Módulos en Python

Introducción

Un módulo o module en Python es un fichero .py que alberga un conjunto de funciones, variables o clases y que puede ser usado por otros módulos. Nos permiten reutilizar código y organizarlo mejor en namespaces. Por ejemplo, podemos definir un módulo mimodulo.py con dos funciones suma() y resta().

# mimodulo.py

def suma(a, b):

return a + b

def resta(a, b):

return a - b

Una vez definido, dicho módulo puede ser usado o importado en otro fichero, como mostramos a continuación. Usando import podemos importar todo el contenido.

# otromodulo.py

import mimodulo

print(mimodulo.suma(4, 3)) # 7

print(mimodulo.resta(10, 9)) # 1

También podemos importar únicamente los componentes que nos interesen como mostramos a continuación.

from mimodulo import suma, resta

print(suma(4, 3)) # 7

print(resta(10, 9)) # 1

Por último, podemos importar todo el módulo haciendo uso de \*, sin necesidad de usar mimodulo.\*.

from mimodulo import \*

print(suma(4, 3)) # 7

print(resta(10, 9)) # 1

Rutas y Uso de sys.path

Normalmente los módulos que importamos están en la misma carpeta, pero es posible acceder también a módulos ubicados en una subcarpeta. Imaginemos la siguiente estructura:

.

├── ejemplo.py

├── carpeta

│ └── modulo.py

Donde modulo.py contiene lo siguiente:

# modulo.py

def hola():

print("Hola")

Desde nuestro ejemplo.py, podemos importar el módulo modulo.py de la siguiente manera:

from carpeta.modulo import \*

print(hola())

# Hola

Es importante notar que Python busca los módulos en las rutas indicadas por el sys.path. Es decir, cuando se importa un módulo, lo intenta buscar en dichas carpetas. Puedes ver tu sys.path de la siguiente manera:

import sys

print(sys.path)

Como es obvio, verás que la carpeta de tu proyecta está incluida, pero ¿y si queremos importar un módulo en una ubicación distinta? Pues bien, podemos añadir al sys.path la ruta en la que queremos que Python busque.

import sys

sys.path.append(r'/ruta/de/tu/modulo')

Una vez realizado esto, los módulos contenidos en dicha carpeta podrán ser importados sin problema como hemos visto anteriormente.

Cambiando los Nombres con as

Por otro lado, es posible cambiar el nombre del módulo usando as. Imaginemos que tenemos un módulo moduloconnombrelargo.py.

# moduloconnombrelargo.py

hola = "hola"

En vez de usar el siguiente import, tal vez queramos asignar un nombre más corto al módulo.

import moduloconnombrelargo

print(moduloconnombrelargo.hola)

Podemos hacerlo de la siguiente manera con as:

import moduloconnombrelargo as m

print(m.hola)

## Listando dir

La función dir() nos permite ver los nombres (variables, funciones, clases, etc) existentes en nuestro namespace. Si probamos en un módulo vacío, podemos ver como tenemos varios nombres rodeados de \_\_. Se trata de nombres que Python crea por debajo.

print(dir())

# ['\_\_annotations\_\_', '\_\_builtins\_\_', '\_\_cached\_\_', '\_\_doc\_\_',

# '\_\_file\_\_', '\_\_loader\_\_', '\_\_name\_\_', '\_\_package\_\_', '\_\_spec\_\_']

Por ejemplo, \_\_file\_\_ es creado automáticamente y alberga el nombre del fichero .py.

print(\_\_file\_\_)

#/tu/ruta/tufichero.py

Imaginemos ahora que tenemos alguna variable y función definida en nuestro script. Como era de esperar, dir() ahora nos muestra también los nuevos nombres que hemos creado, y que por supuesto pueden ser usados.

mi\_variable = "Python"

def mi\_funcion():

pass

print(dir())

#['\_\_annotations\_\_', '\_\_builtins\_\_', '\_\_cached\_\_', '\_\_doc\_\_',

# '\_\_file\_\_', '\_\_loader\_\_', '\_\_name\_\_', '\_\_package\_\_', '\_\_spec\_\_',

# 'mi\_funcion', 'mi\_variable']

Por último, vamos a importar el contenido de un módulo externo. Podemos ver que en el namespace tenemos también los nombres resta y suma, que han sido tomados de mimodulo.

from mimodulo import \*

print(dir())

# ['\_\_annotations\_\_', '\_\_builtins\_\_', '\_\_cached\_\_',

# '\_\_doc\_\_', '\_\_file\_\_', '\_\_loader\_\_', '\_\_name\_\_',

# '\_\_package\_\_', '\_\_spec\_\_', 'resta', 'suma']

El uso de dir() también acepta parámetros de entrada, por lo que podemos por ejemplo pasar nuestro modulo y nos dará más información sobre lo que contiene.

import mimodulo

print(dir(mimodulo))

# ['\_\_builtins\_\_', '\_\_cached\_\_', '\_\_doc\_\_',

# '\_\_file\_\_','\_\_loader\_\_', '\_\_name\_\_',

# '\_\_package\_\_', '\_\_spec\_\_', 'resta', 'suma']

print(mimodulo.\_\_name\_\_)

# mimodulo

print(mimodulo.\_\_file\_\_)

# /tu/ruta/mimodulo.py

Excepción ImportError

Importar un módulo puede lanzar una excepción, cuando se intenta importar un módulo que no ha sido encontrado. Se trata de ModuleNotFoundError.

import moduloquenoexiste

# ModuleNotFoundError: No module named 'moduloquenoexiste'

Dicha excepción puede ser capturada para evitar la interrupción del programa.

try:

import moduloquenoexiste

except ModuleNotFoundError as e:

print("Hubo un error:", e)

Módulos y Función Main

Un problema muy recurrente es cuando creamos un módulo con una función como en el siguiente ejemplo, y añadimos algunas sentencias a ejecutar.

# modulo.py

def suma(a, b):

return a + b

c = suma(1, 2)

print("La suma es:", c)

Si en otro módulo importamos nuestro modulo.py, tal como está nuestro código el contenido se ejecutará, y esto puede no ser lo que queramos.

# otromodulo.py

import modulo

# Salida: La suma es: 3

Dependiendo de la situación, puede ser importante especificar que únicamente queremos que se ejecute el código si el módulo es el \_\_main\_\_. Con la siguiente modificación, si hacemos import modulo desde otro módulo, este fragmento ya no se ejecutará al ser el módulo main otro.

# modulo.py

def suma(a, b):

return a + b

if (\_\_name\_\_ == '\_\_main\_\_'):

c = suma(1, 2)

print("La suma es:", c)

Recargando Módulos

Es importante notar que los módulos solamente son cargados una vez. Es decir, no importa el número de veces que llamemos a import mimodulo, que sólo se importará una vez.

Imaginemos que tenemos el siguiente módulo que imprime el siguiente contenido cuando es importado.

# mimodulo.py

print("Importando mimodulo")

def suma(a, b):

return a + b

def resta(a, b):

return a - b

A pesar de que llamamos tres veces al import, sólo vemos una única vez el contenido del print.

import mimodulo

import mimodulo

import mimodulo

# Importando mimodulo

Si queremos que el módulo sea recargado, tenemos que ser explícitos, haciendo uso de reload.

import mimodulo

import importlib

importlib.reload(mimodulo)

importlib.reload(mimodulo)