Status	Finished
Started	Friday, 22 November 2024, 11:27 AM
Completed	Friday, 22 November 2024, 11:27 AM
Duration	24 secs
Marks	3.00/3.00
Grade	**10.00** out of 10.00 (**100**%)

Information

Dónde escribir una función

En Python una función se puede escribir junto con el código que la usa, en el mismo archivo:

archivo **my_program.py**

```
def my_func(a: int, b: float) -> bool:
    return a > b

num1 = int(input("Entra un número entero: "))
num2 = int(input("Entra un número entero: "))
print(my_func(num1, num2))
```

Pero un código puede llamar a una función escrita en otro archivo. Para ello, debe incluir previamente una cláusula de importación:

archivo **my_program.py**

```
import functions
```

```
num1 = int(input("Entra un número entero: "))
num2 = int(input("Entra un número entero: "))
print(functions.my_func(num1, num2))
```

archivo **functions.py**

```
def my_func(a: int, b: float) -> bool:
    return a > b
```

Esta segunda aproximación independiza a la función del código que la usa, lo que permite que sea usada desde piezas de código escritas en diferentes archivos.

Information

Módulos

La programación modular es una técnica de diseño de software que enfatiza la separación de la funcionalidad de un programa en módulos independientes e intercambiables, de manera que cada uno contiene todo lo necesario para ejecutar sólo un aspecto de la funcionalidad deseada. Por ejemplo, en un juego, podríamos tener, entre otros, un módulo para controlar la lógica del juego, otro para interactuar con el usuario y un tercero para mostrar el escenario de juego en la pantalla.

La programación modular permite gestionar la complejidad, dividiendo una funcionalidad compleja en otras más simples, y permite reusar el código, ya que los módulos deben ser independientes e intercambiables, es decir, un módulo desarrollado inicialmente para un programa podría usarse en otro y un programa que usa un módulo podría cambiarlo por otro mejor.

En Python, cada fichero con extensión **.py** alberga un **módulo**, que se llama como el archivo que lo contiene (sin la extensión). Para usar, desde un módulo distinto, las funcionalidades ofrecidas por un módulo, solo hay que poner su nombre en una cláusula de **importación**:

módulo **my_program.py**

```
import functions
```

```
num1 = int(input("Entra un número entero: "))
num2 = int(input("Entra un número entero: "))
print(functions.my_func(num1, num2))
print(functions.pi)
```

módulo **functions.py**

```
pi = 3.14

def my_func(a: int, b: float) -> bool:
    return a > b
```

El módulo functions del ejemplo anterior ofrece dos elementos que se pueden utilizar: la variable pi y la función my_func. Ambos, se pueden utilizar, una vez importado el módulo, escribiendo su nombre precedido del nombre del módulo separado por un punto.

```
print(functions.my_func(num1, num2))
print(functions.pi)
```

Un módulo proporciona un ***namespace***, un espacio de nombres en el que podemos declarar elementos sin entrar en conflicto con elementos de otros módulos que puedan tener el mismo nombre, ya que se diferenciarían al prefijarlos con el nombre del módulo (por ejemplo, podríamos usar en el mismo programa dos módulos que declarasen una función llamada *my_func* sin que se confundan).

En el caso de que un módulo ofrezca varios elementos pero sólo necesitemos usar algunos, los elementos que queramos se pueden importar individualmente, como muestra el siguiente ejemplo:

módulo **my_program.py**

```
from functions import my_func

num1 = int(input("Entra un número entero: "))
num2 = int(input("Entra un número entero: "))
print(my_func(num1, num2))
```

Los elementos importados directamente no pueden ser prefijados con el nombre del módulo a la hora de usarlos. Para evitar conflictos si queremos usar elementos con el mismo nombre de distintos módulos, podemos, o importar los módulos, para poder prefijar los elementos que queremos usar, o usar **alias**:

```
from module1 import suma as suma1
from module2 import suma as suma2
```

También, se pueden usar alias al importar un módulo completo, generalmente, para usar un nombre más corto y manejable:

```
import functions as funcs
```

Question 1

Complete

Mark 1.00 out of 1.00

Suponiendo que existe un módulo, llamado "module_1" que ofrece, entre otros muchos elementos, una función llamada "function_1", ¿cuál de las siguientes sentencias importa, específicamente, dicha función?

Select one:

- from module_1 import function_1
- import function_1 from module_1
- import module_1. function_1

Information

El módulo __main__

Un módulo puede incluir secuencias de instrucciones que no forman parte de ninguna función:

módulo **functions.py**

```
def my_func(a: int, b: float) -> bool:
    return a > b

print("Esto no forma parte de una función")
```

Tales instrucciones "libres" se ejecutan cuando se carga el módulo (la primera vez que es objeto de una importación), a diferencia de las funciones, que sólo se ejecutan cuando se las llama:

módulo **my_program.py**

```
from functions import my_func

radi = int(input("Entra un número entero: "))
num2 = int(input("Entra un número entero: "))
print(my_func(num1, num2))
```

Resultado:

```
Esto no forma parte de una función
Entra un número entero: 1
Entra un número entero: 2
False
```

Muchas veces no queremos que este código "libre" se ejecute cuando el módulo es importado, sino solo cuando es el módulo "principal" (inicial) del programa, para lo cual comprobamos si el módulo responde al nombre "__main__":

módulo **functions.py**

```
def my_func(a: int, b: float) -> bool:
    return a > b

if __name__ ==  "__main__":
    print("Esto no forma parte de una función")
```

El nombre __main__ (nótense los dos guiones bajos a cada lado) es asignado automáticamente al módulo que inicia la ejecución de un programa. De esta manera, el código controlado por esa sentencia `if` sólo se ejecuta si el módulo en que está es ejecutado directamente, pero no cuando es importado por otro que se empezó a ejecutar primero.

Question 2

Complete

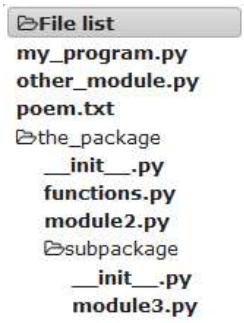
Mark 1.00 out of 1.00

¿Qué nombre se asigna automáticamente al primer módulo que se ejecuta en un programa Python?

Answer: __main__

Information

Paquetes



En programación modular, el concepto "paquete" (en inglés, *package*) suele usarse para referirse a una agrupación lógica de módulos: distintas funcionalidades se implementan en diferentes módulos, que pueden incluir funciones, variables, y otros elementos relacionados, y un conjunto de módulos con funcionalidades relacionadas se agrupan en un **paquete**.

En Python, al igual que un módulo se implementa en un archivo, un paquete, que es un conjunto de módulos, y por tanto implica un conjunto de archivos, se corresponde, físicamente, con una carpeta o directorio en el sistema de ficheros.

Para que un directorio sea un paquete debe incluir un "archivo de configuración" llamado **__init__.py**, que puede estar vacío, y que sirve para controlar el acceso a los módulos contenidos en el paquete desde módulos externos.

La figura muestra una estructura formada por dos módulos (*my_program* y *other_module*) que no forman parte de ningún paquete más un paquete de nombre *the_package*, conteniendo este, otros dos módulos (*functions* y *module2*) más un subpaquete de nombre *subpackage,* el cual contiene a su vez un único módulo (*module3*).

Al importar un módulo, hay que especificar el paquete al que pertenece, como se hace en el siguiente ejemplo:

módulo **my_program.py**

```
from the_package.functions import my_func

num1 = int(input("Entra un número entero: "))
num2 = int(input("Entra un número entero: "))
print(my_func(num1, num2))
```

Question 3

Complete

Mark 1.00 out of 1.00

¿Cuáles son ciertas?

Select one or more:

- [x] En Python, un paquete es un directorio que contiene archivos con la extensión .py
- [x] En Python, un paquete es una agrupación lógica de módulos
- [ ] En Python, un paquete es una pieza de código que se puede ejecutar de forma independiente