Informatyka w Edukacji, XV
UMK Toruń, 2018

PROGRAMOWANIE W PYTHONIE – ALGORYTMY TABLICOWE A LISTY

Grażyna Szabłowicz-Zawadzka
http://metodycy.torun.pl/
m.informatyka@metodycy.torun.pl

1. Lista – typ sekwencyjny zmienny

Do realizacji algorytmów tablicowych w języku Python można zastosować listę. Lista to **typ sekwencyjny zmienny**, a więc możliwe jest **przypisywanie wartości** pojedynczym elementom tego typu. Listy **mogą zawierać wartości różnego typu**.

Tabela 1 Typ sekwencyjny — lista

POLECENIE	OPIS POLECENIA
`lista = [1, 2.0, 'trzy', 4, 5.0]`	przypisanie wartości zmiennej lista

`lista[0]`
`lista[-1]`
`lista[-2]`
	odwołanie do pierwszego elementu listy (`1`),
	odwołanie do ostatniego elementu listy (`5.0`),
	odwołanie do przedostatniego elementu listy (`4`)

`lista[2:]`	od elementu `lista[2]`, wynik: `['trzy', 4, 5.0]`
`lista[1::2]`	co drugi element od elementu `lista[1]`, wynik: `[2.0, 4]`
`lista[0:4:2]`	od lista[0]do elementu `lista[3]` co drugi, wynik: `[1.0, 'trzy']`
`lista *= 2`	wynik: `[1, 2.0, 'trzy', 4, 5.0, 1, 2.0, 'trzy', 4, 5.0]`
`lista = lista[:2]`	skracanie listy, wynik: `[1, 2.0]`
`len(lista)`	długość listy (liczba elementów), wynik: `5.0`
`lista[1] = 6`	przypisanie elementowi `lista[1]` wartości `6`, instrukcja przypisania (elementom listy można przypisywać wartości), wynik: `[1, 6, 'trzy', 4, 5.0]`
`lista[0] += 5`	element `lista[0]` zwiększony o `5`, wynik: `[6, 2.0, 'trzy', 4, 5.0]`

`del lista[2]`	usunięcie elementu `lista[2]` z listy, wynik: `[1, 2.0, 4, 5.0]`
`== != < >`	relacje na listach (listy można porównywać)
`1 in lista 2 not in lista`	sprawdzanie, czy w liście są określone wartości
`lista[i], lista[k] = lista[k], lista[i]`	zamiana elementów listy

Tabela 2 Indeksowanie list

INDEKSOWANIE LIST	
`lista = [7, 8, 9, 2, 5, 6, 1, 10, 3, 4]`	
`lista`	`[7, 8, 9, 2, 5, 6, 1, 10, 3, 4]`
`lista[2:]`	`[9, 2, 5, 6, 1, 10, 3, 4]`
`lista[3::2]`	`[2, 6, 10, 4]`
`lista[::-1]`	`[4, 3, 10, 1, 6, 5, 2, 9, 8, 7]`
`lista[::-2]`	`[4, 10, 6, 2, 8]`
`lista[::3]`	`[7, 2, 1, 4]`
`lista[:4:-2]`	`[4, 10, 6]`
`lista[:4:2]`	`[7, 9]`
`lista[0:5:2]`	`[7, 9, 5]`
`lista[3:9:2]`	`[2, 6, 10]`

Tabela 3 Metody na listach

METODY NA LISTACH	
`lista = [7, 8, 9, 2, 5, 6, 1, 10, 3, 4]`	
`lista.append(11)`	dołączanie elementu do listy, wynik: `[7, 8, 9, 2, 5, 6, 1, 10, 3, 4, 11]`
`lista.extend([13, 14])`	dołączanie listy `[13, 14]` do listy `[7, 8, 9, 2, 5, 6, 1, 10, 3, 4]`, wynik: *`[7, 8, 9, 2, 5, 6, 1, 10, 3, 4, 13, 14]`*
`lista.count(6)`	obliczanie, ile razy podana wartość 6 występuje w liście, wynik: *`1`*
`lista.index(1)`	wyznaczanie pozycji (licząc od 0) pierwszego wystąpienia podanej wartości `1`, wynik: `6`

`lista.insert(4, 0)`	wstawianie liczby `0` do listy na pozycję `4`, wynik: `[7, 8, 9, 2, 0, 5, 6, 1, 10, 3, 4]`
`lista.pop(3)`	zwracanie wartości z podanej pozycji `3` i usunięcie tego element z listy, wynik: `2`, `[7, 8, 9, 5, 6, 1, 10, 3, 4]`
`lista.remove(8)`	usunięcie z listy pierwszej znalezionej wartości `8`, wynik: `[7, 9, 2, 5, 6, 1, 10, 3, 4]`
`lista.reverse()`	odwracanie kolejności elementów listy, wynik: `[4, 3, 10, 1, 6, 5, 2, 9, 8, 7]`
`lista.sort()`	niemalejące porządkowanie listy, wynik: `[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`

2. Lista – przykłady do analizy

Przykład 1. Proste działania na liście jednowymiarowej

Przeanalizuj kody programów wykonujących następujące operacje:

a) obliczenie sumy wszystkich elementów listy,

b) obliczenie iloczynu tych elementów listy, które są mniejsze od `8`,

c) obliczenie liczby tych elementów listy, których numer jest podzielny przez `3`,

d) zwiększenie o `2` wartości tych elementów listy, które mają nieparzysty numer zawarty w przedziale `[1, 7]` i wypisanie tej listy.

Porównaj zaproponowane rozwiązania dla punktów a i b.

Rozwiązanie 1

```
def zad_a(T):
    n = len(T)
    s = 0
    for i in range(n):
        s += T[i]
    return s

def zad_b(T):
    n = len(T)
    s = 1
    for i in range(n):
        if T[i] < 8:
            s *= T[i]
    return s
```

```
def zad_c(T):
    n = len(T)
    s = 0
    for i in range(n):
        if i % 3 == 0:
            s += 1
    return s

def zad_d(T):
    n = len(T)
    for i in range(1, 8, 2):
        T[i] += 2
    return T

print("zad_a = ", zad_a([7, 9, 4, 2, 1, 3, 4, 5, 5]))
print("zad_b = ", zad_b([7, 9, 4, 2, 1, 3, 4, 5, 5]))
print("zad_c = ", zad_c([7, 9, 4, 2, 1, 3, 4, 5, 5]))
print("zad_d = ", zad_d([7, 9, 4, 2, 1, 3, 4, 5, 5]))
```

Wyniki:

zad_a = `40`
zad_b = `16800`
zad_c = `3`
zad_d = `[7, 11, 4, 4, 1, 5, 4, 7, 5]`

Rozwiązanie 2

```
def zad_a(T):
    s = 0
    for k in T:
        s += k
    return s

def zad_b(T):
    s = 1
    for k in T:
        if k < 8:
            s *= k
    return s

print("zad_a = ", zad_a([7, 9, 4, 2, 1, 3, 4, 5, 5]))
print("zad_b = ", zad_b([7, 9, 4, 2, 1, 3, 4, 5, 5]))
```

Wyniki:

zad_a = `40`
zad_b = `16800`

Przykład 2. Funkcje logiczne — sprawdzanie własności elementów listy jednowymiarowej

Przeanalizuj kody programów, które realizują następujące polecenia:

a) sprawdzenie, czy na liście znajduje się element o wartości różnej od 5,

b) sprawdzenie, czy na liście wszystkie elementy mają wartość mniejszą od 10,

c) sprawdzenie, czy na liście jest element o wartości podanej przez użytkownika.

Porównaj zaproponowane rozwiązania.

Rozwiązanie 1

```
def zad_a(T):
    n = len(T)
    for i in range(n):
        if T[i] != 5:
            return True
    return False

def zad_b(T):
    n = len(T)
    for i in range(n):
        if T[i] >= 10:
            return False
    return True

def zad_c(T, k):
    n = len(T)
    for i in range(n):
        if T[i] == k:
            return True
    return False

print("zad_a = ", zad_a([5, 7, 9, 4, 3, 4, 5, 2, 1]))
print("zad_b = ", zad_b([5, 7, 9, 4, 3, 4, 5, 2, 1]))
print("zad_c = ", zad_c([5, 7, 9, 4, 3, 4, 5, 2, 1], 6))
```

Wyniki:

zad_a = `True`
zad_b = `True`
zad_c = `False`

Rozwiązanie 2

```
def zad_a(T):
```

```
    for k in T:
        if k != 5:
            return True
    return False

def zad_b(T):
    for k in T:
        if k >= 10:
            return False
    return True

def zad_c(T, k):
    if k in T:
        return True
    return False

print("zad_a = ", zad_a([5, 7, 9, 4, 3, 4, 5, 2, 1]))
print("zad_b = ", zad_b([5, 7, 9, 4, 3, 4, 5, 2, 1]))
print("zad_c = ", zad_c([5, 7, 9, 4, 3, 4, 5, 2, 1], 6))
```

Wyniki:

zad_a = `True`
zad_b = `True`
zad_c = `False`

Przykład 3. Listy dwuwymiarowe

Przeanalizuj kod programu wykonującego następujące operacje na liście dwuwymiarowej:

a) wypisywanie elementów listy z podziałem na wiersze,

b) obliczenie sumy tych elementów listy, które są podzielne przez 5,

c) zmniejszenie o 4 tych elementów listy, które nie znajdują się na głównej przekątnej i wypisanie tej listy.

Porównaj zaproponowane rozwiązania dla punktów a i b.

Rozwiązanie 1

```
def zad_a(T, m, n):
    for i in range(m):
        print(T[i])

def zad_b(T, m, n):
    s = 0
    for i in range(m):
        for j in range(n):
```

```
            if T[i][j] % 5 == 0:
                s += T[i][j]
    return s

def zad_c(T, m, n):
    for i in range(m):
        for j in range(n):
            if i != j:
                T[i][j] -= 4
        print(T[i])

zad_a([[2, 3, 4, 5], [8, 9, 4, 5], [7, 6, 4, 5]], 3, 4)
print("suma = ", zad_b([[2, 3, 4, 5], [8, 9, 4, 5], [7, 6, 4,
5]], 3, 4))
zad_c([[2, 3, 4, 5], [8, 9, 4, 5], [7, 6, 4, 5]], 3, 4)
```

Wyniki:

```
[2, 3, 4, 5]
[8, 9, 4, 5]
[7, 6, 4, 5]
suma = 15
[2, -1, 0, 1]
[4, 9, 0, 1]
[3, 2, 4, 1]
```

Rozwiązanie 2

```
def zad_a(T):
    for P in T:
        print(P)

def zad_b(T):
    s=0
    for P in T:
        for k in P:
            if k % 5 == 0:
                s += k
    return s

zad_a([[2, 3, 4, 5], [8, 9, 4, 5], [7, 6, 4, 5]])
print("suma = ", zad_b([[2, 3, 4, 5], [8, 9, 4, 5], [7, 6, 4,
5]]))
```

Wyniki:

```
[2, 3, 4, 5]
[8, 9, 4, 5]
[7, 6, 4, 5]
suma =  15
```

3. Zadania do wykonania

Zadanie 1.

Napisz program wykonujący następujące operacje na liście jednowymiarowej:

a) wypisanie wszystkich elementów listy,
b) obliczenie liczby tych elementów listy, których numer nie jest podzielny przez 4,
c) zmniejszenie o 3 tych elementów listy, które mają nieparzysty numer zawarty w przedziale [3, 7] i wypisanie tej listy,
d) sprawdzenie, czy na liście znajduje się element o wartości nieujemnej,
e) obliczenie iloczynu tych elementów listy, których wartość jest różna od 6,
f) sprawdzenie, czy na liście znajduje się element, którego wartość nie zawiera się w przedziale [4, 8].

Zadanie 2.

Napisz program realizujący następujące operacje na liście dwuwymiarowej:

a) wypisywanie elementów listy z podziałem na wiersze,
b) zamiana dwóch wierszy listy: numer 0 i numer 2,
c) zamiana dwóch kolumn listy wskazanych przez użytkownika.

Zadanie 3.

Napisz program realizujący następujące operacje na liście dwuwymiarowej:

a) wypisywanie elementów listy z podziałem na wiersze,
b) obliczenie sumy tych elementów listy, których wartość jest większa od 4,
c) obliczenie liczby tych elementów listy, których wartość jest różna od 0,
d) sprawdzenie, czy na liście znajduje się element, którego wartość jest mniejsza od 5.

Literatura

1. Dawson M., *Python dla każdego. Podstawy programowania*, Helion, Gliwice 2013.
2. Lutz M., *Python. Wprowadzenie*, Helion, Gliwice 2011.
3. Sysło M.M., *Algorytmy*, Helion, Gliwice 2016.
4. Szabłowicz-Zawadzka G., *Informatyka. Podręcznik dla szkół ponadgimnazjalnych. Informatyka Europejczyka, część I, Wydanie III,* Helion, Gliwice 2017.
5. Szabłowicz-Zawadzka G., *Python – programowanie na maturze*, Helion, Gliwice 2018.