Rutinas y Subrutinas en Python - Parte 3

Materia: Algoritmos y Programación

Prof. D.Sc. BARSEKH-ONJI Aboud

Facultad de Ingeniería
Universidad Anáhuac México

20 de octubre de 2025

Agenda

1. El Ámbito (Scope) de las Variables

¿Dónde "viven"las variables?

Definición: Ámbito (Scope)

El **ámbito** de una variable se refiere a la parte del programa donde esa variable es **visible** y puede ser **utilizada**.
Cuando usamos funciones, creamos dos tipos de ámbitos:

- El ámbito **Global** (el programa principal).
- El ámbito **Local** (el interior de una función).

La Analogía de la Oficina

Imagina que el **programa principal** es la **oficina** completa, con una **pizarra pública** en la pared.
Cada **función** es un **trabajador** con su propia **libreta de notas privada**.

Variables Locales (La Libreta Privada)

Definición: Variable Local

Una variable que se crea **DENTRO** de una función se llama **variable local**.

- **Nace** cuando se llama a la función.
- **Vive** solo mientras la función se está ejecutando.
- **Muere** (se destruye) cuando la función termina (con `return` o al final).
- Es **totalmente invisible** para el resto del programa.

Variables Locales (La Libreta Privada)

```
1  def calcular_promedio(lista):
2      # 'suma' y 'prom' son variables LOCALES
3      suma = 0
4      for nota in lista:
5          suma += nota
6      prom = suma / len(lista)
7      return prom
8
9  # El programa principal NO PUEDE ver 'suma' o 'prom'.
10 # Son la "libreta privada" de la funcion.
11
```

Las Variables Locales están Aisladas

¡El programa principal NO puede verlas!

Si intentas acceder a una variable local desde fuera de su función, obtendrás un `NameError`.

```
def mi_funcion():
    # 'variable_secreta' es LOCAL
    variable_secreta = "Hola123"
    print("Dentro de la funcion, puedo verla:")
    print(variable_secreta)

# --- Programa Principal ---
mi_funcion()

# Esta linea causara un ERROR
print("\nFuera de la funcion, intentando ver...")
print(variable_secreta)

```

Las Variables Locales están Aisladas

Salida en Consola

```
Dentro de la funcion, puedo verla:
Hola123

Fuera de la funcion, intentando ver...
Traceback (most recent call last):
  File "test.py", line 12, in <module>
    print(variable_secreta)
NameError: name 'variable_secreta' is not defined
```

Variables Globales (La Pizarra Pública)

Definición: Variable Global

Una variable que se crea **FUERA** de todas las funciones (en el programa principal) se llama **variable global**.

- **Nace** cuando el script comienza.
- **Vive** durante toda la ejecución del programa.
- Es **visible** para todas las funciones (pueden **LEERLA**).

Variables Globales (La Pizarra Pública)

```python
# 'nombre_app' es una variable GLOBAL
nombre_app = "Sistema de Calificaciones v1.0"

def imprimir_bienvenida():
    # La funcion PUEDE LEER la variable global
    print(f"Bienvenido a: {nombre_app}")

# --- Programa Principal ---
imprimir_bienvenida()
print(f"Fin del programa {nombre_app}")

```

El Conflicto: ¿Leer o Modificar?

¡Cuidado! El Error más Común

¿Qué pasa si una función intenta **modificar** una variable global?
Python crea una **nueva variable LOCAL** con el mismo nombre (la "ensombrece").

```
# 'x' es GLOBAL
x = 100
print(f"X (global) antes: {x}")

def intentar_modificar_x():
    # Python crea una NUEVA 'x' LOCAL
    x = 5
    print(f"X (dentro) vale: {x}")

# --- Programa Principal ---
intentar_modificar_x()
print(f"X (global) despues: {x}")

```

El Conflicto: ¿Leer o Modificar?

Salida en Consola

```
X (global) antes: 100
X (dentro) vale: 5
X (global) despues: 100
```

Análisis

La función **no** cambió la `x` global. Solo creó una `x` local y privada que se destruyó al terminar. La `x` global sigue valiendo 100.

La Palabra Clave: ‘global‘

¿Cómo forzar la modificación?

Si **realmente** necesitas modificar una variable global desde una función (lo cual es una práctica peligrosa), debes usar la palabra clave `global`.
Es como decirle a Python: "¡Oye! No crees una variable local, estoy hablando de la de la pizarra pública".

La Palabra Clave: ‘global‘

Código (con ‘global‘)

```
# 'x' es GLOBAL
x = 100
print(f"X (global) antes: {x}")

def modificar_x_real():
    # AVISO: Usare la 'x' global
    global x
    x = 5
    print(f"X (dentro) vale: {x}")

# --- Programa Principal ---
modificar_x_real()
print(f"X (global) despues: {x}")

```

Salida en Consola

```
X (global) antes: 100
X (dentro) vale: 5
X (global) despues: 5
```

Análisis

Esta vez, la x global **sí fue modificada** y el cambio permanece.

¡No uses ‘global‘! (Buenas Prácticas)

¿Por qué es una mala idea?

Usar la palabra clave `global` hace que tu código sea **impredecible** y **difícil de depurar**.

- Crea çódigo espagueti": una función puede cambiar valores en cualquier parte del programa sin previo aviso.
- La función ya no es "pura". Depende de una variable externa y oculta.

La Solución Correcta: ¡Usar ‘return‘!

Si necesitas que una función cambie un valor, haz que lo **devuelva** con `return` y re-asígnalo en el programa principal.

¡No uses ‘global‘! (Buenas Prácticas)

MALO (con ‘global‘)

```
contador = 0
def incrementar():
    global contador
    contador += 1

incrementar()
print(contador) # 1

```

BUENO (con ‘return‘)

```
contador = 0
def incrementar(valor_actual):
    return valor_actual + 1

contador = incrementar(contador)
print(contador) # 1

```

Resumen y Conclusiones

Lo que aprendimos

- **Subrutinas (Funciones):** Son bloques de código reutilizables definidos con `def`. Nos ayudan a organizar el código y evitar repetirnos (Principio DRY).
- **Argumentos y Parámetros:** Son la forma de **pasar información** (ingredientes) *hacia adentro* de una función.
- **Valor de Retorno (‘return‘):** Es la forma en que una función **devuelve un resultado** (un cálculo) *hacia afuera*.
- **Ámbito (Scope):**
  - **Variables Locales:** Viven *dentro* de la función. Son privadas y se destruyen al terminar.
  - **Variables Globales:** Viven *fuera* de la función. Son públicas.

La Regla de Oro

Una función debe ser como una "caja negra": recibir datos solo por sus **parámetros** y entregar resultados solo por su **‘return‘**. Evita siempre que puedas las variables globales.