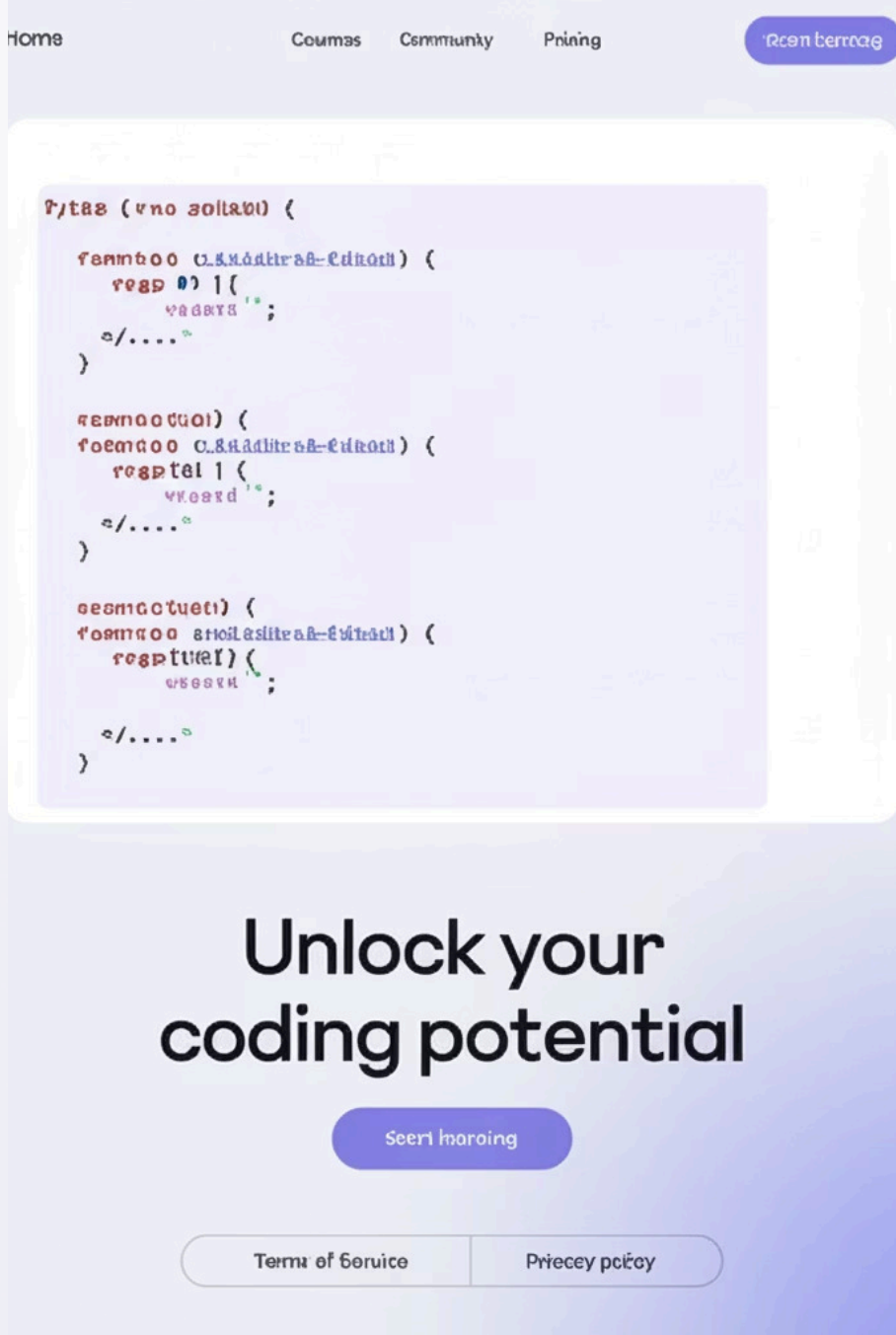
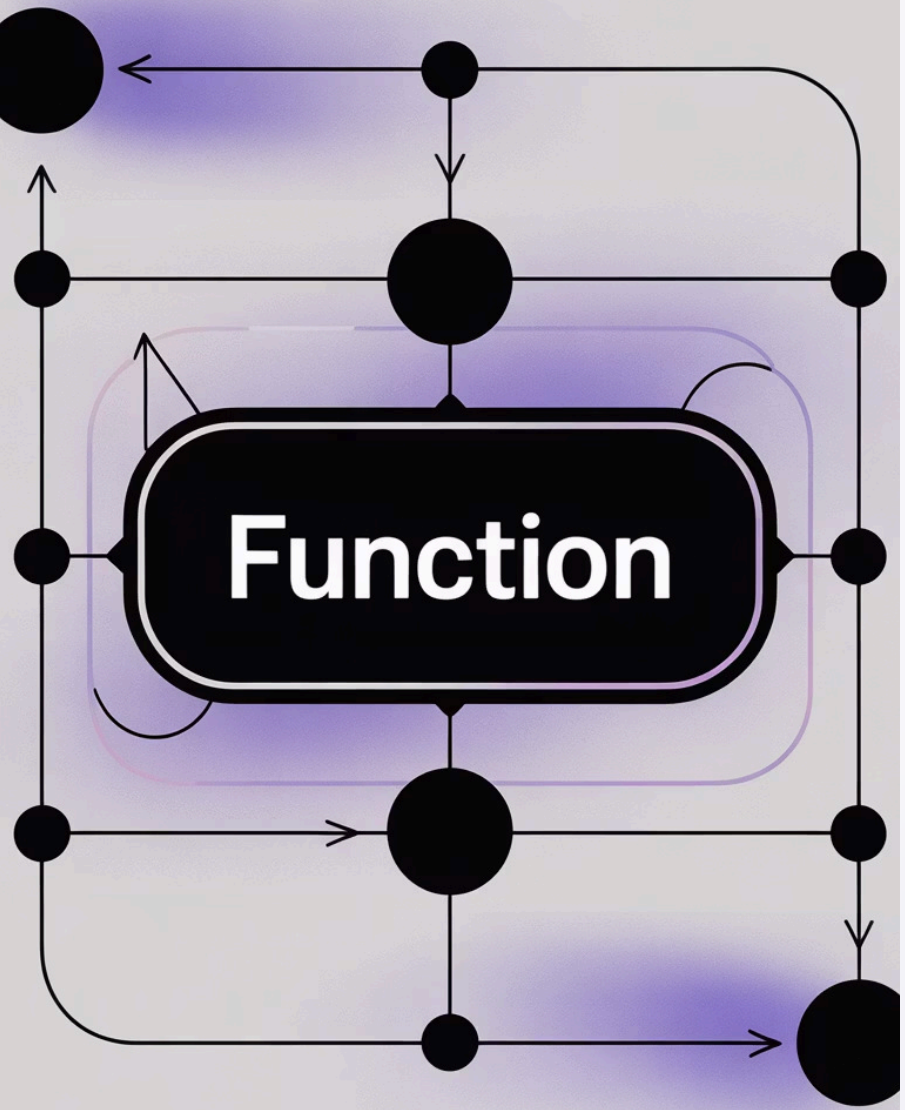
UTN - Programación I: Funciones en Python

Bienvenidos a la Tecnicatura Universitaria en Programación a Distancia de la **Universidad Tecnológica Nacional.** En esta presentación exploraremos las funciones en Python, una herramienta fundamental para resolver problemas complejos.





¿Qué es una función?

Bloque de código reutilizable

Encapsula lógica para cumplir una tarea específica.

Caja negra

Recibe datos, procesa internamente y devuelve resultados.

Solución modular

Transforma problemas complejos en soluciones manejables.

Elementos de una función

Entrada (argumentos)

Datos suministrados a la función: números, textos o estructuras complejas.

Proceso (cuerpo)

Instrucciones que procesan la información recibida.

Salida (retorno)

Resultado final obtenido después del procesamiento.

Ventajas de usar funciones

Reutilización de código

Escribe una vez, utiliza muchas veces. Evita repetir las mismas líneas en diferentes partes del programa. (Cálculos, impresos, procresos complejos, etc)

Mantenimiento simplificado

Modificar una función actualiza automáticamente todas sus instancias.

Un solo cambio afecta a todo el programa.

Abstracción

Oculta complejidad interna para facilitar el uso. Solo necesitas conocer qué hace, no cómo lo hace.

print("Hola Mundo ")

LiquidarComisiones(vendedor, periodo)

imprimir PDF(file)

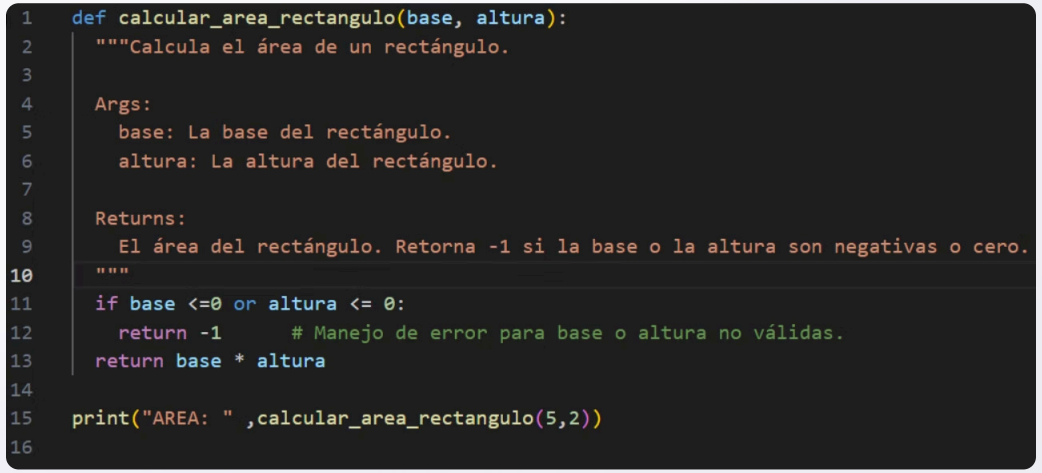
Legibilidad mejorada

Código más organizado y comprensible. Cada función representa un propósito claro y específico.

```
def saludar(mensaje):
    print(mensaje)

saludar("")
```

Ejemplo de Función

- En este ejemplo implementamos una función para calcular el área de un rectángulo de forma eficiente
- Incluimos comentarios explicativos como buena práctica de programación
- Implementamos validación de datos para manejar entradas inválidas (números negativos o cero)
- Demostramos cómo invocar la función con diferentes valores de parámetros
- La función retorna el resultado del cálculo que puede ser utilizado en otras operaciones



```python
def calcular_area_rectangulo(base, altura):
  """Calcula el área de un rectángulo.

  Args:
    base: La base del rectángulo.
    altura: La altura del rectángulo.

  Returns:
    El área del rectángulo. Retorna -1 si la base o la altura son negativas o cero.
  """
  if base <=0 or altura <= 0:
    return -1      # Manejo de error para base o altura no válidas.
  return base * altura

print("AREA: " ,calcular_area_rectangulo(5,2))

```

Tipos de funciones

Funciones integradas (built-in)

Ya incluidas en el lenguaje Python.

No necesitan definirse, solo llamarse.

Ejemplos: print(), len(), input()

Funciones definidas por el usuario

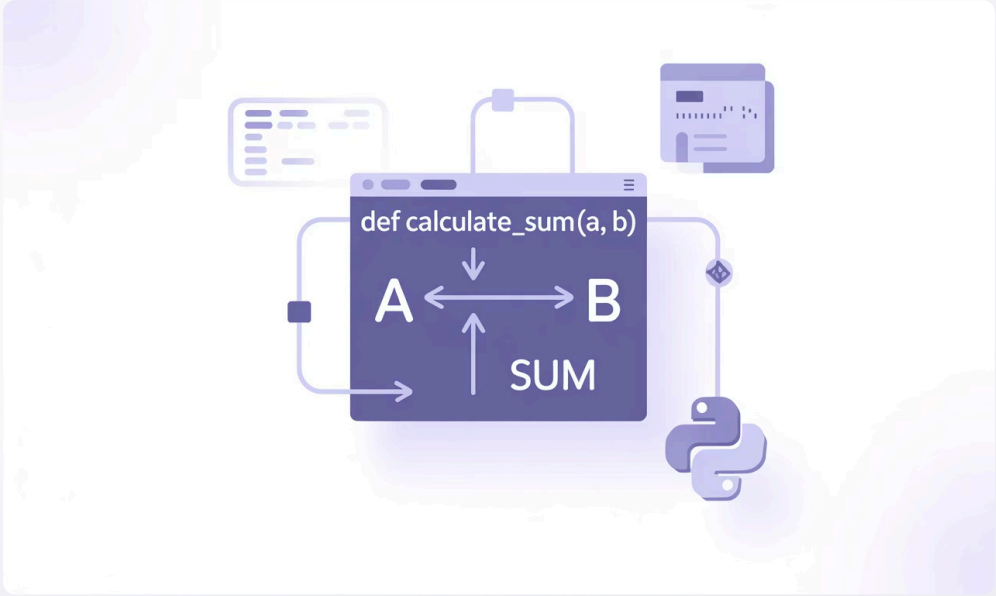
Creadas para resolver tareas específicas.

Permiten organizar mejor el código.

Se definen con la palabra clave "def".

Ejemplo de funcion en python

```
#Definición de la función

def sumar(a, b):

    resultado = a + b

    return resultado

# Uso de la función

x = 5

y = 3

suma = sumar(x, y)

print("La suma es:", suma)
```



Parámetros por valor

Copia del valor original

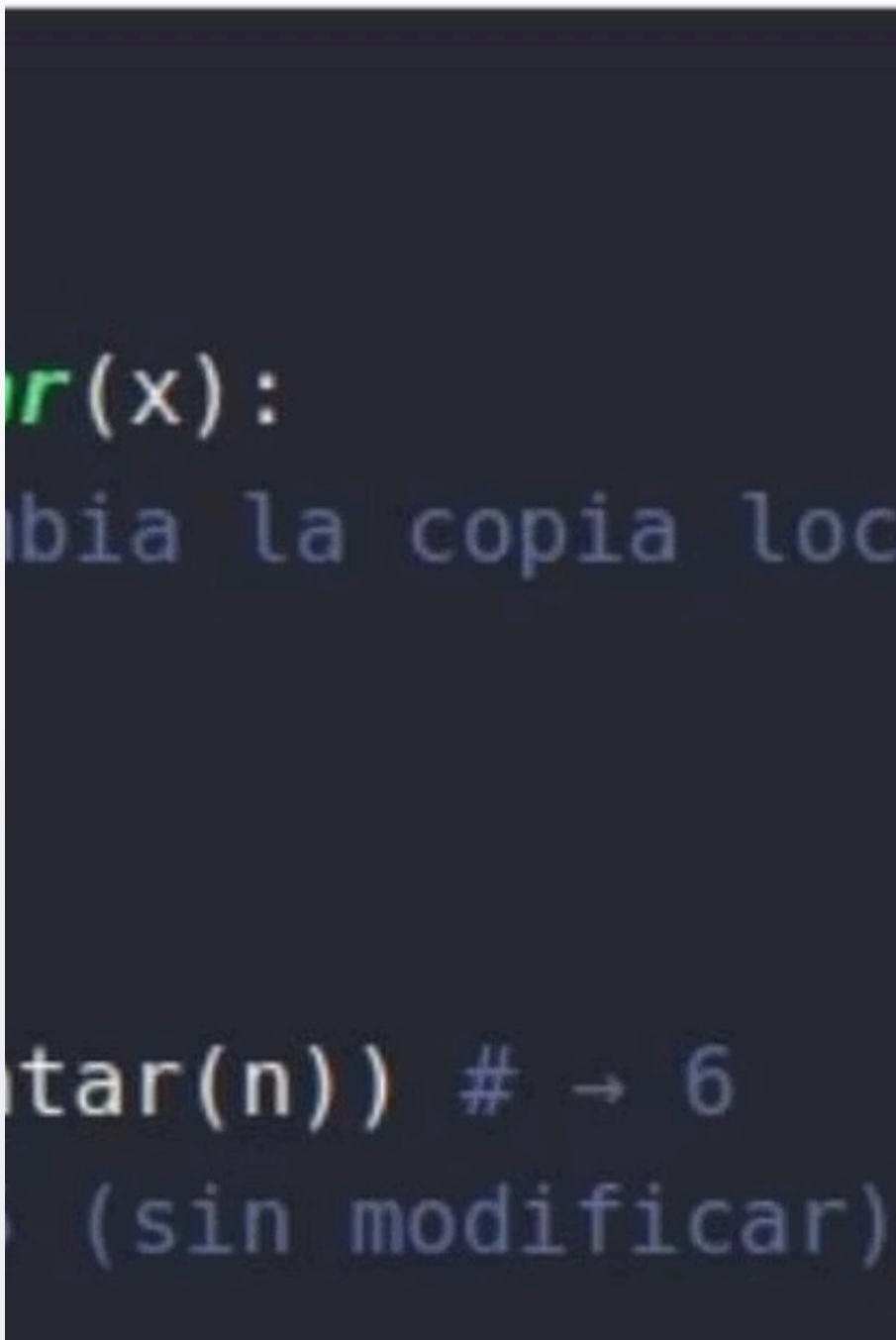
La función recibe **una copia**, no el valor original.

Cambios encapsulados

Modificaciones dentro de la función **no afectan** la variable externa.

Tipos inmutables

Comportamiento típico con int, float, str.



Parámetros por referencia

Referencia al objeto original

No se crea una copia, se pasa una referencia al objeto.

Mismo espacio en memoria

La función y el programa principal apuntan al mismo espacio.

Precaución

Útil pero puede provocar errores si no se controla bien.

Tipos mutables

Comportamiento típico con **list, dict** o set en Python.

Múltiples parámetros

Múltiples entradas

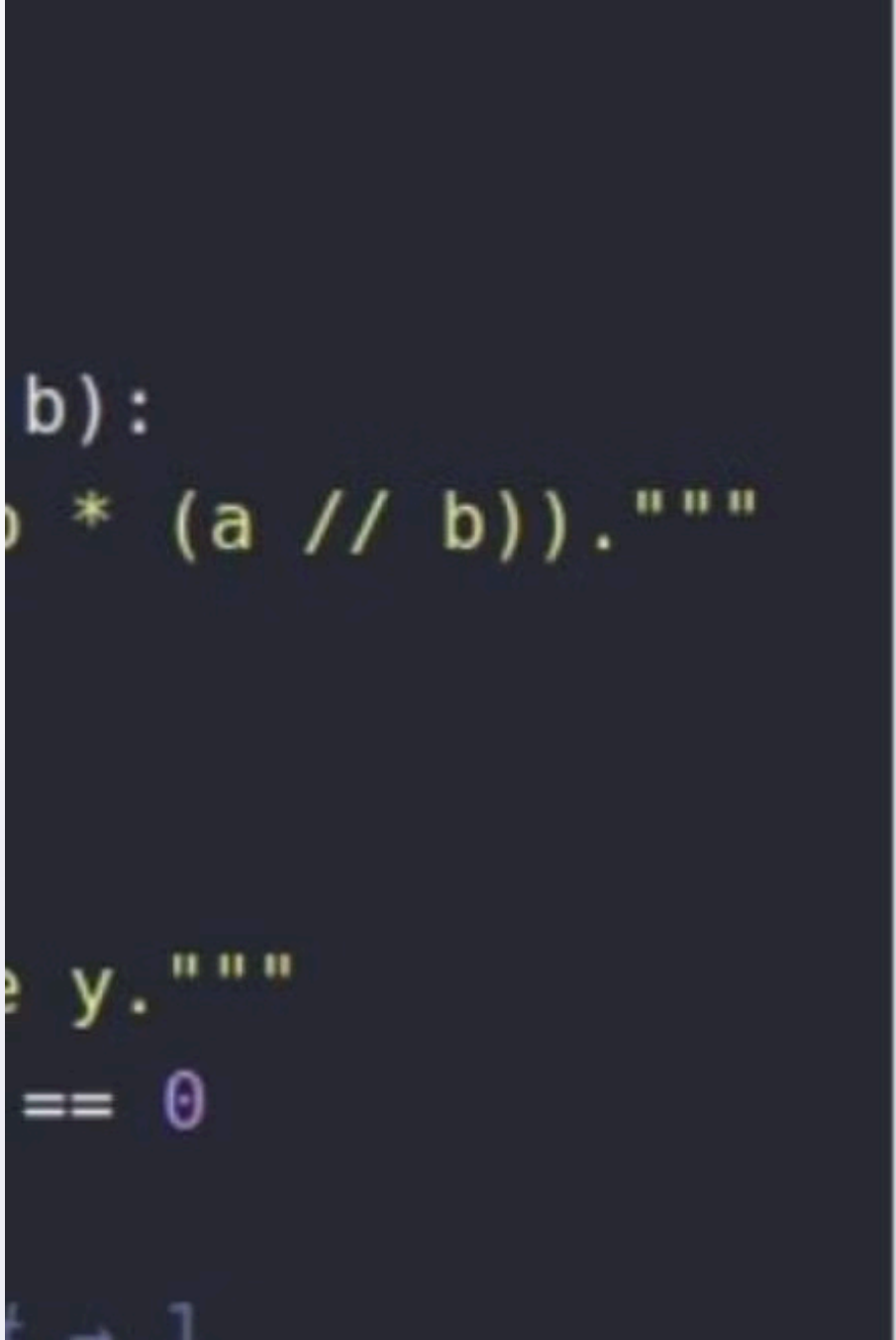
Las funciones pueden recibir más de un parámetro para trabajar con varios datos simultáneamente.

Operaciones complejas

Permite comparar, combinar o realizar cálculos entre valores distintos.

Ejemplos prácticos

Calcular restos o verificar si un número es múltiplo de otro.



Composición de funciones

ƒ(×)

Función interna

Se ejecuta primero

↑

Resultado intermedio

Pasa como argumento

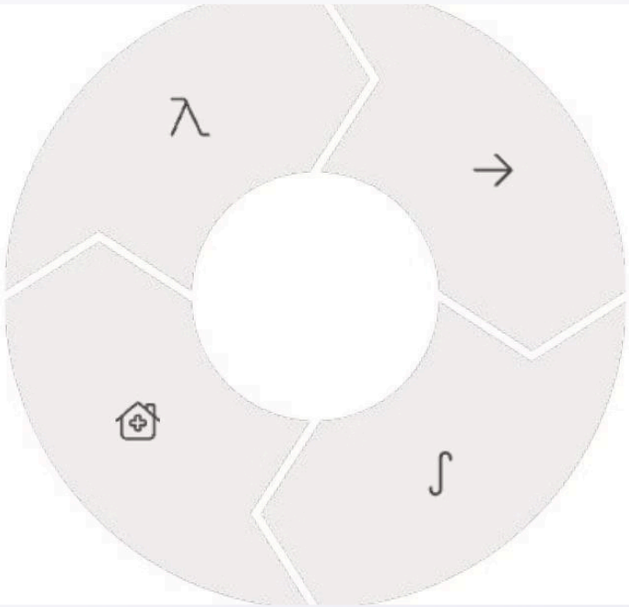
☆

Función externa

Procesa el resultado final

Función A

Primera función que se ejecuta



Resultado intermedio

El valor retornado por la primera función

Resultado final

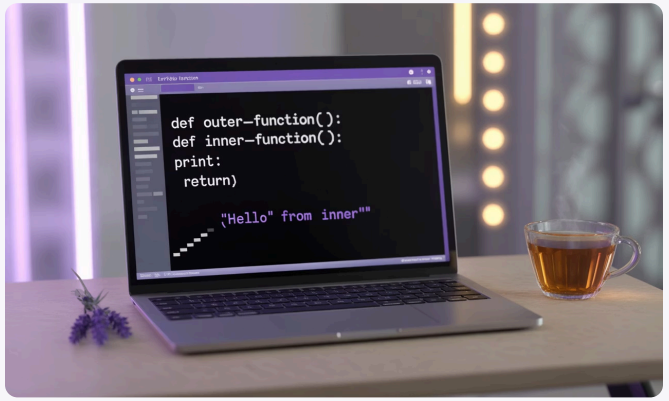
El valor final después de encadenar funciones

Función B

Recibe como entrada el resultado de la función A

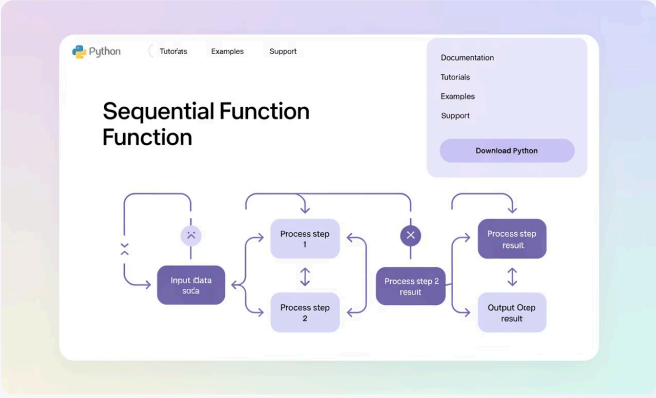
La composición ocurre cuando se usa el resultado de una función como argumento de otra, **encadenando** llamadas.

Ejemplo de composición



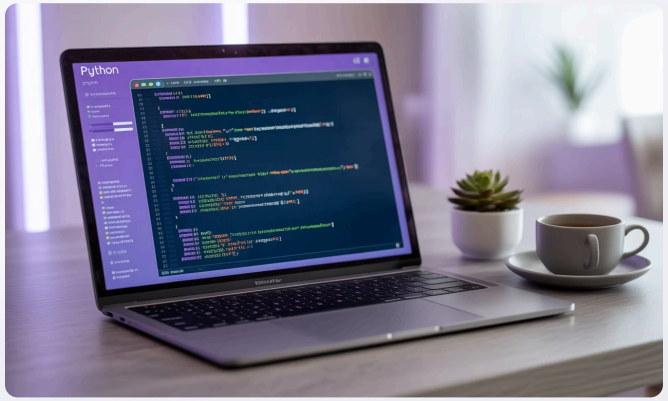
Código anidado

Python ejecuta primero la función más interna y usa su resultado como entrada para la siguiente.



Ejecución secuencial

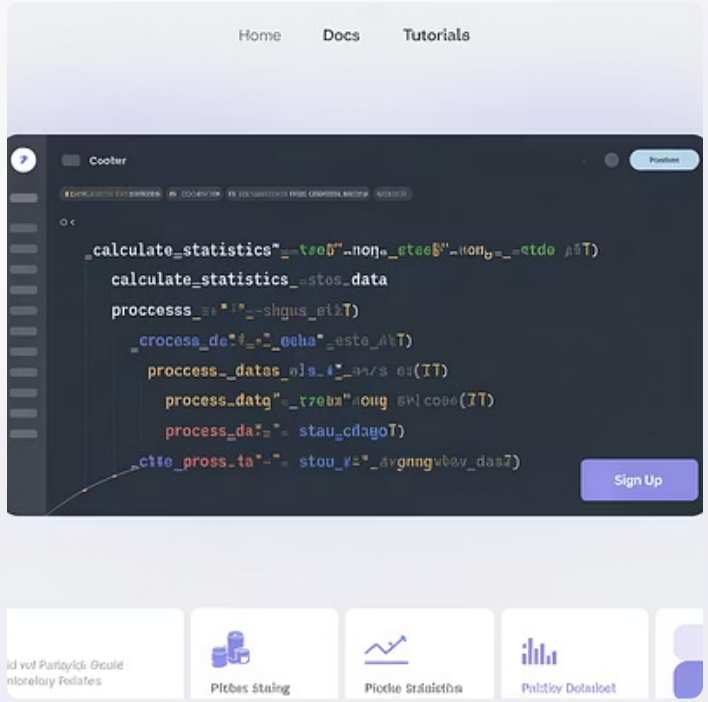
Permite escribir código más compacto y expresivo.



Alternativa con variables

Para mayor claridad, usar variables intermedias cuando hay muchas funciones anidadas.

Nombres descriptivos



Elegir nombres que indiquen con claridad lo que hace la función. Por ejemplo, calcular_promedio() es mejor que func1. Los buenos nombres hacen el código más legible y mantenible.

Una responsabilidad por función

Propósito único

Cada función debe tener una única tarea bien definida.

División de tareas

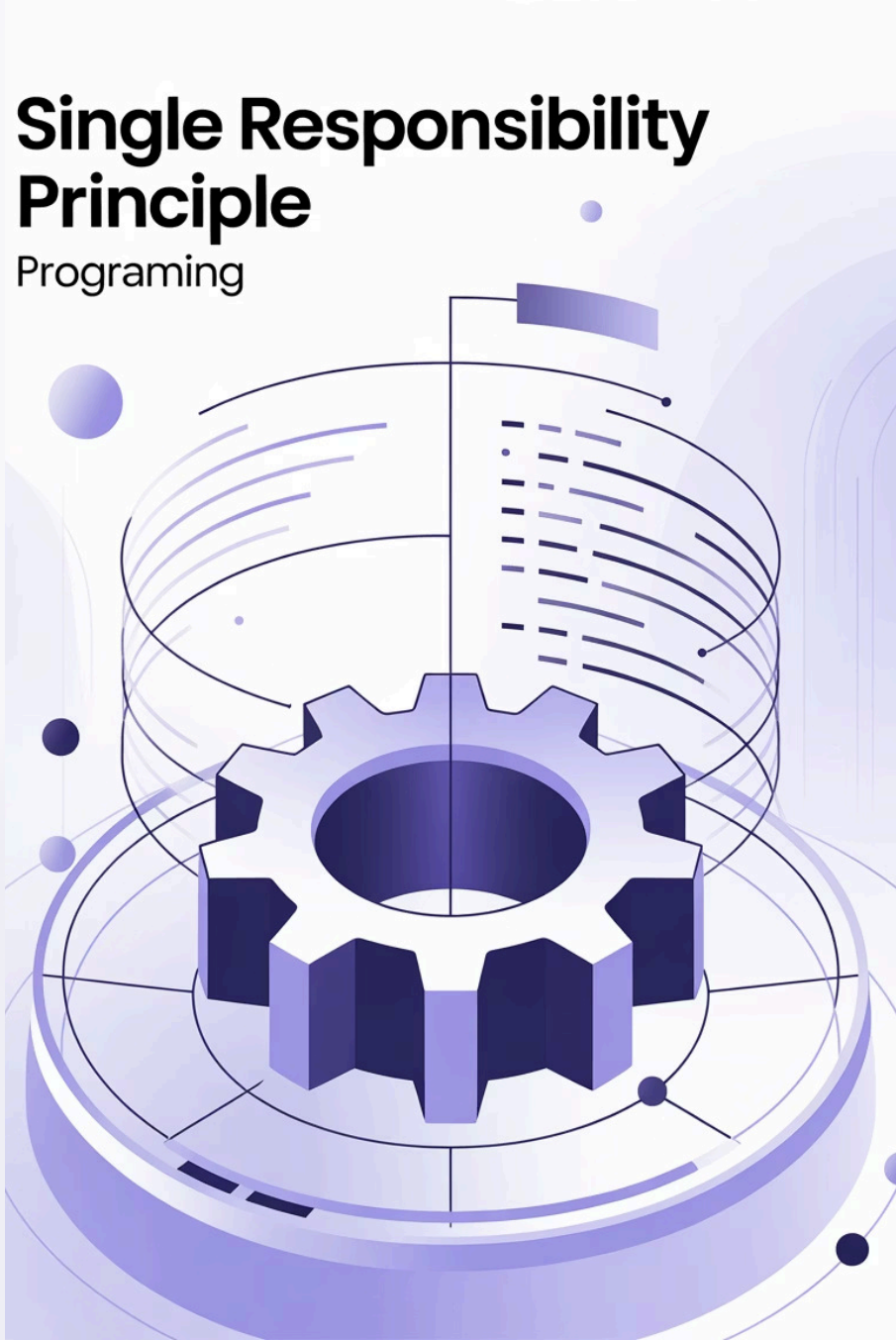
Si una función hace demasiadas cosas, conviene dividirla.

Código más claro

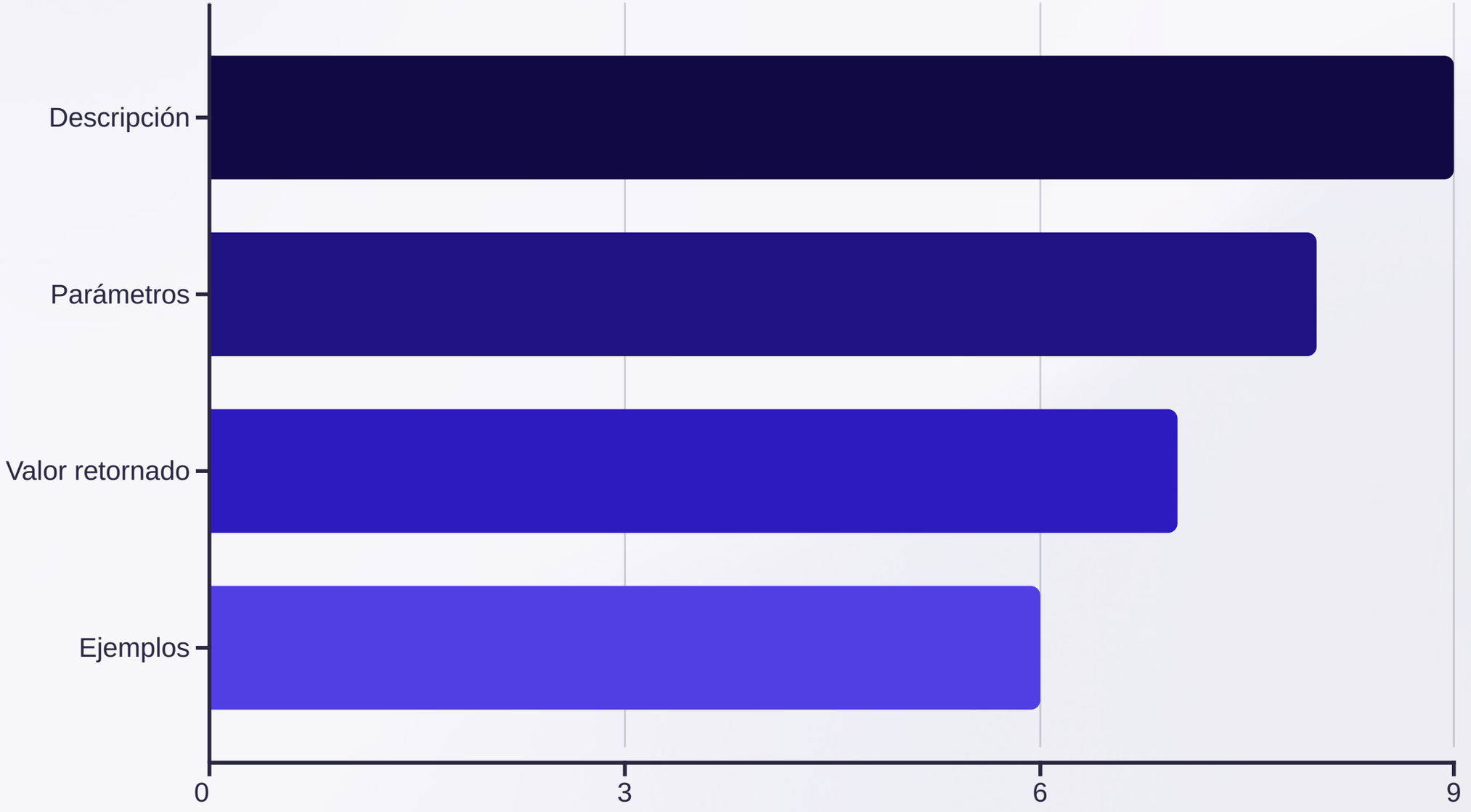
Facilita entender qué hace cada parte del programa.

Menos errores

Funciones pequeñas son más fáciles de probar y depurar.



Documentación con Docstrings



Agrega una breve descripción al principio de cada función usando comillas triples (""" """). Explica qué hace, qué parámetros recibe y qué devuelve.

Evitar código duplicado

1

Identificar patrones

Busca bloques de código que se repiten.

2

Extraer a función

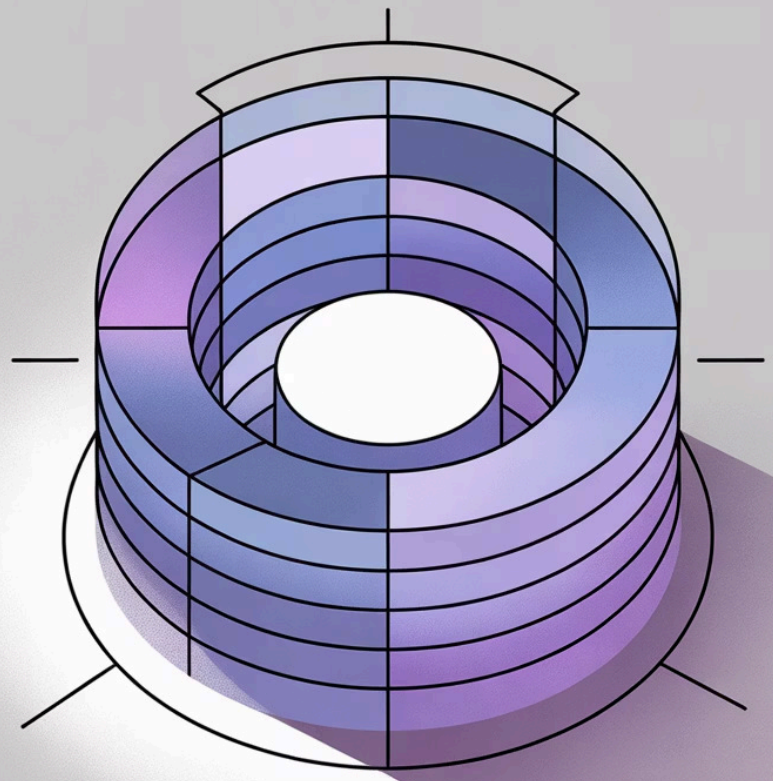
Convierte el código repetido en una función reutilizable.

3

Reemplazar

Sustituye cada repetición con una llamada a la función.

EDGE CASE TESTING



Probar con casos límite Testing

Valores extremos

Prueba con números muy grandes o muy pequeños.

Entradas vacías

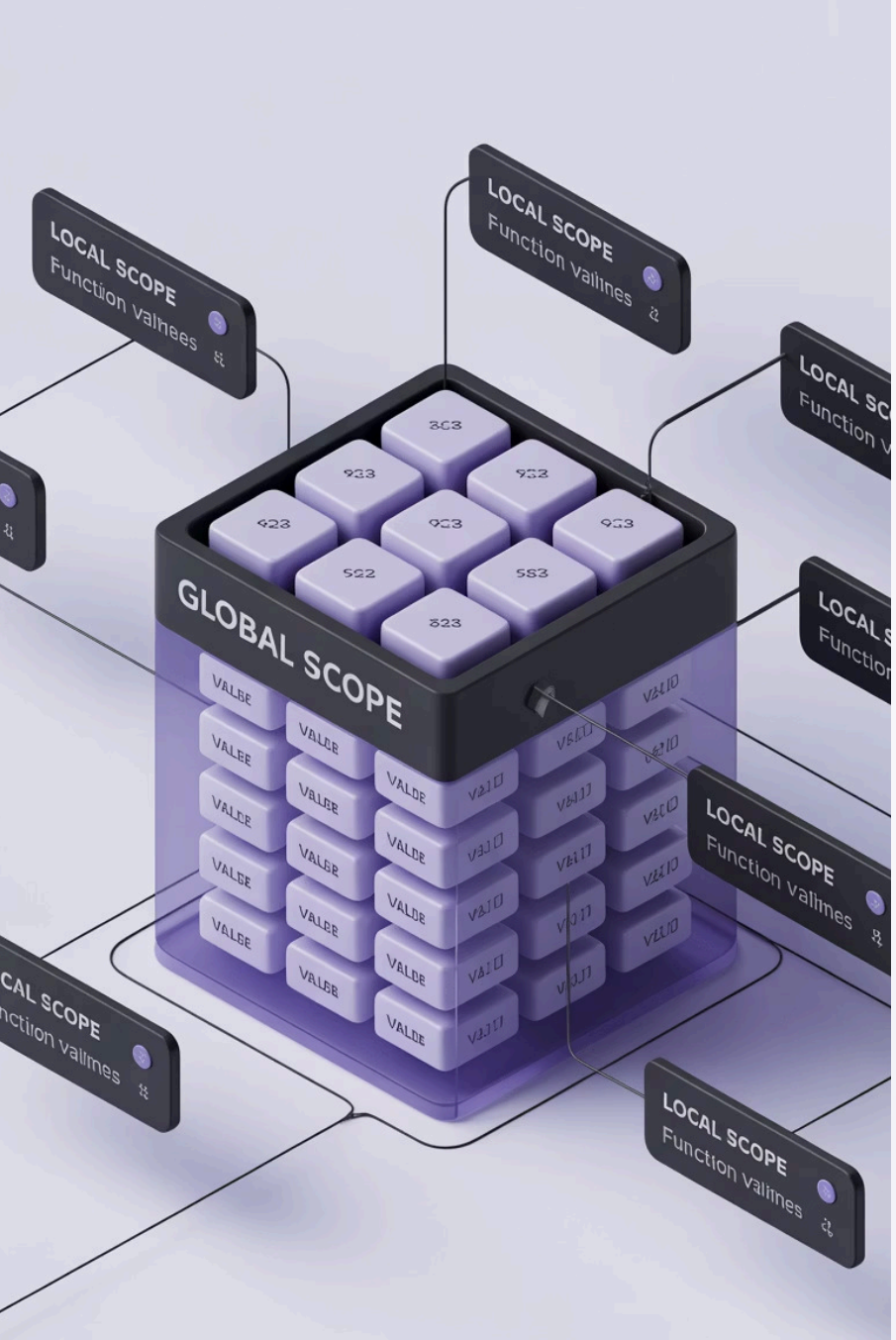
Verifica el comportamiento con listas vacías o strings vacíos.

Datos inesperados

Comprueba qué ocurre con tipos de datos incorrectos.

Código robusto

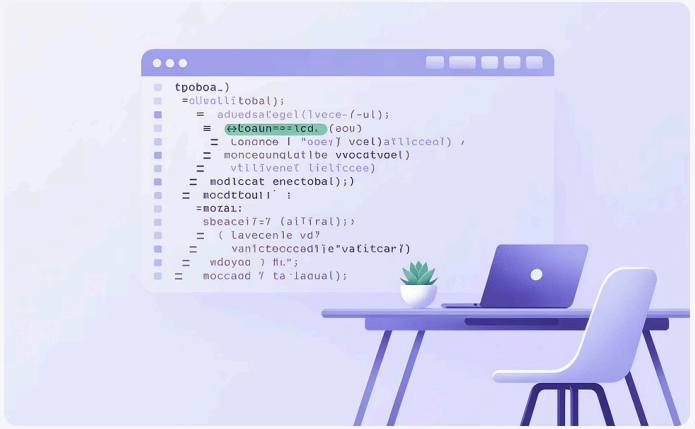
Asegura que la función maneja correctamente todos los casos.



Ámbitos de variables

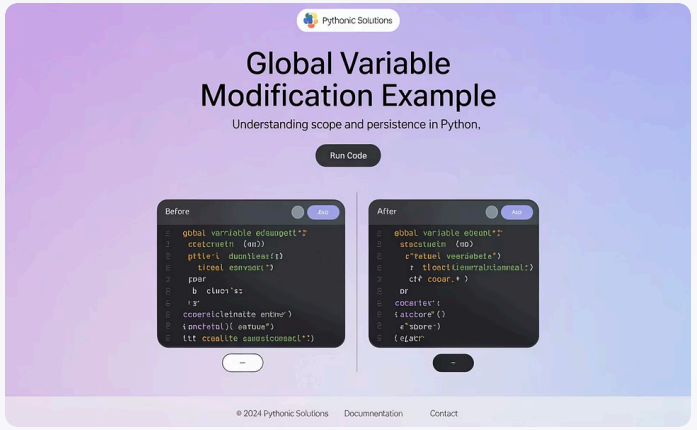
Tipo de ámbito	Descripción	Accesibilidad
Ámbito local	Variables declaradas dentro de una función solo existen dentro de esa función.	No pueden ser accedidas desde fuera.
Ámbito global	Variables creadas fuera de cualquier función son globales.	Accesibles desde todo el archivo.

Modificar variables globales



Palabra clave global

Permite modificar variables globales desde dentro de una función.



Ejemplo práctico

```
a = 10              # global

def cambiar():
    global a
    a = 20

cambiar()

print(a)            # 20
```

Usar con precaución

El uso excesivo de variables globales puede hacer el código difícil de mantener.

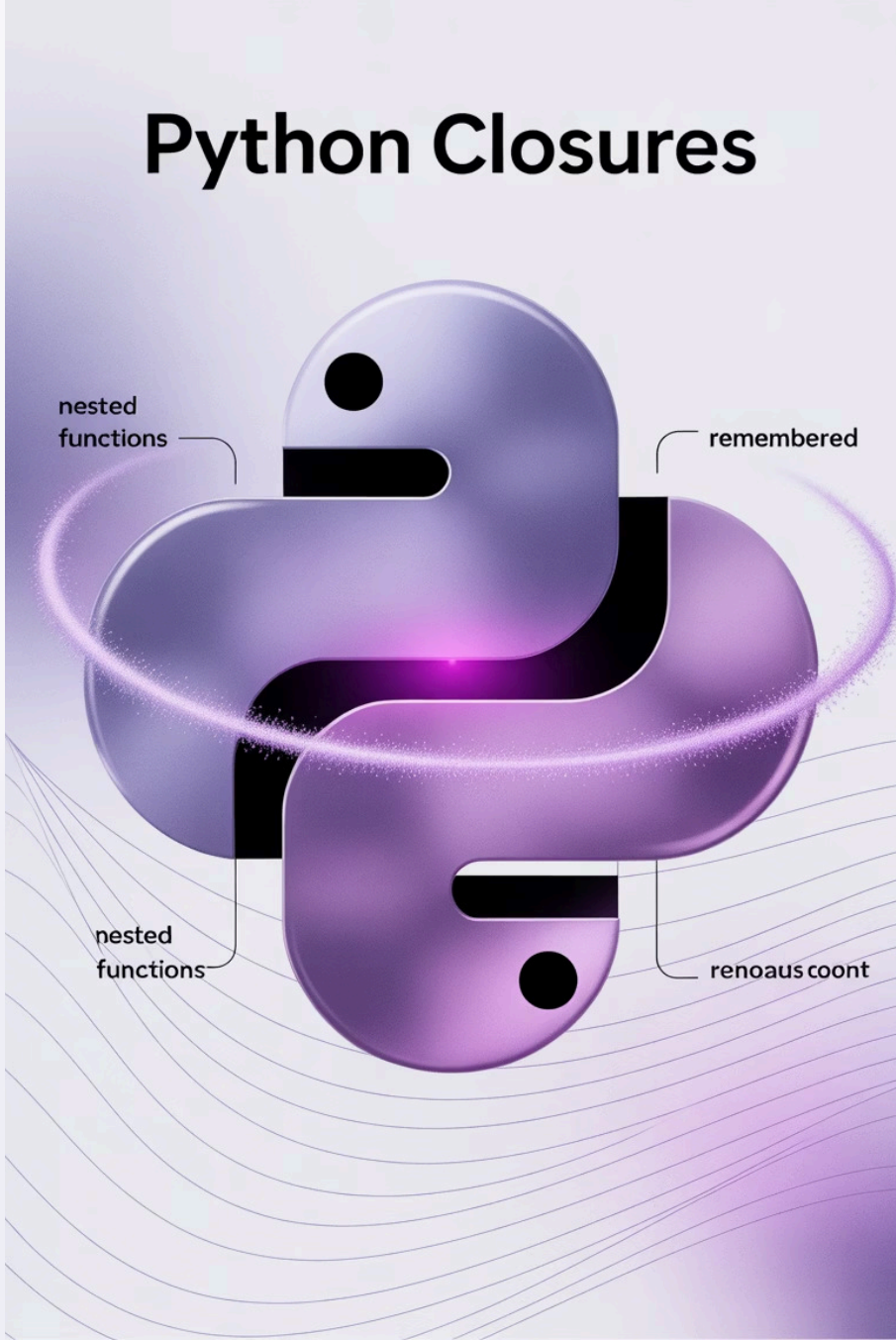
Closures

En Python, un **closure** (o **clausura**) es una función **interna** que recuerda el **estado de las variables del entorno donde fue creada**, incluso después de que ese entorno haya terminado su ejecución. o también se dice que es cuando **una función interna recuerda valores de la función externa**, incluso después de que esa función externa ya terminó.

¿Cómo se forma un closure?

Para que exista un closure, deben cumplirse estas condiciones:

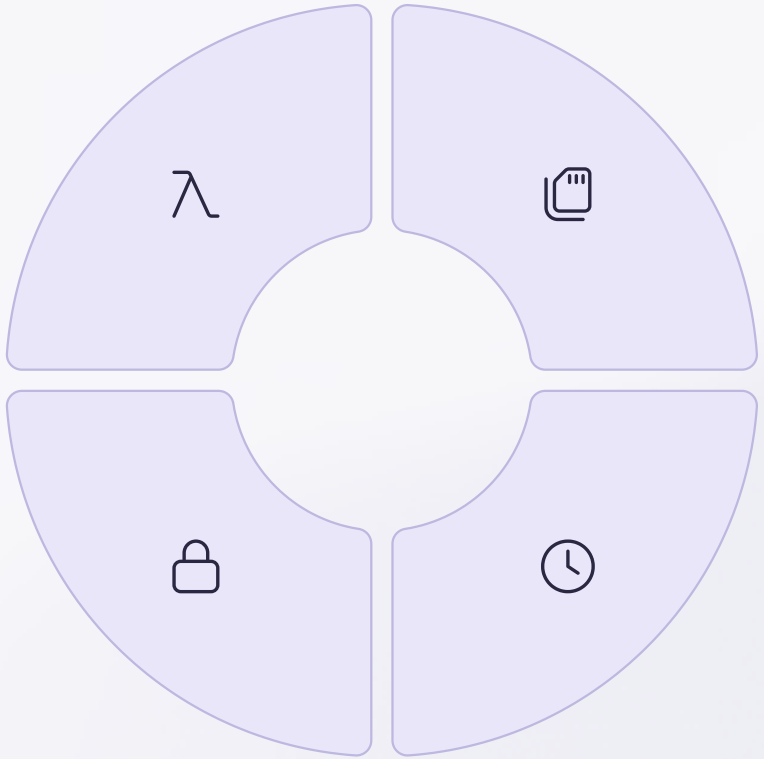
1. **Tener una función definida dentro de otra (función anidada)**.
2. La función interna **usa variables de la función externa**.
3. La función externa **retorna** la función interna.



Closures

Función anidada

Una función definida dentro de otra función.



Recuerda el entorno

Mantiene acceso a variables de la función externa.

Encapsulación

Permite mantener estado privado entre llamadas.

Persiste en el tiempo

Funciona incluso después de que la función externa termine.

Ejemplo de Closure

Código

```
def crear_contador():
    contador = 0

    def incrementar():
        nonlocal contador
        contador += 1
        return contador

    return incrementar

mi_contador = crear_contador()
print(mi_contador())  # 1
print(mi_contador())  # 2
```

Explicación

La función interna "incrementar" recuerda el valor de "contador" incluso después de que "crear_contador" haya terminado.

Cada vez que llamamos a "mi_contador()", accede y modifica la variable "contador" que está en su closure.

Esto permite mantener un estado persistente sin usar variables globales.

Resumen: Funciones en Python

Herramientas fundamentales

Las funciones son bloques esenciales para organizar y reutilizar código.

Estructura clara

Entrada, proceso y salida definen el comportamiento de cada función.

Buenas prácticas

Nombres descriptivos, responsabilidad única y documentación adecuada.

Control de ámbito

Entender los ámbitos local y global mejora el diseño del programa.

¡Gracias!

Esperamos que esta introducción a las funciones en Python te haya resultado útil y esclarecedora.

Recuerda: las funciones bien diseñadas son la clave para un código limpio, mantenible y eficiente.