Міністерство освіти і науки України
Державний заклад
«Південноукраїнський національний педагогічний університет
імені К. Д. Ушинського»
Кафедра прикладної математики та інформатики

Кобякова Людмила Миколаївна

Методичні рекомендації
«Python. Частина 2: Рядки. Списки»

для самостійної підготовки і виконання лабораторних робіт
з дисципліни «Програмування»
для здобувачів першого (бакалаврського) рівня вищої освіти
1 року навчання спеціальності 014.09 Середня освіта (інформатика)

Одеса – 2025

УДК: 004.42+004.432

Рекомендовано до друку Вченою Радою Державного закладу «Південноукраїнський національний педагогічний університет імені К.Д.Ушинського»

(протокол № 12 від « 27 » березня 2025 року).

Кобякова Л.М. **Python. Частина 2: Рядки. Списки:** Методичні рекомендації для самостійної підготовки і виконання лабораторних робіт з дисципліни «Програмування» для здобувачів першого (бакалаврського) рівня вищої освіти спеціальності 014.09 Середня освіта (інформатика). Одеса: Університет Ушинського, 2025. 43 с.

Рецензенти:

Роговська Марія Георгіївна, доцент, кандидат фізико-математичних наук, в.о. зав. кафедри фізико-математичних наук Державного університету інтелектуальних технологій та зв'язку

Бойко Ольга Павлівна, кандидат фізико-математичних наук, доцент, доцент кафедри прикладної математики та інформатики Південноукраїнського національного педагогічного університету імені К. Д. Ушинського.

Поданий у методичних рекомендаціях матеріал охоплює питання тем «Рядки», «Списки» мовою програмування Python, що формують уявлення про методику розробки програм, є фундаментом програмування і важливою ланкою професійної освіти для спеціалістів у галузі прикладної математики та інформатики.

Реалізація навчальних прикладів акцентує увагу на принципах побудови ефективних алгоритмів, способах опису вхідних і вихідних даних для вирішення прикладних задач.

Зміст

Вступ

Навчальна дисципліна ОК 11 «Програмування» для здобувачів першого (бакалаврського) рівня вищої освіти 1-го року навчання спеціальності 014.09 Середня освіта (інформатика) за навчальним планом підготовки фахівця охоплює 12 кредитів загальною кількістю 360 годин. Аудиторні заняття – 120 годин (лекції – 20 годин, лабораторні роботи — 100 години), 240 годин відведено на самостійну роботу студентів. Зміст навчальної дисципліни розподілено за програмою на 3 модуля, перший присвячено основам інформатики, другий охоплює питання процедурного програмування мовою програмування Python.

Мета навчальної дисципліни: формування базових понять процедурного програмування, розвиток критичного мислення, покращення та збагачення системи понять, знань, умінь і навичок програмування, які є необхідною частиною підготовки фахівця в галузі інформатики.

Передумови для вивчення дисципліни: для опанування розділами курсу «Програмування»: необхідно щоб студенти володіли на рівні, не нижче достатнього, навчальним матеріалом, передбаченим програмами з інформатики та математики загальноосвітніх навчальних закладів.

Методичні рекомендації «Python. Частина 2: Рядки. Списки» розкривають питання тем методів і алгоритмів обробки рядків і списків навчальної програми першого модуля навчальної дисципліни ОК 11 «Програмування». За програмою навчальної дисципліни заняття розраховані на 12 годин аудиторної і 24 години самостійної роботи.

Розроблені лабораторні роботи є основою для засвоєння процедурного програмування, а в подальшому – об’єктно-орієнтованого програмування та практичним підґрунтям для дисциплін комп’ютерного циклу.

Лабораторна робота №6

РЯДКИ

Зміст

Рядок складається з символів Unicode

Приклади:

1) рядок `"abc"` складається з 3 символів `"a", "b", "c"`
2) рядок `"a_1"` складається з 3 символів `"a", "_", "1"`
3) в порожньому рядку `""` 0 символів

1. Рядкові (текстові) змінні: надання значень

Для надання значень рядковим змінним (змінним типу str) є 4 способи:

1 спосіб: одинарні лапки

```
1  s = 'The flower blooms beautifully'
2  print(s)
3  print( type(s) )
```

```
Y  На екрані:
   The flower blooms beautifully
   <class 'str'>
```

2 спосіб: подвійні лапки

```
1  s = ”You are a student”
2  print(s)
```

Якщо у рядку є текст у лапках (наприклад, назва книги), то один вид лапок використовується для рядка, інший – для тексту в лапках (назви книги).

```
1  s1 = '"Піп" - герой роману "Великі надії"'
2  s2 = "Автор роману 'Джейн Ейр' – Шарлотта Бронте"
```

3 спосіб: потрійні одинарні/подвійні лапки використовуються для багаторядкового тексту

```
1  s = """There was no possibility of taking a walk that day.
   We had been wandering, indeed, in the leafless shrubbery an
   hour in the morning; but since dinner (Mrs. Reed, when
   there was no company, dined early) the cold winter wind had
   brought with it clouds so sombre, and a rain so
   penetrating, that further out-door exercise was now out of
   the question."""
2  print( s )
```

```
На екрані:
There was no possibility of taking a walk that day.
We had been wandering, indeed, in the leafless shrubbery an
hour in the morning; but since dinner (Mrs. Reed, when
there was no company, dined early) the cold winter wind had
brought with it clouds so sombre, and a rain so
penetrating, that further out-door exercise was now out of
the question.
```

```
1  s = '''I was glad of it: I never liked long walks,
   especially on chilly afternoons: dreadful to me was the
   coming home in the raw twilight, with nipped fingers and
   toes, and a heart saddened by the chidings of Bessie, the
   nurse, and humbled by the consciousness of my physical
   inferiority to Eliza, John, and Georgiana Reed.'''
2  print( s )
```

4 спосіб: довгий рядок можна розбити на частини та розмістити на різних рядках коду. У цьому випадку весь рядок розташовується у круглих дужках, а частини – у лапках

```
1  text = ("Laudate omnes gentes laudate "
```

```
        "Magnificat in secula" )
print(text)
```

```
На екрані:
Laudate omnes gentes laudate Magnificat in secula
```

2. Об'єднання та повторення рядків

1) конкатенація (об'єднання) – додавання (+) рядків

```
s1 = "aaa"
s2 = "bbb"
s3 = s1 + s2    # aaabbb
print(s3)
```

Якщо потрібно скласти рядок і число, то необхідно перетворити число до рядкового типу за допомогою функції `str()`

```
name = "Петров"
age  = 38
info = "Name: " + name + " Age: " + str(age)
print(info)
```

```
На экрані:
Name: Петров Age: 38
```

2) реплікація (повторення) – «множення» рядка (*) на ціле число

```
s1 = "aaa"

s2 = s1 * 2    # aaaaaa
print(s2)

s3 = 2 * s1    # aaaaaa
print(s3)
```

3. Керуючі символи, або escape-послідовності

Рядок може містити спеціальні керуючі символи.

Вони мають вигляд: `\символ`

Керуючий символ	Дія (що робить)
`\\`	Вставляє в рядок символ слеш (\)
`\’`	Вставляє в рядок одинарну лапку
`\"`	Вставляє в рядок подвійну лапку
`\n`	Здійснює перехід на наступний рядок
`\t`	Вставляє табуляцію (4 відступа)

```
s = "Message: \n\"Hello, World!\""
```

```
2  print(s)
```

```
На екрані:
Message:
"Hello, World!"
```

4. r-рядки

Керуючі символи можуть створювати проблеми. Наприклад,

```
1  # Шлях до файлу
2  path = "C:\python\name.txt"
3  print(path)
```

```
На екрані:
C:\python
ame.txt
```

`\n` у цьому рядку сприймається як керуюча послідовність, що здійснює перехід на наступний рядок, що й відбувається.

Якщо потрібно, щоб Python сприймав рядок так, як він написаний, використовуються **r**-рядки, тобто. перед рядком ставиться буква **r** (read – читати)

```
1  # Шлях до файлу
2  path = r"C:\python\name.txt"
3  print(path)
```

```
На екрані:
C:\python\name.txt
```

5. Функції ord(), chr() і len()

Рядок складається з символів Unicode.

Функція **ord(символ)** повертає код символу в кодуванні Unicode.

```
1  print( ord("1") )     #  49
2  print( ord("A") )     #  65
3  print( ord("a") )     #  97
```

Функція **chr(код символу)** повертає символ з кодом символу в кодуванні Unicode.

```
1  print( chr(49) )      #  1
2  print( chr(65) )      #  A
3  print( chr(97) )      #  a
```

Функція **len(строка)** повертає довжину рядка – кількість символів

```
1  s = "Hello, World!"
2  print( len(s) )       # 13
```

6. Порівняння рядків

Порівняння рядків здійснюється у лексикографічному порядку, тобто. посимвольно з урахуванням коду символу в Unicode за правилами:

1) цифровий символ умовно менший за будь-який алфавітний символ, тому що `код(цифрового символу) < коду(алфавітного символу)`

2) алфавітний символ у верхньому регістрі умовно менший за будь-який алфавітний символ у нижньому регістрі, тому що

`код(«великої» букви») < коду(«маленької букви»)`

```
1  s1 = "abc1abc"
2  s2 = "abcAabc"
3  s3 = "abcaabc"
4
5  print(s1 > s2)
6  print(s2 < s3)
```

```
На екрані:
False  # тому що код(“1”) < кода(“A”)
True   # тому що код(“A”) < кода(“a”)
```

```
1  s1 = "abcabc"
2  s2 = "abcabc"
3  s3 = "abcbbc"
4
5  print(s1 == s2)    # True
6  print(s2 != s3)    # True
```

7. Індексація рядків

Для звернення до символу рядка використовується індекс – порядковий номер.

Python підтримує 2 типи індексації:

– *додатну* (зліва направо, відлік починається з початку рядка):

від 0 до довжини (рядку) – 1;

– *від’ємну* (справа наліво, відлік починається з кінця рядка):

від –1 до –довжина (рядку)

Рядок	P	y	t	h	o	n
Додатні індекси	0	1	2	3	4	5
Від’ємні індекси	–6	–5	–4	–3	–2	–1

Щоб отримати певний символ рядка, потрібно після імені рядкової змінної вказати його індекс у квадратних дужках

```
s = "abcdef"

print( s[2] )       # "c"
print( s[-4] )      # "c"
```

Працюючи із символами слід враховувати, що рядок – незмінний (immutable) тип даних. Тому спроба змінити один символ призведе до помилки. Ми можемо лише перевстановити значення рядка (як цілого), надавши йому інший вираз

```
s = "abcdef"
s[2] = "C"      # Так неможна! Помилка
```

```
На екрані:
s[2] = "C"
~~~~~~^^^
TypeError: 'str' object does not support item assignment
```

```
s = "abcdef"
s = "abCdef"    # А так можна
```

8. Отримання підрядка (зрізи)

Можна отримати з рядка не лише окремі символи, а й частину рядка – підрядок.

Синтаксис:

- `string[ :  end]` – витягується послідовність символів, починаючи з **0**-го по індекс **end** невключно (тобто. по **end-1**-ший символ)
- `string[ start : ]` – витягується послідовність символів, починаючи з індекса **start** по останній включно
- `string[ start : end ]` – витягується послідовність символів, починаючи з індекса **start** по **end** невключно
- `string[ start : end : step ]` – витягується послідовність символів, починаючи з індекса **start** по **end** невключно через крок **step**

```
string = "абракадабра"

# с 0-го по 3-й символ
```

```
sub_string_1 = string[ : 4]
print( sub_string_1 )           # абра

# с 4-го по останній символ
sub_string_2 = string[ 4 : ]
print( sub_string_2 )           # кадабра

# с 4-го по 8-й символ
sub_string_3 = string[ 4 : 9 ]
print( sub_string_3 )           # кадаб

# кожний 3-й символ
sub_string_4 = string[ : : 3]
print( sub_string_4 )           # аадр

# з останнього по 0-й (рядок у зворотньому порядку)
sub_string_5 = string[ : : -1]
print( sub_string_5 )           # арбадакарба

# с -3-го по -довжина(рядку)+1-ший
n = len(string)
sub_string_6 = string[ -3 : -n : -1]
print( sub_string_6 )           # бадакарб
```

Зрізи можна використовувати для змінення значення рядкової змінної

```
s = "abcdef"
s = s[:2] + "!!!" + s[4:]   # "ab!!!ef"
```

9. Перевірка рядка на входження/приналежність до іншого рядка

Щоб визначити, чи є один рядок підрядком іншого рядка, використовується оператор `in`

```
s1 = "abcdef"
s2 = "bcd"
s3 = "aba"

print( s2 in s1 )      # True
print( s3 in s1 )      # False
```

10. Перебір символів рядку

1 спосіб

```
string = "abcd"
for symbol in string :
    print( symbol )
```

```
На екрані:
a
b
c
d
```

2 спосіб: використати індекси

```
1  s = "abcd"
2
3  n = len(s)          # довжина рядку
4  for i in range(n) :
5      print( s[i] )
```

```
На екрані:
a
b
c
d
```

11. Методи роботи з рядками

Усі методи можна згрупувати у 4 категорії:

- Перетворення рядків
- Оцінка та класифікація рядків
- Зміна регістру
- Пошук, підрахунок та заміна символів

11.1. Перетворення рядків

`“розділювач”.join(список/кортеж рядків)` об’єднує список/кортеж рядків в один рядок, вставляючи між ними розділювач

```
1  spisok_strok = [ "I", “really”, “love”, “you”, “again” ]
2  result_stroka = “ ”.join( spisok_strok )
3  print( result_stroka )
```

```
На екрані:
I really love you again
```

```
1  kortezh_strok = ( "1", “22”, “333”, “4444” )
2  result_stroka = “!”.join( kortezh_strok )
3  print( result_stroka )
```

```
На екрані:
1!22!333!4444
```

`split(розділювач, кількість)` розбиває рядок на підрядки по розділювачу і формує список підрядків.

Якщо розділювач не вказано, розбиває по пробілу.

Якщо кількість не вказана, розбиває по всім входженням розділювача в рядок.

Якщо кількість (`num`) вказана, розбиває по першим `num` входженням розділювача в рядок.

```
s = "aaa bbb ccc ddd"

x = s.split()           # ['aaa', 'bbb', 'ccc', 'ddd']
print(x)

y = s.split(" ", 2)     # ['aaa', 'bbb', 'ccc ddd']
print(y)
```

`partition(розділювач)` розбиває рядок по розділювачу на три підрядки і повертає кортеж із трьох елементів – підрядок до розділювача, розділювач та підрядок після розділювача

```
s = "abc!ABC!cba"

x = s.partition('!')
print(x)
```

```
На екрані:
('abc', '!', 'ABC!cba')
```

`ljust(width):` якщо довжина рядка менша за параметр `width`, то праворуч від рядка додаються пробіли, щоб доповнити значення `width`, а сам рядок вирівнюється по лівому краю

`rjust(width)` якщо довжина рядка менша за параметр `width`, то ліворуч від рядка додаються пробіли, щоб доповнити значення `width`, а сам рядок вирівнюється праворуч.

`center(width)` якщо довжина рядка менша за параметр `width`, то ліворуч і праворуч від рядка рівномірно додаються пробіли, щоб доповнити значення `width`, а сам рядок вирівнюється по центру

```
s = "abcdefgh"

```

```
3   s1 = s.ljust(12)
4   print("!" + s1 + "!")     # !abcdefgh    !
5
6   s2 = s.rjust(12)
7   print("!" + s2 + "!")     # !    abcdefgh!
8
9   s3 = s.center(12)
10  print("!" + s3 + "!")     # !  abcdefgh  !
```

На екрані:

```
!abcdefgh    !
!    abcdefgh!
!  abcdefgh  !
```

11.2. Оцінка та класифікація рядків

Спеціальні методи визначення типу символів:

• `isalpha()`: повертає `True`, якщо рядок складається лише з алфавітних символів

• `isalnum()` оцінює, чи складається рядок з букв і цифр, чи в ньому є якісь інші символи:

• `isdigit()`: повертає `True`, якщо всі символи рядка – цифри

• `isnumeric()`: повертає `True`, якщо рядок являє собою число, дріб або римську цифру

11.3. Зміна регістру

`lower()`: переводить рядок у нижній регістр

`upper()`: переводить рядок у верхній регістр

`title()`: перші символи всіх слів рядку переводяться в верхній регістр

`capitalize()`: переводить в верхній регістр першу літеру тільки самого першого слова рядку

```
1   s1 = "AbCdEfG"
2
3   r1 = s1.lower()
4   print(r1)              # abcdefg
5
6   r2 = s1.upper()        # ABCDEFG
7   print(r2)
```

```
s = "вечоріло. хмари пливли небом"

r1 = s.title()       # Вечоріло. Хмари Пливли Небом
print(r1)

r2 = s.capitalize()  # Вечоріло. хмари пливли небом
print(r2)
```

11.4. Пошук, підрахунок та заміна символів

`find(str, start, end)`: повертає індекс підрядка в рядку.

Якщо підрядок не знайдений, повертає число –1

`start` і `end` – необов'язкові параметри

`start` задає номер символу, з якого починається пошук підрядку в рядку

`end` задає номер символу, до якого здійснюється пошук підрядку в рядку

```
#    0123456789
s = "abbabccbad"

index1 = s.find('ab')       #  0
print(index1)

index2 = s.find("ac")       # -1
print(index2)

index3 = s.find("ab", 2)    #  3
print(index3)

index4 = s.find("ad", 2, 7) # -1
print(index4)
```

`replace(old, new, num)`: будує новий рядок, в якому замінює у вихідному рядку один підрядок на інший

`num` – необов'язковий параметр

`old` – підрядок, який шукається і замінюється

`new` – підрядок, на який робиться заміна

`num` – кількість замін, які потрібно зробити

```
string = "abbabccabd"

newstring1 = string.replace('ab', 'AB')    # ABbABccABd
print(newstring1)
```

```
5
6   newstring2 = string.replace('ab', 'AB', 2) # ABbABccabd
7   print(newstring2)
```

12. Завдання для самостійного розв'язку

1) Дан текст. Надрукувати всі наявні у ньому цифри, не повторюючи ті, які вже зустрічалися

2) Дан текст. Знайти максимальне і мінімальне з наявних у ньому чисел.

3) Даний текст. Знайти найбільшу кількість однакових символів, що йдуть поспіль.

4) Дано два слова. Визначити, чи можна з літер першого з них отримати друге. Розглянути два варіанти:

1. повторювані букви другого слова можуть у першому слові не повторюватися;

2. кожна літера другого слова повинна входити до першого слова стільки ж разів, скільки і другого.

5) Дано речення. Визначити:

а) кількість слів, що починаються з літери н;

б) кількість слів, що закінчуються літерою р

в) кількість слів, а яких зустрічається літера о

Лабораторна робота №7

СПИСКИ

Список (`list`) – тип даних, призначений для зберігання і обробки набору/послідовності елементів.

Список може складатися з елементів **різних типів** на відміну від **масиву**, який містить елементи **одного типу**.

Зміст

1. Створення порожнього списку

1 спосіб: за допомогою квадратних дужок

```
1  x = []        # x – порожній список
2
3  # Виведення списку на екран
4  print(x)      # []
```

2 спосіб: використати функцію **`list()`**

```
1  x = list()    # x – порожній список
2  print(x)      # []
```

2. Створення списку

1 спосіб – в квадратних дужках перелічити елементи

Список може складатися з елементів різних типів.

```
1  c = [1, 3.5, "aaa", [1, 3, 5]]
```

2 спосіб – використати генератор списку, тобто вказати формулу, за якою обчислюються елементи

Синтаксис генератора списку

1. `[ вираз(i)  for i in range(n) ]`
2. `[ вираз(x)  for x in список]`
3. `[ вираз(x)  for x in список  if  умова(x) ]`

Приклад

Побудувати список, елементи якого перші 10 непарних чисел

```
1  n = 10     # кількість елементів списку
2  d = [2*i+1   for i in range(n)]   # генератор списку
3  print(d)   # [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
```

`2*i+1, i = 0,...,n-1` – перші `n` непарних чисел

Приклад

Використовуючи генератор, побудувати список випадкових цілих чисел з діапазону $[-10, 20]$

```
1  import random as rand # підключення бібліотеки random
2
3  n = 10                # кількість елементів списку
4  # мінімальне і максимальне значення елементів списку
5  xmin, xmax = -10, 20
6
7  # генератор списку
8  x = [ rand.randint(xmin, xmax)   for i in range(n) ]
9  print(x)
```

Функція `randint(a, b)` генерує псевдовипадкове ціле число з діапазону `[a, b]` (квадратні дужки означають, що границі діапазону `a, b` включені в діапазон)

Приклад

Побудувати список, який складається з квадратів перших n невід`ємних цілих чисел

```
1  n = 10
2  v  = list( range(n) )   # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
3  print(v)
4
5  vv = [element**2  for element in v]   # [0, 1, 4, 9, 16,
   25, 36, 49, 64, 81]
6  print(vv)
```

Приклад: *Вибірка*

Побудувати список непарних елементів вихідного списку

```
1  # Вихідний список
2  n = 10
3  v  = list( range(n) )   # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
4  print(v)
5
6  # Вибірка: список непарних елементів вихідного списку
7  w  = [element   for element in v   if element % 2 == 1]
   # [1, 3, 5, 7, 9]
8  print(w)
```

3 спосіб – ввести елементи списку з клавіатури (2 варіанта див. нижче)

1 варіант коду в п.18

2 варіант коду в п.18

4 спосіб – використати конструктор `list( аргумент )`

аргумент – набір значень

```
list(рядок)  = список із символів рядка
list(кортеж) = список із елементів кортежа
list(список) = список із елементів списка
list(range(a, b, h)) = список із елементів діапазону   a,
a+h, ... a+k*h (< b)
```

Приклади

```
1  symbols = list("Hello!")    # ['H', 'e', 'l', 'l', 'o',
   '!']
```

```
1  kortage = (1, 4, 2, 6)
2  x = list( kortage )      #  [1, 4, 2, 6]
```

```
1  spisok = [1, 4, 2, 6]
2  x = list( spisok )       #  [1, 4, 2, 6]
```

```
1  diapazon = range(10)
2  x = list( diapazon )    # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
1  diapazon = range(0, 103, 20)
2  x = list( diapazon )     # [0, 20, 40, 60, 80, 100]
```

3. Виведення списку на екран (друк списку)

```
1  # Заданий список
2  x = [ 2, 8, 3, 0, -8, 7, 9, 7, -13, 5, 13 ]
```

1 спосіб – друк звичайної змінної

```
3  print(x)    # [ 2, 8, 3, 0, -8, 7, 9, 7, -13, 5, 13 ]
```

2 спосіб – список друкується без квадратних дужок

– елементи розділені пробілами

```
3  print(*x)   # 2 8 3 0 -8 7 9 7 -13 5 13
```

– елементи розділені рядком-значенням параметру `sep`

```
3  print(*x, sep = " ! ")
   На екрані:
   2 ! 8 ! 3 ! 0 ! -8 ! 7 ! 9 ! 7 ! -13 ! 5 ! 13
```

3 спосіб – використати цикл (див. нижче п. Перебір елементів списку)

4 спосіб – форматований друк

```
3  print( "%4d" * len(x)   %   tuple(x) )
   На екрані:
      2   8   3   0  -8   7   9   7 -13   5  13
```

Синтаксис (одного із способів) **форматованого друку**

```
print( форматуючий_рядок   %   кортеж_змінних)
```

1) `len(x)` – кількість елементів списку

2) Список `x` складається з цілих елементів.

Форматуючий рядок `"%4d" * len(x)` виділяє кожному (цілому) елементу по 4 позиції

3) `tuple(x)`: список x, перетворений в кортеж.

4. Нумерація (індексація) елементів списку

Додатна нумерація елементів списку проводиться від початку к кінцю і починається з 0, тобто перший елемент списку має номер 0, останній: n–1.

Від'ємна нумерація проводиться від кінця к початку і починається з –1, тобто останній елемент списку має номер –1, перший: –n.

```
Додатна нумерація     0   1   2   3   4
Список x              5   2   6   8   1
```

```
Від'ємна нумерація  -5  -4  -3  -2  -1
```

5. Звертання до елемента списку

`ім'я_списку[ індекс_елемента ]`

```
#     0   1   2   3   4   5   6   7 - додатна нумерація
x = [ 9,  2,  1,  8,  3, 11,  0,  5 ]
#    -8  -7  -6  -5  -4  -3  -2  -1 - від'ємна нумерація

# Друк елемента списку з індексом 5
print( x[5] )   # 11
# Друк елемента списку з індексом -3
print( x[-3] )  # 11
```

```
# Змінення значення елемента з індексом 2
x[2] = 4
```

6. Список (`list`) – змінюваний тип даних (`mutable`)

«Змінюваність» (`mutable`) типу даних `list` означає, що можна

- додати елемент в список
- видалити елемент списку
- змінити елемент списку.

«Незмінюваність» (`immutable`) типу даних означає, що частину/елемент послідовності змінити неможливо, потрібно змінити все, цілком (як ціле).

Список – змінюваний тип даних.

Рядок, кортеж – незмінювані типи даних.

```
x = [4, 6, 2, 1]
print(x)

x[1] = 100   # Змінення значення елемента списку
print(x)     # [4, 100, 2, 1]
```

7. Зрізи

Зріз – це частина списку.

Як побудувати зріз?

`список[ : end]` – частина списку з початку до елемента з індексом `end` (невключно), тобто буде побудований список, який складається з еле-

ментів `список[0] ... список[end-1]`

`список[ start : ]` – частина списку, починаючи з елемента з індексом `start` до кінця списку

`список[start : end]` – частина списку, починаючи з елемента з індексом `start` до елемента з індексом `end` (невключно)

`список[start : end : step]`: параметр `step` вказує на шаг, через який будуть копіюватися елементи зі списку. За замовчуванням цей параметр дорівнює `1`.

```
1   #     0   1   2   3   4   5   6   7  – додатна нумерація
2   x = [ 9,  2,  1,  8,  3, 11,  0,  5 ]
3   #    -8  -7  -6  -5  -4  -3  -2  -1 – від'ємна нумерація
4
5   print( x[:])            # весь список x
6   print( x[ : 4] )        # [9, 2, 1, 8]
7   print( x[4 : ] )        # [3, 11, 0, 5]
8   print( x[3 : 6+1] )     # [8, 3, 11, 0]
9   print( x[2 : 6 : 2] )   # [1, 3]
10  print( x[-6 : -2] )     # [1, 8, 3, 11]
11
12  print( x[ : : -1] )     # [5,0,11,3,8,1,2,9]  (список пе-
    ревернутий)
13  print( x[ : : -2] )     # [5, 11, 8, 2]
```

Зрізи дозволяють замінити частину списка

```
1   n = 10
2   g = [ 2*i   for i in range(n)]
3   print(g)     # [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
4
5   g[3 : 8] = [100, 200]
6   print(g)     # [0, 2, 4, 100, 200, 16, 18]
```

8. Копіювання списків

Дано список `x`. Потрібно створити його копію.

У чому проблема?

Що ми робимо, коли потрібно створити копію змінної?

Присвоюємо її значення іншій змінній.

```
1   x = [1, 10]
2   y = x
```

```
3
4   print(f"x = {x}")       # x = [1, 10]
5   print(f"y = {y}")       # y = [1, 10]
```

Змінимо один елемент списку `x` і надрукуємо списки `x` і `y`

```
1   x[1] = 5
2
3   print(f"x = {x}")      # x = [1, 5]
4   print(f"y = {y}")      # y = [1, 5]
```

Що ми бачимо?

Ми змінили список x, а в результаті – змінився елемент і в x, і в y.

Чому це сталося? Давайте розумітися.

Нагадаємо, що змінна в Python-і – це посилання на фізичний об'єкт. Фізичним об'єктом в даному випадку є список.

Що є результатом дії

`x = [1, 10]` ?

1. створюється фізичний об'єкт – список [1, 10]
2. змінна x не дорівнює цьому списку, вона дорівнює посиланню на нього.

Тому в результаті присвоювання

`y = x`

не створюється новий список, а змінна y стає ще одним посиланням на список [1, 10], тобто фізичний об'єкт один, а посилань на нього 2: змінні x и y.

Тому коли ми змінюємо (фізичний) список, звертаючись до нього за допомогою змінної x, а потім виводимо змінні x і y на екран, ми бачимо на екрані, що «змінилися обидва списка»

!!! ***Змінився фізичний список (він один), на який посилалися 2 змінні x і y.***

Тобто присвоювання призводить не до створення копії списку, а до створення копії посилання на нього.

І що робити?

Є ще 2 інструмента:

1) Метод `copy()` створює поверхневу копію списку

2) Функція `deepcopy()` з бібліотеки `copy` створює глибоку копію

списку.

Пояснимо на прикладах, у чому полягає різниця між поверхневою і глибокою копіями списку

Застосовуючи метод `список.copy()`, створимо поверхневу копію списку

```
1  x = [1, 2, [10, 20], 4]
2  print(f"x = {x}")          # x = [1, 2, [10, 20], 4]
3
4  # Створимо поверхневу копію списку x
5  y = x.copy()
6  print(f"y = {y}")          # y = [1, 2, [10, 20], 4]
```

Змінимо елемент списка x, **який є числом,** і подивимось, чи призведе ця дія до змінення відповідного елемента списка y

```
1  x[0] = 100
2
3  print(f"x = {x}")       #  x = [100, 2, [10, 20], 4]
4  print(f"y = {y}")       #  y = [1, 2, [10, 20], 4]
```

Ні, не привело.

Експериментуємо далі.

Ми пам'ятаємо, що проблема виникала при створенні копії списку.

У списку x є підсписок [10, 20] – елемент x[2]. Змінимо елемент цього підсписка, і подивимось, чи змінився відповідний елемент списку y.

```
1  # У підсписку [10, 20] змінимо 20 на 3
2  x[2][1] = 3
3
4  print(f"x = {x}")      # x = [100, 2, [10, 3], 4]
5  print(f"y = {y}")      # y = [1, 2, [10, 3], 4]
```

Що ми бачимо?

У підсписках списків x і y змінились відповідні елементи.

Чому?

При створенні поверхневої копії копіюється "перший шар". Це приводить до того, що копіюються числа/рядки і посилання на підсписки, а це значить, що копії підсписків не створюються

x = [1, 2, посилання, 4] y = [1, 2, посилання, 4]

| |

----------> [10, 3] <----------

Створимо глибоку копію списку, використовуючи функцію `deepcopy()` з бібліотеки `copy`

```
1  import copy
2
3  x = [1, 2, [10, 20], 4]
4  print(f"x = {x}")            #  x = [1, 2, [10, 20], 4]
5
6  # Створюємо глибоку копію списку x
7  y = copy.deepcopy(x)
8  print(f"y = {y}")            #  y = [1, 2, [10, 20], 4]
```

Робимо те ж саме: змінюємо елемент підсписка списку `x`, і подивимось, чи привело це до змінення відповідного елемента підсписка списка `y`?

```
1  x[2][1] = 3
2
3  print(f"x = {x}")      #  x = [1, 2, [10, 3],  4]
4  print(f"y = {y}")      #  y = [1, 2, [10, 20], 4]
```

Ні, не привело, список `y` не змінився.

При глибокому копіюванні копіюються всі "шари" списка, тобто для елементів всіх типів створюються копії фізичних об'єктів, а не копіюються посилання на них.

9. Розкладання списку на окремі елементи і підсписки

Можна розкласти список/зріз списку на окремі елементи.

При цьому кількість змінних повинна дорівнювати числу елементів списку

```
1  x = [1, 3, 5]
2
3  # Розкладання списку на окремі елементи
4  a1, a2, a3 = x
5
6  print(a1, a2, a3)    # 1 3 5
7  print( type(a1) )    # <class 'int'>
```

```
1  x = list( range(10) )   # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
print(x)

# Розкладання списку на окремі елементи
a1, a2, a3, a4 = x[0 : 4]    # 0 1 2 3
print(a1, a2, a3, a4)
```

Список може бути розділений на окремі елементи або набір елементів і підсписки за допомогою оператора *****

```
x = list( range(10))   # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print(x)

# Розкладання списку на елементи і підсписок
a, *b, c, d = x
print(f'a = {a}\n b = {b}\n c = {c}\n d = {d}')
```

```
На екрані:
a = 0
b = [1, 2, 3, 4, 5, 6, 7]
c = 8
d = 9
```

10. Перебір елементів списку

Список, як і рядок, – об'єкти, які перебираються (ітеруються).

1 спосіб – цикл по елементам списку

```
spisok = [1, 3, 5, 7]
for  element  in  spisok :
    print(element)
```

```
На екрані:
1
3
5
7
```

Друк елементів не в стовпчик, а в рядок

```
spisok = [1, 3, 5, 7]
for element in spisok :
    print(element,  end = " ")
```

```
На екрані:
1 3 5 7
```

2 спосіб – цикл по індексам елементів списку

```
spisok = [1, 3, 5, 7]

n = len(spisok)   # довжина списку (кількість елементів)
```

```
4
5
```
```
for i in range(n) :
    print( spisok[i] )
```

```
На екрані:
1
3
5
7
```

11. Функція zip

Функція `zip` дозволяє одночасно виконувати дії з декількома об'єктами, що ітеруються (списками, рядками і т.ін.).

```
1
2
3
4
```
```
numbers = [10, 20, 30]
symbols = ['a', 'b', 'c', 'd', 'e']
for num, symb in zip(numbers, symbols) :
    print(f"{num} \t {symb}")
```

```
На екрані:
10      a
20      b
30      c
```

Вираз `zip(numbers, symbols)` створює об'єкт-ітератор, з якого при кожному обороті циклу витягується кортеж, що складається з 2 елементів: перший – з списку `numbers`, другий – з списку `symbols`.

У списку `symbols` на 2 елемента більше. Функція `zip` повертає ітератор, який зупиняється, коли вичерпується найкоротша послідовність.

Виведення на екран zip-об'єкта

```
1
2
3
4
5
6
```
```
numList = [0, 1, 2]
engList = ['zero', 'one', 'two']
espList = ['cero', 'uno', 'dos']

zip_object = zip(numList, engList, espList)
print( zip_object )
```

```
На екрані:
<zip at 0x28d49611740>
```

Щоб вивести zip-об'єкт на екран, перетворимо його на список

```
1
```
```
print( list(zip_object) )
```

```
На екрані:
[(0, 'zero', 'cero'), (1, 'one', 'uno'), (2, 'two', 'dos')]
```

Скористаємося `zip` і циклом `for` для одночасного перебору 3-х спис-

ків:

```
1  for num, eng, esp in zip(numList, engList, espList):
2      print(f'{num} англійською {eng} і {esp} – іспанською')
```

На екрані:

```
0 англійською zero і cero – іспанською
1 англійською one і uno – іспанською
2 англійською two і dos – іспанською
```

Побудуємо список поєднаних zip-ом 3-х списків:

```
1  x = list( zip(espList, engList, numList) )
2  print(x)
```

На екрані:

```
[('cero', 'zero', 0), ('uno', 'one', 1), ('dos', 'two', 2)]
```

Відсортуємо по `espList` (він 1-й)

```
1  x.sort()
2  print(x)
```

На екрані:

```
[('cero', 'zero', 0), ('dos', 'two', 2), ('uno', 'one', 1)]
```

Розпакуємо в 3 змінні:

```
1  spanish, english, numeric = zip(*words)
2  print(spanish)       #  ('cero', 'dos', 'uno')
3  print(english)       #  ('zero', 'two', 'one')
4  print(numeric)       #  (0, 2, 1)
```

12. Функція enumerate

Функція enumerate() створює пари, що складаються з автоматичного лічильника і елементів об’єкта, що ітерується. Ці пари упаковані в кортежі.

```
1  for index, symbol in enumerate( ['a', 'b', 'c'] ) :
2      print(f"{index} \t {symbol}")
```

На екрані:

```
0       a
1       b
2       c
```

В enumerate можно вказати початкове значення індексу

```
1  upperCase = ['A', 'Б', 'В', 'Г']
2  lowerCase = ['a', 'б', 'в', 'г']
3
4  for i, (upper, lower) in enumerate(zip(upperCase,
                                          lowerCase), start = 1):
5      print(f'{i}: {upper} и {lower}')
```

```
На екрані:
1: А и а
2: Б и б
3: В и в
4: Г и г
```

13. Порівняння списків за допомогою операторів

== (дорівнює) та != (не дорівнює)

Найпростіший спосіб переконатися, що списки ідентичні, використати оператор `==`.

Цей оператор порівнює списки елемент за елементом і повертає `True`, якщо всі елементи та їх порядок співпадають, і `False` – в протилежному випадку.

```
1  a = [1, 3, 5]
2  b = [1, 3, 5]
3  c = [1, 5, 3]
4
5  print( a == b )     # True
6  print( a == c )     # False
7  print( a != c )     # True
```

14. Об'єднання списків (додавання)

```
1  x = [1, 3, 5]
2  y = [2, 4, 6]
3
4  s = x + y          # [1, 3, 5, 2, 4, 6]
5  print(s)
```

15. Реплікація ("множення")

```
1  x = [0, 1] * 3     # [0, 1, 0, 1, 0, 1]
2  print(x)
```

16. Методи і функції обробки списків

16.1. Функції `dir()` і `help()`

Про перелік всіх методів списку можна довідатися за допомогою стандартної функції

```
dir(ім'я_списку або list)
```

В отриманому списку методів потрібно дивитися імена без подвійних підкреслень.

1	dir(list)
	На екрані: ['__add__', '__class__', '__class_getitem__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'append', 'clear', 'copy', 'count', 'extend', 'index',

```
 'insert',
 'pop',
 'remove',
 'reverse',
 'sort']
```

або так:

```
x = [1, 2]  # Довільний список, можна порожній
dir(x)      # Перелік методів списку див. у попередньому прикладі
```

Для отримання інформації про конкретний метод слід скористатися вбудованою функцією `help()`, передавши їй як аргумент ім'я методу, пов'язане з об'єктом чи класом.

Наприклад, `help(a.pop)` або `help(list.index)`.

```
help(list.index)
```

На екрані:

```
Help on method_descriptor:

index(self, value, start=0, stop=9223372036854775807, /)
    Return first index of value.

    Raises ValueError if the value is not present.
```

16.2. Функції len(), max() і min()

`len()` – довжина списку (кількість елементів у списку)

```
x = [1, 3, 5, 7]

n = len(x)     # 4
print(f"Довжина списку {x} : {n}")
```

Функції `min(список)` і `max(список)` знаходять мінімальний і максимальний елементи списку

```
x = [10, 3, 1, 7]
print(x)

xmin = min(x)      # 1
xmax = max(x)      # 10

print(f"xmin = {xmin}, xmax = {xmax}")
```

16.3. Додавання (вставка) елементів до списку:

методи `append()`, `extend()`, `insert()`

Метод `список.append(елемент)` додає елемент в кінець списку

```
1  x = [2, 4, 8]
2  print("Заданий список      x: ", x)
3
4  x.append( 100 )    #  [2, 4, 8, 100]
5  print("Результуючий список x: ", x)
```

```
1  x = [0, 1, 2]
2  print("Заданий список   x: ", x)
3
4  x.append( [100, 200] )     #  [0, 1, 2, [100, 200]]
5  print("Результуючий список x: ", x)
```

Приклад на завдання «зробити вибірку»

Дано вихідний список.

Побудувати список, що складається з елементів вихідного списку, кратних 3

1 спосіб – використати `append`

```
1  # Заповнимо вихідний список випадковими числами з {-10,
   -9, ..., 20}
2  import random as rand
3
4  n = 10
5  xmin, xmax = -10, 20
6  x = [ rand.randint(-10, 20)  for i in range(n) ]
7  print(x)
8
9  # Список y – вибірка зі списку x елементів, кратних 3
10 y = []
11 for element in x :
12     if element % 3 == 0 :
13         y.append(element)
14 print(y)
```

```
На екрані:
[16, -10, 15, -4, -9, 1, -5, -9, 14, 10]
[15, -9, -9]
```

2 спосіб – використати генератор списка (працює бистріше за append)

```
1   import random as rand
2
3   n = 10
4   xmin, xmax = -10, 20
5   x = [ rand.randint(-10, 20)  for i in range(n) ]
6   print(x)
7
8   # Генератор списку робить із списку x вибірку елементів,
    кратних 3
9   y = [element   for element in x    if element % 3 == 0]
10  print(y)
```

```
На екрані:
[20, -2, 1, 14, 6, 9, 0, -10, 10, -8]
[6, 9, 0]
```

Метод `список.extend( набір_елементів )` додає набір_елементів у кінець списка

```
1   x = [0, 1, 2]
2   print("Заданий список   x: ", x)
3
4   x.extend( [100, 200] )       # [0, 1, 2, 100, 200]
5   print("Результуючий список x: ", x)
```

Метод `список.insert(індекс, елемент)` вставляє елемент за індексом

```
1   x = [0, 1, 2, 3, 4]
2   print("Заданий список   x: ", x)
3
4   x.insert(2, 100)        # [0, 1, 100, 2, 3, 4]
5   print("Результуючий список x: ", x)
```

16.4. Видалення елементів:

методи `remove()`, `clear()`, `pop()`, оператор `del`

Метод `remove(item)`: видаляє елемент `item`.

Видаляється лише перше входження елемента.

Якщо елемент не знайдений, генерує виняток `ValueError`

```
1   x = [0, 1, 0, 2, 0, 3]
2   print("Заданий список   x: ", x)
3
4   x.remove(0)    # [1, 0, 2, 0, 3]
5   print("Результуючий список x: ", x)
```

```
1  x = [0, 1, 0, 2, 0, 3]
2  print("Заданий список   x: ", x)
3
4  x.remove(100)  # ValueError: list.remove(x): x not in list
   print("Результуючий список x: ", x)
5
```

На екрані:

```
---------------------------------------------------------------
--------------
ValueError                                Traceback (most recent call
last)
~\AppData\Local\Temp/ipykernel_11664/3758559791.py       in
<module>
      2 print("Заданий список   x: ", x)
      3
----> 4 x.remove(100)   #   ValueError: list.remove(x): x
not in list
      5 print("Результуючий список x: ", x)

ValueError: list.remove(x): x not in list
```

Метод `clear()`: видалення всіх елементів зі списку

```
1  x = [0, 1, 2]
2  print("Заданий список   x: ", x)
3
4  x.clear()       #  []
5  print("Результуючий список x: ", x)
```

Метод `pop([index])` видаляє та повертає елемент за індексом `index`.

Якщо індекс не передано, просто видаляє останній елемент.

```
1   n = 10
2   x = [i   for i in range(n)]    # x = [0, 1, 2, 3, 4, 5,
    6, 7, 8, 9]
3   print("Заданий список   x: ", x)
4
5   # Видалення елемента з індексом 4
6   x.pop(4)            # x = [0, 1, 2, 3,    5, 6, 7, 8, 9]
7   print("Отриманий список x: ", x)
8
9   # Видалення останнього елемента
10  x.pop()             # x = [0, 1, 2, 3,    5, 6, 7, 8   ]
11  print("Отриманий список x: ", x)
```

```
12
13  # Видалення елемента з індексом 2
14  index = x.pop(2)    # x = [0, 1,    3,    5, 6, 7, 8   ]
15  print("Отриманий список x: ", x)
16  print("Індекс видаленого із списку x елемента: ", index)
    # 2
```

Оператор `del`

```
del видалений_елемент або набір_елементів (зріз)
```

```
1   n = 8
2   x = [10*i for i in range(n)]    # x = [0, 10, 20, 30, 40,
    50, 60, 70]
3   print("Заданий список   x: ", x)
4
5   # Видалення елементу з індексом 2
6   del x[2]
7   print("Отриманий список x: ", x)   # x = [0, 10, 30, 40,
    50, 60, 70]
8
9   # Видалення елементів з індексами 0..2
10  del x[ : 3]
11  print("Отриманий список x: ", x)  # x = [40, 50, 60, 70]
12
13  # Видалення елементів з індексами від 2 до останнього
14  del x[2 : ]
15  print("Отриманий список x: ", x)    # x = [40, 50]
```

16.5. Пошук у списку індекса елементу: метод `index()`

`index(item[, start, stop])`: повертає індекс елемента `item`.

Пошук елемента `item` здійснюється серед елементів списку, починаючи з індексу `start` до `stop` включно.

Якщо початковий індекс не вказано, пошук здійснюється від початку списку.

Якщо елемент не знайдений, генерує виняток `ValueError`

```
1   #    0  1  2  3  4  5  6  7  -- індекси елементів списку
2   x = [1, 4, 3, 2, 5, 7, 3, 8]
3   print(x)
4
5   ind1 = x.index(3)            # 2
6   print("Індекс першої    3-ки у списку x: ",  ind1)
7
```

```
# пошук здійснюється в частині списку за 1-м входженням 3-ки до списку
ind2 = x.index(3, ind1+1)    # 6
print("Індекс наступної 3-ки у списку x: ",  ind2)
```

100 відсутнє у списку, генерується виняток `ValueError` та повідомлення `"100 is not in list"`

```
ind3 = x.index(100)   # ValueError: 100 is not in list
```

16.6. Перевірка наявності елемента: оператор `in`

Якщо певний елемент не знайдений, то методи `remove` та `index` генерують виняток.

Щоб уникнути подібної ситуації, перед операцією з елементом можна перевіряти його наявність (приналежність до списку) за допомогою ключового слова `in`:

```
people = ["Tom", "Bob", "Alice", "Sam"]

if "Alice" in people:
    people.remove("Alice")
print(people)       # ["Tom", "Bob", "Sam"]
```

Вираз `if "Alice" in people` повертає `True`, якщо елемент `"Alice"` є в списку `"people"`. Тому конструкція `if "Alice" in people` може виконати наступний блок інструкцій залежно від наявності елемента у списку.

16.7. Обчислення кількості входжень елемента до списку: метод `count()`

Метод `count(елемент)` обчислює, скільки разів у списку зустрічається елемент

```
x = [1, 4, 1, 2, 5, 1, 7, 1, 8]
print(*x)

count_1 = x.count(1)            # 4
print(f"Кол-во вхождений 1 в список x = {count_1}")

count_100 = x.count(100)        # 0
print(f"Кол-во вхождений 1 в список x = {count_100}")
```

16.8. Сортування: метод `sort()`, функція `sorted()`

Метод `sort()` сортує сам список

```
1  x = [0, -1, 3, 2]
2  print(x)
3
4  # Метод sort() сортує список за зростанням
5  x.sort()   #  [-1, 0, 2, 3]
6  print(x)
```

```
1  x = [0, -1, 3, 2]
2  print(x)
3
4  # Метод sort(reverse = True) сортує список за спаданням
5  x.sort(reverse = True)   #  [3, 2, 0, -1]
6  print(x)
```

В наступному прикладі метод сортує список, застосовуючи до елементів функцію `key`.

```
1  x = [7, 5, 8, 2, 11, 1, 14]
2  x.sort(key = lambda x: x % 2) # [8, 2, 14, 7, 5, 11, 1]
3  print(x)
```

На початку відсортованого списку парні, а потом – непарні елементи.

Функція `sorted()` не змінює сортований список, а всі відсортовані елементи поміщає в новий список, який повертається як результат.

sorted(list): сортує список list

sorted(list, key): сортує список list, застосовуючи до елементів функцію key

```
1  x = [0, -1, 3, 2]
2  print(x)
3
4  xx = sorted(x)
5  print(x)           #  [0, -1, 3, 2]
6  print(xx)          #  [-1, 0, 2, 3]
```

```
1  people = ["Tom", "bob", "alice", "Sam", "Bill"]
2
3  sorted_people = sorted(people, key = str.lower)
4  print(sorted_people)
   На екрані:
```

```
["alice", "Bill", "bob", "Sam", "Tom"]
```

16.9. Метод `reverse()`

змінює порядок прямування елементів на протилежний

```
x = [1, 20, 300]

x.reverse()     # [300, 20, 1]
print(x)
```

17. Фільтрація `filter()`

Фільтрація – ще один спосіб розв'язання завдання на "зробити вибірку"

`filter(функція_перевірки_наявності/відсутності_у_елемента_властивості, список)`

1 спосіб: написати функцію перевірки у елемента наявності/відсутності властивості

2 спосіб: використати lambda-вираз

```
# 1 спосіб: написати функцію перевірки у елемента наявності/відсутності властивості

def perevirka(value) :
    return  value % 3 == 0

##################################################
x = [12, 1, -9, -7, -10, -10, 17, 7, 7, 2]

# Фільтрація списку x – вибірка елементів, кратних 3
y = list( filter( perevirka, x) )   # [12, -9]
print(y)
```

```
# 2 спосіб: використати lambda-вираз
x = [12, 1, -9, -7, -10, -10, 17, 7, 7, 2]
print(x)

y = list( filter( lambda element: element % 3 == 0, x) )
print(y)
```

```
На екрані:
[12, 1, -9, -7, -10, -10, 17, 7, 7, 2]
[12, -9]
```

18. Проекція: `map()`

`map(fun, список)` застосовує функцію `fun` або `lambda`-вираз до всіх елементів списку

```
1  def square(value) :
2      return value ** 2
3
4  x = [1, 3, 5]
5  print(x)
6
7  # Застосування функції square (піднесення у ступінь 2)
   до всіх елементів списку x
8  y = list( map(square, x) )  # [1, 9, 25]
9  print(y)
```

```
1  # lambda-вирази
2  pow_2    = lambda x : x **2
3  is_odd   = lambda x : x % 2 != 0
4  muptiply = lambda x, y : x * y
```

```
5  n = 6
6  x = [i for i in range(n)]    #  [0, 1, 2, 3, 4, 5]
7  print(x)
```

```
8  xx1 = list(map(pow_2, x))    #  [0, 1, 4, 9, 16, 25]
9  print(xx1)
```

```
10 xx2 = list(map(is_odd, x))     # [False, True, False,
   True, False, True]
11 print(xx2)
```

```
12 print(x)                              # [0, 1, 2, 3, 4, 5]
13
14 y = [element+1 for element in x]  # [1, 2, 3, 4, 5, 6]
15 print(y)
16
17 xx3 = list(map(muptiply, x, y))  # [0, 2, 6, 12, 20, 30]
18 print(xx3)
```

Введення елементів списку з клавіатури

```
1  # 1 спосіб -- метод append
```

```
2
3   n = 4
4   x = []
5   for i in range(n) :
6       element = int(input("Введіть елемент списку: "))
7       x.append(element)
8   print(x)
```

```
На екрані:
Введіть елемент списку: 1
Введіть елемент списку: 3
Введіть елемент списку: 5
Введіть елемент списку: 9
[1, 3, 5, 9]
```

```
1   # 2 спосіб -- функція map
2   x = list( map(int, input("Введіть елементи списку через
    пробіл: ").split()))
3   print( type(x) )
4   print( x )
```

```
На екрані:
Введіть елементи списку через пробіл: 1 3 0
<class 'list'>
[1, 3, 0]
```

19. Завдання для самостійного розв'язку

№1. Напишіть програму, яка вимагатиме у користувача цілі числа і зберігати їх у вигляді списку.

Індикатором закінчення вводу повинен бути нуль.

Потім програма повинна вивести на екран усі введені користувачем числа (крім нуля) у порядку зростання – одне значення у рядку.

Не використовувати для сортування метод `sort` та функцію `sorted`.

Написати будь-яке сортування самостійно.

№2 У даній вправі розробити програму, в якій користувач вводить з клавіатури слова, доки він не залишить рядок введення порожнім.

Після цього на екрані мають бути показані слова, введені користувачем, але без повторів, кожне по одному разу. При цьому слова повинні бути відображені в тому порядку, в якому їх вводили з клавіатури.

Наприклад, якщо користувач на запит програми введе наступний спи-

сок слів:

```
first
second
first
third
second
```

програма має вивести:

```
first
second
third
```

№3 Власним дільником числа називається кожен його дільник, відмінний від числа.

Напишіть код, який будує список всіх дільників заданого числа.

№4. Користувач вводить рядок.

Видаліть розділові знаки (точки, коми, знаки оклику та знаки питання, тире, двокрапки та крапки із комами; дефіс і апостроф розділовими знаками не є)

№5. Решето Ератосфена – алгоритм, винайдений понад 2000 років, і служить для знаходження всіх простих чисел від 2 до деякого цілого числа n.

Опис алгоритму:

Виписуємо всі цілі числа від 0 до n. Викреслюємо 0 та 1 як непрості числа. Встановлюємо значення змінної p, що дорівнює 2. Поки p менше n, викреслюємо всі числа, кратні p, крім p. Наступне значення p дорівнює наступному невикресленому числу

Виводимо всі числа, що залишилися незакресленими

У комп'ютерній симуляції не варто «викреслювати» елемент шляхом його видалення зі списку – натомість краще присвоїти йому значення 0.

Після завершення алгоритму всі ненульові числа у списку і будуть простими.

Література

1. Креневич А.П. Python у прикладах і задачах. Частина 1. Структурне програмування: навч. посіб. з дисципліни "Інформатика та програмування". К.: ВПЦ "Київський Університет", 2017. 206 с.
2. Ceder Naomi. The Quick Python Book. Manning Publications. 2025. 585p.
3. Downey Allen B. Think Python. Boston: O`Reilly, 2016. 300p.
4. Slatkin Brett. Effective Python: 59 specific ways to write better Python. NY.: Addison-Wesley, 2015. 683p.
5. Shaw Zed A. Python 3. NY.: Addison-Wesley, 2019. 370p.