PyLadies

(Vienna) 14.5.2020

Who?

International mentorship group with a focus on helping more women become active participants and leaders in the Python open-source community.

Our mission is to promote, educate and advance a diverse Python community through outreach, education, conferences, events and social gatherings.

Agenda for today (remote)

---

1. Exceptions
2. Testing (Unittest)
3. Web testing (Selenium)

Goals

- Make tests for all of your codes
- Prevent crashes
- Be awesome and share love for Python!

Resources not only for this lecture

- Cheat sheet for Python: https://github.com/ehmatthes/pcc/releases/download/v1.0.0/beginners_python_cheat_sheet_pcc_all.pdf
- Geeks for geeks: https://www.geeksforgeeks.org/automated-software-testing-with-python/

Errors, Exceptions

- Every programming language has a way to tell you that there is an error (syntactic or other) in your code
- Python doing it via exceptions - tells you a line of code where it realized the error
- You are the one to fix it or avoid it

```
print(Tell me a joke)
  File "<stdin>", line 1
    print(Tell me a joke)
                ^
SyntaxError: invalid syntax
```

Errors, Exceptions

- Python can list more lines. - Nested functions
- Notion of error line does not have to 100% correspond to where we made it (indentation errors)

```
def outer_function():
    return inner_function(0.0)
def inner_function(divisor):
    return 1 / divisor
print(outer_function())
```

```
Traceback (most recent call last):
  File "test.py", line 7, in <module>
    print(outer_function())
  File "test.py", line 2, in outer_function
    return inner_function(0.0)
  File "test.py", line 5, in inner_function
    return 1 / divisor
ZeroDivisionError: float division by zero
```

Raising an error

- You can create exceptions using **raise**
- Usually by writing name of exception class and the message as a string as first parameter
- Built-in exceptions -> docs

```
LIST_MAX = 10
def verify_number(number):
    if 0 <= number < LIST_MAX:
        print('OK!')
    else:
        raise ValueError('{n} is not in the
list!'.format(n=number))
```

Try / except / finally

- To avoid program fail on Exception use **try/except** block
- Python runs code in **try**, if error occurs, instead runs except block, always runs **finally** block
- all previous steps inside **try** until the **Exception** are valid
- Only first except block able to handle **Exception** evaluates

```
try:
    ... potentially faulty code
except ValueError:
    ... do something else
except NameError:
    ... do nothing? - pass
finally:
    ... always finish with this
```

Try / except / finally

- Do not catch them all - catching **KeyboardInterrupt** Exception would prevent cancelling program with **Ctrl + C**
- Catching a general exception will catch child entity (ArithmeticError also catches ZeroDivisionError)
- use **try/except** when you expect that a fault can happen - wrong user input, connection problems, someone using your library in a wrong way
- Custom-made Exceptions based on **Exception** class

Quest - handling user input with custom exceptions

- Simple letter guess game with input
- save random lowercase character of the alphabet
- user tries to guess it until it gets it right, but a hint is given each time if his guess was before or after letter

  Create custom exception for: lower, higher, is a letter of alphabet

Testing

- Bigger programs are hard to test, by running them you will not discover all problems
- Basic python library - unittest
- Easier wrapper - pytest library
- pip install pytest

Tests

- in pytest (also in unittest) test functions need to start with name test_ or end with _test

```
def sum_it(a, b):
    return a + b

def test_sum_it():
    assert sum_it(1, 2) == 3
```

Assert

- Assert command is evaluating expression; if the result is not True, raise an exception. This exception causes failing of the test.
- Can be seen as:

```
if not (a == b):
    raise AssertionError('Test failed!')
```

How to run tests

```
python -m pytest -v test_sum_it.py
```

- This will run whole script and run all functions starting with test_
- You can also just type name of the folder and python will run all the tests in files there
- usually have test in separate file and import module into the test file only

Positive x Negative tests

- Usually testing if code is working correctly - positive testing
- But you can also test if something is behaving as expected when you insert wrong values (if it returns correct message and so on)
- In general, negative tests are important to correctly handle errors
- Better to first write a new test to fail, then correct it

Multiple tests in a file?

- By default all the tests will be executed
- You can select them by parameter k in script call searching for matching pattern:

```
python -m pytest -v test_sum_it.py -k sum_
```

- or by creating markers
  @pytest.mark.set1 - above function definition, then call with -m argument

```
python -m pytest -v test_sum_it.py -m set1
```

Pytest and methods

- Pytest has lot of methods to ease the testing
- Fixture - when we want to run some code before every test method -> @pytest.fixture
- Parametrize - used for testing multiple arguments

```
@pytest.mark.parametrize("input1, input2, output",[(5,5,10),(3,5,12)])
def test_add(input1, input2, output):
    assert input1+input2 == output,"failed"
```

Exercises - test already existing functions

---

1. Write a function and test for calculation of power (eg. $2^2=4$
2. Test **sum(iterable, start)** (on few types of iterables)
3. Write parameterized test for fibonacci sequence
4. Test string operations (like lowercase, uppercase, character replacement)
5. Test correct behavior of a method from a class example from last workshop

Web automated testing

- same reason, different way - website should work in an expected way
- automated user actions with python
- easy testing of feature interactions

Web testing requirements

- web driver of your choice - eg. chrome
- library Selenium for python - pip install selenium
- Website you want to test
- write your test code

Example

```
# signing element clicked
browser.find_element_by_xpath("//a[@id ='signing-link']").click()
time.sleep(2)
# Login clicked
browser.find_element_by_xpath("//a[@id ='login-email']").click()
# username send
a = browser.find_element_by_xpath("//input[@id ='ld-email']")
a.send_keys(username)
# password send
b = browser.find_element_by_xpath("//input[@id ='ld-password']")
b.send_keys(password)
# submit button clicked
browser.find_element_by_xpath("//input[@id ='ld-submit-global']").click()
print('Login Successful')
browser.close()
```

Usage

- Not only web testing, usable in many other cases
- Web scraping with selenium - automated data download
- Automated form filling you don’t like
- and many other..

If you have question or you need help with web testing, let me know!

Sum it up

- Usually after successful test, code is automatically released
- Failed test are usually automatically reported
- Try to always test your code

Test your project from last workshop

Test you blackjack code!

As said before, testing is important and can detect problems with your code before you publish it somewhere

You will probably discover errors you didn’t think about when you were writing your code.

Write smaller tests to test individual components, if this is done, test whole game

Resources and materials general

- advent of code - adventofcode.com
- hackerrank - hackerrank.com
- Django Girls django tutorial -> (29th August in Vienna)
- https://www.practicepython.org
- Nice Python exercices at one place
  https://github.com/tystar86/python_exercises
- https://automatetheboringstuff.com
- https://diveintopython3.problemsolving.io

Next topics

Databases

Graphics

Flask

GUI

fill the form regarding your interests please :) → https://forms.gle/UtfgVGe6AhhRwx539

Thank you and see you next time

Coding session - **28.5.2020**

Next workshop - **6.6.2020** Flask!