Listy, krotki (tuple)

1. Listy

Lista to typ danych używany do przechowywania wielu obiektów. Przypomina dobrze znaną z innych języków tablicę. Stanowi uporządkowaną i modyfikowalną strukturę danych (różnego typu) oddzielanych od siebie przecinkami, umieszczonych w nawiasie kwadratowym, np.:

```
moja lista = [1, None, True, "Jestem łańcuchem znaków", 256, 0]
```

Listy mogą być indeksowane i aktualizowane, np.:

```
moja_lista = [1, None, True, 'Jestem łańcuchem znaków', 256, 0]
print(moja_lista[3]) # daje na wyjściu: Jestem łańcuchem znaków
print(moja_lista[-1]) # daje na wyjściu: 0
moja_lista[1] = '?'
print(moja_lista) # daje na wyjściu: [1, '?', True, 'Jestem łańcuchem znaków', 256, 0]

moja_lista.insert(0, "first")
moja_lista.append("last")
print(moja_lista) # daje na wyjściu: ['first', 1, '?', True, 'Jestem łańcuchem znaków', 256, 0, 'last']
```

Listy mogą być zagnieżdżone, np.:

```
moja \ lista = [1, 'a', ["list", 64, [0, 1], False]].
```

Elementy list jak i same listy mogą być usuwane, np.:

```
moja_lista = [1, 2, 3, 4]
del moja_lista[2]
print(moja_lista) # daje na wyjściu: [1, 2, 4]
del moja lista # usuwa całą listę
```

po elementach listy można iterować przy użyciu pętli for, np.:

Funkcja len() może być użyta do sprawdzenia długości listy, np.:

```
moja_lista = ["biały", "różowy", "niebieski", "żółty", "zielony"]
print(len(moja_lista)) # daje na wyjściu 5
del moja_lista[2]
print(len(moja lista)) # daje na wyjściu 4
```

2. Przetwarzanie List

Jeśli masz listę lista_1, to następujące przypisanie: lista_2 = lista_1 nie tworzy kopii listy lista_1, ale powoduje, że zmienne lista_1 i lista_2 wskazują jedną i tę samą listę w pamięci. Na przykład:

```
pojazdy_1 = ['auto', 'rower', 'motocykl']
print(pojazdy_1) # daje na wyjściu: ['auto', 'rower', 'motocykl']

pojazdy_2 = pojazdy_1
del pojazdy_1[0] # usuwa 'auto'
print(pojazdy 2) # daje na wyjściu: ['rower', 'motocykl']
```

Jeśli chcesz skopiować listę lub część listy, możesz to zrobić, wykonując wycinanie:

```
kolory = ['czerwony', 'zielony', 'pomarańczowy']
skopiuj_cale_kolory = kolory[:]  # skopiuj całą listę
skopiuj czesc kolory = kolory[0:2] # skopiuj część listy
```

Możesz również stosować negatywne indeksy do wykonywania wycinania. Na przykład:

```
przykladowa_lista = ["A", "B", "C", "D", "E"]
nowa_lista = przykladowa_lista[2:-1]
print(nowa lista) # daje na wyjściu: ['C', 'D']
```

Parametry start oraz end są opcjonalne przy wykonywaniu wycinania: lista[start:end], np.:

```
moja_lista = [1, 2, 3, 4, 5]
wycinek_pierwszy = moja_lista[2: ]
wycinek_drugi = moja_lista[:2]
wycinek_trzeci = moja_lista[-2: ]

print(wycinek_pierwszy) # daje na wyjściu: [3, 4, 5]
print(wycinek_drugi) # daje na wyjściu: [1, 2]
print(wycinek_trzeci) # daje na wyjściu: [4, 5]
```

Możesz usuwać wycinki używając instrukcji del:

```
moja_lista = [1, 2, 3, 4, 5]
del moja_lista[0:2]
print(moja_lista) # daje na wyjściu: [3, 4, 5]

del moja_lista[:]
print(moja_lista) # usuwa zawartość listy, daje na wyjściu: []
```

Możesz sprawdzić, czy niektóre elementy istnieją w liście, lub nie używając słów kluczowych in oraz not in, np.:

```
moja_lista = ["A", "B", 1, 2]
print("A" in moja_lista) # daje na wyjściu: True
print("C" not in moja_lista) # daje na wyjściu: True
print(2 not in moja_lista) # daje na wyjściu: False
```

3. Sortowanie list

Możesz używać metody sort() do sortowania elementów listy, np.:

```
moja_lista = [5, 3, 1, 2, 4]
print(moja_lista)
moja_lista.sort()
print(moja_lista) # daje na wyjściu: [1, 2, 3, 4, 5]
```

Istnieje również metoda o nazwie reverse(), której możesz użyć do odwrócenia listy, np.:

```
moja_lista = [5, 3, 1, 2, 4]
print(moja_lista)
moja_lista.reverse()
print(moja lista) # daje na wyjściu: [4, 2, 1, 3, 5]
```

Operator **in** sprawdza czy dany element znajduje się na liście. Operator **not in** zwraca prawdę gdy podanego elementu nie ma na liście.

4. Krotki (tuple)

Krotki są niezmieniane, co oznacza, że nie można zmienić ich elementów (nie można rozszerzać krotek, modyfikować ani usuwać elementów krotek). Poniższy fragment kodu spowoduje zgłoszenie wyjątku:

```
moja_krotka = (1, 2.0, "lancuch znakow", [3, 4], (5, ), True)
moja_krotka[2] = "guitar"  # a TypeError exception will be raised
```

Możesz jednak usunąć krotkę jako całość:

```
moja_krotka = 1, 2, 3,
del moja_krotka
print(moja krotka)  # NameError: name 'moja krotka' is not defined
```

Jeśli chcemy, możemy iterować po elementach krotki (Przykład 1), sprawdzić czy krotka zawiera (lub nie) jakiś element (Przykład 2). Możemy także użyć funkcji len() i zbadać liczbę elementów zawartych w krotce (Przykład 3). Co więcej, możemy ze sobą łączyć/multiplikować elementy krotki (Przykłady:

```
t1 = (1, 2, 3)
for elem in t1:
    print(elem)
```

Krotkę można powołać do życia również za pomocą wbudowanej funkcji Pythona tuple(). Taki zabieg jest szczególnie przydatny w sytuacji gdy chcemy dokonać konwersji jakiegoś obiektu iterowalnego, czyli takiego, który pozwala na dostęp do zawartych w nim elementów w sposób sekwencyjny (np. lista, łańcuch znaków) do krotki:

```
moja_krotka = tuple((1, 2, "lancuch znakow"))
print(moja_krotka)
lst = [2, 4, 6]
print(lst)  # daje na wyjściu: [2, 4, 6]
print(type(lst))  # daje na wyjściu: <class 'list'>
tup = tuple(lst)
print(tup)  # daje na wyjściu: (2, 4, 6)
print(type(tup))  # daje na wyjściu: <class 'tuple'>
```