

Variables globales

Edison Achalma

Escuela Profesional de Economía, Universidad Nacional de San Cristóbal de Huamanga

Resumen

Este abstract será actualizado una vez que se complete el contenido final del artículo.

Tabla de contenidos

Introduction	2
0.1 Motivación	2
0.2 Intuición	2
0.3 Modelo mental	2
0.4 Desarrollo	2
0.4.1 1. Variables globales (acceso)	2
0.4.2 2. Variable local con mismo nombre	2
0.4.3 3. La palabra clave global	3
0.4.4 4. Crear variable global dentro de función	3
0.4.5 5. Modificar variables globales vs locales	3
0.5 Ejemplos	3
0.6 Errores comunes	4
0.7 Buenas prácticas	5
0.8 Ejercicio	5
0.9 Síntesis	7
0.10 Qué sigue	7
1 Publicaciones Similares	7

Edison Achalma  <https://orcid.org/0000-0001-6996-3364>

El autor no tiene conflictos de interés que revelar. Los roles de autor se clasificaron utilizando la taxonomía de roles de colaborador (CRediT; <https://credit.niso.org/>) de la siguiente manera: Edison Achalma: conceptualización, metodología, análisis formal, investigación, recursos, curación de datos, redacción, visualización, supervisión, administración del proyecto

La correspondencia relativa a este artículo debe dirigirse a Edison Achalma, Escuela Profesional de Economía, Universidad Nacional de San Cristóbal de Huamanga, Portal Independencia N° 57, Ayacucho, AYA 5001, Perú, Email: elmer.achalma.09@unsch.edu.pe

Variables globales

0.1 Motivación

Cuando trabajas con funciones, necesitas entender cómo Python maneja las variables en diferentes contextos. Las variables globales son accesibles desde cualquier parte de tu programa, pero modificarlas dentro de funciones requiere cuidado. Dominar el alcance (scope) de las variables es esencial para escribir código predecible y evitar errores difíciles de depurar.

0.2 Intuición

Imagina que tienes una pizarra en la entrada de tu casa (variable global) donde toda la familia puede escribir mensajes. Todos pueden leer lo que está en la pizarra. Pero si alguien escribe un mensaje en su propia libreta personal (variable local) con el mismo título, no afecta lo que está en la pizarra. Python funciona igual: una variable dentro de una función es como una libreta privada, separada de la pizarra global.

0.3 Modelo mental

Python sigue las reglas **LEGB** para la resolución de variables:

1. **Local** → dentro de la función actual
2. **Enclosing** → funciones externas (anidadas)
3. **Global** → variables definidas fuera de funciones
4. **Built-in** → funciones y variables incorporadas de Python

Cuando usas una variable dentro de una función, Python busca en ese orden: primero local, luego global si no encuentra local.

0.4 Desarrollo

0.4.1 1. Variables globales (acceso)

Las variables creadas fuera de funciones son globales y accesibles dentro de funciones:

```
x = "awesome" # Variable global

def myfunc():
    print("Python is " + x) # Accede a la variable global

myfunc() # Salida: Python is awesome
```

0.4.2 2. Variable local con mismo nombre

Si defines una variable con el mismo nombre dentro de una función, **es una variable local independiente**:

```
x = "awesome" # Global

def myfunc():
    x = "fantastic" # Local (independiente)
    print("Python is " + x) # Usa la local
```

```
myfunc() # Salida: Python is fantastic
print("Python is " + x) # Salida: Python is awesome (global)
```

0.4.3 3. La palabra clave global

Para **modificar** una variable global dentro de una función, usa global:

```
x = "awesome" # Global

def myfunc():
    global x # Declara que vas a usar/modificar la global
    x = "fantastic" # Modifica la global

myfunc()
print("Python is " + x) # Salida: Python is fantastic
```

0.4.4 4. Crear variable global dentro de función

Puedes crear una nueva variable global desde dentro de una función:

```
def myfunc():
    global x
    x = "nueva variable" # Crea una variable global

myfunc()
print(x) # Salida: nueva variable
```

0.4.5 5. Modificar variables globales vs locales

```
contador = 0 # Global

def incrementar():
    global contador # Necesario para modificar
    contador += 1

def mostrar():
    print(contador) # Solo lectura, no necesita global

incrementar()
incrementar()
mostrar() # Salida: 2
```

0.5 Ejemplos

Mínimo — Acceso a global:

```

nombre = "Ana" # Global

def saludar():
    print("Hola", nombre) # Accede a la global

saludar() # Hola Ana

```

Progresivo — Local vs Global:

```

mensaje = "Global"

def funcion1():
    mensaje = "Local"
    print("Dentro:", mensaje) # Local

def funcion2():
    print("Fuera:", mensaje) # Global

funcion1() # Dentro: Local
funcion2() # Fuera: Global
print("Principal:", mensaje) # Principal: Global

```

Real — Contador de visitas:

```

visitas = 0 # Global

def visitar_pagina():
    global visitas
    visitas += 1
    print(f"Visitante #{visitas}")

def mostrar_estadisticas():
    print(f"Total de visitas: {visitas}")

# Simulación de visitas
visitar_pagina() # Visitante #1
visitar_pagina() # Visitante #2
visitar_pagina() # Visitante #3
mostrar_estadisticas() # Total de visitas: 3

```

0.6 Errores comunes

Error	Síntoma	Solución
Modificar global sin global	Se crea una local en lugar de modificar	Añade global variable
Usar variable sin inicializar	UnboundLocalError	Inicializa la variable antes de usarla

Error	Síntoma	Solución
Confundir scope	La variable no tiene el valor esperado	Usa <code>global</code> para modificar, no para leer
Crear muchas globales	Código difícil de seguir	Usa parámetros en lugar de globales

0.7 Buenas prácticas

1. **Evita usar variables globales a menos que sea necesario:** pasa valores como argumentos a las funciones.
2. **Usa globales solo para configuraciones o contadores:**

```
# Aceptable
CONFIG = {"debug": True, "timeout": 30}

# Buen uso de global
visitas = 0
def incrementar_visitas():
    global visitas
    visitas += 1
```

3. **Nombra las globales con mayúsculas** (por convención):

```
MAX_INTENTOS = 3 # Constante global
BASE_URL = "https://api.ejemplo.com" # Configuración
```

4. **Si necesitas modificar una variable, considera pasar por parámetros:**

```
# Uso global innecesario
saldo = 100
def sumar(monto):
    global saldo
    saldo += monto

# Mejor usar parámetros
def sumar(saldo, monto):
    return saldo + monto
```

5. **Documenta las variables globales** para que sea claro qué hacen.

0.8 Ejercicio

Básico — Análisis de alcance

Analiza el siguiente código y determina qué imprime cada `print()`:

```
mensaje = "Hola mundo"

def funcion_a():
    mensaje = "Hola desde A"
    print(mensaje) # 1

def funcion_b():
    global mensaje
    mensaje = "Modificado globalmente"
    print(mensaje) # 2

def funcion_c():
    print(mensaje) # 3

funcion_a()
print(mensaje) # 4
funcion_b()
print(mensaje) # 5
funcion_c()
```

Solución (no mires hasta haberlo intentado):

```
mensaje = "Hola mundo"

def funcion_a():
    mensaje = "Hola desde A" # Variable local
    print(mensaje) # 1. Imprime: Hola desde A

def funcion_b():
    global mensaje # Acceso a la global
    mensaje = "Modificado globalmente" # Cambia la global
    print(mensaje) # 2. Imprime: Modificado globalmente

def funcion_c():
    print(mensaje) # 3. Imprime la global (que es: Modificado globalmente)

funcion_a()
# 4. Imprime: Hola mundo (la global no cambió)

funcion_b()
# 5. Imprime: Modificado globalmente (la global cambió)

funcion_c()
# 6. Imprime: Modificado globalmente (la global sigue igual)
```

Salida esperada:

```
Hola desde A
Hola mundo
Modificado globalmente
Modificado globalmente
Modificado globalmente
```

0.9 Síntesis

- **Global:** variable definida fuera de funciones, accesible en todo el programa.
- **Local:** variable definida dentro de una función, solo accesible allí.
- **Leer global** no requiere `global`; solo usas el nombre.
- **Modificar global** requiere `global` variable.
- **Crear global** dentro de función: usa `global` para crearla.
- `global` declara que la variable pertenece al scope global.
- **Buena práctica:** minimiza el uso de variables globales.

0.10 Qué sigue

- **Relacionado:** funciones (alcance de variables, parámetros y argumentos)
- **Siguiente tema:** Funciones — aprenderás a crear y usar funciones, y a manejar correctamente el alcance de variables en diferentes contextos.

1 Publicaciones Similares

Si te interesó este artículo, te recomendamos que explores otros blogs y recursos relacionados que pueden ampliar tus conocimientos. Aquí te dejo algunas sugerencias:

1. [!\[\]\(2cbb40928a34ecf5ce700a63c52aa374_img.jpg\) Instalacion De Anaconda](#)
2. [!\[\]\(ce05ba64c497267b6ad2e23c0c6ca4e1_img.jpg\) Configurar Entorno Virtual Python Anaconda](#)
3. [!\[\]\(3ce668af77d7d4613bd5f40b461563b3_img.jpg\) 01 Introducion A La Programacion Con Python](#)
4. [!\[\]\(387cb8ccc1ffc04d4578691a69e06f86_img.jpg\) 02 Variables Expresiones Y Statements Con Python](#)
5. [!\[\]\(a66c868b0d540407d014071bb67988fc_img.jpg\) 03 Objetos De Python](#)
6. [!\[\]\(0d8c9935ad518c25b1e28b49b0e478f2_img.jpg\) 04 Ejecucion Condicional Con Python](#)
7. [!\[\]\(fcf0e61878c20117a254f63a80c035bc_img.jpg\) 05 Iteraciones Con Python](#)
8. [!\[\]\(9fdb55ba1a0ca3d26427df15121f776f_img.jpg\) 06 Funciones Con Python](#)
9. [!\[\]\(429c63034d14d338c605e1ce2f47f527_img.jpg\) 07 Dataframes Con Python](#)
10. [!\[\]\(57cb4bddaabec833baf5496d75408fee_img.jpg\) 08 Prediccion Y Metrica De Performance Con Python](#)
11. [!\[\]\(ec5157b5792cabaad33b19ff4bcbe7a6_img.jpg\) 09 Metodos De Machine Learning Para Clasificacion Con Python](#)
12. [!\[\]\(5b0c8b27764ac8fd185202b8a6c8918e_img.jpg\) 10 Metodos De Machine Learning Para Regresion Con Python](#)
13. [!\[\]\(0fe6a212958ad9d5281279b469b4b118_img.jpg\) Visualizacion De Datos Con Python](#)

Esperamos que encuentres estas publicaciones igualmente interesantes y útiles. ¡Disfruta de la lectura!