Wstęp do programowania
semestr zimowy 2024/2025

Dr Anna Muranova
UWM w Olsztynie

Wykład 5

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

`https://www.ascii-code.com/`

- `chr()` – zamienia liczbę całkowitą na znak
  `print(chr(97))#a`
- `ord()` – zamienia znak na liczbę całkowitą odpowiadającą pozycji w tabeli znaków
  `print(ord('a'))#97`
- `len()` – długość napisu
  `print(len('ala ma kota'))#11`
- `str()` – rzutuje argument na napis
  `print(str(9.0)*2)#9.09.0`

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

f-ciągi pojawiły się w wersji języka Python 3.6 i obecnie są zalecanym sposobem formatowania tekstu. Aby utworzyć f-ciąg, należy:

- Wpisać literę *f* lub *F*, a zaraz po niej otwierający cudzysłów lub apostrof.
- Wewnątrz łańcucha umieścić zmienne lub wyrażenia ujęte w nawiasy klamrowe ({}). Ich wartości zostaną automatycznie zamienione na tekst.

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)
```

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=" | ")
```

Wybrane metody klasy string 1

- `capitalize()`
  Zwraca kopię napisu z pierwszym znakiem zmienionym na wielką literę.

```
txt = "ala ma kota"
x = txt.capitalize()
print(x)#Ala ma kota
print(txt)#ala ma kota
```

- `count(napis, początek, koniec)`
  Zwraca ilość nienachodzących na siebie wystąpień napisu `napis` w zakresie `[początek:koniec]`. Opcjonalne argumenty `początek` i `koniec` są interpretowane tak samo, jak w operacji wycinania.

```
txt = "I love apples, apple are my favorite fruit"
x = txt.count("apple", 1, 24)
print(x)#2
```

Wybrane metody klasy string 2

- `endswith(przyrostek, początek, koniec)`
  Zwraca wynik sprawdzenia, czy napis jest zakończony napisem `przyrostek`. Przy wystąpieniu argumentu `początek`, sprawdzenie rozpoczyna się od tego znaku. Przy wystąpieniu argumentu `koniec`, porównanie zakończy się na tym znaku.

```
txt = "I love apples, apple are my favorite fruit"
print(txt.endswith('fruit'))#True
```

- `expandtabs([wielkość])`
  Zwraca kopię napisu ze wszystkimi znakami tabulacji zastąpionymi przez znaki spacji. Jeśli `wielkość` nie zostanie podana, przyjmuje się rozmiar tabulacji jako 8 znaków.

```
txt = "H\te\tl\tl\to"
x =  txt.expandtabs(3)
print(x)#H  e  l  l  o
print(txt.expandtabs())#H       e       l       l       o
print(txt)#H e l l o
```

Wybrane metody klasy string 3

- `find(podnapis, początek, koniec)`
  Zwraca najniższy indeks takiego wystąpienia napisu `podnapis`, aby napis był zawarty w wycinku `[początek:koniec]`. Opcjonalne argumenty `początek` i `koniec` są interpretowane tak samo, jak w operacji wycinania. Zwraca -1, jeśli napis `podnapis` nie został znaleziony.

```
txt = "Hello, welcome to my world."
x = txt.find("welcome")
print(x)#7
```

  Funkcja `find` powinna być używana tylko wtedy, gdy chcemy poznać pozycję napisu `podnapis` w danym napisie. Jeżeli chcemy tylko sprawdzić, czy napis `podnapis` występuje w danym napisie, to należy użyć operatora `in`: `podnapis in napis`

```
txt = "Hello, welcome to my world."
x = "welcome" in txt
print(x)#True
```

Wybrane metody klasy string 4

- `isalnum()`
  Zwraca wynik sprawdzenia, czy wszystkie znaki napisu są znakami alfanumerycznymi i napis składa się przynajmniej z jednego znaku.
- `isalpha()`
  Zwraca wynik sprawdzenia, czy wszystkie znaki napisu są literami i napis składa się przynajmniej z jednego znaku.
- `isdigit()`
  Zwraca wynik sprawdzenia, czy wszystkie znaki napisu są cyframi.
  `islower()`
  Zwraca wynik sprawdzenia, czy wszystkie litery napisu są małymi literami i napis zawiera przynajmniej jedną małą literę.

```
txt = 'napis100'
print(txt.isalnum())#True
print(txt.isalpha())#False
print(txt.isdigit())#False
print(txt.islower())#True
```

Wybrane metody klasy string 5

- `isspace()`
  Zwraca wynik sprawdzenia, czy wszystkie znaki napisu są białymi znakami i napis składa się przynajmniej z jednego znaku.
- `istitle()`
  Zwraca wynik sprawdzenia, czy napis ma strukturę tytułu, to znaczy każdy wyraz napisu musi zaczynać się wielką literą i składać wyłącznie z małych liter lub znaków nieliterowych.
- `isupper()`
  Zwraca wynik sprawdzenia, czy wszystkie litery napisu są wielkimi literami i napis zawiera przynajmniej jedną wielką literę.

```
txt = 'This Is A Title Number 1'
print(txt.isspace())#False
print(txt.istitle())#True
print(txt.isupper())#False
```

Wybrane metody klasy string 6

- `ljust(szerokość)`
  Zwraca kopię napisu wyrównaną do lewej w napisie o szerokości `szerokość`. Wypełnienie jest uzyskane za pomocą znaków spacji. Jeśli `szerokość` jest mniejsza od `len(s)`, zwracany jest oryginalny napis.
- `lstrip(chars)`
  Zwraca kopię napisu z usuniętymi znakami z początku napisu. W przypadku, gdy argument `chars` nie został podany, lub ma wartość `None`, usunięte zostaną białe znaki. Jeżeli argument ten jest podany i nie ma wartości `None`, musi być typu napisowego. Z początku napisu, na rzecz którego wywołana została ta metoda, zostaną usunięte znaki wchodzące w skład argumentu `chars`.

```
txt = 'Ala ma 3 kota'
print(txt.ljust(15),'i psa')#Ala ma 3 kota   i psa
print(txt.lstrip('Al'))#a ma 3 kota
print(txt)#Ala ma 3 kota
```

Wybrane metody klasy string 7

- `rjust(szerokość)`
  Zwraca kopię napisu wyrównaną do prawej w napisie o szerokości `szerokość`. Wypełnienie jest uzyskane za pomocą znaków spacji. Jeśli `szerokość` jest mniejsza od `len(s)`, zwracany jest oryginalny napis.
- `rstrip(chars)`
  Zwraca kopię napisu z usuniętymi znakami z końca napisu. W przypadku, gdy argument `chars` nie został podany, lub ma wartość `None`, usunięte zostaną białe znaki. Jeżeli argument ten jest podany i nie ma wartości `None`, musi być typu napisowego. Z końca napisu, na rzecz którego wywołana została ta metoda, zostaną usunięte znaki wchodzące w skład argumentu `chars`.

```
txt = 'Ala ma 3 kota'
print('Napis',txt.rjust(15))#Napis    Ala ma 3 kota
print(txt.rstrip('a'))#Ala ma 3 kot
print(txt)#Ala ma 3 kota
```

Wybrane metody klasy string 8

- `lower()`
  Zwraca kopię napisu zamienionego na małe litery.
- `upper()`
  Zwraca kopię napisu zamienionego na duży litery.

```
txt = 'Ala ma 3 kota'
print(txt.lower())#ala ma 3 kota
print(txt.upper())#ALA MA 3 KOTA
print(txt))#Ala ma 3 kota
```

Wybrane metody klasy string 9

- `replace(stary, nowy, ile)`
  Zwraca kopię napisu z wszystkimi wystąpieniami napisu `stary` zastąpionymi przez `nowy`. Jeśli zostanie podany argument `ile`, zostanie zastąpiona tylko podana ilość wystąpień.
- `rfind(napis , początek, koniec)`
  Zwraca najwyższy indeks wystąpienia napisu `napis`, takiego, aby napis był zawarty w przedziale `[początek, koniec)`. Opcjonalne argumenty `początek` i `koniec` są interpretowane tak samo, jak w operacji wycinania. Zwraca -1, jeśli napis nie został znaleziony.

```
txt = 'Ala ma 3 kota i jeszcze kota'
print(txt)#Ala ma 3 kota i jeszcze kota
print(txt.replace('kota', 'psa'))#Ala ma 3 psa i jeszcze psa
print(txt)#Ala ma 3 kota i jeszcze kota
print(txt.rfind('psa'))#-1
print(txt.rfind('kota'))#24
print(txt.find('kota'))#9
```

Wybrane metody klasy string 10

- `startswith(prefix, start, end)`
  Zwraca wynik sprawdzenia, czy napis zaczyna się napisem `prefix`. Przy wystąpieniu argumentu `start`, sprawdzenie rozpoczyna się od tego znaku. Przy wystąpieniu argumentu `end`, porównanie zakończy się na tym znaku.
- `strip(chars)`
  Zwraca kopię napisu z usuniętymi znakami z początku i końca napisu. W przypadku, gdy argument `chars` nie został podany, lub ma wartość `None`, usunięte zostaną białe znaki. Jeśli argument ten jest podany i nie ma wartości `None`, musi być typu napisowego. Z początku i końca napisu, na rzecz którego wywołana została ta metoda, zostaną usunięte znaki wchodzące w skład argumentu `chars`.

```
txt = 'Ala ma 3 kota i jeszcze kota'
print(txt.startswith('Ala'))#True
print(txt.startswith('ma',3))#False
print(txt.startswith('ma',4))#True
print(txt.strip('Ala kota'))#ma 3 kota i jeszcze
print(txt)#Ala ma 3 kota i jeszcze kota
```

Wybrane metody klasy string 11

- `swapcase()`
  Zwraca kopię napisu z małymi literami zamienionymi na wielkie, a wielkimi na małe.
- `title()`
  Zwraca kopię napisu zamienioną na strukturę tytułu, to znaczy każdy wyraz napisu zostaje zamieniony na rozpoczynający się wielką literą, z pozostałymi literami zamienionymi na małe.
- `center(szerokość)`
  Zwraca kopię napisu wyrównaną po centrum w napisie o szerokości `szerokość`. Wypełnienie jest uzyskane za pomocą znaków spacji. Jeśli `szerokość` jest mniejsza od `len(s)`, zwracany jest oryginalny napis.

```
txt = 'Ala ma 3 kota i 2 PSA'
print(txt.swapcase())#aLA MA 3 KOTA I 2 psa
print(txt.title())#Ala Ma 3 Kota I 2 Psa
print('Napis',txt.center(30),'!')
#Napis     Ala ma 3 kota i 2 PSA      !
print(txt)#Ala ma 3 kota i 2 PSA
```

Wybrane metody klasy string 12

- `zfill(szerokość)`
  Zwraca napis uzupełniony z lewej strony zerami do podanej szerokości. W przypadku, gdy wartość argumentu jest mniejsza od długości napisu, zostanie zwrócony oryginalny napis.

```
txt = 'Ala'
num = '1371'
print('Napis:',txt.rjust(10))#Napis:        Ala
print('Napis:',txt.zfill(10))#Napis: 0000000Ala
print('Napis:',num.rjust(10))#Napis:       1371
print('Napis:',num.zfill(10))#Napis: 0000001371
```

Inne:
funkcje wbudowane dot. napisów
`https://docs.python.org/3.10/library/stdtypes.html#string-methods`

Podział stałych 1

Stałe zdefiniowane w tym module to:

- `string.ascii_letters`
  Konkatenacja stałych `ascii_lowercase` i `ascii_uppercase` opisanych poniżej. Ta wartość nie zależy od ustawień regionalnych.
- `string.ascii_lowercase`
  Małe litery alfabetu łacińskiego: `'abcdefghijklmnopqrstuvwxyz'`. Ta wartość nie zależy od ustawień regionalnych i nie ulega zmianie.
- `string.ascii_uppercase`
  Wielkie litery alfabetu łacińskiego: `'ABCDEFGHIJKLMNOPQRSTUVWXYZ'`. Ta wartość nie zależy od ustawień regionalnych i nie ulega zmianie.
- `string.digits`
  Ciąg znaków: `'0123456789'`.
- `string.hexdigits`
  Ciąg znaków: `'0123456789abcdefABCDEF'`.
- `string.octdigits`
  Ciąg znaków: `'01234567'`.

Podział stałych 2

- `string.punctuation`
  Ciąg znaków ASCII, które są uważane za znaki interpunkcyjne w lokalizacji C: `"!#$%&'()*+,-./:;<=>?@[\]^_‘| "`.
- `string.printable`
  Ciąg znaków ASCII, które są uważane za znaki drukowalne. Jest to połączenie `digits`, `ascii_letters`, `punctuation`, i `whitespace`.
- `string.whitespace`
  Ciąg zawierający wszystkie znaki ASCII uznawane za odstępy. Zawiera spację, tabulator, znak nowej linii, powrotu karetki, znak nowej strony i tabulator pionowy.

```
import string

print(string.ascii_letters)
```

Link: `https://docs.python.org/3.10/library/string.html?highlight=string#module-string`

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. WAŻNE

- 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)#[4, 5, 100, 7.3, 9, -22, 23]
```

- `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: pętli

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

List 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]`

List 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 – funkcje

`count()` – Zwraca liczbę wystąpień określonej wartości w krotce

`index()` – Przeszukuje krotkę pod kątem określonej wartości i zwraca pozycję, w której została znaleziona.

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])
```

Funkcja zwraca krotki

```
def mm(list):
min = max = list[0]
for i in list:
    if i < min: min = i
    if i > max: max = i
return min, max

x = mm([1,2,3])
print(type(x))#<class 'tuple'>
print(x[1])
print(x[0])
```