|  |
| --- |
| **Ámbitos o Scope** |
| En Python, los ámbitos o scopes determinan la accesibilidad y visibilidad de los distintos tipos de objetos en diferentes partes del código.  Un objeto en ciertos ámbitos puede ser modificado reescrito, con funciones o con métodos de los objetos - Accesibilidad total.  Un objeto en ciertos ámbitos solo puede mostrar su contenido – Visibilidad y utilizar solo los métodos y atributos que no modifica el objeto.  Un objetos con un solo dato creado en la raiz ( root - sobre el margen) tipo str, int, float, bool, puede ser leídos y no modificados si no se les marca como globales.  Un objetos múltiple – Colección con un conjunto de datos creado en la raiz ( root - sobre el margen) tipo list, set, dict,etc (no tuplas o frozensets), puede ser leídos y modificados sin necesidad de marcarlo como globales.  Si los objetos se crean en una función o método no existen en otros ámbitos - otras funciones o métodos   |  | | --- | | El siguiente contenido se tomo de bibliografía para los que requieren mas información pero no son parte de este curso. |   El **scope** en Python define el contexto en el que las variables y objetos son accesibles o referenciados. Existen cuatro niveles de ámbito en Python, conocidos como **LEGB**:   * **L (Local)**: Variables definidas dentro de una función. Solo son accesibles dentro de esa función. * **E (Enclosing)**: Variables definidas en funciones envolventes (funciones dentro de funciones). Se puede acceder a ellas desde las funciones internas. * **G (Global)**: Variables definidas en el nivel superior del archivo o módulo. Son accesibles en cualquier parte del script o módulo donde se definen, a menos que haya una variable local con el mismo nombre. * **B (Built-in)**: Variables y funciones predefinidas por Python, como print(), len(), etc.   **Reglas clave del scope**:   * Las variables locales tienen prioridad sobre las globales dentro de una función. * Para modificar una variable global dentro de una función, se debe declarar explícitamente usando la palabra clave global.  |  | | --- | | **LEGB** | | Ámbito Local (Local Scope):  El ámbito local se refiere al ámbito dentro de una función o método.  Las variables definidas dentro de una función solo son accesibles dentro de esa función.  Si se intenta acceder a una variable local fuera de su función, se producirá un error. | | Ámbito Encerrado (Enclosing Scope)  El ámbito encerrado se refiere al ámbito que se encuentra alrededor de una función anidada.  Ocurre cuando hay una función definida dentro de otra función.  Las variables definidas en el ámbito encerrado son accesibles tanto dentro de la función anidada como en la función externa. | | Ámbito Global (Global Scope):  El ámbito global se refiere al ámbito fuera de cualquier función o clase.  Las variables definidas en el ámbito global son accesibles desde cualquier parte del código, incluyendo dentro de funciones y clases. | | Ámbito de Nombre Integrado (Built-in Scope):  El ámbito de nombre integrado se refiere a las funciones y variables predefinidas que están disponibles en Python sin necesidad de importar ningún módulo.  Incluye funciones y objetos integrados como print(), len(), range(), etc.  Estas funciones y variables están disponibles en cualquier parte del código.  Es importante tener en cuenta que hay una jerarquía en la resolución de nombres en Python, lo que significa que Python busca una variable en el ámbito local primero, luego en el ámbito encerrado, seguido del ámbito global y finalmente en el ámbito de nombre integrado. Si no se encuentra la variable en ningún ámbito, se genera un error de "NameError". |   En este ejemplo, la variable x tiene un ámbito global y es accesible en todas partes del código. La variable y tiene un ámbito local dentro de la función func(), mientras que la variable z tiene un ámbito encerrado dentro de la función nested\_func(). Las variables x, y y z son accesibles dentro de sus respectivos ámbitos.   |  | | --- | | # Ámbito global  x = 10  y = "Sin Datos root"  z = "Sin Datos root"  def func():  # Ámbito local  y = 20  def nested\_func():  # Ámbito encerrado  z = 30  print(f"dentro de función anidada: {x=} {y=} {z=}") # Accede a las variables x, y y z  nested\_func()  print(f"dentro de función: {x=} {y=} {z=}") # Accede a las variables x e y  func()  print(f"root: {x=} {y=} {z=}") # Accede a la variable x |   ·   |  | | --- | | dentro de función anidada: x=10 y=20 z=30  dentro de función: x=10 y=20 z='Sin Datos root'  root: x=10 y='Sin Datos root' z='Sin Datos root' |   · |

.

|  |
| --- |
| **Global y Nonlocal** |
| **Global**:  Permite modificar una variable global dentro de una función. Sin global, las variables globales solo pueden ser leídas, no modificadas.   |  | | --- | | x = 10 def modificar\_global():  global x # Declara que se modificará la variable global 'x'  x = 20 modificar\_global() print(x) # Salida: 20 |   . |
| **Nonlocal**:  Se utiliza para modificar una variable en un ámbito envolvente (Enclosing Scope), pero que no sea globa   |  | | --- | | def outer\_function():  x = "enclosing"  def inner\_function():  nonlocal x # Permite modificar 'x' en el ámbito de outer\_function  x = "modificado por inner\_function"  inner\_function()  print("Dentro de outer\_function:", x) # Salida: "modificado por inner\_function" outer\_function() |   . |

.

|  |
| --- |
| **Objetos Mutabilidad e Inmutabilidad creados en main** |
| **Objetos Inmutables en Funciones**   * **Lectura**: Los objetos inmutables (como int, str, tuple, etc.) **pueden leerse** dentro de una función sin necesidad de pasarlos como parámetro o declararlos globales. Python busca estas variables siguiendo la regla de ámbitos (LEGB).  |  | | --- | | x = 10 # Inmutable  def leer\_x():  print(x) # Lee la variable global 'x'  leer\_x() # Salida: 10) |  * **Escritura**: Si intentas **modificar** un objeto inmutable (sin usar global), Python creará una **nueva variable local** dentro de la función, sin modificar la original. Para modificar un objeto inmutable en el ámbito global o en un ámbito superior, **debes usar** la palabra clave global o nonlocal, de lo contrario, solo se modificará una copia local en la función.  |  | | --- | | x = 10 # Inmutable  def modificar\_x():  x = 20 # Esto crea una nueva variable local 'x', no afecta la global  print("Dentro de la función:", x) # Salida: 20  modificar\_x()  print("Fuera de la función:", x) # Salida: 10 (la variable global no se modifica) |   Si quieres modificar la variable global x, necesitarás declarar x como global:   |  | | --- | | x = 10 # Inmutable  def modificar\_global\_x():  global x  x = 20 # Modifica la variable global 'x'  print("Dentro de la función:", x) # Salida: 20  modificar\_global\_x()  print("Fuera de la función:", x) # Salida: 20 (la variable global se modifica) |   . |
| **Objetos Mutables en Funciones**   * **Lectura**: Al igual que los objetos inmutables, los **objetos mutables** (como list, dict, set, etc.) **pueden leerse** dentro de una función sin necesidad de pasarlos como parámetros o declararlos globales. Python seguirá la regla de ámbitos para buscarlos.  |  | | --- | | lista = [1, 2, 3] # Mutable  def leer\_lista():  print(lista) # Lee la variable global 'lista'  leer\_lista() # Salida: [1, 2, 3] |  * **Escritura**: Los objetos mutables **pueden modificarse sin necesidad de usar** global, porque al operar sobre el objeto (por ejemplo, agregar o modificar un elemento), no estamos reasignando la variable, sino alterando el contenido del objeto referenciado. Este comportamiento es distinto al de los inmutables.  |  | | --- | | lista = [1, 2, 3] # Mutable  def modificar\_lista():  lista.append(4) # Modifica el contenido de la lista  print("Dentro de la función:", lista) # Salida: [1, 2, 3, 4]  modificar\_lista()  print("Fuera de la función:", lista) # Salida: [1, 2, 3, 4] (se modificó) |   En este ejemplo, la lista **mutable** lista se modifica dentro de la función, y ese cambio es visible fuera de la función, **sin necesidad de declarar** lista **como global**, porque no estamos reasignando la variable, sino modificando su contenido.  . |
| Sin embargo, si intentas **reasignar** el objeto mutable a un nuevo objeto sin usar global, crearás una nueva variable local en la función, de manera similar a los inmutables:   |  | | --- | | lista = [1, 2, 3] # Mutable  def reasignar\_lista():  lista = [10, 20, 30] # Esto crea una nueva lista local  print("Dentro de la función:", lista) # Salida: [10, 20, 30]  reasignar\_lista()  print("Fuera de la función:", lista) # Salida: [1, 2, 3] (la lista global no se modificó) |   Si deseas modificar la lista global mediante **reasignación**, necesitarás usar global:   |  | | --- | | lista = [1, 2, 3] # Mutable  def reasignar\_global\_lista():  global lista  lista = [10, 20, 30] # Modifica la variable global 'lista'  print("Dentro de la función:", lista) # Salida: [10, 20, 30]  reasignar\_global\_lista()  print("Fuera de la función:", lista) # Salida: [10, 20, 30] (se modificó) |   . |

.