Зміст

Рядки Python

Рядки

Рядки в Python оточені або одинарними, або подвійними лапками.

```
"привіт" те саме, що 'привіт'
```

Ви можете відобразити рядковий літерал за допомогою `print()` функції:

```
print("Hello")
print('Hello')
```

Цитати всередині цитат

Ви можете використовувати лапки всередині рядка, якщо вони не збігаються з лапками, що оточують рядок:

```
print("It's alright")
print("He is called 'Johnny'")
print('He is called "Johnny"')
```

Призначити рядок змінній

Призначення рядка змінній виконується за допомогою імені змінної, після якого йде знак рівності та рядок:

```
a = "Hello"
print(a)
```

Багаторядкові рядки

Ви можете призначити багаторядковий рядок змінній, використовуючи три лапки:

```
# Використання трьох подвійних лапок
a = """Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua."""
print(a)

# Використання трьох одинарних лапок
a = '''Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.'''
print(a)
```

> **Примітка:** У результаті розриви рядків вставляються в ту саму позицію, що й у коді.

Рядки як масиви

Рядки в Python являють собою масиви байтів, що представляють символи Unicode. Окремий символ є просто рядком довжиною 1.

Доступ до елементів рядка

```
a = "Hello, World!"
print(a[1])  # Отримайте символ у позиції 1
```

Цикл через рядок

```
for x in "banana":
    print(x)
```

Довжина рядка

```
a = "Hello, World!"
print(len(a))  # Функція len() повертає довжину рядка
```

Перевірка рядка

Перевірка наявності

```
txt = "The best things in life are free!"
print("free" in txt)  # Перевірка наявності слова "free"

if "free" in txt:
    print("Yes, 'free' is present.")
```

Перевірка відсутності

```
txt = "The best things in life are free!"
print("expensive" not in txt)  # Перевірка відсутності слова "expensive"

if "expensive" not in txt:
    print("No, 'expensive' is NOT present.")
```

Нарізка рядків

Основи нарізки

```
b = "Hello, World!"
print(b[2:5])  # Символи з позиції 2 до 5 (не включено)
```

Нарізка з початку

```
print(b[:5])  # Символи від початку до позиції 5 (не включено)
```

Нарізка до кінця

```
print(b[2:])  # Символи з позиції 2 і до кінця
```

Негативна індексація

```
print(b[-5:-2])  # Символи від "o" до "d" (не включено)
```

Зміна рядків

Верхній і нижній регістр

```
a = "Hello, World!"
print(a.upper())  # Верхній регістр
print(a.lower())  # Нижній регістр
```

Видалення пробілів

```
a = " Hello, World! "
print(a.strip())  # Видаляє пробіли з початку та кінця
```

Замінити рядок

```
a = "Hello, World!"
print(a.replace("H", "J"))  # Замінює "H" на "J"
```

Розділити рядок

```
a = "Hello, World!"
print(a.split(","))  # Розбиває рядок на ['Hello', ' World!']
```

Конкатенація рядків

```
a = "Hello"
b = "World"
c = a + b
print(c)  # Об'єднання без пробілу

c = a + " " + b
print(c)  # Об'єднання з пробілом
```

Python – Форматування рядків

Форматування рядків

У Python не можна поєднувати рядки та числа таким чином:

```
age = 36
txt = "My name is John, I am " + age
```

```
print(txt)
```

Однак можна комбінувати рядки та числа за допомогою **f-рядків** або методу `format()`.

F-рядки

F-рядки були представлені в Python 3.6 і є рекомендованим способом форматування рядків.

Щоб створити f-рядок, додайте літеру `f` перед рядковим літералом і використовуйте фігурні дужки `{}` як заповнювачі для змінних або виразів.

Приклад: створення f-рядка:

```
age = 36
txt = f"My name is John, I am {age}"
print(txt)
```

Заповнювачі та модифікатори

Заповнювачі можуть містити змінні, операції, функції та модифікатори для форматування значень.

Приклад: додавання заповнювача для змінної `price`:

```
price = 59
txt = f"The price is {price} dollars"
print(txt)
```

Заповнювач може також містити модифікатор для форматування значення. Наприклад, `.2f` означає число з фіксованою комою з 2 десятковими знаками.

Приклад: відображення ціни з 2 десятковими знаками:

```
price = 59
txt = f"The price is {price:.2f} dollars"
print(txt)
```

Заповнювач може містити Python-код, наприклад, математичні операції:

Приклад: виконання математичної операції в заповнювачі:

```
txt = f"The price is {20 * 59} dollars"
print(txt)
```

Python – Ескейп-символи

Ескейп-символи

Щоб вставити недопустимі символи в рядок, використовуйте **ескейп-символ** `\`.

Наприклад, подвійні лапки всередині рядка, оточеного подвійними лапками, викликають помилку:

```
txt = "We are the so-called "Vikings" from the north."
```

Щоб уникнути помилки, використовуйте ескейп-символ `\"`:

Приклад: використання ескейп-символу:

```
txt = "We are the so-called \"Vikings\" from the north."
```

Інші ескейп-символи

Код	Результат
`\'`	Одинарна лапка
`\\`	Зворотна коса риска
`\n`	Новий рядок
`\r`	Повернення каретки
`\t`	Табуляція
`\b`	Backspace
`\f`	Form Feed
`\ooo`	Вісімкове значення
`\xhh`	Шістнадцяткове значення

Рядкові методи

Python має набір вбудованих методів для роботи з рядками. Усі методи повертають нові значення, не змінюючи оригінальний рядок.

Метод	Опис
`capitalize()`	Перетворює перший символ у верхній регістр
`casefold()`	Перетворює рядок у нижній регістр
`center()`	Повертає вирівняний по центру рядок

Метод	Опис
`count()`	Повертає кількість входжень значення
`encode()`	Повертає закодовану версію рядка
`endswith()`	Перевіряє, чи закінчується рядок певним значенням
`expandtabs()`	Встановлює розмір табуляції
`find()`	Шукає значення та повертає його позицію
`format()`	Форматує значення в рядку
`format_map()`	Форматує значення в рядку
`index()`	Шукає значення та повертає його позицію
`isalnum()`	Перевіряє, чи всі символи є алфавітно-цифровими
`isalpha()`	Перевіряє, чи всі символи є літерами
`isascii()`	Перевіряє, чи всі символи є ASCII
`isdecimal()`	Перевіряє, чи всі символи є десятковими
`isdigit()`	Перевіряє, чи всі символи є цифрами
`isidentifier()`	Перевіряє, чи є рядок ідентифікатором
`islower()`	Перевіряє, чи всі символи в нижньому регістрі
`isnumeric()`	Перевіряє, чи всі символи є числовими
`isprintable()`	Перевіряє, чи всі символи є друкованими
`isspace()`	Перевіряє, чи всі символи є пробілами
`istitle()`	Перевіряє, чи рядок відповідає правилам заголовка
`isupper()`	Перевіряє, чи всі символи у верхньому регістрі
`join()`	Об'єднує елементи ітерації в рядок
`ljust()`	Повертає вирівняний ліворуч рядок
`lower()`	Перетворює рядок у нижній регістр
`lstrip()`	Видаляє пробіли зліва
`maketrans()`	Повертає таблицю перекладу
`partition()`	Розділяє рядок на три частини
`replace()`	Замінює значення в рядку
`rfind()`	Шукає значення та повертає останню позицію
`rindex()`	Шукає значення та повертає останню позицію
`rjust()`	Повертає вирівняний праворуч рядок

Метод	Опис
`rpartition()`	Розділяє рядок на три частини
`rsplit()`	Розділяє рядок за вказаним роздільником
`rstrip()`	Видаляє пробіли справа
`split()`	Розділяє рядок за вказаним роздільником
`splitlines()`	Розділяє рядок за розривами рядків
`startswith()`	Перевіряє, чи починається рядок з певного значення
`strip()`	Видаляє пробіли з обох боків
`swapcase()`	Змінює регістр символів
`title()`	Перетворює перший символ кожного слова у верхній регістр
`translate()`	Повертає перекладений рядок
`upper()`	Перетворює рядок у верхній регістр
`zfill()`	Додає нулі на початок рядка

Булеві значення

У програмуванні часто потрібно знати, чи є вираз істинним (True) або хибним (False).

Ви можете оцінити будь-який вираз у Python і отримати одну з двох відповідей: True або False.

Коли ви порівнюєте два значення, вираз оцінюється, і Python повертає булеву відповідь:

Приклад

```
print(10 > 9)
print(10 == 9)
print(10 < 9)
```

Коли ви запускаєте умову в операторі `if`, Python повертає True або False:

Приклад

Виведіть повідомлення залежно від того, чи умова є True або False:

```
a = 200
b = 33

if b > a:
    print("b більше за a")
else:
    print("b не більше за a")
```

Оцінка значень і змінних

Функція `bool()` дозволяє оцінити будь-яке значення та повертає True або False.

Приклад
Оцініть рядок і число:

```
print(bool("Hello"))
print(bool(15))
```

Приклад
Оцініть дві змінні:

```
x = "Hello"
y = 15

print(bool(x))
print(bool(y))
```

Більшість значень є True

Майже будь-яке значення оцінюється як True, якщо воно має певний вміст.

Будь-який рядок є True, окрім порожніх рядків.

Будь-яке число є True, окрім 0.

Будь-який список, кортеж, множина та словник є True, окрім порожніх.

Приклад
Наступне поверне True:

```
bool("abc")
bool(123)
bool(["apple", "cherry", "banana"])
```

Деякі значення є False

Фактично, не так багато значень оцінюються як False, окрім порожніх значень, таких як `()`, `[]`, `{}`, `""`, число `0` і значення `None`. І, звісно, значення `False` оцінюється як False.

Приклад
Наступне поверне False:

```
bool(False)
bool(None)
bool(0)
bool("")
bool(())
bool([])
bool({})
```

Ще одне значення, або об'єкт у цьому випадку, оцінюється як False, якщо це об'єкт, створений із класу з функцією `__len__`, яка повертає 0 або False:

Приклад

```
class myclass():
    def __len__(self):
        return 0

myobj = myclass()
print(bool(myobj))
```

Функції можуть повертати булеві значення

Ви можете створювати функції, які повертають булеве значення:

Приклад
Виведіть відповідь функції:

```
def myFunction():
    return True

print(myFunction())
```

Ви можете виконувати код залежно від булевої відповіді функції:

Приклад
Виведіть "YES!", якщо функція повертає True, інакше виведіть "NO!":

```
def myFunction():
    return True

if myFunction():
    print("YES!")
else:
    print("NO!")
```

Python також має багато вбудованих функцій, які повертають булеве значення, наприклад, функція `isinstance()`, яка може використовуватися для визначення, чи є об'єкт певного типу даних:

Приклад

Перевірте, чи є об'єкт цілим числом:

```
x = 200
print(isinstance(x, int))
```

Оператори Python

Оператори використовуються для виконання операцій над змінними та значеннями.

У прикладі нижче ми використовуємо оператор `+`, щоб додати два значення:

Приклад

```
print(10 + 5)
```

Python поділяє оператори на наступні групи:

- Арифметичні оператори
- Оператори присвоєння
- Оператори порівняння
- Логічні оператори
- Оператори ідентичності
- Оператори членства
- Побітові оператори

Арифметичні оператори Python

Арифметичні оператори використовуються з числовими значеннями для виконання стандартних математичних операцій:

Оператор	Назва	Приклад
`+`	Додавання	`x + y`
`-`	Віднімання	`x - y`
`*`	Множення	`x * y`
`/`	Ділення	`x / y`
`%`	Остача від ділення	`x % y`
`**`	Піднесення до степеня	`x ** y`

/

Оператор	Назва	Приклад
`//`	Цілочисельне ділення	`x // y`

Оператори присвоєння Python

Оператори присвоєння використовуються для присвоєння значень змінним:

Оператор	Приклад	Те саме, що й
`=`	`x = 5`	`x = 5`
`+=`	`x += 3`	`x = x + 3`
`-=`	`x -= 3`	`x = x - 3`
`*=`	`x *= 3`	`x = x * 3`
`/=`	`x /= 3`	`x = x / 3`
`%=`	`x %= 3`	`x = x % 3`
`//=`	`x //= 3`	`x = x // 3`
`**=`	`x **= 3`	`x = x ** 3`
`&=`	`x &= 3`	`x = x & 3`
`|=`	`x |= 3`	`x = x | 3`
`^=`	`x ^= 3`	`x = x ^ 3`
`>>=`	`x >>= 3`	`x = x >> 3`
`<<=`	`x <<= 3`	`x = x << 3`
`:=`	`print(x := 3)`	`x = 3`

Оператори порівняння Python

Оператори порівняння використовуються для порівняння двох значень:

Оператор	Назва	Приклад
`==`	Рівність	`x == y`
`!=`	Нерівність	`x != y`
`>`	Більше	`x > y`
`<`	Менше	`x < y`
`>=`	Більше або дорівнює	`x >= y`
`<=`	Менше або дорівнює	`x <= y`

Логічні оператори Python

Логічні оператори використовуються для комбінування умов:

Оператор	Опис	Приклад
`and`	Повертає `True`, якщо обидві умови істинні	`x < 5 and x < 10`
`or`	Повертає `True`, якщо хоча б одна умова істинна	`x < 5 or x < 4`
`not`	Змінює результат на протилежний	`not(x < 5 and x < 10)`

Оператори ідентичності Python

Оператори ідентичності використовуються для порівняння об'єктів, щоб перевірити, чи вони є одним і тим самим об'єктом у пам'яті:

Оператор	Опис	Приклад
`is`	Повертає `True`, якщо обидві змінні є одним і тим самим об'єктом	`x is y`
`is not`	Повертає `True`, якщо обидві змінні не є одним і тим самим об'єктом	`x is not y`

Оператори членства Python

Оператори членства використовуються для перевірки, чи міститься послідовність у об'єкті:

Оператор	Опис	Приклад
`in`	Повертає `True`, якщо послідовність міститься в об'єкті	`x in y`
`not in`	Повертає `True`, якщо послідовність не міститься в об'єкті	`x not in y`

Побітові оператори Python

Побітові оператори використовуються для порівняння (бінарних) чисел:

Оператор	Назва	Опис	Приклад
`&`	AND	Встановлює кожен біт у `1`, якщо обидва біти дорівнюють `1`	`x & y`
`|`	OR	Встановлює кожен біт у `1`, якщо хоча б один із бітів дорівнює `1`	`x | y`
`^`	XOR	Встановлює кожен біт у `1`, якщо тільки один із бітів дорівнює `1`	`x ^ y`
`~`	NOT	Інвертує всі біти	`~x`
`<<`	Зсув вліво	Зсуває біти вліво, додаючи нулі справа	`x << 2`

Оператор	Назва	Опис	Приклад
`>>`	Зсув вправо	Зсуває біти вправо, додаючи копії лівого біта	`x >> 2`

Пріоритет операторів

Пріоритет операторів описує порядок, у якому виконуються операції.

Приклад

Дужки мають найвищий пріоритет, тому вирази в дужках обчислюються першими:

```
print((6 + 3) - (6 + 3))
```

Множення `*` має вищий пріоритет, ніж додавання `+`, тому множення виконується перед додаванням:

```
print(100 + 5 * 3)
```

Порядок пріоритету описаний у таблиці нижче, починаючи з найвищого:

Оператор	Опис
`()`	Дужки
`**`	Піднесення до степеня
`+x -x ~x`	Унарний плюс, унарний мінус, побітове NOT
`* / // %`	Множення, ділення, цілочисельне ділення, остача
`+ -`	Додавання, віднімання
`<< >>`	Побітові зсуви
`&`	Побітове AND
`^`	Побітове XOR
`|`	Побітове OR
`== != > >= < <= is is not in not in`	Порівняння, ідентичність, членство
`not`	Логічне NOT
`and`	Логічне AND
`or`	Логічне OR

Якщо два оператори мають однаковий пріоритет, вираз обчислюється зліва направо.

Приклад

Додавання + і віднімання - мають однаковий пріоритет, тому вираз обчислюється зліва направо:

```
print(5 + 4 - 7 + 3)
```

Списки в Python

```
mylist = ["apple", "banana", "cherry"]
```

Списки

Списки використовуються для зберігання кількох елементів у одній змінній.

Списки є одним із чотирьох вбудованих типів даних у Python, які використовуються для зберігання колекцій даних. Інші три — це кортежі (Tuple), множини (Set) та словники (Dictionary), кожен із яких має свої особливості та призначення.

Списки створюються за допомогою квадратних дужок:

Приклад

Створення списку:

```
thislist = ["apple", "banana", "cherry"]
print(thislist)
```

Елементи списку

Елементи списку впорядковані, змінювані та допускають дублікати.

- **Впорядковані**: Елементи мають визначений порядок, який не змінюється. Нові елементи додаються в кінець списку.
- **Змінювані**: Можна змінювати, додавати та видаляти елементи після створення списку.
- **Дублікати**: Списки можуть містити елементи з однаковими значеннями.

Приклад

Список із дубльованими значеннями:

```
thislist = ["apple", "banana", "cherry", "apple", "cherry"]
print(thislist)
```

Довжина списку

Щоб визначити кількість елементів у списку, використовуйте функцію `len()`:

```
thislist = ["apple", "banana", "cherry"]
print(len(thislist))
```

Типи даних у списках

Елементи списку можуть бути будь-якого типу даних:

```
list1 = ["apple", "banana", "cherry"]  # рядки
list2 = [1, 5, 7, 9, 3]               # числа
list3 = [True, False, False]          # булеві значення
```

Список може містити елементи різних типів:

```
list1 = ["abc", 34, True, 40, "male"]
```

Конструктор `list()`

Списки також можна створювати за допомогою конструктора `list()`:

```
thislist = list(("apple", "banana", "cherry"))  # подвійні круглі дужки
print(thislist)
```

Доступ до елементів

Елементи списку доступні за індексом:

```
thislist = ["apple", "banana", "cherry"]
print(thislist[1])  # другий елемент
```

Негативна індексація

Використовується для доступу до елементів з кінця списку:

```
thislist = ["apple", "banana", "cherry"]
print(thislist[-1])  # останній елемент
```

Діапазон індексів

Можна вказати діапазон індексів для отримання підсписку:

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon",
"mango"]
print(thislist[2:5])  # елементи з 3-го по 5-й
```

Перевірка наявності елемента

Перевірити, чи є елемент у списку, можна за допомогою оператора `in`:

```
thislist = ["apple", "banana", "cherry"]
if "apple" in thislist:
    print("Так, 'apple' є у списку")
```

Зміна елементів

Змінити значення елемента можна за його індексом:

```
thislist = ["apple", "banana", "cherry"]
thislist[1] = "blackcurrant"
print(thislist)
```

Зміна діапазону значень

Можна змінити кілька елементів одночасно:

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "mango"]
thislist[1:3] = ["blackcurrant", "watermelon"]
print(thislist)
```

Вставка елементів

Для вставки нового елемента використовуйте метод `insert()`:

```
thislist = ["apple", "banana", "cherry"]
thislist.insert(2, "watermelon")
print(thislist)
```

Додавання елементів

Щоб додати елемент у кінець списку, використовуйте метод `append()`:

Приклад

Використання методу `append()` для додавання елемента:

```
thislist = ["apple", "banana", "cherry"]
thislist.append("orange")
print(thislist)
```

Вставка елементів Щоб вставити елемент у список на вказану позицію, використовуйте метод `insert()`.

Метод `insert()` вставляє елемент на вказаний індекс:

Приклад

Вставка елемента на другу позицію:

```
thislist = ["apple", "banana", "cherry"]
thislist.insert(1, "orange")
print(thislist)
```

Примітка: У результаті прикладів вище списки тепер міститимуть 4 елементи.

Розширення списку Щоб додати елементи з іншого списку до поточного списку, використовуйте метод `extend()`.

Приклад

Додавання елементів списку `tropical` до `thislist`:

```
thislist = ["apple", "banana", "cherry"]
tropical = ["mango", "pineapple", "papaya"]
thislist.extend(tropical)
print(thislist)
```

Елементи будуть додані в кінець списку.

Додавання будь-якого ітерованого об'єкта Метод `extend()` дозволяє додавати не лише списки, а й будь-які ітеровані об'єкти (кортежі, множини, словники тощо).

Приклад

Додавання елементів кортежу до списку:

```
thislist = ["apple", "banana", "cherry"]
thistuple = ("kiwi", "orange")
thislist.extend(thistuple)
print(thislist)
```

Видалення вказаного елемента

Метод `remove()` видаляє вказаний елемент.

Приклад

Видалення "banana":

```
thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
print(thislist)
```

Якщо в списку є кілька елементів із вказаним значенням, метод `remove()` видаляє перше входження.

Приклад

Видалення першого входження "banana":

```
thislist = ["apple", "banana", "cherry", "banana", "kiwi"]
thislist.remove("banana")
print(thislist)
```

Видалення за індексом

Метод `pop()` видаляє елемент за вказаним індексом.

Приклад

Видалення другого елемента:

```
thislist = ["apple", "banana", "cherry"]
thislist.pop(1)
print(thislist)
```

Якщо індекс не вказано, метод `pop()` видаляє останній елемент.

Приклад

Видалення останнього елемента:

```
thislist = ["apple", "banana", "cherry"]
thislist.pop()
print(thislist)
```

Ключове слово `del` також видаляє елемент за вказаним індексом.

Приклад

Видалення першого елемента:

```
thislist = ["apple", "banana", "cherry"]
del thislist[0]
print(thislist)
```

Ключове слово `del` може також видалити весь список.

Приклад

Видалення всього списку:

```
thislist = ["apple", "banana", "cherry"]
del thislist
```

Очищення списку

Метод `clear()` очищає список, залишаючи його порожнім.

Приклад

Очищення вмісту списку:

```
thislist = ["apple", "banana", "cherry"]
thislist.clear()
print(thislist)
```

Перебір списку

Ви можете перебирати елементи списку за допомогою циклу `for`.

Приклад

Виведення всіх елементів списку по одному:

```
thislist = ["apple", "banana", "cherry"]
for x in thislist:
    print(x)
```

Перебір за індексами

Використовуйте функції `range()` та `len()` для створення ітерації за індексами.

Приклад

Виведення всіх елементів за їх індексами:

```
thislist = ["apple", "banana", "cherry"]
for i in range(len(thislist)):
    print(thislist[i])
```

Використання циклу `while`

Ви можете використовувати цикл `while` для перебору елементів списку за індексами.

Приклад

Виведення всіх елементів за допомогою циклу `while`:

```
thislist = ["apple", "banana", "cherry"]
i = 0
while i < len(thislist):
    print(thislist[i])
    i += 1
```

Перебір за допомогою генераторів списків

Генератори списків пропонують найкоротший синтаксис для перебору списків.

Приклад

Короткий запис циклу для виведення всіх елементів списку:

```
thislist = ["apple", "banana", "cherry"]
[print(x) for x in thislist]
```

Генератори списків

Генератори списків дозволяють створювати нові списки на основі існуючих.

Приклад

Створення нового списку, що містить лише фрукти з літерою "a":

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = [x for x in fruits if "a" in x]
print(newlist)
```

Синтаксис

```
newlist = [expression for item in iterable if condition == True]
```

Сортування списків

Метод `sort()` сортує список за алфавітом або числово за зростанням.

Приклад
Сортування списку за алфавітом:

```
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
thislist.sort()
print(thislist)
```

Сортування за спаданням:

```
thislist.sort(reverse=True)
print(thislist)
```

Зворотний порядок

Метод `reverse()` змінює порядок елементів на протилежний.

Приклад
Зворотний порядок елементів:

```
thislist = ["banana", "Orange", "Kiwi", "cherry"]
thislist.reverse()
print(thislist)
```

Копіювання списку

Ви не можете скопіювати список, просто написавши `list2 = list1`, тому що: `list2` буде лише посиланням на `list1`, і зміни, внесені в `list1`, автоматично відобразяться і в `list2`.

Використання методу `copy()`

Ви можете скористатися вбудованим методом списків `copy()`, щоб створити копію списку.

Приклад
Створення копії списку за допомогою методу `copy()`:

```
thislist = ["apple", "banana", "cherry"]
mylist = thislist.copy()
print(mylist)
```

Використання методу `list()`

Інший спосіб створити копію — використати вбудований метод `list()`.

Приклад

Створення копії списку за допомогою методу `list()`:

```
thislist = ["apple", "banana", "cherry"]
mylist = list(thislist)
print(mylist)
```

Використання оператора зрізу

Ви також можете створити копію списку, використовуючи оператор зрізу `:`.

Приклад

Створення копії списку за допомогою оператора `:`:

```
thislist = ["apple", "banana", "cherry"]
mylist = thislist[:]
print(mylist)
```

Об'єднання двох списків

Існує кілька способів об'єднати або конкатенувати два чи більше списків у Python.

Використання оператора `+`

Один із найпростіших способів — використання оператора `+`.

Приклад

Об'єднання двох списків:

```
list1 = ["a", "b", "c"]
list2 = [1, 2, 3]

list3 = list1 + list2
print(list3)
```

Додавання елементів одного списку до іншого

Інший спосіб об'єднати два списки — додати всі елементи `list2` до `list1` по одному.

Приклад

Додавання `list2` до `list1`:

```
list1 = ["a", "b", "c"]
list2 = [1, 2, 3]

for x in list2:
    list1.append(x)

print(list1)
```

Використання методу `extend()`

Ви також можете скористатися методом `extend()`, щоб додати елементи одного списку до іншого.

Приклад

Використання методу `extend()` для додавання `list2` до кінця `list1`:

```
list1 = ["a", "b", "c"]
list2 = [1, 2, 3]

list1.extend(list2)
print(list1)
```

Методи списків

Python має набір вбудованих методів, які можна використовувати зі списками.

Метод	Опис
`append()`	Додає елемент у кінець списку
`clear()`	Видаляє всі елементи зі списку
`copy()`	Повертає копію списку
`count()`	Повертає кількість елементів із заданим значенням
`extend()`	Додає елементи списку (або будь-якого ітератора) до кінця поточного списку
`index()`	Повертає індекс першого елемента із заданим значенням
`insert()`	Додає елемент у вказану позицію
`pop()`	Видаляє елемент у вказаній позиції
`remove()`	Видаляє елемент із заданим значенням
`reverse()`	Змінює порядок елементів списку на зворотний

Метод	Опис
`sort()`	Сортує список

Python Кортежі (Tuples)

Що таке кортеж?

Кортежі використовуються для зберігання кількох елементів в одній змінній. Це один із чотирьох вбудованих типів даних у Python для зберігання колекцій даних (інші три — це список (List), множина (Set) і словник (Dictionary)), кожен із яких має свої особливості та застосування.

Кортеж — це колекція, яка:

- **Упорядкована** (ordered)
- **Незмінна** (unchangeable)

Кортежі записуються у круглих дужках.

Приклад

Створення кортежу:

```
thistuple = ("apple", "banana", "cherry")
print(thistuple)
```

Елементи кортежу

Властивості:

1. **Упорядкованість**: Елементи мають визначений порядок, який не змінюється.
2. **Незмінність**: Після створення кортежу його елементи не можна змінювати, додавати чи видаляти.
3. **Дублікати дозволені**: Кортежі можуть містити однакові значення.

Приклад

Кортеж із дубльованими значеннями:

```
thistuple = ("apple", "banana", "cherry", "apple", "cherry")
print(thistuple)
```

Довжина кортежу

Щоб визначити кількість елементів у кортежі, використовуйте функцію `len()`.

Приклад

```
thistuple = ("apple", "banana", "cherry")
print(len(thistuple))
```

Кортеж із одним елементом

Щоб створити кортеж із одним елементом, необхідно додати кому після цього елемента.

Приклад

```
thistuple = ("apple",)
print(type(thistuple))  # Це кортеж

thistuple = ("apple")
print(type(thistuple))  # Це рядок
```

Типи даних у кортежах

Елементи кортежу можуть бути будь-якого типу даних, включаючи змішані типи.

Приклад

```
tuple1 = ("apple", "banana", "cherry")  # Рядки
tuple2 = (1, 5, 7, 9, 3)               # Цілі числа
tuple3 = (True, False, False)          # Логічні значення
tuple4 = ("abc", 34, True, 40, "male") # Змішані типи
```

Конструктор `tuple()`

Кортеж також можна створити за допомогою конструктора `tuple()`.

Приклад

```
thistuple = tuple(("apple", "banana", "cherry"))  # Зверніть увагу на
подвійні круглі дужки
print(thistuple)
```

Доступ до елементів кортежу

За індексом

Використовуйте квадратні дужки для доступу до елементів за їх індексом.

Приклад

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[1])  # Виведе "banana"
```

Від'ємні індекси

Від'ємні індекси починаються з кінця.

Приклад

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[-1])  # Виведе "cherry"
```

Діапазон індексів

Можна вказати діапазон індексів для отримання підкортежу.

Приклад

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon",
"mango")
print(thistuple[2:5])  # Виведе ('cherry', 'orange', 'kiwi')
```

Перевірка наявності елемента

Використовуйте ключове слово `in`, щоб перевірити, чи міститься елемент у кортежі.

Приклад

```
thistuple = ("apple", "banana", "cherry")
if "apple" in thistuple:
    print("Так, 'apple' є у кортежі")
```

Зміна кортежів

Кортежі незмінні, але є обхідні шляхи.

Зміна значень

Перетворіть кортеж на список, змініть список і перетворіть його назад у кортеж.

Приклад

```
x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
x = tuple(y)
print(x)
```

Додавання елементів

1. **Перетворення на список**:

   ```
   thistuple = ("apple", "banana", "cherry")
   y = list(thistuple)
   y.append("orange")
   thistuple = tuple(y)
   ```

2. **Додавання кортежу до кортежу**:

   ```
   thistuple = ("apple", "banana", "cherry")
   y = ("orange",)
   thistuple += y
   print(thistuple)
   ```

Видалення елементів

Перетворіть кортеж на список, видаліть елемент і перетворіть його назад у кортеж.

Приклад

```
thistuple = ("apple", "banana", "cherry")
y = list(thistuple)
y.remove("apple")
thistuple = tuple(y)
```

Розпакування кортежів

Приклад

```
fruits = ("apple", "banana", "cherry")
(green, yellow, red) = fruits
print(green)
print(yellow)
print(red)
```

Використання зірочки *

Якщо кількість змінних менша за кількість елементів, використовуйте *, щоб зібрати залишок у список.

Приклад

```
fruits = ("apple", "banana", "cherry", "strawberry", "raspberry")
(green, yellow, *red) = fruits
print(green)
print(yellow)
print(red)
```

Перебір елементів кортежу

Ви можете перебирати елементи кортежу за допомогою циклу `for`.

Приклад

Ітерація через елементи та виведення значень:

```
thistuple = ("apple", "banana", "cherry")
for x in thistuple:
    print(x)
```

Дізнайтеся більше про цикли `for` у розділі Python For Loops.

Перебір за індексами

Ви також можете перебирати елементи кортежу, звертаючись до їхніх індексів.

Використовуйте функції `range()` та `len()` для створення відповідного ітератора.

Приклад

Виведення всіх елементів за їхніми індексами:

```

```

```
thistuple = ("apple", "banana", "cherry")
for i in range(len(thistuple)):
    print(thistuple[i])
```

Використання циклу `while`

Ви можете перебирати елементи кортежу за допомогою циклу `while`.

Використовуйте функцію `len()` для визначення довжини кортежу, починайте з 0 і перебирайте елементи, звертаючись до їхніх індексів. Не забудьте збільшувати індекс на 1 після кожної ітерації.

Приклад

Виведення всіх елементів за допомогою циклу `while`:

```
thistuple = ("apple", "banana", "cherry")
i = 0
while i < len(thistuple):
    print(thistuple[i])
    i = i + 1
```

Дізнайтеся більше про цикли `while` у розділі Python While Loops.

Об'єднання кортежів

Для об'єднання двох або більше кортежів можна використовувати оператор `+`.

Приклад

Об'єднання двох кортежів:

```
tuple1 = ("a", "b", "c")
tuple2 = (1, 2, 3)

tuple3 = tuple1 + tuple2
print(tuple3)
```

Множення кортежів

Якщо потрібно помножити вміст кортежу на певну кількість разів, використовуйте оператор `*`.

Приклад

Множення кортежу `fruits` на 2:

```
fruits = ("apple", "banana", "cherry")
mytuple = fruits * 2

print(mytuple)
```

Методи кортежів

Python має два вбудовані методи, які можна використовувати з кортежами.

Метод	Опис
`count()`	Повертає кількість разів, коли вказане значення зустрічається в кортежі
`index()`	Шукає вказане значення в кортежі та повертає позицію, де його знайдено

Python Множини (Sets)

Множина (Set)

Множини використовуються для зберігання кількох елементів в одній змінній.

Множина є одним із 4 вбудованих типів даних у Python, які використовуються для зберігання колекцій даних. Інші три — це Список (List), Кортеж (Tuple) і Словник (Dictionary), кожен із яких має свої особливості та застосування.

Множина — це колекція, яка є **неупорядкованою**, **незмінною*** і **неіндексованою**.

> **Примітка:** Елементи множини незмінні, але ви можете видаляти елементи та додавати нові.

Множини записуються за допомогою фігурних дужок.

Приклад

Створення множини:

```
thisset = {"apple", "banana", "cherry"}
print(thisset)
```

> **Примітка:** Множини є неупорядкованими, тому ви не можете бути впевнені в порядку елементів.

Елементи множини

Елементи множини є **неупорядкованими**, **незмінними** та **унікальними**.

Неупорядкованість

Елементи множини не мають визначеного порядку. Вони можуть з'являтися в різному порядку кожного разу, коли ви їх використовуєте, і до них не можна звертатися за індексом або ключем.

Незмінність

Елементи множини не можна змінювати після створення. Однак ви можете видаляти елементи та додавати нові.

Унікальність

Множини не можуть містити два елементи з однаковим значенням.

Приклад

Дублікати ігноруються:

```
thisset = {"apple", "banana", "cherry", "apple"}
print(thisset)
```

> **Примітка:** Значення `True` і `1` вважаються однаковими в множинах і розглядаються як дублікати.

```
thisset = {"apple", "banana", "cherry", True, 1, 2}
print(thisset)
```

Довжина множини

Щоб визначити кількість елементів у множині, використовуйте функцію `len()`.

Приклад

Отримання кількості елементів у множині:

```
thisset = {"apple", "banana", "cherry"}
print(len(thisset))
```

Типи даних елементів множини

Елементи множини можуть бути будь-якого типу даних.

Приклад

Різні типи даних у множині:

```
set1 = {"apple", "banana", "cherry"}
set2 = {1, 5, 7, 9, 3}
set3 = {True, False, False}
```

Множина може містити різні типи даних:

```
set1 = {"abc", 34, True, 40, "male"}
```

Конструктор `set()`

Можна також створити множину за допомогою конструктора `set()`.

Приклад

Створення множини за допомогою конструктора:

```
thisset = set(("apple", "banana", "cherry"))  # зверніть увагу на подвійні
круглі дужки
print(thisset)
```

Додавання елементів

Ви можете додавати нові елементи до множини за допомогою методу `add()`.

Приклад

Додавання елемента до множини:

```
thisset = {"apple", "banana", "cherry"}
thisset.add("orange")
print(thisset)
```

Видалення елементів

Для видалення елемента з множини використовуйте методи `remove()` або `discard()`.

Приклад

Видалення елемента за допомогою `remove()`:

```
thisset = {"apple", "banana", "cherry"}
thisset.remove("banana")
print(thisset)
```

> **Примітка:** Якщо елемент для видалення не існує, `remove()` викличе помилку.

Видалення елемента за допомогою `discard()`:

```
thisset = {"apple", "banana", "cherry"}
thisset.discard("banana")
print(thisset)
```

> **Примітка:** Якщо елемент для видалення не існує, `discard()` **не викличе помилку**.

Очищення множини

Метод `clear()` очищає множину:

```
thisset = {"apple", "banana", "cherry"}
thisset.clear()
print(thisset)
```

Ключове слово `del` повністю видаляє множину:

```
thisset = {"apple", "banana", "cherry"}
del thisset
```

Цикл по елементах Ви можете перебирати елементи множини за допомогою циклу `for`:

Приклад

Переберіть множину та виведіть значення:

```
thisset = {"apple", "banana", "cherry"}

for x in thisset:
    print(x)
```

Об'єднання множин

Існує кілька способів об'єднати дві або більше множин у Python.

Методи `union()` та `update()` об'єднують всі елементи з обох множин.

Метод `intersection()` залишає ТІЛЬКИ дублікати.

Метод `difference()` залишає елементи з першої множини, яких немає в інших множинах.

Метод `symmetric_difference()` залишає всі елементи, КРІМ дублікатів.

Об'єднання

Метод `union()` повертає нову множину з усіма елементами обох множин.

Приклад

Об'єднайте `set1` і `set2` у нову множину:

```
set1 = {"a", "b", "c"}
set2 = {1, 2, 3}

set3 = set1.union(set2)
print(set3)
```

Ви можете використовувати оператор `|` замість методу `union()`, і результат буде таким самим.

Приклад

Використовуйте `|` для об'єднання двох множин:

```
set1 = {"a", "b", "c"}
set2 = {1, 2, 3}

set3 = set1 | set2
print(set3)
```

Об'єднання кількох множин

Всі методи та оператори об'єднання можна використовувати для об'єднання кількох множин.

Приклад

Об'єднайте кілька множин за допомогою методу `union()`:

```
set1 = {"a", "b", "c"}
set2 = {1, 2, 3}
set3 = {"John", "Elena"}
set4 = {"apple", "bananas", "cherry"}
```

```
myset = set1.union(set2, set3, set4)
print(myset)
```

Приклад

Використовуйте | для об'єднання кількох множин:

```
set1 = {"a", "b", "c"}
set2 = {1, 2, 3}
set3 = {"John", "Elena"}
set4 = {"apple", "bananas", "cherry"}

myset = set1 | set2 | set3 | set4
print(myset)
```

Об'єднання множини та кортежу

Метод `union()` дозволяє об'єднувати множину з іншими типами даних, такими як списки або кортежі.

Приклад

Об'єднайте множину з кортежем:

```
x = {"a", "b", "c"}
y = (1, 2, 3)

z = x.union(y)
print(z)
```

> **Примітка:** Оператор | дозволяє об'єднувати тільки множини з множинами, а не з іншими типами даних, як це можливо з методом `union()`.

Оновлення

Метод `update()` додає всі елементи з однієї множини до іншої.

Метод `update()` змінює оригінальну множину і не повертає нову.

Приклад

Метод `update()` додає елементи з `set2` до `set1`:

```
set1 = {"a", "b", "c"}
set2 = {1, 2, 3}
```

```
set1.update(set2)
print(set1)
```

> **Примітка:** Методи `union()` та `update()` виключають будь-які дублікати.

Перетин

Залиште ТІЛЬКИ дублікати.

Метод `intersection()` повертає нову множину, яка містить тільки ті елементи, що присутні в обох множинах.

Приклад

Об'єднайте `set1` і `set2`, але залиште тільки дублікати:

```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}

set3 = set1.intersection(set2)
print(set3)
```

Ви можете використовувати оператор `&` замість методу `intersection()`, і результат буде таким самим.

Приклад

Використовуйте `&` для об'єднання двох множин:

```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}

set3 = set1 & set2
print(set3)
```

> **Примітка:** Оператор `&` дозволяє об'єднувати тільки множини з множинами, а не з іншими типами даних, як це можливо з методом `intersection()`.

Метод `intersection_update()` також залишає ТІЛЬКИ дублікати, але змінює оригінальну множину замість повернення нової.

Приклад

Залиште елементи, які існують у `set1` і `set2`:

```
```

```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}

set1.intersection_update(set2)
print(set1)
```

Різниця

Метод `difference()` повертає нову множину, яка містить тільки ті елементи з першої множини, яких немає в інших множинах.

Приклад

Залиште всі елементи з `set1`, яких немає в `set2`:

```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}

set3 = set1.difference(set2)
print(set3)
```

Ви можете використовувати оператор `-` замість методу `difference()`, і результат буде таким самим.

Приклад

Використовуйте `-` для об'єднання двох множин:

```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}

set3 = set1 - set2
print(set3)
```

Симетрична різниця

Метод `symmetric_difference()` залишає тільки ті елементи, які НЕ присутні в обох множинах.

Приклад

Залиште елементи, які не присутні в обох множинах:

```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}
```

```
set3 = set1.symmetric_difference(set2)
print(set3)
```

Ви можете використовувати оператор ^ замість методу symmetric_difference(), і результат буде таким самим.

Приклад

Використовуйте ^ для об'єднання двох множин:

```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}

set3 = set1 ^ set2
print(set3)
```

Методи множин

Python має набір вбудованих методів, які можна використовувати для роботи з множинами.

Метод	Скорочення	Опис
add()		Додає елемент до множини
clear()		Видаляє всі елементи з множини
copy()		Повертає копію множини
difference()	-	Повертає множину, що містить різницю між двома або більше множинами
difference_update()	-=	Видаляє елементи з множини, які також є в іншій множині
discard()		Видаляє вказаний елемент
intersection()	&	Повертає множину, що є перетином двох інших множин
intersection_update()	&=	Видаляє елементи, які не присутні в інших множинах
isdisjoint()		Повертає, чи мають множини спільні елементи
issubset()	<=	Повертає, чи є одна множина підмножиною іншої
issuperset()	>=	Повертає, чи є одна множина надмножиною іншої
pop()		Видаляє елемент з множини
remove()		Видаляє вказаний елемент

Метод	Скорочення	Опис
`symmetric_difference()`	`^`	Повертає множину з симетричною різницею двох множин
`union()`	`|`	Повертає множину, що містить об'єднання множин
`update()`	`|=`	Оновлює множину об'єднанням з іншими множинами

Python Словники

Словник у Python

```
thisdict = {
    "brand": "Ford",
    "model": "Mustang",
    "year": 1964
}
```

Словники використовуються для зберігання даних у форматі пар ключ:значення.

Словник — це колекція, яка є впорядкованою*, змінною та не допускає дублікатів.

> **Примітка:** Починаючи з версії Python 3.7, словники є впорядкованими. У Python 3.6 і раніше словники були невпорядкованими.

Словники записуються за допомогою фігурних дужок і містять ключі та значення.

Приклад

Створіть і виведіть словник:

```
thisdict = {
    "brand": "Ford",
    "model": "Mustang",
    "year": 1964
}
print(thisdict)
```

Елементи словника

Елементи словника впорядковані, змінні та не допускають дублікатів. Вони представлені у вигляді пар ключ:значення, і до них можна звертатися за іменем ключа.

Приклад

Виведіть значення ключа `"brand"`:

```
thisdict = {
    "brand": "Ford",
    "model": "Mustang",
    "year": 1964
}
print(thisdict["brand"])
```

Впорядкованість

- **Впорядковані:** Елементи мають визначений порядок, який не змінюється.
- **Невпорядковані:** Елементи не мають визначеного порядку, і до них не можна звертатися за індексом.

Змінність

Словники є змінними, тобто ви можете змінювати, додавати або видаляти елементи після створення словника.

Дублікати

Словники не можуть мати два елементи з однаковим ключем. Якщо ключ дублюється, нове значення перезапише попереднє.

Приклад

```
thisdict = {
    "brand": "Ford",
    "model": "Mustang",
    "year": 1964,
    "year": 2020
}
print(thisdict)
```

Довжина словника

Використовуйте функцію `len()`, щоб визначити кількість елементів у словнику.

Приклад

```
print(len(thisdict))
```

Типи даних у словнику

Значення у словнику можуть бути будь-якого типу даних.

Приклад

```
thisdict = {
    "brand": "Ford",
    "electric": False,
    "year": 1964,
    "colors": ["red", "white", "blue"]
}
```

Конструктор `dict()`

Словник також можна створити за допомогою конструктора `dict()`.

Приклад

```
thisdict = dict(name="John", age=36, country="Norway")
print(thisdict)
```

Доступ до елементів

До елементів словника можна звертатися за допомогою квадратних дужок або методу `get()`.

Приклад

```
x = thisdict["model"]
x = thisdict.get("model")
```

Методи словника

Ось список основних методів для роботи зі словниками:

Метод	Опис
`clear()`	Видаляє всі елементи зі словника
`copy()`	Повертає копію словника

Метод	Опис
`fromkeys()`	Створює словник із заданими ключами та значенням
`get()`	Повертає значення за заданим ключем
`items()`	Повертає список пар ключ:значення
`keys()`	Повертає список ключів словника
`pop()`	Видаляє елемент із заданим ключем
`popitem()`	Видаляє останню вставлену пару ключ:значення
`setdefault()`	Повертає значення ключа, якщо ключ не існує — додає його зі значенням
`update()`	Оновлює словник заданими парами ключ:значення
`values()`	Повертає список значень словника

Вкладені словники

Словник може містити інші словники.

Приклад

```
myfamily = {
    "child1": {"name": "Emil", "year": 2004},
    "child2": {"name": "Tobias", "year": 2007},
    "child3": {"name": "Linus", "year": 2011}
}
```

Цикли у словниках

Ви можете ітеруватися через ключі, значення або пари ключ:значення.

Приклад

```
for key in thisdict:
    print(key)

for value in thisdict.values():
    print(value)

for key, value in thisdict.items():
    print(key, value)
```