Wizualizacja danych
semestr letni 2024

Dr Anna Muranova
UWM w Olsztynie

Wykład 2

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

type hinting – wskazywanie typów

Nie będziemy tego używać.
Przykład:

```
a: int = 5
print(a)
a = "Olsztyn"
print(a)
```

5
Olsztyn

Int – typ całkowity

- bez kropki dziesiętnej
- może być dowolnie długi (ograniczenie ilość pamięci)

```
print(123123123123123123123123123123123123123123 + 1)
```

```
123123123123123123123123123123123123123123124
```

Jaki system liczbowy?

```
print(101)
print(101)
print(0x101) # zero-x
print(0o101) # zero-litera o
print(0b101) # zero-b
print(0X101) # zero-x
print(0O101) # zero-litera o
print(0B101) # zero-b
```

Float – typ zmiennoprzecinkowy

```
print(4.2)
print(4.)
print(.5)
print(.3e4)
print(3e4)
print(2e-2)
print(1.79e308)
print(1.8e308)
print(5e-324)
print(1e-325)
```

Complex – typ zespolony (dot. liczb zespolonych)

```
print(2+3j)
print(2+5*1j)
a = 1j
print(5*a)
```

```
print(4+5*j)
```

```
a = j
```

bool – typ logiczny

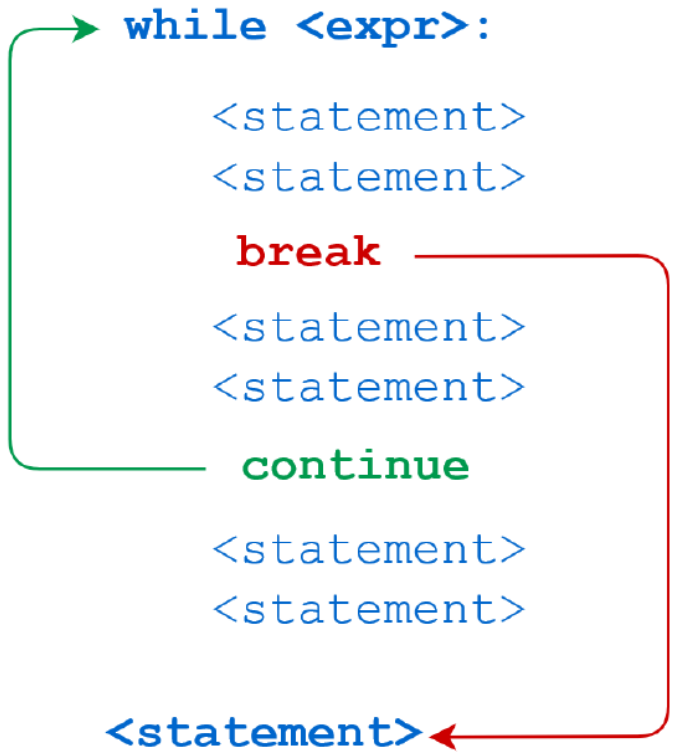
```
print(True)
print(False)
```

Pętli w Python: while

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

```
i = 0
while i < 5:
    print(i)
    i = i + 1
```

break/continue



. . . break . . .

```
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```

. . . . continue . . .

```
i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)
```

. . . while . . . else

```
i = 1
while i < 6:
  print(i)
  i += 1
else:
  print("i is no longer less than 6")
```

. . . while . . . else

Uwaga! `else` nie wykonuje sie, jezeli petla jest zakonczona z powodu `break`.
Porownaj:

```
i = 1
while i < 6:
    print(i)
    i += 1
    if i==3:
        break
else:
  print("i is no longer less than 6")
```

```
i = 1
while i < 6:
    print(i)
    i += 1
    if i==3:
        break
print("i is no longer less than 6")#błąd logiczny!!!
```

. . . while . . . else

Porównaj jeszcze:

```
i = 1
while i < 6:
    print(i)
    i += 1
    if i==3:
        break
else:
  print("i is no longer less than 6")
```

```
i = 1
while i < 6:
    print(i)
    i += 1
    if i==3:
        break
    else:
        print("i is no longer less than 6")#błąd logiczny!!!
```

Pętli w Python: for

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

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

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)))
```

Pętli w Python: for

```
for x in range(2, 30, 3):
  print(x)
```

Liczby parzyste od 0 do n:

```
n = 24
for x in range(0, n+1, 2):
  print(x)
```

lub

```
n = 24
for x in range(0, n+1):
    if n % 2 == 0:
        print(x)
```

. . . for . . . else

```
for x in range(6):
  print(x)
else:
  print("Finally finished!")
```

lub

```
for x in range(6):
  if x == 3: break
  print(x)
else:
  print("Finally finished!")
```

. . . for . . . else

Porównaj:

```
for x in range(6):
  if x == 3: break
  print(x)
else:
  print("Finally finished!")
```

```
for x in range(6):
  if x == 3: break
  print(x)
print("Finally finished!")
```

. . . for . . . else

Porównaj:

```
for x in range(6):
  if x == 3: break
  print(x)
else:
  print("Finally finished!")
```

```
for x in range(6):
  if x == 3: break
  print(x)
  else:
      print("Finally finished!")
```

Zagnieżdżone pętle

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

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

Zagnieżdżone pętle

```
for i in range(3):
    for j in range(3):
        print(i, "*", j, "=", i * j)
```

. . . pass . . .

Pętla for nie może być pusta, używaj pass.

```
for x in [0, 1, 2]:
  pass
```

Uwaga: najczęściej pass jest używany, jeżeli kod będzie dodany później.

Zagnieżdżone pętle: ciekawostki

```
for i in range(3):
    print(i)
    for i in range(3):
        pass
```

```
for i in range(3):
    for i in range(3):
        pass
    print(i)
```

Zagnieżdżone pętle: ciekawostki

```
a = 0
b = 0
for i in range(3):
    for i in range(3):
        a += 1
    b += 1
print('a=',a)
print('b=',b)
```

Typ **string**

Napisy

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

Inne nazwy: string, napisy, łańcuchy znaków

- Abstrakcyjnie: na końcu każdego napisu jest znak “zerowy” - będzie widać lepiej w C/C++

Tablica znaków ASCII
https://upload.wikimedia.org/wikipedia/commons/5/5c/ASCIITable- wide.pdf
26

Typ string

https://www.w3schools.com/python/python_strings.asp

```
a = "Olsztyn"
print(a)
print(a[3])
#a[2] = 'w'
```

```
a = "Olsztyn"
b = "Gdańsk"
print(a + b)
print(a * 2)
print(2 * a)
```

Specjalne funkcje

- `chr()` – zamienia liczbę całkowitą na znak
- `ord()` – zamienia znak na liczbę całkowitą odpowiadającą pozycji w tabeli znaków
- `len()` – długość napisu
- `str()` – rzutuje argument na napis

Krojenie **string**

https://www.w3schools.com/python/python_strings_slicing.asp
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])
```

Porządek leksykograficzny

```
print("A" < "a")
print("Abc" < "aTw")
print("vccx" < "123")
print("ABC" < "AB")
print("AB" < "ABC")
print("ABC" < "abc")
```

Pętla przez **string**

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

Sprawdzanie **string**

```
txt = "The best things in life are free!"
print("free" in txt)
```

```
txt = "The best things in life are free!"
if "free" in txt:
  print("Yes, 'free' is present.")
```

```
txt = "The best things in life are free!"
print("expensive" not in txt)
```

```
txt = "The best things in life are free!"
if "expensive" not in txt:
  print("No, 'expensive' is NOT present.")
```

Formatowanie napisów

- trzy różne konwencje
- niektóre rzeczy nie działają w każdej wersji 3.x
- warto zastanowić się czy warto używać tych konstrukcji? czasem może lepiej skorzystać z funkcji print?

styl printf

Zaczerpnięty z języka C – stare.
https://docs.python.org/3.10/library/stdtypes.html#old-stringformatting

```
a = "abc"
str = "a to %s" % a
print(str)
b = 4
c = 5
str2 = "%d + %d = %d" % (b, c, b + c)
print(str2)
```

styl format

https://docs.python.org/3.10/library/string.html#formatstrings

```
a = "abc"
str = "a to {}".format(a)
print(str)
b = 4.2
c = 5
str2 = "{0} + {1} = {2}".format(b, c, b + c)
print(str2)
```

styl f-Strings

https://docs.python.org/3.10/reference/lexical_analysis.html#f-strings

```
a = "abc"
str = f"a to {a}"
print(str)
b = 4.2
c = 5
str2 = f"{b} + {c} = {b+c}"
print(str2)
```

```
b = 4.2
c = 5
str2 = f"{b:f} + {c:d} = {b+c:e}"
print(str2)
```

Dodatkowe

- podział stałych
  `https://docs.python.org/3.10/library/string.html?highlight=string#module-string`
- funkcje wbudowane dot. napisów
  `https://docs.python.org/3.10/library/stdtypes.html#stringmethods`