Podstawy Python 2

Wojciech Barczyński

Spis treści

1. Podstawy przypomnienie

Nasz pierwszy program, naturalnie Hello World.

1. Przygotujemy nasz projekt:

```
$ mkdir -p GIT_USERNAME/nauka_pythona

$ cd GIT_USERNAME/nauka_pythona
```

2. Utwórz plik main.py :

```
print("Hello World!")
```

Uruchom twój pierwszy program:

```
python main.py
```

3. Wprowadzenie do zmiennych 1

```
imie = 'Wiktoria'
print("Hello World!" + imie)
```

4. Wprowadzenie do zmiennych 2

```
imie = 'Wiktoria'
ilosc_ksiazek = 10
srednia = 4.5

print("Hello World!" + imie)
print("Ma " + str(ilosc_ksiazek))
print("Srednia: " + str(srednia))
```

Zmodyfikuj powyższy program, aby był o czymś innym, np:, samochodach, samolotach

5. Kalkulator, środowisko python jest doskonałe do obliczeń:

```
$ python
>>> 1 + 3
# uwaga!:
>>> 10 / 3
>>> 10.0 / 3
>>> float(10) / 3
# ze zlotówek na Euro
>>> 4000 / 4.3
# 30%
>>> (4000.0 * 33) / 100
# zmienne też działają
>>> cena_euro = (4000.0 ) / 4.3
>>> ( cena_euro * 33 ) / 1000
>>> cena_euro
```

Zadanie domowe: przez najbliższy tydzień wykorzystaj pythona do szybkich obliczeń.

6. Krótko o IPython:

```
$ ipython
$ print('jak w python ale wspiera komendy bash')
$ ! ls /home/tester
$ moj_home = ! ls /home/tester
$ moj_home
```

Wskazówka: Warto poświęcić czas, aby poznać bliżej IPython.

7. Funkcje string:

```
marka = 'Pegout'
ilosc_drzwi = 5
pojemność = 1.3

marka_up = marka.upper()

print("Samochod " + marka + " ma " + str(ilosc_drzwi) + " dzwi")
print(marka_up)
print("Pojemnosc po zmianach: " + str(pojemnosc * 2))
```

Korzystając z dokumentacji, znajdź funkcję, która zmieni wszystkie litery w nazwie samochodu na małe litery.

Wykorzystaj dodatkowo dowolną funkcję, którą znajdziesz w dokumentacji.

Co zwrócą następujący kod:

```
marka[-1]
marka[1:-1]
```

8. Napisz program ukrywające hasło:

```
secret -> s****t
haslo -> h***o
```

9. Czytelniejszy kod z *string.format* :

```
imie = 'Ala'
zwierze = 'kot'
print("{0} ma {1}a".format(imie, zwierze))
```

10. Listy 1:

```
samochody = ['syrena', 'poloneza']

print(samochody[0])
print(samochody[1])

for s in samochody:
    print(s)

for idx in range( len(samochody) ) :
    print("idx: " + str(idx) + " : " + samochody[idx])
```

W osobnym pliku napisz program o czymś innym, np., zwierzętach, samolotach, itp.

Co się stanie, gdy dodasz następującą linie do swojego programu:

```
print(samochody[3])
```

11. Listy 2:

```
samochody = ['syrena', 'poloneza', 'fiat', 'kia']
ilosc = [3, 5]

for idx in range( len(samochody) ) :
    print("idx: " + str(idx) + ": " + samochody[idx])
    print(samochody[idx] + " ma ilosc drzwi " + str(ilosc[idx]))
```

W osobnym pliku napisz program o czymś innym, np., zwierzętach, samolotach, itp.

Co zwrócą poniższe linie:

```
samochod[-1]
samochod[1,-1]
samochod[1:]
samochod[10]
```

12. Dictionary 1:

```
samolot = {'name': 'boeing',
           'przebieg': 10000,
           'type': 'pasazerski'}

# in python3 samolot.items()
for key, value in samolot.iteritems():
    print("{0}:{1}".format(key, value))

#
for key in samolot:
    print("{0}:{1}".format(key, samolot[key]))
```

13. Funkcje 1:

```
def print_dict(d):
    for key in samolot:
        print("{0}:{1}".format(key, d[key]))

if __name__ == "__main__":
    samolot = {'name': 'boeing',
           'przebieg': 10000,
           'type': 'pasazerski'}

    print_dict(samolot)
```

14. Funkcje 2:

```
def calucate_vat(netto):
    vat = float(netto * 23)/100
    return vat

if __name__ == "__main__":
    vat = calucate_vat(1000)
    print("{0}".format(vat))
```

15. Możesz również wrzucać całą funkcjonalność w funkcję main():

```
def main():
    print "Tutaj program"

if __name__ == "__main__":
    main()
```

2. Praca z plikami

W ćwiczeniu poznamy, jak zapisywać i odczytywać pliki:

1. Zapisz pliku:

```
file = open("testfile.txt", "w")
file.write("Hello World!")
file.close()
```

Zadanie: korzystając z pętli zapisz do pliku dane o samochodach.

2. Odczyt pliku:

```
file = open("testfile.txt", "r")
print(file.read())
```

3. Bezpieczna praca z plikami:

```
with open('testfile.txt', 'w') as f:
      print(f.read())
```

3. Zadania

Lista zadań do przećwiczenia Pythona z wykładowcą, kilka pomysłów:

1. Magazyn produktów, deklaracja danych oraz raport cen oraz ilości produktu

2. Magazyn z zapisem do pliku raportu

3. Magazyn z zapisem do pliku raportu w CSV

4. Odczyt stanu listy zwierząt z pliku, dodanie nowych i zapis do pliku

Pamiętaj:

- Kiedy zaimplementujesz to, w osobnym terminalu, od razu uruchom program,
- Nie bój się wrzucać print dla zrozumienia co się dzieje,
- Rozwiązuj program w iteracjach, nie od razu musisz mieć zapis do pliku czy podzielony program na funkcje.

Pomocna bibliotek standardowa do wyświetlenia danych.

```
import pprint

s = {"jedno": "ala ma kota",
   "drugie": [1,2,3,4],
   "trzecie": "wiele halasu o",
   "czwarte": range(5,15)}

pprint.pprint(s)
```

Najłatwiejszy sposób pobrania danych od użytkowników:

- *raw_input(<str>)*
- *input*(*<str>)*

Miejsce na notatki:

4. Pierwszy test jednostkowy

Unit tests w oparciu o bibliotekę pytest na postawie przykładowego projektu na githubie.

1. Nasza implementacja kalkulatora – calculator.py:

```
def calculate(a, b, operacja):
    if operacja == "+":
        return a + b
```

2. Dodajmy też test w pliku – test.py:

```
import unittest

from calculator import calculate

class TestCalculator(unittest.TestCase):
    def test_dodawanie(self):
        r = calculate(1, 2, '+')
        self.assertEqual(r, 3)
```

3. Uruchom testy, będzie zielony:

```
$ pytest *.py
```

   Zauważ, aby widzieć print w kodzie:

```
$ pytest --verbose -s *.py
```

4. Zmień test w taki sposób, żeby był na czerwono. Potem ponownie, żeby był zielony.

5. Zadanie, zaimplementuj pozostałe operacje (odejmowanie, mnożenie, dzielenie), praktykując TDD:
    1. Napisz test dla pojedynczej operacji
    2. Test nie będzie przechodził
    3. Zaimplementuj dokładnie jedną operacje matematyczną
    4. Jeśli test przechodzi idź do punktu 1

6. Dodajmy nowy kalkulator str_calculator dla łańcuchów znaków:
    - Dodaj do pliku, calculator.py:

```
def str_calculator(a, b, operacja):
    if operacja == 'concat':
        return a + b
```

- Dodaj do testy dla str_calculator w pliku – test.py:

```
from calculator import str_calculator

class TestStringCalculator(unittest.TestCase):
    def test_concat(self):
        r = str_calculator("a", "b", 'concat')
        self.assertEqual(r, 'ab')
```

7. Praktykując TDD, zaimplementuj 3 inne funkcje naszego kalkulatora:
   - Sprawdzające czy *a* jest w *b*
   - Sprawdzające czy *b* kończy się *a*
   - Dowolną inną operacje

5. Klasy – Podstawy [na następnym zjeździe]

Klasy, w uproszczeniu, pozwalają one w jednym miejscu mieć dane i funkcje do ich przetwarzania

1. Napisz prostą klasę w oparciu o przykład:

```
class MyClass:
    def hello_msg(self):
        return 'hello world'

mc = MyClass()
print(mc.hello_msg())
```

2. Konstruktor:

```
class Samochod:
    def __init__(self, marka, model):
        self.marka = marka
        self.model = model

s = Samochod("fiat", "126")
print(s.marka + " " + s.model)
```