МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Кафедра "Захист інформації"

Програмування з використанням списків

МЕТОДИЧНІ ВКАЗІВКИ
до лабораторної роботи № 5
з курсу «Програмування скриптовими мовами»
для студентів спеціальності
«Кібербезпека»

Затверджено
на засіданні кафедри
"Захист інформації"
протокол № 01 від 29.08.2024 р.

Львів – 2024

Програмування з використанням списків: Методичні вказівки до лабораторної роботи № 5 з курсу «Програмування скриптовими мовами» для студентів спеціальності «Кібербезпека» / Укл. *Я. Р. Совин* – Львів: Національний університет "Львівська політехніка", 2024. – 21 с.

Укладач: Я. Р. Совин, канд. техн. наук, доцент

Відповідальний за випуск: В. Б. Дудикевич, д.т.н., професор

Рецензенти: А. Я. Горпенюк, канд. техн. наук, доцент
Ю. Я. Наконечний, канд. техн. наук, доцент

Мета роботи – ознайомитись з списками та їх можливостями у мові Python**.**

1. ТЕОРЕТИЧНІ ВІДОМОСТІ

Ми переходимо до вивчення більш складних вбудованих типів даних мови Python - *списків*, *кортежів*, *словників* та *множин* - це нумеровані набори об'єктів. Кожен елемент набору містить лише посилання на об'єкт - з цієї причини вони можуть містити об'єкти довільного типу даних і мати необмежену ступінь вкладеності. Позиція елемента в наборі задається індексом. Нумерація елементів починається з 0, а не з 1.

Списки (Lists) є впорядкованими *послідовностями* елементів, які можуть динамічно змінюватися (шляхом добавляння, заміни, перестановки чи вилучення елементів) та містити різнотипні елементи. Як і всі послідовності, вони підтримують звернення до елементу за індексом, отримання зрізу, конкатенацію (оператор +), повторення (оператор *), перевірку на входження (оператор *in*) і невходження (оператор *not in*).

Списки відносяться до *змінюваних* (mutable) типів даних. Це означає, що ми можемо не тільки отримати елемент за індексом, а й змінити його:

```
>>> arr = [1, 2, 3] # Створюємо список
>>> arr[0]          # Отримуємо елемент за індексом
1
>>> arr[0] = 50     # Змінюємо елемент за індексом
>>> arr
[50, 2, 3]
```

1.1. Створення списку

Створити список можна наступними способами:

♦ за допомогою функції *list([<Послідовність>])*. Функція дозволяє перетворити будь-яку послідовність в список. Якщо параметр не вказано, створюється порожній список:

```
>>> list()                    # Створюємо порожній список
[]
>>> list("String")            # Перетворимо рядок на список
['S', 't', 'r', 'i', 'n', 'g']
```

♦ вказавши всі елементи списку всередині квадратних дужок:

```
>>> arr = [1, "str", 3, "4"]
>>> arr
[1, 'str', 3, '4']
```

♦ заповнивши список поелементно за допомогою методу *append()*:

```
>>> arr = []          # Створюємо порожній список
>>> arr.append(1)     # Додаємо елемент1 (індекс 0)
>>> arr.append("str") # Додаємо елемент2 (індекс 1)
>>> arr
[1, 'str']
```

При створенні списку в змінній зберігається посилання на об'єкт, а не сам об’єкт (рис. 1):

```
>>> scores = [10, 9, 7, 4, 5]
```

```
>>> values = scores # Копіює посилання на список
>>> scores[3] = 10
>>> print(values[3])
10
```

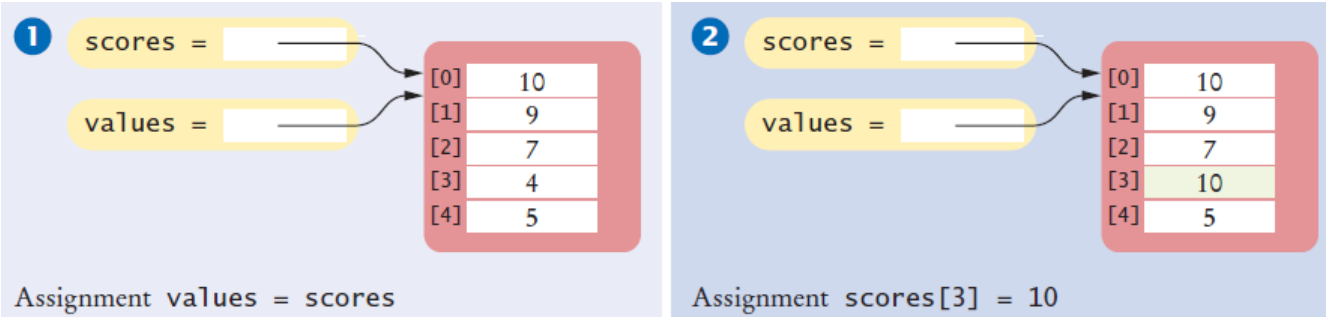

Рис. 1. Дві змінні-списки посилаються на той самий список

Це обов'язково слід враховувати при груповому присвоєнні. Групове присвоювання можна використовувати для чисел і рядків, але для списків цього робити не можна. Розглянемо приклад:

```
>>> x = y = [1, 2]     # Нібито створили два об'єкти
>>> x, y
([1, 2], [1, 2])
```

У цьому прикладі ми створили список з двох елементів і присвоїли значення змінним *x* і *y*. Тепер спробуємо змінити значення в змінній *y*:

```
>>> y[1] = 100         # Змінюємо другий елемент
>>> x, y               # Змінилося значення відразу в двох змінних
([1, 100], [1, 100])
```

Як видно з прикладу, зміна значення в змінній *y* привела також до зміни значення в змінній *x*. Таким чином, обидві змінні посилаються на один і той же об'єкт, а не на два різних об'єкти. Щоб отримати два об'єкти, необхідно виконати роздільне присвоювання:

```
>>> x, y = [1, 2], [1, 2]
>>> y[1] = 100 # Змінюємо другий елемент
>>> x, y
([1, 2], [1, 100])
```

Точно така ж ситуація виникає при використанні оператора повторення *. Наприклад, в наступній інструкції проводиться спроба створення двох вкладених списків за допомогою оператора *:

```
>>> arr = [ [] ] * 2 # Нібито створили два вкладених списки
>>> arr
[[], []]
>>> arr[0].append(5) # Додаємо елемент
>>> arr              # Змінилися два елементи
[[5], [5]]
```

Створювати вкладені списки слід за допомогою методу *append()* всередині циклу:

```
>>> arr = []
>>> for i in range(2): arr.append([])
>>> arr
[[], []]
>>> arr[0].append(5); arr
[[5], []]
```

Можна також скористатися генераторами списків:

```
>>> arr = [[] for i in range(2)]
>>> arr
[[], []]
>>> arr[0].append(5); arr
[[5], []]
```

Перевірити, чи посилаються дві змінні на один і той же об'єкт, дозволяє оператор *is*. Якщо змінні посилаються на один і той же об'єкт, оператор *is* повертає значення *True*:

```
>>> x = y = [1, 2] # Неправильно
>>> x is y         # Змінні містять посилання на один і той же список
True
>>> x, y = [1, 2], [1, 2] # Правильно
>>> x is y                # Це різні об'єкти
False
```

Але що ж робити, якщо необхідно створити копію списку? Перший спосіб полягає в застосуванні операції вилучення зрізу, другий у використанні функції *list()*, а третій - у виклику методу *copy()*:

```
>>> x = [1, 2, 3, 4, 5] # Створили список
>>> # Створюємо копію списку
>>> y = list(x) # або за допомогою зрізу: y = x[:]
>>>             # або викликом методу copy(): y = x.copy()
>>> y
[1, 2, 3, 4, 5]
>>> x is y # Оператор показує, що це різні об'єкти
False
>>> y[1] = 100 # Змінюємо другий елемент
>>> x, y # Змінився тільки список в змінній y
([1, 2, 3, 4, 5], [1, 100, 3, 4, 5])
```

На перший погляд може здатися, що ми отримали копію - оператор *is* показує, що це різні об'єкти, а зміна елемента торкнулася лише значення змінної *y*. У даному випадку ніби все нормально. Але проблема полягає в тому, що списки в мові Python можуть мати необмежену ступінь вкладеності. Розглянемо це на прикладі:

```
>>> x = [1, [2, 3, 4, 5]] # Створили вкладений список
>>> y = list(x)           # Нібито зробили копію списку
>>> x is y                # Різні об'єкти
False
>>> y[1][1] = 100         # Змінюємо елемент
>>> x, y                  # Зміна торкнулася змінної x!!!
([1, [2, 100, 4, 5]], [1, [2, 100, 4, 5]])
```

Тут ми створили список, в якому другий елемент є вкладеним списком, після чого за допомогою функції *list()* спробували створити копію списку. Як і в попередньому прикладі, оператор *is* показує, що це різні об'єкти, але подивіться на результат - зміна змінної *y* торкнулася і значення змінної *x*. Таким чином, функція *list()* і операція витягання зрізу створюють лише *поверхневу копію* списку. Щоб отримати повну копію списку, слід скористатися функцією *deepcopy()* з модуля *copy()*. При глибокому копіюванні створюється новий об’єкт і рекурсивно створюються копії всіх об’єктів, що містяться в оригіналі:

```
>>> import copy          # Підключаємо модуль copy
>>> x = [1, [2, 3, 4, 5]]
```

```
>>> y = copy.deepcopy(x) # Робимо повну копію списку
>>> y[1][1] = 100        # Змінюємо другий елемент
>>> x, y                 # Змінився тільки список в змінній y
([1, [2, 3, 4, 5]], [1, [2, 100, 4, 5]])
```

1.2. Операції над списками

Список є *послідовністю* елементів, кожен з яких має свою позицію або *індекс*. Звернення до елементів списку здійснюється за допомогою квадратних дужок, в яких вказується індекс елемента. Нумерація елементів списку починається з нуля (рис.):

```
>>> values = [32, 54, 67.5, 29, 35, 80, 115, 44.5, 100, 65]; values
[32, 54, 67.5, 29, 35, 80, 115, 44.5, 100, 65]
```

Виведемо всі елементи списку:

```
>>> arr = [1, "str", 3.2, "4"]
>>> arr[0], arr[1], arr[2], arr[3]
(1, 'str', 3.2, '4')
```

За допомогою позиційного присвоювання можна присвоїти значення елементів списку змінним. Кількість елементів праворуч і ліворуч від оператора = має збігатися, інакше буде виведено повідомлення про помилку:

```
>>> x, y, z = [1, 2, 3] # Позиційне присвоювання
>>> x, y, z
(1, 2, 3)
>>> x, y = [1, 2, 3] # Кількість елементів має збігатися
Traceback (most recent call last):
  File "<pyshell#80>", line 1, in <module>
    x, y = [1, 2, 3] # Кількість елементів має збігатися
ValueError: too many values to unpack (expected 2)
```

При позиційному присвоєнні перед однією з змінних зліва від оператора = можна вказати зірочку. У цій змінній буде зберігатися список, що складається з «зайвих» елементів. Якщо таких елементів немає, список буде порожнім:

```
>>> x, y, *z = [1, 2, 3]; x, y, z
(1, 2, [3])
>>> x, y, *z = [1, 2, 3, 4, 5]; x, y, z
(1, 2, [3, 4, 5])
>>> x, y, *z = [1, 2]; x, y, z
(1, 2, [])
>>> *x, y, z = [1, 2]; x, y, z
([], 1, 2)
>>> x, *y, z = [1, 2, 3, 4, 5]; x, y, z
(1, [2, 3, 4], 5)
>>> *z, = [1, 2, 3, 4, 5]; z
[1, 2, 3, 4, 5]
```

Так як нумерація елементів списку починається з 0, індекс останнього елемента буде на одиницю менше кількості елементів. Отримати кількість елементів списку дозволяє функція *len()*:

```
>>> arr = [1, 2, 3, 4, 5]
>>> len(arr)          # Отримуємо кількість елементів
5
>>> arr[len(arr) - 1] # Отримуємо останній елемент
5
```

В якості індексу можна вказати від'ємне значення. В цьому випадку зміщення буде відраховуватися від кінця списку, а точніше - щоб отримати позитивний індекс, значення віднімається із загальної кількості елементів списку:

```
>>> arr = [1, 2, 3, 4, 5]
>>> arr[-1], arr[len(arr) - 1] # Звернення до останнього елемента
(5, 5)
```

Так як списки відносяться до змінюваних типів даних, ми можемо змінити елемент за індексом:

```
>>> arr = [1, 2, 3, 4, 5]
>>> arr[0] = 600 # Зміна елемента за індексом
>>> arr
[600, 2, 3, 4, 5]
```

Крім того, списки підтримують операцію вилучення зрізу, яка повертає зазначений фрагмент списку. Формат операції:

```
[<Початок>:<Кінець>:<Крок>]
```

Всі параметри не є обов'язковими. Якщо параметр *<Початок>* не вказано, використовується значення 0. Якщо параметр *<Кінець>* не вказано, повертається фрагмент до кінця списку. Слід також зауважити, що елемент з індексом, зазначеним в цьому параметрі, не входить у фрагмент, що повертається. Якщо параметр *<Крок>* не вказано, використовується значення 1. В якості значення параметрів можна вказати негативні значення. Тепер розглянемо кілька прикладів:

♦ спочатку отримаємо поверхневу копію списку:

```
>>> arr = [1, 2, 3, 4, 5]
>>> m = arr[:]; m # Створюємо поверхневу копію
[1, 2, 3, 4, 5]
```

♦ потім виведемо символи у зворотному порядку:

```
>>> arr[::-1] # Крок -1
[5, 4, 3, 2, 1]
```

♦ виведемо список без першого і останнього елементів:

```
>>> arr[1:] # Без першого елемента
[2, 3, 4, 5]
>>> arr[:-1] # Без останнього елемента
[1, 2, 3, 4]
```

♦ отримаємо перші два елементи списку:

```
>>> arr[0:2] # Символ з індексом 2 не входить в діапазон
[1, 2]
```

♦ а так отримаємо останній елемент:

```
>>> arr[-1:] # Останній елемент списку
[5]
```

♦ і, нарешті, виведемо фрагмент від другого елементу до четвертого включно:

```
>>> arr[1:4] # Повертаються елементи з індексами 1, 2 і 3
[2, 3, 4]
```

За допомогою зрізу можна змінити фрагмент списку. Якщо зрізу привласнити порожній список, то елементи, що потрапили в зріз, будуть видалені:

```
>>> arr[1:3] = [6,7] # Змінюємо значення елементів з індексами 1 і 2
>>> arr
[1, 6, 7, 4, 5]
>>> arr[1:3] = []    # Видаляємо елементи з індексами 1 і 2
>>> arr
```

```
[1, 4, 5]
```

Об'єднати два списки в один список дозволяє оператор +. Результатом об'єднання буде новий список:

```
>>> arr1 = [1, 2, 3, 4, 5]
>>> arr2 = [6, 7, 8, 9]
>>> arr3 = arr1 + arr2
>>> arr3
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Замість оператора + можна використовувати оператор +=. Слід враховувати, що в цьому випадку елементи додаються до поточного списку:

```
>>> arr  = [1, 2, 3, 4, 5]
>>> arr += [6, 7, 8, 9]
>>> arr
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Крім розглянутих операцій, списки підтримують операцію повторення і перевірку на входження. Повторити список вказану кількість разів можна за допомогою оператора *, а виконати перевірку на входження елемента в список дозволяє оператор *in*:

```
>>> [1, 2, 3] * 3 # Операція повторення
[1, 2, 3, 1, 2, 3, 1, 2, 3]
>>> 2 in [1,2,3,4,5], 6 in [1,2,3,4,5] # Перевірка на входження
(True, False)
```

Оператор == використовується для порівняння чи містять два списки однакові елементи в однаковому порядку:

```
>>> [1, 4, 9] == [1, 4, 9], [1, 4, 9] == [4, 9, 1]
(True, False)
```

1.3. Багатомірні списки

Будь-який елемент списку може містити об'єкт довільного типу. Наприклад, елемент списку може бути числом, рядком, списком, кортежем, словником і т. д. Створити вкладений список можна, наприклад, так:

```
>>> arr = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Вираз всередині дужок може розташовуватися на декількох рядках. Отже, попередній приклад можна записати інакше:

```
>>> arr = [
               [1, 2, 3],
               [4, 5, 6],
               [7, 8, 9]
          ]
```

Щоб отримати значення елемента у вкладеному списку, слід вказати два індекси:

```
>>> arr[1][1]
5
```

Елементи вкладеного списку також можуть мати елементи довільного типу. Кількість вкладень не обмежена, і ми можемо створити об'єкт будь-якого ступеня складності. В цьому випадку для доступу до елементів вказується кілька індексів поспіль:

```
>>> arr = [ [1, ["a", "b"], 3], [4, 5, 6], [7, 8, 9] ]
>>> arr[0][1][0]
```

```
'a'
>>> arr = [ [1, { "a": 10, "b": ["s", 5] } ] ]
>>> arr[0][1]["b"][0]
's'
```

1.4. Перебір елементів списку

Перебрати всі елементи списку можна за допомогою циклу *for*:

```
>>> arr = [1, 2, 3, 4, 5]
>>> for i in arr: print(i, end = " ")
1 2 3 4 5
```

Слід зауважити, що змінну *i* всередині циклу можна змінити, але якщо вона посилається на незмінний тип даних (наприклад, на число або рядок), це не відіб'ється на початковому списку

```
>>> arr = [1, 2, 3, 4] # Елементи мають незмінний тип (число)
>>> for i in arr: i += 10
>>> arr # Список не змінився
[1, 2, 3, 4]
>>> arr = [[1, 2], [3, 4]] # Елементи мають змінний тип (список)
>>> for i in arr: i[0] += 10
>>> arr # Список змінився
[[11, 2], [13, 4]]
```

Для генерації індексів елементів можна скористатися функцією *range()*, яка повертає об'єкт-діапазон з підтримкою ітерації. За допомогою такого діапазону всередині циклу *for* можна отримати поточний індекс. Функція *range()* має такий вигляд:

```
range([<Початок>,]<Кінець>[,<Крок>])
```

Перший параметр задає початкове значення. Якщо він не вказаний, використовується значення 0. У другому параметрі вказується кінцеве значення. Слід зауважити, що це значення не входить в діапазон значень, що повертається. Якщо параметр *<Крок>* не вказано, використовується значення 1. Для прикладу помножимо кожен елемент списку на 2:

```
>>> arr = [1, 2, 3, 4]
>>> for i in range(len(arr)):
     arr[i] *= 2
>>> print(arr) # Результат виконання: [2, 4, 6, 8]
```

Можна також скористатися функцією *enumerate(<Об'єкт>[,start = 0])*, яка на кожній ітерації циклу *for* повертає кортеж з індексу і значення поточного елемента списку. Помножимо кожен елемент списку на 2:

```
>>> arr = [1, 2, 3, 4]
>>> for i, elem in enumerate(arr):
     arr[i] *= 2
>>> print(arr) # Результат виконання: [2, 4, 6, 8]
```

Крім того, перебрати елементи можна за допомогою циклу *while*, але потрібно пам'ятати, що він виконується повільніше циклу *for*. Для прикладу помножимо кожен елемент списку на 2, використовуючи цикл *while*:

```
>>> arr = [1, 2, 3, 4]
>>> i, c = 0, len(arr)
>>> while i < c:
     arr[i] *= 2
```

```
    i += 1
>>> print(arr) # Результат виконання: [2, 4, 6, 8]
```

1.5. Генератори списків і вирази-генератори

У попередньому розділі ми змінювали елементи списку наступним чином:

```
>>> arr = [1, 2, 3, 4]
>>> for i in range(len(arr)):
        arr[i] *= 2
>>> print(arr) # Результат виконання: [2, 4, 6, 8]
```

За допомогою *генераторів списків* або *спискових включень* (list comprehension) той самий код можна записати більш компактно, до того ж генератори списків працюють швидше циклу *for*. Проте замість зміни вихідного списку повертається новий список:

```
>>> arr = [1, 2, 3, 4]
>>> arr = [i * 2 for i in arr]
>>> print(arr) # Результат виконання: [2, 4, 6, 8]
```

Як видно з прикладу, ми помістили цикл *for* всередині квадратних дужок, а також змінили порядок слідування параметрів, - інструкція, яка виконується всередині циклу, знаходиться перед циклом. Зверніть увагу і на те, що вираз всередині циклу не містить оператора присвоювання, - на кожній ітерації циклу буде генеруватися новий елемент, якому неявним чином присвоюється результат виконання виразу всередині циклу. В результаті буде створений новий список, що містить змінені значення елементів вихідного списку.

Ще приклади:

```
>>> s = [s * 3 for s in "Python"]
>>> s
['PPP', 'yyy', 'ttt', 'hhh', 'ooo', 'nnn']
>>> arr = [i for i in range(1, 15)]
>>> arr
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
>>> squares = [x ** 2 for x in range(10)]
>>> squares
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
>>> lst = ['ABC', 23.4, 7, 'Wow', 16, 'xyz', 10]
>>> [x for x in lst if type(x) != str] # Відфільтровуємо рядки
[23.4, 7, 16, 10]
>>> [x for x in lst if type(x) == str] # Відфільтровуємо не рядки
['ABC', 'Wow', 'xyz']
```

Генератори списків можуть мати складну структуру - наприклад, складатися з декількох вкладених циклів *for* і (або) містити оператор розгалуження *if* після циклу.

```
>>> arr = [-4, -2, 0, 2, 4]
>>> [x for x in arr if x >= 0] # Відфільтруємо від’ємні числа
[0, 2, 4]
```

Ще для прикладу отримаємо парні елементи списку і помножимо їх на 10:

```
>>> arr = [1, 2, 3, 4]
>>> arr = [i * 10 for i in arr if i % 2 == 0]
>>> print(arr) # Результат виконання: [20, 40]
```

Цей код еквівалентний коду:

```
arr = []
for i in [1, 2, 3, 4]:
    if i% 2 == 0:           # Якщо число парне
        arr.append (i * 10) # Додаємо елемент
print (arr)                 # Результат виконання: [20, 40]
```

Ускладнимо наш приклад - отримаємо парні елементи вкладеного списку і помножимо їх на 10:

```
>>> arr = [[1, 2], [3, 4], [5, 6]]
>>> arr = [j * 10 for i in arr for j in i if j % 2 == 0]
>>> print(arr) # Результат виконання: [20, 40, 60]
```

Цей код еквівалентний коду:

```
arr = []
for i in [[1, 2], [3, 4], [5, 6]]:
    for j in i:
        if j% 2 = 0:            # Якщо число парне
            arr.append (j * 10) # Додаємо елемент
print (arr) # Результат виконання: [20, 40, 60]
```

Генератори теж можуть бути вкладені:

```
>>> arr = [[y * 3 for y in range(x)] for x in range(5)]
>>> arr
[[], [0], [0, 3], [0, 3, 6], [0, 3, 6, 9]]
```

У генераторах можна застосовувати функції до всіх елементів:

```
>>> arr = [-4, -2, 0, 2, 4]
>>> [abs(x) for x in arr]
[4, 2, 0, 2, 4]
```

Якщо вираз розмістити всередині не квадратних, а круглих дужок, то буде повертатися не список, а ітератор. Такі конструкції називаються *виразами-генераторами*. Як приклад підсумуємо парні числа в списку:

```
>>> arr = [1, 4, 12, 45, 10]
>>> sum((i for i in arr if i % 2 == 0))
26
```

1.6. Функції та методи для роботи зі списками

1.6.1. Додавання і видалення елементів списку

Для додавання і видалення елементів списку використовуються наступні методи:

♦ *append(<Об'єкт>)* – додає один об'єкт в кінець списку. Метод змінює поточний список і нічого не повертає:

```
>>> arr = [1, 2, 3]
>>> arr.append(4); arr # Додаємо число
[1, 2, 3, 4]
>>> arr.append([5, 6]); arr # Додаємо список
[1, 2, 3, 4, [5, 6]]
>>> arr.append((7, 8)); arr # Додаємо кортеж
[1, 2, 3, 4, [5, 6], (7, 8)]
>>> friends = []               # Крок 1
>>> friends.append("Harry")    # Крок 2
>>> friends.append("Emily")    # Крок 3
>>> friends.append("Bob")      # Крок 3
```

```
>>> friends.append("Cari")    # Крок 3
```

♦ *extend(<Послідовність>)* – додає елементи послідовності в кінець списку. Метод змінює поточний список і нічого не повертає:

```
>>> arr = [1, 2, 3]
>>> arr.extend([4, 5, 6]) # Додаємо список
>>> arr.extend((7, 8, 9)) # Додаємо кортеж
>>> arr.extend("abc") # Додаємо букви з рядка
>>> arr
[1, 2, 3, 4, 5, 6, 7, 8, 9, 'a', 'b', 'c']
```

Додати кілька елементів можна за допомогою операції конкатенації або оператора +=:

```
>>> arr = [1, 2, 3]
>>> arr + [4, 5, 6] # Повертає новий список
[1, 2, 3, 4, 5, 6]
>>> arr += [4, 5, 6] # Змінює поточний список
>>> arr
[1, 2, 3, 4, 5, 6]
```

Крім того, можна скористатися операцією присвоювання значення зрізу:

```
>>> arr = [1, 2, 3]
>>> arr[len(arr):] = [4, 5, 6] # Змінює поточний список
>>> arr
[1, 2, 3, 4, 5, 6]
```

♦ *insert(<Індекс>,<Об'єкт>)* – додає один об'єкт в зазначену позицію. Решта елементів зміщуються. Метод змінює поточний список і нічого не повертає:

```
>>> arr = [1, 2, 3]
>>> arr.insert(0, 0); arr # Вставляємо 0 в початок списку
[0, 1, 2, 3]
>>> arr.insert(-1, 20); arr # Можна вказати негативні числа
[0, 1, 2, 20, 3]
>>> arr.insert(2, 100); arr # Вставляємо 100 в позицію 2
[0, 1, 100, 2, 20, 3]
>>> arr.insert(10, [4, 5]); arr # Додаємо список
[0, 1, 100, 2, 20, 3, [4, 5]]
```

Метод *insert()* створений виключно для зручності, бо все що він робить можна зробити шляхом присвоювання, оскільки *list.insert(n, elem)* еквівалентно *list[n:n] = [elem]*

```
>>> arr = [1, 2, 3]
>>> arr[0:0] = [0]
>>> arr
[0, 1, 2, 3]
```

Метод *insert()* дозволяє додати тільки один об'єкт. Щоб додати кілька об'єктів, можна скористатися операцією присвоювання значення зрізу. Додамо кілька елементів в початок списку:

```
>>> arr = [1, 2, 3]
>>> arr[:0] = [-2, -1, 0]
>>> arr
[-2, -1, 0, 1, 2, 3]
```

♦ *pop([<Індекс>])* – видаляє елемент, розташований за вказаним індексом, і повертає його. Якщо індекс не вказано, то видаляє і повертає останній елемент списку. Якщо елемента з вказаним індексом немає, або список порожній, генерується виключення *IndexError*:

```
>>> arr = [1, 2, 3, 4, 5]
>>> arr.pop() # Видаляємо останній елемент списку
5
>>> arr # Список змінився
[1, 2, 3, 4]
>>> arr.pop(0) # Видаляємо перший елемент списку
1
>>> arr # Список змінився
[2, 3, 4]
```

Видалити елемент списку дозволяє також оператор *del*:

```
>>> arr = [1, 2, 3, 4, 5]
>>> del arr[4]; arr # Видаляємо останній елемент списку
[1, 2, 3, 4]
>>> del arr[: 2]; arr # Видаляємо перший і другий елементи
[3, 4]
```

Оператор *del list[n]* еквівалентний *list[n:n+1] = []*:

```
>>> arr = [1, 2, 3, 4, 5]
>>> arr[4:5] = []; arr
[1, 2, 3, 4]
```

♦ *remove(<Значення>)* – видаляє перший елемент, що містить вказане значення. Якщо елемент не знайдений, генерується виключення *valueError*. Метод змінює поточний список і нічого не повертає:

```
>>> arr = [1, 2, 3, 1, 1]
>>> arr.remove(1) # Видаляє тільки перший елемент
>>> arr
[2, 3, 1, 1]
>>> arr.remove(5) # Такого елемента немає
Traceback (most recent call last):
  File "<pyshell#224>", line 1, in <module>
    arr.remove(5) # Такого елемента немає
ValueError: list.remove(x): x not in list
```

♦ *clear()* – видаляє всі елементи списку, очищаючи його. Ніякого результату при цьому не повертається:

```
>>> arr = [1, 2, 3, 1, 1]
>>> arr.clear()
>>> arr
[]
```

Якщо необхідно видалити всі повторювані елементи списку, то можна перетворити список в множину, а потім множину назад перетворити в список. Зверніть увагу на те, що список має містити тільки незмінні об'єкти (наприклад, числа, рядки або кортежі). В іншому випадку генерується виключення *TypeError*:

```
>>> arr = [1, 2, 3, 1, 1, 2, 2, 3, 3]
>>> s = set(arr) # Перетворимо список в множину
>>> s
{1, 2, 3}
>>> arr = list(s) # Перетворимо множину в список
>>> arr # Все повтори були видалені
[1, 2, 3]
```

1.6.2. Пошук елемента в списку

Виконати перевірку на входження елемента в список дозволяє оператор *in*: якщо елемент входить в список, то повертається значення *True*, в іншому випадку - *False*. Аналогічний оператор *not in* виконує перевірку на невходження елемента в список: якщо елемент відсутній у списку, повертається *True*, в іншому випадку - *False*:

```
>>> 2 in [1, 2, 3, 4], 6 in [1, 2, 3, 4, 5] # Перевірка на входження
(True, False)
>>> 2 not in [1,2,3,4], 6 not in [1,2,3,4,5] # Перевірка невходження
(False, True)
```

Проте, обидва ці оператора не дають ніякої інформації про місцезнаходження елемента всередині списку. Щоб дізнатися індекс елементу всередині списку, слід скористатися методом *index()*. Формат методу:

```
index(<Значення>[,<Початок>[,<Кінець>]])
```

Метод *index()* повертає індекс елемента, що має вказане значення. якщо значення не входить в список, то генерується виключення *ValueError*. Якщо другий і третій параметри не вказані, пошук буде проводитися з початку і до кінця списку:

```
>>> arr = [1, 2, 1, 2, 1]
>>> arr.index(1), arr.index(2)
(0, 1)
>>> arr.index(1, 1), arr.index(1, 3, 5)
(2, 4)
>>> arr.index(3)
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    arr.index(3)
ValueError: 3 is not in list
```

Дізнатися загальну кількість елементів із зазначеним значенням дозволяє метод *count(<Значення>)*. Якщо елемент не входить в список, повертається значення 0:

```
>>> arr = [1, 2, 1, 2, 1]
>>> arr.count(1), arr.count(2)
(3, 2)
>>> arr.count(3) # Елемент не входить в список
0
```

За допомогою функцій *max()* і *min()* можна дізнатися максимальне і мінімальне значення з усіх, що входять до списку, відповідно:

```
>>> arr = [1, 2, 3, 4, 5]
>>> max(arr), min(arr)
(5, 1)
```

Функція *any(<Послідовність>)* повертає значення *True*, якщо в послідовності існує хоча б один елемент, який в логічному контексті повертає значення *True*. Якщо послідовність не містить таких елементів, повертається значення *False*:

```
>>> any([0, None]), any([0, None, 1]), any([])
(False, True, False)
```

Функція *all(<Послідовність>)* повертає значення *True*, якщо всі елементи послідовності в логічному контексті повертають значення *True* або послідовність не містить елементів:

```
>>> all([0, None]), all([0, None, 1]), all([ ]), all(["str", 10])
(False, False, True, True)
```

1.6.3. Реверс, перемішування і випадковий вибір з списку

Метод *reverse()* змінює порядок проходження елементів списку на протилежний. Метод змінює поточний список і нічого не повертає:

```
>>> arr = [1, 2, 3, 4, 5]
>>> arr.reverse () # Змінюється поточний список
>>> arr
[5, 4, 3, 2, 1]
```

Якщо необхідно змінити порядок проходження і отримати новий список, слід скористатися функцією *reversed(<Послідовність>)*. Вона повертає ітератор, який можна перетворити в список за допомогою функції *list()* або генератора списків:

```
>>> arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> reversed(arr)
<list_reverseiterator object at 0x0000000002CB3550>
>>> list(reversed(arr)) # Використання функції list()
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
>>> [i for i in reversed(arr)] # Використання генератора списків
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

Функція *shuffle(<Список>[,<Число від 0.0 до 1.0>])* з модуля *random* перемішує список випадковим чином. Функція перемішує сам список і нічого не повертає. Якщо другий параметр не вказано, використовується значення, яке повертається функцією *random()*:

```
>>> import random # Підключаємо модуль random
>>> arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> random.shuffle(arr) # Перемішуємо список випадковим чином
>>> arr
[9, 1, 5, 6, 7, 8, 10, 2, 4, 3]
```

Отримати елементи зі списку випадковим чином дозволяють наступні функції з модуля *random*:

♦ *choice(<Послідовність>)* - повертає випадковий елемент з будь-якої послідовності (рядки, списки, кортежі):

```
>>> import random # Підключаємо модуль random
>>> random.choice (["s","t","r"]) # Список
's'
```

♦ *sample(<Послідовність>,<Кількість елементів>)* - повертає список з зазначеної кількості елементів. У цей список потраплять елементи з послідовності, вибрані випадковим чином. Як послідовності можна вказати будь-які об'єкти, що підтримують ітерації:

```
>>> arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> random.sample(arr, 2)
[7, 3]
```

1.6.4. Сортування списку

Відсортувати список дозволяє метод *sort()*. Він має такий вигляд:

```
sort([key = None][,reverse = False])
```

Всі параметри не є обов'язковими. Метод змінює поточний список і нічого не повертає. Відсортуємо список по зростанню з параметрами за замовчуванням:

```
>>> arr = [2, 7, 10, 4, 6, 8, 9, 3, 1, 5]
>>> arr.sort() # Змінює поточний список
>>> arr
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Щоб відсортувати список за спаданням, слід в параметрі *reverse* вказати значення *True*:

```
>>> arr = [2, 7, 10, 4, 6, 8, 9, 3, 1, 5]
>>> arr.sort(reverse = True) # Сортування за спаданням
>>> arr
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

У параметрі *key* методу *sort()* можна вказати функцію, що виконує будь-яку дію над кожним елементом списку. В якості єдиного параметра вона повинна приймати значення чергового елемента списку, а в якості результата – повертати результат дій над ним. Цей результат буде брати участь в процесі сортування, але значення самих елементів списку не зміняться.

Метод *sort()* сортує сам список і не повертає ніякого значення. В деяких випадках необхідно отримати відсортований список, а поточний список залишити без змін. Для цього слід скористатися функцією *sorted()*. Функція має наступний формат:

```
sorted(<Послідовність>[,key = None][,reverse = False])
```

У першому параметрі вказується список, який необхідно відсортувати. Решта параметрів еквівалентні параметрам методу *sort()*. Ось приклад використання функції *sorted()*:

```
>>> arr = [2, 7, 10, 4, 6, 8, 9, 3, 1, 5]
>>> sorted(arr) # Повертає новий список!
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> sorted(arr, reverse = True) # Повертає новий список!
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

1.5.6. Заповнення списку числами

Створити список, що містить діапазон чисел, можна за допомогою функції *range()*. Вона повертає діапазон, який перетворюється в список викликом функції *list()*. Функція *range()* має такий вигляд:

```
range([<Початок>,]<Кінець>[,<Крок>])
```

Перший параметр задає початкове значення, а якщо він не вказаний, використовується значення 0. У другому параметрі вказується кінцеве значення (не входить в діапазон, що повертається). Якщо параметр *<Крок>* не вказаний, використовується значення 1. Як приклад заповнимо список числами від 0 до 10:

```
>>> list(range(11))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Створимо список, що складається з діапазону чисел від 1 до 15:

```
>>> list(range(1, 16))
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
```

Тепер змінимо порядок проходження чисел на протилежний:

```
>>> list(range(15, 0, -1))
[15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

Якщо необхідно отримати список з випадковими числами (або випадковими елементами з іншого списку), то слід скористатися функцією *sample(<Послідовність>,<Кількість елементів>)* з модуля *random*:

```
>>> import random
>>> arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> random.sample(arr, 3)
[4, 10, 7]
>>> random.sample(range(300), 5)
[99, 283, 34, 94, 285]
```

У табл. 1 підсумовано основні розглянуті вище операції з списками.

Таблиця 1

Найпоширеніші оператори та функції роботи зі списками

Операція	Опис	Приклад
[]	Створює пустий список	*x = []*
[elem1, elem2, ... , elemN]	Створює список з заданих елементів	*x = [1, 2, 3, "abc"]*
lst[idx]	Елемент списку *lst* з індексом *idx*	*x[5] = 17*
lst[from:to]	Створює зріз у вигляді елементів списку *lst* починаючи з позиції *from* до позиції *to* не включаючи її	*y = x[3:7]* *x[0:3] = [3, 4, 5]*
lst[from:to] = []	Видаляє елементи списку *lst* починаючи з позиції *from* до позиції *to* не включаючи її	*x[1:3] = []*
len(lst)	Повертає кількість елементів у списку *lst*	*len(x)*
lst1 + lst2	Створює новий список об'єднанням елементів у обох списках *lst1* і *lst2*	*x = [1, 2] + [3, 4]*
*lst * num*	Створює новий список повторенням елементів у списку *lst num* раз	*x = [1, 2] * 3*
del lst[idx]	Видаляє елемент з індеком *idx* з списку *lst*	*del x[0]*
del lst[from:to]	Видаляє елементи списку *lst* починаючи з позиції *from* до позиції *to* не включаючи її. Аналогічно *lst[from:to] = []*	*del x[1:3]*
sum(lst)	Обчислює суму значень в списку	*sum([1, 3, 7])*
max(lst)	Повертає максимальне значення в списку	*max([1, 3, 7])*
min(lst)	Повертає мінімальне значення в списку	*min([1, 3, 7])*
lst1 == lst2	Тестує, чи два списки містять однакові елементи в однаковому порядку	*[1, 2] == [1, 2]*
elem in lst	Повідомляє, чи присутній елемент *elem* в списку *lst*	*"y" in x*

lst.append(elem)	Додає один елемент *elem* в кінець списку *lst.* Аналогічно *lst[len(lst):len(lst)]=[elem]*	*x.append("y")*
lst1.extend(lst2)	Додає список *lst2* в кінець списку *lst1.* Аналогічно *lst[len(lst):len(lst)]=elem*	*x.extend([3, 4])*
lst.insert(idx, elem)	Вставляє елемент *elem* в довільну позицію з індексом *idx* списку *lst.* Аналогічно *lst[idx:idx]=[elem]*, при *idx >= 0*	*x.insert(2, "y")*
lst.index(elem)	Повертає індекс елементу *elem* в списку *lst*	*x.index("y")*
lst.pop()	Видаляє останній елемент списку *lst*	*x.pop()*
lst.pop(idx)	Видаляє елемент списку *lst* з індексом *idx.* Аналогічно *x = lst[idx], del lst[idx]; return x*	*x.pop(3)*
lst.remove(elem)	Видаляє елементу *elem* з списку *lst.* Аналогічно *del lst[lst.index(elem)]*	*x.remove("y")*
lst.sort()	Сортує елементи в списку *lst* від меншого до більшого	*x.sort()*
lst.reverse()	Переставляє елементи в списку *lst* у зворотньому порядку	*x.reverse()*
lst.count(elem)	Підраховує кількість входжень значення *elem* в списку *lst*	*x.count("y")*
lst.copy()	Повертає поверхневу копію списку *lst*	*y = x.copy()*
lst.clear()	Видаляє всі елементи списку	*x.clear()*

2. ЗАВДАННЯ

2.1. Домашня підготовка до роботи

1. Вивчити теоретичний матеріал.

2.2. Виконати в лабораторії

1. Написати програму яка створює і виводить список, що містить послідовність цілих чисел з *n* елементів задану формулою згідно таблиці 2 Список створити двома способами: з допомогою циклу та генератору списків. Для створеного списку:
    a. Виведіть елементи з індексами від 3 до 5.
    b. Замініть перший елемент останнім.
    c. Об’єднайте початковий список і отриманий на кроці b.
    d. Додайте до списку ще три елементи зі значеннями перших трьох.
    e. Виведіть максимальне і мінімальне значення в списку.
    f. Видаліть всі елементи менші за середньоарифметичне значення.

Табл. 2

Варіант	Формула	Кількість елементів списку	Початкове значення n	Крок
1	7n + 3	10	2	1
2	n^2	15	-2	2
3	n^3+ n	20	0	1
4	5n + 7	12	4	3
5	3n + 1	8	2	6
6	n^2 + 3n + 5	17	1	4
7	4n + 5	11	-5	1
8	n^2 + 7n	16	4	4
9	n^2+ 3n + 1	22	1	3
10	9n + 4	13	0	1
11	7n - 2	10	4	1
12	3n^2 + 6n + 1	18	7	8
13	9n + 1	11	-5	1
14	5n^2 + 6n	16	2	4
15	2n^2+ 9n - 1	22	5	1
16	2n + 11	13	4	5
17	5n + 6	10	4	3
18	4n^2 + 8n + 2	18	7	4
19	4n + 11	11	-5	1
20	3n^2 + 2n + 1	15	2	3
21	5n^2+ 7n - 2	22	1	2
22	3n + 11	14	0	2
23	6n + 7	12	4	7
24	2n^2 + 8n + 5	20	3	5
25	3n^2 + 4n + 1	14	2	4

2. Написати програму яка створює і виводить двовимірний список з 5 елементів (це потрібно, щоб при запуску програми кожен раз не вводити початкові дані, тобто в базі даних має бути вже 5 записів). Програма не повинна використовувати функції чи класи. Кожен елемент списку представляє собою список, який містить опис атрибутів об’єкту згідно таблиці 3. Організуйте діалоговий режим із вводом з клавіатури, який дозволяє робити такі операції:
    a. Вивести весь список.
    b. Додавати елементи до списку.
    c. Відсортувати список за заданим атрибутом.
    d. Видаляти елементи за заданим атрибутом.
    e. Видаляти елемент за заданим індексом.
    f. Виводити всі елементи за заданим атрибутом.

Табл. 3

Варіант	Об’єкт	Атрибути

1	Автомобіль	Виробник, модель, рік випуску, пробіг, макс. швидкість
2	Книга	Автор, назва, видавництво, рік видання, кількість сторінок
3	Пасажирський літак	Виробник, модель, рік випуску, кількість пасажирів, макс. швидкість
4	Студент	ПІБ, група, рік вступу, середній бал
5	Користувач	Логін, пароль, телефон, е-мейл, кількість авторизацій
6	Фільм	Назва, режисер, рік випуску, бюджет, тривалість, розмір файлу
7	Пиво	Назва, виробник, міцність, ціна, термін зберігання в днях
8	Працівник	Назва фірми, посада, телефон, е-мейл, оклад
9	Пісня	Виконавець, назва, альбом, тривалість, розмір файлу
10	Смартфон	Виробник, модель, ціна, ємність батареї, обсяг пам’яті
11	Процесор	Виробник, модель, тактова частота, ціна, розсіювана потужність
12	Квартира	Власник, адреса, поверх, площа, кількість кімнат
13	Собака	Порода, кличка, вік, вага
14	Ноутбук	Виробник, процесор, ціна, діагональ, час роботи від акумулятора
15	Країна	Назва, столиця, населення, площа, ВВП
16	Нобелівський лауреат	Прізвище, рік народження, країна, рік вручення, галузь,
17	Купюра	Валюта, номінал, рік випуску, курс до долара
18	Дисципліна	Назва, викладач, семестр, балів за поточний контроль, балів за екзамен
19	Річка	Назва, континент, довжина, басейн, середньорічний стік
20	Місто	Назва, країна, населення, рік заснування, площа

Вивід інформації повинен здійснюватися відформатований по стовпцях. Приклад:

```
Меню
            1 - Друк списку
            2 - Додати елемент до списку
            3 - Відсортувати список за заданим атрибутом
            4 - Видалити елемент за заданим індексом
            5 - Видалити елемент за заданим атрибутом
            6 - Вивести елементи із заданим атрибутом
            7 - Вихід

Виберіть операцію натиснувши відповідну цифру: 1

       Виробник          Модель        Рік випуску     Пробіг, км    Макс. швидкість, км/год
 1.        Audi             A5              2012          300000              270
 2.         BMW             X5              2014          200000              290
 3.      Toyota          Camry              2017          100000              280
 4.  Volkswagen       Passat B5              2016          150000              250
 5.       Skoda         Octavia              2013          250000              275
```

Під час операцій вивід інформації теж повинен здійснюватися відформатований по стовпцях. Приклад:

```
Меню
             1 - Друк списку
             2 - Додати елемент до списку
             3 - Відсортувати список за заданим атрибутом
             4 - Видалити елемент за заданим індексом
             5 - Видалити елемент за заданим атрибутом
             6 - Вивести елементи із заданим атрибутом
             7 - Вихід

Виберіть операцію натиснувши відповідну цифру: 6

За яким атрибутом вивести:
             1 - Виробник
             2 - Модель
             3 - Рік випуску
             4 - Пробіг
             5 - Макс. швидкість

Виберіть атрибут натиснувши відповідну цифру: 3
Вкажіть значення атрибуту: 2017
         Виробник           Модель          Рік випуску     Пробіг, км    Макс. швидкість, км/год
  1.      Toyota             Camry             2017          100000                280
```

3. ЗМІСТ ЗВІТУ

1. Мета роботи.
2. Повний текст завдання згідно варіанту.
3. Лістинг програми.
4. Результати роботи програм (у текстовій формі та скріншот).
5. Висновок.

4. КОНТРОЛЬНІ ЗАПИТАННЯ

1. Які переваги дає використання списку?
2. Як можна звернутися до окремого елементу списку?
3. Які можливі способи заповнення списку?

5. СПИСОК ЛІТЕРАТУРИ

1. Learn to Program with Python 3. A Step-by-Step Guide to Programming, Second Edition / Irv Kalb. – Mountain View: Apress, 2018. – 361 p.
2. The Python Workbook. A Brief Introduction with Exercises and Solutions, Second Edition / Ben Stephenson. – Cham: Springer, 2014. – 218 p.
3. Python Pocket Reference, Fifth Edition / Mark Lutz. – Sebastopol: O'Reilly Media, Inc., 2014. – 264 p.
4. Learn Python 3 the Hard Way / Zed A. Shaw. – Boston: Addison-Wesley, 2017. – 321 p.
5. A Python Book: Beginning Python, Advanced Python, and Python Exercises / Dave Kuhlman. – Boston: MIT, 2013. – 278 p.

НАВЧАЛЬНЕ ВИДАННЯ

Програмування з використанням списків

МЕТОДИЧНІ ВКАЗІВКИ
до лабораторної роботи № 5
з курсу «Програмування скриптовими мовами»
для студентів спеціальності
«Кібербезпека»

Укладач: Я. Р. Совин, канд. техн. наук, доцент