Programowanie Python 1

(CP1S02005)

Politechnika Białostocka - Wydział Elektryczny

Cyfryzacja przemysłu, sem. II, studia stacjonarne I stopnia

Rok akademicki 2023/2024

Wykład nr 3 (20.03.2024)

dr inż. Jarosław Forenc

Plan wykładu nr 3

- Pętla for, funkcja range()
- Instrukcje break i continue
- Listy i pętla for
- Pętla while

Python - pętla for

- Pętla for jest używana do iteracji przez elementy kolekcji lub innego iterowanego obiektu
- Składnia pętli for:

```
for element in kolekcja:
    # kod wykonywany dla każdego elementu
```

  - element - zmienna przyjmująca wartości elementów z kolekcji podczas każdej iteracji
  - kolekcja - lista, krotka, słownik, zbiór lub inna kolekcja iterowalna
- Podczas każdej iteracji element przyjmuje wartości kolejnego elementu z kolekcji i dla tej wartości wykonywane są instrukcje

Python - funkcja range()

- Funkcja range() służy do generowania serii liczb całkowitych, używana jest powszechnie w iteracjach, zwłaszcza w pętli for

```
range(start, stop, step)
```

  - start - liczba całkowita, od której zaczyna się seria, domyślnie ma wartość 0 (parametr opcjonalny)
  - stop - liczba całkowita, na której kończy się seria (nie jest wliczana w serię)
  - step - krok, o który seria się zwiększa, domyślnie ma wartość 1 (parametr opcjonalny)

- Zwraca obiekt typu „range”, który reprezentuje serię liczb całkowitych
- Nie generuje listy od razu, ale tworzy obiekt, który generuje liczby na żądanie

Python - pętla for i funkcja range()

- generowanie liczb całkowitych z zakresu [0, 4]

```
for i in range(5):
    print(i)
```

```
0
1
2
3
4
```

```
for i in range(5):
    print(i, end = " ")
print()
```

```
0 1 2 3 4
```

Python - pętla for i funkcja range()

- generowanie liczb całkowitych z zakresu [2, 4]

```
for i in range(2,5):
    print(i, end = " ")
```

```
2 3 4
```

- generowanie liczb całkowitych z zakresu [1, 10] z krokiem 2

```
for i in range(1,11,2):
    print(i, end = " ")
```

```
1 3 5 7 9
```

Python - pętla for i funkcja range()

- błędne wartości start i stop spowodują wygenerowanie pustego ciągu

```
for i in range(9,0):
    print(i, end = " ")
```

```

```

- wartość step może być ujemna

```
for i in range(9,0,-1):
    print(i, end = " ")
```

```
9 8 7 6 5 4 3 2 1
```

Python - pętla for i funkcja range()

- wygenerowanie n+1 liczb rzeczywistych z przedziału [start, stop]

```
start = 0
stop = 1
n = 10
step = (stop - start) / n
for i in range(n + 1):
    value = start + i * step
    print(value)
```

```
0.0
0.1
0.2
0.30000000000000004
0.4
0.5
0.6000000000000001
0.7000000000000001
0.8
0.9
1.0
```

- w pętli for może być wykonanych kilka instrukcji (wymagają wcięcia o takiej samej wielkości)

Python - pętla for i instrukcja break

- instrukcja break jest używana wewnątrz pętli for, aby przerwać jej działanie, niezależnie od tego, ile iteracji zostało wykonanych
- Przykład: poszukiwanie pierwszej liczby podzielnej zarówno przez 3, jak i przez 7 w przedziale [1, 100]

```
for liczba in range(1, 101):
    if liczba % 3 == 0 and liczba % 7 == 0:
        print("Znaleziona liczba to:", liczba)
        break
```

```
Znaleziona liczba to: 21
```

Python - pętla for i instrukcja break

- instrukcja break jest używana wewnątrz pętli for, aby przerwać jej działanie, niezależnie od tego, ile iteracji zostało wykonanych
- Przykład: poszukiwanie pierwszej liczby podzielnej zarówno przez 3, jak i przez 7 w przedziale [1, 100] (wynik bez break)

```
for liczba in range(1, 101):
    if liczba % 3 == 0 and liczba % 7 == 0:
        print("Znaleziona liczba to:", liczba)
```

```
Znaleziona liczba to: 21
Znaleziona liczba to: 42
Znaleziona liczba to: 63
Znaleziona liczba to: 84
```

Python - pętla for i instrukcja continue

- instrukcja continue służy do przerwania aktualnej iteracji pętli for i przejścia do następnej iteracji
- Przykład: wyświetlenie tylko liczb nieparzystych z przedziału [1, 10]

```
for liczba in range(1, 11):
    if liczba % 2 == 0:
        continue
    print(liczba, end = " ")
```

```
1 3 5 7 9
```

Python - pętla for, najczęstsze błędy

- brak dwukropka (:)

```
for i in range(5)
    print(i)
```

```
PS C:\Users\jaros> python -u "d:\MyApp.py"
  File "d:\MyApp.py", line 1
    for i in range(5)
                     ^
SyntaxError: expected ':'
```

Python - pętla for, najczęstsze błędy

- brak wcięcia w instrukcji po pętli for

```
for i in range(5):
print(i)
```

```
PS C:\Users\jaros> python -u "d:\MyApp.py"
  File "d:\MyApp.py", line 2
    print(i)
    ^
IndentationError: expected an indented block
                after 'for' statement on line 1
```

Python - pętla for, najczęstsze błędy

- niepotrzebne wcięcie

```
for i in range(5):
    print(i)
    print("Koniec")
```

```
0
Koniec
1
Koniec
2
Koniec
3
Koniec
4
Koniec
```

Python - pętla for, najczęstsze błędy

- próba iterowania obiektu, który nie jest iterowalny

```
value = 10
for i in value:
    print(i)
```

```
PS C:\Users\jaros> python -u "d:\MyApp.py"
Traceback (most recent call last):
  File "d:\MyApp.py", line 2, in <module>
    for i in value:
TypeError: 'int' object is not iterable
```

Python - zagnieżdżanie pętli for

- Przykład: tabliczka mnożenie

```
for i in range(1, 11):
    for j in range(1, 11):
        wynik = i * j
        print(f"{wynik:4}", end=" ")
    print()
```

```
   1    2    3    4    5    6    7    8    9   10
   2    4    6    8   10   12   14   16   18   20
   3    6    9   12   15   18   21   24   27   30
   4    8   12   16   20   24   28   32   36   40
   5   10   15   20   25   30   35   40   45   50
   6   12   18   24   30   36   42   48   54   60
   7   14   21   28   35   42   49   56   63   70
   8   16   24   32   40   48   56   64   72   80
   9   18   27   36   45   54   63   72   81   90
  10   20   30   40   50   60   70   80   90  100
```

Python - listy

- Lista - struktura danych przechowująca kolekcję elementów ułożonych w określonej kolejności
  - tworząc listę jej elementy umieszczamy wewnątrz nawiasów kwadratowych, rozdzielając je przecinkami

```
bierne = ["rezystor", "cewka", "kondensator"]
moja_lista = [1, 2, 3, "a", "b", "c", 2.5]
czynne = []
```

  - listy są dynamiczne - w trakcie działania programu można dodawać do nich elementy i usuwać je
  - listy mogą zawierać elementy różnych typów danych (liczby, łańcuchy znaków, inne listy, krotki, słowniki)
  - elementy listy nie muszą być ze sobą powiązane

Python - listy

- do wyświetlenia listy można zastosować funkcję print()

```
bierne = ["rezystor", "cewka", "kondensator"]
print(bierne)
```

```
['rezystor', 'cewka', 'kondensator']
```

```
moja_lista = [1, 2, 3, "a", "b", "c", 2.5]
czynne = []
print(moja_lista)
print(czynne)
```

```
[1, 2, 3, 'a', 'b', 'c', 2.5]
[]
```

Python - listy

- elementy listy są indeksowane, pierwszy element ma indeks 0, drugi element ma indeks 1, itd.
- odwołując się do elementów listy podajemy nazwę listy i w nawiasach kwadratowych numer elementu

```
bierne = ["rezystor", "cewka", "kondensator"]
print(bierne[0])
print(bierne[1])
print(f"Element bierny to: {bierne[0]}")
```

```
rezystor
cewka
Element bierny to: rezystor
```

Python - listy

- ostatni element listy ma dodatkowy indeks wynoszący -1

```
bierne = ["rezystor", "cewka", "kondensator"]
print(f"Ostatni element to: {bierne[-1]}")
```

```
Ostatni element to: kondensator
```

- indeks -2 określa drugi element od końca listy
- indeks -3 określa trzeci element od końca listy

```
bierne = ["rezystor", "cewka", "kondensator"]
print(bierne[-2])
```

```
cewka
```

Python - listy

- można modyfikować wartości elementów zapisanych na liście

```
bierne = ["rezystor", "cewka", "kondensator"]
bierne[2] = "capacitor"
print(bierne)
```

```
['rezystor', 'cewka', 'capacitor']
```

- dodanie elementu na końcu listy - metoda append()

```
bierne = ["rezystor", "cewka", "kondensator"]
bierne.append("dioda")
print(bierne)
```

```
['rezystor', 'cewka', 'kondensator', 'dioda']
```

Python - listy

- wstawienie elementu w dowolnym miejscu listy - metoda insert()

```
bierne = ["rezystor", "cewka", "kondensator"]
bierne.insert(0,"dioda")
print(bierne)
```

```
['dioda', 'rezystor', 'cewka', 'capacitor']
```

- usunięcie elementu listy o znanym indeksie - funkcja del

```
bierne = ["rezystor", "cewka", "kondensator"]
del bierne[2]
print(bierne)
```

```
['rezystor', 'cewka']
```

Python - listy

- metoda pop() usuwa ostatni element z listy i zwraca jego wartość

```
bierne = ["rezystor", "cewka", "kondensator"]
print(bierne)
element = bierne.pop()
print(f"Usunięty element to: {element}")
print(bierne)
```

```
['rezystor', 'cewka', 'kondensator']
Usunięty element to: kondensator
['rezystor', 'cewka']
```

Python - listy

- argumentem metody pop() może być indeks usuwanego elementu

```
bierne = ["rezystor", "cewka", "kondensator"]
print(bierne)
element = bierne.pop(1)
print(f"Usunięty element to: {element}")
print(bierne)
```

```
['rezystor', 'cewka', 'kondensator']
Usunięty element to: cewka
['rezystor', 'kondensator']
```

Python - listy

- usunięcie elementu na podstawie wartości - metoda remove()

```
bierne = ["rezystor", "cewka", "kondensator"]
print(bierne)
bierne.remove("rezystor")
print(bierne)
```

```
['rezystor', 'cewka', 'kondensator']
['cewka', 'kondensator']
```

- metoda remove() usuwa tylko pierwsze wystąpienie elementu

Python - listy

- usunięcie wszystkich elementów z listy - metoda clear()

```
bierne = ["rezystor", "cewka", "kondensator"]
bierne.clear()
print(bierne)
```

```
[]
```

- określenie wielkości listy - funkcja len()

```
bierne = ["rezystor", "cewka", "kondensator"]
ile = len(bierne)
print(f"Liczba elementów listy to: {ile}")
```

```
Liczba elementów listy to: 3
```

Python - pętla for i listy

- wyświetlenie elementów listy - w zmiennej element umieszczane są kolejne elementy listy bierne

```
bierne = ["rezystor", "cewka", "kondensator"]
for element in bierne:
    print(element)
```

```
rezystor
cewka
kondensator
```

- w powyższej metodzie nie musimy znać liczby elementów na liście

Python - pętla for i listy

- stosując funkcję enumerate() możemy otrzymać indeksy elementów listy

```
bierne = ["rezystor", "cewka", "kondensator"]
for indeks, element in enumerate(bierne):
    print(f"Indeks: {indeks}, Wartość: {element}")
```

```
Indeks: 0, Wartość: rezystor
Indeks: 1, Wartość: cewka
Indeks: 2, Wartość: kondensator
```

Python - pętla for i listy

- utworzenie listy liczb przy zastosowaniu funkcji range()

```
liczby = list(range(1,10))
for nr in liczby:
    print(nr)
```

```
1
2
3
4
5
6
7
8
9
```

Python - pętla while

- Pętla while służy do wykonywania bloku kodu dopóki warunek jest spełniony (przyjmuje wartość True)
- Składnia pętli while:

```
while warunek:
    # kod wykonywany dopóki warunek jest prawdziwy
```

  - Przykład:

```
numer = 1
while numer <= 5:
    print(numer)
    numer = numer + 1
```

```
1
2
3
4
5
```

Python - pętla while i instrukcja break

- instrukcja break jest używana wewnątrz pętli while, aby przerwać jej działanie
- Przykład: poszukiwanie pierwszej liczby podzielnej zarówno przez 3, jak i przez 7 w przedziale [1, 100]

```
liczba = 1
while liczba <= 100:
    if liczba % 3 == 0 and liczba % 7 == 0:
        print("Znaleziona liczba to:", liczba)
        break
    liczba += 1
```

```
Znaleziona liczba to: 21
```

Python - pętla while i instrukcja continue

- instrukcja continue służy do przerwania aktualnej iteracji pętli while i powrót na początek pętli
- Przykład: wyświetlenie tylko liczb nieparzystych z przedziału [1, 10]

```
liczba = 0
while liczba < 10:
    liczba = liczba + 1
    if liczba % 2 == 0:
        continue
    print(liczba, end = " ")
```

```
1 3 5 7 9
```

Koniec wykładu nr 3

Dziękuję za uwagę!