Wizualizacja danych
semestr letni 2024

Dr Anna Muranova
UWM w Olsztynie

Wykład 6

Wprowadzenie do Python: powtórzenie

Podstawowe operacje arytmetyczne:

`+, -, *, /, //, %, **`
`>, <, <=, >=, !=, ==`

Styl PEP8

- wymowa: pi-i-pi-ejt
- we wcięciach stosujemy spacje (a nie tabulatory)
- każdy poziom wcięcia powinien składać się z 4 spacji
- wiersz powinien składać się z maksymalnie 79 znaków
- dwie linie między funkcjami najwyższego poziomu i między klasami.
- pojedyncza linia między funkcjami w klasie
- pełna wersja `https://www.python.org/dev/peps/pep-0008/`

Operator warunkowe: if...

```
a = 33
b = 200
if b > a:
  print("b is greater than a")
```

Bez wcięć będzie błąd kompilacji

```
a = 33
b = 200
if b > a:
print("b is greater than a")
```

Operator warunkowe: if... else..., elif

```
a = 200
b = 33
if b > a:
  mmin  = a
else:
  mmin = b
print(mmin)
```

```
a = 200
if a > 0:
  print("a is positive")
elif a < 0:
  print("a is negative")
else:
  print("a is 0")
```

Typu danych w Python

Zmienna

- najprościej: przechowuje pewną wartość:

```
x = 5
y = "John"
```

Python obsługuje następujące typy danych:

- numeryczne (liczbowe): int, float, complex
- tekstowe: str
- sekwencji: list, tuple
- odwzorowania (mapping type): dict
- zestawów (set types): set, frozenset
- logiczne: bool
- binarne: bytes, bytearray

Są one wbudowane tzn. można z nich korzystać bez konieczności importowania zewnętrznej biblioteki

Pętli w Python: while, for

`for` używamy jeżeli wiemy z góry ilość przebiegów pętli, `while` – jeżeli nie.

```
for i in <collection>:
    <loop body>
```

```
for x in range(6):
    print(x)
```

```
while <expr>:
    <statement(s)>
```

```
s = 0
i = 0
while s <= 15:
    s+=i**2
    i+=1
print(i)
```

range

Generuje nam ciąg liczb (dedykowany typ **range**). Trzeba zamienić na listę "by podejrzeć".
Uwaga : wszystkie argumenty muszą być w typie całkowitym. Jeden argument – to "koniec" – ciąg tworzą liczby naturalne od 0 do $n-1$. Krok domyślny to 1.

Dwa argumenty - to "początek" i "koniec". Krok domyślny to 1. Wtedy wyświetlone są liczby całkowite z przedziału lewostronnie domkniętego .

Trzy argumenty – to "początek", "koniec" oraz krok.

```
print(list(range(5)))
print(list(range(1, 11)))
print(list(range(0, 30, 5)))
print(list(range(0, 10, 3)))
print(list(range(0, -10, -1)))
print(list(range(0)))
print(list(range(1, 0)))
```

Typ **string**

Napisy

- typ sekwencyjny do przechowywania znaków, jest niezmienny
- w języku Python nie ma oddzielnego typu znakowego apostrofy i cudzysłów można stosować zamiennie, ale konsekwentnie

Inne nazwy: string, napisy, łańcuchy znaków

Funkcje: `len()`, `str()`, `<`, `>`, `in`

Pętla przez **string**:

```
for x in "banana":
  print(x)
```

https://www.w3schools.com/python/python_strings_slicing.asp
https://docs.python.org/3/library/string.html
https://www.geeksforgeeks.org/python-string-methods/

Typ string

```
my_str = "Ala ma psa i 2 kota, a Marek ma 3 malpy!"
print(my_str.lower())
print(my_str)
print(my_str.upper())
print(my_str)
print(my_str.replace('a','b'))
print(my_str)

print('234'.isnumeric())
print('234.3'.isnumeric())
print('a'.isnumeric())
print('aBBa'.isalpha())
print('234a'.isalnum())
print('234.3'.isalnum())
print(my_str.isalnum())

print(''.join([x  for x in my_str if x.isnumeric()]))
print(''.join([x  for x in my_str if x.isalpha()]))
print(''.join([x  for x in my_str if x.isalnum()]))
```

Porządek leksykograficzny

```
print("A" < "a")
print("Abc" < "aTw")
print("vccx" < "123")
print("ABC" < "AB")
print("AB" < "ABC")
print("ABC" < "abc")
print('Ab'>'b')
print('Ab'>'a')
```

Sortowanie

```
print(max(["banana", "Orange", "cherry"]))
print(max(["banana", "Orange", "cherry"], key=str.lower))
thislist = ["banana", "Orange", "Kiwi", "cherry"]
thislist.sort()
print(thislist)
thislist = ["banana", "Orange", "Kiwi", "cherry"]
thislist.sort(reverse=True)
print(thislist)
print(max("banana", "Orange", "cherry"))
print(max("banana", "Orange", "cherry", key=str.lower))
```

Krojenie **string**

Uwaga! Pierwszy znak ma indeks 0

[początek:koniec:krok]

krok – domyślenie 1

```
b = "Hello, World!"
print(b[2:5])
print(b[:5])
print(b[:5:1])
print(b[:5:-1])
print(b[2:])
print(b[2::1])
print(b[2::-1])
print(b[-5:-2])
```

https://www.w3schools.com/python/python_strings_slicing.asp
https://www.geeksforgeeks.org/string-slicing-in-python/

Złożone typu danych w Python

- **List (Lista)** jest kolekcją uporządkowaną i zmienną. Zezwala na duplikowanych członków:

```
my_list = [1, 2, True, 'ala', 1, -2.5]
```

- **Tuple (Krotka)** to kolekcja uporządkowana i niezmienna. Zezwala na duplikowanych członków:

```
my_tuple = (1, 2, True, 'ala', 1, -2.5)
```

- **Set (Zbiór)** to kolekcja, która jest nieuporządkowana, niezmienna (ale można dodawać i usuwać elementy) i nieindeksowana. Brak duplikatów członków.:

```
my_set = {1, 2, True, frozenset(('ala', -2.5))}
print(my_set)
```

- **Dictionary (Słownik)** jest kolekcją zmienną. Brak duplikatów członków. Uporządkowany po Python 3.7 i wyżej, nie uporządkowany w Python 3.6 i niżej.:

```
my_dict = {'klucz1' : 1, 1: 2, False: True,
 'jeszczeklucz' : 'ala', 2 : -2.5}
print(my_dict)
print(my_dict[False])
```

Listy: dostęp do elementów i krojenie

Indeks – od zera

- ```
  a = [1, 3, 'abc', False, -2.3, 'XYZ', 9.3]
  print(a[1])
  print(a[-1])
  ```
- `[początek:koniec:krok]`

  ```
  a = [1, 3, 'abc', False, -2.3, 'XYZ', 9.3]
  print(a[6:0:-2])
  print(a[6::-2])
  ```
  ```

Listy comprehensions

- ```
  newlist = [x for x in range(10)]
  ```
- ```
  squares = []
  for x in range(5):
      squares.append(x ** 2)
  print(squares)
  ```
- ```
  squares = [x**2 for x in range(5)]
  print(squares)
  ```
- ```
  [print(x) for x in thislist]
  ```

Listy comprehensions z . . . if . . .

- `newlist = [expression for item in iterable if condition == True]`
- Przykład

  ```
  newlist = [x for x in fruits if x != "apple"]
  ```

- ```
  numbers = [1, -6, 8, 9, 10, 2, 4, 3, 11]
  newlist = [0 if x < 5 else 1 for x in numbers]
  print(newlist)
  ```
  ```

Krotki

```
krotka = 123, 'abc', True
krotka2 = (123, 'abc', True)
print(krotka[2])
#krotka[0] = 1

thistuple = ("apple", "banana", "cherry")
for x in thistuple:
  print(len(x), end='|')

print('\n')
[print(len(x), end='|') for x in thistuple]
print([x.upper() for x in thistuple])
```

Zbiory

```
test_set =  {2, 3, 'abc', -3, 5, 2}

print([2*i for i in test_set])
```

Słownik

```
d = {"one": 1, "two": 2, "three": 3, "four": 4}

print(d)
print(d["one"])
print ([x for x in d])
print ([x for x in d.values()])
print ([x for x in d.keys()])

print ({y:x for x,y in d.items()})

d1 = {"one": 1, "two": 2, "three": 3, "four": 4, "anotherfour":4}
print ({y:x for x,y in d1.items()})
```

Funkcje wspólne dla listy, krotki, zbioru i słownika:

`max()`, `min()`, `len()`

Uwaga! Dla słownika maximum i minimum działają na kluczach:

```
d = {"one": 1, "two": 2, "three": 3, "four": 4}

print(max(d))
```

Funkcje

```
def my_max(a,b):
    if a>b:
        return a
    return b

print(my_max(2,-5))
print(my_max('ab','Ala'))
#print(my_max(2, 'ala'))
def my_max(a,b,c):
    m = a
    if b > a:
        m = b
    if c > m:
        m = c
    return m

print(my_max(2,-5, 4))
print(my_max("banana", "Orange", "cherry"))
#print(my_max(2, 3))
```

Funkcje

```
def my_max(a,b):
    if a>b:
        return a
    return b

print(my_max(2,-5))
print(my_max('ab','Ala'))
#print(my_max(2, 'ala'))
```

Funkcje: *args

```
def my_max(*args):
    m = args[0]
    for a in args:
        if a > m:
            m = a
    return m

print(my_max(2,-5, 4))
print(my_max("banana", "Orange", "cherry"))
print(my_max(2, 3))
print(my_max("ala", "ma","dwa","duzych","czarnych","kota"))
```