**Notas de Python**

**Índice**.

1. [**Módulos**.](#Modulos)
   1. [**Más sobre módulos**.](#Mas_sobre_modulos)
      1. Ejecutando módulos como scripts.
      2. La ruta de búsqueda del módulo.
      3. “Compilado” archivos de Python.
   2. [**Módulos standard**.](#Modulos_Standard)
   3. [**La función** dir()](#La_funcion_dir)
   4. [**Paquetes**.](#Paquetes)
      1. Importando \* From Package.
      2. Referencia Intra-package.
      3. Paquetes en múltiples directorios.

**Módulos**

Si sale del intérprete de Python y vuelve a ingresar, las definiciones que usted realizo (funciones y variables) se pierden. Por lo tanto, si usted quiere escribir un programa algo más largo, es mejor utilizar un editor de texto para preparar la entrada para el intérprete y ejecutarlo con ese archivo como entrada. Esto se conoce como crear un script. A medida que su programa crece, puede dividirlo en varios archivos para facilitar el mantenimiento. También puede usar una función práctica que haya escrito en varios programas sin copiar su definición en cada programa.

Para respaldar esto, Python tiene una forma de poner definiciones en un archivo y usarlas en un script o en una instancia interactiva del intérprete.

Un módulo es un archivo que contiene definiciones y sentencias de Python. El nombre del archivo es el nombre del módulo con el sufijo .py adjunto. Dentro de un módulo, el nombre del módulo (como una cadena) está disponible como el valor de la variable global \_\_name\_\_. Por ejemplo use su editor de texto favorito para crear un archivo llamado fibo.py en el directorio actual con los siguientes contenidos.

# Fibonacci numbers module

def fib(n): # write Fibonacci series up to n

a, b = 0, 1

while a < n:

print(a, end=' ')

a, b = b, a+b

print()

def fib2(n): # return Fibonacci series up to n

result = []

a, b = 0, 1

while a < n:

result.append(a)

a, b = b, a+b

return result

Ahora ingrese al intérprete de Python e importe esté modulo con el siguiente comando:

>>> import fibo

Esto no ingresa los nombres de las funciones definidas en fibo directamente en la tabla del símbolo actual; solo ingresa el nombre del módulo fibo allí. Usando el nombre del módulo puede acceder a las funciones.

>>> fibo.fib(1000)

0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987

>>> fibo.fib2(100)

[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]

>>> fibo.\_\_name\_\_

'fibo'

Si tiene la intención de utilizar una función a menudo, puede asignarla a un nombre local:

>>> fib = fibo.fib

>>> fib(500)

0 1 1 2 3 5 8 13 21 34 55 89 144 233 377

**Más sobre módulos**

Un módulo puede tener sentencias ejecutables, así como definiciones de funciones. Estas sentencias están destinadas a inicializar el módulo. Se ejecutan solo la primera vez que se encuentra el nombre del módulo en una declaración de importación. [[1]](#Pie_de_pagina_1) (También se ejecutan si el archivo se ejecuta como un script).

Cada módulo tiene su propia tabla de símbolos privada, que se utiliza como la tabla de símbolos global para todas las funciones definidas en el módulo. Por lo tanto el autor de un módulo puede usar variables globales en el módulo sin preocuparse por los choques accidentales con las variables globales de un usuario. Por otro lado, si sabe lo que está haciendo, puede tocar las variables globales de un módulo con la misma notación utilizada para referirse a sus funciones, nombremodulo.nombreitem.

Los módulos pueden importar otros módulos. Es habitual, pero no obligatorio, colocar las declaraciones [import](https://docs.python.org/3/reference/simple_stmts.html#import) al comienzo de un módulo (o script, para el caso). Los nombres de los módulos importados se colocan en la tabla de símbolos globales del módulo de importación.

Existe una variante de la declaración [import](https://docs.python.org/3/reference/simple_stmts.html#import) que importa los nombres de un módulo directamente en la tabla de símbolos del módulo de importación. Por ejemplo:

>>> from fibo import fib, fib2

>>> fib(500)

0 1 1 2 3 5 8 13 21 34 55 89 144 233 377

Esto no importa el nombre del módulo desde el que se toman las importaciones en la tabla de símbolos local (por lo que en el ejemplo, fibo no está definido).

Incluso hay un variante para importar todos los nombres que define un módulo:

>>> from fibo import \*

>>> fib(500)

0 1 1 2 3 5 8 13 21 34 55 89 144 233 377

Esto importa todos los nombres, excepto los que comiencen con un guion bajo (\_). En la mayoría de los casos, los programadores de Python no utilizan esta función ya que introduce un conjunto desconocido de nombres en el intérprete, posiblemente ocultando algunas cosas que ya ha definido.

Tenga en cuenta que, en general, la práctica de importar \* desde un módulo o paquete está mal visto, ya que a menudo causa un código poco legible. Sin embargo, está bien usarlo para guardar la escritura en sesiones interactivas.

Si el nombre del módulo es seguido por un [as](https://docs.python.org/3/reference/compound_stmts.html#as), entonces el nombre siguiente al [as](https://docs.python.org/3/reference/compound_stmts.html#as) se enlazará directamente al módulo importado.

>>> import fibo as fib

>>> fib.fib(500)

0 1 1 2 3 5 8 13 21 34 55 89 144 233 377

Esto efectivamente está importando el módulo de la misma manera que import fibo lo hace, con la única diferencia de que está disponible como fib.

También se puede usar cuando se utiliza [from](https://docs.python.org/3/reference/simple_stmts.html#from) con efectos similares:

>>> from fibo import fib as fibonacci

>>> fibonacci(500)

0 1 1 2 3 5 8 13 21 34 55 89 144 233 377

**Nota:** Por razones de eficiencia, cada módulo solo se importa una vez por sesión de intérprete. Por lo tanto, si cambia sus módulos, debe reiniciar el interprete – o si es solo un módulo que desea probar de forma interactiva, use [importlib.reload()](https://docs.python.org/3/library/importlib.html#importlib.reload), ejemplo, import importlib; importlib.reload(nombredelmodulo).

**Ejecutando módulos como scripts**

Cuando ejecuta un módulo de Python con

python fibo.py <arguments>

El código en el módulo se ejecutará, como si lo importara, pero con el \_\_name\_\_ establecido en “\_\_main\_\_”. Eso significa que al agregar este código al final de su módulo

if \_\_name\_\_ == "\_\_main\_\_":

import sys

fib(int(sys.argv[1]))

Puede hacer que el archivo sea utilizable como un script, así como un módulo importable, porque el código que analiza la línea de comandos se ejecuta si el módulo se ejecuta como el archivo “main”

$ python fibo.py 50

0 1 1 2 3 5 8 13 21 34

Si el módulo es importado, el código no se ejecuta:

>>> import fibo

>>>

Esto se usa a menudo para proporcionar una interfaz de usuario conveniente para un módulo o para propósitos de prueba (la ejecución del módulo como un script ejecuta un conjunto de pruebas).

**La ruta de búsqueda del módulo**

Cuando un módulo llamado spam es importado, el intérprete primero busca un módulo incorporado con ese nombre. Si no se encuentra, luego busca un archivo llamado spam.py en una lista de directorios que ofrece la variable [sys.path](https://docs.python.org/3/library/sys.html#sys.path). [sys.path](https://docs.python.org/3/library/sys.html#sys.path) se inicializa desde estas ubicaciones.

* El directorio que contiene el script de entrada (o el directorio actual cuando no se especifica ningún archivo).
* [PYTHONPATH](https://docs.python.org/3/using/cmdline.html#envvar-PYTHONPATH) (una lista de nombres de directorio, con la misma sintaxis que la variable de Shell PATH).
* El predeterminado dependiente de la instalación.

**Nota:** En los sistemas de archivos que admiten enlaces simbólicos, el directorio que contiene el script de entrada se calcular después de seguir el enlace simbólico. En otras palabras, el directorio que contiene el enlace simbólico **no** se agrega a la ruta de búsqueda del módulo.

Después de la inicialización, los programas de Python pueden modificar sys.path. El directorio que contiene el script que se está ejecutando se ubica al comienzo de la ruta de búsqueda, antes de la ruta standard de la biblioteca. Esto significa que los scripts en este directorio se cargarán en lugar de los módulos del mismo nombre en el directorio de la biblioteca. Esto es un error a menos que el reemplazo esté destinado. Consulte la sección Standard Modules para más información.

**Archivos de Python “Compilados”**

Para acelerar la carga de módulos, Python almacena en caché la versión compilada de cada módulo en el directorio \_\_pycache\_\_ bajo el nombre module.version.pcy, donde la versión codifica el formato del archivo compilado; generalmente contiene el número de versión de Python. Por ejemplo, en la versión 3.3 de CPython la versión compilada de spam.py se almacenaría en caché como \_\_pyche\_\_/spam.cpython-33.pyc. Esta conversión de nomenclatura permite que los módulos compilados de diferentes lanzamientos y diferentes versiones de Python coexistan.

Python compara la fecha de modificación de la fuente con la versión compilada para ver si está desactualizada y necesita ser recompilada. Esto es un proceso completamente automático. Además, los módulos compilados son independientes de la plataforma, por lo que la misma biblioteca se puede compartir entre sistemas con diferentes arquitecturas.

Pyhton no comprueba el caché en dos circunstancias. Primero, siempre recompila y no almacena el resultado para el módulo que se carga directamente desde la línea de comando. En segundo lugar, no comprueba el caché si no hay un módulo fuente.

Algunos consejos para expertos:

* Usted puede usar los modificadores [-o](https://docs.python.org/3/using/cmdline.html#cmdoption-o) ó [-oo](https://docs.python.org/3/using/cmdline.html#cmdoption-oo) en el comando Python para reducir el tamaño de un módulo compilado. El modificador -o elimina declaraciones assert, el modificador -oo elimina declaraciones assert y cadenas de texto \_\_doc\_\_. Dado que algunos programas pueden depender de tenerlos disponibles, solo debe usar esta opción si sabe lo que está haciendo. Los módulos “optimizados” tienen una etiqueta opt- y suelen ser más pequeños. Las futuras versiones pueden cambiar los efectos de la optimización.
* Un programa no se ejecuta más rápido cuando se lee de un archivo .pyc que cuando se lee de un archivo .py; lo único que es más rápido acerca de los archivos .pyc es la velocidad con la que se cargan.
* El módulo [compileall](https://docs.python.org/3/library/compileall.html#module-compileall) puede crear archivos .pyc para todos los módulos en un directorio.
* Hay más detalles sobre este proceso, incluido un diagrama de flujo de las decisiones, en [**PEP 3147**](https://www.python.org/dev/peps/pep-3147/).

**Módulos Standard**

Python viene con una biblioteca de módulos standard, que se describe en un documento separado, la Python Library References (en adelante “Library References”). Algunos modulo están integrados en el intérprete; estos proporcionan acceso a operaciones que no forman parte del núcleo del lenguaje pero que, sin embargo, están integradas, ya sea por eficiencia o para proporcionar acceso a primitivas del sistema operativo como las llamadas al sistema. El conjunto de dichos módulos es una opción de configuración que también depende de la plataforma subyacente. Por ejemplo, el módulo [winreg](https://docs.python.org/3/library/winreg.html#module-winreg) solo se proporciona en sistemas Windows. Un módulo en particular merece algo de atención: [sys](https://docs.python.org/3/library/sys.html#module-sys), que está integrado en cada intérprete de Python. Las variables sys.ps1 y sis.ps2 definen las cadenas utilizadas como indicaciones primarias y secundarias:

>>> import sys

>>> sys.ps1

'>>> '

>>> sys.ps2

'... '

>>> sys.ps1 = 'C> '

C> print('Yuck!')

Yuck!

C>

Estas dos variables solo se definen si el intérprete esta en modo interactivo.

La variable sys.path es una lista de cadenas que determina la ruta de búsqueda de los módulos para el intérprete. Se inicializa en una ruta predeterminada tomada de la variable de entorno [PYTHONPATH](https://docs.python.org/3/using/cmdline.html#envvar-PYTHONPATH), o de un valor predeterminado incorporado si [PYTHONPATH](https://docs.python.org/3/using/cmdline.html#envvar-PYTHONPATH) no está configurado. Puedes modificarlo usando las operaciones de lista estándar:

>>> import sys

>>> sys.path.append('/ufs/guido/lib/python')

**La función** [dir()](https://docs.python.org/3/library/functions.html#dir)

La función incorporada [dir()](https://docs.python.org/3/library/functions.html#dir) se usa para averiguar que nombres define un módulo. Devuelve una lista ordenada de cadenas:

>>> import fibo, sys

>>> dir(fibo)

['\_\_name\_\_', 'fib', 'fib2']

>>> dir(sys)

['\_\_displayhook\_\_', '\_\_doc\_\_', '\_\_excepthook\_\_', '\_\_loader\_\_', '\_\_name\_\_',

'\_\_package\_\_', '\_\_stderr\_\_', '\_\_stdin\_\_', '\_\_stdout\_\_',

'\_clear\_type\_cache', '\_current\_frames', '\_debugmallocstats', '\_getframe',

'\_home', '\_mercurial', '\_xoptions', 'abiflags', 'api\_version', 'argv',

'base\_exec\_prefix', 'base\_prefix', 'builtin\_module\_names', 'byteorder',

'call\_tracing', 'callstats', 'copyright', 'displayhook',

'dont\_write\_bytecode', 'exc\_info', 'excepthook', 'exec\_prefix',

'executable', 'exit', 'flags', 'float\_info', 'float\_repr\_style',

'getcheckinterval', 'getdefaultencoding', 'getdlopenflags',

'getfilesystemencoding', 'getobjects', 'getprofile', 'getrecursionlimit',

'getrefcount', 'getsizeof', 'getswitchinterval', 'gettotalrefcount',

'gettrace', 'hash\_info', 'hexversion', 'implementation', 'int\_info',

'intern', 'maxsize', 'maxunicode', 'meta\_path', 'modules', 'path',

'path\_hooks', 'path\_importer\_cache', 'platform', 'prefix', 'ps1',

'setcheckinterval', 'setdlopenflags', 'setprofile', 'setrecursionlimit',

'setswitchinterval', 'settrace', 'stderr', 'stdin', 'stdout',

'thread\_info', 'version', 'version\_info', 'warnoptions']

La función [dir()](https://docs.python.org/3/library/functions.html#dir) sin argumentos, lista los nombres que ha definido actualmente:

>>> a = [1, 2, 3, 4, 5]

>>> import fibo

>>> fib = fibo.fib

>>> dir()

['\_\_builtins\_\_', '\_\_name\_\_', 'a', 'fib', 'fibo', 'sys']

Tenga en cuenta que lista todos los tipos de nombres: variables, módulos, funciones, etc.

[dir()](https://docs.python.org/3/library/functions.html#dir) no lista los nombres de funciones y variables incorporadas. Si desea una lista de estos, se definen en el módulo estándar [builtins](https://docs.python.org/3/library/builtins.html#module-builtins):

>>> import builtins

>>> 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', 'NameError', 'None', 'NotADirectoryError', 'NotImplemented',

'NotImplementedError', 'OSError', 'OverflowError',

'PendingDeprecationWarning', 'PermissionError', 'ProcessLookupError',

'ReferenceError', 'ResourceWarning', 'RuntimeError', 'RuntimeWarning',

'StopIteration', 'SyntaxError', 'SyntaxWarning', 'SystemError',

'SystemExit', 'TabError', 'TimeoutError', 'True', 'TypeError',

'UnboundLocalError', 'UnicodeDecodeError', 'UnicodeEncodeError',

'UnicodeError', 'UnicodeTranslateError', 'UnicodeWarning', 'UserWarning',

'ValueError', 'Warning', 'ZeroDivisionError', '\_', '\_\_build\_class\_\_',

'\_\_debug\_\_', '\_\_doc\_\_', '\_\_import\_\_', '\_\_name\_\_', '\_\_package\_\_', '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']

**Paquetes**

**Notas de pie de página.**

[1] De hecho, las definiciones de funciones también son ‘sentencias’ que son ‘ejecutadas’; la ejecución de una definición de función de nivel de módulo ingresa el nombre de la función en la tabla de símbolos globales del módulo.

***Fuente:***[*https://docs.python.org/3/tutorial/modules.html#modules*](https://docs.python.org/3/tutorial/modules.html#modules)

*Septiembre de 2018*