loop().multi(
    check_iter().has_equal_value(),
    check_body().multi(
        set_context('a', 1).has_equal_output(),
        set_context('b', 2).has_equal_output()
    )
)
```

Result

```
Great work!
```

No error

13.12.5 Example 5

No PEC

Solution code

```
my_dict = {'a': 1, 'b': 2}
for key, value in my_dict.items():
    print(key + " - " + str(value))
```

Student code

```
my_dict = {'a': 1, 'b': 2}
for k, v in my_dict.items():
    print(k + ' - ' + str(v))
```

Student output

```
b - 2
a - 1
```

SCT

```
Ex().check_object('my_dict').has_equal_value()
Ex().check_for_loop().multi(
    check_iter().has_equal_value(),
    check_body().multi(
        set_context('a', 1).has_equal_output(),
        set_context('b', 2).has_equal_output()
    )
)
```

Result

```
Great work!
```

No error

13.12.6 Example 6

No PEC

Solution code

```
my_dict = {'a': 1, 'b': 2}
for key, value in my_dict.items():
    print(key + " - " + str(value))
```

Student code

```
my_dict = {'a': 1, 'b': 2}
for first, second in my_dict.items():
    mess = first + ' - ' + str(second)
    print(mess)
```

Student output

```
b - 2
a - 1
```

SCT

```
Ex().check_object('my_dict').has_equal_value()
Ex().check_for_loop().multi(
    check_iter().has_equal_value(),
    check_body().multi(
        set_context('a', 1).has_equal_output(),
        set_context('b', 2).has_equal_output()
    )
)
```

Result

```
Great work!
```

No error

13.12.7 Example 7

No PEC

Solution code

```
def shout_echo(word1, echo=1):
    echo_word = word1 * echo
    shout_words = echo_word + '!!!'
    return shout_words
```

Student code

```
def shout_echo(word1, echo=1):
    echo_word = word1 * echo
    shout_words = echo_word + '!!!'
    return shout_words
```

No output

SCT

```
Ex().check_function_def('shout_echo').test_correct(
    multi(
        check_call("f('hey', 3)").has_equal_value(),
        check_call("f('hi', 2)").has_equal_value(),
        check_call("f('hi')").has_equal_value()
    ),
    check_body().set_context('test', 1).multi(
        has_equal_value(name = 'echo_word'),
        has_equal_value(name = 'shout_words')
    )
)
```

Result

```
Great work!
```

No error

13.12.8 Example 8

No PEC

Solution code

```
def shout_echo(word1, echo=1):
    echo_word = word1 * echo
    shout_words = echo_word + '!!!'
    return shout_words
```

Student code

```
def shout_echo(w, e=1):
    ew = w * e
    return ew + '!!!'
```

No output

SCT

```
Ex().check_function_def('shout_echo').test_correct(
    multi(
        check_call("f('hey', 3)").has_equal_value(),
        check_call("f('hi', 2)").has_equal_value(),
        check_call("f('hi')").has_equal_value()
    ),
    check_body().set_context('test', 1).multi(
        has_equal_value(name = 'echo_word'),
        has_equal_value(name = 'shout_words')
    )
)
```

Result

```
Great work!
```

No error

13.12.9 Example 9

No PEC

Solution code

```
def shout_echo(word1, echo=1):
    echo_word = word1 * echo
    shout_words = echo_word + '!!!'
    return shout_words
```

Student code

```
def shout_echo(a, b=1):
    return a * b + '!!!'
```

No output

SCT

```
Ex().check_function_def('shout_echo').test_correct(
    multi(
        check_call("f('hey', 3)").has_equal_value(),
        check_call("f('hi', 2)").has_equal_value(),
        check_call("f('hi')").has_equal_value()
    ),
    check_body().set_context('test', 1).multi(
        has_equal_value(name = 'echo_word'),
        has_equal_value(name = 'shout_words')
    )
)
```

Result

```
Great work!
```

No error

13.12.10 Example 10

No PEC

Solution code

```
def counter(lst, key):
    count = 0
    for l in lst:
        count += l[key]
    return count
```

Student code

```
def counter(lst, key):
    count = 0
    for l in lst:
        count += l[key]
    return count
```

No output

SCT

```
Ex().check_function_def('counter').test_correct(
    multi(
        check_call("f([{'a': 1}], 'a')").has_equal_value(),
        check_call("f([{'b': 1}, {'b': 2}], 'b')").has_equal_value()
    ),
    check_body().set_context([{'a': 1}, {'a': 2}], 'a').set_env(count = 0).check_for_
↪loop().multi(
        check_iter().has_equal_value(),
        check_body().set_context({'a': 1}).has_equal_value(name = 'count')
    )
)
```

Result

```
Great work!
```

No error

13.13 test_converters.py

13.13.1 Example 1

No PEC

Solution code

```
x = {'a': 1, 'b': 2}; print(x.keys())
```

Student code

```
x = {'b':2, 'a': 1}; print(x.keys())
```

Student output

```
dict_keys(['b', 'a'])
```

SCT

```
Ex().check_function('print').check_args('value').has_equal_value()
```

Result

```
Great work!
```

No error

13.13.2 Example 2

No PEC

Solution code

```
x = {'france':'paris', 'spain':'madrid'}.items()
```

Student code

```
x = {'spain':'madrid', 'france':'paris'}.items()
```

No output

SCT

```
Ex().check_object('x').has_equal_value()
```

Result

```
Great work!
```

No error

13.13.3 Example 3

PEC

```
import requests; from bs4 import BeautifulSoup
```

Solution code

```
soup = BeautifulSoup(requests.get('https://www.python.org/~guido/').text, 'html5lib');
↪ print(soup.title); a_tags = soup.find_all('a')
```

Student code

```
soup = BeautifulSoup(requests.get('https://www.python.org/~guido/').text, 'html5lib');
↪ print(soup.title); a_tags = soup.find_all('a')
```

Student output

```
<title>Guido's Personal Home Page</title>
```

SCT

```
Ex().check_object('soup').has_equal_value(); Ex().check_function('print').check_args(
↪'value').has_equal_value(); Ex().check_object('a_tags').has_equal_value()
```

Result

```
Great work!
```

No error

13.13.4 Example 4

PEC

```
import numpy as np; import h5py; file = 'LIGO_data.hdf5'; from urllib.request import␣
↪urlretrieve; urlretrieve('https://s3.amazonaws.com/assets.datacamp.com/production/
↪course_998/datasets/L-L1_LOSC_4_V1-1126259446-32.hdf5', 'LIGO_data.hdf5')
```

Solution code

```
data = h5py.File(file, 'r'); group = data['strain']
```

Student code

```
data = h5py.File(file, 'r'); group = data['strain']
```

No output

SCT

```
Ex().check_object('file').has_equal_value(); Ex().check_object('data').has_equal_
↪value(); Ex().check_object('group').has_equal_value()
```

Result

```
Great work!
```

No error

13.14 test_has_no_error.py

13.14.1 Example 1

No PEC

No solution code

Student code

```
c
```

No output

SCT

```
Ex().has_no_error()
```

Result

```
Have a look at the console: your code contains an error. Fix it and try again!
```

Error

```
name 'c' is not defined
```

13.14.2 Example 2

No PEC

No solution code

Student code

```
a = 3
```

No output

SCT

```
Ex().has_no_error()
```

Result

```
Great work!
```

No error

13.15 test_messaging.py

13.15.1 Example 1

No PEC

Solution code

```
round(1, 3)
```

No student code

No output

SCT

```
Ex().check_function("round").check_args(1).has_equal_value()
```

Result

```
Did you call <code>round()</code>?
```

No error

13.15.2 Example 2

No PEC

Solution code

```
round(1, 3)
```

Student code

```
round(1)
```

No output

SCT

```
Ex().check_function("round").check_args(1).has_equal_value()
```

Result

```
Check your call of <code>round()</code>. Did you specify the second argument?
```

No error

13.15.3 Example 3

No PEC

Solution code

```
round(1, 3)
```

Student code

```
round(1, a)
```

No output

SCT

```
Ex().check_function("round").check_args(1).has_equal_value()
```

Result

```
Check your call of <code>round()</code>. Did you correctly specify the second␣
↪argument? Running it generated an error: <code>name 'a' is not defined</code>.
```

Error

```
name 'a' is not defined
```

13.15.4 Example 4

No PEC

Solution code

```
round(1, 3)
```

Student code

```
round(1, 2)
```

No output

SCT

```
Ex().check_function("round").check_args(1).has_equal_value()
```

Result

```
Check your call of <code>round()</code>. Did you correctly specify the second␣
↪argument? Expected <code>3</code>, but got <code>2</code>.
```

No error

13.15.5 Example 5

No PEC

Solution code

```
round(1, 3)
```

Student code

```
round(1, ndigits = 2)
```

No output

SCT

```
Ex().check_function("round").check_args(1).has_equal_value()
```

Result

```
Check your call of <code>round()</code>. Did you correctly specify the second␣
↪argument? Expected <code>3</code>, but got <code>2</code>.
```

No error

13.15.6 Example 6

No PEC

Solution code

```
round(1, 3)
```

Student code

```
round(1)
```

No output

SCT

```
Ex().check_function("round").check_args("ndigits").has_equal_value()
```

Result

```
Check your call of <code>round()</code>. Did you specify the argument <code>ndigits</
↪code>?
```

No error

13.15.7 Example 7

No PEC

Solution code

```
round(1, 3)
```

Student code

```
round(1, a)
```

No output

SCT

```
Ex().check_function("round").check_args("ndigits").has_equal_value()
```

Result

```
Check your call of <code>round()</code>. Did you correctly specify the argument <code>
↪ndigits</code>? Running it generated an error: <code>name 'a' is not defined</code>.
```

Error

```
name 'a' is not defined
```

13.15.8 Example 8

No PEC

Solution code

```
round(1, 3)
```

Student code

```
round(1, 2)
```

No output

SCT

```
Ex().check_function("round").check_args("ndigits").has_equal_value()
```

Result

```
Check your call of <code>round()</code>. Did you correctly specify the argument <code>
↪ndigits</code>? Expected <code>3</code>, but got <code>2</code>.
```

No error

13.15.9 Example 9

No PEC

Solution code

```
round(2)
```

Student code

```
round(3)
```

No output

SCT

```
Ex().check_function("round").check_args(0).has_equal_ast()
```

Result

```
Check your call of <code>round()</code>. Did you correctly specify the first argument?
↪ Expected <code>2</code>, but got <code>3</code>.
```

No error

13.15.10 Example 10

No PEC

Solution code

```
round(2)
```

Student code

```
round(1 + 1)
```

No output

SCT

```
Ex().check_function("round").check_args(0).has_equal_ast()
```

Result

```
Check your call of <code>round()</code>. Did you correctly specify the first argument?
↪ Expected <code>2</code>, but got <code>1 + 1</code>.
```

No error

13.15.11 Example 11

No PEC

Solution code

```
list("test")
```

Student code

```
list("wrong")
```

No output

SCT

```
Ex().check_function("list", signature = False).check_args(0).has_equal_ast()
```

Result

```
Check your call of <code>list()</code>. Did you correctly specify the first argument?␣
↪Expected <code>"test"</code>, but got <code>"wrong"</code>.
```

No error

13.15.12 Example 12

No PEC

Solution code

```
list("test")
```

Student code

```
list("te" + "st")
```

No output

SCT

```
Ex().check_function("list", signature = False).check_args(0).has_equal_ast()
```

Result

```
Check your call of <code>list()</code>. Did you correctly specify the first argument?␣
↪Expected <code>"test"</code>, but got <code>"te" + "st"</code>.
```

No error

13.15.13 Example 13

PEC

```
a = 3
b=3
```

Solution code

```
round(b)
```

Student code

```
round(a)
```

No output

SCT

```
Ex().check_function("round", signature = False).check_args(0).has_equal_ast()
```

Result

```
Check your call of <code>round()</code>. Did you correctly specify the first argument?
↪ Expected <code>b</code>, but got <code>a</code>.
```

No error

13.15.14 Example 14

PEC

```
a = 3
b=3
```

Solution code

```
round(b)
```

Student code

```
round(b + 1 - 1)
```

No output

SCT

```
Ex().check_function("round", signature = False).check_args(0).has_equal_ast()
```

Result

```
Check your call of <code>round()</code>. Did you correctly specify the first argument?
↪ Expected <code>b</code>, but got <code>b + 1 - 1</code>.
```

No error

13.15.15 Example 15

No PEC

Solution code

```
import pandas as pd; pd.DataFrame({'a': [1, 2, 3]})
```

Student code

```
import pandas as pd
```

No output

SCT

```
test_function_v2('pandas.DataFrame')
```

Result

```
Did you call <code>pd.DataFrame()</code>?
```

No error

13.15.16 Example 16

No PEC

Solution code

```
import pandas as pd; pd.DataFrame({'a': [1, 2, 3]})
```

Student code

```
import pandas as pad
```

No output

SCT

```
test_function_v2('pandas.DataFrame')
```

Result

```
Did you call <code>pad.DataFrame()</code>?
```

No error

13.15.17 Example 17

No PEC

Solution code

```
import numpy as np; x = np.random.rand(1)
```

Student code

```
import numpy as nump
```

No output

SCT

```
test_function_v2('numpy.random.rand')
```

Result

```
Did you call <code>nump.random.rand()</code>?
```

No error

13.15.18 Example 18

No PEC

Solution code

```
import numpy as np; x = np.random.rand(1)
```

Student code

```
from numpy.random import rand as r
```

No output

SCT

```
test_function_v2('numpy.random.rand')
```

Result

```
Did you call <code>r()</code>?
```

No error

13.15.19 Example 19

No PEC

Solution code

```
round(1)
round(2)
round(3)
```

No student code

No output

SCT

```
Ex().multi([ check_function("round", index=i).check_args(0).has_equal_value() for i␣
↪in range(3) ])
```

Result

```
Did you call <code>round()</code>?
```

No error

13.15.20 Example 20

No PEC

Solution code

```
round(1)
round(2)
round(3)
```

Student code

```
round(1)
```

No output

SCT

```
Ex().multi([ check_function("round", index=i).check_args(0).has_equal_value() for i␣
↪in range(3) ])
```

Result

```
Did you call <code>round()</code> twice?
```

No error

13.15.21 Example 21

No PEC

Solution code

```
round(1)
round(2)
round(3)
```

Student code

```
round(1)
round(5)
```

No output

SCT

```
Ex().multi([ check_function("round", index=i).check_args(0).has_equal_value() for i␣
↪in range(3) ])
```

Result

```
Check your second call of <code>round()</code>. Did you correctly specify the first␣
↪argument? Expected <code>2</code>, but got <code>5</code>.
```

No error

13.15.22 Example 22

No PEC

Solution code

```
round(1)
round(2)
round(3)
```

Student code

```
round(1)
round(2)
```

No output

SCT

```
Ex().multi([ check_function("round", index=i).check_args(0).has_equal_value() for i␣
↪in range(3) ])
```

Result

```
Did you call <code>round()</code> three times?
```

No error

13.15.23 Example 23

No PEC

Solution code

```
round(1)
round(2)
round(3)
```

Student code

```
round(1)
round(2)
round(5)
```

No output

SCT

```
Ex().multi([ check_function("round", index=i).check_args(0).has_equal_value() for i␣
↪in range(3) ])
```

Result

```
Check your third call of <code>round()</code>. Did you correctly specify the first␣
↪argument? Expected <code>3</code>, but got <code>5</code>.
```

No error

13.15.24 Example 24

PEC

```
import pandas as pd; df = pd.DataFrame({'a': [1, 2, 3], 'b': ['x', 'x', 'y']})
```

Solution code

```
df.groupby('b').a.value_counts(normalize = True)
```

Student code

```
df.groupby('a')
```

No output

SCT

```
from pythonwhat.signatures import sig_from_obj
import pandas as pd
Ex().check_function('df.groupby').check_args(0).has_equal_ast()
Ex().check_function('df.groupby.a.value_counts', signature = sig_from_obj(pd.Series.
↪value_counts)).check_args('normalize').has_equal_ast()
```

Result

```
Check your call of <code>df.groupby()</code>. Did you correctly specify the first␣
↪argument? Expected <code>'b'</code>, but got <code>'a'</code>.
```

No error

13.15.25 Example 25

PEC

```
import pandas as pd; df = pd.DataFrame({'a': [1, 2, 3], 'b': ['x', 'x', 'y']})
```

Solution code

```
df.groupby('b').a.value_counts(normalize = True)
```

Student code

```
df.groupby('b').a.value_counts()
```

No output

SCT

```
from pythonwhat.signatures import sig_from_obj
import pandas as pd
Ex().check_function('df.groupby').check_args(0).has_equal_ast()
Ex().check_function('df.groupby.a.value_counts', signature = sig_from_obj(pd.Series.
↪value_counts)).check_args('normalize').has_equal_ast()
```

Result

```
Check your call of <code>df.groupby.a.value_counts()</code>. Did you specify the␣
↪argument <code>normalize</code>?
```

No error

13.15.26 Example 26

PEC

```
import pandas as pd; df = pd.DataFrame({'a': [1, 2, 3], 'b': ['x', 'x', 'y']})
```

Solution code

```
df.groupby('b').a.value_counts(normalize = True)
```

Student code

```
df[df.b == 'x'].groupby('b').a.value_counts()
```

No output

SCT

```
from pythonwhat.signatures import sig_from_obj
import pandas as pd
Ex().check_function('df.groupby').check_args(0).has_equal_ast()
Ex().check_function('df.groupby.a.value_counts', signature = sig_from_obj(pd.Series.
↪value_counts)).check_args('normalize').has_equal_ast()
```

Result

```
Check your call of <code>df.groupby.a.value_counts()</code>. Did you specify the␣
↪argument <code>normalize</code>?
```

No error

13.15.27 Example 27

No PEC

Solution code

```
x = 5
```

No student code

No output

SCT

```
Ex().check_object("x").has_equal_value()
```

Result

```
Did you define the variable <code>x</code> without errors?
```

No error

13.15.28 Example 28

No PEC

Solution code

```
x = 5
```

Student code

```
x = 2
```

No output

SCT

```
Ex().check_object("x").has_equal_value()
```

Result

```
Did you correctly define the variable <code>x</code>? Expected <code>5</code>, but␣
↪got <code>2</code>.
```

No error

13.15.29 Example 29

PEC

```
import pandas as pd
```

Solution code

```
df = pd.DataFrame({"a": [1, 2, 3]})
```

Student code

```
df = 3
```

No output

SCT

```
test_data_frame('df', columns=['a'])
```

Result

```
Did you correctly define the pandas DataFrame <code>df</code>? Is it a DataFrame?
```

No error

13.15.30 Example 30

PEC

```
import pandas as pd
```

Solution code

```
df = pd.DataFrame({"a": [1, 2, 3]})
```

Student code

```
df = 3
```

No output

SCT

```
import pandas as pd; Ex().check_object('df', typestr = 'pandas DataFrame').is_
↪instance(pd.DataFrame).check_keys('a').has_equal_value()
```

Result

```
Did you correctly define the pandas DataFrame <code>df</code>? Is it a DataFrame?
```

No error

13.15.31 Example 31

PEC

```
import pandas as pd
```

Solution code

```
df = pd.DataFrame({"a": [1, 2, 3]})
```

Student code

```
df = pd.DataFrame({"b": [1]})
```

No output

SCT

```
test_data_frame('df', columns=['a'])
```

Result

```
Did you correctly define the pandas DataFrame <code>df</code>? There is no column
↪<code>'a'</code>.
```

No error

13.15.32 Example 32

PEC

```
import pandas as pd
```

Solution code

```
df = pd.DataFrame({"a": [1, 2, 3]})
```

Student code

```
df = pd.DataFrame({"b": [1]})
```

No output

SCT

```
import pandas as pd; Ex().check_object('df', typestr = 'pandas DataFrame').is_
↪instance(pd.DataFrame).check_keys('a').has_equal_value()
```

Result

```
Did you correctly define the pandas DataFrame <code>df</code>? There is no column
↪<code>'a'</code>.
```

No error

13.15.33 Example 33

PEC

```
import pandas as pd
```

Solution code

```
df = pd.DataFrame({"a": [1, 2, 3]})
```

Student code

```
df = pd.DataFrame({"a": [1]})
```

No output

SCT

```
test_data_frame('df', columns=['a'])
```

Result

```
Did you correctly define the pandas DataFrame <code>df</code>? Did you correctly set␣
↪the column <code>'a'</code>? Expected something different.
```

No error

13.15.34 Example 34

PEC

```
import pandas as pd
```

Solution code

```
df = pd.DataFrame({"a": [1, 2, 3]})
```

Student code

```
df = pd.DataFrame({"a": [1]})
```

No output

SCT

```
import pandas as pd; Ex().check_object('df', typestr = 'pandas DataFrame').is_
↪instance(pd.DataFrame).check_keys('a').has_equal_value()
```

Result

```
Did you correctly define the pandas DataFrame <code>df</code>? Did you correctly set␣
↪the column <code>'a'</code>? Expected something different.
```

No error

13.15.35 Example 35

PEC

```
import pandas as pd
```

Solution code

```
df = pd.DataFrame({"a": [1, 2, 3]})
```

Student code

```
y = 3; df = pd.DataFrame({"a": [1]})
```

No output

SCT

```
test_data_frame('df', columns=['a'])
```

Result

```
Did you correctly define the pandas DataFrame <code>df</code>? Did you correctly set␣
↪the column <code>'a'</code>? Expected something different.
```

No error

13.15.36 Example 36

PEC

```
import pandas as pd
```

Solution code

```
df = pd.DataFrame({"a": [1, 2, 3]})
```

Student code

```
y = 3; df = pd.DataFrame({"a": [1]})
```

No output

SCT

```
import pandas as pd; Ex().check_object('df', typestr = 'pandas DataFrame').is_
↪instance(pd.DataFrame).check_keys('a').has_equal_value()
```

Result

```
Did you correctly define the pandas DataFrame <code>df</code>? Did you correctly set␣
↪the column <code>'a'</code>? Expected something different.
```

No error

13.15.37 Example 37

No PEC

Solution code

```
x = {"a": 2}
```

Student code

```
x = {}
```

No output

SCT

```
Ex().check_object("x").check_keys("a").has_equal_value()
```

Result

```
Did you correctly define the variable <code>x</code>? There is no key <code>'a'</code>
↪.
```

No error

13.15.38 Example 38

No PEC

Solution code

```
x = {"a": 2}
```

Student code

```
x = {"b": 3}
```

No output

SCT

```
Ex().check_object("x").check_keys("a").has_equal_value()
```

Result

```
Did you correctly define the variable <code>x</code>? There is no key <code>'a'</code>
↪.
```

No error

13.15.39 Example 39

No PEC

Solution code

```
x = {"a": 2}
```

Student code

```
x = {"a": 3}
```

No output

SCT

```
Ex().check_object("x").check_keys("a").has_equal_value()
```

Result

```
Did you correctly define the variable <code>x</code>? Did you correctly set the key
↪<code>'a'</code>? Expected <code>2</code>, but got <code>3</code>.
```

No error

13.15.40 Example 40

No PEC

Solution code

```
x = {"a": 2}
```

Student code

```
y = 3; x = {"a": 3}
```

No output

SCT

```
Ex().check_object("x").check_keys("a").has_equal_value()
```

Result

```
Did you correctly define the variable <code>x</code>? Did you correctly set the key
↪<code>'a'</code>? Expected <code>2</code>, but got <code>3</code>.
```

No error

13.15.41 Example 41

No PEC

Solution code

```
x = round(1.23)
```

Student code

```
round(2.34)
```

No output

SCT

```
Ex().check_function('round').check_args(0).has_equal_value(incorrect_msg = 'argrwong')
Ex().check_object('x', missing_msg='objectnotdefined').has_equal_value(
↪'objectincorrect')
```

Result

```
argrwong
```

No error

13.15.42 Example 42

No PEC

Solution code

```
x = round(1.23)
```

Student code

```
round(1.23)
```

No output

SCT

```
Ex().check_function('round').check_args(0).has_equal_value(incorrect_msg = 'argrwong')
Ex().check_object('x', missing_msg='objectnotdefined').has_equal_value(
↪'objectincorrect')
```

Result

```
objectnotdefined
```

No error

13.15.43 Example 43

No PEC

Solution code

```
x = round(1.23)
```

Student code

```
x = round(1.23) + 1
```

No output

SCT

```
Ex().check_function('round').check_args(0).has_equal_value(incorrect_msg = 'argrwong')
Ex().check_object('x', missing_msg='objectnotdefined').has_equal_value(
↪'objectincorrect')
```

Result

```
objectincorrect
```

No error

13.15.44 Example 44

No PEC

Solution code

```
def test(a = 1): return a
```

No student code

No output

SCT

```
Ex().check_function_def('test').check_args('a').has_equal_part('is_default', msg='not␣
↪default').has_equal_value()
```

Result

```
The system wants to check the definition of <code>test()</code> but hasn't found it.
```

No error

13.15.45 Example 45

No PEC

Solution code

```
def test(a = 1): return a
```

Student code

```
def test(b): return b
```

No output

SCT

```
Ex().check_function_def('test').check_args('a').has_equal_part('is_default', msg='not␣
↪default').has_equal_value()
```

Result

```
Check the definition of <code>test()</code>. Did you specify the argument <code>a</
↪code>?
```

No error

13.15.46 Example 46

No PEC

Solution code

```
def test(a = 1): return a
```

Student code

```
def test(a): return a
```

No output

SCT

```
Ex().check_function_def('test').check_args('a').has_equal_part('is_default', msg='not␣
↪default').has_equal_value()
```

Result

```
Check the definition of <code>test()</code>. Did you correctly specify the argument
↪<code>a</code>? not default
```

No error

13.15.47 Example 47

No PEC

Solution code

```
def test(a = 1): return a
```

Student code

```
def test(a = 2): return a
```

No output

SCT

```
Ex().check_function_def('test').check_args('a').has_equal_part('is_default', msg='not␣
↪default').has_equal_value()
```

Result

```
Check the definition of <code>test()</code>. Did you correctly specify the argument
↪<code>a</code>? Expected <code>1</code>, but got <code>2</code>.
```

No error

13.15.48 Example 48

No PEC

Solution code

```
def test(a, b): print(a + b); return a + b
```

No student code

No output

SCT

```
Ex().check_function_def('test').multi(
    check_call('f(1, 2)').has_equal_value(),
    check_call('f(1, 2)').has_equal_output(),
    check_call('f(3, 1)').has_equal_value(),
    check_call('f(1, "2")').has_equal_error()
)
```

Result

```
The system wants to check the definition of <code>test()</code> but hasn't found it.
```

No error

13.15.49 Example 49

No PEC

Solution code

```
def test(a, b): print(a + b); return a + b
```

Student code

```
def test(a, b): return 1
```

No output

SCT

```
Ex().check_function_def('test').multi(
    check_call('f(1, 2)').has_equal_value(),
    check_call('f(1, 2)').has_equal_output(),
    check_call('f(3, 1)').has_equal_value(),
    check_call('f(1, "2")').has_equal_error()
)
```

Result

```
Check the definition of <code>test()</code>. To verify it, we reran <code>test(1, 2)</
↪code>. Expected <code>3</code>, but got <code>1</code>.
```

No error

13.15.50 Example 50

No PEC

Solution code

```
def test(a, b): print(a + b); return a + b
```

Student code

```
def test(a, b): return a + b
```

No output

SCT

```
Ex().check_function_def('test').multi(
    check_call('f(1, 2)').has_equal_value(),
    check_call('f(1, 2)').has_equal_output(),
    check_call('f(3, 1)').has_equal_value(),
    check_call('f(1, "2")').has_equal_error()
)
```

Result

```
Check the definition of <code>test()</code>. To verify it, we reran <code>test(1, 2)</
↪code>. Expected the output <code>3</code>, but got <code>no printouts</code>.
```

No error

13.15.51 Example 51

No PEC

Solution code

```
def test(a, b): print(a + b); return a + b
```

Student code

```
def test(a, b):
    if a == 3:
        raise ValueError('wrong')
    print(a + b)
    return a + b
```

No output

SCT

```
Ex().check_function_def('test').multi(
    check_call('f(1, 2)').has_equal_value(),
    check_call('f(1, 2)').has_equal_output(),
    check_call('f(3, 1)').has_equal_value(),
    check_call('f(1, "2")').has_equal_error()
)
```

Result

```
Check the definition of <code>test()</code>. To verify it, we reran <code>test(3, 1)</
↪code>. Running the higlighted expression generated an error: <code>wrong</code>.
```

No error

13.15.52 Example 52

No PEC

Solution code

```
def test(a, b): print(a + b); return a + b
```

Student code

```
def test(a, b): print(int(a) + int(b)); return int(a) + int(b)
```

No output

SCT

```
Ex().check_function_def('test').multi(
    check_call('f(1, 2)').has_equal_value(),
    check_call('f(1, 2)').has_equal_output(),
    check_call('f(3, 1)').has_equal_value(),
    check_call('f(1, "2")').has_equal_error()
)
```

Result

```
Check the definition of <code>test()</code>. To verify it, we reran <code>test(1, "2")
↪</code>. Running the higlighted expression didn't generate an error, but it should!
```

No error

13.15.53 Example 53

No PEC

Solution code

```
echo_word = (lambda word1, echo: word1 * echo)
```

Student code

```
echo_word = (lambda word1, echo: word1 * echo * 2)
```

No output

SCT

```
Ex().check_lambda_function().check_call("f('test', 2)").has_equal_value()
```

Result

```
Check the first lambda function. To verify it, we reran it with the arguments <code>(
↪'test', 2)</code>. Expected <code>testtest</code>, but got <code>testtesttesttest</
↪code>.
```

No error

13.15.54 Example 54

No PEC

Solution code

```
class A(str):
  def __init__(self): pass
```

No student code

No output

SCT

```
Ex().check_class_def('A').multi( check_bases(0).has_equal_ast(), check_body().check_
↪function_def('__init__').check_body().has_equal_ast() )
```

Result

```
The system wants to check the class definition of <code>A</code> but hasn't found it.
```

No error

13.15.55 Example 55

No PEC

Solution code

```
class A(str):
  def __init__(self): pass
```

Student code

```
def A(x): pass
```

No output

SCT

```
Ex().check_class_def('A').multi( check_bases(0).has_equal_ast(), check_body().check_
↪function_def('__init__').check_body().has_equal_ast() )
```

Result

```
The system wants to check the class definition of <code>A</code> but hasn't found it.
```

No error

13.15.56 Example 56

No PEC

Solution code

```
class A(str):
  def __init__(self): pass
```

Student code

```
class A(): pass
```

No output

SCT

```
Ex().check_class_def('A').multi( check_bases(0).has_equal_ast(), check_body().check_
↪function_def('__init__').check_body().has_equal_ast() )
```

Result

```
Check the class definition of <code>A</code>. Are you sure you defined the first base␣
↪class?
```

No error

13.15.57 Example 57

No PEC

Solution code

```
class A(str):
  def __init__(self): pass
```

Student code

```
class A(int): pass
```

No output

SCT

```
Ex().check_class_def('A').multi( check_bases(0).has_equal_ast(), check_body().check_
↪function_def('__init__').check_body().has_equal_ast() )
```

Result

```
Check the class definition of <code>A</code>. Did you correctly specify the first␣
↪base class? Expected <code>str</code>, but got <code>int</code>.
```

No error

13.15.58 Example 58

No PEC

Solution code

```
class A(str):
  def __init__(self): pass
```

Student code

```
class A(str):
  def __not_init__(self): pass
```

No output

SCT

```
Ex().check_class_def('A').multi( check_bases(0).has_equal_ast(), check_body().check_
↪function_def('__init__').check_body().has_equal_ast() )
```

Result

```
Check the class definition of <code>A</code>. Did you correctly specify the body? The␣
↪system wants to check the definition of <code>__init__()</code> but hasn't found it.
```

No error

13.15.59 Example 59

No PEC

Solution code

```
class A(str):
  def __init__(self): pass
```

Student code

```
class A(str):
  def __init__(self): print(1)
```

No output

SCT

```
Ex().check_class_def('A').multi( check_bases(0).has_equal_ast(), check_body().check_
↪function_def('__init__').check_body().has_equal_ast() )
```

Result

```
Check the definition of <code>__init__()</code>. Did you correctly specify the body?␣
↪Expected <code>pass</code>, but got <code>print(1)</code>.
```

No error

13.15.60 Example 60

No PEC

Solution code

```
import pandas as pd
```

No student code

No output

SCT

```
Ex().has_import('pandas', same_as=True)
```

Result

```
Did you import <code>pandas</code>?
```

No error

13.15.61 Example 61

No PEC

Solution code

```
import pandas as pd
```

Student code

```
import pandas
```

No output

SCT

```
Ex().has_import('pandas', same_as=True)
```

Result

```
Did you import <code>pandas</code> as <code>pd</code>?
```

No error

13.15.62 Example 62

No PEC

Solution code

```
import pandas as pd
```

Student code

```
import pandas as pan
```

No output

SCT

```
Ex().has_import('pandas', same_as=True)
```

Result

```
Did you import <code>pandas</code> as <code>pd</code>?
```

No error

13.15.63 Example 63

No PEC

Solution code

```
import pandas as pd
```

No student code

No output

SCT

```
Ex().has_import('pandas', same_as=True, not_imported_msg='wrong', incorrect_as_msg=
↪'wrongas')
```

Result

```
wrong
```

No error

13.15.64 Example 64

No PEC

Solution code

```
import pandas as pd
```

Student code

```
import pandas
```

No output

SCT

```
Ex().has_import('pandas', same_as=True, not_imported_msg='wrong', incorrect_as_msg=
↪'wrongas')
```

Result

```
wrongas
```

No error

13.15.65 Example 65

No PEC

Solution code

```
import pandas as pd
```

Student code

```
import pandas as pan
```

No output

SCT

```
Ex().has_import('pandas', same_as=True, not_imported_msg='wrong', incorrect_as_msg=
↪'wrongas')
```

Result

```
wrongas
```

No error

13.15.66 Example 66

No PEC

Solution code

```
my_dict = {'a': 1, 'b': 2}
for key, value in my_dict.items():
    x = key + ' - ' + str(value)
    print(x)
```

Student code

```
my_dict = {'a': 1, 'b': 2}
for key, value in my_dict.items(): x = key + ' -- ' + str(value)
```

No output

SCT

```
Ex().check_for_loop().check_body().set_context('a', 1).multi(has_equal_value(name = 'x
↪'), has_equal_output())
```

Result

```
Check the first for loop. Did you correctly specify the body? Are you sure you␣
↪assigned the correct value to <code>x</code>?
```

No error

13.15.67 Example 67

No PEC

Solution code

```
my_dict = {'a': 1, 'b': 2}
for key, value in my_dict.items():
    x = key + ' - ' + str(value)
    print(x)
```

Student code

```
my_dict = {'a': 1, 'b': 2}
for key, value in my_dict.items(): x = key + ' - ' + str(value)
```

No output

SCT

```
Ex().check_for_loop().check_body().set_context('a', 1).multi(has_equal_value(name = 'x
↪'), has_equal_output())
```

Result

```
Check the first for loop. Did you correctly specify the body? Expected the output
↪<code>a - 1</code>, but got <code>no printouts</code>.
```

No error

13.15.68 Example 68

No PEC

Solution code

```
result = (num for num in range(31))
```

Student code

```
result = (num for num in range(3))
```

No output

SCT

```
Ex().check_generator_exp().multi(check_iter().has_equal_value(), check_body().set_
↪context(4).has_equal_value())
```

Result

```
Check the first generator expression. Did you correctly specify the iterable part?␣
↪Expected <code>range(0, 31)</code>, but got <code>range(0, 3)</code>.
```

No error

13.15.69 Example 69

No PEC

Solution code

```
result = (num for num in range(31))
```

Student code

```
result = (num*2 for num in range(31))
```

No output

SCT

```
Ex().check_generator_exp().multi(check_iter().has_equal_value(), check_body().set_
↪context(4).has_equal_value())
```

Result

```
Check the first generator expression. Did you correctly specify the body? Expected
↪<code>4</code>, but got <code>8</code>.
```

No error

13.15.70 Example 70

No PEC

No solution code

Student code

```
c
```

No output

SCT

```
Ex().has_no_error()
```

Result

```
Have a look at the console: your code contains an error. Fix it and try again!
```

Error

```
name 'c' is not defined
```

13.15.71 Example 71

No PEC

Solution code

```
a = 1; b = a + 1; c = b + 1; print(c)
```

No student code

No output

SCT

```
Ex().test_correct(
    has_printout(0),
    F().test_correct(
        check_object('c').has_equal_value(),
        F().test_correct(
            check_object('b').has_equal_value(),
            check_object('a').has_equal_value()
        )
    )
)
```

Result

```
Did you define the variable <code>a</code> without errors?
```

No error

13.15.72 Example 72

No PEC

Solution code

```
a = 1; b = a + 1; c = b + 1; print(c)
```

Student code

```
a = 1
```

No output

SCT

```
Ex().test_correct(
    has_printout(0),
    F().test_correct(
        check_object('c').has_equal_value(),
        F().test_correct(
            check_object('b').has_equal_value(),
            check_object('a').has_equal_value()
        )
    )
)
```

Result

```
Did you define the variable <code>b</code> without errors?
```

No error

13.15.73 Example 73

No PEC

Solution code

```
a = 1; b = a + 1; c = b + 1; print(c)
```

Student code

```
a = 1; b = a + 1
```

No output

SCT

```
Ex().test_correct(
    has_printout(0),
    F().test_correct(
        check_object('c').has_equal_value(),
        F().test_correct(
            check_object('b').has_equal_value(),
            check_object('a').has_equal_value()
        )
    )
)
```

Result

```
Did you define the variable <code>c</code> without errors?
```

No error

13.15.74 Example 74

No PEC

Solution code

```
a = 1; b = a + 1; c = b + 1; print(c)
```

Student code

```
a = 1; b = a + 1; c = b + 1
```

No output

SCT

```
Ex().test_correct(
    has_printout(0),
    F().test_correct(
        check_object('c').has_equal_value(),
        F().test_correct(
            check_object('b').has_equal_value(),
            check_object('a').has_equal_value()
        )
    )
)
```

Result

```
Have you used <code>print(c)</code> to do the appropriate printouts?
```

No error

13.15.75 Example 75

No PEC

Solution code

```
a = 1; b = a + 1; c = b + 1; print(c)
```

Student code

```
print(4)
```

Student output

```
4
```

SCT

```
Ex().test_correct(
    has_printout(0),
    F().test_correct(
        check_object('c').has_equal_value(),
        F().test_correct(
            check_object('b').has_equal_value(),
            check_object('a').has_equal_value()
        )
    )
)
```

Result

```
Did you define the variable <code>a</code> without errors?
```

No error

13.15.76 Example 76

No PEC

Solution code

```
a = 1; b = a + 1; c = b + 1; print(c)
```

Student code

```
c = 3; print(c + 1)
```

Student output

```
4
```

SCT

```
Ex().test_correct(
    has_printout(0),
    F().test_correct(
        check_object('c').has_equal_value(),
        F().test_correct(
            check_object('b').has_equal_value(),
            check_object('a').has_equal_value()
        )
    )
)
```

Result

```
Have you used <code>print(c)</code> to do the appropriate printouts?
```

No error

13.15.77 Example 77

No PEC

Solution code

```
a = 1; b = a + 1; c = b + 1; print(c)
```

Student code

```
b = 3; c = b + 1; print(c)
```

Student output

```
4
```

SCT

```
Ex().test_correct(
    has_printout(0),
    F().test_correct(
        check_object('c').has_equal_value(),
        F().test_correct(
            check_object('b').has_equal_value(),
            check_object('a').has_equal_value()
        )
    )
)
```

Result

```
Did you define the variable <code>a</code> without errors?
```

No error

13.15.78 Example 78

No PEC

Solution code

```
a = 1; b = a + 1; c = b + 1; print(c)
```

Student code

```
a = 2; b = a + 1; c = b + 1
```

No output

SCT

```
Ex().test_correct(
    has_printout(0),
    F().test_correct(
        check_object('c').has_equal_value(),
        F().test_correct(
            check_object('b').has_equal_value(),
            check_object('a').has_equal_value()
        )
    )
)
```

Result

```
Did you correctly define the variable <code>a</code>? Expected <code>1</code>, but␣
↪got <code>2</code>.
```

No error

13.15.79 Example 79

No PEC

Solution code

```
for i in range(2):
    for j in range(2):
        print(str(i) + "+" + str(j))
```

Student code

```
for i in range(2):
    for j in range(2):
        print(str(i) + "-" + str(j))
```

Student output

```
0-0
0-1
1-0
1-1
```

SCT

```
Ex().check_for_loop().check_body().check_for_loop().check_body().has_equal_output()
```

Result

```
Check the first for loop. Did you correctly specify the body? Expected the output
↪<code>1+1</code>, but got <code>1-1</code>.
```

No error

13.15.80 Example 80

No PEC

Solution code

```
for i in range(2):
    for j in range(2):
        print(str(i) + "+" + str(j))
```

Student code

```
for i in range(2):
    for j in range(2):
        print(str(i) + "-" + str(j))
```

Student output

```
0-0
0-1
1-0
1-1
```

SCT

```
Ex().check_for_loop().check_body().check_for_loop().disable_highlighting().check_
↪body().has_equal_output()
```

Result

```
Check the first for loop. Did you correctly specify the body? Check the first for␣
↪loop. Did you correctly specify the body? Expected the output <code>1+1</code>, but␣
↪got <code>1-1</code>.
```

No error

13.15.81 Example 81

No PEC

Solution code

```
x = [1]
```

Student code

```
x = [0]
```

No output

SCT

```
Ex().check_object('x').has_equal_value()
```

Result

```
Did you correctly define the variable <code>x</code>? Expected <code>[1]</code>, but␣
↪got <code>[0]</code>.
```

No error

13.15.82 Example 82

No PEC

Solution code

```
x = [1]
```

Student code

```
x = [0]
```

No output

SCT

```
Ex().has_equal_value(name = 'x')
```

Result

```
Are you sure you assigned the correct value to <code>x</code>?
```

No error

13.15.83 Example 83

No PEC

Solution code

```
x = [1]
```

Student code

```
x = [0]
```

No output

SCT

```
Ex().has_equal_value(expr_code = 'x[0]')
```

Result

```
Running the expression <code>x[0]</code> didn't generate the expected result.
```

No error

13.15.84 Example 84

No PEC

Solution code

```
u = 'valid'
if u:
  print('')
```

Student code

```
u = 'valid'
if u:
  print('')
```

Student output

SCT

```
Ex().check_if_else().check_test().has_equal_value(expr_code = '__focus__ == \'valid\'
↪')
```

Result

```
Great work!
```

No error

13.15.85 Example 85

No PEC

Solution code

```
u = 'valid'
if u:
  print('')
```

Student code

```
u = 'wrong'
if u:
  print('')
```

Student output

SCT

```
Ex().check_if_else().check_test().has_equal_value(expr_code = '__focus__ == \'valid\'
↪')
```

Result

```
Check the first if statement. Did you correctly specify the condition? Running the␣
↪expression <code>u == 'valid'</code> didn't generate the expected result.
```

No error

13.15.86 Example 86

PEC

```
offset = 8
```

Solution code

```
if offset > 8: x = 5
else: x = round(2.123)
```

No student code

No output

SCT

```
Ex().check_if_else().multi(
    check_test().multi([ set_env(offset = i).has_equal_value() for i in range(7,10)␣
↪]),
    check_body().has_code(r'x\s*=\s*5'),
    check_orelse().check_function('round').check_args(0).has_equal_value()
)
```

Result

```
The system wants to check the first if statement but hasn't found it.
```

No error

13.15.87 Example 87

PEC

```
offset = 8
```

Solution code

```
if offset > 8: x = 5
else: x = round(2.123)
```

Student code

```
if offset > 10: x = 5
else: x = round(2.123)
```

No output

SCT

```
Ex().check_if_else().multi(
    check_test().multi([ set_env(offset = i).has_equal_value() for i in range(7,10)␣
↪]),
    check_body().has_code(r'x\s*=\s*5'),
    check_orelse().check_function('round').check_args(0).has_equal_value()
)
```

Result

```
Check the first if statement. Did you correctly specify the condition? Expected <code>
↪True</code>, but got <code>False</code>.
```

No error

13.15.88 Example 88

PEC

```
offset = 8
```

Solution code

```
if offset > 8: x = 5
else: x = round(2.123)
```

Student code

```
if offset > 8: x = 7
else: x = round(2.123)
```

No output

SCT

```
Ex().check_if_else().multi(
    check_test().multi([ set_env(offset = i).has_equal_value() for i in range(7,10)␣
↪]),
```

(continues on next page)

(continued from previous page)

```
    check_body().has_code(r'x\s*=\s*5'),
    check_orelse().check_function('round').check_args(0).has_equal_value()
)
```

Result

```
Check the first if statement. Did you correctly specify the body? Could not find the␣
↪correct pattern in your code.
```

No error

13.15.89 Example 89

PEC

```
offset = 8
```

Solution code

```
if offset > 8: x = 5
else: x = round(2.123)
```

Student code

```
if offset > 8: x = 5
else: x = 8
```

No output

SCT

```
Ex().check_if_else().multi(
    check_test().multi([ set_env(offset = i).has_equal_value() for i in range(7,10)␣
↪]),
    check_body().has_code(r'x\s*=\s*5'),
    check_orelse().check_function('round').check_args(0).has_equal_value()
)
```

Result

```
Check the first if statement. Did you correctly specify the else part? Did you call
↪<code>round()</code>?
```

No error

13.15.90 Example 90

PEC

```
offset = 8
```

Solution code

```
if offset > 8: x = 5
else: x = round(2.123)
```

Student code

```
if offset > 8: x = 5
else: x = round(2.2121314)
```

No output

SCT

```
Ex().check_if_else().multi(
    check_test().multi([ set_env(offset = i).has_equal_value() for i in range(7,10)␣
↪]),
    check_body().has_code(r'x\s*=\s*5'),
    check_orelse().check_function('round').check_args(0).has_equal_value()
)
```

Result

```
Check your call of <code>round()</code>. Did you correctly specify the first argument?
↪ Expected <code>2.123</code>, but got <code>2.2121314</code>.
```

No error

13.15.91 Example 91

No PEC

Solution code

```
x = 4
```

Student code

```
x = 3
```

No output

SCT

```
Ex().check_object('x').has_equal_value(incorrect_msg="__JINJA__:You did {{stu_eval}},␣
↪but should be {{sol_eval}}!")
```

Result

```
You did 3, but should be 4!
```

No error

13.15.92 Example 92

No PEC

Solution code

```
x = 4
```

Student code

```
x = 3
```

No output

SCT

```
Ex().check_object('x').has_equal_value(incorrect_msg="You did {{stu_eval}}, but␣
↪should be {{sol_eval}}!")
```

Result

```
You did 3, but should be 4!
```

No error

13.16 test_set_context.py

13.16.1 Example 1

No PEC

Solution code

```
def shout_echo(word1, echo=1):
    echo_word = word1 * echo
    shout_words = echo_word + '!!!!'
    return shout_words
```

Student code

```
def shout_echo(word1, echo=1):
    echo_word = word1 * echo
    shout_words = echo_word + '!!!!'
    return shout_words
```

No output

SCT

```
Ex().check_function_def('shout_echo').check_body().set_context('test', 1).has_equal_
↪value(name = 'shout_words')
```

Result

```
Great work!
```

No error

13.16.2 Example 2

No PEC

Solution code

```
def shout_echo(word1, echo=1):
    echo_word = word1 * echo
    shout_words = echo_word + '!!!'
    return shout_words
```

Student code

```
def shout_echo(word1, echo=1):
    echo_word = word1 * echo
    shout_words = echo_word + '!!!'
    return shout_words
```

No output

SCT

```
Ex().check_function_def('shout_echo').check_body().set_context(word1 = 'test', echo =␣
↪1).has_equal_value(name = 'shout_words')
```

Result

```
Great work!
```

No error

13.16.3 Example 3

No PEC

Solution code

```
def shout_echo(w, echo=1):
    echo_word = w * echo
    shout_words = echo_word + '!!!'
    return shout_words
```

Student code

```
def shout_echo(word1, echo=1):
    echo_word = word1 * echo
    shout_words = echo_word + '!!!'
    return shout_words
```

No output

SCT

```
Ex().check_function_def('shout_echo').check_body().set_context('test', 1).has_equal_
↪value(name = 'shout_words')
```

Result

```
Great work!
```

No error

13.16.4 Example 4

No PEC

Solution code

```
x = { m:len(m) for m in ['a', 'b', 'c'] }
```

Student code

```
x = { m:len(m) for m in ['a', 'b', 'c'] }
```

No output

SCT

```
Ex().check_dict_comp().check_key().set_context('a').has_equal_value()
```

Result

```
Great work!
```

No error

13.16.5 Example 5

No PEC

Solution code

```
x = { m:len(m) for m in ['a', 'b', 'c'] }
```

Student code

```
x = { m:len(m) for m in ['a', 'b', 'c'] }
```

No output

SCT

```
Ex().check_dict_comp().check_key().set_context(m = 'a').has_equal_value()
```

Result

```
Great work!
```

No error

13.16.6 Example 6

No PEC

Solution code

```
x = { n:len(n) for n in ['a', 'b', 'c'] }
```

Student code

```
x = { m:len(m) for m in ['a', 'b', 'c'] }
```

No output

SCT

```
Ex().check_dict_comp().check_key().set_context('a').has_equal_value()
```

Result

```
Great work!
```

No error

13.16.7 Example 7

No PEC

Solution code

```
x = { m*2:len(m) for m in ['a', 'b', 'c'] }
```

Student code

```
x = { m:len(m) for m in ['a', 'b', 'c'] }
```

No output

SCT

```
Ex().check_dict_comp().check_key().set_context('a').has_equal_value()
```

Result

```
Check the first dictionary comprehension. Did you correctly specify the key part?␣
↪Expected <code>aa</code>, but got <code>a</code>.
```

No error

13.17 test_has_expr.py

13.17.1 Example 1

No PEC

Solution code

```
a = 0
for i in range(0): a = a + 1
```

Student code

```
a = 0
for i in range(0): pass
```

No output

SCT

```
test_for_loop(body = test_object_after_expression('a'))
```

Result

```
Check the first for loop. Did you correctly specify the body? Are you sure you␣
↪assigned the correct value to <code>a</code>?
```

No error

13.17.2 Example 2

No PEC

Solution code

```
a = 0
for i in range(0): a = a + 1
```

Student code

```
a = 0
for i in range(0): a = a - 1
```

No output

SCT

```
test_for_loop(body = test_object_after_expression('a'))
```

Result

```
Check the first for loop. Did you correctly specify the body? Are you sure you␣
↪assigned the correct value to <code>a</code>?
```

No error

13.17.3 Example 3

No PEC

Solution code

```
a = 0
for i in range(0): a = a + 1
```

Student code

```
a = 0
for i in range(0): a = a + 1
```

No output

SCT

```
test_for_loop(body = test_object_after_expression('a'))
```

Result

```
Great work!
```

No error

13.18 test_has_chosen.py

13.18.1 Example 1

No PEC

No solution code

Student code

```
selected_option = 1
```

No output

SCT

```
test_mc(2, ['a', 'b', 'c'])
```

Result

```
a
```

No error

13.18.2 Example 2

No PEC

No solution code

Student code

```
selected_option = 1
```

No output

SCT

```
Ex().has_chosen(2, ['a', 'b', 'c'])
```

Result

```
a
```

No error

13.18.3 Example 3

No PEC

No solution code

Student code

```
selected_option = 2
```

No output

SCT

```
test_mc(2, ['a', 'b', 'c'])
```

Result

```
b
```

No error

13.18.4 Example 4

No PEC

No solution code

Student code

```
selected_option = 2
```

No output

SCT

```
Ex().has_chosen(2, ['a', 'b', 'c'])
```

Result

```
b
```

No error

13.18.5 Example 5

No PEC

No solution code

Student code

```
selected_option = 3
```

No output

SCT

```
test_mc(2, ['a', 'b', 'c'])
```

Result

```
c
```

No error

13.18.6 Example 6

No PEC

No solution code

Student code

```
selected_option = 3
```

No output

SCT

```
Ex().has_chosen(2, ['a', 'b', 'c'])
```

Result

```
c
```

No error

13.19 test_check_function.py

13.19.1 Example 1

PEC

```
def my_fun(a, b): pass
```

Solution code

```
my_fun(1, 2)
```

Student code

```
my_fun(x = 2)
```

No output

SCT

```
Ex().check_function('my_fun')
```

Result

```
Have you specified the arguments for <code>my_fun()</code> using the right syntax?
```

Error

```
my_fun() got an unexpected keyword argument 'x'
```

13.19.2 Example 2

No PEC

Solution code

```
def my_func(a): return my_func_in_return(b)
```

Student code

```
def my_func(a): return my_func_in_return(b)
```

No output

SCT

```
Ex().check_function_def('my_func').check_body().check_function('my_func_in_return',␣
↪signature=False)
```

Result

```
Great work!
```

No error

13.19.3 Example 3

No PEC

Solution code

```
0 < len([])
```

Student code

```
0 < len([])
```

No output

SCT

```
Ex().check_function('len')
```

Result

```
Great work!
```

No error

13.19.4 Example 4

No PEC

Solution code

```
len([]) < 3
```

Student code

```
len([]) < 3
```

No output

SCT

```
Ex().check_function('len')
```

Result

```
Great work!
```

No error

13.19.5 Example 5

No PEC

Solution code

```
0 < len([]) < 3
```

Student code

```
0 < len([]) < 3
```

No output

SCT

```
Ex().check_function('len')
```

Result

```
Great work!
```

No error

13.20 test_debug.py

13.20.1 Example 1

No PEC

Solution code

```
x = 123
```

Student code

```
x = 123
```

No output

SCT

```
success_msg('great')
```

Result

```
great
```

No error

13.20.2 Example 2

No PEC

Solution code

```
# Example, do not modify!
print(5 / 8)

# Put code below here
print(7 + 10)
```

Student code

```
# Example, do not modify!
print(5 / 8)

# Put code below here
print(7 + 10)
```

Student output

```
0.625
17
```

SCT

```
Ex().has_printout(1, not_printed_msg = "__JINJA__:Have you used `{{sol_call}}` to␣
↪print out the sum of 7 and 10?")
success_msg("Great! On to the next one!")
```

Result

```
Great! On to the next one!
```

No error

13.21 test_check_list_comp.py

13.21.1 Example 1

PEC

```
x = {'a': 2, 'b':3, 'c':4, 'd':'test'}
```

Solution code

```
[key + str(val) for key,val in x.items() if isinstance(key, str) if isinstance(val,␣
↪int)]
```

No student code

No output

SCT

```
test_list_comp(index=1,
               not_called_msg=None,
               comp_iter=lambda: test_expression_result(),
               iter_vars_names=True,
               incorrect_iter_vars_msg=None,
               body=lambda: test_expression_result(context_vals = ['a', 2]),
               ifs=[lambda: test_function_v2('isinstance', params = ['obj'], do_eval␣
↪= [False]),
                    lambda: test_function_v2('isinstance', params = ['obj'], do_eval␣
↪= [False])],
               insufficient_ifs_msg=None)
```

Result

```
The system wants to check the first list comprehension but hasn't found it.
```

No error

13.21.2 Example 2

PEC

```
x = {'a': 2, 'b':3, 'c':4, 'd':'test'}
```

Solution code

```
[key + str(val) for key,val in x.items() if isinstance(key, str) if isinstance(val,␣
↪int)]
```

Student code

```
[key for key in x.keys()]
```

No output

SCT

```
test_list_comp(index=1,
               not_called_msg=None,
               comp_iter=lambda: test_expression_result(),
               iter_vars_names=True,
               incorrect_iter_vars_msg=None,
               body=lambda: test_expression_result(context_vals = ['a', 2]),
               ifs=[lambda: test_function_v2('isinstance', params = ['obj'], do_eval␣
↪= [False]),
                    lambda: test_function_v2('isinstance', params = ['obj'], do_eval␣
↪= [False])],
               insufficient_ifs_msg=None)
```

Result

```
Check the first list comprehension. Did you correctly specify the iterable part?␣
↪Expected <code>[('a', 2), ('b', 3), ('c', 4), ('d', 'test')]</code>, but got <code>[
↪'a', 'b', 'c', 'd']</code>.
```

No error

13.21.3 Example 3

PEC

```
x = {'a': 2, 'b':3, 'c':4, 'd':'test'}
```

Solution code

```
[key + str(val) for key,val in x.items() if isinstance(key, str) if isinstance(val,␣
↪int)]
```

Student code

```
[a + str(b) for a,b in x.items()]
```

No output

SCT

```
test_list_comp(index=1,
               not_called_msg=None,
               comp_iter=lambda: test_expression_result(),
               iter_vars_names=True,
               incorrect_iter_vars_msg=None,
               body=lambda: test_expression_result(context_vals = ['a', 2]),
               ifs=[lambda: test_function_v2('isinstance', params = ['obj'], do_eval␣
↪= [False]),
                    lambda: test_function_v2('isinstance', params = ['obj'], do_eval␣
↪= [False])],
               insufficient_ifs_msg=None)
```

Result

```
Check the first list comprehension. Have you used the correct iterator variables?
```

No error

13.21.4 Example 4

PEC

```
x = {'a': 2, 'b':3, 'c':4, 'd':'test'}
```

Solution code

```
[key + str(val) for key,val in x.items() if isinstance(key, str) if isinstance(val,␣
↪int)]
```

Student code

```
[key + '_' + str(val) for key,val in x.items()]
```

No output

SCT

```
test_list_comp(index=1,
               not_called_msg=None,
               comp_iter=lambda: test_expression_result(),
               iter_vars_names=True,
               incorrect_iter_vars_msg=None,
               body=lambda: test_expression_result(context_vals = ['a', 2]),
               ifs=[lambda: test_function_v2('isinstance', params = ['obj'], do_eval␣
↪= [False]),
                    lambda: test_function_v2('isinstance', params = ['obj'], do_eval␣
↪= [False])],
               insufficient_ifs_msg=None)
```

Result

```
Check the first list comprehension. Did you correctly specify the body? Expected
↪<code>a2</code>, but got <code>a_2</code>.
```

No error

13.21.5 Example 5

PEC

```
x = {'a': 2, 'b':3, 'c':4, 'd':'test'}
```

Solution code

```
[key + str(val) for key,val in x.items() if isinstance(key, str) if isinstance(val,␣
↪int)]
```

Student code

```
[key + str(val) for key,val in x.items()]
```

No output

SCT

```
test_list_comp(index=1,
               not_called_msg=None,
               comp_iter=lambda: test_expression_result(),
               iter_vars_names=True,
               incorrect_iter_vars_msg=None,
               body=lambda: test_expression_result(context_vals = ['a', 2]),
               ifs=[lambda: test_function_v2('isinstance', params = ['obj'], do_eval␣
↪= [False]),
                    lambda: test_function_v2('isinstance', params = ['obj'], do_eval␣
↪= [False])],
               insufficient_ifs_msg=None)
```

Result

```
Check the first list comprehension. Have you used 2 ifs?
```

No error

13.21.6 Example 6

PEC

```
x = {'a': 2, 'b':3, 'c':4, 'd':'test'}
```

Solution code

```
[key + str(val) for key,val in x.items() if isinstance(key, str) if isinstance(val,␣
↪int)]
```

Student code

```
[key + str(val) for key,val in x.items() if hasattr(key, 'test') if hasattr(key, 'test
↪')]
```

No output

SCT

```
test_list_comp(index=1,
               not_called_msg=None,
               comp_iter=lambda: test_expression_result(),
               iter_vars_names=True,
               incorrect_iter_vars_msg=None,
               body=lambda: test_expression_result(context_vals = ['a', 2]),
               ifs=[lambda: test_function_v2('isinstance', params = ['obj'], do_eval␣
↪= [False]),
                    lambda: test_function_v2('isinstance', params = ['obj'], do_eval␣
↪= [False])],
               insufficient_ifs_msg=None)
```

Result

```
Check the first list comprehension. Did you correctly specify the first if? Did you␣
↪call <code>isinstance()</code>?
```

No error

13.21.7 Example 7

PEC

```
x = {'a': 2, 'b':3, 'c':4, 'd':'test'}
```

Solution code

```
[key + str(val) for key,val in x.items() if isinstance(key, str) if isinstance(val,␣
↪int)]
```

Student code

```
[key + str(val) for key,val in x.items() if isinstance(key, str) if hasattr(key, 'test
↪')]
```

No output

SCT

```
test_list_comp(index=1,
               not_called_msg=None,
               comp_iter=lambda: test_expression_result(),
               iter_vars_names=True,
               incorrect_iter_vars_msg=None,
               body=lambda: test_expression_result(context_vals = ['a', 2]),
               ifs=[lambda: test_function_v2('isinstance', params = ['obj'], do_eval␣
↪= [False]),
                    lambda: test_function_v2('isinstance', params = ['obj'], do_eval␣
↪= [False])],
               insufficient_ifs_msg=None)
```

Result

```
Check the first list comprehension. Did you correctly specify the second if? Did you␣
↪call <code>isinstance()</code>?
```

No error

13.21.8 Example 8

PEC

```
x = {'a': 2, 'b':3, 'c':4, 'd':'test'}
```

Solution code

```
[key + str(val) for key,val in x.items() if isinstance(key, str) if isinstance(val,␣
↪int)]
```

Student code

```
[key + str(val) for key,val in x.items() if isinstance(key, str) if isinstance(key,␣
↪str)]
```

No output

SCT

```
test_list_comp(index=1,
               not_called_msg=None,
               comp_iter=lambda: test_expression_result(),
               iter_vars_names=True,
               incorrect_iter_vars_msg=None,
               body=lambda: test_expression_result(context_vals = ['a', 2]),
               ifs=[lambda: test_function_v2('isinstance', params = ['obj'], do_eval␣
↪= [False]),
                    lambda: test_function_v2('isinstance', params = ['obj'], do_eval␣
↪= [False])],
               insufficient_ifs_msg=None)
```

Result

```
Check your call of <code>isinstance()</code>. Did you correctly specify the argument
↪<code>obj</code>? Expected <code>val</code>, but got <code>key</code>.
```

No error

13.21.9 Example 9

PEC

```
x = {'a': 2, 'b':3, 'c':4, 'd':'test'}
```

Solution code

```
[key + str(val) for key,val in x.items() if isinstance(key, str) if isinstance(val,␣
↪int)]
```

Student code

```
[key + str(val) for key,val in x.items() if isinstance(key, str) if isinstance(val,␣
↪str)]
```

No output

SCT

```
test_list_comp(index=1,
               not_called_msg=None,
               comp_iter=lambda: test_expression_result(),
               iter_vars_names=True,
               incorrect_iter_vars_msg=None,
               body=lambda: test_expression_result(context_vals = ['a', 2]),
               ifs=[lambda: test_function_v2('isinstance', params = ['obj'], do_eval␣
↪= [False]),
                    lambda: test_function_v2('isinstance', params = ['obj'], do_eval␣
↪= [False])],
               insufficient_ifs_msg=None)
```

Result

```
Great work!
```

No error

13.21.10 Example 10

PEC

```
x = {'a': 2, 'b':3, 'c':4, 'd':'test'}
```

Solution code

```
[key + str(val) for key,val in x.items() if isinstance(key, str) if isinstance(val,␣
↪int)]
```

No student code

No output

SCT

```
test_list_comp(index=1,
               not_called_msg='notcalled',
               comp_iter=lambda: test_expression_result(incorrect_msg = 'iterincorrect
↪'),
               iter_vars_names=True,
               incorrect_iter_vars_msg='incorrectitervars',
               body=lambda: test_expression_result(context_vals = ['a', 2], incorrect_
↪msg = 'bodyincorrect'),
               ifs=[lambda: test_function_v2('isinstance', params = ['obj'], do_eval␣
↪= [False], not_called_msg = 'notcalled1', incorrect_msg = 'incorrect2'),
                    lambda: test_function_v2('isinstance', params = ['obj'], do_eval␣
↪= [False], not_called_msg = 'notcalled2', incorrect_msg = 'incorrect2')],
               insufficient_ifs_msg='insufficientifs')
```

Result

```
notcalled
```

No error

13.21.11 Example 11

PEC

```
x = {'a': 2, 'b':3, 'c':4, 'd':'test'}
```

Solution code

```
[key + str(val) for key,val in x.items() if isinstance(key, str) if isinstance(val,␣
↪int)]
```

Student code

```
[key for key in x.keys()]
```

No output

SCT

```
test_list_comp(index=1,
               not_called_msg='notcalled',
               comp_iter=lambda: test_expression_result(incorrect_msg = 'iterincorrect
↪'),
               iter_vars_names=True,
               incorrect_iter_vars_msg='incorrectitervars',
               body=lambda: test_expression_result(context_vals = ['a', 2], incorrect_
↪msg = 'bodyincorrect'),
               ifs=[lambda: test_function_v2('isinstance', params = ['obj'], do_eval␣
↪= [False], not_called_msg = 'notcalled1', incorrect_msg = 'incorrect2'),
                    lambda: test_function_v2('isinstance', params = ['obj'], do_eval␣
↪= [False], not_called_msg = 'notcalled2', incorrect_msg = 'incorrect2')],
               insufficient_ifs_msg='insufficientifs')
```

Result

```
iterincorrect
```

No error

13.21.12 Example 12

PEC

```
x = {'a': 2, 'b':3, 'c':4, 'd':'test'}
```

Solution code

```
[key + str(val) for key,val in x.items() if isinstance(key, str) if isinstance(val,␣
↪int)]
```

Student code

```
[a + str(b) for a,b in x.items()]
```

No output

SCT

```
test_list_comp(index=1,
               not_called_msg='notcalled',
               comp_iter=lambda: test_expression_result(incorrect_msg = 'iterincorrect
↪'),
               iter_vars_names=True,
               incorrect_iter_vars_msg='incorrectitervars',
               body=lambda: test_expression_result(context_vals = ['a', 2], incorrect_
↪msg = 'bodyincorrect'),
               ifs=[lambda: test_function_v2('isinstance', params = ['obj'], do_eval␣
↪= [False], not_called_msg = 'notcalled1', incorrect_msg = 'incorrect2'),
                    lambda: test_function_v2('isinstance', params = ['obj'], do_eval␣
↪= [False], not_called_msg = 'notcalled2', incorrect_msg = 'incorrect2')],
               insufficient_ifs_msg='insufficientifs')
```

Result

```
Check the first list comprehension. incorrectitervars
```

No error

13.21.13 Example 13

PEC

```
x = {'a': 2, 'b':3, 'c':4, 'd':'test'}
```

Solution code

```
[key + str(val) for key,val in x.items() if isinstance(key, str) if isinstance(val,␣
↪int)]
```

Student code

```
[key + '_' + str(val) for key,val in x.items()]
```

No output

SCT

```
test_list_comp(index=1,
               not_called_msg='notcalled',
               comp_iter=lambda: test_expression_result(incorrect_msg = 'iterincorrect
↪'),
               iter_vars_names=True,
               incorrect_iter_vars_msg='incorrectitervars',
               body=lambda: test_expression_result(context_vals = ['a', 2], incorrect_
↪msg = 'bodyincorrect'),
               ifs=[lambda: test_function_v2('isinstance', params = ['obj'], do_eval␣
↪= [False], not_called_msg = 'notcalled1', incorrect_msg = 'incorrect2'),
                    lambda: test_function_v2('isinstance', params = ['obj'], do_eval␣
↪= [False], not_called_msg = 'notcalled2', incorrect_msg = 'incorrect2')],
               insufficient_ifs_msg='insufficientifs')
```

Result

```
bodyincorrect
```

No error

13.21.14 Example 14

PEC

```
x = {'a': 2, 'b':3, 'c':4, 'd':'test'}
```

Solution code

```
[key + str(val) for key,val in x.items() if isinstance(key, str) if isinstance(val,␣
↪int)]
```

Student code

```
[key + str(val) for key,val in x.items()]
```

No output

SCT

```
test_list_comp(index=1,
               not_called_msg='notcalled',
               comp_iter=lambda: test_expression_result(incorrect_msg = 'iterincorrect
↪'),
               iter_vars_names=True,
               incorrect_iter_vars_msg='incorrectitervars',
               body=lambda: test_expression_result(context_vals = ['a', 2], incorrect_
↪msg = 'bodyincorrect'),
               ifs=[lambda: test_function_v2('isinstance', params = ['obj'], do_eval␣
↪= [False], not_called_msg = 'notcalled1', incorrect_msg = 'incorrect2'),
                    lambda: test_function_v2('isinstance', params = ['obj'], do_eval␣
↪= [False], not_called_msg = 'notcalled2', incorrect_msg = 'incorrect2')],
               insufficient_ifs_msg='insufficientifs')
```

Result

```
Check the first list comprehension. insufficientifs
```

No error

13.21.15 Example 15

PEC

```
x = {'a': 2, 'b':3, 'c':4, 'd':'test'}
```

Solution code

```
[key + str(val) for key,val in x.items() if isinstance(key, str) if isinstance(val,␣
↪int)]
```

Student code

```
[key + str(val) for key,val in x.items() if hasattr(key, 'test') if hasattr(key, 'test
↪')]
```

No output

SCT

```
test_list_comp(index=1,
               not_called_msg='notcalled',
               comp_iter=lambda: test_expression_result(incorrect_msg = 'iterincorrect
↪'),
               iter_vars_names=True,
               incorrect_iter_vars_msg='incorrectitervars',
               body=lambda: test_expression_result(context_vals = ['a', 2], incorrect_
↪msg = 'bodyincorrect'),
               ifs=[lambda: test_function_v2('isinstance', params = ['obj'], do_eval␣
↪= [False], not_called_msg = 'notcalled1', incorrect_msg = 'incorrect2'),
                    lambda: test_function_v2('isinstance', params = ['obj'], do_eval␣
↪= [False], not_called_msg = 'notcalled2', incorrect_msg = 'incorrect2')],
               insufficient_ifs_msg='insufficientifs')
```

Result

```
notcalled1
```

No error

13.21.16 Example 16

PEC

```
x = {'a': 2, 'b':3, 'c':4, 'd':'test'}
```

Solution code

```
[key + str(val) for key,val in x.items() if isinstance(key, str) if isinstance(val,␣
↪int)]
```

Student code

```
[key + str(val) for key,val in x.items() if isinstance(key, str) if hasattr(key, 'test
↪')]
```

No output

SCT

```
test_list_comp(index=1,
               not_called_msg='notcalled',
               comp_iter=lambda: test_expression_result(incorrect_msg = 'iterincorrect
↪'),
               iter_vars_names=True,
               incorrect_iter_vars_msg='incorrectitervars',
               body=lambda: test_expression_result(context_vals = ['a', 2], incorrect_
↪msg = 'bodyincorrect'),
               ifs=[lambda: test_function_v2('isinstance', params = ['obj'], do_eval␣
↪= [False], not_called_msg = 'notcalled1', incorrect_msg = 'incorrect2'),
                    lambda: test_function_v2('isinstance', params = ['obj'], do_eval␣
↪= [False], not_called_msg = 'notcalled2', incorrect_msg = 'incorrect2')],
               insufficient_ifs_msg='insufficientifs')
```

Result

```
notcalled2
```

No error

13.21.17 Example 17

PEC

```
x = {'a': 2, 'b':3, 'c':4, 'd':'test'}
```

Solution code

```
[key + str(val) for key,val in x.items() if isinstance(key, str) if isinstance(val,␣
↪int)]
```

Student code

```
[key + str(val) for key,val in x.items() if isinstance(key, str) if isinstance(key,␣
↪str)]
```

No output

SCT

```
test_list_comp(index=1,
               not_called_msg='notcalled',
               comp_iter=lambda: test_expression_result(incorrect_msg = 'iterincorrect
↪'),
               iter_vars_names=True,
               incorrect_iter_vars_msg='incorrectitervars',
               body=lambda: test_expression_result(context_vals = ['a', 2], incorrect_
↪msg = 'bodyincorrect'),
               ifs=[lambda: test_function_v2('isinstance', params = ['obj'], do_eval␣
↪= [False], not_called_msg = 'notcalled1', incorrect_msg = 'incorrect2'),
                    lambda: test_function_v2('isinstance', params = ['obj'], do_eval␣
↪= [False], not_called_msg = 'notcalled2', incorrect_msg = 'incorrect2')],
               insufficient_ifs_msg='insufficientifs')
```

Result

```
incorrect2
```

No error

13.21.18 Example 18

PEC

```
x = {'a': 2, 'b':3, 'c':4, 'd':'test'}
```

Solution code

```
[key + str(val) for key,val in x.items() if isinstance(key, str) if isinstance(val,␣
↪int)]
```

Student code

```
[key + str(val) for key,val in x.items() if isinstance(key, str) if isinstance(val,␣
↪str)]
```

No output

SCT

```
test_list_comp(index=1,
               not_called_msg='notcalled',
               comp_iter=lambda: test_expression_result(incorrect_msg = 'iterincorrect
↪'),
               iter_vars_names=True,
               incorrect_iter_vars_msg='incorrectitervars',
               body=lambda: test_expression_result(context_vals = ['a', 2], incorrect_
↪msg = 'bodyincorrect'),
```

(continues on next page)

(continued from previous page)

```
                ifs=[lambda: test_function_v2('isinstance', params = ['obj'], do_eval␣
↪= [False], not_called_msg = 'notcalled1', incorrect_msg = 'incorrect2'),
                     lambda: test_function_v2('isinstance', params = ['obj'], do_eval␣
↪= [False], not_called_msg = 'notcalled2', incorrect_msg = 'incorrect2')],
                insufficient_ifs_msg='insufficientifs')
```

Result

```
Great work!
```

No error

13.21.19 Example 19

No PEC

Solution code

```
[[col for col in range(5)] for row in range(5)]
```

Student code

```
[[col + 1 for col in range(5)] for row in range(5)]
```

No output

SCT

```
test_list_comp(1, body = lambda: test_list_comp(1, body = lambda: test_expression_
↪result(context_vals = [4])))
```

Result

```
Check the first list comprehension. Did you correctly specify the body? Expected
↪<code>4</code>, but got <code>5</code>.
```

No error

13.21.20 Example 20

No PEC

Solution code

```
[[col for col in range(5)] for row in range(5)]
```

Student code

```
[[col for col in range(5)] for row in range(5)]
```

No output

SCT

```
test_list_comp(1, body = lambda: test_list_comp(1, body = lambda: test_expression_
↪result(context_vals = [4])))
```

Result

```
Great work!
```

No error

13.21.21 Example 21

PEC

```
x = {'a':1, 'b':2}
```

Solution code

```
[key for key, value in x.items()]
```

Student code

```
[a for a in x.items()]
```

No output

SCT

```
test_list_comp(1, iter_vars_names=False)
```

Result

```
Check the first list comprehension. Have you used 2 iterator variables?
```

No error

13.21.22 Example 22

PEC

```
x = {'a':1, 'b':2}
```

Solution code

```
[key for key, value in x.items()]
```

Student code

```
[a for a in x.items()]
```

No output

SCT

```
Ex().check_list_comp(0).has_context()
```

Result

```
Check the first list comprehension. Have you used the correct iterator variable names?
↪ Was expecting <code>(key, value)</code> but got <code>a</code>.
```

No error

13.21.23 Example 23

PEC

```
x = {'a':1, 'b':2}
```

Solution code

```
[key for key, value in x.items()]
```

Student code

```
[a for a,b in x.items()]
```

No output

SCT

```
test_list_comp(1, iter_vars_names=False)
```

Result

```
Great work!
```

No error

13.21.24 Example 24

PEC

```
x = {'a':1, 'b':2}
```

Solution code

```
[key for key, value in x.items()]
```

Student code

```
[a for a,b in x.items()]
```

No output

SCT

```
Ex().check_list_comp(0).has_context()
```

Result

```
Great work!
```

No error

13.22 test_v2_only.py

13.22.1 Example 1

No PEC

Solution code

```
x = round(5)
```

Student code

```
x = round(4)
```

No output

SCT

```
test_object('x')
```

Result

```
Did you correctly define the variable <code>x</code>? Expected <code>5</code>, but␣
↪got <code>4</code>.
```

No error

13.22.2 Example 2

No PEC

Solution code

```
x = round(5)
```

Student code

```
x = round(4)
```

No output

SCT

```
Ex().test_object('x')
```

Result

```
Did you correctly define the variable <code>x</code>? Expected <code>5</code>, but␣
↪got <code>4</code>.
```

No error

13.22.3 Example 3

No PEC

Solution code

```
x = round(5)
```

Student code

```
x = round(4)
```

No output

SCT

```
test_function('round')
```

Result

```
Check your call of <code>round()</code>. Did you correctly specify the first argument?
↪ Expected <code>5</code>, but got <code>4</code>.
```

No error

13.22.4 Example 4

No PEC

Solution code

```
x = round(5)
```

Student code

```
x = round(4)
```

No output

SCT

```
Ex().check_object('x').has_equal_value()
```

Result

```
Did you correctly define the variable <code>x</code>? Expected <code>5</code>, but␣
↪got <code>4</code>.
```

No error

13.22.5 Example 5

No PEC

Solution code

```
x = round(5)
```

Student code

```
x = round(4)
```

No output

SCT

```
Ex() >> check_object('x').has_equal_value()
```

Result

```
Did you correctly define the variable <code>x</code>? Expected <code>5</code>, but␣
↪got <code>4</code>.
```

No error

13.22.6 Example 6

No PEC

Solution code

```
x = round(5)
```

Student code

```
x = round(4)
```

No output

SCT

```
x = check_object('x').has_equal_value(); Ex() >> x
```

Result

```
Did you correctly define the variable <code>x</code>? Expected <code>5</code>, but␣
↪got <code>4</code>.
```

No error

13.22.7 Example 7

No PEC

Solution code

```
x = round(5)
```

Student code

```
x = round(4)
```

No output

SCT

```
Ex().check_object('x').has_equal_value()
```

Result

```
Did you correctly define the variable <code>x</code>? Expected <code>5</code>, but␣
↪got <code>4</code>.
```

No error

13.22.8 Example 8

No PEC

Solution code

```
x = round(5)
```

Student code

```
x = round(4)
```

No output

SCT

```
Ex() >> check_object('x').has_equal_value()
```

Result

```
Did you correctly define the variable <code>x</code>? Expected <code>5</code>, but␣
↪got <code>4</code>.
```

No error

13.22.9 Example 9

No PEC

Solution code

```
x = round(5)
```

Student code

```
x = round(4)
```

No output

SCT

```
Ex().check_or(check_object('x').has_equal_value(), check_object('x').has_equal_
↪value())
```

Result

```
Did you correctly define the variable <code>x</code>? Expected <code>5</code>, but␣
↪got <code>4</code>.
```

No error

13.22.10 Example 10

No PEC

Solution code

```
x = round(5)
```

Student code

```
x = round(4)
```

No output

SCT

```
x = check_object('x').has_equal_value(); Ex() >> x
```

Result

```
Did you correctly define the variable <code>x</code>? Expected <code>5</code>, but␣
↪got <code>4</code>.
```

No error

For details, questions and suggestions, contact us.

Python Module Index

p

Index

I

M

O

P

R

S

U