Documentation, Unit Tests

ÉCOLE NORMALE SUPÉRIEURE DE LYON

Ali Farnudi, Fall 2021

Why document code?

Pros

- **Accelerates team member communication**
- short on-board time
- Organise big projects
- **High development speed**

Cons

- Time (and money) consuming
- Quickly gets out of date
- Developers don’t like it

Wrap up last week's work

Finish the station model

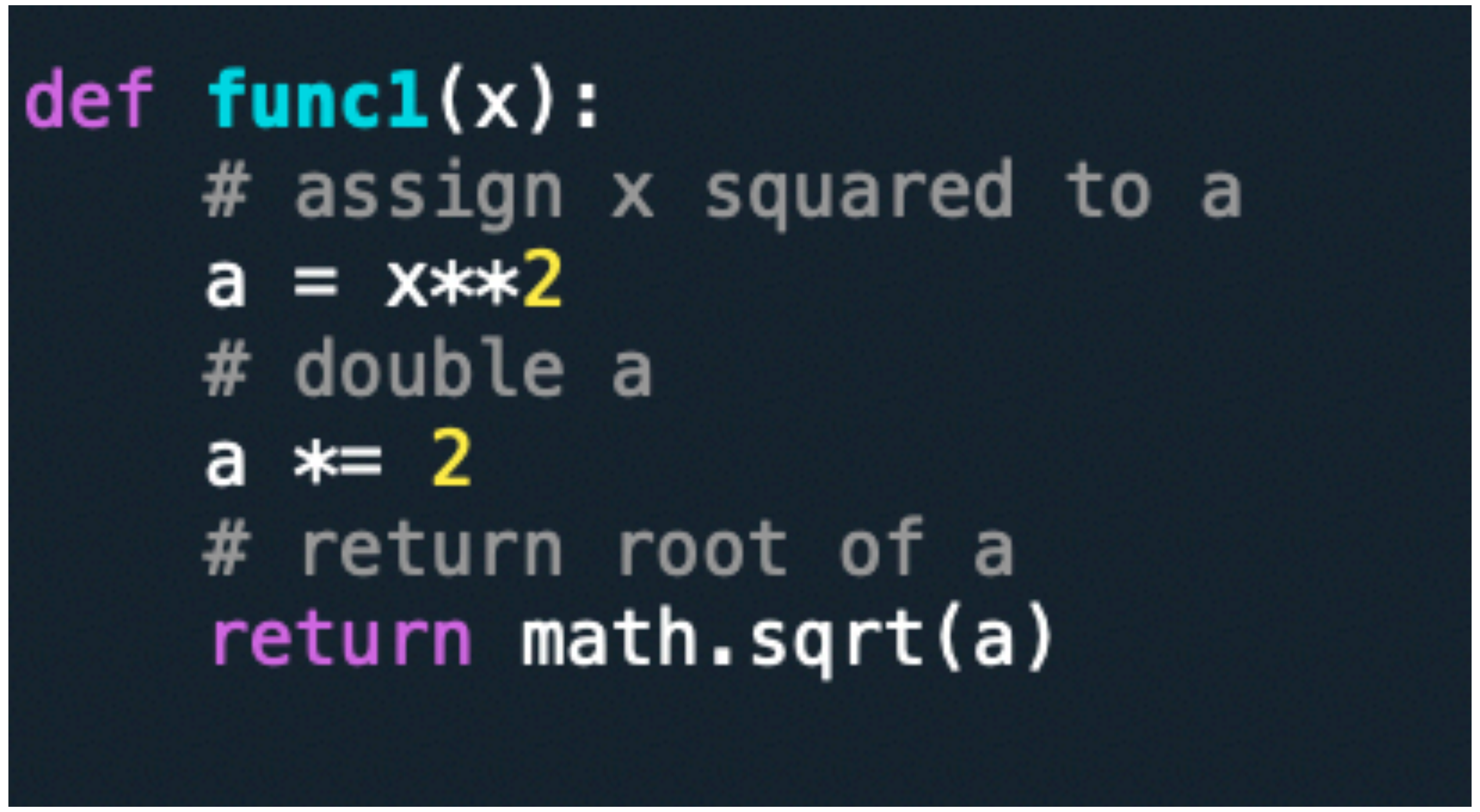
- Finish the code
- Re-factorise the module
- Check with PEP8
- Add documentation
- Check with PEP 257
- Generate a simple website

- pycodestyle (PEP 8)
- pydocstyle (PEP 257)
- Black

PEP 257 website

Hands on - pydocstyle, black, etc

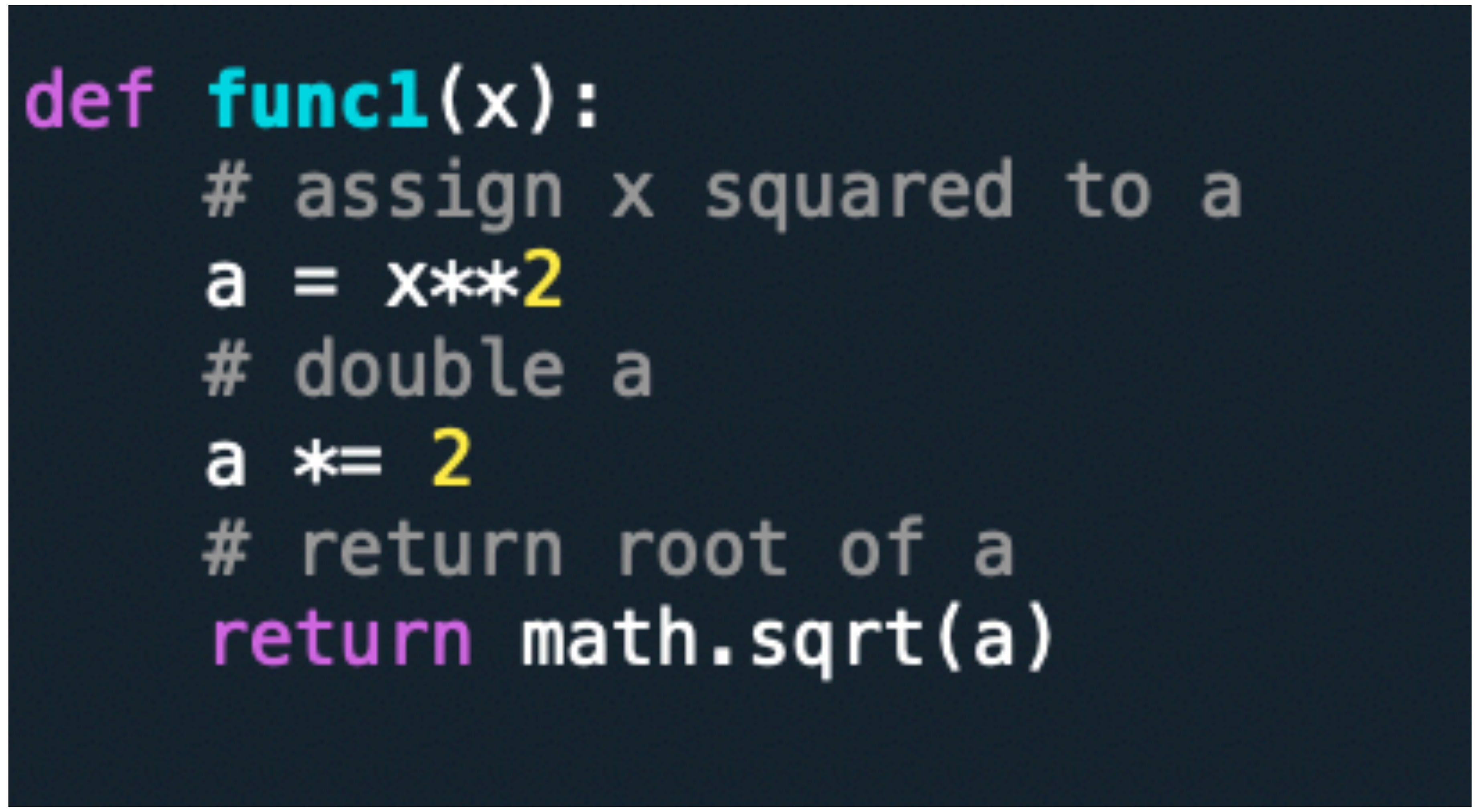
Comments



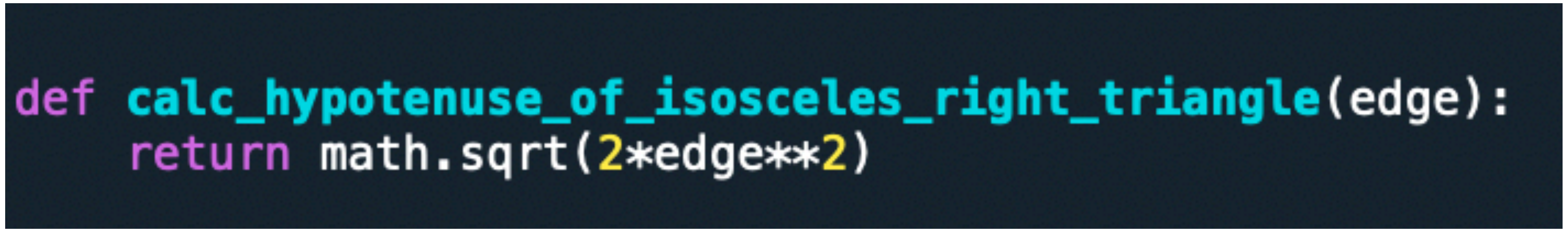
```
def func1(x):
    # assign x squared to a
    a = x**2
    # double a
    a *= 2
    # return root of a
    return math.sqrt(a)
```

Comments

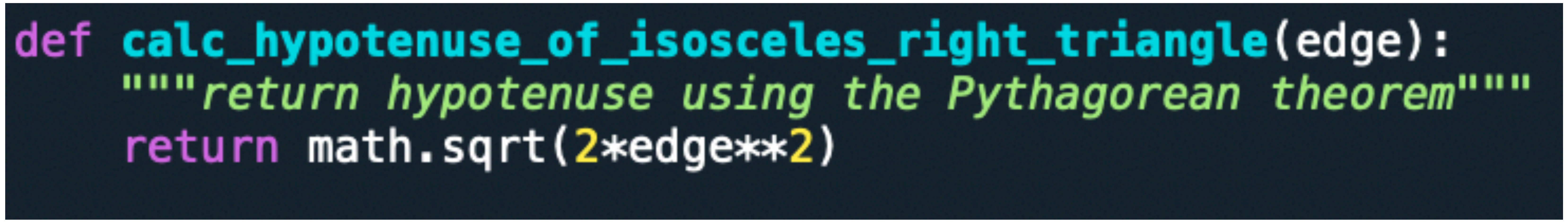
- Explain intentions, not what code does
- Deviations from standard
- Unexpected choices of implementation



```python
def func1(x):
    # assign x squared to a
    a = x**2
    # double a
    a *= 2
    # return root of a
    return math.sqrt(a)
```



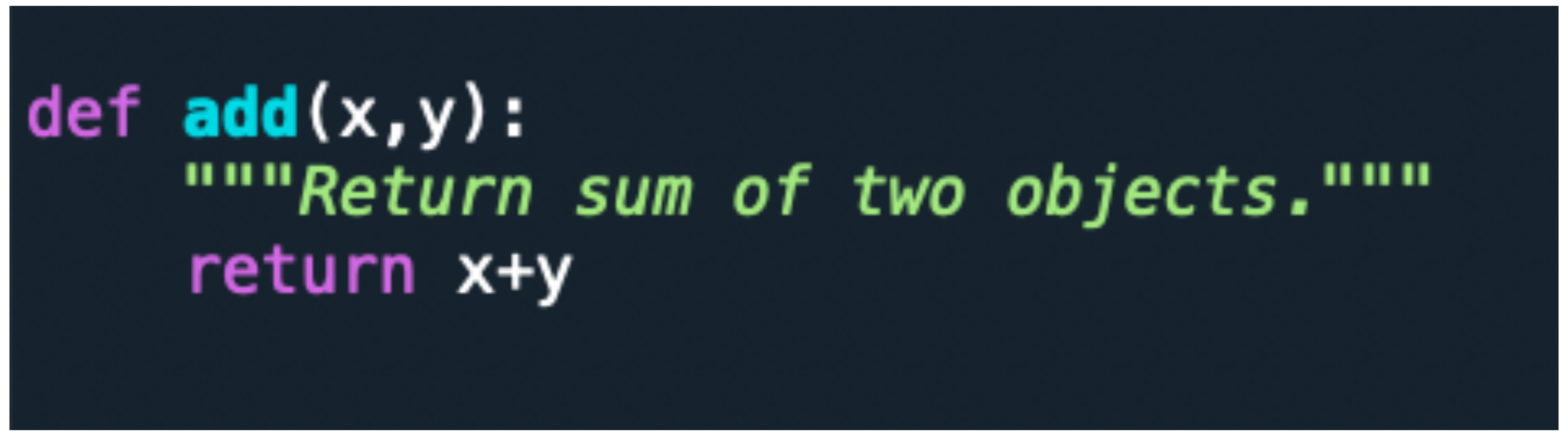
```python
def calc_hypotenuse_of_isosceles_right_triangle(edge):
    return math.sqrt(2*edge**2)
```



```python
def calc_hypotenuse_of_isosceles_right_triangle(edge):
    """return hypotenuse using the Pythagorean theorem"""
    return math.sqrt(2*edge**2)
```

Docstrings and PEP 257

one-line

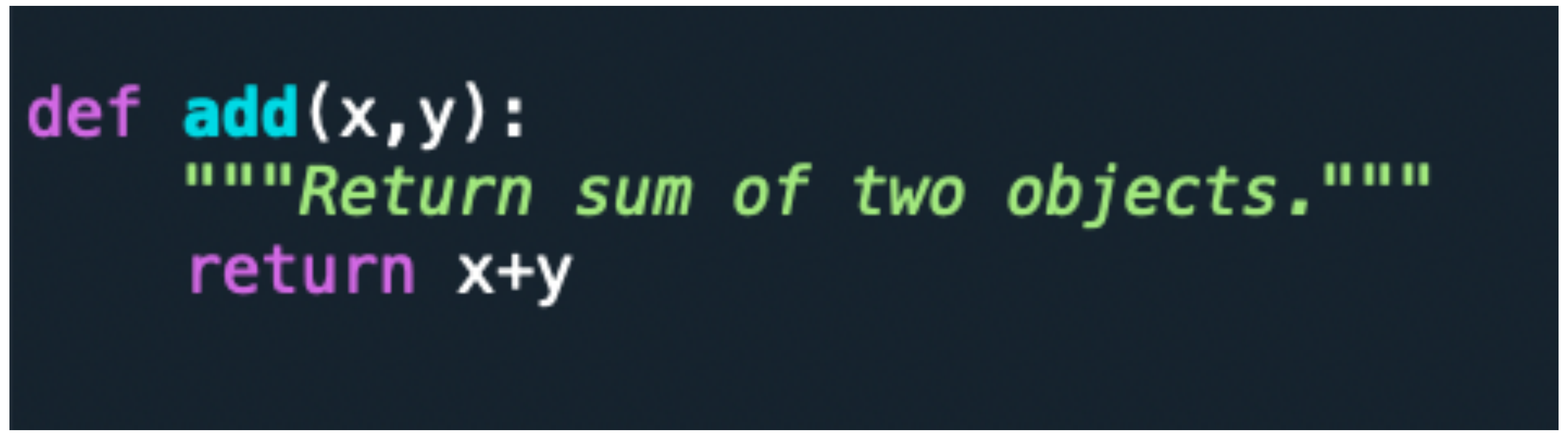


```
def add(x,y):
    """Return sum of two objects."""
    return x+y
```

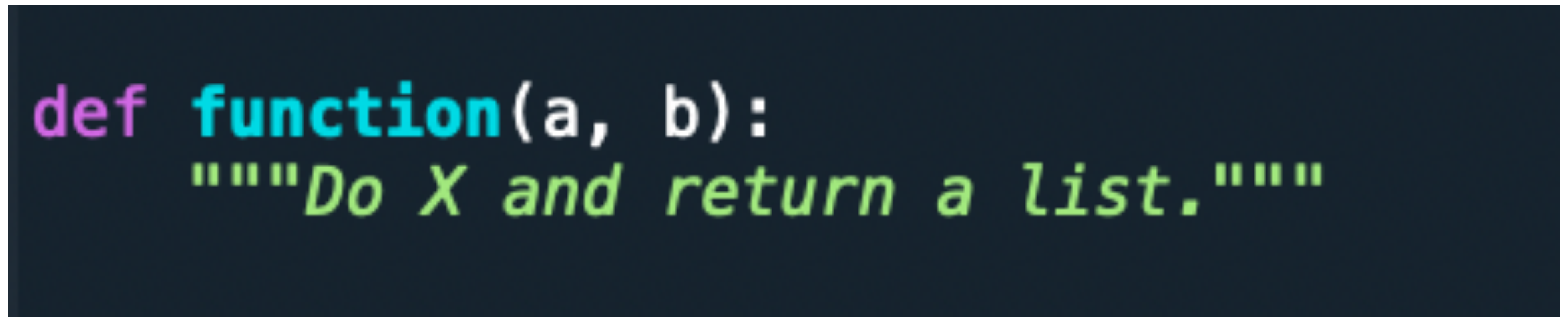
- String literal: The first statement in a module, function, etc
- All modules, functions, and classes should normally have docstrings
- “““triple double quotes”””

Docstrings and PEP 257

one-line



```
def add(x,y):
    """Return sum of two objects."""
    return x+y
```



```
def function(a, b):
    """Do X and return a list."""
```

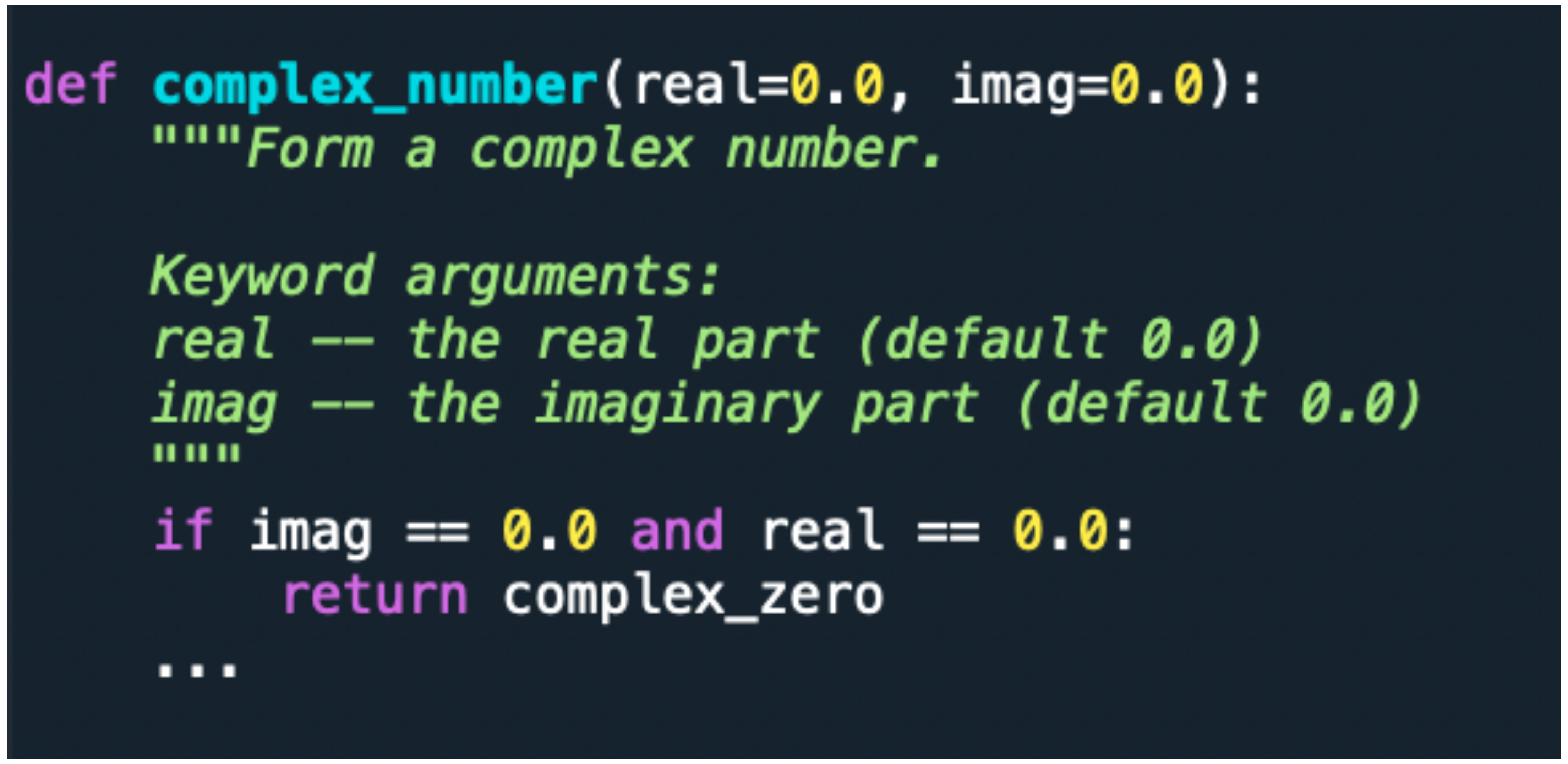
- String literal: The first statement in a module, function, etc
- All modules, functions, and classes should normally have docstrings
- """triple double quotes"""

- It should be a command ("Do this", "Return that"),
- Don’t describe: "Returns the pathname ..."
- Nature of return value should be mentioned

Docstrings and PEP 257

Multi-line

- A summary line (like a one-line) + a blank line,
- More description
- Everythin is indented the same as the quotes
- Numpy Style
- Pandas docstrings
- Google Style



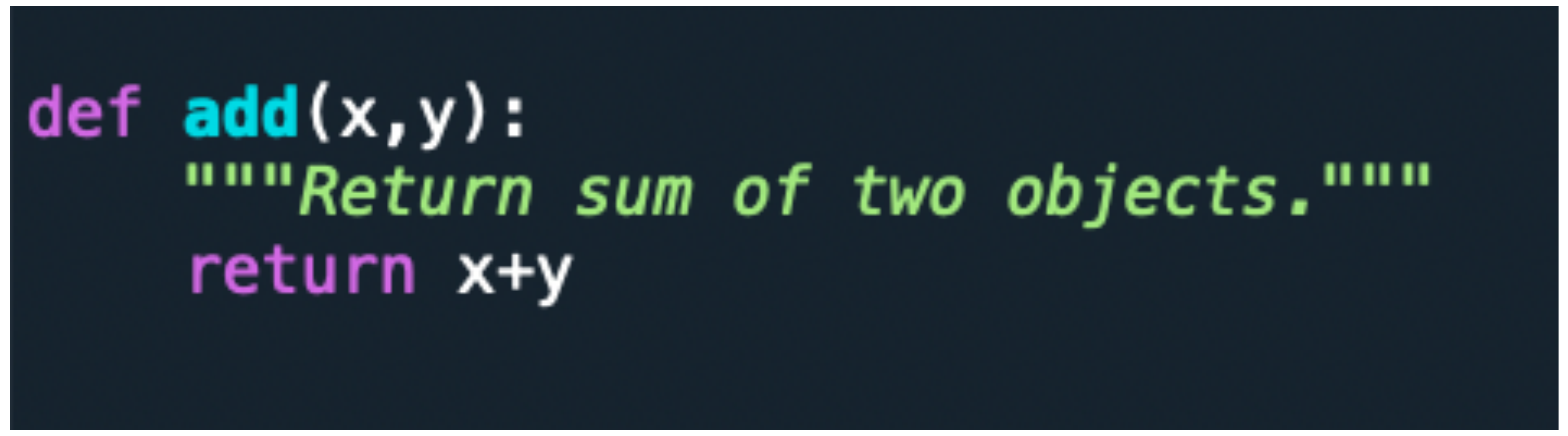
```python
def complex_number(real=0.0, imag=0.0):
    """Form a complex number.

    Keyword arguments:
    real -- the real part (default 0.0)
    imag -- the imaginary part (default 0.0)
    """
    if imag == 0.0 and real == 0.0:
        return complex_zero
    ...
```

Unit Testing

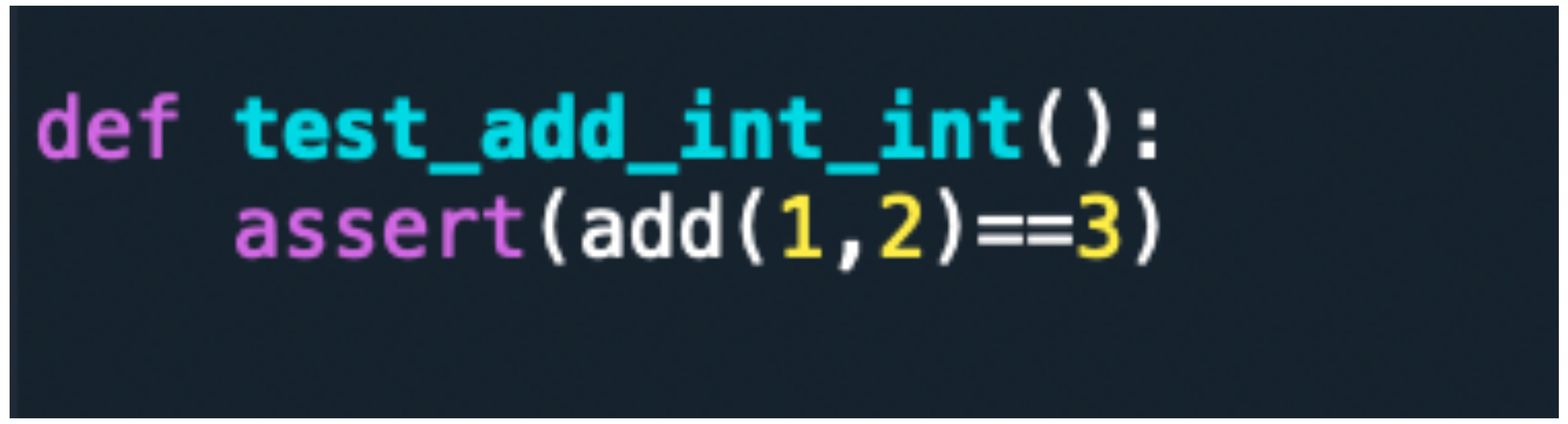
Unit tests

Check if a single unit of code works as expected/desired

They should be small, precise, and independent



```
def add(x,y):
    """Return sum of two objects."""
    return x+y
```



```
def test_add_int_int():
    assert(add(1,2)==3)
```

Why Unit Tests?

- **Fix bugs** + Make sure they are not reproduced
- Helps with refactoring
- Like a documentation (but it is compiled/interpreted)

UT frameworks

- **Python**: pytest, nose, doctest, unittest
- **C++**: Catch, Google Test, Boost.Test, CppUnit, ...
- Get tools to make things easier (automation, reports, fixtures, ...)

UT frameworks

- **Python**: pytest, nose, doctest, unittest
- **C++**: Catch, Google Test, Boost.Test, CppUnit, ...
- Get tools to make things easier (automation, reports, fixtures, ...)

Moc Objects

- Commonly used in testing OO code
- Create objects that are difficult include
  - Non-controlled or non-deterministic behaviour (current time, current temperature, ...)
  - State difficult to reproduce (network error, large database, ...)

UT frameworks

- **Python**: pytest, nose, doctest, unittest
- **C++**: Catch, Google Test, Boost.Test, CppUnit, …
- Get tools to make things easier (automation, reports, fixtures, …)

Moc Objects

- Commonly used in testing OO code
- Create objects that are difficult include
  - Non-controlled or non-deterministic behaviour (current time, current temperature, …)
  - State difficult to reproduce (network error, large database, …)

Test fixtures

- Set up (preconditions)
- Assert
- Tear down (postconditions)

Hands on - Unit Test 1

- Standalone
- Capitalise
- Factorial
- Accumulator

Test Driven Development

- Write unit tests that fails
- Write the minimum (sensible) code to pass them
- Refactor

full test coverage and less useless code

Hands on - Wallet test

Best practices

- If you find a bug, turn it into a test case
- When debugging, write tests
- Always leave the code in a better state than you found it in

Hands on - add fixture to the palindrome tests

Hands on - Station tests