



TIN HỌC CƠ SỞ

Phần 2: Ngôn ngữ lập trình Python

CHƯƠNG 12.

MODULE VÀ PACKAGE TRONG PYTHON

Tin cơ sở (LT): 010100229802 - DHKL16A1HN, - DHKL16A2HN

GV. Cao Diệp Thắng

Mục tiêu chương

Nắm vững:

- ☐ Khái niệm, cách tạo, sử dụng module
- ☐ Khái niệm namespace và scope
- ☐ Package của python
- ☐ Thư viện chuẩn,



Nội dung

- 1 Khái niệm module
- 2 Xây dựng/import module
- 3 Namespace & Scope
- 4 Package
- 5 Python standar library



12.1. MODULE VÀ PACKAGE

12.1.1. Khái niệm Module:

- ❑ Modules là cách mà lập trình viên phân hóa chương trình ra các nhánh nhỏ để dễ quản lý và gọi lại chúng khi nào cần, như vậy chương trình của chúng ta sẽ có tính tái sử dụng, bảo trì cao.
- **Module**: cho phép tổ chức code của Python một cách logic. Việc nhóm các code có liên quan vào cùng một module giúp cho code dễ hiểu và dễ sử dụng.
- **Module** là một đối tượng Python mà lập trình viên có thể kết vào (import) và tham chiếu tới.
- Nói chung, **module** là một tập tin chứa code Python. Trong module có thể định nghĩa **function**, **class**, **variable**
- Một module cũng có thể chèn (include) runnable code.





12.1.2. Xây dựng Module

Ví dụ 12.1

Xây dựng module **my_module.py** có chứa **function** **tingh_bmi(can_nang, chieu_cao)**

```
1  #Xây dựng Module my_module.py có chứa hàm tingh_bmi(can_nang, chieu_cao)
2  import math
3  def tingh_bmi(can_nang, chieu_cao):
4      bmi = can_nang/math.pow(chieu_cao,2)
5      return bmi
```

12.1.3. Import Module

a. Import module: Để import một module đã có (có sẵn trong Python hoặc đã được viết bởi người dùng) vào trong file hiện tại để sử dụng, chúng ta dùng từ khóa **import** với cú pháp như sau [5,6,7]:

Cú pháp: **import** module1, module2...

Trong đó: module1, module2,... là các modules mà lập trình viên muốn import vào file hiện tại.

- Import được đặt ở đầu script
- Module chỉ được load một lần, không phụ thuộc vào số lần nó được import. Điều này ngăn ngừa việc thực hiện module xảy ra nhiều lần.

12.1.3. Import Module,...

Ví dụ 12.2.: giả sử đã viết một module là **my_module**, khi đó để sử dụng được module mymodule, thực hiện như sau:

```
1  #import module my_module
2  import my_module
3  #Gọi hàm trong my_module
4  print(my_module.tinh_bmi(52,1.6))
```

Kết quả :

20.312499999999996



a. *from...import*

Giả sử trong một trường hợp nào đó lập trình viên không muốn sử dụng hết toàn bộ module mà *chỉ muốn sử dụng một số hàm, phương thức,...của module*, khi đó chúng ta sử dụng từ khóa ***from...import*** theo cú pháp như sau:

Cú pháp:

```
from module_name import function_name1, [,function_name2, ...]
```

Trong đó:

- `module_name` là tên của module muốn import.
- `Function_name1,function_name2,...` là những hàm ltv muốn sử dụng trong `module_name`.

Ví dụ 12.3

```
chuong_12 > 🐍 tinh_toan_hai_so.py > ...  
1   #Module tính toán hai số  
2   import math  
3   def tong(x,y):  
4       |       return x+y  
5   def hieu(x,y):  
6       |       return x-y
```



12.1.3. Import Module,...



Ví dụ 12.4. Từ module **tinh_toan_hai_so** đưa vào 2 function **tổng, hiệu**

```
1  #Ví dụ từ module tinh_toan_hai_so đưa vào hai hàm tổng, hiệu
2  from tinh_toan_hai_so import tong, hieu
3  x,y = 3,5
4  print("Tổng  ",x, "+ ", y, " = ", tong(x,y))
5
6  print("Hiệu  ",x, "- ", y, " = ", hieu(x,y))
```

Kết quả:

Tổng	3	+	5	=	8
Hiệu	3	-	5	=	-2

12.1.3. Import Module,...

c. *from...import**

Trong trường hợp muốn import tất cả function/thuộc tính trong modules có và cho phép thì chúng ta sử dụng keyword ***** để import vào **namespace** hiện tại

Cú pháp: `from module_name import*`

Ví dụ từ module **tinhtoa_hai_so** đưa vào tất cả các function/thuộc tính:

`from tinhtoa2so import*`

Ví dụ 12.5.

```
1  #Ví dụ: import tất cả các hàm thuộc tính vào namespace hiện tại
2  #từ module tinh_toan_hai_so
3  from tinh_toan_hai_so import*
4  a,b = 3,5
5  print("Tổng ", a, " + " ,b, " = ", tong(a,b))
6
7  print("Hiệu ", a, " - " ,b, " = ", hieu(a,b))
```



d. Vị trí lưu trữ module

- ❑ Khi import module, Python sẽ tìm kiếm module này ở những vị trí sau:
 - Thư mục hiện hành
 - Nếu không tìm thấy thì tiếp tục tìm trong từng thư mục trong biến **PYTHONPATH** (một biến môi trường với danh sách thư mục).
 - Nếu tiếp tục không tìm thấy thì tìm trong đường dẫn mặc định **/usr/local/lib/python/**.
 - Module tìm kiếm đường dẫn được lưu trữ trong module hệ thống **sys** có tên là **sys.path** variable. **sys.path** variable chứa thư mục hiện hành, PYTHONPATH, và những cài đặt phụ thuộc mặc định (installation-dependent default).

Để xem đường dẫn, nhập lệnh sau:

```
>>> import sys
```

```
>>> sys.path
```

```
['', 'C:\\Users\\PC\\AppData\\Local\\Programs\\Python\\Python36\\Lib\\idlelib',  
'C:\\Users\\PC\\AppData\\Local\\Programs\\Python\\Python36\\python36.zip',  
'C:\\Users\\PC\\AppData\\Local\\Programs\\Python\\Python36\\DLLs',  
'C:\\Users\\PC\\AppData\\Local\\Programs\\Python\\Python36\\lib',  
'C:\\Users\\PC\\AppData\\Local\\Programs\\Python\\Python36',  
'C:\\Users\\PC\\AppData\\Local\\Programs\\Python\\Python36\\lib\\site-packages']
```



12.1.4. Không gian tên - Namespace

Tên (names) và không gian tên (namespaces)

Python, tên (đôi khi còn được gọi là tên tượng trưng) là một mã định danh cho một đối tượng. Trong các chương trình chúng ta đã viết cho đến nay, đã sử dụng khái niệm đặt tên (names) rất nhiều! Xem xét một ví dụ:

```
color = "cyan"
```

Ở ví dụ trên, ta gán **color** làm tên của chuỗi “cyan”.

Python sử dụng hệ thống tên để nó có thể phân biệt giữa từng đối tượng riêng biệt (chẳng hạn như chuỗi hoặc hàm) mà chúng ta xác định. Trong hầu hết các chương trình, Python phải theo dõi hàng trăm và đôi khi thậm chí hàng nghìn tên. Vậy chính xác thì cách để Python lưu trữ tất cả thông tin này như thế nào?

→ Python tạo ra cái được gọi là không gian tên (**namespaces**).



a. Khái niệm không gian tên - namespaces

Một không gian tên là một tập hợp các tên và các đối tượng mà chúng tham chiếu. Python sẽ lưu trữ một từ điển trong đó các khóa là các tên đã được xác định và các giá trị được ánh xạ là các đối tượng mà chúng tham chiếu.

trên không gian tên mà Python tạo ra sẽ là một từ điển giống như sau:

```
{ 'color' : 'cyan' }
```

Trong trường hợp này, nếu chúng ta in ra giá trị của biến color bằng lệnh print(color),

```
>>> print(color)
cyan
```

Python sẽ tìm kiếm không gian tên được xác định ở trên để tìm khóa có tên color và cung cấp giá trị để chạy trong chương trình của chúng ta. Vì vậy, chúng ta sẽ thấy đầu ra là 'cyan'.





Giả sử ta có các đoạn mã script A, Script B như sau”

Script A

```
first_name = 'Nguyễn Văn'  
last_name = 'Minh'
```

```
def print_name():
```

namespaces A

```
{ first_name = 'Nguyễn Văn'  
  last_name = 'Minh'
```

```
  print_name():<function print_name at 0x39fx9s>  
}
```

Script B

```
num = 5
```

```
def countdown():
```

namespaces A

```
{ num : 5
```

```
  countdown():<function countdown at 0x38fx6s>  
}
```

b. Các loại không gian tên - namespace

Khi trình thông dịch Python chỉ chạy mà không có bất kỳ mô-đun, phương thức, lớp nào do người dùng định nghĩa, v.v. Một số hàm như **print()**, **id ()** luôn hiện diện, đây là những **name space** được tích hợp sẵn (**Built name space**). Khi người dùng tạo một mô-đun, một **name space chung (Global namespace)** sẽ được tạo ra, sau đó việc tạo các hàm cục bộ sẽ tạo ra name space cục bộ (**Local name space**).

Built-in namespace bao gồm **Global name space** và **Global namespace** bao gồm **Local name space** .[]

Built-in namespace

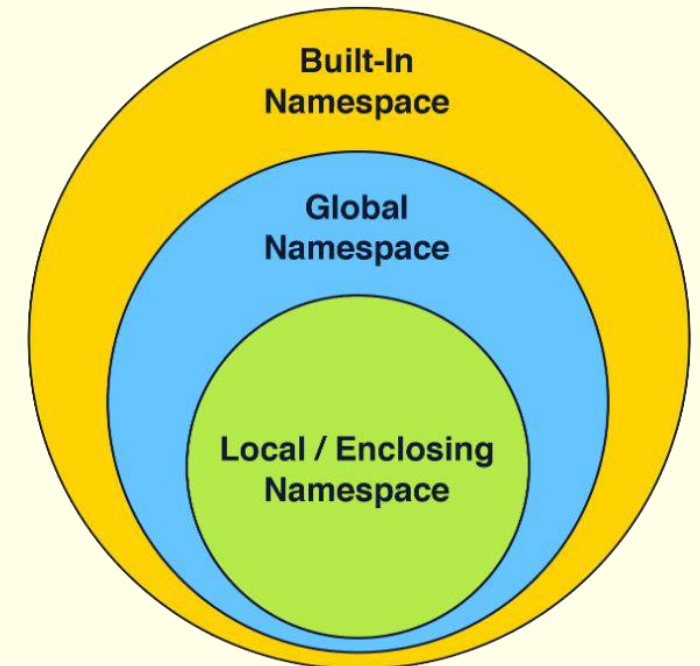
Không gian tên này bao gồm các hàm và tên ngoại lệ được tích hợp sẵn trong Python.

Global namespace

Không gian tên này bao gồm tên từ các mô-đun được import trong dự án. Nó được tạo khi chúng ta tạo ra mô-đun và nó kéo dài cho đến khi tập lệnh kết thúc.

Local/Enclosing namespace

Các tên cục bộ bên trong hàm nằm dưới một vùng tên cục bộ. Không gian tên này được tạo khi hàm được gọi và phạm vi kết thúc khi giá trị được trả về.

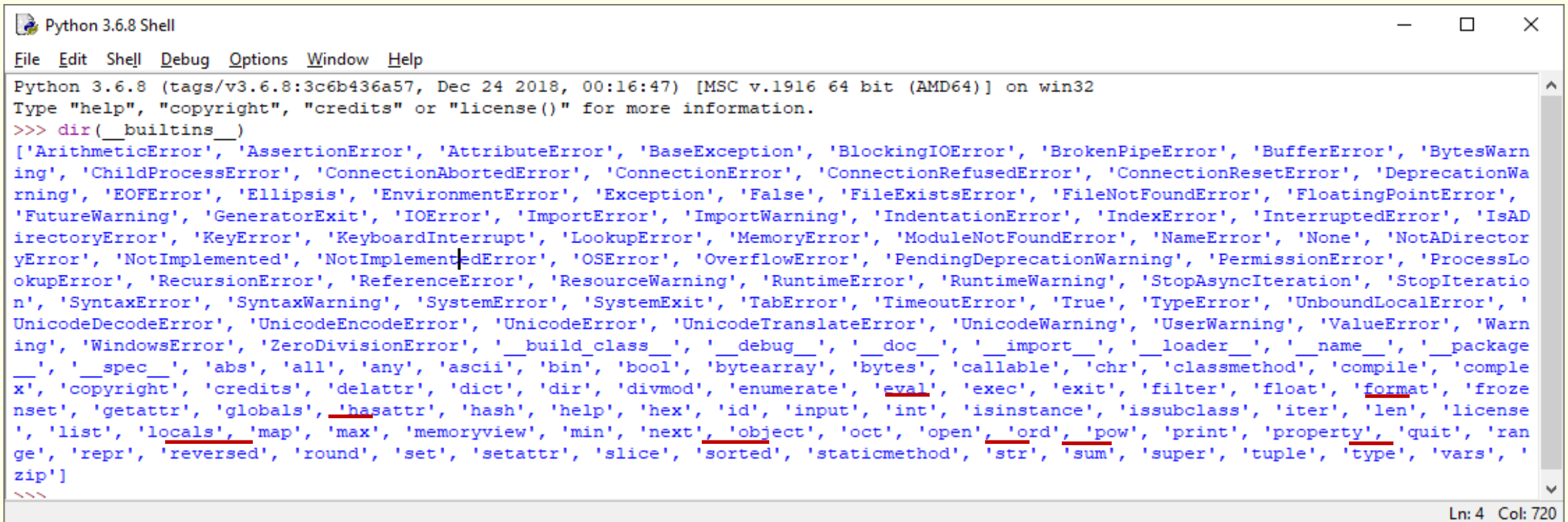


b. Các loại không gian tên – namespace,...

Built in namespace: Không gian tên dựng sẵn chứa tên của tất cả các đối tượng dựng sẵn (built in) của Python. Chúng luôn có sẵn khi Python đang chạy. Có thể liệt kê các đối tượng trong không gian tên dựng sẵn bằng lệnh:

```
>>> dir(__builtins__)
```

Kết quả: Có thể quan sát thấy các hàm tích hợp sẵn như max() và len() và các loại đối tượng như int và str,....



```
Python 3.6.8 Shell
File Edit Shell Debug Options Window Help
Python 3.6.8 (tags/v3.6.8:3c6b436a57, Dec 24 2018, 00:16:47) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> dir(__builtins__)
['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException', 'BlockingIOError', 'BrokenPipeError', 'BufferError', 'BytesWarning', 'ChildProcessError', 'ConnectionAbortedError', 'ConnectionError', 'ConnectionRefusedError', 'ConnectionResetError', 'DeprecationWarning', 'EOFError', 'Ellipsis', 'EnvironmentError', 'Exception', 'False', 'FileExistsError', 'FileNotFoundError', 'FloatingPointError', 'FutureWarning', 'GeneratorExit', 'IOError', 'ImportError', 'ImportWarning', 'IndentationError', 'IndexError', 'InterruptedError', 'IsADirectoryError', 'KeyError', 'KeyboardInterrupt', 'LookupError', 'MemoryError', 'ModuleNotFoundError', 'NameError', 'None', 'NotADirectoryError', 'NotImplemented', 'NotImplementededError', 'OSError', 'OverflowError', 'PendingDeprecationWarning', 'PermissionError', 'ProcessLookupError', 'RecursionError', 'ReferenceError', 'ResourceWarning', 'RuntimeError', 'RuntimeWarning', 'StopAsyncIteration', 'StopIteration', 'SyntaxError', 'SyntaxWarning', 'SystemError', 'SystemExit', 'TabError', 'TimeoutError', 'True', 'TypeError', 'UnboundLocalError', 'UnicodeDecodeError', 'UnicodeEncodeError', 'UnicodeError', 'UnicodeTranslateError', 'UnicodeWarning', 'UserWarning', 'ValueError', 'Warning', 'WindowsError', 'ZeroDivisionError', '__build_class__', '__debug__', '__doc__', '__import__', '__loader__', '__name__', '__package__', '__spec__', 'abs', 'all', 'any', 'ascii', 'bin', 'bool', 'bytearray', 'bytes', 'callable', 'chr', 'classmethod', 'compile', 'complex', 'copyright', 'credits', 'delattr', 'dict', 'dir', 'divmod', 'enumerate', 'eval', 'exec', 'exit', 'filter', 'float', 'format', 'frozenset', 'getattr', 'globals', 'hasattr', 'hash', 'help', 'hex', 'id', 'input', 'int', 'isinstance', 'issubclass', 'iter', 'len', 'license', 'list', 'locals', 'map', 'max', 'memoryview', 'min', 'next', 'object', 'oct', 'open', 'ord', 'pow', 'print', 'property', 'quit', 'range', 'repr', 'reversed', 'round', 'set', 'setattr', 'slice', 'sorted', 'staticmethod', 'str', 'sum', 'super', 'tuple', 'type', 'vars', 'zip']
>>>
```

- Trình thông dịch Python tạo không gian tên dựng sẵn (Built-in namespace) khi nó khởi động.
- Không gian tên này vẫn tồn tại cho đến khi trình thông dịch chấm dứt.

Global namespace : Không gian tên toàn cục chứa bất kỳ tên nào được xác định ở cấp độ của chương trình chính. Python tạo không gian tên chung khi phần thân chương trình chính bắt đầu và nó vẫn tồn tại cho đến khi trình thông dịch kết thúc.

Một cách chính xác, **Global namespace** có thể không phải là không gian tên toàn cục duy nhất tồn tại. Trình thông dịch cũng tạo một không gian tên chung cho bất kỳ module nào mà chương trình *import*.

Local namespace

Trình thông dịch tạo một không gian tên mới bất cứ khi nào một hàm thực thi. Không gian tên đó là cục bộ của hàm (**local namespace**) và vẫn tồn tại cho đến khi hàm kết thúc.



c. Thời gian tồn tại của một namespace

Thời gian tồn tại của một namespace phụ thuộc vào phạm vi của các đối tượng, nếu phạm vi của một đối tượng kết thúc, thời gian tồn tại của namespace đó cũng kết thúc. Do đó, không thể truy cập các đối tượng bên trong namespace từ một .

Ví dụ 12.6a.

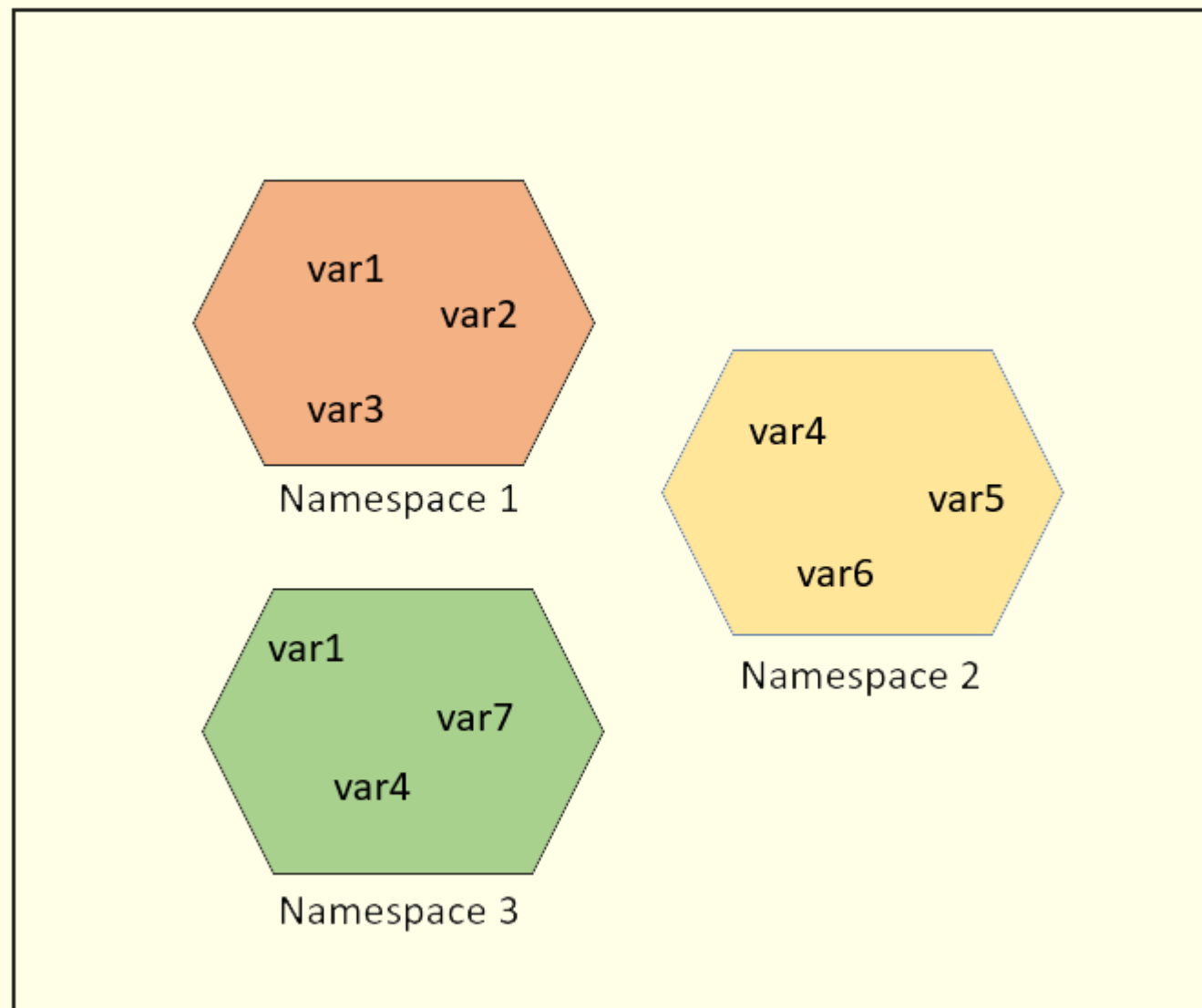
```
# var1 is in the global namespace
var1 = 5
def some_func():
    # var2 is in the local namespace
    var2 = 6
    print("giá trị biến var2 trong local name hàm some_func là : ", var2)
    def some_inner_func():
        #var3 is in the nested local namespace #
        var3 = 7
        print("Giá trị biến var3 trong nested local namespace là: ", var3)
    some_inner_func()
print("giá trị biến var trong không gian global namespace là: ", var1)
some_func()
```

Kết quả thực hiện ví dụ 12.6a

```
giá trị biến var trong không gian global namespace là: 5
giá trị biến var2 trong local name hàm some_func là : 6
Giá trị biến var3 trong nested local namespace là: 7
```



Trên thể hiện hình 12.2 chúng ta có thể thấy một tên đối tượng có thể cùng xuất hiện trong nhiều namespace vì sự tách biệt giữa các namespace của chúng.



Ví dụ 12.6b.

```
# var is in the global namespace
var = 5
def some_func():
    # var is in the local namespace
    var = 6
    print("giá trị biến var2 trong local hàm some_func là : ",var)
    def some_inner_func():
        #var3 is in the nested local namespace #
        var = 7
        print("Giá trị biến var3 trong nested local namespace là: ", var)
    some_inner_func()
print("giá trị biến var trong không gian global namespace là: ", var)

some_func()

print("Giá trị biến var sau khi gọi hàm some_func() là ", var)
```

Kết quả thực hiện ví dụ 12.6a

```
giá trị biến var trong không gian global namespace là: 5
giá trị biến var2 trong local hàm some_func là : 6
Giá trị biến var3 trong nested local namespace là: 7
Giá trị biến var sau khi gọi hàm some_func() là 5
```

- Để xem các tên được định nghĩa trong module, sử dụng hàm `dir()`

Cú pháp:

`dir(module_name)`

Ví dụ 12.7.

```
1  import my_module
2  all_names = dir(my_module)
3
4  print(all_names)
```

Kết quả:

```
['__builtins__', '__cached__', '__doc__',
'__file__', '__loader__', '__name__', '__package__',
'__spec__', 'math', 'tinh_bmi']
```



12.1.5. Phạm vi - scope

a. Khái niệm phạm vi

- ❑ Phạm vi đề cập đến vùng code mà từ đó đối tượng trong Python có thể truy cập được. Do đó, người ta không thể truy cập bất kỳ đối tượng cụ thể nào từ bất kỳ nơi nào trong code, việc truy cập phải được cho phép theo phạm vi của đối tượng.
- ❑ Với không gian tên, ta có thể xác định tất cả các tên bên trong một chương trình. Tuy nhiên, điều này không có nghĩa là chúng ta có thể sử dụng tên biến ở bất kỳ đâu. Các biến này cũng chỉ có thể được sử dụng trong một số phạm vi trong chương trình. Chúng ta có thể truy cập trực tiếp các biến mà không cần bất kỳ tiền tố nào khi chúng ở trong không gian tên.
- ❑ Thời gian tồn tại của một không gian tên trong Python phụ thuộc vào phạm vi của các đối tượng. Khi phạm vi của một đối tượng kết thúc, thời gian tồn tại của không gian tên cũng kết thúc. Vì vậy, chúng ta không thể truy cập các đối tượng của phạm vi không gian tên bên trong từ một không gian tên bên ngoài.



Ví dụ 12.8.

```
1 # Python program showing a scope of object
2 def some_func():
3     print("Inside some_func")
4     def some_inner_func():
5         var = 10
6         print("Inside inner function, value of var:", var)
7     some_inner_func()
8     print("Try printing var from outer function: ", var)
9 some_func()
```

var là biến cục bộ trong hàm con **some_inner_func()**, dòng lệnh 5 gán cho **var** có giá trị là 10

gọi hàm **some_inner_func()** bên trong hàm **some_func()**

Kết quả:

Inside some_func
Inside inner function, value of var: 10

Traceback (most recent call last):

```
File "<pyshell#2>", line 1, in <module>
    some_func()
```

```
File "<pyshell#1>", line 7, in some_func
    print("Try printing var from outer function: ", var)
```

NameError: name 'var' is not defined

dòng số 8 trong hàm **some_func()** thực hiện việc truy xuất biến **var** bên ngoài hàm con **some_inner_func()** sẽ gây ra lỗi vì biến **var** không tồn tại bên ngoài hàm con

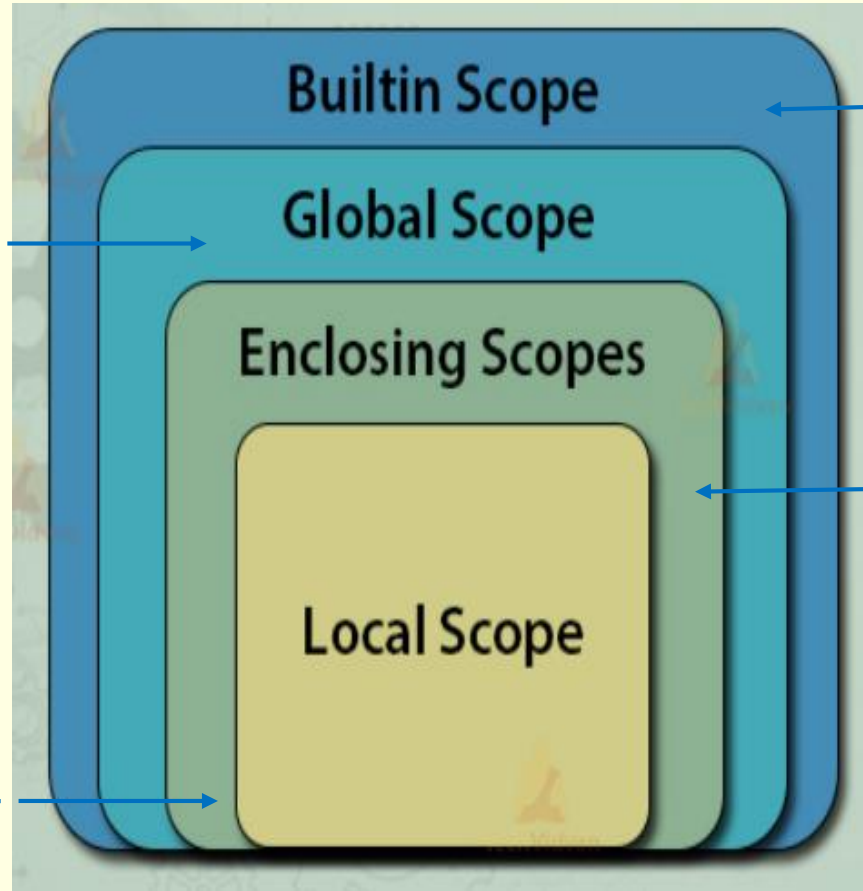


b. Các loại phạm vi

Khi một tên được tham chiếu trong Python , trình thông dịch sẽ tìm kiếm tên trong không gian tên bắt đầu từ phạm vi nhỏ nhất trong sơ đồ và dần di chuyển đến phạm vi ngoài cùng.

Global Scope- Phạm vi toàn cục là phạm vi cấp mô-đun. Khi **import** các mô-đun, chúng sẽ được thêm vào phạm vi toàn cục.

Local Scope - Phạm vi cục bộ, là phạm vi trong cùng chứa danh sách các tên cục bộ chỉ có sẵn trong hàm.



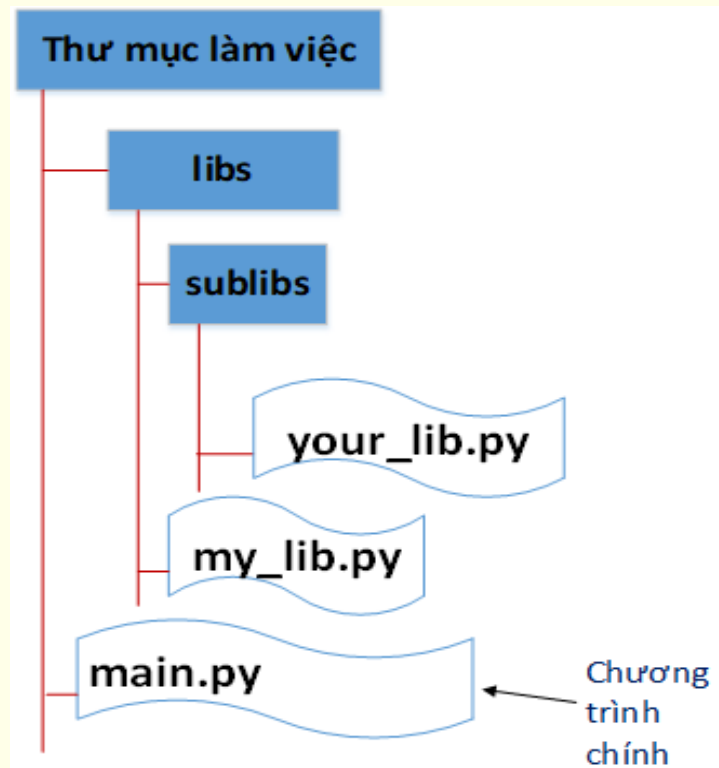
Builtin Scope - Phạm vi ngoài cùng là phạm vi tên dựng sẵn. Trình thông dịch tìm kiếm các tên trong phạm vi này cuối cùng.

Enclosing Scope - Phạm vi bao quanh được nhìn thấy trong các hàm lồng nhau nơi tên được tìm kiếm trong hàm bên ngoài gần nhất.



12.2. PACKAGE

Package trong Python là một thư mục chứa một hoặc nhiều modules hay các package khác nhau. Package được tạo ra nhằm mục đích phân bố các modules có cùng chức năng và để dễ quản lý source code.



```
import libs.sublibs.your_lib
#from libs.sublibs import your_lib
```

```
from libs import my_lib as mylib
```



Hình 12.4. Cấu trúc dạng lưu trữ các module, subpackage

12.3. PYTHON'S STANDARD LIBRARY

Thư viện chuẩn của Python rất lớn, cung cấp rất nhiều các phương thức, chức năng... phục vụ cho việc viết code. Thư viện chứa built-in function, built-in Constant, built-in Type, built-in Exception...

Python thể hiện rất nhiều chức năng tích hợp thông qua thư viện tiêu chuẩn, có thể tìm được danh sách đầy đủ các mô-đun thư viện tiêu chuẩn tại địa chỉ sau: <https://docs.python.org/3/library/index.html>

Một số mô-đun quan trọng trong thư viện chuẩn Python là:

- ☐ **math** cho các tiện ích toán học
- ☐ **re** cho các biểu thức chính quy
- ☐ **json** làm việc với JSON
- ☐ **datetime** làm việc với ngày tháng
- ☐ **sqlite3** sử dụng SQLite
- ☐ **os** cho các tiện ích Hệ điều hành
- ☐ **random** để tạo số ngẫu nhiên
- ☐ **statistics** cho các tiện ích thống kê
- ☐ **requests** để thực hiện các yêu cầu mạng HTTP
- ☐ **http** để tạo máy chủ HTTP
- ☐ **urllib** để quản lý các URL



Câu hỏi thảo luận

1. Nêu khái niệm module trong python?
2. Phân biệt module và package trong python?
3. Chức năng của module?
4. Khái niệm ?

Bài tập vận dụng

Bài tập chương 12 trong TLHT Tin cơ sở

