§10 МОДУЛІ

Один з основних принципів у сучасному програмуванні, що дозволяє спростити розробку та підвищити універсальність програм це принцип модульності, згідно з яким програма розділяється на окремі іменовані блоки, що називаються модулями.

Модуль – функціонально завершений фрагмент програми, що оформлено у вигляді окремого файла з вихідним кодом, призначений для використання в інших програмах.

Модулі дозволяють розділяти складні задачі на дрібніші. Вони проектуються таким чином, щоб надати програмістам зручний для багаторазового використання функціонал (інтерфейс) у вигляді набору підпрограм, змінних, класів тощо. Програма, що має модульну структуру, зазвичай складається з сукупності модулів, серед яких виділяють головний модуль.

Отже, модулі виконують принаймні дві важливі функції:

- Повторне використання коду без необхідності явного його дублювання.
- Керування адресний простором – модуль це високорівнева організація програм, що дозволяє уникнути конфліктів у іменуванні змінних, функцій, класів, тощо.

У Python модуль це окремий файл, що містить вихідний код – змінні, підпрограми, класи, алгоритми тощо. Отже, створення власного модуля нічим не відрізняється від написання звичайної програми в окремому файлі. Всі об'єкти модуля можуть бути використані в головній програмі або інших модулях. Для цього їх потрібно імпортувати. Можна імпортувати модуль повністю або його окремі об'єкти. Імпорт проводиться з використанням імені модуля. Ім'я модуля – це ім'я файлу без розширення "`.py`".

Імпорт та використання модулів

Імпорт модуля

До цього моменту ми вже неодноразово використовували функції з різних модулів і вже знаємо, що імпорт модуля здійснюється за допомогою ключового слова **`import`**:

```
import modul
```

тут `modul` – ім'я модуля.

Як правило інструкцію імпорта модуля розміщують на початку програми.

Імпорт модуля фактично запускає виконання програми, що міститься у модулі. Тому не рекомендується імпортувати модулі отримані з ненадійних джерел. Це може пошкодити дані на вашому комп'ютері.

Під час такого імпорту, ми отримуємо доступ до всіх елементів імпортованого модуля. Кожен модуль утворює власний простір імен, що є глобальною областю видимості для всіх визначених у ньому об'єктів. Для того, щоб об'єкти цього модуля не потрапили у конфлікт з іншими глобальними іменами або іншими модулями, під час доступу до елементів модуля потрібно використовувати префікс, що складається з імені модуля разом з крапкою, наприклад

```
modul.func1()
```

тут `modul` – ім'я модуля, `func1` – функція, що міститься у цьому модулі.

```
import math
print(math.pi)
```

Однією інструкцією **`import`** можна імпортувати кілька модулів. У такому разі імена модулів перелічуються через кому:

```
import modul_1, modul_2, modul_3
```

Проте такий підхід не є рекомендованим, оскільки це знижує читабельність коду. Порівняйте візуально вищенаведену інструкцію імпорта з таким

```
import modul_1
import modul_2
import modul_3
```

Використання псевдонімів

Якщо ім'я модуля занадто довге або таке, що може призвести до конфлікту з іншими об'єктами програми то для нього можна створити псевдонім (аліас) за допомогою ключового слова **`as`**:

```
import math as m
```

При цьому всі елементи цього модуля стають доступними у програмі через ім'я `m`, а не через `math`:

```
print(m.pi) # замість префікс math вказується m
```

Фактично створення аліасу модуля, це зміна імені модуля при імпорті.

Імпорт окремих елементів модуля

У Python є можливість підключення не всього модуля, а його окремих елементів. Для цього використовується інструкція **from**, що має такий синтаксис

```
from modul import obj
```

тут `modul` – ім'я модуля, `obj` – об'єкт (змінна, функція, клас тощо) що міститься у цьому модулі.

У разі імпорту з використанням інструкції **from**, об'єкт доступний за іменем `obj` (без префіксу, на відміну від випадку імпорту всього модуля). Наприклад

```
from math import pi
print(pi)       # префікс math не вказується
```

Можна одночасно імпортувати кілька об'єктів з одного модуля. Тоді імпортовані об'єкти перелічуються через кому

```
from math import pi, e, factorial
```

Для імпорту всіх об'єктів модуля використовується інструкція

```
from modul import *
```

Відмінність від стандартного імпорту модуля полягає у тому, що всі об'єкти модуля доступні за своїми іменами без префіксу (без імені модуля).

```
from math import *
print(pi)           # префікс math не вказується
print(factorial(4)) # префікс math не вказується
```

Звернемо увагу, що під час такого способу імпорту об'єктів з модулів можливе перекриття імен, якщо два модулі надають для імпорту одне і те ж ім'я об'єкту.

Під час імпорту окремих елементів, з використанням інструкції **from**, для них можна створювати псевдоніми за допомогою ключового слова **as**. Наприклад,

```
from math import factorial as f
print(f(4))
```

Також використанні псевдонімів вирішує проблему імпорту об'єктів з різних модулів, що мають однакові імена. Імпорт об'єктів модуля з використанням інструкції **from** робить імпортовані об'єкти доступними лише для читання.

Повторний імпорт модулів

Потрібно пам'ятати, що модуль завантажується лише один раз під час виконання програми, незалежно від того, скільки разів інструкція для його імпорту трапляється у програмі. Завантаження відбувається при виконанні першої інструкції імпорту і всі наступні операції імпорту цього модуля будуть повертати вже завантажений об'єкт модуля, навіть якщо сам модуль при цьому було змінено.

Тому, якщо необхідно повторно імпортувати модуль, використовують інструкцію `reload()` модуля `importlib`:

```
from importlib import reload
reload(modul)
```

Створення модулів

Як вже стало зрозуміло, створення власного модуля не вимагає від програміста якихось додаткових зусиль. Проте існує кілька правил, яких потрібно дотримуватися при створенні власних модулів:

- Під час іменування модуля потрібно дотримуватися тих же правил, що і під час створення змінних – можна використовувати лише латинські літери, цифри та символ "_", причому першим символом у імені модуля має бути не цифра.
- Не можна називати модуль так як вбудовані функції чи ключові слова.
- Розташовувати модуль потрібно у файловій системі так, щоб до нього був доступ для імпорту з вашої програми або інших модулів, що його використовують.

Розглянемо приклад створення модуля, що описує дві функції – піднесення числа до квадрату та кубу. Створимо файл з кодом Python, з іменем `my_module.py`:

```
def my_pow2(x):
    return x ** 2

def my_pow3(x):
    return x ** 3
```

Модуль `my_module` готовий. Використаємо його у нашій програмі. Створимо інший файл з іменем `main.py`, що буде містити такий код

```
import my_module  # імпорт модуля

res2 = my_module.my_pow2(2.0) # використання функції з модуля
res3 = my_module.my_pow3(2.0) # використання функції з модуля
```

Головний модуль програми

Головним називається модуль з якого починається виконання програми. На відміну від імпортованих, ім'я яких збігається з іменем файлу у якому він міститься, головний модуль має зарезервоване ім'я "`__main__`". Це ім'я міститься у спеціальній змінній `__name__`, яка створюється під час імпорту або запуску для кожного модуля.

Як було зазначено раніше, під час імпорту модуля, його код повністю виконується. Тому, аналіз імені модуля дозволяє виконувати або не виконувати певні набори інструкцій в залежності від того чи є модуль головним. Наприклад, у випадку виконання інтерпретатором такого коду

```
if __name__ == "__main__":
    process_some_code
```

набір інструкцій `process_some_code` буде виконано лише у тому випадку, якщо програма виконується як головний модуль, а не імпортований.

Розташування модулів у файловій системі.

Python поширюється з бібліотекою стандартних модулів, що налічує понад 200 модулів. Вони забезпечують підтримку таких задач як інтерфейс до операційної системи, керування об'єктами, робота в мережі, інтернет, графічний інтерфейс тощо.

Стандартні модулі вбудовані у інтерпретатор. Для їх використання потрібно лише їх імпортувати у програмі командою `import`.

Крім стандартних, існує ціла низка додаткових спеціалізованих модулів, які призначено для розв'язання різноманітних задач. Такі модулі потрібно додатково встановлювати. Процедура встановлення такого модуля залежить

від операційної системи, що використовується і, як правило, описується у документації модуля.

Під час імпорту модуля, інтерпретатор шукає файл з модулем у папках у такій послідовності

- Серед стандартних вбудованих модулів (каталоги Python).
- У папці програми, що імпортує модуль.
- У папках, що зазначені у змінній оточення PYTHONPATH операційної системи.

Список цих папок міститься у змінній `sys.path`.

```
>>> import sys
>>> print(sys.path)
['D:\\Repository\\Test', 'C:\\IDE\\python\\python35.zip',
'C:\\IDE\\python\\DLLs', 'C:\\IDE\\python\\lib',
'C:\\IDE\\python', 'C:\\IDE\\python\\lib\\site-packages']
```

Отже, є можливість під час виконання вашої програми розширити список каталогів, де інтерпретатор буде шукати модулі для імпорту, додавши у змінну `sys.path` новий каталог

```
>>> sys.path.append('C:/my_modules')
```

Також можна задати або доповнити змінну оточення PYTHONPATH у операційній системі шляхами до каталогів де знаходяться користувацькі модулі.

Пакети

Пакет – це спосіб структурування простору імен модулів на базі файлової системи.

Отже, якщо модулі це файли, що містять вихідний код, то пакети – це каталоги, що містять модулі. Пакети формуються у вигляді ієрархії папок, що містять модуль.

Пакети дозволяють структурувати колекції модулів, що утворюють великі бібліотеки – пакетний імпорт робить код більш читабельним і значно пришвидшує пошук.

Застосування пакетів робить безпечним використання імен модулів у багатомодульних пакетах.

Структура пакета

Розглянемо приклад пакета **TCP** – кореневий каталог **TCP** (Рис.10.1). У ньому знаходиться два підкаталоги **Server** і **Client**. Кожен з каталогів містить файл (модуль) `__init__.py` Цей файл може бути або порожнім або містити код ініціалізації пакета – код котрий виконується під час імпорту пакету. У ранніх версіях Python файл `__init__.py` вказував, що каталог, який його містить, є пакетом. Починаючи з версії 3.3 файл `__init__.py` є необов'язковим.

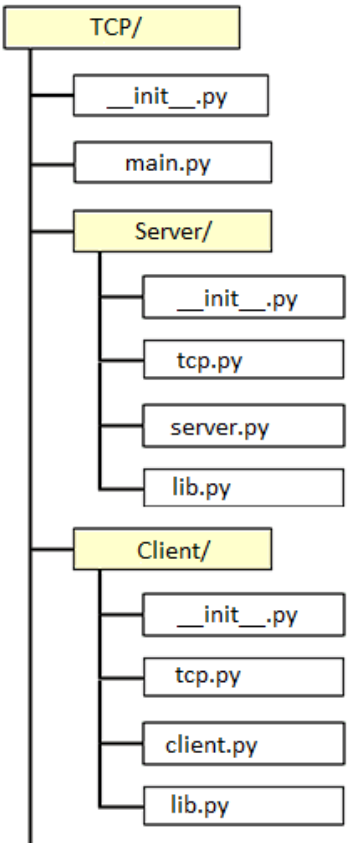

Рис. 10.1. Структура пакета TCP.

Для того, щоб імпортувати модуль, потрібно вказати повний шлях (з урахуванням вкладення всіх каталогів пакета) до модуля, що імпортується

через символ "." (крапку). Наприклад, імпорт модуля `lib.py` з пакету **Client** буде виглядати таким чином:

```
>>> import TCP.Client.lib
```

При цьому посилання на функцію у програмі також має бути повним:

```
>>> TCP.Client.lib.connect()
```

Такий спосіб часто буває не зручним з огляду на його громіздкість, тому рекомендується при імпорті використовувати вищеописаний механізм псевдонімів (аліасів). Наприклад,

```
>>> import TCP.Client.lib as cl_lib
```

Тоді відповідно виклик функції `connect()` з імпортованого модуля буде виглядати як

```
>>> cl_lib.connect()
```

Також звернемо увагу, що у пакеті містяться два модулі з однаковими іменами – `lib.py`. Проте, зважаючи на спосіб імпорту модулів, жодного конфлікту у програмі не виникне – ці два модулі знаходяться у різних папках пакету.

Задачі для самостійної роботи

10.1. Опишіть модуль, у якому описати функції для обчислення значень елементарних математичних функцій (див. задачу 3.59). Використовуючи цей модуль, знайдіть значення арифметичного виразу:

$$\frac{\ln 4 + e^3 - \sin 3^{3/2}}{\cos 2 + \mathrm{ch} 4}.$$

10.2. Опишіть модуль роботи з квадратними матрицями та векторами. У модулі опишіть такі:
операції введення/виведення:

a) зчитування вектора з клавіатури;
b) зчитування матриці з клавіатури;
c) зчитування вектора з текстового файла;
d) зчитування матриці з текстового файла;
e) виведення вектора на екран;
f) виведення матриці на екран;
g) запис вектора у текстовий файл;