System development life-cycle
Testing

Petr Svarny, 2020

From just programming to good programming.

Analysis,
Design,
Tests,
Style.

Testing

- Code that verifies the functionality of our main code
- Usually automated
- Can be used for development (Test Driven Development)
  - Test defines what behaviour I expect
- Various focuses:
  - Unit testing
  - Integration
  - User
  - … (see for example here)

Importance of multiple testing

Python basic tests

See for example Real Python or python-guide

```
assert sum([1, 2, 3]) == 6, "Should be 6"
```

Usually written as procedures:

```
def test_sum():
    assert sum([1, 2, 3]) == 6, "Should be 6"
```

Test runner example - unittest

- Part of Python

```
import unittest


class TestSum(unittest.TestCase):

   def test_sum(self):
       self.assertEqual(sum([1, 2, 3]), 6, "Should be 6")

   def test_sum_tuple(self):
       self.assertEqual(sum((1, 2, 2)), 6, "Should be 6")

if __name__ == '__main__':
   unittest.main()
```

```
python test_sum_unittest.py
```

Test runner example - pytest

- Separate package
- Runs all test_ files in the project

```
def test_answer():
    assert func(3) == 5
```

```
py.test
```

More advanced tools

Hypothesis - allows various parametrizations of the inputs, i.e. test more at once

Mock - helps class testing by allowing mock classes

Exercise

Using unittest, write tests for the following method:

```python
def positive_sum(in_list):
    """Adds together the members of a list.

    Args:
        in_list (list): List of positive numbers.

    Returns:
        int, float: Depending on the input numbers, returns the sum of them.

    Raises:
        TypeError: In case the in_list is not a list.
        ValueError: In case the any number in the list is negative.
    """
    if not type(in_list) == list:
        raise TypeError("The input is supposed to be list.")
    if any([x < 0 for x in in_list]):
        raise ValueError("The list members are supposed to be non-negative.")
    return sum(in_list)
```