Subrutinas en Python (1)

Materia: Algoritmos y Programación

Prof. D.Sc. BARSEKH-ONJI Aboud

Facultad de Ingeniería
Universidad Anáhuac México

3 de noviembre de 2025

Agenda

1. Parte Teórica: Subrutinas

2. Paso de Argumentos

¿Qué es una Subrutina?

Definición

Una **subrutina** (que en Python llamaremos **función**) es un bloque de código **reutilizable** que realiza una tarea específica.

La Analogía de la Receta

Piensa en una subrutina como una **receta de cocina**:

- Tiene un **nombre** (ej. 'Receta para pastel de chocolate').
- Contiene una serie de **pasos** (instrucciones).
- Cada vez que quieres hacer el pastel, no re-escribes la receta, simplemente **sigues** la receta que ya existe.

¿Qué es una Subrutina?

El Principio DRY: 'Don't Repeat Yourself'

El objetivo principal de las subrutinas es **NO REPETIR CÓDIGO**. Si notas que estás escribiendo las mismas 5 líneas de código en tres lugares diferentes, ¡es momento de crear una subrutina!

Definir vs. Llamar

Para usar una subrutina, hay dos pasos clave que a menudo se confunden:

1. Definir la Subrutina

Esto es como **escribir la receta**. Le decimos a Python qué pasos debe seguir y cómo se llama.

- Se usa la palabra clave `def`.
- El código dentro debe estar **indentado**.

```
# Definicion de la subrutina
def saludar():
    print('¡Hola, bienvenido!')
    print('Esta es mi primera
    subrutina.')

```

2. Llamar a la Subrutina

Esto es como **preparar la receta**. Le decimos a Python: '¡Ejecuta ese bloque de código ahora!'

- Se hace escribiendo su nombre seguido de paréntesis `()`.

```
# Llamada (o invocacion)
saludar()
saludar()

```

Definir vs. Llamar

¡Importante!

Definir una subrutina no la ejecuta. Es solo el plano. ¡Debes **llamarla** para que haga algo!

Agenda

1. Parte Teórica: Subrutinas

2. Paso de Argumentos

¿Por qué necesitamos "Argumentos"?

El Problema

Nuestra subrutina `saludar()` es muy aburrida. Siempre hace exactamente lo mismo. ¿Qué pasa si queremos que salude a una persona **específica**?

```
# ?Como le decimos a saludar()
# que salude a 'Ana'?
saludar() # Imprime '¡Hola, bienvenido!'

# ?Y si ahora queremos que salude a 'Luis'?
saludar() # Sigue imprimiendo '¡Hola, bienvenido!'

```

¿Por qué necesitamos "Argumentos"?

La Solución: Parámetros y Argumentos

Necesitamos una forma de **pasar información** (como el nombre 'Ana') **hacia adentro** de la subrutina.

- En la analogía de la receta, estos son los **ingredientes**. La receta 'Hacer pastel' necesita 'Harina', 'Huevos', 'Chocolate'.

Parámetros vs. Argumentos

Es crucial entender esta terminología:

Parámetro (El Espacio de Estacionamiento)

Un **parámetro** es la variable que se declara **dentro de los paréntesis** al **definir** la subrutina. Es un "espacio reservado" que espera recibir un valor.

```
# 'nombre' es un PARAMETRO
def saludar_a(nombre):
    print(f'¡Hola, {nombre}! Bienvenido.')

```

Parámetros vs. Argumentos

Argumento (El Coche)

Un **argumento** es el **valor real** que se envía a la subrutina al **llamarla**. Es el "coche" que ocupa el espacio de estacionamiento.

```
# 'Ana' y 'Luis' son ARGUMENTOS
saludar_a('Ana')
saludar_a('Luis')

```

Ejemplo 1: Subrutina con Múltiples Argumentos

Objetivo

Crear una subrutina que presente a una persona, recibiendo su nombre y su edad.

Salida en Consola

```
--- Presentacion 1 ---
Les presento a Elena.
Tiene 30 anios.

--- Presentacion 2 ---
Les presento a Miguel.
Tiene 25 anios.
```

¡Importante!

El orden de los argumentos importa. El primer argumento ('Elena') va al primer parámetro (nombre), y el segundo ('30') va al segundo (edad).

Ejemplo 1: Subrutina con Múltiples Argumentos

Código en Python

```
# Definimos con dos parametros: 'nombre' y 'edad'
def presentar_persona(nombre, edad):
    print(f'Les presento a {nombre}.')
    print(f'Tiene {edad} anios.')

# Llamamos con dos argumentos:
print('--- Presentacion 1 ---')
presentar_persona('Elena', 30)

print('\n--- Presentacion 2 ---')
presentar_persona('Miguel', 25)

```

Ejemplo 2: Reutilizando Código Complejo

Objetivo

¡Recordemos nuestro ejemplo de las tablas de multiplicar! En lugar de escribir el código cada vez, encapsulémoslo en una subrutina.

Salida (Fragmento)

```
--- Tabla del 5 ---
5 x 1 = 5
...
5 x 10 = 50
--- Tabla del 7 ---
7 x 1 = 7
...
7 x 10 = 70
--- Tabla del 9 ---
... etc.
```

Ejemplo 2: Reutilizando Código Complejo

Código en Python

```python
# Definimos la subrutina que recibe el
# numero de la tabla que queremos imprimir
def imprimir_tabla(numero_tabla):
    print(f'\n--- Tabla del {numero_tabla} ---')

    # Usamos el parametro en nuestro ciclo
    for i in range(1, 11):
        resultado = numero_tabla * i
        print(f'{numero_tabla} x {i} = {resultado}')

# AHORA ES MUY FACIL USARLO:
# Llamamos a la subrutina varias veces
imprimir_tabla(5)
imprimir_tabla(7)
imprimir_tabla(9)

```

Resumen

Lo que aprendimos hoy

- **Subrutina (Función en Python):** Es un bloque de código con nombre, definido con `def`, que realiza una tarea.
- **DRY:** El objetivo es 'No Repetirse' y organizar mejor el código.
- **Definir vs. Llamar:** Escribir la 'receta' (`def`) no es lo mismo que 'prepararla' (llamarla con `()`).
- **Argumentos:** Son los 'ingredientes' que pasamos a la subrutina para que pueda trabajar con ellos.

Para la Próxima Clase...

Todas nuestras subrutinas de hoy *imprimen* cosas, pero no nos *devuelven* un resultado.
¿Qué pasa si quiero una subrutina que **calcule** un promedio y me entregue el número para que yo pueda guardarlo en una variable?
Eso es una **Función** y usaremos la palabra clave `return`.