Lista

Lista jest podstawowym kontenerem danych języka python. Możemy w niej przechowywać dane różnego typu.
Deklaracji dokonujemy w następujący sposób:
nazwa_listy = [] – w nawiasach kwadratowych umieszczamy elementy, które chcemy umieścić w liście podczas deklaracji, podczas deklaracji listy można jej nie uzupełniać danymi.

Najpopularniejszy zbiór metod typu listy

- append – dodaje element na koniec listy
- insert – dodaje element na wybrane miejsce listy
- pop – jeżeli nie podamy żadnego indesku usuwa ostatni element z listy, jeżeli podamy indeks z listy usunięty zostanie element na wybranej pozycji
- remove – usuwa z listy pierwszą napotkaną wartość, która została podana jako argument
- del – z listy zostanie usunięty element o podanym indeksie
- extend – dodanie sekwencji do listy, sekwencja zostaje dodana na końcu listy
- reverse – odwraca kolejność listy
- sort – sortowanie

```
lista = ['wyraz', 4.23, 5.6, 2, 10, [4,5,6]]

lista.append('slow')
lista.append(5.75)
print(lista)

lista.insert(1, 'pierwszy')
lista.insert(7, 3)
print(lista)

lista.pop()
print(lista)

lista.pop(2)
print(lista)

lista.remove('pierwszy')
print(lista)

del lista[5]
print(lista)

lista.extend((2, 2, 'n'))
print(lista)
```

lista.reverse()
print(lista)

nowa_lista = [7, 5, 8.2, 1, 2.2, 1.1, 10, 6, 3]
nowa_lista.sort()
print(nowa_lista)

Słownik

Jest kontenerem danych przechowującym zbiór danych w postaci klucz:wartość. Za indeksowanie w słowniku odpowiadają klucze.

#Deklaracja słownika
slownik = {1: 22, 2: 22, 3:33, 4.5:10, "cos":"ktos", 4.6:'wartosci'}

Dodanie klucza wraz z wartością do słownika
slownik['klucz'] = 'wartosc'
slownik['6'] = 1.1

Wyświetlenie wartości klucza
print(slownik[4.5])

Usuwanie ze słownika za pomocą klucza
slownik.pop(4.5)

Wyświetlenie wszystkich kluczy słownika
print(slownik.keys())

Wyświetlenie wszystkich wartości słownika
print(slownik.values())

Usunięcie pary klucz:wartość za pomocą del
del slownik[1]

Wprowadzanie danych

Aby wprowadzić dane należy użyć komendy inputa

napis = input("Wpisz dowolny komunikat: ")

print(napis)

print(type(napis))

Dane, które wprowadzamy za pomocą komendy input są typu string, jeżeli chcemy wprowadzić jakąś liczbę musimy dokonać rzutowania typów na typ int lub float, po jej wczytaniu. Przykłady rzutowań do typu int i float.

liczba = input("Wpisz dowolną liczbę: ")

print(liczba)

print(type(liczba))

liczba = int(liczba) # miejsce rzutowania do int

print(type(liczba))

liczba = input("Wpisz dowolną liczbę: ")

print(liczba)

print(type(liczba))

liczba = float(liczba) # miejsce rzutowania do float

print(liczba)

print(type(liczba))

Do wprowadzania danych możemy użyć także komendy readline() i write(s), musimy pamiętać jednak od zaimportowaniu modułu sys. Przykład wczytania danych za pomocą komendy readline().

import sys as system

system.stdout.write("wpisz dowolny komunikat: ")

napis = system.stdin.readline()

system.stdout.write(napis)

Instrukcja Warunkowa

Składnia

```
if warunek_1:
        Instrukcje_1
elif warunek_2:
        Instrukcje_2:
…
elif warunek_n:
        Instrukcje_n
else:
        Inne_instrukcje]
```

Operatory porównania wykorzystywane w instrukcjach warunkowych

== - operator równości, sprawdza czy x jest równy y

!= - sprawdza czy jeden obiekt różni się od drugiego, sprawdza czy x różni się od y

> - większy niż, sprawdza czy x jest większy od y

< - mniejszy niż, sprawdza czy x jest mniejszy od y

>= - większy niż lub równy, sprawdza czy x jest większy lub równy y

<= - mniejszy niż lub równy, sprawdza czy x jest mniejszy lub równy y

Przykład pierwszy: Pobieramy dwie liczby całkowite i sprawdzamy, która jest większa

```
a = input("podaj pierwszą liczbę: ")
b = input("podaj pierwszą liczbę: ")
a = int(a)
b = int(b)
#przy wyświetlaniu zmieniamy liczbe na string
if a > b:
   print("liczba " + str(a) + " jest większa")
elif a < b:
   print("liczba " + str(b) + " jest większa")
else:
   print("wprowadzone liczby są równe")
```

Przykład drugi: Pobiera dwie liczby całkowite i sprawdza czy liczby są równe:

```
a = input("podaj pierwszą liczbę: ")

b = input("podaj pierwszą liczbę: ")

a = int(a)

b = int(b)

if a == b:

    print("wprowadzone liczby są równe")

else:

    print("wprowadzone liczby nie są równe")
```

W instrukcjach warunkowych możemy używać również operatorów logiczny AND(&) lub OR(|)

Przykład trzeci: Pobieramy cztery liczb całkowite i sprawdzamy czy liczba a jest większa od liczb b i liczba c jest większa od liczby d

```
a = input("podaj pierwszą liczbę: ")

b = input("podaj drugą liczbę: ")

c = input("podaj trzecią liczbę: ")

d = input("podaj czwartą liczbę: ")

a = int(a)

b = int(b)

c = int(c)

d = int(d)

if (a > b) & (c > d):

    print("liczba a jest większa od liczby b i liczba c jest większa od liczby d")

else:

    print("liczba a jest mniejsza od liczby b lub liczba c jest mniejsza od liczby d")
```

Wynikiem działania operatorów porównania i operatorów logicznych jest typ bool czyli TRUE lub FALSE

Instrukcja iteracyjna for

for licznik in sekwencja:
    Instrukcje
[else:
    inne_instrukcje]

Sekwencją może być łańcuch, lista lub krotka. Od obliczenia sekwencji zaczyna się działanie instrukcji iteracyjnej. Licznik przyjmuje wartość pierwszego elementu wykonuje instrukcje, następnie przyjmuje wartość kolejnego elementu itd.

Do utworzenia sekwencji możemy użyć funkcji range:

rang(start, stop, step)

Przykład pierwszy: chcemy wyświetlić liczby od 1 do 5

```
for x in range(1, 6, 1):
  print(x)
```

Przykład drugi: tworzymy swoją listę i chcemy jej użyć jako sekwencji do wyświetlenia wartości

```
lista = ['a', 5, 6, 7.5]
for x in lista:
  print(x)
```

Przykład trzeci: wyświetlamy elementy z utworzonej listy, po zakończeniu pętli wyświetlamy komunikat

```
lista = ['a', 5, 6, 7.5]
for x in lista:
  print(x)
else:
  print("Wyświetlanie zakończone")
```

Instrukcja iteracyjna while

while warunek:
    instrukcje
[else:
    inne_instrukcje]

Przykład pierwszy: wyświetlamy liczby od 0 do 10, po zakończeniu pętli wyświetlamy komunikat ile liczb zostało wyświetlonych

```
z = 0
while z != 10:
```

```
    print(z)
    z += 1
else:
    print("Wyświetlony zostało " + str(z) + " liczb")
```

Instrukcja break i continue

Instrukcje umieszczamy w pętlach. Sterują działaniem pętli.
Break – przerywa działanie pętli w której się znajduje (ale nie wszystkich pętli jeśli pętle zagnieżdżamy)
Continue – kończy przebieg aktualnej iteracji pętli i przechodzi do następnego przebiegu.

Przykład: Sprwadzamy czy różnica między podaną liczbą a liczbą z listy równa będzie 0

```
lista = [4, 6, 2, 3, 5, 9, 1]
print("Podaj liczbę a sprawdzę czy różnica między wpisaną liczbą a liczbą z listy równa się 0")
liczba = input("wpisz swoją liczbe: ")
licznik = 0
while licznik < lista.__len__():
    if int(liczba) - lista[licznik] == 0:
        print(liczba + " - " + str(lista[licznik]) + " = 0")
        break
    else:
        licznik += 1
else:
    print("Żadna z liczb, które są w liści nie dała odpowiedniego wyniku")
```

Instrukcje iteracyjne zagnieżdżenia

Pętle, instrukcje warunkowe możemy umieszczać jedna w drugiej. Nazywamy to zagnieżdżaniem.

Przykład Tworzymy dwie listy i robimy sumę poszczególnych elementów

```
lista = [4, 6, 2, 3, 5, 9, 1]
lista2 = [7, 3, 4, 6]
suma = []
```

```
for a in lista:
    for b in lista2:
        wynik = a + b
        suma.append(wynik)
    print(suma)
```

Obsługa błędów

Mamy 3 rodzaje błędów:
1. Błędy składniowe – powstają gdy piszemy program niezgodnie z gramatyką języka np. błędy w definicjach funkcji, niezamknięte nawiasy czy cudzysłów bez pary.
2. Błędy czasu wykonania – powodują przerwanie lub niewłaściwe działanie. Mamy możliwość ich przechwycenia i wymuszenia odpowiedniej reakcji np. użytkownik podaje litery a miał wpisywać liczby.
3. Błędy logiczne – błędy w algorytmie lub programie. Nie wykrywalne przez interpreter ale możliwe przez człowieka po analizie programu

Obsługę błędów realizujemy przez specjalny blok instrukcji.
Składnia:

```
try:
        instrukcje
except nazwa_bledu_1:
        awaryjne_instrukcje_1
…
[except nazwa_bledu_n:
        awaryjne_instrukcje_n]
[else:
        blok_bez_błędu]
```

Prostsza składnia:

```
try:
        instrukcje
finally:
        blok_zamykający
```

Przykład

Podajemy dwie liczby do dzielenia, chcemy wyłapać dzielenie przez 0

```
print("Proszę podać pierwszą liczbę")
licz1 = input()
print("Proszę podać drugą liczbę")
licz2 = input()
```

```
try:
    wynik = int(licz1) / int(licz2)
    print("Wynik= " + str(wynik))
except ZeroDivisionError: #nazwa błędu dzielenia przez zero
    print("Tylko Chuck Norris może dzielić przez zero!")
```