Wizualizacja danych
semestr letni 2024

Dr Anna Muranova
UWM w Olsztynie

Wykład 3

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`

print()

`print(object(s), sep=separator, end=end, file=file, flush=flush)`

- `sep='separator'` Opcjonalnie. Sposób rozdzielenia obiektów, jeśli jest ich więcej niż jeden. Wartość domyślna to " ".
- `koniec='koniec'` Opcjonalnie. Końcówka. Wartość domyślna to "\n" (przesunięcie wiersza)
- `file` Opcjonalnie. Obiekt z metodą zapisu. Wartość domyślna to `sys.stdout`
- `flush` Opcjonalnie. Wartość logiczna określająca, czy dane wyjściowe są opróżniane (True), czy buforowane (False). Wartość domyślna to False.

```
print("Hello World")
print("Hello World", end="|")
print("Hello World", end="\n")
print("Hello", "how are you?")
print("Hello", "how are you?", sep="---")
print("Hello", "how are you?", sep="\n")
print("Hello", "how are you?", sep=" | ")
```

Złożone typu danych w Python

- **List (Lista)** jest kolekcją uporządkowaną i zmienną. Zezwala na duplikowanych członków.
- **Tuple (Krotka)** to kolekcja uporządkowana i niezmienna. Zezwala na duplikowanych członków.
- **Set (Zbiór)** to kolekcja, która jest nieuporządkowana, niezmienna (ale można dodawać i usuwać elementy) i nieindeksowana. Brak duplikatów członków.
- **Dictionary (Słownik)** jest kolekcją zmienną. Brak duplikatów członków. Uporządkowany po Python 3.7 i wyżej, nie uporządkowany w Python 3.6 i niżej.

Listy

Lista w Pythonie to tzw. typ sekwencyjny umożliwia przechowywanie elementów różnych typów.

Cechy:

- zmienny (**mutable**) - umożliwia przypisanie wartości pojedynczym elementom do zapisu używamy nawiasów kwadratowych
- poszczególne elementy rozdzielamy przecinkami
- każdy element listy ma przyporządkowany indeks
- elementy listy są numerowane od zera
- listy są uporządkowane
- listy są dynamiczne (mogą mieć różną długość)
- listy mogą być zagnieżdżone

Uwaga! Listy w języku Python są specyficzną strukturą danych nie zawsze dostępną w innych językach programowania. Pojęcie listy w całej informatyce"szersze". Wyróżnia się np. listy jednokierunkowe, które nie muszą mieć indeksu. Nie będziemy takich przypadków analizować.

Listy: pisownia

```
nazwa = [element1, element2, ..., elementN]
```

- Pusta lista:

  ```
  a = []
  b = list()
  ```

- Lista z liczbami:

  ```
  a = [2, 3, 4.5, 5, -3.2]
  ```

- Lista mieszana:

  ```
  b = ['abcd', 25+3j, True, 1]
  ```

Listy: właściwości

- Kolejność ma znaczenie:

```
a = [1, 2, 3, 4]
b = [4, 3, 2, 1]
print(a == b)
```

- Elementy na liście nie muszą być unikalne

```
a = [1, 2, 3, 4, 2]
b = [1, 2, 3, 4]
print(a)
print(a == b)
```

- Elementy mogą być różnego typu

```
a = [1, 2, 3, '2']
print(a)
```

Listy: dostęp do elementów

- Indeks – od zera

```
a = [1, 3, 'abc', False, -2.3, 'XYZ', 9.3]
print(a[1])
print(a[4])
print(a[0])
#print(a[7])
```

- Ujemny indeks

```
a = [1, 3, 'abc', False, -2.3, 'XYZ', 9.3]
print(a[-1])
print(a[-5])
print(a[-7])
```

Listy: krojenie

- ```
  a = [1, 3, 'abc', False, -2.3, 'XYZ', 9.3]
  print(a[1:4])
  print(a[-5:-2])
  print(a[:4])
  ```
- ```
  a = [1, 3, 'abc', False, -2.3, 'XYZ', 9.3]
  print(a[2:])
  print(a[0:6:2])
  print(a[1:6:2])
  ```
- ```
  a = [1, 3, 'abc', False, -2.3, 'XYZ', 9.3]
  print(a[6:0:-2])
  print(a[::-1])
  print(a[:])
  ```
- ```
  a = [1, 3, 'abc', False, -2.3, 'XYZ', 9.3]
  print(a[::2])
  print(a[::-2])
  ```

Listy: specjalne funkcji

- Długość

```
a = [1, 3, 'abc', False, -2.3, 'XYZ', 9.3]
print(len(a))
```

Implementacja samodzielna długości:

```
def dlugosc(lista):
    x = 0
    for i in lista:
        x += 1
    return x
a = [1, 3, 'abc', False, -2.3, 'XYZ', 9.3]
print(dlugosc(a))
```

Listy: specjalne funkcji

- Maksimum i minimum?

  Działa wtedy gdy mamy porządek:
  - liczby <=
  - napisy – porządek leksykograficzny

```
a = [4,-5,3.4,-11.2]
print(min(a))
print(max(a))
b = ['abc', 'ABCd', 'krt', 'abcd']
print(min(b))
print(max(b))
```

Listy: modyfikacja

- ```
  a = [4,-5,3.4,-11.2]
  a[2] = 'a'
  print(a)
  ```
- ```
  a = [4,-5,3.4,-11.2]
  a[2] = ['a','b']
  print(a)
  ```
- ```
  a = [4,-5,3.4,-11.2]
  a[1:2] = ['a','b']
  print(a)
  ```
- ```
  a = [4,-5,3.4,-11.2]
  a[1:3] = ['a','b']
  print(a)
  ```
- ```
  a = [1, 3, 'abc', False, -2.3, 'XYZ', 9.3]
  del a[2]
  print(a)
  del a[::-2]
  print(a)
  del a[:]
  print(a)
  ```
  ```

Listy: dodawanie i mnożenie przez liczbę całkowitą

- ```
  a = [4,-5,3.4,-11.2]
  b = ['a','b','c']
  print(a+b)
  ```
- ```
  a = [4,-5,3.4,-11.2]
  print(a*2)
  print(2*a)
  ```

Listy: inne funkcje

- `list.append(x)` – dodaje element na końcu listy.
  Równoważnie `a[len(a):] = [x]`

```
a = [1, 3, 'abc', False]
a.append(5.3)
print(a)
```

- `list.extend(iterable)` – dodaje elementy z argumenty na koniec listy.
  Równoważnie: `a[len(a):] = iterable`

```
a = [1, 3, 'abc', False]
b = [3, -2]
a.extend(b)
print(a)
```

- Różnice?

```
a = [1, 3, 'abc', False]
b = [3, -2]
a.append(b)
print(a)
```

Listy: inne funkcje

- `list.insert(i, x)` – wstawia element `x` na pozycji `i`

```
a = [1, 3, 'abc', False]
a.insert(0,'w')
print(a)
a.insert(4,9.0)
print(a)
```

- `list.remove(x)` – usuwa element z listy (pierwszy od początku)

```
a = [1, 3, 'abc', False]
a.remove(False)
print(a)
b = [3, 4, 5, 3]
b.remove(3)
print(b)
```

Listy: inne funkcje

- `list.pop()` – usuwa i zwraca ostatni element
  `list.pop(i)` – usuwa i zwraca element na pozycji `i`

```
a = [1, 3, 'abc', False]
print(a.pop())
print(a)
b = [3, -4, 6.2, 7]
print(b.pop(3))
print(b)
```

- `list.clear()` – usuwa wszystkie elementy z listy.
  Równoważnie: `del a[:]`

```
a = [1, 3, 'abc', False]
a.clear()
print(a)
```

Listy: inne funkcje

- `list.index(x)` – zwraca indeks elementu `x` (o ile istnieje, inaczej błąd), w przypadku duplikatów pierwszy z lewej
- `list.index(x,start)` – zwraca indeks elementu `x` (o ile istnieje, inaczej błąd) zaczynając od pozycji start, w przypadku duplikatów pierwszy z lewej
- `list.index(x,start,end)` – zwraca indeks elementu `x` (o ile istnieje, inaczej błąd) zaczynając od pozycji `start` a kończąc na `end-1`, w przypadku duplikatów pierwszy z lewej

Listy: inne funkcje

Przykłady

- ```
  a = [1, 3, 1, 4, 5, 2, 3]
  print(a.index(3))
  print(a.index(3, 5))
  print(a.index(3, 1, 4))
  ```
- ```
  a = ['abc','xyz','abc','efg']
  print(a.index('abc'))
  print(a.index('abc', 2))
  print(a.index('abc', 1, 4))
  ```

Listy: inne funkcje

- `list.count(x)` – zwraca ile razy występuję element x na liście

```
a = ['abc','xyz','abc','efg']
print(a.count('abc'))
print(a.count(4))
```

- `list.sort()` – sortuje listę (o ile elementy można posortować)

```
a = ['abc','xyz','abc','efg']
a.sort()
print(a)
```

- `list.reverse()` – odwraca kolejność elementów na liście (nie ma nic związku z sortowaniem!)

```
a = [4, 5, -2, 7.3, 9, -22, 23]
a.reverse()
print(a)
```

Listy: sortowanie według funkcji

- ```
  def myfunc(n):
    return abs(n - 50)

  thislist = [100, 50, 65, 82, 23]
  thislist.sort(key = myfunc)
  print(thislist)
  ```
- ```
  thislist = ["banana", "Orange", "Kiwi", "cherry"]
  thislist.sort()
  print(thislist)
  ```
- ```
  thislist = ["banana", "Orange", "Kiwi", "cherry"]
  thislist.sort(key = str.lower)
  print(thislist)
  ```
  ```

Listy: sortowanie w odwrotnym porządku

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

Listy: kopie i przypisanie

- Spójrzmy na przykład jak działa operator przypisania dla list.

```
a = [4, 5, -2, 7.3, 9, -22, 23]
b = a
b[2] = 100
print(b)
print(a)
```

- `list.copy()` – tworzy kopię listy

```
a = [4, 5, -2, 7.3, 9, -22, 23]
b = a.copy()
b[2] = 100
print(b)
print(a)
```

Listy: petli

- ```
  thislist = ["banana", "Orange", "Kiwi", "cherry"]
  for x in thislist:
    print(x)
  ```
- ```
  thislist = ["banana", "Orange", "Kiwi", "cherry"]
  for x in thislist:
    print(x[0])
  ```

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"]
```

- Porównaj:

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = []

for x in fruits:
  if x != "apple":
    newlist.append(x)

print(newlist)
```

Listy comprehensions z . . . if . . . : przykłady

- ```
  newlist = [x for x in range(10) if x < 5]
  print(newlist)
  ```
- ```
  fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
  newlist = [x for x in fruits if "a" in x]
  print(newlist)
  ```
- ```
  fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
  newlist = [x.upper() for x in fruits]
  print(newlist)
  ```
- ```
  fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
  newlist = ['hello' for x in fruits]
  print(newlist)
  ```
- ```
  numbers = [1, -6, 8, 9, 10, 2, 4, 3, 11]
  newlist = [0 if x < 5 else 1 for x in numbers]
  print(newlist)
  ```
  ```

Krotki (tuple)

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


print(tuple(range(5)))


a, b = 2,5
print(a)
```

Krotki (tuple): właściwości

- Pozwala na duplikaty

```
thistuple = ("apple", "banana", "cherry", "apple", "cherry")
print(thistuple)
```

- Pozwala na różne typu elementów

```
tuple1 = ("abc", 34, True, 40, "male")
print(tuple1)
```

- Długość

```
thistuple = ("apple", "banana", "cherry")
print(len(thistuple))
```

- Kilka elementów

```
thistuple = ("apple",)
print(type(thistuple))

#NOT a tuple
thistuple = ("apple")
print(type(thistuple))
```

Krotki: pętli

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

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

Krotki i listy

```
thistuple = ("apple", "banana", "cherry")
print(thistuple)
print(type(thistuple))

thislist = list(thistuple)
print(thislist)
print(type(thislist))

thattuple = tuple(thislist)
print(thattuple)
print(type(thattuple))


print([x.upper() for x in thistuple])
```