Würgeschlange 3 - Lesson 7

Tobias Maschek, Viktor Reusch
https://github.com/jemx/wise1920-python

mit Materialien von Felix Döring, Felix Wittwer https://github.com/fsr/python-lessons
Lizenz: CC BY 4.0 https://creativecommons.org/licenses/by/4.0/

10. Dezember 2019

Python-Kurs

Gliederung

1. Docstrings

2. Tests

Docstrings

Docstrings - Call the Doctor

- Pakete, Module, Klassen und Funktionen können dokumentiert werden.
- Dokumentation dient dazu, Code für die Nachwelt verständlich zu hinterlassen.
- Ermöglicht Reflexion über den geschriebenen Code und seinen Zweck
- Hilft dem schnellen Verständnis beim Lesen *unbekannten* Codes
- IDEs können mit Dokumentation für Parameter und Rückgabetypen arbeiten
- Docstrings stehen zwischen nach Klassen- und Methodenköpfen
- Auch am Anfang von Modulen und in der __init__.py nutzbar

Type Annotations - Type the Duck Typing

- Dokumentation von Parameter- und Rückgabe-Typen
- Kann von IDEs automatisch verwendet werden
- Nur Dokumentation $\rightarrow$ **Funktion akzeptiert immer noch falsche Typen**
- typing Paket ermöglicht spezielle Typen zu annotieren (List, Union)
- Type Annoations sind auch im Code mit dem type: TYP möglich

Docstrings & Type Annotations — 1

```python
"""A math helper module"""
import math

def sin(angle: float, degree: bool = False) -> float:
    """Calculate the sinus function for this angle.

    :param angle: the angle that calculate the sin for
    :param degree: should be set to True, if the angle is given in degrees (default False)
    :return: the sinus of the angle
    """
    if degree:
        angle = angle / 180 * math.pi
    return math.sin(angle)
```

Docstrings & Type Annotations — 2

```python
from typing import Optional, List, Union

def pop_and_increment_if_exists(list: List[Union[int, float]], index: int =
    -1) -> Optional[int]:
    """Removes an element from the list and returns this element incremented
    by 1

    :param list: a list of integers or floats
    :param index: the index of the element to pop (default -1)
    :return: the poped element plus 1 or None if the element was not found
    """
    try:
        element = list.pop(index)  # type: int
        return element + 1
    except IndexError:
        return None
```

Tests

But why tests?

- Automatische Überprüfung von Code (nach Änderungen, vor Releases)
- Kein manuelles Ausprobieren
- Testen aller möglichen Grenzfälle
- Ermöglicht Testen von internen, von außen unerreichbaren, Funktionen
- Ermöglicht unabhängiges Testen einzelner Komponenten
- Validiert erfolgreiche Integration in anderen Code
- Keine manuellen, aufwendigen Abnahmetests sondern *continuous integration* oder *test driven development*

Offizielle Python Dokumentation

Unittest - Beispiel

```python
import unittest

class TestExample(unittest.TestCase):
    def test_abs(self):  # tests need to start with test
        self.assertEqual(abs(-4), 4)
        self.assertNotEqual(abs(-1.5), -1.5)

    def test_even_odd(self):
        self.assertFalse(4 % 2)
        self.assertTrue(3 % 2)

    def test_dict(self):
        d = {1: 123, "a": "abc"}
        self.assertNotIn(2, d)
        with self.assertRaises(KeyError):
            d[2]
if __name__ == '__main__':
    unittest.main()
```

Unittest - Beispiel

```python
# test.py

import unittest
from prime import is_prime

class TestExample(unittest.TestCase):
    def test_abs(self):
        self.assertFalse(is_prime(2))
        self.assertTrue(is_prime(13))


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

```python
# prime.py

def is_prime(x):
    for y in range(1, x):
        if x % y == 0:
            return False
    return True
```

To assert or to assertNot, that is the question! — hamlet.py

Methode	**Überprüft**
assertEqual(a, b)	a == b
assertNotEqual(a, b)	a != b
assertTrue(x)	bool(x) is True
assertFalse(x)	bool(x) is False
assertIn(a, b)	a in b
assertIsInstance(a, b)	isinstance(a, b)
assertIsNone(x)	x is None
assertAlmostEqual(a, b)	a und b *(float)* bis auf Rundungsfehler gleich
assertGreater(a, b)	a > b
assertGreaterEqual(a, b)	a >= b

Unittest - setUp tearDown

```python
# tests.py
import unittest
import requests

class FileTestCase(unittest.TestCase):
    def setUp(self):  # needs to be called exactly setUp
        self.session = requests.Session()
        self.session.get('https://httpbin.org/cookies/set/mycookie/
    i_love_cookies')

    def test_cookie(self):
        response = self.session.get('https://httpbin.org/cookies')
        self.assertIn("i_love_cookies", response.text)

    def tearDown(self):
        self.session.close()
```

Ausführen mit: *python -m unittest tests.FileTestCase.test_cookie*

Aufgabe 7-1

Der Weihnachtsmann hat seine Wichtel Alice und Bob beauftragt, ein Python-Programm zu schreiben, das entscheidet, ob ein Kind artig war und ein Geschenk bekommt.
Leider haben Eve und Malory den Code in die Hände bekommen und mit ihm Schabernack getrieben.
Hilf Alice und Bob ihr Programm zu retten!

- Schreibt Tests, die die Funktionalität des Algorithmus überprüfen.
- Führt die Tests aus, und schaut, was alles schief geht.
- **Danach** korrigiert die Fehler.
- Die vorhanden Tests dürfen dabei **nicht (!!!)** verändert werden.
- Die Dokumentation wurde von Eve und Malory nicht verändert!

Den zu testenden Code gibt es unter
https://github.com/jemx/wise1920-python/tree/master/07lesson/code/