## Контрольні запитання

1. Що таке одновимірний масив? Для чого використовують одномірні масиви? Як їх описують в Python?

2. Як в програмі використати значення конкретного елемента одновимірного масиву?

3. Для чого в програмах використовуються двовимірні масиви? Як їх описують в Python?

4. Скільки індексів характеризують конкретний елемент двовимірного масиву?

5. Як в програмі використати значення конкретного елемента двовимірного масиву?

6. Який індекс двовимірного масиву змінюється швидше при послідовному розміщенні елементів масиву в оперативній пам'яті?

## Аудиторна робота

**Приклад 7.1**. Список

1. Список – впорядкована послідовність певних значень, які можуть повторюватися. Для створення списку необхідно записати його елементи через кому у квадратних дужках

```
int_list = [1,2,3,5] #  список із чотирьох цілих чисел
char_list = ['a', 'c', 'z', 'x'] #  список  із чотирьох символів
empty_list = []# empty_list – пустий список

print('Список чисел:', int_list)
print('Список символів:', char_list)
print('Пустий список:', empty_list)
```

Результат

```
Список чисел: [1, 2, 3, 5]
Список символів: ['a', 'c', 'z', 'x']
Пустий список: []
```

2. Значення елемента можна отримувати за його індексом (номером). Індексація починається з нуля.

```
# Створення списку чисел
my_list = [5, 7, 9, 1, 1, 2]
# Виведення  першого елемент
print (my_list [0]); print (my_list)
# Введення індексу
index = int (input ('Введіть номер елемента:'))
# Отримання відповідного елемента
element=my_list[index]
# Виведення  його значення на екран
print (element)
```

Результат

```
[5, 7, 9, 1, 1, 2]
Введіть номер елемента: 2
9
```

3. Якщо використати від'ємні індекси, то обхід елементів розпочинається з останнього. Індекс останнього елемента списку – «–1», передостаннього – «–2».

```
# Створення списку чисел
my_list = [5, 7, 9, 1, 1, 2]
# Отримання передостаннього значення
pre_last = my_list [-2] # pre_last == «1»
print (pre_last)
# Обчислення суми першого і останнього значень
result = my_list[0] + my_list[-1];print(result)
```

Результат

```
1
7
```

4. Методи обробки списків:

– метод *append* додає значення в список

```
my_list = []# Створення пустого списку
my_list.append(3); my_list.append(5)
my_list.append(my_list[0]+my_list[1])

print(my_list) # Виведення списку на екран
```

Результат

```
[3, 5, 8]
```

- видалення елемента списку

```
my_list = [5, 1, 5, 7, 8, 1, 0, -23] # Створення списку чисел
print(my_list) # Виведення списку
# Оператор del видаляє заданий елемент
del my_list[2]
print(my_list) # Виведення списку
```

Результат

```
[5, 1, 5, 7, 8, 1, 0, -23]
[5, 1, 7, 8, 1, 0, -23]
```

5. Заміна елементу списка

```
my_list = [5, 1, 5, 7, 8, 1, 0, -23] # Створення списку чисел
print(my_list) # Виведення списку
length = len(my_list) # Отримання довжини списка
# Введення индексу
index = length
while not -length <= index < length:
 index=int(input('Введіть індекс ел-та списку (від %d до
%d):'
                 % (-length, length - 1)))
# Введення нового значення
value = int(input('Введіть нове значення заданого ел-та: '))
my_list[index]=value # зміна елемента списку
print(my_list) # Виведення списку на екран
```

Результат

```
[5, 1, 5, 7, 8, 1, 0, -23]
```

```
Введіть індекс елемента списку (від -8 до 7): 7
Введіть нове значення заданого елемента: 5
[5, 1, 5, 7, 8, 1, 0, 5]
```

6. Виведення квадратів чисел зі списку

```
my_list = [5, 1, 5, 7, 8, 1, 0, -23] # Створення списку чисел
for x in my_list:
    print('{}^2={}'.format(x, x ** 2))
```

Результат

```
5 ^ 2 = 25
1 ^ 2 = 1
5 ^ 2 = 25
7 ^ 2 = 49
8 ^ 2 = 64
1 ^ 2 = 1
0 ^ 2 = 0
-23 ^ 2 = 529
```

**7.** Функція-конструктор, яка створює список із значення іншого типу

```
empty_list=list() # Створення пустого списку
print(empty_list)

# Створення списку з послідовності range
numbers = list(range(10))
print(numbers)

# Створення списку символів із рядка
chars = list("a string"); print(chars)
```

Результат

```
[]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
['a', ' ', 's', 't', 'r', 'i', 'n', 'g']
```

8. Числа Фібоначчі

```
n=10 # Кількість чисел в послідовності
# Список чисел Фібоначчі (напочатку має дві одиниці)
fibs = [1, 1]
# Повторюємо (n-2) рази, тому що два числа вже є в списку
for i in range(n-2):
   # Додаємо суму двох останніх чисел
   fibs.append(fibs[i]+fibs[i+1])
print(fibs) # Виведення списку на екран
```

Результат

```
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
```

**Приклад 7.2**. Зріз списка – отримання групи елементів за їхніми індексами

1.

```
my_list = [5, 7, 9, 1, 1, 2] # Створення списку чисел
# Отримання зрізу списка від нульового (першого) елемента
# (включаючи його) до третього (четвертого) (не включаючи)
```

```
sub_list = my_list[0:3]
print(sub_list) # Виведення отриманого списку
# Виведення елементів списка від другого до переостаннього
print(my_list[2:-2])
# Виведення ел-тів списка від 4-ого (5-ого) до 5-ого (6-ого)
print(my_list[4:5])
```

Результат

```
[5, 7, 9]
[9, 1]
[1]
```

2. Зріз з використанням кроку

```
my_list = [5, 7, 9, 1, 1, 2]
# Вибір кожного другого ел-та списку (починаючи з першого),
# не включаючи останній елемент
sub_list = my_list[0:-1:2]
print(sub_list) # виведення отриманого списку
# Виведення елементів від 2-ого (3-ього) до передостаннього
# з кроком 2
print(my_list[2:-2:2])
# Виведення ел-тів списка, крім першого, в зворотному
порядку
print(my_list[-1:0:-1])
```

Результат

```
[5, 9, 1]
[9]
[2, 1, 1, 9, 7]
```

3. Будь-який параметр зріза можна пропустити (при умові дотримання правильного розміщення двокрапок)

```
# За замовчуванням початок списка – 0, кінець – довжина списка,
крок – 1
my_list=[5, 7, 9, 1, 1, 2]
# Виведення елементів списка від 2-ого (3-ього) значення до кінця
print(my_list[2:])
# Виведення всіх ел-тів списка від початку до передостаннього ел-
та
print(my_list[:-2])
# Виведення всіх елементів списку в зворотному порядку
print(my_list[::-1])
[-1:0:-1])
```

Результат

```
[9, 1, 1, 2]
[5, 7, 9, 1]
[2, 1, 1, 9, 7, 5]
```

**Приклад 7.3**. Для перевірки входження елемента в список використовують операцію *in*

1. Перевірка, чи є задане число у списку

```
my_list = [5, 7, 9, 1, 1, 2]
# Введення значення
```

```
value=int(input('Введіть число: '))
if value in my_list:
    print('Число входить у список')
else:
    print('Число не входить у список')
```

Результат

```
Введіть число: 5
Число входить у список
```

2. Визначення кількості елеметів списка

```
my_list = [1, 5, 1, 3, 7, 8, 124]
print(len(my_list))
```

Результат

```
7
```

**Приклад 7.4**. Нехай задано два списки цілих випадкових чисел від 0 до 5: [$a_1$, ..., $a_n$] і [$b_1$, ..., $b_n$], $n$=10. Написати програму їх формування. Вивести списки на екран.

```
import random
a=[random.randint(0,5) for i in range(0,10)]
b=[random.randint(0,5) for j in range(0,10)]
print(a); print(b)
```

Результат

```
[4, 3, 5, 5, 1, 4, 5, 2, 1, 5]
[4, 0, 4, 5, 4, 2, 2, 2, 4, 5]
```

**Приклад 7.5**. Нехай згенеровано список із цілих випадкових чисел та нулів [$a_1$, ..., $a_n$]. Написати програму визначення елементів, розміщених після першого нульового. Вивести на екран початковий та отриманий списки.

```
import random
arr=random.sample(range(-6,6), 12)
print("our random list: " , arr)
flag=0
while flag==0:
 if 0 in arr: # check if we have at list one zero in list
  first_zero_index=arr.index(0)#find index of first zero in list
  flag=1;
 else:
   print("we have not at list one zero in list: ")
arr1=[]
if flag==1:
 for i in arr[first_zero_index:]:
   arr1.append(i)
print (arr1)
```

Результат

```
our random list:  [-2, 3, 5, -5, -6, -4, 4, 1, 0, -3, -1, 2]
[0, -3, -1, 2]
```

**Приклад 7.6**. Нехай задано список-матриця цілих випадкових чисел (додатних та від'ємних) [[$a_{11}$, ..., $a_{1n}$], ..., [$a_{m1}$, ..., $a_{mn}$]]. Написати програму, яка

визначить список мінімальних елементів кожного рядка (результат записати в інший список). Вивести на екран початкову матрицю та отриманий список.

```
import random
n=3 # кількість стовпчиків
m=4 # кількість рядків
x=[0]*m
for i in range(m):
    x[i]=[0]*n
y=[0]*m; i=0
while i<m :
j=0; k=0
while j < n :
     x[i][j] = random.randint(-5,5)
     j+=1
y[i] = min(x[i][0:m])
k+=1; i+= 1
print('x= ', x); print('min = ', y)
```

## Теоретична частина

## 1. СПИСКИ

Список − колекція інших об'єктів, змінюваний об'єкт, вони можуть бути вкладеними, збільшуватися і зменшуватися, містити об'єкти будь-яких типів. Завдяки спискам можна створювати і обробляти в своїх сценаріях структури даних будь-якого ступеня складності. Нижче наводяться основні властивості списків, списки в мові Python − це:

1. *Впорядковані колекції об'єктів довільних типів.*

2. *Доступ до елементів за зсувом*. Можна використовувати операцію індексування для отримання окремих об'єктів зі списку за їхнім зсувом.

3. *Змінна довжина, гетерогенність і довільна кількість рівнів вкладеності*. Списки можуть збільшуватися і зменшуватися безпосередньо (їх довжина може змінюватися), вони можуть містити не тільки односимвольні рядки, а й будь-які інші об'єкти (списки гетерогенні). Списки можуть містити інші складні об'єкти, вони підтримують можливість створення довільної кількості рівнів вкладеності, тому є можливість створювати зі списків списки списків.

4. *Відносяться до категорії змінних об'єктів.*

У табл. 6.1 наведено найбільш типові операції, які застосовують до списків. Коли список визначається виразом (літерально), його записують як послідовність об'єктів в квадратних дужках, розділених комами, наприклад:

```
x='123'; a=list(x); x=''.join(a); print(x)
>>1
```

Вкладені списки описують як вкладені послідовності квадратних дужок (рядок 3 у табл. 6.1), а порожні списки визначають як порожню пару квадратних дужок (рядок 1 у табл. 6.1).

Таблиця 6.1

*Літерали списків і операції*

| Операція | Інтерпретація |
|---|---|
| L = [] | Пустий список |
| L = [0, 1, 2, 3] | Чотири елемента з індексами 0..3 |
| L = [‘abc’, [‘def’, ghi’]] | Вкладені списки |
| L = list(‘spam’) | Створення списка із рядка |
| L = list(range(−4, 4)) | Створення списка із безперервної послідовності цілих чисел |
| L[i] | Індекс |
| L[i][j] | індекс індекса |
| L[i:j] | зріз |
| len(L) | довжина |
| L1 + L2 | Конкатенація |
| L * 3 | дублювання |
| for x in L: print(x) | Обхід у циклі, x – змінна для присвоювання |
| 3 in L | перевірка входження |
| | Методи: |
| L.append(4) | додавання елемента «4» у список |
| L.extend([5,6,7]) | додавання списка у список |
| L.insert(1, X) | додавання списку елементів у вказану позицію |
| | Методи: |
| L.index(1) | Визначення зсуву елемента за заданим значенням |
| L.count() | Підрахунок кількості елементів |
| L.sort() | Сортування |
| L.reverse() | Зміна порядку слідування елементів на зворотний |
| | Зменшення списку |
| del L[k] | видалення елемента |
| del L[i:j] | видалення групи елементів |
| L.pop() | видалення останнього елемента й повернення його значення |
| L.remove(2) | видалення елементі з визначеними значеннями |
| L[i:j] = [] | |
| L[i] = 1 | Присвоювання за індексом |
| L[i:j] = [4,5,6] | Присвоювання зрізу значень |
| L=[x**2 for x in range(5)] | Генератори списків |
| list(map(ord, ‘spam’)) | відображення |

Оскільки списки є послідовностями, вони, як і рядки, підтримують оператори + і * (для списків вони так само виконують операції конкатенації і повторення), а в результаті виходить новий список:

```
>>> len([1, 2, 3]) # Довжина
3
>>> [1, 2, 3] + [4, 5, 6] # Конкатенація
[1, 2, 3, 4, 5, 6]
>>> ['Ni!']*4 # Повторення
['Ni!', 'Ni!', 'Ni!', 'Ni!']
```

Не можна виконати операцію конкатенації для списку і рядка, якщо попередньо не перетворити список в рядок (використовуючи, наприклад, функцію *str* або оператор форматування %) або рядок в список (за допомогою вбудованої функцією *list*):

```
>>> str([1, 2]) + "34" # То же, что и "[1, 2]" + "34"
'[1, 2]34'
>>> [1, 2] + list("34") # То же, что и [1, 2] + ["3", "4"]
[1, 2, '3', '4']
```

*Методи списків.* Об'єкти списків в Python підтримують специфічні методи, багато з яких змінюють сам список безпосередньо:

```
>>> L.append('please')
# Виклик метода додавання елемента у кінець списка
>>> L
['eat', 'more', 'SPAM!', 'please']
>>> L.sort() # Сортування елементів списка ('S'<'e')
>>> L
['SPAM!', 'eat', 'more', 'please']
```

*Методи* – це функції, пов'язані з певним типом об'єктів.

Метод *append* додає один елемент (посилання на об'єкт) в кінець списку. На відміну від операції конкатенації, метод *append* приймає один об'єкт-список. За своєю дією вираз L.append (X) схожий на вираз L+[X], але в першому випадку змінюється сам список, а в другому – створюється новий список. На відміну від операції конкатенації (+), метод *append* не створює новий об'єкт, тому зазвичай він виконується швидше. Існує можливість імітувати роботу методу *append* за допомогою операції присвоювання зрізу: вираз L[len (L):]=[X] відповідає виклику L.append (X), а вираз L [:0] = [X] відповідає операції додавання в початок списку. В обох випадках видаляється порожній сегмент списку і вставляється елемент X, при цьому змінюється сам список L, так само швидко, як при використанні методу *append*.

Метод *sort* виконує перевпорядкуванне елементів в списку. За замовчуванням він використовує стандартні оператори порівняння мови Python (в даному випадку виконується порівнювання рядків) і виконує сортування в порядку зростання значень. Існує можливість змінити порядок сортування за допомогою іменованих аргументів – спеціальних синтаксичних конструкцій типу «= name == value», які використовують під час виклику функцій для передачі параметрів налаштування за їхніми іменами. Іменований аргумент *key* у виклику методу *sort* дозволяє визначити власну функцію порівняння, яка приймає один аргумент і повертає значення, яке буде використано в операції порівняння, а іменований аргумент *reverse* дозволяє виконати сортування не в порядку зростання, а в порядку убування:

```
>>> L = ['abc', 'ABD', 'aBe']
>>> L.sort() # Сортування з урахуванням регістру символів
>>> L
['ABD', 'aBe', 'abc']
>>> L = ['abc', 'ABD', 'aBe']
```

```
>>> L.sort(key=str.lower)
# Приведення символів до нижнього регістру
>>> L
['abc', 'ABD', 'aBe']
>>>
>>> L = ['abc', 'ABD', 'aBe']
>>> L.sort(key=str.lower, reverse=True)
# Змінює напрямок сортування
>>> L
['aBe', 'ABD', 'abc']
```

Методи *append* і *sort* змінюють сам об'єкт списку і не повертають список у вигляді результату (точніше кажучи, обидва методи повертають значення None). Якщо ви написали інструкцію L = L.append (X), ви не отримаєте змінене значення L (насправді ви взагалі втратите посилання на список) – використання атрибутів *append* і *sort*, призводить до зміни самого об'єкта, тому немає ніяких причин виконувати повторне присвоювання. Вбудована функція *sorted* здатна сортувати списки і будь-які інші послідовності, вона повертає новий список з результатом сортування (оригінальний список при цьому не змінюється):

```
>>> L = ['abc', 'ABD', 'aBe']
>>> sorted(L, key=str.lower, reverse=True)
# функція сортування
['aBe', 'ABD', 'abc']
>>> L = ['abc', 'ABD', 'aBe']
>>> sorted([x.lower() for x in L], reverse=True)
# елементи попередньо
['abe', 'abd', 'abc'] # зменюються
```

В останньому прикладі перед сортуванням за допомогою генератора списків виконується приведення символів до нижнього регістра, і значення елементів в отриманому списку відрізняються від значень елементів в оригінальному списку. В останньому прикладі виконується сортування тимчасового списку, створеного в процесі сортування. Іноді вбудована функція *sorted* може виявитися більш зручною, ніж метод *sort*.

Метод *reverse* змінює порядок проходження елементів в списку на зворотний, а методи *extend* і *pop* вставляють кілька елементів в кінець списку і видаляють елементи з кінця списку відповідно. Крім того, існує вбудована функція *reversed*, яка нагадує вбудовану функцію *sorted*, але її необхідно обгорнути в виклик функції *list*, тому що вона повертає ітератор:

```
>>> L=[1,2]; L.extend([3,4,5]) # Додавання елементів у кінець
списку
>>> L
[1, 2, 3, 4, 5]
>>> L.pop() # видаляє і повертає останній елемент списка
5
>>> L
[1, 2, 3, 4]
>>> L.reverse() # Змінює порядок слідування елементів на зворотний
>>> L
```

```
[4, 3, 2, 1]
>>> list(reversed(L))
# Вбудована функція сортування в зворотному порядку
[1, 2, 3, 4]
```

Інші методи списків дозволяють видаляти елементи з певними значеннями (*remove*), вставляти елементи у визначену позицію (*insert*), визначяти зсув елемента за заданим значенням (index) тощо:

```
>>> L = ['spam', 'eggs', 'ham']
>>> L.index('eggs') # індекс об'єкта
1
>>> L.insert(1, 'toast') # Вставка у потрібну позицію
>>> L
['spam', 'toast', 'eggs', 'ham']
>>> L.remove('eggs') # видалення елемента із визначеним значенням
>>> L
['spam', 'toast', 'ham']
>>> L.pop(1) # видалення елемента у вказаній позиції
'toast'
>>> L
['spam', 'ham']
```

Можна використати інструкцію *del* для видалення елемента або зрізу безпосередньо зі списка:

```
>>> L
['SPAM!', 'eat', 'more', 'please']
>>> del L[0] # видалення одного елемента списка
>>> L
['eat', 'more', 'please']
>>> del L[1:] # видалення цілого сегмента списка
>>> L # То же, что и L[1:] = []
['eat']
```

Можна видаляти зрізи списку, привласнюючи їм порожній список (L[*i:j*]=[]) – інтерпретатор спочатку видалить зріз, який визначається зліва від оператора =, а потім вставить порожній список. Присвоювання порожнього списку за індексом елемента призведе до збереження посилання на порожній список в цьому елементі, а не до його видалення:

```
>>> L = ['Already', 'got', 'one']; L[1:] = []
>>> L
['Already']
>>> L[0] = []
>>> L
[[]]
```

## 2. МАСИВИ

Для зберігання і обробки в програмах складних видів інформації використовують структурні типи. Їх утворюють шляхом об'єднання простих елементів даних (компонентів). Компоненти можуть бути при цьому однорідними або різнорідними. Необхідність у масивах виникає щоразу, коли в

пам'яті потрібно зберігати велику, але скінченну кількість однотипних впорядкованих даних.

*Масив* – це структура даних, яку можна розглядати як набір змінних однакового типу, що мають загальне ім'я.

Доступ до будь-якого елементу масиву здійснюється за його номером. У масиви можна об'єднати результати експериментів, списки прізвищ співробітників, різні складні структури даних.

У масиві дані різняться своїм порядковим номером (індексом). Якщо кожний елемент масиву визначається за допомогою одного номера, то такий масив називається *одновимірним*, якщо за двома — то *двовимірним*.

*Двовимірний масив* — це таблиця з рядків і стовпчиків. У таблицях перший номер вказує на рядок, а другий — на положення елемента в рядку. Усі рядки таблиці мають однакову довжину.

*Одновимірний масив* (лінійна таблиця) може бути набором чисел, сукупністю символьних даних чи елементів іншої природи (навіть масив масивів). Так само, як і в послідовності, в одновимірному масиві можна вказати елемент з конкретним номером, наприклад a5, або записати загальний вигляд елемента, використовуючи як індекс змінну *i*, вказуючи діапазон її зміни: *a*[*i*], *i*=1, 2, ... , *n*. Для позначення окремої компоненти до імені масиву додається індекс, який і виділяє потрібну компоненту (наприклад, $a_1$, …, $a_{50}$).

Найменший індекс називається *нижньою межею*, найбільший – *верхньою межею*, а число елементів – *розміром масиву*. Розмір масиву фіксується при описі і в процесі виконання програми не змінюється. Індекси можна обчислювати. Найчастіше в якості типу індексу використовується обмежений цілий тип.

Базові типи мови Python підтримують можливість створення вкладених конструкцій довільної глибини і в будь-яких комбінаціях (наприклад, зображення матриць, або «багатовимірних масивів»). Це можна зробити за допомогою списку, що містить вкладені списки:

```
>>> M = [[1, 2, 3], # Матриця 3x3 у вигляді вкладених списків
[4, 5, 6], # Вираз в квадратних дужках може
[7, 8, 9]] # займати декілька рядків
>>> M
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Тут реалізовано список, який містить три інших списків. В результаті отримано матрицю чисел 3×3. Звернутися до такої структури можна різними способами:

```
>>> M[1] # отримати рядок 2
[4, 5, 6]
>>> M[1][2]
# отримати рядок 2, а потім елемент 3 в цьому рядку
6
```

Один з найпростіших способів подання матриць (багатовимірних масивів) полягає у використанні вкладених списків. Нижче наводиться приклад

двовимірного масиву розміром 3х3, побудованого на базі списків на Python:

```
>>> matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> matrix[1]
[4, 5, 6]
>>> matrix[1][1]
5
>>> matrix[2][0]
7
>>> matrix = [[1, 2, 3],
... [4, 5, 6],
... [7, 8, 9]]
```

Нижче наведено приклад двовимірного масиву:

```
>>> a = [[1, 2, 3], [4, 5, 6]]
>>> for i in a
…    for j in i
…         print ("i=", i, "j=", j)
>>>i=[1, 2, 3] j=1
>>>i=[1, 2, 3] j=2
>>>i=[1, 2, 3] j=3
>>>i=[4, 5, 6] j=4
>>>i=[4, 5, 6] j=5
>>>i=[4, 5, 6] j=6
```

## 3. МОДУЛЬ RANDOM

Модуль *random* із стандартної бібліотеки також необхідно імпортувати. Цей модуль дозволяє отримати випадкові дійсні числа в діапазоні від 0 до 1, випадкові цілі числа в заданому діапазоні, послідовність випадкових елементів, виконати випадковий вибір (в тому числі і із списка), тощо:

```
>>> import random
>>> random.random()
0.44694718823781876
>>> random.randint(1, 10)
5
>>>  random.choice(['Life  of  Brian','Holy  Grail','Meaning  of
Life'])
'Life of Brian'
>>>  random.choice(['Life  of  Brian','Holy  Grail','Meaning  of
Life'])
'Holy Grail'
>>> random.choice([1, 2, 3, 4])
1
```

До складу модуля *random* входять такі функції:

1) *random.randin (a,b)* – випадкове ціле число від *a* до *b*;
2) *random.random*() – випадкове число з інтервалу [0, 1);
3) *random.choice*(*x*) – обирає випаджковий елемент послідовності;
4) *random.shuffle(x)* – перемішує елементи послідовності;
5) *random.uniform(a,b)* – випадкове дійсне число від *a* до *b*.