**Модуль 4: Структури даних**

**Колекції**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson04/lesson-04#%D0%BA%D0%BE%D0%BB%D0%B5%D0%BA%D1%86%D1%96%D1%97)

Колекція у Python — програмний об'єкт (змінна-контейнер), що зберігає набір значень одного або різних типів.

**Простою мовою**, колекція — це сховище інформації, каталог, побудований за якимось принципом.

Сховища бувають різними: може бути мішок, з якого ти дістаєш вміст навмання, може бути розкладений за абеткою каталог, в якому дуже легко щось знайти, або труба, з якої можна дістати тільки те, що лежить з краю тощо

Залежно від цілей, можна і треба застосовувати різні колекції. Наприклад, щоб швидко знайти щось, використовують словники; для зберігання впорядкованих даних, коли порядок має значення — списки та кортежі; для зберігання тільки унікальних елементів — множини; для зберігання символів — рядки тощо.

У цьому уроці ми вивчимо основні колекції Python і навчимося працювати з ними. Це основа мови і знання можливостей колекцій є не лише дуже корисною навичкою, але й визначальною для розробника на Python.

Основні властивості колекцій:

* Впорядкованість. Впорядкований контейнер (колекція) дає можливість звертатися до своїх елементів за індексом (номером) і гарантує, що порядок елементів зберігається.
* Змінність. Якщо колекція змінювана, то її вміст можна змінювати, не створюючи нову колекцію. Наприклад додавати, видаляти, заміняти елементи колекції.
* Унікальність. Унікальність стосується вмісту колекції, чи допускається зберігання однакових об'єктів.

**Списки**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson04/lesson-04#%D1%81%D0%BF%D0%B8%D1%81%D0%BA%D0%B8)

Список — **впорядкований змінюваний** контейнер даних. Списки не диктують розробнику тип даних, який можна помістити в цей контейнер, і можуть містити будь-які типи даних у будь-якому зручному порядку.

Для створення порожнього списку існує два способи:

my\_list = list()  
  
empty\_list = []

Щоб створити заповнений список:

not\_empty = [1, 2, 'user']

**Впорядковані контейнери**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson04/lesson-04#%D0%B2%D0%BF%D0%BE%D1%80%D1%8F%D0%B4%D0%BA%D0%BE%D0%B2%D0%B0%D0%BD%D1%96-%D0%BA%D0%BE%D0%BD%D1%82%D0%B5%D0%B9%D0%BD%D0%B5%D1%80%D0%B8)

**Доступ за індексом**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson04/lesson-04#%D0%B4%D0%BE%D1%81%D1%82%D1%83%D0%BF-%D0%B7%D0%B0-%D1%96%D0%BD%D0%B4%D0%B5%D0%BA%D1%81%D0%BE%D0%BC)

Під впорядкованістю варто розуміти властивість контейнера зберігати порядок елементів при роботі з ним, видаляючи елемент, додаючи новий, роблячи вставку/видалення з кінця/початку/середини, ви гарантуєте, що усі елементи, окрім тих, яких безпосередньо торкнулася операція, зберігають порядок.

Найкориснішою властивістю впорядкованості є можливість доступу до елементів контейнера за індексом цього елемента в контейнері.

У Python синтаксис доступу за індексом виглядає так:

some\_iterable = ["a", "b", "c"]  
first\_letter = some\_iterable[0]  
middle\_one = some\_iterable[1]  
last\_letter = some\_iterable[2]

У першому рядку ми створили список з трьох перших літер англійського алфавіту.

У другому рядку ми зберегли у змінну first\_letter літеру "a" — перший елемент some\_iterable. **Індекс** у Python починається з 0, як і в більшості мов програмування, та індексом "a" є 0.

Третій рядок — це звернення до другого елементу some\_iterable, його індекс дорівнює 1 — це літера "b" і ми зберігаємо її у middle\_one.

Четвертий рядок — це звернення до останнього елементу some\_iterable, літери "c", ми збережемо її у last\_letter і її індекс дорівнює 2.

Python підтримує індексування елементів з кінця. Для цього потрібно додати - і вказати номер елементу з кінця. Оскільки у Python -0 == 0, то перший елемент з кінця — це -1, другий — -2 і так далі. Наш приклад можна переписати, використовуючи індексування з кінця, ось так:

some\_iterable = ["a", "b", "c"]  
first\_letter = some\_iterable[-3]  
middle\_one = some\_iterable[-2]  
last\_letter = some\_iterable[-1]

Найкориснішою властивістю списків є змінність списків, ви можете змінити значення будь-якого елементу списку:

some\_iterable = ["a", "b", "c"]  
some\_iterable[1] = "Z"  
print(some\_iterable) *# ["a", "Z", "c"]*

В цьому прикладі ми змінили другий елемент списку some\_iterable (другий елемент — це елемент з індексом 1) на 'Z'.

**Зрізи у Python (Slice)**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson04/lesson-04#%D0%B7%D1%80%D1%96%D0%B7%D0%B8-%D1%83-python-slice)

Для впорядкованих контейнерів є спеціальний синтаксис, коли нам необхідно отримати деяку послідовність елементів з контейнера. Наприклад, якщо ми хочемо отримати перші 5 літер рядку:

some\_str = "This is awesome string"  
first\_five = some\_str[0:5]

first\_five в цьому прикладі буде містити рядок 'This '.

Синтаксис полягає у зазначенні індексу першого елементу зрізу, індексу, до якого (не включно) брати елементи в нову послідовність, та кроку, з яким брати елементи між початковим та кінцевим індексом у квадратних дужках, розділивши їх двокрапкою.

Візьмімо список чисел від 1 до 10 і збережемо окремо парні, не парні та кратні 3.

numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
  
odd\_numbers = numbers[0:9:2]  
even\_numbers = numbers[1:9:2]  
three\_numbers = numbers[2:9:3]

У odd\_numbers ми беремо числа, починаючи з індексу 0 до 10 з кроком 2 (отримаємо [1, 3, 5, 7, 9]). У even\_numbers ми беремо числа, починаючи з індексу 1 до 10 з кроком 2 (отримаємо [2, 4, 6, 8]) У three\_numbers ми беремо числа, починаючи з індексу 2 до 10 з кроком 3 (отримаємо [3, 6, 9]).

Ви можете не вказувати початковий, кінцевий індекс або крок, пропускаючи його. За замовчуванням Python візьме зріз з початку до останнього елемента з кроком 1. Перепишемо попередній приклад у скороченому синтаксисі:

numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
  
odd\_numbers = numbers[::2]  
even\_numbers = numbers[1::2]  
three\_numbers = numbers[2:9:3]  
  
numbers\_copy = numbers[:]

numbers\_copy в цьому прикладі — це зріз, який бере всі елементи numbers від початку і до кінця з кроком 1.

Також важливо пам'ятати, що в зріз не входить елемент з індексом, до якого брати елементи.

numbers = [0, 1, 2, 3]  
first\_three = numbers[0:3] *# [0, 1, 2]*

В цьому прикладі елемент з індексом 3 не увійде у first\_three.

**Використання методів об'єктів**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson04/lesson-04#%D0%B2%D0%B8%D0%BA%D0%BE%D1%80%D0%B8%D1%81%D1%82%D0%B0%D0%BD%D0%BD%D1%8F-%D0%BC%D0%B5%D1%82%D0%BE%D0%B4%D1%96%D0%B2-%D0%BE%D0%B1%D1%94%D0%BA%D1%82%D1%96%D0%B2)

Об'єкт в програмуванні — деяка сутність у цифровому просторі, що має певний стан і поведінку, має певні властивості (атрибути) та операції над ними (методи).

Доступ до методів об'єктів у Python синтаксично відбувається за допомогою символу крапки після імені об'єкту і зазначення імені методу або атрибуту, до якого потрібно отримати доступ.

numbers = ['a', 'b']  
numbers.append('c')  
print(numbers) *# ['a', 'b', 'c']*

В цьому прикладі ми використали метод append, який є у списків і він є у списку numbers. Цей метод додає новий елемент в кінець списку. Як аргумент цей метод отримує елемент, який потрібно додати до списку. Аргументи вказуються в дужках.

Якщо метод не вимагає аргументів (наприклад метод clear), то дужки будуть порожніми:

num = [1, 2]  
num.clear()  
print(num) *# []*

**Методи списків**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson04/lesson-04#%D0%BC%D0%B5%D1%82%D0%BE%D0%B4%D0%B8-%D1%81%D0%BF%D0%B8%D1%81%D0%BA%D1%96%D0%B2)

* Додавання елементу в кінець списку: my\_list.append(element)

chars = ['a', 'b']  
chars.append('c')  
print(chars) *# ['a', 'b', 'c']*

* видалення елементу зі списку викличе помилку, якщо такого елементу немає в списку: my\_list.remove(element)

chars = ['a', 'b']  
chars.remove('b')  
print(chars) *# ['a']*

* Повернути i-ий елемент та видалити його зі списку i\_element = my\_list.pop(i). За замовчуванням i = -1

chars = ['a', 'b']  
last = chars.pop(1)  
print(chars) *# ['a']*  
print(last) *# 'b'*

* Розширити список a\_list елементами з b\_list: a\_list.extend(b\_list)

chars = ['a', 'b']  
numbers = [1, 2]  
  
chars.extend(numbers)  
print(chars) *# ['a', 'b', 1, 2]*

* Вставити x на позицію з індексом i: my\_list.insert(i, x)

chars = ["a", "b"]  
chars.insert(1, "c")  
print(chars) *# ['a', 'c', 'b']*

* Очистити список: my\_list.clear()

chars = ['a', 'b']  
last = chars.clear() print(chars) *# []*

* Знайти індекс першого елемента у списку, що дорівнює x: index = my\_list.index(x)

chars = ['a', 'b', 'c', 'd']  
c\_ind = chars.index('c') print(c\_ind) *#2*

* Повернути кількість елементів у списку, що дорівнюють x: x\_number = my\_list.count(x)

chars = ['a', 'b', 'c', 'a']  
a\_count = chars.count('a')  
print(a\_count) *# 2*

* Відсортувати список за зростанням: my\_list.sort(key=None, reverse=False)

chars = ['z', 'a', 'b']  
chars.sort()  
print(chars) *# ['a', 'b', 'z']*

* Змінити порядок елементів у списку на зворотний: my\_list.reverse()

chars = ['a', 'b']  
chars.reverse()  
print(chars) *# ['b', 'a']*

* Повернути копію списку: copy\_of\_my\_list = my\_list.copy()

chars = ['a', 'b']  
chars\_copy = chars.copy()  
chars == chars\_copy *# True*

**Словники**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson04/lesson-04#%D1%81%D0%BB%D0%BE%D0%B2%D0%BD%D0%B8%D0%BA%D0%B8)

Словник — це контейнер, який зберігає пари ключ-значення. Ключем може бути будь-який незмінний тип даних Python (число, рядок, кортеж тощо). Неможливо використовувати у якості ключа списки, словники, множини або будь-які інші змінювані типи даних. Значенням словника може бути будь-який тип даних Python, включаючи призначені для користувача типи.

**Методи словників**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson04/lesson-04#%D0%BC%D0%B5%D1%82%D0%BE%D0%B4%D0%B8-%D1%81%D0%BB%D0%BE%D0%B2%D0%BD%D0%B8%D0%BA%D1%96%D0%B2)

Деякі методи словників, що найчастіше використовуються:

* pop(key) — повертає значення елементу і видаляє пару ключ-значення зі словника

chars = {'a': 1, 'b': 2}  
b\_num = chars.pop('b')  
print(chars) *# {'a': 1}*  
print(b\_num) *# 2*

* update(another\_dict) — розширює словник значеннями з іншого словника

chars = {'a': 1, 'b': 2}  
chars.update({"c": 3})  
print(chars) *# {'a': 1, 'b': 2, "c": 3}*

* clear() — очищає словник, не створюючи нового

chars = {'a': 1, 'b': 2}  
chars.clear()  
print(chars) *# {}*

* copy() — повертає копію словника

chars = {'a': 1, 'b': 2}  
chars\_copy = chars.copy()  
chars\_copy == chars *# True*

* get(key[, default]) — не викликає виключення, якщо ключа немає в словнику, повертає default, за замовчуванням default=None.

chars = {'a': 1, 'b': 2}  
c\_idx = chars.get('c', -1)  
print(c\_idx) *# -1*

**Цикли та словники**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson04/lesson-04#%D1%86%D0%B8%D0%BA%D0%BB%D0%B8-%D1%82%D0%B0-%D1%81%D0%BB%D0%BE%D0%B2%D0%BD%D0%B8%D0%BA%D0%B8)

Ітерування за словником — це блок коду, що дуже часто зустрічається, і корисно вміти це робити.

Спершу варто сказати, що словник сам по собі — це ітерований контейнер і за ним можна ітеруватися в циклі for без необхідності заводити якийсь зовнішній лічильник тощо. Створимо словник, в якому ключами будуть числа, а значеннями — числівники англійською:

numbers = {  
 1: "one",  
 2: "two",  
 3: "three"  
}

Тепер давайте просто пройдемо словником в циклі та виведемо, що нам повертає ітератор на кожній ітерації:

for key in numbers:  
 print(key)

У виведенні ви побачите:

1  
2  
3

Ітеруючи за словником, ви перебираєте ключі словника. Таку саму поведінку можна отримати, використовуючи метод keys, але так ви явно вкажете, що хочете перебрати ключі:

for key in numbers.keys():  
 print(key)

Відповідь буде точно такою самою:

1  
2  
3

Часто необхідно перебрати саме значення словника, для цього скористаємося методом values:

for val in numbers.values():  
 print(val)

У виведенні буде:

one  
two  
tree

І переберемо пари ключ значення, використовуючи метод items. На кожній ітерації ми отримаємо пару (ключ, значення):

for key, value in numbers.items():  
 print(key, value)

Виведення:

1 one  
2 two  
3 three

Що **не можна** робити, поки ітеруєтеся за словником: не можна видаляти елементи із словника, не можна додавати елементи у словник. Але можна перезаписувати значення, якщо ви ітеруєтеся за ключами.

**Множини**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson04/lesson-04#%D0%BC%D0%BD%D0%BE%D0%B6%D0%B8%D0%BD%D0%B8)

Множини — це неврегульований контейнер, який містить тільки унікальні елементи. У множину можна додавати тільки незмінні типи даних.

Є тільки один спосіб створити порожню множину:

a = set()  
print(a) *# set()*

Для створення заповненої множини достатньо передати будь-який об'єкт, що ітерується, в функцію set:

a = set('hello')  
print(a) *# {'e', 'h', 'l', 'o'}*

Або ж скористатися синтаксисом з фігурними дужками (як у словниках), але елементи у фігурних дужках просто перелічити через кому без двокрапок:

b = {1, 2, 3, 4}

Унікальність має на увазі, що якщо множина вже містить такий елемент, то спроба додати ще один такий самий нічого не змінить.

numbers = {1, 2, 3, 1, 2, 3}  
print(numbers) *# {1, 2, 3}*

**Методи множин**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson04/lesson-04#%D0%BC%D0%B5%D1%82%D0%BE%D0%B4%D0%B8-%D0%BC%D0%BD%D0%BE%D0%B6%D0%B8%D0%BD)

Множини підтримують наступні методи:

* add(elem) — додає елемент в множину

numbers = {1, 2, 3}  
numbers.add(4)  
print(numbers) *# {1, 2, 3, 4}*

* remove(elem) — видаляє елемент з множини, викликає виняток, якщо такого елементу немає

numbers = {1, 2, 3}  
numbers.remove(3)  
print(numbers) *# {1, 2}*

* discard(elem) — видаляє елемент з множини і не викликає виняток, якщо його немає

numbers = {1, 2, 3}  
numbers.discard(2)  
print(numbers) *# {1, 3}*

**Математичні операції над множинами**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson04/lesson-04#%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D1%87%D0%BD%D1%96-%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D1%96%D1%97-%D0%BD%D0%B0%D0%B4-%D0%BC%D0%BD%D0%BE%D0%B6%D0%B8%D0%BD%D0%B0%D0%BC%D0%B8)

Давайте детальніше розглянемо, які корисні математичні операції можна робити над множинами. Спершу створимо множини a та b:

a = set('hello')  
print(a) *# {'e', 'h', 'l', 'o'}*  
  
b = set('hi there!')  
print(b) *# {'r', ' ', 'i', 'e', '!', 'h', 't'}*

Щоб знайти загальні елементи для двох множин, виконаємо над ними операцію & (AND):

a & b *# {'e', 'h'}*

Знайдемо усі елементи з двох множин, окрім загальних, за допомогою оператора ^ (XOR):

a ^ b *# {' ', '!', 'i', 'l', 'o', 'r', 't'}*

Об'єднання множин, або просто усі елементи з обох множин знаходяться за допомогою оператора | (OR):

a | b *# {' ', '!', 'e', 'h', 'i', 'l', 'o', 'r', 't'}*

Множини — це дуже потужний інструмент, коли необхідно знайти унікальні елементи в якомусь наборі і прибрати дублікати. Множини — це також найшвидший спосіб знайти загальні або відмінні елементи з декількох наборів.

**Використання кортежів як ключів**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson04/lesson-04#%D0%B2%D0%B8%D0%BA%D0%BE%D1%80%D0%B8%D1%81%D1%82%D0%B0%D0%BD%D0%BD%D1%8F-%D0%BA%D0%BE%D1%80%D1%82%D0%B5%D0%B6%D1%96%D0%B2-%D1%8F%D0%BA-%D0%BA%D0%BB%D1%8E%D1%87%D1%96%D0%B2)

Незмінність кортежів обмежує їх застосування, у порівнянні зі списками, але дає можливість використати кортежі як ключі для словника, або елементи множини. Наприклад, розглянемо набір точок на площині (кортежі). Їх можна використовувати як ключі у словнику:

points = {  
 (0, 0): "O"  
 (1, 1): "A"  
 (2, 2): "B"  
}

У словнику points використовуються кортежі з координатами точок як ключі, а значення — це імена (назва) точок.

При спробі змінити кортеж ви отримаєте повідомлення про помилку:

not\_empty[3] = 's'  
---------------------------------------------------------------------------  
TypeError Traceback (most recent call last)  
<ipython-input-6-0e33ea041f91> in <module>()  
----> 1 not\_empty[3] = 's'  
  
TypeError: 'tuple' object does not support item assignment

Є одна важлива особливість створення кортежів — це створення кортежів з одним елементом. Річ у тому, що Python використовує круглі дужки як математичні символи. Якщо написати вираз на кшталт a = (1), виникає невизначеність. Інтерпретатор розуміє такий вираз, як суто математичний, і просто прибирає зайві дужки, присвоюючи a значення 1. Щоб вказати, що це не математична операція, а саме кортеж, потрібно додати кому після значення. Тоді інтерпретатор однозначно зрозуміє, що ви хочете створити кортеж з одним елементом:

a = (1, )

**Рядки**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson04/lesson-04#%D1%80%D1%8F%D0%B4%D0%BA%D0%B8)

Робота з текстовими даними у Python реалізована через str-об'єкти або рядки.  
**Рядок** — це незмінна впорядкована послідовність символів в деякому кодуванні. За замовчуванням використовується кодування UTF-8, але можна працювати майже з усіма відомими таблицями кодування символів. Для того щоб створити змінну типу "рядок", необхідно певний набір символів взяти в лапки.

* Варіант 1. Одинарні лапки (апостроф) 'some text'
* Варіант 2. Подвійні лапки "some text".

Різні варіанти використання лапок обумовлені тим, що при використанні одинарних лапок, можна в рядку вказати подвійні і навпаки.

game\_string = 'My "Game"'

**Впорядкована** послідовність означає, що до елементів рядку можна звертатися за індексом:

s = "Hello world!"  
print(s[0]) *# H*  
print(s[-1]) *# !*

**Незмінна** послідовність означає, що якщо рядок вже створений, то змінити його не можна, можна тільки створити новий.

s = "Hello world!"  
s[0] = "Q" *# Тут буде викликано виняток (помилка) TypeError*  
**Малі методи**

Для того, щоб усі літери рядка перевести у верхній регістр, використовується метод upper:

s = "Hello"  
s.upper()  
print(s) *# Виведе 'HELLO'*

Для переведення в нижній регістр використовується метод lower():

s = "Some Text"  
print(s.lower()) *# Виведе 'some text'*

Щоб перевірити, що рядок починається з підрядка, є метод startswith:

s = "Bill Jons"  
print(s.startswith("Bi")) *# Виведе True*

Щоб перевірити, що рядок закінчується підрядком, використовується метод endswith:

s = "hello.jpg"  
print(s.endswith("jpg")) *# Виведе True*

Цей метод зручно використовувати для перевірки розширення файлів.

**Загальні для усіх колекцій операції**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson04/lesson-04#%D0%B7%D0%B0%D0%B3%D0%B0%D0%BB%D1%8C%D0%BD%D1%96-%D0%B4%D0%BB%D1%8F-%D1%83%D1%81%D1%96%D1%85-%D0%BA%D0%BE%D0%BB%D0%B5%D0%BA%D1%86%D1%96%D0%B9-%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D1%96%D1%97)

Такі різні списки, кортежі, словники, множини і рядки об'єднують в одну групу — колекції, тому що в них є загальні властивості:

* перевірка на входження;
* кількість елементів;
* перебір усіх елементів в циклі for.

**Перевірка на входження**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson04/lesson-04#%D0%BF%D0%B5%D1%80%D0%B5%D0%B2%D1%96%D1%80%D0%BA%D0%B0-%D0%BD%D0%B0-%D0%B2%D1%85%D0%BE%D0%B4%D0%B6%D0%B5%D0%BD%D0%BD%D1%8F)

Будь-яка колекція дозволяє перевірити, чи містить колекція цей елемент (чи є там такий самий). Для цього використовується оператор in.

Наприклад, перевірка на те, що користувач не використовує простий пароль і в паролі не зустрічається послідовність "qwerty" або "123":

password = "qwerty123"  
if "qwerty" in password or "123" in password:  
 print("This password is too weak!")

Оператор in перевіряє наявність елементу ('qwerty', '123') у контейнері (password) та повертає True або False.

prime\_numbers = {2, 3, 5, 7, 11, 13, 17, 19, 23}  
is\_prime = 3 in prime\_numbers

Перевірка на те, що число 3 присутнє у множині перших дев'яти простих чисел prime\_numbers.

Із словниками перевірка на входження перевіряє, що елемент присутній серед ключів словника.

user = {  
 "name": "Bill",  
 "surname": "Bosh",  
 "age": 22  
}  
  
if "age" in user:  
 print(f"User is {user['age']} years old.")

**Кількість елементів**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson04/lesson-04#%D0%BA%D1%96%D0%BB%D1%8C%D0%BA%D1%96%D1%81%D1%82%D1%8C-%D0%B5%D0%BB%D0%B5%D0%BC%D0%B5%D0%BD%D1%82%D1%96%D0%B2)

Будь-яка колекція дозволяє дізнатися кількість елементів у ній за допомогою функції len.

password = input("Password: ")  
if len(password) < 8:  
 print("Your password is too short")

Наприклад, перевірка довжини пароля введеного користувача може бути реалізована за допомогою функції len.

**Перебір усіх елементів колекції в циклі for**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson04/lesson-04#%D0%BF%D0%B5%D1%80%D0%B5%D0%B1%D1%96%D1%80-%D1%83%D1%81%D1%96%D1%85-%D0%B5%D0%BB%D0%B5%D0%BC%D0%B5%D0%BD%D1%82%D1%96%D0%B2-%D0%BA%D0%BE%D0%BB%D0%B5%D0%BA%D1%86%D1%96%D1%97-%D0%B2-%D1%86%D0%B8%D0%BA%D0%BB%D1%96-for)

alphabet = "abcdefghijklmnopqrstuvwxyz"  
for char in alphabet:  
 print(char)

По будь-якій колекції можна пройти за допомогою циклу for і на кожній ітерації в циклі буде отриманий один з елементів цієї колекції. У прикладі літери алфавіту з alphabet виводяться в консоль по черзі в циклі for.

Давайте проітеруємо список some\_iterable у циклі та виведемо в консоль, що ми отримуємо на кожній ітерації:

some\_iterable = ["a", "b", "c"]  
  
for i in some\_iterable:  
 print(i)

У консолі ми побачимо:

a  
b  
c

odd\_numbers = [1, 3, 5, 7, 9]  
for i in odd\_numbers:  
 print(i \*\* 2)

Код з цього прикладу виведе в консоль квадрати перших п'яти непарних чисел.

**Робота з файловою системою, пакет pathlib**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson04/lesson-04#%D1%80%D0%BE%D0%B1%D0%BE%D1%82%D0%B0-%D0%B7-%D1%84%D0%B0%D0%B9%D0%BB%D0%BE%D0%B2%D0%BE%D1%8E-%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%BE%D1%8E-%D0%BF%D0%B0%D0%BA%D0%B5%D1%82-pathlib)

Для роботи з файловою системою в стандартному постачанні Python йде пакет pathlib, він містить безліч корисних функцій для роботи з файловою системою. Детальний перелік можливостей pathlib рекомендуємо вивчити на [сторінці офіційної документації](https://docs.python.org/3/library/pathlib.html).

Основний інструмент у pathlib — Path, об'єкт, який є шляхом (адреса у файловій системі). В основному робота з файловою системою ведеться через Path. Path варто сприймати як вказівку на файл або папку. Щоб створити такий Path, достатньо викликати Path як функцію та передати у якості аргументу рядок-адресу у файловій системі:

from pathlib import Path  
  
p = Path('/home/user/Downloads') *# p Вказує на папку /home/user/Downloads*

Можна викликати Path без аргументів, тоді ви отримаєте вказівник на папку, в якій ви зараз знаходитеся.

from pathlib import Path  
  
p = Path() *# p Вказує на папку, з якої був запущений Python*

У Path є ряд корисних методів та атрибутів:

* p.parent вказує на батьківську папку;
* p.name повертає лише ім'я (рядок) папки або файлу, на який вказує p;
* p.suffix повертає рядком розширення файлу, на який вказує p, починаючи з крапки;

p = Path('setup.py')  
p.suffix *# '.py'*

* p.exists() повертає True або False, залежно від того, чи існує такий файл або папка;
* p.is\_dir() повертає True, якщо p вказує на папку, та False, якщо на файл, або такий шлях не існує;
* p.is\_file() повертає True, якщо p вказує на файл, та False, якщо на папку, або такий шлях не існує;
* p.iterdir() повертає ітератор за всіма файлам та папками всередині папки p;

from pathlib import Path  
  
p = Path('/home/user/Downloads') *# p Вказує на папку /home/user/Downloads*  
for i in p.iterdir():  
 print(i.name) *# Виведе у циклі імена всіх папок та файлів у /home/user/Downloads*

Варто розуміти, що i в цьому прикладі також будуть об'єктами Path, але вказувати вони будуть на файли та папки всередині '/home/user/Downloads'.

З концепцією об'єктів ми познайомимося дещо пізніше, поки достатньо знати, що i — це як і p теж вказівники на папки і файли, але кожен i вказує на якийсь свій файл або папку.

**Обробка аргументів командного рядка**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson04/lesson-04#%D0%BE%D0%B1%D1%80%D0%BE%D0%B1%D0%BA%D0%B0-%D0%B0%D1%80%D0%B3%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D1%96%D0%B2-%D0%BA%D0%BE%D0%BC%D0%B0%D0%BD%D0%B4%D0%BD%D0%BE%D0%B3%D0%BE-%D1%80%D1%8F%D0%B4%D0%BA%D0%B0)

Коли запускаємо збережений у файлі Python скрипт, є можливість передати йому при запуску якісь аргументи, як у функцію. Тоді наш скрипт може ці аргументи прийняти і якось змінити свою поведінку. Зробити це можна за допомогою пакета sys, в якому є список argv, де з'являються всі аргументи, з якими був запущений скрипт.

Цікавою особливістю sys.argv є те, що першим елементом цього списку буде назва самого файлу скрипту. Всі аргументи будуть у sys.argv у вигляді рядків в тому самому порядку, в якому вони були передані під час виклику. Python розділяє аргументи пробілами і в sys.argv пробіли не потрапляють.

Щоб зрозуміти як працює sys.argv, можете зробити простий скрипт echo.py, який буде виводити у консоль всі передані при виклику аргументи.

import sys  
  
for arg in sys.argv:  
 print(arg)

Якщо викликати скрипт з таким вмістом (припустимо він називається echo.py) командою: python echo.py test --user -hello some text, то у консолі ви побачите:

echo.py  
test  
--user  
-hello  
some  
text

Таким чином, якщо вам потрібно обробити перший аргумент запуску:

import sys  
  
  
def main():  
 print(sys.argv[1])

Під час виклику такого скрипту командою python echo.py 123 ви побачите у консолі 123. Зверніть увагу, що всі елементи списку sys.argv — це рядки. Якщо ви очікуєте, що користувач повинен ввести число (ціле або дробове), то вам потрібно перетворити рядок в потрібний вам тип самостійно.