9. Módulos y Paquetes

9.1. Módulos

Un **módulo** en Python es un **archivo** que contiene código Python (funciones, variables, clases, etc.) y que puede ser reutilizado en otros programas mediante la instrucción import.

Los módulos permiten **organizar mejor el código** dividiéndolo en partes reutilizables, evitando la duplicación y facilitando el mantenimiento.

Características generales de los módulos

- **Son archivos** .py
  Un módulo es simplemente un archivo de Python con extensión .py.
- **Se pueden importar en otros programas**
  Puedes importar un módulo en otro script utilizando import nombre_modulo.
- **Pueden contener cualquier código Python**
  Dentro de un módulo puedes definir:
  - Funciones
  - Variables
  - Clases
  - Bloques de ejecución
- **Existen módulos estándar y personalizados**
  - Python trae consigo una **biblioteca estándar** con muchos módulos predefinidos (math, random, os, etc.).
  - También puedes **crear tus propios módulos** para organizar mejor el código.

Main.py

```python
import Funciones

nombre = Funciones.presentarse()
Funciones.saludar(nombre)

```

Este es el archivo principal, este importa al módulo Funciones.

Funciones.py *

```python
def presentarse() -> str:
    nombre = input("Ingrese su nombre")
    return nombre

def saludar(nombre:str):
    print(f"Hola {nombre}, bienvenido!")

```

Y este es archivo que .py que contiene funciones que son usadas en el archivo Main.py.

Formas alternativas de importar módulos

```python
from Funciones import presentarse, saludar

nombre = presentarse()
saludar(nombre)
```

De esta forma podemos importar solo las funciones o bloques de código que necesitemos, la diferencia con import Funciones es que cuando llamamos a una función de ese módulo no tenemos que aclarar a que módulo pertenece.
Si quisieramos importar todo el módulo y no tener que especificar que en cada función a que módulo pertenece podemos hacer lo siguiente:

```python
from Funciones import *

nombre = presentarse()
saludar(nombre)
```

En este caso el * indica que queremos importar todo.

9.2. Paquetes en Python

Un **paquete** en Python es una carpeta que contiene **módulos** organizados en una estructura jerárquica. Sirven para **ordenar mejor el código** cuando el proyecto es grande y tiene múltiples módulos.

En términos simples, un paquete es **una carpeta que contiene archivos** .py y un archivo especial llamado __init__.py, que permite que Python reconozca la carpeta como un paquete.

Características generales de los paquetes

- **Son directorios que contienen módulos:** Un paquete es una **carpeta** con archivos .py dentro (los módulos).
- **Deben incluir un archivo __init__.py:** Aunque puede estar vacío, su presencia indica que la carpeta es un paquete de Python.
- **Permiten estructurar mejor los programas:** En lugar de tener todos los módulos en una sola carpeta, se pueden agrupar por categorías.
- **Se pueden importar igual que los módulos:** Puedes importar módulos desde un paquete usando import paquete.modulo.

```python
import paquete.Funciones as fn
import paquete.Operaciones as op

nombre = fn.presentarse()
fn.saludar(nombre)
print(op.sumar(5, 5))
```

"as" es un alias para importar el módulo de un paquete, en este caso, si no le ponemos alias estamos obligados a usar paquete.Operaciones.sumar(5, 5).

Main.py

```python
import paquete.Funciones as fn
import paquete.Operaciones

nombre = fn.presentarse()
fn.saludar(nombre)
print(paquete.Operaciones.sumar(5, 5))
```

Como vimos anteriormente, también podemos importar funciones específicas del módulo de un paquete de la siguiente forma:

Main.py

```python
import paquete.Funciones as fn
from paquete.Operaciones import sumar

nombre = fn.presentarse()
fn.saludar(nombre)
print(sumar(5, 5))
```

9.2.1. Comunicación entre módulos

Importaciones Absolutas

Se refieren a la ruta completa desde la raíz del paquete. Se usan cuando queremos importar un módulo dentro del mismo paquete, pero especificando su nombre completo como venimos viendo.

Importaciones Relativas

Las **importaciones relativas** permiten importar módulos dentro del mismo paquete sin necesidad de escribir el nombre completo. Se usan . (un punto) para referirse al **mismo paquete** y .. (dos puntos) para referirse al **paquete padre**.

Funciones.py

```python
from .Operaciones import sumar

def presentarse() -> str:
    nombre = input("Ingrese su nombre")
    return nombre

def saludar(nombre:str):
    print(f"Hola {nombre}, bienvenido!")

def calcular_edad_futura(edad:int, anios:int):
    total = sumar(edad, anios)
    print(f"En {anios} tendras {total} años")
```

Main.py

```python
import paquete.Funciones as fn
from paquete.Operaciones import sumar

nombre = fn.presentarse()
fn.saludar(nombre)
print(sumar(5, 5))

fn.calcular_edad_futura(25, 10)

```

9.3. Biblioteca Estándar de Python

Python cuenta con una poderosa **biblioteca estándar** que incluye un gran número de módulos y funciones listas para usar, sin necesidad de instalarlas por separado. Estos módulos permiten realizar tareas comunes como manipulación de archivos, matemáticas avanzadas, trabajo con fechas, gestión de red y mucho más.
Para utilizar un módulo de la biblioteca estándar, basta con **importarlo** en tu código de las mismas formas que venimos viendo.

Algunos ejemplos:

math: Funciones matemáticas avanzadas
datetime: Manejo de fechas y tiempos
os: Interacción con el sistema operativo
random: Generación de números aleatorios
json: Manejo de datos en formato JSON

9.4. Módulos de terceros

Además de la **biblioteca estándar**, Python permite instalar **módulos de terceros** desarrollados por la comunidad para ampliar sus capacidades. Estos módulos pueden facilitar tareas como el análisis de datos, el desarrollo web, la inteligencia artificial y mucho más.
Son **paquetes adicionales** creados por la comunidad de Python y publicados en el **Python Package Index (PyPI)**, un repositorio donde se alojan miles de bibliotecas listas para usar.
Para instalar un módulo de terceros, se usa el **gestor de paquetes pip**, que viene incluido en Python desde la versión 3.4.
Los comandos de **pip** pueden ejecutarse tanto desde el **Símbolo del sistema/Terminal** como desde las **consolas de los IDEs**. Esto hace que instalar y gestionar paquetes sea **rápido y flexible**, según el entorno en el que estés trabajando.