podstawy

January 11, 2021

1 Podstawy Pythona: cz. 1

1.1 9 stycznia 2021

2 Typy, zmienne, wyświetlenie i komentarze

[210]:
```
print('Hello Python!')
```

```
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 ””

[211]:
```
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 = 3`.
- Typ zmiennej może się zmieniać w czasie trwania programu.

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

```
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`

[11]:
```
login = "t.dwojak"
index = 351913
number = 45.6
print("email:", login, ' ,', index)
```

```
email: t.dwojak  , 351913
```

2.3 Napisy

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

[14]:
```
name = "Tomasz"
surname = 'Dwojak'
full_name = name + ' ' + surname

print(full_name)
```

```
Tomasz Dwojak
```

[4]:
```

```

```
wartość zmiennej to 42
```

2.4 Wartości logiczne

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

[17]:
```
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.

[19]:
```
sto_slownie = '100'
wartosc_nominalna = int(sto_slownie)
wartosc_nominalna += 20
sto_slownie += "20"

print(wartosc_nominalna)
print(sto_slownie)
```

```
120
10020
```

[29]:
```
dane = 2 ** 4
print(dane)
```

```
16
```

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""" """

[23]:
```
print("Bardzo ważna wiadomość") # A to jest komentarz
"""
Komentarz
wielo-
linijkowy
"""
# print("Nie chcę być wydrukowanym")
print("A teraz chcę")
```

```
Bardzo ważna wiadomość
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.

[37]:
```
x = []
x = list()

oceny = [5, 4, 3, 5, 5.6]
roznosci = [3.14, "pi", ["pi"], 3]

litery = ['a', 'b', 'c']

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

```
Liczba elementów 5
Najwyższa ocena 5.6
Najniższa ocena 3
a
```

[39]:
```
boroughs = ['The Bronx', 'Brooklyn', 'Manhattan', 'Queens', 'Staten Island']

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

```
The Bronx
Brooklyn
```

[40]:
```
oceny = [3, 4, 5, 6]
oceny[0] = 6

print(oceny)
```

```
[6, 4, 5, 6]
```

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

[46]:
```
engines = []
engines.append('duck-duck-go')
engines.append("yahoo")
print(engines)
```

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

[48]:
```
mainstream_engines = ["google", 'bing']
engines.append(mainstream_engines)
```

```
print(engines)
```

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

[50]:
```
engines = ['duck-duck-go', 'yahoo']
mainstream_engines = ["google", 'bing']
all_engines = engines + mainstream_engines + engines
print(all_engines)
```

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

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

[59]:
```
liczby = [1, 2, 3, 2, 3, 1, 2, 4]

#liczby.pop()
liczby.remove(1)
print(liczby)
```

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

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

[62]:
```
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

[69]:
```
oceny = [1, 3, 2, 3, 1, 2, 4]

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

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

3.4 Zagnieżdżenie list

[71]:
```python
nested = [[1, 2, 3], [4, 5, 6], [7, 8, 9, 10]]

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

```
[[0, 0, 0], 5, 6]
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.

[79]:
```python
slownik = {}

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

```
3
```

[82]:
```python
oceny = {}
oceny["Jan"] = 3
print(oceny)
```

```
{'Jan': 3}
```

[83]:
```python
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]}
```

[84]:
```
code_responses = {
    404: 'Brak strony',
    500: 'Błąd wewnętrzny serwera',
    200: 'Wszystko OK',
}

print(code_responses[200])
```

```
Wszystko OK
```

[88]:
```
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_items([(404, 'Brak strony'), (500, 'Błąd wewnętrzny serwera'), (200,
'Wszystko OK')])
```

[91]:
```
klasa = {
    'studenci': {
        'imie': 'Jan',
        'oceny': [3, 4]
    },
    'zjazdy': ['lipiec', 'sierpień']
}

print(klasa['zjazdy'][1])
```

```
sierpień
```

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`

[93]:
```
a = 5
b = 5

print(a == b)
```

```
True
```

[96]:
```
error_code = "404"

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

```
True
False
```

[100]:
```
numbers = [1, 2, 3]
print(numbers == [1, 2, 3])
```

```
True
```

[101]:
```
print('Python' != 'python')
```

```
True
```

[103]:
```
print(1000 > 999)
print(999 >= 999)
```

```
True
True
```

[107]:
```
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.

[115]:
```
error_code = 405

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

print("?")
```

```
?
```

[117]:
```
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!
```

[118]:
```
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
```

[123]:
```
print(bool( [ [] ] ))
```

```
True
```

- 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()`.

[127]:
```
n = 4
for i in range(n):
    print('element', i)
```

```
element 0
element 1
element 2
element 3
```

[128]:
```
l = ["jeden", "dwa", "trzy"]

for item in l:
    print(item)
```

```
jeden
dwa
trzy
```

[129]:
```
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`

[133]:
```
oceny = {'Jan': 2, 'Alina': 3, 'Beata':7}

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

```
Jan : 2
Alina : 3
Beata : 7
```

[137]:
```
for klucz, wartosc in oceny.items():
    print(klucz, ':', wartosc)
```

```
Jan : 2
Alina : 3
Beata : 7
```

[139]:
```
for i in range(99):
    if (i + 1) % 10 == 0:
        break
    if i < 6:
        continue
    print(i)
print("Koniec")
```

```
6
7
8
Koniec
```

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.

[142]:
```
text = "Wiadomość"

print(text)
```

```
Wiadomość
```

[146]:
```
text = 'ąężźł'
print(text[1])
```

```
ę
```

[147]:
```
text = 'Python'
l_text = list(text)

print(l_text)
```

```
['P', 'y', 't', 'h', 'o', 'n']
```

[148]:
```
text = "Python"
for znak in text:
    print(znak)
```

```
P
y
```

```
t
h
o
n
```

[153]:
```python
slowa = ['Bardzo' , 'ważna', 'wiadomość']
# text = slowa[0] + ' ' + slowa[1] + ' ' + slowa[2]
lacznik = '<>'
print(lacznik.join(slowa))
# print(text)
```

```
Bardzo<>ważna<>wiadomość
```

[157]:
```python
napis = 'Wiadomość'

if 'dm' in napis:
    print(True)
else:
    print(False)
```

```
False
```

[161]:
```python
text = "Bardzo ważna wiadomość"
tokens = text.split(' ')
print(' '.join(tokens))
```

```
Bardzo ważna wiadomość
```

[163]:
```python
text = "Nie wszyscy lubią spacje na końcu linii."
print(text)
print(text.strip('.'))
```

```
Nie wszyscy lubią spacje na końcu linii.
Nie wszyscy lubią spacje na końcu linii
```

7.1 Zadanie 6a i 6b

8 Funkcje

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

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

[164]:
```
def print_welcome():
    print("Witaj!")

print_welcome()
```

```
Witaj!
```

[167]:
```
def max2(a, b):
    if a >= b:
        return a
    return b
m = max2(56, 512)
```

[168]:
```
def srednia(lista):
    s = 0
    for item in lista:
        s += item
    return s / len(lista)

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

```
8.0
```

[172]:
```
def count(lista, item=0):
    counter = 0
    for element in lista:
        if element == item:
            counter += 1
    return counter

count([5, 5, 5, 4, 0], 5)
# count(lista=[5, 5, 5, 4], item=5)
# count(item=5, lista=[5, 5, 5, 4])
```

[172]:
```
3
```

8.1 Zadania 7a, 7b

9 Obsługa plików tekstowych

- Funkcja `open` pozwala obsugę plików (czytanie i zapis)

[173]:
```
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.
```

[175]:
```
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.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

[176]:
```
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.
```

[180]:
```
tree_per_sqkm = {
    "Brazil": 39542,
    "Bulgaria": 24987,
```

```
}

with open('./zalesienie.txt', mode='w') as plik:
    for country, num_trees in tree_per_sqkm.items():
#         plik.write(country + ' ' + str(num_trees) + '\n')
        print(country, num_trees, file=plik)
```

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`.

[181]:
```
import collections

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

print(counter[1])
```

```
1
```

[184]:
```
from math import sqrt, pow

print(sqrt(1024))
```

```
32.0
```

[8]:
```
import multiprocessing as mp

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

[185]:
```
import csv

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

print(dict(data[0]))
```

```
{'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'}
```

[4]:
```
from datetime import date
print(date.today())

data = date(2000, 10, 23)

print(data)
```

```
2021-01-09
2000-10-23
```

[192]:
```
import random

print(random.random())
```

```
0.5628789049205796
```

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

11 Różności

11.1 Krotki

- Krotka jest podobna do listy, ale nie można jej zmieniać.
- Stąd podczas tworzenia musimy podać wszystkie jej elementy.
- Krotkę tworzymy za pomocą `()` lub `tuple()`.

[195]:
```
dates = ('2000-01-01', '2001-04-11', '1999-03-12')

print(dates)

print(dates[0])

print(min(dates))
```

```
('2000-01-01', '2001-04-11', '1999-03-12')
2000-01-01
1999-03-12
```

[199]:
```
dates = ('2000-01-01',)

print(dates)
```

```
('2000-01-01',)
```

[203]:
```
boroughs = ['Brooklyn', 'The Bronx']

t = tuple(boroughs)
print(list(t))
```

```
['Brooklyn', 'The Bronx']
```

[205]:
```
t = ('A', 'B', 'C')
d = t + ('E',)
d
```

[205]:
```
('A', 'B', 'C', 'E')
```

[207]:
```
d = ('A', 'B', 'C')
d_list = list(d)
d_list[0] = 'E'
new_d = tuple(d_list)
```

[18]:
```
students = {
    ('Ala', 'Beata'): 1,
    ('Cecylia', 'Darek'): 2,
    ('Eryka', 'Franek'): 3,
}
```

11.2 List comprehensions

[208]:
```
l = [0,0,0,0,0]

l = [i * i for i in range(10)]

l = []
for i in range(10):
    l.append(i * i)

print(l)
```

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

[209]:
```
boroughs = ['Brooklyn', 'The Bronx', 'Queens']

first_letters = [borough[0] for borough in boroughs]
first_letters
```

[209]:
```
['B', 'T', 'Q']
```

[26]:
```
numbers = [3,2,5,6,2,1,7,8]
```

```
greater_than_5 = [number for number in numbers if number > 5]
```

11.3 Podstawy Obiektowości

- W Pythonie każda wartość jest obiektem, tzn. może zostać przypisana do zmiennej lub zostać przekazane do funkcji.
- Każda wartość może mieć metody lub atrybuty.

[30]:
```
from datetime import date

d = date(1999, 12, 12)

d.month # atrybut
d.weekday() # metoda
```

[30]: 6

[32]:
```
import csv

def max(a, b):
    pass

my_best_function = max
my_best_library = csv
```

[ ]:
```

```