📌 LECCIÓN 5: FUNCIONES Y MODULARIZACIÓN DEL CÓDIGO

🎯 **Objetivo:**
Aprender a **crear funciones** en Python, cómo **modularizar** el código para hacerlo más limpio y reutilizable, y entender la importancia de la modularidad en proyectos grandes.

🔷 1. ¿Qué es una función en Python?

Una **función** es un bloque de código con un nombre propio que puede recibir datos (parámetros) y, opcionalmente, devolver un valor con return. Sirve para realizar una tarea especifica y permite **reutilizar código u operaciones** para mantener el programa ordenado sin necesidad de escribirlo repetidamente.

Analogía cotidiana

Imagina una **receta** escrita en una tarjeta:

- El *nombre* de la receta es la función.
- Los *ingredientes* son los parámetros.
- El *plato terminado* que sacas del horno es el valor que devuelve (return). Reutilizas la misma receta cada vez que quieras el mismo resultado, sin reescribir pasos.

Sintaxis básica de una función:

```
def nombre_de_la_funcion(parametros):
    # Bloque de código
    return resultado
```

- **def**: Palabra clave para definir una función.
- **nombre_de_la_funcion**: El nombre que le das a la función.
- **parametros**: Información que pasa la función para que la use.
- **return**: La función devuelve un valor (esto es opcional).

📝 **Ejemplo de función simple:**

```
def saludar():
    print("¡Hola, bienvenido a Python!")

saludar()  # Llamada a la función
```

🔷 2. Funciones con parámetros

Puedes pasar **valores a las funciones** para que realicen tareas personalizadas con esos valores.

Sintaxis de función con parámetros:

```
def saludar(nombre):
    print("¡Hola, " + nombre + "!")
```

- **nombre** es el parámetro que pasa a la función para ser usado dentro de ella.

📝 Ejemplo con parámetro:

```
def saludar(nombre):
    print("¡Hola, " + nombre + "!")

saludar("Juan")  # Salida: ¡Hola, Juan!
```

🔷 3. Funciones con valor de retorno

A veces, una función necesita **devolver un valor** para usarlo en otro lugar del programa.

Sintaxis de función con retorno:

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

- La función **devuelve** el resultado de la operación.

📝 Ejemplo con retorno:

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

resultado = suma(5, 3)
print(resultado)  # Salida: 8
```

🔷 4. ¿Por qué usar funciones?

Las funciones ayudan a que el código sea **más limpio**, **reutilizable** y **fácil de mantener**. Sin ellas, tendrías que repetir el mismo bloque de código en varios lugares.

Ventajas de usar funciones:

- **Reutilización de código**: No tienes que escribir el mismo código muchas veces.
- **Claridad**: Organiza el código en bloques lógicos.
- **Facilidad de mantenimiento**: Si necesitas cambiar algo, lo haces en un solo lugar.

🔷 5. Parámetros y valor de retorno

Parámetros: Datos de entrada que la función necesita para trabajar.
Return: Resultado que la función entrega al terminar.

Analogía cotidiana

Una **cafetera automática**:

- Colocas agua y café (parámetros).
- Pulsas el botón (llamada a la función).
- Obtienes una taza de café (valor de retorno).

```
def cafe(tazas):
    return f"{tazas} taza(s) de café listas ☕"

print(cafe(2))
```

En Python, el prefijo f delante de una cadena indica un f-string (formatted string literal). Esto permite incrustar valores o expresiones dentro de la cadena usando llaves { }, y Python sustituye cada expresión por su resultado.

🔷 6. Modularización del código

La **modularización** consiste en dividir un programa en pequeñas piezas lógicas (funciones, archivos o módulos) que colaboran entre sí. Cada pieza cumple una tarea clara.

Analogía cotidiana

Tu **caja de herramientas**:

- Destornillador, martillo y llave inglesa están separados, cada uno sirve para una tarea.
- Cuando un tornillo se afloja, solo tomas el destornillador, no toda la caja de herramientas.

Ventajas

1. Reutilización: usas la misma función en muchos lugares.
2. Mantenimiento: cambias una pieza sin romper las demás.
3. Colaboración: en proyectos grandes, cada persona trabaja en un módulo diferente.

Ejemplo de modularización

Archivo matematicas.py (crear este archivo en la misma carpeta como un archivo aparte)

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

def restar(a, b):
  return a – b
```

Archivo principal (donde se esté realizando el trabajo)

```
import matematicas as m

print(m.sumar(4, 2))
print(m.restar(5, 1))
```

📌 **Ejercicio 1:**

1️⃣ Crea una función que:

- Solicite dos números al usuario y devuelva la **suma** de esos números.
- Llama a esta función e imprime el resultado.

📌 **Ejercicio 2:**

2️⃣ Crea una función que:

- Solicite un **número** al usuario.
- Devuelva **si el número es par o impar**.
- Llama a la función e imprime el resultado.

📌 **Evaluación de la Lección 5**

Antes de continuar, responde y ejecuta estos ejercicios:

1️⃣ **¿Qué es una función en Python y por qué es útil?**
2️⃣ **¿Qué crees que va a devolver este código? Justifica antes de probarlo:**

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

resultado = multiplicar(4, 2)
```

```
print(resultado)
```

3️⃣ **Realiza los Ejercicios 1 y 2, y comparte tu código para revisión.**

✅ **Cuando termines:**

- Guarda tu trabajo en **Google Drive** (Archivo > Guardar en Drive).
- Comparte el código aquí para revisión.

¡Excelente trabajo! Estamos un poco más cerca del final, ¡Felicitaciones! 🚀😊

🖊️ **Autor:** L. Nova
📅 **Fecha de creación:** 9 de julio de 2025