Зміст

Сфера дії Python

Змінна доступна лише в регіоні, де вона створена. Це називається обсягом.

Локальний обсяг

Змінна, створена всередині функції, належить до локальної області видимості цієї функції та може використовуватися лише всередині цієї функції.

Приклад: Локальна змінна

Змінна, створена всередині функції, доступна лише всередині цієї функції:

```
def myfunc():
    x = 300
    print(x)

myfunc()
```

Функція всередині функції

Змінна `x` недоступна поза функцією, але вона доступна для будь-якої функції всередині функції:

Приклад: Доступ до локальної змінної з вкладеної функції

```
def myfunc():
    x = 300
    def myinnerfunc():
```

```
        print(x)
    myinnerfunc()

myfunc()
```

Глобальний обсяг

Змінна, створена в основному тексті коду Python, є глобальною змінною та належить до глобальної області видимості.

Глобальні змінні доступні з будь-якої області видимості, як глобальної, так і локальної.

Приклад: Глобальна змінна

Змінна, створена поза функцією, є глобальною і може використовуватися будь-ким:

```
x = 300

def myfunc():
    print(x)

myfunc()

print(x)
```

Іменування змінних

Якщо ви працюєте з однаковою назвою змінної всередині та поза функцією, Python розглядатиме їх як дві окремі змінні:

Приклад: Локальна та глобальна змінні з однаковою назвою

```
x = 300

def myfunc():
    x = 200
    print(x)

myfunc()

print(x)
```

Ключове слово `global`

Якщо вам потрібно створити глобальну змінну в локальній області, використовуйте ключове слово `global`.

Приклад: Створення глобальної змінної

```
def myfunc():
    global x
    x = 300

myfunc()

print(x)
```

Приклад: Зміна глобальної змінної всередині функції

```
x = 300

def myfunc():
    global x
    x = 200

myfunc()

print(x)
```

Ключове слово `nonlocal`

Ключове слово `nonlocal` використовується для роботи зі змінними всередині вкладених функцій. Воно дозволяє змінювати змінну зовнішньої функції.

Приклад: Використання `nonlocal`

```
def myfunc1():
    x = "Jane"
    def myfunc2():
        nonlocal x
        x = "hello"
    myfunc2()
    return x

print(myfunc1())
```

Модулі Python

Що таке модуль?

Модуль — це бібліотека коду, файл, що містить набір функцій, які ви хочете включити у свою програму.

Створення модуля

Щоб створити модуль, збережіть потрібний код у файлі з розширенням `.py`:

Приклад: Створення модуля

Збережіть цей код у файлі під назвою `mymodule.py`:

```
def greeting(name):
    print("Hello, " + name)
```

Використання модуля

Імпортуйте модуль за допомогою оператора `import`:

Приклад: Виклик функції з модуля

```
import mymodule

mymodule.greeting("Jonathan")
```

Змінні в модулі

Модуль може містити функції та змінні всіх типів (масиви, словники, об’єкти тощо).

Приклад: Використання змінної з модуля

Збережіть цей код у файлі `mymodule.py`:

```
person1 = {
    "name": "John",
    "age": 36,
    "country": "Norway"
}
```

Імпортуйте модуль і отримайте доступ до змінної:

```
import mymodule

a = mymodule.person1["age"]
print(a)
```

Перейменування модуля

Ви можете створити псевдонім для модуля, використовуючи ключове слово `as`:

Приклад: Створення псевдоніма

```
import mymodule as mx

a = mx.person1["age"]
print(a)
```

Вбудовані модулі

Python має кілька вбудованих модулів, які можна імпортувати будь-коли.

Приклад: Використання вбудованого модуля

```
import platform

x = platform.system()
print(x)
```

Функція `dir()`

Функція `dir()` перераховує всі імена функцій або змінних у модулі.

Приклад: Використання `dir()` для модуля

```
import platform

x = dir(platform)
print(x)
```

Імпорт частин модуля

Ви можете імпортувати лише частини модуля за допомогою ключового слова `from`.

Приклад: Імпорт частини модуля

```
from mymodule import person1

print(person1["age"])
```

Примітка: Під час імпорту за допомогою `from` не використовуйте назву модуля для доступу до елементів.

Python Дата та Час

Дати в Python

У Python дата не є окремим типом даних, але ми можемо імпортувати модуль `datetime`, щоб працювати з датами як з об'єктами.

Приклад: Виведення поточної дати

Імпортуйте модуль `datetime` та виведіть поточну дату:

```
import datetime

x = datetime.datetime.now()
print(x)
```

Результат

При виконанні коду вище результат буде таким:

```
2025-04-14 13:20:15.010893
```

Дата містить рік, місяць, день, годину, хвилину, секунду та мікросекунду.

Модуль `datetime` має багато методів для отримання інформації про об'єкт дати.

Приклад: Рік та день тижня

```
import datetime

x = datetime.datetime.now()

print(x.year)
print(x.strftime("%A"))
```

Створення об'єктів дати

Щоб створити дату, можна використовувати клас-конструктор `datetime()` модуля `datetime`.

Приклад: Створення об'єкта дати

```
import datetime

x = datetime.datetime(2020, 5, 17)

print(x)
```

Клас `datetime()` також приймає параметри для часу та часової зони (година, хвилина, секунда, мікросекунда, часовий пояс), але вони є необов'язковими і за замовчуванням мають значення `0` (або `None` для часової зони).

Метод `strftime()`

Об'єкт `datetime` має метод для форматування дати у зручний для читання рядок.

Метод називається `strftime()` і приймає один параметр — формат, який визначає вигляд повернутого рядка.

Приклад: Виведення назви місяця

```
import datetime

x = datetime.datetime(2018, 6, 1)

print(x.strftime("%B"))
```

Довідник форматів

Директива	Опис	Приклад
`%a`	День тижня, коротка версія	Wed
`%A`	День тижня, повна версія	Wednesday
`%w`	День тижня як число (0-6)	3
`%d`	День місяця (01-31)	31
`%b`	Назва місяця, коротка версія	Dec
`%B`	Назва місяця, повна версія	December
`%m`	Місяць як число (01-12)	12

Директива	Опис	Приклад
`%y`	Рік, коротка версія (без століття)	18
`%Y`	Рік, повна версія	2018
`%H`	Година (00-23)	17
`%I`	Година (00-12)	05
`%p`	AM/PM	PM
`%M`	Хвилина (00-59)	41
`%S`	Секунда (00-59)	08
`%f`	Мікросекунда (000000-999999)	548513
`%z`	Зміщення UTC	+0100
`%Z`	Часовий пояс	CST
`%j`	День року (001-366)	365
`%U`	Номер тижня року (неділя — перший день)	52
`%W`	Номер тижня року (понеділок — перший день)	52
`%c`	Локальна версія дати та часу	Mon Dec 31 17:41:00 2018
`%C`	Століття	20
`%x`	Локальна версія дати	12/31/18
`%X`	Локальна версія часу	17:41:00
`%%`	Символ `%`	%
`%G`	Рік за ISO 8601	2018
`%u`	День тижня за ISO 8601 (1-7)	1
`%V`	Номер тижня за ISO 8601 (01-53)	01

Python Математика

Вбудовані математичні функції

Функції `min()` та `max()` використовуються для знаходження найменшого або найбільшого значення в ітерації.

Приклад: Мінімум та максимум

```
x = min(5, 10, 25)
y = max(5, 10, 25)
```

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

Функція `abs()` повертає абсолютне (позитивне) значення числа.

Приклад: Абсолютне значення

```
x = abs(-7.25)

print(x)
```

Функція `pow(x, y)` повертає значення `x` у степені `y` (тобто `x^y`).

Приклад: Степінь числа

```
x = pow(4, 3)

print(x)
```

Модуль Math

Python має вбудований модуль `math`, який розширює список математичних функцій.

Щоб використовувати його, потрібно імпортувати модуль:

```
import math
```

Приклад: Квадратний корінь

Метод `math.sqrt()` повертає квадратний корінь числа:

```
import math

x = math.sqrt(64)

print(x)
```

Приклад: Округлення

Метод `math.ceil()` округлює число вгору до найближчого цілого, а метод `math.floor()` округлює вниз:

```
import math

x = math.ceil(1.4)
y = math.floor(1.4)

print(x) # повертає 2
print(y) # повертає 1
```

Константа `math.pi`

Константа `math.pi` повертає значення числа Пі (3.14...):

```
import math

x = math.pi
print(x)
```

Python Регулярні Вирази (RegEx)

Регулярний вираз (RegEx) — це послідовність символів, яка утворює шаблон для пошуку.

RegEx можна використовувати для перевірки, чи містить рядок заданий шаблон пошуку.

Модуль RegEx

Python має вбудований пакет `re`, який можна використовувати для роботи з регулярними виразами.

Імпортуйте модуль `re`:

```
import re
```

RegEx у Python

Після імпорту модуля `re` ви можете почати використовувати регулярні вирази:

Приклад
Перевірте, чи рядок починається з "The" і закінчується "Spain":

```
import re

txt = "The rain in Spain"
x = re.search("^The.*Spain$", txt)
```

Функції RegEx

Модуль `re` пропонує набір функцій для пошуку збігів у рядку:

Функція	Опис
`findall`	Повертає список, що містить усі збіги
`search`	Повертає об'єкт Match, якщо є збіг у будь-якому місці рядка
`split`	Повертає список, де рядок розділено в кожному місці збігу
`sub`	Замінює один або кілька збігів на заданий рядок

Мета-символи

Мета-символи мають спеціальне значення:

Символ	Опис	Приклад
`[]`	Набір символів	`[a-m]`
`\`	Сигналізує спеціальну послідовність	`\d`
`.`	Будь-який символ (крім символу нового рядка)	`he..o`
`^`	Починається з	`^hello`
`$`	Закінчується на	`planet$`
`*`	Нуль або більше повторень	`he.*o`
`+`	Одне або більше повторень	`he.+o`
`?`	Нуль або одне повторення	`he.?o`
`{}`	Точно задана кількість повторень	`he.{2}o`
`|`	Або	`falls|stays`
`()`	Групування	

Прапори

Ви можете додати прапори до шаблону під час використання регулярних виразів.

Прапор	Скорочення	Опис
`re.ASCII`	`re.A`	Повертає лише ASCII-збіги
`re.DEBUG`		Повертає інформацію для налагодження
`re.DOTALL`	`re.S`	Дозволяє символу `.` відповідати всім символам (включаючи новий рядок)
`re.IGNORECASE`	`re.I`	Нечутливий до регістру пошук

Прапор	Скорочення	Опис
`re.MULTILINE`	`re.M`	Повертає збіги на початку кожного рядка
`re.UNICODE`	`re.U`	Повертає Unicode-збіги (за замовчуванням у Python 3)
`re.VERBOSE`	`re.X`	Дозволяє пробіли та коментарі в шаблонах для кращої читабельності

Спеціальні послідовності

Спеціальна послідовність — це `\`, за яким слідує один із символів у списку нижче:

Символ	Опис	Приклад
`\A`	Збіг, якщо символи на початку рядка	`\AThe`
`\b`	Збіг на початку або в кінці слова	`r"\bain"`
`\B`	Збіг, якщо символи НЕ на початку або в кінці слова	`r"\Bain"`
`\d`	Збіг, якщо рядок містить цифри (0-9)	`\d`
`\D`	Збіг, якщо рядок НЕ містить цифр	`\D`
`\s`	Збіг, якщо рядок містить пробіл	`\s`
`\S`	Збіг, якщо рядок НЕ містить пробілів	`\S`
`\w`	Збіг, якщо рядок містить будь-які символи слова (a-Z, 0-9, _)	`\w`
`\W`	Збіг, якщо рядок НЕ містить символів слова	`\W`
`\Z`	Збіг, якщо символи в кінці рядка	`Spain\Z`

Набори

Набір — це набір символів у квадратних дужках `[]` зі спеціальним значенням:

Набір	Опис
`[arn]`	Збіг, якщо один із символів (a, r або n) присутній
`[a-n]`	Збіг для будь-якої малої літери між a та n
`[^arn]`	Збіг для будь-якого символу, крім a, r та n
`[0123]`	Збіг, якщо один із зазначених чисел (0, 1, 2 або 3) присутній
`[0-9]`	Збіг для будь-якої цифри між 0 та 9
`[0-5][0-9]`	Збіг для будь-якого двозначного числа від 00 до 59
`[a-zA-Z]`	Збіг для будь-якої літери (малої або великої) між a та z
`[+]`	У наборах символи `+`, `*`, `.`, `|`, `()`, `$`, `{}` не мають спеціального значення

Функція `findall()`

Функція `findall()` повертає список, що містить усі збіги.

Приклад
Виведіть список усіх збігів:

```
import re

txt = "The rain in Spain"
x = re.findall("ai", txt)
print(x)
```

Якщо збігів не знайдено, повертається порожній список:

```
import re

txt = "The rain in Spain"
x = re.findall("Portugal", txt)
print(x)
```

Функція `search()`

Функція `search()` шукає збіг у рядку та повертає об'єкт Match, якщо збіг знайдено.

Приклад
Знайдіть перший пробіл у рядку:

```
import re

txt = "The rain in Spain"
x = re.search("\s", txt)

print("Перший пробіл знаходиться на позиції:", x.start())
```

Якщо збігів не знайдено, повертається значення `None`:

```
import re

txt = "The rain in Spain"
x = re.search("Portugal", txt)
print(x)
```

Функція `split()`

Функція `split()` повертає список, де рядок розділено в кожному місці збігу.

Приклад
Розділіть рядок у кожному пробілі:

```
import re

txt = "The rain in Spain"
x = re.split("\s", txt)
print(x)
```

Ви можете обмежити кількість розділень, вказавши параметр `maxsplit`:

```
import re

txt = "The rain in Spain"
x = re.split("\s", txt, 1)
print(x)
```

Функція `sub()`

Функція `sub()` замінює збіги на заданий текст.

Приклад
Замініть кожен пробіл на число 9:

```
import re

txt = "The rain in Spain"
x = re.sub("\s", "9", txt)
print(x)
```

Ви можете обмежити кількість замін, вказавши параметр `count`:

```
import re

txt = "The rain in Spain"
x = re.sub("\s", "9", txt, 2)
print(x)
```

Об'єкт Match

Об'єкт Match містить інформацію про пошук і результат.

Приклад

Виконайте пошук, який поверне об'єкт Match:

```
import re

txt = "The rain in Spain"
x = re.search("ai", txt)
print(x)  # це виведе об'єкт
```

Об'єкт Match має властивості та методи для отримання інформації про пошук:

- `.span()` повертає кортеж із початкової та кінцевої позицій збігу.
- `.string` повертає рядок, переданий у функцію.
- `.group()` повертає частину рядка, де був збіг.

Приклад

Виведіть позицію (початок і кінець) першого збігу:

```
import re

txt = "The rain in Spain"
x = re.search(r"\bS\w+", txt)
print(x.span())
```

Виведіть рядок, переданий у функцію:

```
import re

txt = "The rain in Spain"
x = re.search(r"\bS\w+", txt)
print(x.string)
```

Виведіть частину рядка, де був збіг:

```
import re

txt = "The rain in Spain"
x = re.search(r"\bS\w+", txt)
print(x.group())
```

Примітка: Якщо збігів немає, повертається значення `None` замість об'єкта Match.

Python JSON

JSON — це синтаксис для зберігання та обміну даними.

JSON — це текст, написаний у форматі JavaScript Object Notation.

JSON у Python

Python має вбудований модуль `json`, який дозволяє працювати з даними у форматі JSON.

Приклад: Імпорт модуля `json`

```
import json
```

Парсинг JSON — Перетворення з JSON у Python

Якщо у вас є JSON-рядок, ви можете розпарсити його за допомогою методу `json.loads()`. Результатом буде Python-словник.

Приклад: Перетворення з JSON у Python

```
import json

# JSON-рядок:
x = '{ "name":"John", "age":30, "city":"New York"}'

# Парсинг JSON:
y = json.loads(x)

# Результат — Python-словник:
print(y["age"])
```

Перетворення з Python у JSON

Якщо у вас є Python-об'єкт, ви можете перетворити його у JSON-рядок за допомогою методу `json.dumps()`.

Приклад: Перетворення з Python у JSON

```
import json

# Python-об'єкт (словник):
x = {
    "name": "John",
    "age": 30,
```

```
    "city": "New York"
}

# Перетворення у JSON:
y = json.dumps(x)

# Результат — JSON-рядок:
print(y)
```

Типи Python-об'єктів, які можна конвертувати у JSON

- `dict`
- `list`
- `tuple`
- `string`
- `int`
- `float`
- `True`
- `False`
- `None`

Приклад: Конвертація Python-об'єктів у JSON

```
import json

print(json.dumps({"name": "John", "age": 30}))
print(json.dumps(["apple", "bananas"]))
print(json.dumps(("apple", "bananas")))
print(json.dumps("hello"))
print(json.dumps(42))
print(json.dumps(31.76))
print(json.dumps(True))
print(json.dumps(False))
print(json.dumps(None))
```

Відповідність між Python та JSON

Python	JSON
`dict`	Object
`list`	Array
`tuple`	Array
`str`	String
`int`	Number

Python	JSON
`float`	Number
`True`	`true`
`False`	`false`
`None`	`null`

Приклад: Python-об'єкт з усіма допустимими типами даних

```
import json

x = {
    "name": "John",
    "age": 30,
    "married": True,
    "divorced": False,
    "children": ("Ann", "Billy"),
    "pets": None,
    "cars": [
        {"model": "BMW 230", "mpg": 27.5},
        {"model": "Ford Edge", "mpg": 24.1}
    ]
}

print(json.dumps(x))
```

Форматування результату

Метод `json.dumps()` має параметри для полегшення читання результату.

Приклад: Використання параметра `indent`

```
json.dumps(x, indent=4)
```

Приклад: Використання параметра `separators`

```
json.dumps(x, indent=4, separators=(". ", " = "))
```

Приклад: Сортування ключів за допомогою параметра `sort_keys`

```
json.dumps(x, indent=4, sort_keys=True)
```

Python PIP

Що таке PIP?

PIP — це менеджер пакетів для Python.

> **Примітка:** Якщо у вас Python версії 3.4 або новішої, PIP встановлено за замовчуванням.

Що таке пакет?

Пакет містить усі файли, необхідні для модуля. Модулі — це бібліотеки Python-коду, які можна включити у ваш проєкт.

Перевірка встановлення PIP

Перейдіть у командний рядок до директорії скриптів Python і введіть:

```
pip --version
```

Встановлення PIP

Якщо PIP не встановлено, завантажте та встановіть його з офіційної сторінки PIP.

Завантаження пакета

Для завантаження пакета використовуйте команду `pip install`.

Приклад: Завантаження пакета `camelcase`

```
pip install camelcase
```

Використання пакета

Після встановлення пакета його можна імпортувати у ваш проєкт.

Приклад: Використання пакета `camelcase`

```
import camelcase

c = camelcase.CamelCase()

txt = "hello world"

print(c.hump(txt))
```

Видалення пакета

Для видалення пакета використовуйте команду `pip uninstall`.

Приклад: Видалення пакета `camelcase`

```
pip uninstall camelcase
```

Список встановлених пакетів

Для перегляду списку встановлених пакетів використовуйте команду `pip list`.

Приклад: Список встановлених пакетів

```
pip list
```

Результат:

```
Package         Version
-----------------------
camelcase       0.2
mysql-connector 2.1.6
pip             18.1
pymongo         3.6.1
setuptools      39.0.1
```

Python Спроба та Обробка Винятків

Блок `try` дозволяє перевірити блок коду на наявність помилок.

Блок `except` дозволяє обробляти помилки.

Блок `else` дозволяє виконувати код, якщо помилок не виникло.

Блок `finally` дозволяє виконувати код незалежно від результату блоків `try` та `except`.

Обробка Винятків

Коли виникає помилка (виняток), Python зазвичай зупиняє виконання програми та генерує повідомлення про помилку.

Ці винятки можна обробляти за допомогою оператора `try`:

Приклад

Блок `try` згенерує виняток, оскільки змінна `x` не визначена:

```
try:
    print(x)
except:
    print("Виникла помилка")
```

Оскільки блок `try` викликає помилку, буде виконано блок `except`.

Без блоку `try` програма завершиться з помилкою:

Приклад

Цей код викличе помилку, оскільки змінна `x` не визначена:

```
print(x)
```

Кілька Винятків

Можна визначити стільки блоків `except`, скільки потрібно, наприклад, якщо ви хочете виконати спеціальний код для певного типу помилки:

Приклад

Вивести повідомлення, якщо блок `try` викликає `NameError`, і інше повідомлення для інших помилок:

```
try:
    print(x)
except NameError:
    print("Змінна x не визначена")
except:
    print("Щось пішло не так")
```

Дивіться більше типів помилок у довіднику вбудованих винятків Python.

`else`

Можна використовувати ключове слово `else` для визначення блоку коду, який буде виконано, якщо помилок не виникло:

Приклад

У цьому прикладі блок `try` не викликає помилок:

```
try:
    print("Привіт")
except:
    print("Щось пішло не так")
else:
    print("Все добре")
```

finally

Блок `finally`, якщо він вказаний, буде виконаний незалежно від того, чи виникла помилка в блоці `try`.

Приклад

```
try:
    print(x)
except:
    print("Щось пішло не так")
finally:
    print("Блок 'try except' завершено")
```

Це може бути корисним для закриття об'єктів та очищення ресурсів:

Приклад

Спроба відкрити та записати у файл, який недоступний для запису:

```
try:
    f = open("demofile.txt")
    try:
        f.write("Lorum Ipsum")
    except:
        print("Щось пішло не так під час запису у файл")
    finally:
        f.close()
except:
    print("Щось пішло не так під час відкриття файлу")
```

Програма може продовжити виконання, не залишаючи файл відкритим.

Виклик Винятку

Як розробник Python, ви можете викликати виняток, якщо виникає певна умова.

Для виклику (або підняття) винятку використовуйте ключове слово `raise`.

Приклад

Викликати помилку та зупинити програму, якщо `x` менше 0:

```
x = -1

if x < 0:
    raise Exception("Вибачте, числа нижче нуля заборонені")
```

Ключове слово `raise` використовується для виклику винятку.

Ви можете визначити, який тип помилки викликати, і текст, який буде виведено користувачеві.

Приклад

Викликати `TypeError`, якщо `x` не є цілим числом:

```
x = "hello"

if not type(x) is int:
    raise TypeError("Дозволені лише цілі числа")
```

Python Введення Даних Користувача

Python дозволяє отримувати введення від користувача.

Це означає, що ми можемо запитати у користувача дані.

Метод введення відрізняється у Python 3.6 та Python 2.7.

- Python 3.6 використовує метод `input()`.
- Python 2.7 використовує метод `raw_input()`.

Приклад

Наступний приклад запитує ім'я користувача, і після введення воно виводиться на екран:

Python 3.6

```
username = input("Введіть ім'я користувача:")
print("Ім'я користувача: " + username)
```

Python 2.7

```
username = raw_input("Введіть ім'я користувача:")
print("Ім'я користувача: " + username)
```

Python зупиняє виконання, коли доходить до функції `input()`, і продовжує після того, як користувач введе дані.

Форматування рядків у Python

F-рядки (F-String) були введені в Python 3.6 і зараз є рекомендованим способом форматування рядків.

До Python 3.6 для цього використовувався метод `format()`.

F-рядки

F-рядки дозволяють форматувати вибрані частини рядка. Щоб вказати, що рядок є F-рядком, просто додайте літеру `f` перед рядковим літералом, наприклад:

Приклад

Створення F-рядка:

```
txt = f"The price is 49 dollars"
print(txt)
```

Заповнювачі та модифікатори

Для форматування значень у F-рядках використовуйте заповнювачі `{}`. Заповнювач може містити змінні, операції, функції та модифікатори для форматування значення.

Приклад

Додайте заповнювач для змінної `price`:

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

Заповнювач також може включати модифікатор для форматування значення. Наприклад, `.2f` означає число з фіксованою точкою та двома десятковими знаками:

Приклад

Відображення ціни з двома десятковими знаками:

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

Форматувати значення можна безпосередньо:

Приклад

Відображення значення `95` з двома десятковими знаками:

```
txt = f"The price is {95:.2f} dollars"
print(txt)
```

Виконання операцій у F-рядках

У заповнювачах можна виконувати операції Python.

Приклад

Математична операція в заповнювачі:

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

Математичні операції зі змінними:

Приклад

Додавання податків перед відображенням ціни:

```
price = 59
tax = 0.25
txt = f"The price is {price + (price * tax)} dollars"
print(txt)
```

Використання умовних операторів `if...else`:

Приклад

Повернення "Expensive", якщо ціна перевищує 50, інакше "Cheap":

```
price = 49
txt = f"It is very {'Expensive' if price > 50 else 'Cheap'}"
print(txt)
```

Виконання функцій у F-рядках

У заповнювачах можна виконувати функції.

Приклад

Використання методу `upper()` для перетворення значення в верхній регістр:

```
fruit = "apples"
txt = f"I love {fruit.upper()}"
print(txt)
```

Можна використовувати власні функції:

Приклад

Функція для перетворення футів у метри:

```
def myconverter(x):
    return x * 0.3048

txt = f"The plane is flying at a {myconverter(30000)} meter altitude"
print(txt)
```

Інші модифікатори

На початку розділу ми розглянули модифікатор `.2f`. Ось інші модифікатори, які можна використовувати:

Приклад

Використання коми як роздільника тисяч:

```
price = 59000
txt = f"The price is {price:,} dollars"
print(txt)
```

Типи форматування

Тип	Опис
`:<`	Вирівнювання вліво
`:>`	Вирівнювання вправо
`:^`	Центрування
`:=`	Знак у крайньому лівому положенні

Тип	Опис
`:+`	Додавання знака для позитивних і негативних чисел
`:-`	Знак лише для негативних чисел
`: `	Додатковий пробіл перед позитивними числами
`:,`	Роздільник тисяч комою
`:_`	Роздільник тисяч підкресленням
`:b`	Двійковий формат
`:c`	Символ Unicode
`:d`	Десятковий формат
`:e`	Науковий формат (нижній регістр `e`)
`:E`	Науковий формат (верхній регістр `E`)
`:f`	Число з фіксованою точкою
`:F`	Число з фіксованою точкою (верхній регістр для `inf` і `nan`)
`:g`	Загальний формат
`:G`	Загальний формат (верхній регістр для наукових позначень)
`:o`	Вісімковий формат
`:x`	Шістнадцятковий формат (нижній регістр)
`:X`	Шістнадцятковий формат (верхній регістр)
`:n`	Числовий формат
`:%`	Формат відсотків

Метод `format()`

До Python 3.6 для форматування рядків використовувався метод `format()`. Він все ще доступний, але F-рядки швидші та рекомендовані.

Метод `format()` також використовує фігурні дужки `{}` як заповнювачі, але синтаксис трохи відрізняється.

Приклад

Додавання заповнювача для відображення ціни:

```
price = 49
txt = "The price is {} dollars"
print(txt.format(price))
```

Форматування значення з двома десятковими знаками:

Приклад

```
txt = "The price is {:.2f} dollars"
```

Кілька значень

Для використання кількох значень додайте їх у метод `format()`:

Приклад

```
quantity = 3
itemno = 567
price = 49
myorder = "I want {} pieces of item number {} for {:.2f} dollars."
print(myorder.format(quantity, itemno, price))
```

Індексні номери

Використовуйте індексні номери `{0}` для точного розташування значень:

Приклад

```
quantity = 3
itemno = 567
price = 49
myorder = "I want {0} pieces of item number {1} for {2:.2f} dollars."
print(myorder.format(quantity, itemno, price))
```

Використання одного значення кілька разів:

Приклад

```
age = 36
name = "John"
txt = "His name is {1}. {1} is {0} years old."
print(txt.format(age, name))
```

Іменовані індекси

Використовуйте іменовані індекси `{carname}`:

Приклад

```
myorder = "I have a {carname}, it is a {model}."
print(myorder.format(carname="Ford", model="Mustang"))
```

Відкриття файлів у Python

Робота з файлами є важливою частиною будь-якого веб-додатка.

Python має кілька функцій для створення, читання, оновлення та видалення файлів.

Робота з файлами

Ключова функція для роботи з файлами в Python — це функція `open()`.

Функція `open()` приймає два параметри: ім'я файлу та режим.

Існує чотири різні методи (режими) відкриття файлу:

- `"r"` - Читання - Значення за замовчуванням. Відкриває файл для читання, видає помилку, якщо файл не існує.
- `"a"` - Додавання - Відкриває файл для додавання, створює файл, якщо він не існує.
- `"w"` - Запис - Відкриває файл для запису, створює файл, якщо він не існує.
- `"x"` - Створення - Створює вказаний файл, видає помилку, якщо файл вже існує.

Крім того, ви можете вказати, чи слід обробляти файл у текстовому або бінарному режимі:

- `"t"` - Текстовий - Значення за замовчуванням. Текстовий режим.
- `"b"` - Бінарний - Бінарний режим (наприклад, для зображень).

Синтаксис

Щоб відкрити файл для читання, достатньо вказати ім'я файлу:

```
f = open("demofile.txt")
```

Код вище еквівалентний:

```
f = open("demofile.txt", "rt")
```

Оскільки `"r"` для читання та `"t"` для тексту є значеннями за замовчуванням, їх можна не вказувати.

> **Примітка:** Переконайтеся, що файл існує, інакше ви отримаєте помилку.

Відкриття файлу на сервері

Припустимо, у нас є наступний файл, розташований у тій самій папці, що й Python:

demofile.txt

```
Hello! Welcome to demofile.txt
This file is for testing purposes.
Good Luck!
```

Щоб відкрити файл, використовуйте вбудовану функцію `open()`.

Функція `open()` повертає об'єкт файлу, який має метод `read()` для читання вмісту файлу:

Приклад

```
f = open("demofile.txt", "r")
print(f.read())
```

Якщо файл знаходиться в іншому місці, потрібно вказати шлях до файлу:

Приклад

```
f = open("D:\\myfiles\\welcome.txt", "r")
print(f.read())
```

Читання частини файлу

За замовчуванням метод `read()` повертає весь текст, але ви також можете вказати, скільки символів потрібно повернути:

Приклад

```
f = open("demofile.txt", "r")
print(f.read(5))
```

Читання рядків

Ви можете повернути один рядок, використовуючи метод `readline()`:

Приклад

```
f = open("demofile.txt", "r")
print(f.readline())
```

Щоб прочитати два перші рядки, викличте `readline()` двічі:

Приклад

```
f = open("demofile.txt", "r")
print(f.readline())
print(f.readline())
```

Щоб прочитати весь файл рядок за рядком, використовуйте цикл:

Приклад

```
f = open("demofile.txt", "r")
for x in f:
    print(x)
```

Закриття файлів

Завжди закривайте файл після завершення роботи з ним.

Приклад

```
f = open("demofile.txt", "r")
print(f.readline())
f.close()
```

> **Примітка:** Закриття файлів є важливою практикою. У деяких випадках зміни у файлі можуть не відображатися, доки файл не буде закрито.

Запис у існуючий файл

Щоб записати в існуючий файл, додайте параметр до функції `open()`:

- `"a"` - Додавання - додає вміст у кінець файлу.
- `"w"` - Запис - перезаписує весь вміст файлу.

Приклад

```
f = open("demofile2.txt", "a")
f.write("Now the file has more content!")
f.close()

f = open("demofile2.txt", "r")
print(f.read())
```

Приклад

```
f = open("demofile3.txt", "w")
f.write("Woops! I have deleted the content!")
f.close()

f = open("demofile3.txt", "r")
print(f.read())
```

> **Примітка:** Метод `"w"` перезаписує весь файл.

Створення нового файлу

Щоб створити новий файл у Python, використовуйте метод `open()` з одним із наступних параметрів:

- `"x"` - Створення - створює файл, видає помилку, якщо файл вже існує.
- `"a"` - Додавання - створює файл, якщо вказаний файл не існує.
- `"w"` - Запис - створює файл, якщо вказаний файл не існує.

Приклад

```
f = open("myfile.txt", "x")
```

Приклад

```
f = open("myfile.txt", "w")
```

Видалення файлу

Щоб видалити файл, імпортуйте модуль `os` і викличте функцію `os.remove()`:

Приклад

```
import os
os.remove("demofile.txt")
```

Перевірка існування файлу

Щоб уникнути помилки, перевірте, чи існує файл, перед його видаленням:

Приклад

```

```

```
import os
if os.path.exists("demofile.txt"):
    os.remove("demofile.txt")
else:
    print("The file does not exist")
```

Видалення папки

Щоб видалити цілу папку, використовуйте метод `os.rmdir()`:

Приклад

```
import os
os.rmdir("myfolder")
```

> **Примітка:** Ви можете видалити лише порожні папки.

Модулі

Модуль Random

Python має вбудований модуль, який можна використовувати для створення випадкових чисел.

Модуль `random` має набір методів:

Метод	Опис
`seed()`	Ініціалізує генератор випадкових чисел
`getstate()`	Повертає поточний внутрішній стан генератора випадкових чисел
`setstate()`	Відновлює внутрішній стан генератора випадкових чисел
`getrandbits()`	Повертає число, що представляє випадкові біти
`randrange()`	Повертає випадкове число в заданому діапазоні
`randint()`	Повертає випадкове ціле число в заданому діапазоні
`choice()`	Повертає випадковий елемент із заданої послідовності
`choices()`	Повертає список із випадковим вибором із заданої послідовності
`shuffle()`	Перемішує послідовність у випадковому порядку
`sample()`	Повертає вибірку із заданої послідовності
`random()`	Повертає випадкове число з плаваючою точкою між 0 і 1
`uniform()`	Повертає випадкове число з плаваючою точкою між двома параметрами

Метод	Опис
`triangular()`	Повертає випадкове число з плаваючою точкою між двома параметрами, можна також задати середню точку
`betavariate()`	Повертає випадкове число з плаваючою точкою на основі бета-розподілу (використовується в статистиці)
`expovariate()`	Повертає випадкове число на основі експоненціального розподілу
`gammavariate()`	Повертає випадкове число на основі гамма-розподілу
`gauss()`	Повертає випадкове число на основі нормального розподілу
`lognormvariate()`	Повертає випадкове число на основі логнормального розподілу
`normalvariate()`	Повертає випадкове число на основі нормального розподілу
`vonmisesvariate()`	Повертає випадкове число на основі розподілу фон Мізеса
`paretovariate()`	Повертає випадкове число на основі розподілу Парето
`weibullvariate()`	Повертає випадкове число на основі розподілу Вейбулла

Модуль Math в Python

Python має вбудований модуль для виконання математичних завдань.

Модуль `math` має набір методів і констант.

Математичні методи

Метод	Опис
`math.acos()`	Повертає арккосинус числа
`math.acosh()`	Повертає обернений гіперболічний косинус числа
`math.asin()`	Повертає арксинус числа
`math.asinh()`	Повертає обернений гіперболічний синус числа
`math.atan()`	Повертає арктангенс числа в радіанах
`math.atan2()`	Повертає арктангенс y/x в радіанах
`math.atanh()`	Повертає обернений гіперболічний тангенс числа
`math.ceil()`	Округлює число вгору до найближчого цілого
`math.comb()`	Повертає кількість способів вибору k елементів із n без повторень
`math.copysign()`	Повертає число з величиною першого параметра і знаком другого
`math.cos()`	Повертає косинус числа
`math.cosh()`	Повертає гіперболічний косинус числа

Метод	Опис
`math.degrees()`	Перетворює кут із радіан у градуси
`math.dist()`	Повертає евклідову відстань між двома точками
`math.erf()`	Повертає функцію помилки числа
`math.erfc()`	Повертає додаткову функцію помилки числа
`math.exp()`	Повертає E у степені x
`math.expm1()`	Повертає Ex - 1
`math.fabs()`	Повертає абсолютне значення числа
`math.factorial()`	Повертає факторіал числа
`math.floor()`	Округлює число вниз до найближчого цілого
`math.fmod()`	Повертає залишок від ділення x/y
`math.frexp()`	Повертає мантису та експоненту числа
`math.fsum()`	Повертає суму всіх елементів у ітерабельному об'єкті
`math.gamma()`	Повертає гамма-функцію числа
`math.gcd()`	Повертає найбільший спільний дільник двох чисел
`math.hypot()`	Повертає евклідову норму
`math.isclose()`	Перевіряє, чи два значення близькі
`math.isfinite()`	Перевіряє, чи є число скінченним
`math.isinf()`	Перевіряє, чи є число нескінченним
`math.isnan()`	Перевіряє, чи є значення NaN
`math.isqrt()`	Округлює квадратний корінь вниз до найближчого цілого
`math.ldexp()`	Повертає обернене значення math.frexp()
`math.lgamma()`	Повертає логарифм гамма-функції числа
`math.log()`	Повертає натуральний логарифм числа
`math.log10()`	Повертає логарифм числа за основою 10
`math.log1p()`	Повертає натуральний логарифм 1+x
`math.log2()`	Повертає логарифм числа за основою 2
`math.perm()`	Повертає кількість перестановок k елементів із n
`math.pow()`	Повертає значення x у степені y
`math.prod()`	Повертає добуток усіх елементів у ітерабельному об'єкті
`math.radians()`	Перетворює значення градусів у радіани

Метод	Опис
`math.remainder()`	Повертає найближче значення, яке робить чисельник повністю подільним на знаменник
`math.sin()`	Повертає синус числа
`math.sinh()`	Повертає гіперболічний синус числа
`math.sqrt()`	Повертає квадратний корінь числа
`math.tan()`	Повертає тангенс числа
`math.tanh()`	Повертає гіперболічний тангенс числа
`math.trunc()`	Повертає цілу частину числа

Математичні константи

Константа	Опис
`math.e`	Повертає число Ейлера (2.7182...)
`math.inf`	Повертає позитивну нескінченність
`math.nan`	Повертає значення NaN (не число)
`math.pi`	Повертає число Пі (3.1415...)
`math.tau`	Повертає число тау (6.2831...)

Модуль cMath в Python

Python має вбудований модуль для виконання математичних завдань із комплексними числами.

Методи цього модуля приймають `int`, `float` і `complex` числа. Вони також приймають об'єкти Python, які мають методи `__complex__()` або `__float__()`.

Методи цього модуля майже завжди повертають комплексне число. Якщо значення можна виразити як дійсне число, то у поверненого значення уявна частина дорівнює 0.

Модуль `cmath` має набір методів і констант.

Методи cMath

Метод	Опис
`cmath.acos(x)`	Повертає арккосинус x
`cmath.acosh(x)`	Повертає гіперболічний арккосинус x
`cmath.asin(x)`	Повертає арксинус x
`cmath.asinh(x)`	Повертає гіперболічний арксинус x

Метод	Опис
`cmath.atan(x)`	Повертає арктангенс x
`cmath.atanh(x)`	Повертає гіперболічний арктангенс x
`cmath.cos(x)`	Повертає косинус x
`cmath.cosh(x)`	Повертає гіперболічний косинус x
`cmath.exp(x)`	Повертає значення Ex, де E - число Ейлера
`cmath.isclose()`	Перевіряє, чи два значення близькі
`cmath.isfinite(x)`	Перевіряє, чи є x скінченним числом
`cmath.isinf(x)`	Перевіряє, чи є x нескінченним числом
`cmath.isnan(x)`	Перевіряє, чи є x NaN
`cmath.log(x[, base])`	Повертає логарифм x за основою
`cmath.log10(x)`	Повертає логарифм x за основою 10
`cmath.phase()`	Повертає фазу комплексного числа
`cmath.polar()`	Перетворює комплексне число в полярні координати
`cmath.rect()`	Перетворює полярні координати в прямокутну форму
`cmath.sin(x)`	Повертає синус x
`cmath.sinh(x)`	Повертає гіперболічний синус x
`cmath.sqrt(x)`	Повертає квадратний корінь x
`cmath.tan(x)`	Повертає тангенс x
`cmath.tanh(x)`	Повертає гіперболічний тангенс x

Константи cMath

Константа	Опис
`cmath.e`	Повертає число Ейлера (2.7182...)
`cmath.inf`	Повертає позитивну нескінченність
`cmath.infj`	Повертає комплексну нескінченність
`cmath.nan`	Повертає значення NaN (не число)
`cmath.nanj`	Повертає комплексне значення NaN
`cmath.pi`	Повертає число Пі (3.1415...)
`cmath.tau`	Повертає число тау (6.2831...)

Модуль Requests в Python

Приклад

Зробіть запит до веб-сторінки та виведіть текст відповіді:

```
import requests

x = requests.get('https://w3schools.com/python/demopage.htm')

print(x.text)
```

Опис

Модуль `requests` дозволяє надсилати HTTP-запити за допомогою Python.

HTTP-запит повертає об'єкт `Response` з усіма даними відповіді (вміст, кодування, статус тощо).

Завантаження та встановлення модуля Requests

Перейдіть до командного рядка, де знаходиться PIP, і введіть наступне:

```
pip install requests
```

Синтаксис

```
requests.methodname(params)
```

Методи

Метод	Опис
`delete(url, args)`	Надсилає DELETE-запит до вказаного URL
`get(url, params, args)`	Надсилає GET-запит до вказаного URL
`head(url, args)`	Надсилає HEAD-запит до вказаного URL
`patch(url, data, args)`	Надсилає PATCH-запит до вказаного URL
`post(url, data, json, args)`	Надсилає POST-запит до вказаного URL
`put(url, data, args)`	Надсилає PUT-запит до вказаного URL
`request(method, url, args)`	Надсилає запит вказаного методу до вказаного URL

Модуль Statistics в Python

Python має вбудований модуль для обчислення математичної статистики числових даних.

Модуль `statistics` був доданий у Python 3.4.

Методи статистики

Метод	Опис
`statistics.harmonic_mean()`	Обчислює гармонійне середнє даних
`statistics.mean()`	Обчислює середнє значення даних
`statistics.median()`	Обчислює медіану даних
`statistics.median_grouped()`	Обчислює медіану згрупованих даних
`statistics.median_high()`	Обчислює високу медіану даних
`statistics.median_low()`	Обчислює низьку медіану даних
`statistics.mode()`	Обчислює моду даних
`statistics.pstdev()`	Обчислює стандартне відхилення для всієї популяції
`statistics.stdev()`	Обчислює стандартне відхилення для вибірки
`statistics.pvariance()`	Обчислює дисперсію для всієї популяції
`statistics.variance()`	Обчислює дисперсію для вибірки

Що таке віртуальне середовище?

Віртуальне середовище в Python — це ізольоване середовище на вашому комп'ютері, де ви можете запускати та тестувати свої Python-проєкти.

Переваги використання віртуального середовища:

- Управління залежностями для кожного проєкту окремо.
- Запобігання конфліктам версій пакетів між проєктами.
- Збереження чистоти системної установки Python.
- Можливість тестування з різними версіями Python.

Як створити віртуальне середовище?

Python має вбудований модуль `venv` для створення віртуальних середовищ.

1. Відкрийте командний рядок і перейдіть до папки, де хочете створити проєкт.
2. Виконайте команду для створення віртуального середовища:

Приклад:

```
python -m venv myfirstproject
```

Це створить папку `myfirstproject` зі структурою:

```
myfirstproject
    Include
    Lib
    Scripts
    .gitignore
    pyvenv.cfg
```

Активація віртуального середовища

Для активації віртуального середовища виконайте команду:

Windows:

```
myfirstproject\Scripts\activate
```

macOS/Linux:

```
source myfirstproject/bin/activate
```

Після активації командний рядок зміниться, відображаючи активне середовище:

```
(myfirstproject) C:\Users\Your Name>
```

Встановлення пакетів

Після активації середовища можна встановлювати пакети за допомогою `pip`. Наприклад, встановимо пакет `cowsay`:

Приклад:

```
pip install cowsay
```

Використання пакетів

Створіть файл `test.py` із наступним кодом:

```
import cowsay
```

```
cowsay.cow("Good Mooooorning!")
```

Запустіть файл у віртуальному середовищі:

```
python test.py
```

Результат:

```
  _________________
| Good Mooooorning! |
  =================
                          \
                             \
                                 ^__^
                                 (oo)\_______
                                 (__)\       )\/\
                                         ||----w |
                                         ||     ||
```

Деактивація віртуального середовища

Для виходу з віртуального середовища виконайте:

```
deactivate
```

Видалення віртуального середовища

Щоб видалити віртуальне середовище, просто видаліть його папку:

```
rmdir /s /q myfirstproject
```