

ОСНОВИ СИСТЕМ ШТУЧНОГО ІНТЕЛЕКТУ, НЕЙРОННИХ МЕРЕЖ ТА ГЛИБОКОГО НАВЧАННЯ

Частина 6. ВИСОКОРІВНЕВА МОВА ПРОГРАМУВАННЯ PYTHON

Лекція 6.5. Інтерпретація скриптів. Модулі та типові пакети.

ІНТЕРПРЕТАЦІЯ СКРИПТІВ

Інтерпретатор (interpreter) – програма, необхідна для виконання інших програм, вид транслятору, який здійснює пооператорну (покомандну, строкову) обробку, перетворення у машинний код та виконання програми.

Компілятор (compiler) — програма (набір програм), що перетворює (компілює) деякий вхідний код, написаний певною мовою програмування (*source language*), на семантично еквівалентний код в іншій мові програмування (цільова мова, *target language*), з подальшою можливістю виконання програми комп'ютером.

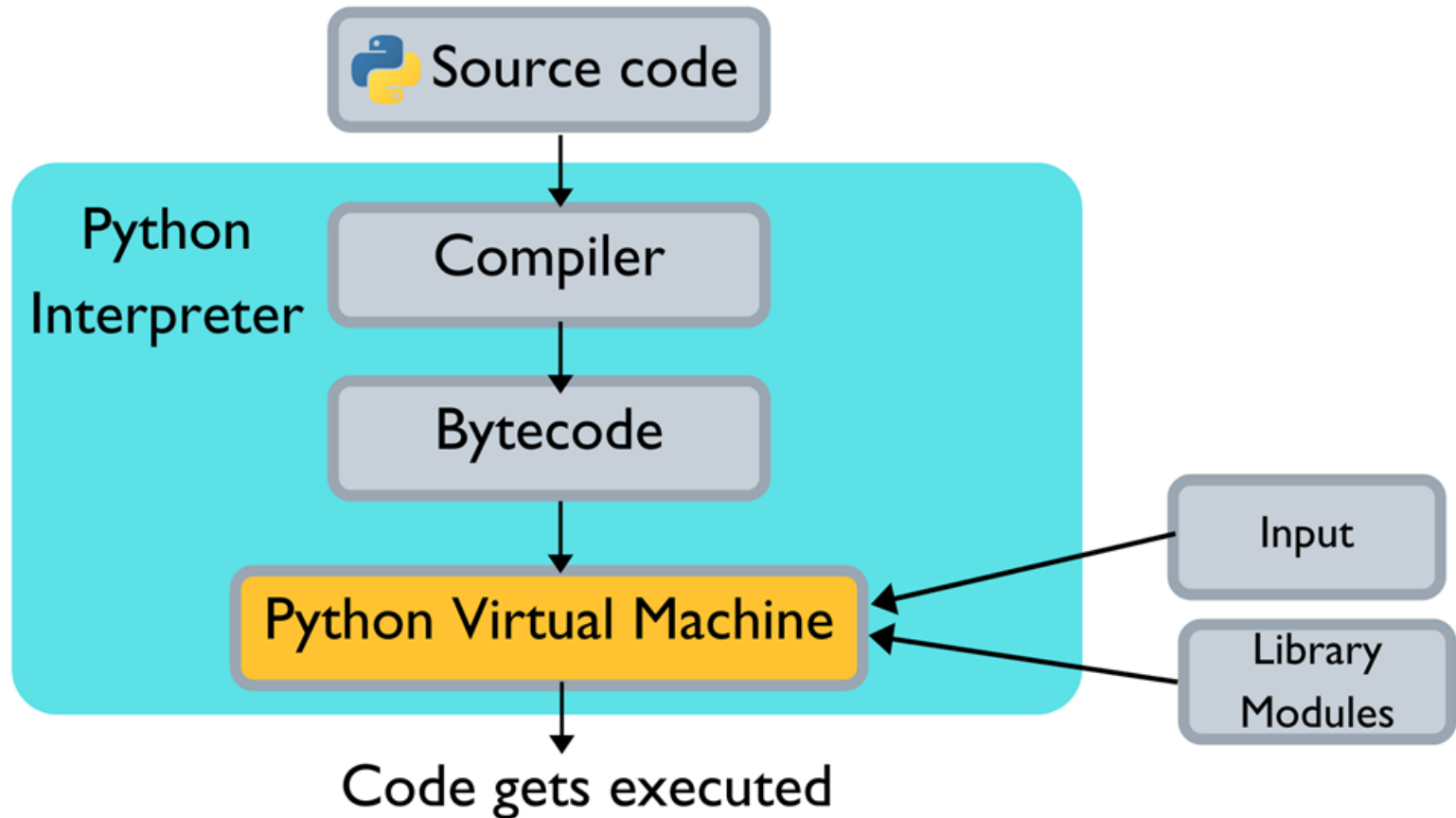
ІНТЕРПРЕТАЦІЯ СКРИПТІВ

Простий інтерпретатор – аналізує і відразу виконує (власне інтерпретація) програму покомандно (або порядково), по мірі надходження тексту програми на вхід інтерпретатора.

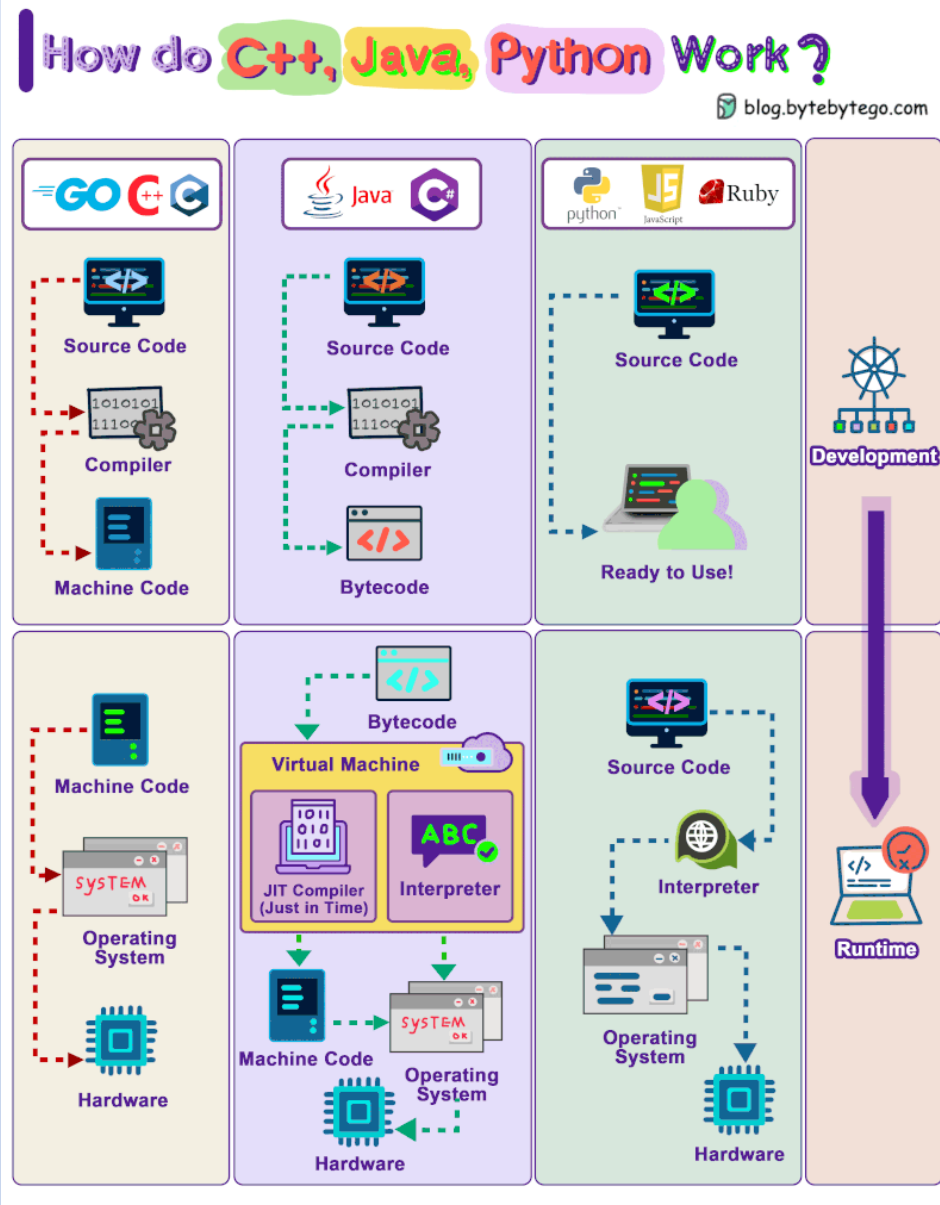
Інтерпретатор компілюючого типу — це система з компілятора, який перекладає текст програми в деяке проміжне представлення (найчастіше в **байт-код**), і власне інтерпретатора, який виконує отриманий проміжний код - так звана віртуальна машина.

Загальна думка: машинний код набагато швидше, але байт-код краще захищений та є переносним.

ІНТЕРПРЕТАЦІЯ СКРИПТІВ



ІНТЕРПРЕТАЦІЯ СКРИПТІВ



ВИКОНАННЯ PYTHON ПРОГРАМИ

Вхідний текст

```
Def foo(x,y)  
  X = 5  
  Y = 7  
  Z=X*Y  
  PRINT(Z)
```

my_mod.py

Компіляція

Байт – код

b'd\x01}\x02d\x02}\x03|.....

```
0 LOAD_CONST  
2 STORE_FAST  
4 LOAD_CONST  
6 STORE_FAST  
...  
8 LOAD_FAST  
10 LOAD_FAST  
12 BINARY MULTIPLAY  
...  
20 CALL FUNCTION
```

my_mod.pyc

Виконання

PVM

Інтерпретація

35

РЕАЛІЗАЦІЇ PYTHON

Implementation	Virtual Machine	Compatible Lang
CPython	CPython VM	C
Jython	JVM	Java
IronPython	CLR	C#
Brython	Javascript Eng.	JavaScript
RubyPython	RubyVM	Ruby
PyPy		Python
JupyterLab Spyder		
IPython / IP[y]	An enhanced Interactive Python. 7.13.0 Released on Feb 29, 2020.	C

МОДУЛІ

ПРОГРАМА

Модуль 1

Модуль 2

**Сукупність
інструкцій**

modul_1.py

```
def func(mssg):  
    print('Mod 1', mssg)
```

modul_2.py

```
def func(mssg):  
    print('Mod 2', mssg)
```

my_main.py

```
import modul_1  
modul_1.func('OhOhOh')
```


МОДУЛІ

ПРОГРАМА

Модуль 1

Модуль 2

**Сукупність
інструкцій**

my_main.py

modul_1.py

```
def func(mssg):  
    print('Mod 1', mssg)
```

modul_2.py

```
def func(mssg):  
    print('Mod 2', mssg)
```

```
import modul_1  
import modul_2  
modul_1.func('OhOhOh')  
modul_1.func('HaHaHa')
```

ПРОСТІР ІМЕН МОДУЛЯ

МОДУЛЬ → ПАКЕТ ІМЕН

- ❑ Інструкції модуля виконуються під час першої спроби імпорту. Перший імпорт - інтерпретатор Python створює **порожній** об'єкт модуля і виконує інструкції в модулі одну за одною, від початку файлу до кінця.
- ❑ Операції присвоювання (не вкладені в `def ()` або `class()`), що виконуються на верхньому рівні, створюють атрибути об'єкта модуля і зберігаються в просторі імен модуля.
- ❑ Область видимості модуля (простір імен) після завантаження модуля перетворюється в **атрибут-словник об'єкта модуля**. Доступ до простору імен модуля можна отримати через атрибут `__dict__` або `dir (M)`.

ІНСТРУКЦІЇ ІМПОРТУ

	Дія
<code>import <module></code>	Файл <i>module</i> завантажується та перетворюється в <i>ім'я змінної</i> , яка посилається на повний <i>об'єкт</i> модуля із усіма його атрибутами
<code>import <module> as <mod_name></code>	<i>ім'я змінної := mod_name</i> (нові, короткі імена)
<code>from <module> import *</code>	Завантажуються <i>всі</i> об'єкти-функції модуля, перетворюються в імена змінних та копіюються в область видимості програми
<code>4. from <module> import <fun1, fun2, ...></code> Імпортуються окремі функції
<code>5 from <module> Import <fun1, fun2,...> as F1,F2,..</code> Імпортуються окремі функції з новими іменами

ІНСТРУКЦІЇ ІМПОРТУ

import , from → !!! інструкції (як і всі інші)

from ...! вставляє простір імен модуля в простір імен програми.

import ...! створює єдине ім'я в просторі імен програми що посилається на об'єкт-модуль (на простір імен модуля).

! ПЕРЕВАГУ
інструкції **import**



Функція	Дія
from <i>imp</i> import <i>reload</i> <i>reload</i> <module>	reload примусово виконує повторну завантаження вже завантаженого модуля і запускає його програмний код. Інструкції присвоювання, що виконуються при повторному запуску, будуть змінювати існуючий об'єкт модуля.

ІМПОРТ МОДУЛЯ

Крок	Дія
1.Пошук	<i>Пошук модуля в робочому середовищі</i>
2.Компіляція	<i>Перетворення в байт-код (*.рус), якщо це необхідно</i>
3.Запуск	<i>Виконується байт-код, створюються всі об'єкти модуля та, відповідно, простір імен модуля (атрибути всіх об'єктів модуля).</i>

ПОШУК МОДУЛЯ

Де шукати модуль ?

	Напрямки пошуку
1.	Домашня директорія програми
2.	Змінна оточення PYTHONPATH , якщо є
3.	Каталоги стандартних бібліотек
4.	Зміст будь-яких файлів *.pht, якщо вони є

import sys

sys.path – перегляд шляхів пошуку

СИНТАКСИС КВАЛІФІКАЦІЇ ІМЕН

Для доступу до атрибутів будь-якого об'єкта використовується синтаксис кваліфікації імені *object.attribute* -> це вираз, що повертає значення, яке присвоєно імені атрибуту.

Звернення до імен, кваліфікуючи їх, явно вказує інтерпретатору об'єкт, атрибут якого потрібно використати.

- Прості змінні (правило **LEGB**), наприклад **X** - пошук імені **X** в поточних областях видимості.
- Кваліфіковані імена **X.Y** - пошук імені **X** спочатку в поточних областях видимості, а потім буде виконаний пошук атрибута **Y** в об'єкті **X** (не в областях видимості).
- Кваліфіковані шляхи **X.Y.Z** - буде проведений пошук імені **Y** в об'єкті **X**, а потім пошук імені **Z** в об'єкті **X.Y**.

ПАКЕТ МОДУЛІВ

ПАКЕТ → директорія (каталог), в якому розташовані модулі

.../work_dir/dir

import dir.modul_1

__init__.py

modul_1

modul_2

__init__.py призначений для виконання дій по першій ініціалізації пакету, створення простору імен для каталогу і реалізації поведінки інструкцій імпорту, наприклад створення файлів, підключення до БД та інше.

МОДУЛЬ як СКРИПТ

Кожен модуль має вбудований атрибут **__name__**, який встановлюється

→ **<modul>**, якщо файл модуля імпортується

→ **"__main__"**, якщо файл модуля запускається як головний файл програми.

Тобто модуль може перевірити власний атрибут **__name__** і визначити, чи був він запущений як самостійна програма або імпортовано іншим модулем.

В модулі можна

```
if __name__ == "__main__":  
    .... # що потрібно  
    ....
```

Рекомендована ЛІТЕРАТУРА

- **Програмування числових методів мовою Python:** підруч. / А. В. Анісімов, А. Ю. Дорошенко, С. Д. Погорілий, Я. Ю. Дорогий ; за ред. А. В. Анісімова. – К. : Видавничо-поліграфічний центр "Київський університет", 2014. – 640 с.
- **Програмування числових методів мовою Python:** навч. посіб. / А. Ю. Дорошенко, С. Д. Погорілий, Я. Ю. Дорогий, Є. В. Глушко ; за ред. А. В. Анісімова. – К. : Видавничо-поліграфічний центр "Київський університет", 2013. – 463 с.
- **Основи програмування Python:** Підручник для студ. спеціальності 122 «Компютерні науки» / А.В.Яковенко; КПІ.- Київ: КПІ, 2018 . – 195 с.
- **Лутц М.** Изучаем Python, 4-е издание. - СПб.: Символ-Плюс. 2011.- 1280 с.: ил.

Контрольні запитання

- Надайте визначення компілятора та інтерпретатора. Наведіть порядок виконання Python скриптів.
- Обґрунтуйте необхідність та вкажіть переваги використання модулів.
- Надайте перелік інструкцій імпорту модулів, поясніть їх відмінності та наведіть відповідні приклади.
- Наведіть способи завдання шляхів пошуку модуля.
- Поясніть сутність поняття кваліфіковані імена та наведіть відповідні приклади.
- Надайте призначення пакетів модулів та поясніть порядок створення пакетів.

The END

Частина 6. Лекція 6.5.