



El futuro digital
es de todos

MinTIC

Modulos

+ Rogerio Orlando Beltrán Castro



¿Qué son los módulos de Python?

El módulo en Python te permite organizar tu código Python y hacer que sea más fácil el mantenimiento más adelante.

Dentro de un módulo, puedes definir funciones, clases y variables.

Puedes enviar tu módulo a tus amigos y compartirlo con otros para que puedan usar

Cómo funciona la sentencia import

Algunos principiantes podrían pensar que la sentencia import funciona como la directiva include en lenguaje C o require de PHP, pero eso está muy lejos de ser verdad porque los otros lenguajes lo único que hace es simplemente capturar el contenido del archivo.

En Python, la situación es un poco diferente. Se realizan tres pasos la primera vez que un programa importa un archivo determinado.

- Encuentra el archivo del módulo.

- Compila el archivo en bytecode si es necesario.

- Construye los objetos del bytecode.

Escribes el archivo importado sin la extensión y sin su ruta de directorio completa.

Entonces Python lo compila en bytecode. Si el archivo fuente se modifica, el archivo de bytecode(.archivoPYC) regenera automáticamente.

Al desplegar tus programas en Python solo como archivos bytecode y evitas enviar la fuente, harás que Python cargue los archivos de código de bytes.

El último paso de la operación de importación es ejecutar el bytecode generado.

Búsqueda de un módulo por Python

Como dijimos anteriormente, simplemente escribe el nombre del módulo sin extensión o ruta completa y Python se encargará del resto.

Entonces, ¿en dónde Python hace la búsqueda de los módulos?

El directorio de inicio de tu programa.

El directorio de variables de entorno PATH.

Directorios de biblioteca estándar.

Cualquier archivo .

El sitio contiene paquetes de tus extensiones de terceros.

La combinación de estos elementos se convierte en la lista `sys.path`.

En realidad, `sys.path` es la ruta donde Python busca módulos.

Búsqueda de un módulo por Python

Puedes obtener la ruta de la siguiente forma:

```
import sys  
print(sys.path)
```

Puedes agregar nuevas rutas personalizadas para buscar módulos usando:

```
sys.path.append('A:/Modulos')
```


Búsqueda de un módulo por Python

Cuando ingresas `import mymodule`, Python cargará cualquiera de los siguientes::

- Archivo de código fuente llamado `mymodule.py`.
- Archivo decódigo de bytes llamado `pyc`.
- El archivo decódigo de bytes optimizado se llama `pyo`.
- Un directorio llamado `mymodule`, para importar paquetes.
- Módulo de extensión compilado como `mymodule.so` en Linux o `mymodule.dll` en Windows.
- Módulo compilado que está escrito en C.
- Archivo ZIP que se extrae automáticamente
- Los archivos Jython que son Python para Java.
- Componentes .NET, escrito en la versión IronPython de Python

Si tiene dos archivos diferentes como `mymodule.py` y `mymodule.dll`, Python cargará el primer archivo encontrado en la ruta de búsqueda durante la búsqueda de izquierda a derecha de `sys.path` . Pero, ¿y si Python encuentra ambos archivos en el mismo directorio?

Cualquiera de ellos puede cargarse primero, así que evita que esto suceda.

Creando un Módulo de Python

Para crear un módulo de Python, no se requiere nada especial, solo abre tu editor, escriba tu código y guarde tu archivo con extensión .py

Todas las funciones definidas del módulo se convierten en sus atributos.

Por ejemplo, si defines una función como la siguiente:

def myfunc:

```
    print("welcome")
```

Puedes llamar a la función myfunc al importar el módulo dentro de tu código.

Ten en cuenta que, al crear un módulo, el nombre del módulo se convierte en una variable (sin la extensión) dentro de tu programa Python, por lo que no puedes nombrar un módulo if.py. Siempre trata de evitar las palabras reservadas de Python.

Instalar módulos de Python desde la fuente

El módulo incluye un archivo especial llamado setup.py que maneja los detalles.

Puedes instalar los módulos de Python desde la fuente de esta manera:

```
$ python setup.py install
```

Este comando instalará el módulo dentro de la carpeta site-packages que podría ser así: C:\Python27\lib\site-packages

O así

```
C:\Users\LikeGeeks\AppData\Local\Programs\Python\Python36-32\lib\site-packages
```


Instalar módulos de Python usando pip

Puedes usar la utilidad pip para instalar módulos y paquetes desde el repositorio PyPI.

La mayoría de los desarrolladores prefieren instalar módulos utilizando este método. Lo mejor de pip es que maneja la comprobación de dependencias.

Puedes instalar módulos Python usando pip de la siguiente forma:

```
$ pip install numpy
```

Además, puedes actualizar tus módulos instalados usando pip también:

```
$ pip install --upgrade numpy
```

Este comando actualizará todas las dependencias de los módulos.

Importación de módulos

Después de escribir tus impresionantes módulos o instalarlos, ahora necesitas reutilizarlos en cualquier parte de tu código, o tal vez enviarlos a tus amigos para que puedan usarlos o compartirlos con otros en la web. ¿Cómo importas módulos en tu código?

Hay dos formas de lograrlo, la primera es usar la sentencia `import`, y la segunda es usar la `sentencia from`. Ambos encontrarán, compilarán y ejecutarán.

La principal diferencia es que la sentencia `import` trae todo el módulo, mientras que la instrucción `from` obtiene atributos específicos de los módulos.

```
import mymodule
```

```
from mymodule import myfunc
```

La sentencia import

Debido a que la sentencia import carga todos los atributos del módulo, si desea utilizar cualquier atributo, debes utilizar el nombre del módulo de la siguiente manera:

```
import mymodule
```

```
+ mymodule.myfunc()
```

La sentencia from

Como mencionamos anteriormente, la sentencia from carga solo atributos especificados. Utilizar la sentencia from permite que escribamos menos debido a que no tienes que especificar el nombre completo del módulo cuando lo llamas.

```
from mymodule import myfunc
```

myfunc() De hecho, en su interior, la sentencia from es una extensión de sentencia import, también se carga el archivo completo, pero se le proporcionarán todos los atributos directamente desde tu código.

Puedes escribir la sentencia de la siguiente manera:

```
from mymodule import *
```

El asterisco aquí significa importar todo el módulo.

El peligro de utilizar la sentencia From

Algunos desarrolladores de Python recomiendan usar siempre la sentencia `import` y evitar la `from`.

¿Pero esto es cierto? ¿La sentencia `from` no es segura?

Bueno, si utilizas el archivo `from` para importar variables, sobrescribirá cualquier variable que tenga el mismo nombre si existe sin previo aviso.

Este problema, por supuesto, no ocurre con la sentencia `import`. Debido a que escribe el nombre del módulo para poder obtener sus atributos, lo cual evita cualquier colisión.

El peligro de utilizar la sentencia From

Debes tener cuidado cuando utilizas la sentencia from. Considera la siguiente situación:

```
#module1
def myfunc:
    print("Hello from module 1")
#module2
def myfunc:
    print("Hello from module 2")
```

Ahora, importemos los módulos anteriores e intentemos usarlos

```
#MyCode
from module1 import myfunc
from module2 import myfunc
myfunc()
```

El resultado aquí será del módulo 2 ya que la función se sobrescribirá.

En realidad, esto no es algo común de encontrar, pero, de todos modos, hay una solución para eso.

Utilizar alias

Para evitar la colisión de nombres como en el ejemplo anterior, puedes utilizar un alias como el siguiente:

```
#MyCode
```

```
from module1 import myfunc as myfunc1
```

```
from module2 import myfunc as myfunc2
```

```
myfunc1()
```

```
myfunc2()
```

Es como cambiar el nombre de las importaciones. A veces los desarrolladores usan alias para acortar los nombres de los módulos que están importando cuando el nombre del módulo es muy grande.

Después de utilizar alias, ¿crees que utilizar las sentencia from es peligroso? Para mí, no lo es.

Alcance del módulo importado

Sabes que no podemos acceder a los atributos del módulo a menos que lo importemos.

Considera la siguiente situación

```
#module1
M = 10
def myfunc():
    global M
    M = 20
#module2
M = 50
import module1
module1.myfunc()
print(M,module1.M)
```

Entonces al ejecutar el modulo2, module1.myfunc () cambia la variable M en el módulo 1 no la M en modulo2.

El resultado será 50 20.

Eso significa que el alcance global de module1.myfunc es el archivo que lo incluye, no el archivo desde el que se ejecuta.

Las sentencias de import no promueven la visibilidad de los atributos, y el archivo importado no puede ver los atributos en el archivo de importación.

Recargando Módulos

Si intentas volver a importar el módulo después de importarlo en la parte superior de su código, Python buscará el módulo ya cargado de nuevo.

Para volver a cargar un módulo, puede utilizar la función `reload` que puede lograrlo.

¿Por qué tenemos que volver a cargar un módulo importado?

La respuesta es simple, esto permite que algunas partes de su programa sean modificadas sin reiniciar su programa. Puedes imaginar cualquier situación que requiera personalización dinámica.

Python solo puede volver a cargar los módulos escritos en Python. Por lo tanto, las extensiones compiladas (como las discutidas anteriormente) no se pueden volver a cargar.

DEBES importar el módulo primero antes de cargarlo. Puede volver a cargar módulos de la siguiente forma:

```
import module1
module1.myfunc()
from importlib import reload
reload(module1)
module1.myfunc()
```

Puede cambiar el código del módulo1 en tu editor de texto y se volverá a cargar.

Recargar módulos es una característica muy poderosa en Python si se usa con cuidado.

¿Cómo creamos un módulo propio?

Pues esto es fácil, primeramente debemos pensar cómo vamos a dividir nuestro proyecto. Si se tratase de una calculadora por ejemplo podríamos separar en un módulo las funciones de las operaciones matemáticas, por el otro la librería gráfica y el ejecutable principal por ejemplo `calc.py` que integra ambos.

Vamos a tomar de ejemplo la calculadora en python

¿Cómo creamos un módulo propio?

Pues bien vamos a exportar esas funciones a un modulo propio, el cual llamaremos calcfunc.py

Así que colocamos calc.py en una carpeta, pues el módulo debe alojarse preferentemente junto al archivo que lo importa o invoca digamos.

¿Cómo creamos un módulo propio?

Entonces copiamos el apartado 1 a nuestro nuevo módulo calcfunc.
Pero vamos a mejorar ese código asqueroso así:

```
def sumar(a, b): return (a + b)
```

```
def restar(a, b): return (a - b)
```

```
def multiplicar(a, b): return (a * b)
```

```
def dividir(a, b): return (a / b)
```

¿Cómo creamos un módulo propio?

Este es nuestro módulo, solo mejoramos el código para que quede en una sola línea y quitamos el print. Porque yo quiero que el solo se encargue de procesar los datos que le enviamos.

Ahora es el momento de modificar el código de la calculadora para importarlo y trabajar con sus funciones. Lo importe de esta manera para evitar tener que colocar calcfunc, si tu simplemente colocas un import calcfunc, te obligará a colocar antes del punto el nombre del mismo.

¿Cómo creamos un módulo propio?

Obviamente ese print que quitamos debemos agregarlo en la `calc.py` para que nos muestre “El resultado es” y también añadir los argumentos `a`, `b` que estamos dando como parámetros en las funciones del módulo, de lo contrario no funcionará. Y (tambores) así queda nuestro código:

¿Cómo creamos un módulo propio?

```
from calcfunc import * #Importamos todas las funciones explicitas de nuestro módulo
                        #para no tener que especificar el modulo.funcion y que nuestro codigo
                        #del apartado 3 quede casi igual.
#####-2-#####
while True: #Creamos un bucle
    try: #Intentamos obtener los datos de entrada
        a = int(input("Ingresa el primer numero: \n")) #Solicitamos el 1er numero al usuario
        b = int(input("Ingresa el segundo numero: \n")) #Solicitamos el 2do numero al usuario
        print (("Que calculo quieres realizar entre"), (a), ("y"), (b), ("?\n")) #Preguntamos el cal
        op = str(input("""" #Ofrecemos las opciones de cálculo las cuales van a llamar a las funciones
        1- Sumar
        2- Restar
        3- Multiplicar
        4- Dividir \n"""))
        #####-3-#####
        if op == '1':
            print("El resultado es {}". format(sumar(a, b)))
            #Aqui tenemos que cambiar las funciones, enviarle los argumentos para sus parametros
            #Y las metemos en un print, directamente para imprimir el resultado
            break
        elif op == '2':
            print("El resultado es {}". format(restar(a, b)))
            break
        elif op == '3':
            print("El resultado es {}". format(multiplicar(a, b)))
            break
        elif op == '4':
            print("El resultado es {}". format(dividir(a, b)))
            break
        else:
            print ("""Has ingresado un numero de opcion erroneo""") #En caso que el numero no
    except:
        print ("Error")
        op = '?'
```

IDLE Shell 3.9.4

File Edit Shell Debug Options Window Help

Python 3.9.4 (tags/v3.9.4:1f2e308, Apr 6 2021, 13:40:21) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>

===== RESTART: A:\Modulos\Calculadora.py =====

Ingresa el primer numero:
5
Ingresa el segundo numero:
6
Que calculo quieres realizar entre 5 y 6 ?

#Ofrecemos las opciones de cálculo las cuales van a llamar a las funciones
1- Sumar
2- Restar
3- Multiplicar
4- Dividir

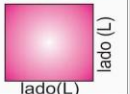

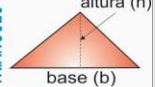
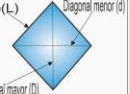
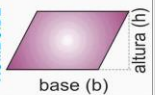

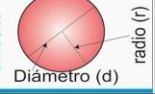

5
Has ingresado un numero de opcion erroneo
Ingresa el primer numero:
4
Ingresa el segundo numero:
6
Que calculo quieres realizar entre 4 y 6 ?

#Ofrecemos las opciones de cálculo las cuales van a llamar a las funciones
1- Sumar
2- Restar
3- Multiplicar
4- Dividir

4
El resultado es 0.6666666666666666
>>> |

Ejercicio

Crear un modulo con las formulas de las figuras geométricas

Áreas y Perímetros			MATEMAMIGAS	
CUADRADO	 lado(L)	ÁREA $A = L \times L$	PERÍMETRO $P = L + L + L + L$	
RECTÁNGULO	 base (b) altura (h)	ÁREA $A = b \times h$	PERÍMETRO $P = b + b + h + h$	
TRIÁNGULO	 base (b) altura (h)	ÁREA $A = \frac{b \times h}{2}$	PERÍMETRO $P = L + L + L$	
ROMBO	 lado(L) Diagonal mayor (D) Diagonal menor (d)	ÁREA $A = D \times d$	PERÍMETRO $P = L + L + L + L$	
ROMBOIDE	 base (b) altura (h)	ÁREA $A = b \times h$	PERÍMETRO $P = b + b + h + h$	
TRAPECIO	 base menor(b) lado(L) altura (h) base mayor (B)	ÁREA $A = \frac{h(B + b)}{2}$	PERÍMETRO $P = B + b + L + L$	
CIRCULO	 radio (r) Diámetro (d)	ÁREA $A = \pi \times r^2$	CIRCUNFERENCIA $C = \pi \times d$	
POLIGONO + 5	 lado(L) apotema (a)	ÁREA $A = \frac{p \times a}{2}$	PERÍMETRO $P = L \times \# \text{ lados}$	