podstawy

January 9, 2021

1 Podstawy Pythona: cz. 1

1.1 9 stycznia 2021

2 Typy, zmienne, wyświetlenie i komentarze

[ ]:
```
print('Hello Python')
```

2.1 Funkcja `print`:

- funkcja print pozwala na wyswietlenie napisu
- `print`(lista argumentów)
- Argumenty podaje się w nawiasie
- napisy umieszcza się w ' ' lub " "

[2]:
```
print('Hello')
print('Python')
print('Hello', 'Python', 3, '!')
```

```
Hello
Python
Hello Python 3 !
```

2.2 Zmienne

- Nazwa zmiennych nie mogę zawierać spacji i zazwyczaj składają się z małych liter i podkreślników.
- Deklarując zmienną musimy podać jej wartość.
- Przypisujemy wartość za pomocą symbolu "=": `zmienna = wartość`.
- Typ zmiennej może się zmieniać w czasie trwania programu.

[2]:
```
wynik = 4 + 6
wynik += 5
wynik = wynik / 3
print(wynik)
```

```
5.0
```

Podstawowe typy danych: * liczby całkowite (`int`): `3`, `1000`, `-100`, `0` * liczby rzeczywiste (`float`): `3.14`, `5.0`, `-0.001` * napisy: (`str`): `"Python"`, `'zalicznie z przedmiotu'`, `"Rock'n'Roll"` * logiczne (`bool`): `True`, `False` * None: `None`

[2]:
```
login = "t.dwojak"
index = 351913
number = 45.6
print("email:", login)
```

```
email: t.dwojak@amu.edu.pl
```

2.3 Napisy

- Tworzymy poprzez umieszczenie tekstu w `""` lub `''`.
- Operator `+` służy do łączenia (konkatenacji) napisów.

[ ]:
```
name = "Tomasz"
surname = 'Dwojak'
full_name = name + ' ' + surname

print('My name is', full_name)
```

[4]:
```

```

```
wartość zmiennej to 42
```

2.4 Wartości logiczne

- Są dwie wartości logiczne: `True` lub `False`.
- Zaprzeczenie odbywa się za pomocą słowa `not`.

[2]:
```
prawda = True
nieprawda = False
print(prawda)
print(nieprawda)
print(not prawda)
```

```
True
False
False
```

2.5 Konwersja typów:

- Konwersja czyli zmiana typu wartości.
- Python na ogół nie wykonuje automatycznej konwersji typów.

[8]:
```
sto_slownie = '100'
wartosc_nominalna = int(sto_slownie)
wartosc_nominalna += 20
```

```
sto_slownie += "20"

print(wartosc_nominalna)
print(sto_slownie)
```

```
120
10020
```

[ ]:
```
ocena = 5
txt = 'moja ocena to ' + str(ocena)

print(txt)
```

2.6 Komentarze

- Komentarze nie są interpretowane (nie zmnieniają sposobu wykonywania programiu)
- Komentarze w Pythonie zaczynają się od znaku '#'
- Istnieją komentarze wielolinijkowe tagowane potrójnym ", czyli""" """

[ ]:
```
print("Bardzo ważna wiadomość") # A to jest komentarz
"""
Komentarz
wielo-
linijkowy
"""
# print("Nie chcę być wydrukowanym")
print("A teraz chcę")
```

Zadanie 1a i 1b.

3 Listy

- W Pythonie nie ma tablic, są listy.
- Tworzymy je za pomocą `[]` lub `list()`.
- Listy mogą przechowywać elementy różnych typów.
- Indeksowanie zaczyna się od 0.
- Funkcja `len` zwraca liczbę elementów listy.
- Przydatne funkcje to:
  - `min`: zwraca najmniejszą wartość z listy.
  - `max`: zwraca największą wartość z listy.
  - `sum` zwracają sumę elementów z listy.

[13]:
```
x = []
x = list()

oceny = [5, 4, 3, 5, 5]
roznosci = [3.14, "pi", ["pi"], 3]
```

```
print("Liczba elementów", oceny)
print("Najwyższa ocena", max(oceny))
print("Najniższa ocena", min(oceny))
```

```
Liczba elementów [5, 4, 3, 5, 5]
Najwyższa ocena 5
Najniższa ocena 3
```

[9]:
```
boroughs = ['The Bronx', 'Brooklyn', 'Manhattan', 'Queens', 'Staten Island']

print(boroughs[0])
biggest_borough = boroughs[1]
```

```
The Bronx
Brooklyn
```

[ ]:
```
oceny = [3, 4, 5, 6]
oceny[0] = 6

print(oceny)
```

3.1 Dodawanie i usuwanie elementów z listy

Istnieją dwie metody: * `append(x)`: dodaje x na koniec listy * `extend(x)`: rozszerza listę o każdy element z x

[13]:
```
engines = []
engines.append('duck-duck-go')
engines.append("yahoo")
print(engines)
```

```
['duck-duck-go', 'yahoo']
```

[9]:
```
mainstream_engines = ["google", 'bing']
engines.extend(mainstream_engines)
print(engines)
```

```
['google', 1, 'sto']
```

[ ]:
```
engines = ['duck-duck-go', 'yahoo']
mainstream_engines = ["google", 'bing']
all_engines = engines + mainstream_engines
print(all_engines)
```

Do usuwania elementów z listy służą następujące metody: * `pop` - usuwanie elementu po indeksie (domyślnie usuwa ostatni element). * `remove` - usuwanie elementu po wartości.

[2]:
```
liczby = [1, 2, 3, 2, 3, 1, 2, 4]

liczby.pop()
liczby.remove(2)
print(liczby)
```

```
[1, 3, 2, 3, 1, 2]
```

3.2 Inne przydatne metody:

- `sort()`: sortuje listę rosnąco (`sort(reverse=True)` – malejąco)
- `count(x)`: zlicza wystąpienia x w liście
- `index(x)`: zwraca indeks pierwszego wystąpienia x

[20]:
```
liczby = [1, 2, 3, 2, 3, 1, 2, 4]
print(liczby.count(1))
print(liczby.index(4))
liczby.sort()
print(liczby)
```

```
2
7
[1, 1, 2, 2, 2, 3, 3, 4]
```

3.3 Indeksowanie

[12]:
```
oceny = [1, 3, 2, 3, 1, 2, 4]
print('pierwszy element:', oceny[0])
print('ostatni element:', oceny[-1])
print('5 pierwszych:', oceny[:5])
print('5 ostatnich', oceny[-5:])
print('od drugiego, do piątego', oceny[1:5])
```

```
pierwszy element: 1
ostatni element: 4
5 pierwszych: [1, 3, 2, 3, 1]
5 ostatnich [2, 3, 1, 2, 4]
od drugiego, do piątego [3, 2, 3, 1]
```

3.4 Zagnieżdżenie list

[ ]:
```
nested = [[1, 2, 3], [4, 5, 6], [7, 8, 9, 10]]

print(nested[1])
print(nested[1][0])
```

3.5 Zadanie 2a, 2b i 2c.

4 Słowniki: `dict`

- pozwala przechować dane w formacie `klucz: wartość`
- Standardowo tworzy się przez `{}` lub `dict()`
- pozwala na dodawanie, usuwanie i zmianę kluczy i wartości
- Odwołanie sie do wartości elementu poprzez `[...]`,
- Funkcja `len` zwróci liczbę elementów w słowniku.

[3]:
```
slownik = {}
oceny = {
    "Justyna" : [5, 5, 5],
    "Bartek" : [3, 4, 5],
    "Ola": [3, 3, 3]
}
print(oceny)
print(oceny['Ola'])
print(len(oceny))
```

```
{'Justyna': [5, 5, 5], 'Bartek': [3, 4, 5], 'Ola': [3, 3, 3]}
[3, 3, 3]
3
```

[6]:
```
oceny = {}
oceny["Jan"] = [3, 4, 5]
print(oceny)
```

```
{'Justyna': [5, 5, 5], 'Jan': [3, 4, 5]}
```

[21]:
```
oceny = {"Jan": [3, 4, 5]}
print(oceny)

oceny["Jan"] = [3, 4, 5, 6]
print(oceny)

oceny["Jan"].append(7)
print(oceny)
```

```
{'Jan': [3, 4, 5]}
{'Jan': [3, 4, 5, 6]}
{'Jan': [3, 4, 5, 6, 7]}
```

[22]:
```
code_responses = {
    404: 'Brak strony',
    500: 'Błąd wewnętrzny serwera',
    200: 'Wszystko OK',
}
```

```
print(code_responses[200])
```

```
Wszystko OK
```

[23]:
```
code_responses = {
    404: 'Brak strony',
    500: 'Błąd wewnętrzny serwera',
    200: 'Wszystko OK',
}

print(code_responses.keys())
print(code_responses.values())
print(code_responses.items())
```

```
dict_keys([404, 500, 200])
dict_values(['Brak strony', 'Błąd wewnętrzny serwera', 'Wszystko OK'])
dict_items([(404, 'Brak strony'), (500, 'Błąd wewnętrzny serwera'), (200,
'Wszystko OK')])
```

[11]:
```
klasa = {
    'studenci': {
        'imie': 'Jan',
        'oceny': [3, 4]
    },
    'zjazdy': ['lipiec', 'sierpień']
}

print(klasa['studenci']['imie'])
```

```
Jan
```

4.1 Zadania 3

5 Instrukcja sterująca `if ... else`

5.1 Operatory porównania

- Porównanie za pomocą `==`
- znak różności: `!=`
- większe, mniejsze: `>`, `>=`, `<`, `<=`
- Sprawdzenie, czy coś jest w liście: `item in l` (zaprzeczenie: `item not in l`)
- Do łączenia warunków służą słowa klucznowe `and` i `or`

[ ]:
```
a = 4
b = 5

print(a==b)
```

[ ]:
```
error_code = "404"

print(error_code == '404')
print(error_code == 404)
```

[4]:
```
numbers = [1,2,3]
print(numbers == [1,2,3])
```

```
True
```

[ ]:
```
print('Python' != 'python')
```

[ ]:
```
print(1000 > 999)
print(999 <= 999)
```

[17]:
```
number = 999
print(number > 800 and number < 1000)
```

```
True
```

5.2 Instrukcja warunkowa `if ... elif ... else:`

- działa tak samo, jak w innych języka.
- Kod po `if`, `elif` i `else` musi być wcięty.
- jest dodatkowa instrukcja następnego warunku *elif*....
- elementy `elif` i `else` są dobrowolne.

[12]:
```
error_code = 404

if error_code == 404:
    print("Strona nie istnieje!")
```

```
Strona nie istnieje!
```

[11]:
```
boroughs = ['The Bronx', 'Brooklyn', 'Manhattan', 'Queens', 'Staten Island']
hotel_borough = 'Brooklyn'

if hotel_borough in boroughs:
    print("Welcome in New York!")
```

```
Welcome in New York!
```

[ ]:
```
number = 300

if number < 0:
    print("Ujemna!")
elif number > 0:
    print("Dodatnia")
```

```
else:
     print("Zero!")
```

Dodatnia

[5]:
```
boroughs = ['The Bronx', 'Brooklyn', 'Manhattan', 'Queens', 'Staten Island']

print('Brooklyn' in boroughs)
```

True

[1]:
```
boroughs = ['The Bronx', 'Brooklyn', 'Manhattan', 'Queens', 'Staten Island']

print('Queens' not in boroughs)
```

False

- każda obiekt ma wartość logiczną `True` lub `False`
- Wartość `False` mają m.in.: `0`, `0.0`, `[]`, `{}`, `''`, `None`.

5.3 Zadania 4a i 4b.

6 Pętla typu `for`

- W Pythonie pętla *for* działa jak pętla *for each* w innych językach;
- Instrukcje zawarte w pętli muszą być wcięte.
- Jeżeli pętla ma się wykonać n razy – użyj funkcji `range()`.

[22]:
```
for i in range(3):
    print('element', i)
```

```
element 0
element 1
element 2
```

[23]:
```
l = ["jeden", "dwa", "trzy"]
for item in l:
    print(item)
```

```
jeden
dwa
trzy
```

[24]:
```
kwadraty = []

for i in range(10):
    kwadraty.append(i ** 2)

print(kwadraty)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

6.1 Słowniki i pętla for

[7]:
```
for osoba in oceny:
    print(osoba,':', oceny[osoba])
```

```
Justyna : [5, 5, 5]
Jan : [3, 4, 5]
```

[8]:
```
for klucz, wartosc in oceny.items():
    print(klucz,':', wartosc)
```

```
Justyna : [5, 5, 5]
Jan : [3, 4, 5]
```

[10]:
```
for i in range(99):
    if (i + 1) % 10 == 0:
        break
    if i < 6:
        continue
    print(i)
```

```
0
1
2
3
4
```

6.2 Zadanie 5a, 5b, 5c, 5d.

7 Praca z tekstem

Tekst (string) w Pythonie * Tworzymy poprzez umieszczenie tekstu w "" lub ''. * Łącznie dwóch stringów (konkatenacja) odbywa się za pomocą operatora + * Napisy w Pythonie są stałe, tzn. nie można ich zmieniać.

[7]:
```
text = 'Wiadomość'

print(text)
print(text[-1])
```

```
Wiadomość
ć
```

[12]:
```
text = 'Python'
text[0] = 'p'
```

[5]:
```
text = "Python"
for znak in text:
    print(znak)
```

```
P
y
t
h
o
n
```

[ ]:
```
slowa = ['Bardzo' , 'ważna', 'wiadomość']
print(' '.join(slowa))
```

[ ]:
```
if 'dom' in napis:
    print(True)
else:
    print(False)
```

[ ]:
```
text = "Bardzo ważna wiadomość"
print(text.split(' '))
```

[ ]:
```
text = "Nie wszyscy lubią spacje na końcu linii.       "
print(text)
print(text.strip(' '))
```

7.1 Zadanie 6a i 6b

8 Funkcje

- wielokrotnie wykorzystanie tego samego kodu
- raz napisany, można wykorzystać w wielu miejscach
- zwiększa czytelność kodu

```
def nazwa_funkcji(arg_1, arg_2, arg_3):
    instrukcja 1
    instrukcja 2
    return jakaś wartość
```

[ ]:
```
def print_welcome():
    print("Witaj!")

print_welcome()
```

[5]:
```
def max2(a, b):
    if a >= b:
        return a
    return b
```

```
print(max2(56, 512))
```

```
512
```

[25]:
```
def srednia(lista):
    s = 0
    for item in lista:
        s += item
    return s / len(lista)

print(srednia([7, 8, 9]))
```

```
8.0
```

[6]:
```
def count(lista, item):
    counter = 0
    for element in lista:
        if element == item:
            counter += 1
    return counter

count([5, 5, 5, 4], 5)
count(lista=[5, 5, 5, 4], item=5)
count(item=5, lista=[5, 5, 5, 4])
```

8.1 Zadania 7a, 7b

9 Obsługa plików tekstowych

- Funkcja open pozwala obsugę plików (czytanie i zapis)

[18]:
```
zen_file = open('./zen_of_python.txt')
zen_text = zen_file.read()

print(zen_text[:95])
zen_file.close()
```

```
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
```

[19]:
```
zen_file = open('./zen_of_python.txt')
i = 0
for line in zen_file:
    print(line.strip())
    i += 1
    if i == 2:
```

```
        break

zen_file.close()
```

```
Beautiful is better than ugly.
Explicit is better than implicit.
```

[20]:
```
with open('./zen_of_python.txt') as zen_file:
    print(zen_file.read()[:95])
```

```
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
```

[ ]:
```
tree_per_sqkm = {
    "Brazil": 39542,
    "Bulgaria": 24987,
}

with open('./lab01/zalesienie.txt', 'w') as plik:
    for country, num_trees in tree_per_sqkm.items():
        plik.write(f'{country}: {num_trees}\n')
```

Zadanie 8

10 Biblioteki

- Python posiada wiele bibliotek standardowych, których są zawsze dostępne.
- Jeszcze więcej bibliotek jest dostępnych na PyPI.
- Żeby skorzystać z biblioteki należy ją dołączyć do programu (*zaimportować*), np. `import sys`.

[34]:
```
import collections

counter = collections.Counter([0, 1, 2, 5, 5, 3, 3, 3, 2])

print(counter[1])
```

```
1
```

[37]:
```
from math import sqrt

print(sqrt(1024))
```

```
32.0
```

[8]:
```
import multiprocessing as mp

pool = mp.Pool(processes=4)
```

[22]:
```
import csv

with open('./gapminder.csv') as csv_file:
    reader = csv.DictReader(csv_file)
    data = list(reader)

print(data[0])
```

```
OrderedDict([('Country', 'Afghanistan'), ('female_BMI', '21.07402'),
('male_BMI', '20.62058'), ('gdp', '1311.0'), ('population', '26528741.0'),
('under5mortality', '110.4'), ('life_expectancy', '52.8'), ('fertility',
'6.2')])
```

Ważniejsze biblioteki: * `os`, `sys`: obsługa rzeczy dt. systemu i środowiska * `datetime`: wszystko co jest związane z czasem * `collections`: zawiera `Counter` i `defaultdict` * `typing`: poprawia anotacje typowania

Zadania 9a i 9b

[ ]:
```

```