

Guía Teórica N°06

Manejo de Ficheros

La programación modular se refiere al proceso de dividir una tarea de programación grande y difícil de manejar en subtareas o módulos separados, más pequeños y más manejables. Luego, los módulos individuales se pueden improvisar como bloques de construcción para crear una aplicación más grande.

Hay varias ventajas de modularizar el código en una gran aplicación:

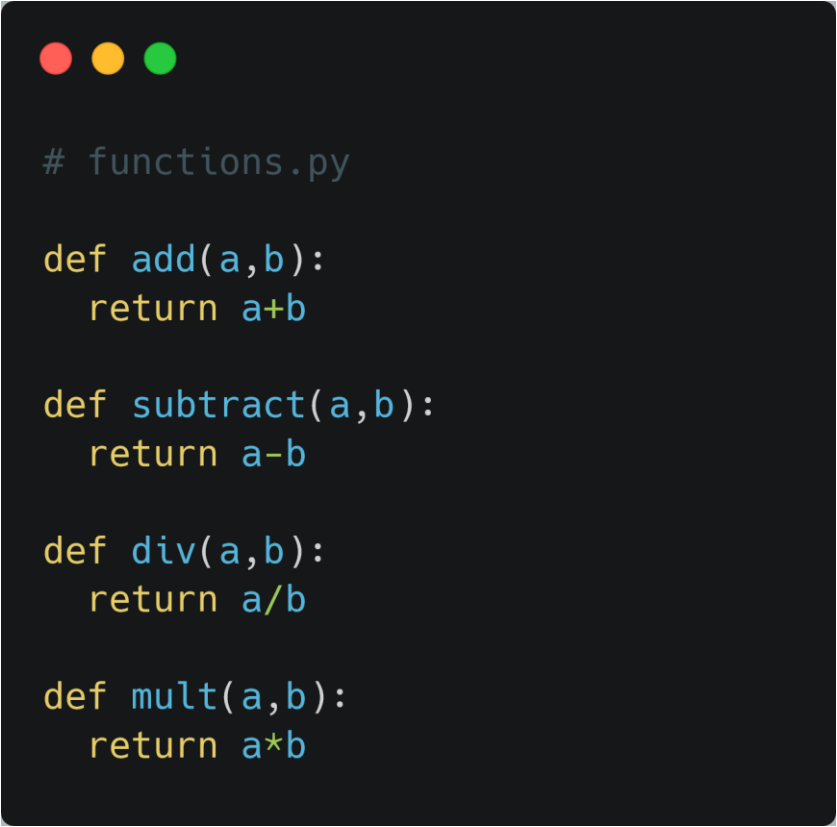
- **Simplicidad:** En lugar de centrarse en todo el problema en cuestión, un módulo normalmente se centra en una parte relativamente pequeña del problema. Si está trabajando en un solo módulo, tendrá un dominio de problemas más pequeño para entender. Esto hace que el desarrollo sea más fácil y menos propenso a errores.
- **Capacidad de mantenimiento:** Los módulos se diseñan normalmente para que impongan límites lógicos entre los diferentes dominios de problemas. Si los módulos están escritos de una manera que minimiza la interdependencia, existe una menor probabilidad de que las modificaciones a un solo módulo tengan un impacto en otras partes del programa. (Es posible que incluso pueda realizar cambios en un módulo sin tener ningún conocimiento de la aplicación fuera de ese módulo). Esto hace que sea más viable para un equipo de muchos programadores trabajar en colaboración en una aplicación grande.
- **Reutilización:** La funcionalidad definida en un solo módulo se puede reutilizar fácilmente (a través de una interfaz definida adecuadamente) por otras partes de la aplicación. Esto elimina la necesidad de duplicar el código.

- **Alcance:** Los módulos generalmente definen un espacio de nombres separado, lo que ayuda a evitar colisiones entre identificadores en diferentes áreas de un programa.

Las funciones, los módulos y los paquetes son construcciones en Python que promueven la modularización del código.

Aquí, la atención se centrará principalmente en los módulos que están escritos en Python. Lo bueno de los módulos escritos en Python es que son extremadamente sencillos de construir. Todo lo que necesita hacer es crear un archivo que contenga código Python legítimo y luego darle al archivo un nombre con extensión **.py**. No se necesita una sintaxis especial o vudú.

Por ejemplo, suponga que ha creado un archivo llamado `functions.py` que contiene lo siguiente:



```
# functions.py

def add(a,b):
    return a+b

def subtract(a,b):
    return a-b

def div(a,b):
    return a/b

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

Ahora vamos a importar este módulo en un archivo llamado `main.py` a través de la sentencia **import** y sus variantes.

Sentencia import:

Esta instrucción que nos ofrece Python tiene el objetivo de importar módulos dentro de otro archivo .py que pertenezca a nuestro proyecto en desarrollo. El uso de esta instrucción es muy sencillo, basta con indicar el nombre del módulo que queremos importar obviando la extensión **.py**.

import functions

De esta manera, estaríamos importando por completo el módulo en nuestro código y trabajaríamos con este como si fuese un objeto definido en Python. Por ejemplo, para hacer uso de la función **add** que se ubica dentro de **functions.py**, tendríamos que colocar **functions.add(10,5)** pero en dado caso de que solamente queramos importar una función en especial del módulo, entonces, tendremos que apoyarnos de la sentencia **from**.

from functions import add

Y de esta manera estaríamos importando únicamente la función **add** del módulo **functions.py** pero si queremos importar absolutamente todo lo que encuentre dentro de módulo, es suficiente indicando lo siguiente:

from functions import *

Por último, podemos importar una función (o clase) de un módulo y modificar el nombre de este dentro del código en el cual vayamos a utilizar sus funcionalidades con la sentencia **as**.

from functions import add as suma

Paquetes en Python:

Suponga que ha desarrollado una aplicación muy grande que incluye muchos módulos. A medida que aumenta el número de módulos, resulta difícil realizar un seguimiento de todos ellos si se depositan en un solo lugar. Esto es particularmente cierto si tienen nombres o funciones similares. Es posible que desee un medio para agruparlos y organizarlos.

Los paquetes permiten una estructuración jerárquica del espacio de nombres del módulo mediante la notación de puntos. De la misma manera que los módulos ayudan a evitar colisiones entre nombres de variables globales, los paquetes ayudan a evitar colisiones entre nombres de módulos.

Crear un paquete es bastante sencillo, ya que hace uso de la estructura de archivos jerárquica inherente del sistema operativo. Considere que se tiene la siguiente estructura de paquete.



Para importar el módulo **mod1.py** basta con colocar el código en que haremos uso del módulo:

```
import pkg.mod1
```

Todas las variaciones de importación que se explicaron anteriormente, aplican de igual forma para cuando nos encontramos trabajando con paquetes en Python. De igual manera, nosotros podemos tener subpaquetes definidos dentro de un paquete padre y el acceso a este es a través de la sintaxis de punto (.).

```
import pkg.pkgChildren.mod1
```

Archivo `__init__.py`:

Los archivos `__init__.py` son necesarios para que Python trate los directorios que contienen el archivo como paquetes. Esto evita que los directorios con un nombre común, como una cadena, oculten involuntariamente módulos válidos que aparecen más adelante en la ruta de búsqueda del módulo. En el caso más simple, `__init__.py` puede ser simplemente un archivo vacío, pero también puede ejecutar el código de inicialización del paquete.

¡Eso ha sido todo!

De esta manera, finalizamos todas las guías teóricas correspondientes al Curso de Programación en Python (Nivel: Intermedio) donde profundizamos de manera teórica en cada uno de los temas tratados en este curso. Ahora es turno de entrar al siguiente bloque del curso donde realizaremos un pequeño proyecto y así concretar esta ruta de aprendizaje que ha sido algo larga pero maravillosa para conocer mucho más sobre uno de los lenguajes de programación más importantes en el mundo de la computación.