Programowanie Python 1

(CP1S02005)

Politechnika Białostocka - Wydział Elektryczny

Cyfryzacja przemysłu, sem. II, studia stacjonarne I stopnia

Rok akademicki 2023/2024

Wykład nr 5 (03.04.2024)

dr inż. Jarosław Forenc

Plan wykładu nr 5

- Lista
  - implementacja
  - metody tworzenia, lista składana
  - indeksy elementów, wycinki
  - funkcje i metody
  - wybrane operacje
- Krotka
  - implementacja
  - metody tworzenia
  - indeksy elementów, wycinki

Python - lista

- Lista (ang. list) - modyfikowalna struktura danych przechowująca kolekcję elementów ułożonych w określonej kolejności

```
liczby = [5, 2, 8, 3, 0, 2, 1]
funkcje = ["print", "input", "sort", "sqrt"]
```

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

Python - lista, metody tworzenia

- użycie pary nawiasów kwadratowych do oznaczenia pustej listy

```
moja_lista = []
print(moja_lista)
```

```
[]
```

- użycie nawiasów kwadratowych, oddzielenie elementów przecinkami

```
imiona1 = ["Ala"]
imiona2 = ["Piotr", "Kuba", "Kacper"]
print(imiona1)
print(imiona2)
```

```
['Ala']
['Piotr', 'Kuba', 'Kacper']
```

Python - lista, metody tworzenia

- zastosowanie konstruktora bezargumentowego: list()

```
moja_lista = list()
print(moja_lista)
```

```
[]
```

- zastosowanie konstruktora z argumentem w postaci obiektu iterowalnego: list(iterable)
- obiekt iterowalny może być sekwencją, kontenerem obsługującym iterację lub obiektem iteratora
- konstruktor buduje listę, której elementy są takie same i w tej samej kolejności, co elementy obiektu iterowalnego

Python - lista, metody tworzenia

- inna lista jako argument konstruktora: list(iterable)

```
imiona = ["Piotr", "Kuba", "Kacper"]
names = list(imiona)
print(names)
```

```
['Piotr', 'Kuba', 'Kacper']
```

- inne obiekty iterowalne jako argumenty konstruktora: list(iterable)

```
litery = list("ABCDE")
liczby = list((1, 2, 3, 4, 5))
print(litery)
print(liczby)
```

```
['A', 'B', 'C', 'D', 'E']
[1, 2, 3, 4, 5]
```

Python - lista, metody tworzenia

- zastosowanie listy składanej (ang. list comprehension)
- lista składana łączy w pojedynczym wierszu kodu pętlę for, utworzenie nowego elementu i jego automatyczne dołączenie do listy

```
nazwa_listy = [wyrażenie for element in sekwencja]
```

- przykład: lista kwadratów liczb z przedziału od 1 do 10

```
kwadraty = [x**2 for x in range(1,11)]
print(kwadraty)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

- pętla for generuje liczby x dla wyrażenia, wartości wyrażenia dodawane są do listy kwadraty (uwaga: po pętli for nie ma dwukropka)

Python - lista, metody tworzenia

- zastosowanie listy składanej (ang. list comprehension)
- możliwe jest również dodanie warunku do listy składanej w celu filtrowania elementów

```
nazwa = [wyrażenie for element in sekwencja if warunek]
```

- przykład: lista kwadratów liczb parzystych z przedziału od 1 do 10

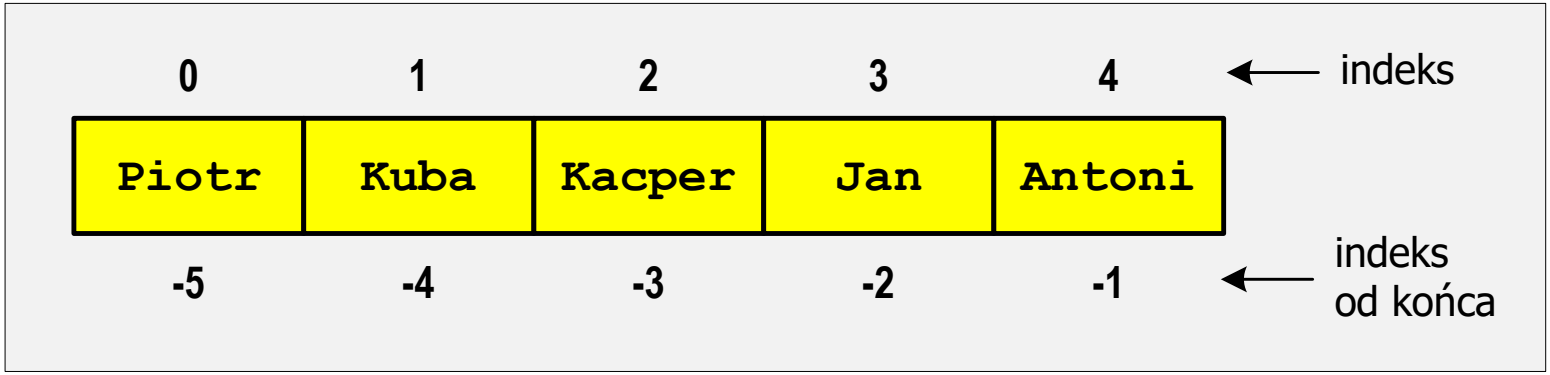
```
kwadraty = [x**2 for x in range(1,11) if x % 2 == 0]
print(kwadraty)
```

```
[4, 16, 36, 64, 100]
```

Python - lista, indeksy elementów

- lista jest uporządkowaną kolekcją, dostęp do jej dowolnego elementu można uzyskać przez podanie jego położenia (indeksu)
- pierwszy element listy ma indeks równy 0, drugi element listy ma indeks równy 1, itd.
- ostatni element listy ma dodatkowy indeks wynoszący -1, drugi element od końca listy ma indeks -2, itd.

```
imiona = ["Piotr", "Kuba", "Kacper", "Jan", "Antoni"]
```



Python - lista, indeksy elementów

- odwołując się do elementów listy podajemy nazwę listy i w nawiasach kwadratowych numer elementu: nazwa_listy[indeks]

```
imiona = ["Piotr", "Kuba", "Kacper", "Jan", "Antoni"]
print(f"Witaj {imiona[1]}!")
print(f"Witaj {imiona[-1]}!")
```

```
Witaj Kuba!
Witaj Antoni!
```

- użycie nieprawidłowego indeksu spowoduje błąd kompilacji

```
print(f"Witaj {imiona[5]}!")
```

```
Traceback (most recent call last):
  File "d:\MyApp.py", line 2, in <module>
    print(f"Witaj {imiona[5]}!")
                   ~~~~~~^^^
IndexError: list index out of range
```

Python - lista, indeksy elementów

- stosując dwukropek (:) można pracować na fragmentach listy nazywanych wycinkami (ang. slices)

```
nazwa_listy[indeks_początkowy : indeks_końcowy : krok]
```

- wycinek rozpoczyna się od elementu o podanym indeksie początkowym i kończy na elemencie o indeksie o jeden mniejszym niż podany indeks końcowy
- każdy z elementów w nawiasach kwadratowych może być pominięty, wtedy przyjmowane są wartości domyślne
- indeks początkowy - wartość domyślna to 0 (indeks pierwszego elementu)
- indeks końcowy - wartość domyślna to len(lista) (liczba elementów listy)
- krok - wartość domyślna to 1

Python - lista, indeksy elementów

- przykłady tworzenia wycinków

```
imiona = ["Piotr", "Kuba", "Kacper", "Jan", "Antoni"]

print(imiona[0:3])
print(imiona[:4])
print(imiona[-3:])

print(imiona[::2])

print(imiona[:])
```

```
['Piotr', 'Kuba', 'Kacper']
['Piotr', 'Kuba', 'Kacper', 'Jan']
['Kacper', 'Jan', 'Antoni']

['Piotr', 'Kacper', 'Antoni']

['Piotr', 'Kuba', 'Kacper', 'Jan', 'Antoni']
```

Python - lista, indeksy elementów

- wyświetlenie listy z wykorzystaniem pętli for

```
imiona = ["Piotr", "Kuba", "Kacper"]
for imię in imiona:
    print(f"Witaj {imię}!")
```

```
Witaj Piotr!
Witaj Kuba!
Witaj Kacper!
```

- wyświetlenie wycinka listy z wykorzystaniem pętli for

```
imiona = ["Piotr", "Kuba", "Kacper"]
for imię in imiona[0:2]:
    print(f"Witaj {imię}!")
```

```
Witaj Piotr!
Witaj Kuba!
```

Python - listy, funkcje i metody

Funkcja	Wynik
`len(lista)`	zwraca liczbę elementów znajdujących się na liście
`min(lista)`	zwraca wartość najmniejszego elementu listy
`max(lista)`	zwraca wartość największego elementu listy
`sum(lista)`	zwraca sumę elementów znajdujących się na liście
`sum(lista,start=0)`	zwraca sumę elementów znajdujących się na liście, start to opcjonalny parametr określający wartość początkową sumy (domyślnie 0)
`sorted(lista)`	zwraca posortowaną listę elementów, nie modyfikuje oryginalnej listy

Python - listy, funkcje i metody

Funkcja	Wynik
`del lista[i:j]`	usuwa elementy listy o indeksach od i do j-1, ten sam wynik daje: lista[i:j] = []
`del lista[i:j:k]`	usuwa elementy listy o indeksach od i do j-1 z krokiem k
`lista.append(x)`	dodaje element x na końcu listy
`lista.clear()`	usuwa wszystkie elementy z listy, ten sam wynik daje: del lista[:]
`lista.copy(x)`	tworzy kopię listy, ten sam wynik daje: lista[:]
`lista.extend(t)`	dodaje na końcu listy zawartość obiektu t

Python - listy, funkcje i metody

Funkcja	Wynik
`lista.insert(i,x)`	wstawia do listy wartość x na pozycji o indeksie i
`lista.pop()`	usuwa ostatni element z listy i zwraca jego wartość
`lista.pop(i)`	usuwa i-ty element z listy i zwraca jego wartość
`lista.remove(x)`	usuwa pierwszy element z listy równy x
`lista.reverse()`	odwraca kolejność elementów znajdujących się na liście
`lista.sort()`	sortuje elementy znajdujące się na liście

Python - lista, operacje (in)

- sprawdzenie, czy wartość znajduje się na liście - słowo kluczowe in

```
import random

liczby = [random.randint(0,9) for _ in range(10)]
print(liczby)

nr = int(input("Podaj liczbę: "))

if nr in liczby:
    print(f"Liczba {nr} znajduje się na liście")
else:
    print(f"Liczba {nr} nie znajduje się na liście")
```

```
[1, 4, 1, 8, 5, 4, 0, 9, 0, 1]
Podaj liczbę: 1
Liczba 1 znajduje się na liście
```

- podkreślenie po słowie kluczowym for jest często stosowane, gdy zmienna iteracyjna nie jest używana wewnątrz pętli

Python - lista, operacje (not in)

- sprawdzenie, czy wartość nie znajduje się na liście - słowo kluczowe not in

```
import random

liczby = [random.randint(0,9) for _ in range(10)]
print(liczby)

nr = int(input("Podaj liczbę: "))

if nr not in liczby:
    print(f"Liczba {nr} nie znajduje się na liście")
else:
    print(f"Liczba {nr} znajduje się na liście")
```

```
[2, 5, 8, 2, 0, 3, 9, 1, 7, 4]
Podaj liczbę: 6
Liczba 6 nie znajduje się na liście
```

Python - lista, operacje (czy lista jest pusta?)

- sprawdzenie, czy lista nie jest pusta

```
bierne = ["rezystor", "cewka", "kondensator"]

if bierne:
    print("Lista 'bierne' nie jest pusta")
else:
    print("Lista 'bierne' jest pusta")
```

```
Lista 'bierne' nie jest pusta
```

- jeśli nazwa listy pojawi się w poleceniu if, to zwracana jest wartość True, gdy lista zawiera przynajmniej jeden element
- w przypadku pustej listy zwracaną wartością będzie False

Python - lista, operacje (kopia listy)

- przypisanie nowej nazwy do istniejącej listy nie utworzy jej kopii

```
bierne = ["rezystor", "cewka"]

new_bierne = bierne
bierne.append("kondensator")

print(f"bierne =     {bierne}")
print(f"new_bierne = {new_bierne}")
```

```
bierne =     ['rezystor', 'cewka', 'kondensator']
new_bierne = ['rezystor', 'cewka', 'kondensator']
```

- obie zmienne prowadzą do tego samego obiektu w pamięci (listy)

Python - lista, operacje (kopia listy)

- utworzenie kopii listy wymaga zastosowania wycinka zawierającego całą listę początkową

```
bierne = ["rezystor", "cewka"]

new_bierne = bierne[:]
bierne.append("kondensator")

print(f"bierne =     {bierne}")
print(f"new_bierne = {new_bierne}")
```

```
bierne =     ['rezystor', 'cewka', 'kondensator']
new_bierne = ['rezystor', 'cewka']
```

- otrzymujemy dwa oddzielne obiekty w pamięci (dwie listy)

Python - lista, operacje (przenoszenie elementów)

- do przenoszenia elementów z jednej listy na drugą listę lepiej jest stosować pętlę while niż pętlę for (wewnątrz pętli for nie należy modyfikować listy, która jest przeglądana)

```
bierne = ["rezystor", "cewka", "kondensator"]
new_bierne = []

while bierne:
    element = bierne.pop()
    print(f"Przenoszony element: {element}")
    new_bierne.append(element)

print(new_bierne)
```

```
Przenoszony element: kondensator
Przenoszony element: cewka
Przenoszony element: rezystor
['kondensator', 'cewka', 'rezystor']
```

Python - lista, operacje (usunięcie elementów)

- usunięcie z listy wartości, które występują wiele razy

```
names = ["Jan", "Ola", "Jan", "Ela", "Ela", "Ula"]
print(names)

element = input("Podaj element do usunięcia: ")
while element in names:
    names.remove(element)

print(names)
```

```
['Jan', 'Ola', 'Jan', 'Ela', 'Ela', 'Ula']
Podaj element do usunięcia: Ela
['Jan', 'Ola', 'Jan', 'Ula']
```

Python - krotka

- Krotka (ang. tuple) - niemodyfikowalna struktura danych przechowująca kolekcję elementów ułożonych w określonej kolejności (lista elementów, której nie można zmienić)

```
liczby = (5, 2, 8, 3, 0, 2, 1)
funkcje = ("print", "input", "sort", "sqrt")
```

  - elementy krotki umieszczane są wewnątrz nawiasów zwykłych i rozdzielane przecinkami
  - krotki mogą zawierać elementy różnych typów danych (liczby, łańcuchy znaków, listy, inne krotki, słowniki)
  - zajmują mniej miejsca w pamięci niż listy, co czyni je bardziej efektywnymi, gdy dane będą tylko odczytywane

Python - krotka, metody tworzenia

- użycie pary nawiasów zwykłych do oznaczenia pustej krotki

```
moja_krotka = ()
print(moja_krotka)
```

```
()
```

- użycie nawiasów zwykłych, oddzielenie elementów przecinkami

```
imiona = ("Piotr", "Kuba", "Kacper")
print(imiona)
```

```
('Piotr', 'Kuba', 'Kacper')
```

Python - krotka, metody tworzenia

- definicja krotki jednoelementowej wymaga dodania na końcu przecinka

```
imie1 = ("Ala",)
imie2 = "Ola",
print(imie1)
print(imie2)
```

```
('Ala',)
('Ola',)
```

- brak przecinka na końcu spowoduje utworzenie zwykłej zmiennej, a nie krotki

```
imie = ("Ala")
print(imie)
```

```
Ala
```

Python - krotka, metody tworzenia

- zastosowanie konstruktora bezargumentowego: tuple()

```
moja_krotka = tuple()
print(moja_krotka)
```

```
()
```

- zastosowanie konstruktora z argumentem w postaci obiektu iterowalnego: tuple(iterable)
- obiekt iterowalny może być sekwencją, kontenerem obsługującym iterację lub obiektem iteratora
- konstruktor buduje krotkę, której elementy są takie same i w tej samej kolejności, co elementy obiektu iterowalnego

Python - krotka, metody tworzenia

- inna krotka (lub lista) jako argument konstruktora: tuple(iterable)

```
imiona = ("Piotr", "Kuba", "Kacper")
names = tuple(imiona)
print(names)
```

```
('Piotr', 'Kuba', 'Kacper')
```

- inne obiekty iterowalne jako argumenty konstruktora: tuple(iterable)

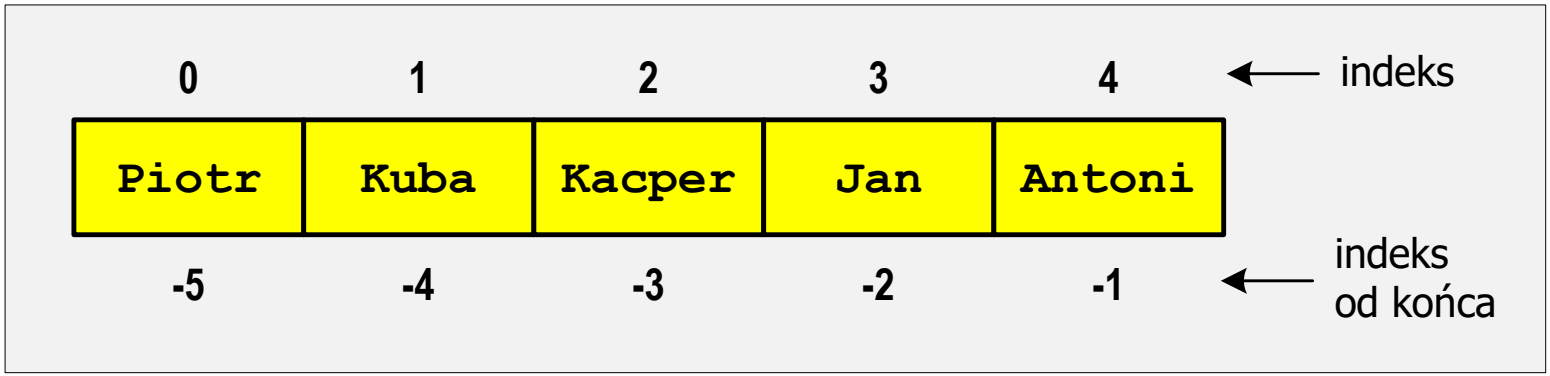
```
litery = tuple("ABCDE")
liczby = tuple((1, 2, 3, 4, 5))
print(litery)
print(liczby)
```

```
('A', 'B', 'C', 'D', 'E')
(1, 2, 3, 4, 5)
```

Python - krotka, indeksy elementów

- krotka, podobnie jak lista, jest uporządkowaną kolekcją, dostęp do jej dowolnego elementu można uzyskać przez podanie jego indeksu

```
imiona = ("Piotr", "Kuba", "Kacper", "Jan", "Antoni")
```



- odwołując się do elementów krotki podajemy nazwę krotki i w nawiasach kwadratowych numer elementu: nazwa_krotki[indeks]
- stosując dwukropek (:) można pracować na fragmentach krotki (wycinkach)

Python - lista, indeksy elementów

- przykłady odwołań do elementów krotki

```
imiona = ("Piotr", "Kuba", "Kacper", "Jan", "Antoni")

print(imiona[0])
print(imiona[1])
print(imiona[-1])

print(imiona[2:])
print(imiona[::2])
```

```
Piotr
Kuba
Antoni

('Kacper', 'Jan', 'Antoni')
('Piotr', 'Kacper', 'Antoni')
```

Python - krotka

- próba zmiany wartości elementu krotki zakończy się błędem

```
imiona = ("Piotr", "Kuba", "Kacper", "Jan", "Antoni")
imiona[0] = "Paweł"
```

```
Traceback (most recent call last):
  File "d:\MyApp.py", line 2, in <module>
    imiona[0] = "Paweł"
    ~~~~~~^^^
TypeError: 'tuple' object does not support item assignment
```

- można nadpisać krotkę nowymi wartościami

```
imiona = ("Piotr", "Kuba", "Kacper", "Jan", "Antoni")
print(imiona)

imiona = ("Marta", "Anna", "Grażyna", "Barbara")
print(imiona)
```

Koniec wykładu nr 5

Dziękuję za uwagę!