



Artículo

# Scopes and namespaces

**David Aroesti** ⌚ 17 de Octubre de 2018

En Python, un name, también conocido como identifier, es simplemente una forma de otorgarle un nombre a un objeto. Mediante el nombre, podemos acceder al objeto. Vamos a ver un ejemplo:

```
my_var = 5

id(my_var) # 4561204416
id(5) # 4561204416
```

En este caso, el identifier `my_var` es simplemente una forma de acceder a un objeto en memoria (en este caso el espacio identificado por el número 4561204416). Es importante recordar que un name puede referirse a cualquier tipo de objeto (aún las funciones).

```
def echo(value):
    return value

a = echo

a('Billy') # 3
```

Ahora que ya entendimos qué es un name podemos avanzar a los namespaces (espacios de nombres). Para ponerlo en palabras llanas, un namespace es simplemente un conjunto de names.



Introducción a Click

En Python, te puedes imaginar que existe una relación que liga a los nombres definidos con sus respectivos objetos (como un diccionario). Pueden coexistir varios namespaces en un momento dado, pero se encuentran completamente aislados. Por ejemplo, existe un namespace específico que agrupa todas las variables globales (por eso puedes utilizar varias funciones sin tener que importar los módulos correspondientes) y cada vez que declaramos un módulo o una función, dicho módulo o función tiene asignado otro namespace.

A pesar de existir una multiplicidad de namespaces, no siempre tenemos acceso a todos ellos desde un punto específico en nuestro programa. Es aquí donde el concepto de scope (campo de aplicación) entra en juego.

Scope es la parte del programa en el que podemos tener acceso a un namespace sin necesidad de prefijos.

En cualquier momento determinado, el programa tiene acceso a tres scopes:

- El scope dentro de una función (que tiene nombres locales)
- El scope del módulo (que tiene nombres globales)
- El scope raíz (que tiene los built-in names)

Cuando se solicita un objeto, Python busca primero el nombre en el scope local, luego en el global, y por último, en el raíz. Cuando anidamos una función dentro de otra función, su scope también queda anidado dentro del scope de la función padre.

```
def outer_function(some_local_name):  
    def inner_function(other_local_name):  
        # Tiene acceso a la built-in function print y al nombre local  
        some_local_name  
        print(some_local_name)
```





- ara poder manipular una variable que se encuentra fuera del scope local podemos utilizar los keywords global y nonlocal.

```
33 some_var_in_other_scope = 10

34 def some_function():
    global some_var_in_other_scope

35     Some_var_in_other_scope += 1
```

36  Escribe aquí tu pregunta + 2

37  Wilson Delgado Estudiante • hace 8 días

38 **Scope** parte del programa desde donde los namespaces pueden ser accedidos

39  1 

40  wilantury Estudiante • hace 19 días

‘nonlocal’: Se usa para obtener el scope en funciones con funciones anidadas.

41 ‘Global’: Se usa para tener el scope del módulo.

42  1 

 wilantury Estudiante • hace 19 días

Para tener acceso a una variable con scope global, se debe anteponer la palabra reservada “global” en el scope que se desee usar.

 1 

 Edwin Jesset Barrientos Gonzales Estudiante • hace 2 meses

Excelente buena info.

 1 

 innacroft Estudiante • hace 3 meses



Introducción a Click



10



andresflm · hace 2 meses

Genial, gracias por los aportes!!

2



sergioabelgordondolimpio · hace 2 meses

buen aporte, mucho mas claro

1



Gerardo Bósquez · hace 3 días

Excelente aporte, explica todo con mucho detalle. Gracias!!

1

[Ver más respuestas](#)

innacraft Estudiante · hace 3 meses

```
defouter_function():  
    b = 20 <----- La variable b se encuentra en el namespace LOCAL  
    definner_func():  
        c = 30 <----- La variable c se encuentra en el namespace LOCAL ANIDADO  
a = 10 <----- La variable a se encuentra en el namespace GLOBAL``
```



5



marcelosanchez21 Estudiante · hace 4 meses

Tengo un duda en el codigo

```
global some_var_in_other_scope  
  
Some_var_in_other_scope += 1
```

En este caso no se está modificando la variable "some\_var\_in\_other\_scope = 10" va que una



Introducción a Click



Daniel Correa Estudiante · hace 4 meses

```
def my_function():  
    x = 1  
  
    def my_sub_function():  
        nonlocal x #gracias a esto digamos que hacemos 'global' pero solo dentro  
de una funcion  
        x = "esto no es 1"  
        print(x) #"esto no es 1"  
  
    my_sub_function()  
    print(x) # "esto no es 1"  
  
my_function()  
print(x) #error
```



1



IsmaelFajardo Estudiante · hace 11 meses

¿Entiendo entonces que 'global' permite a una función, en cualquier nivel, acceder a variables que han sido declaradas en el scope global y, por otro lado, 'nonlocal' le permite a una función anidada ingresar a la variable que se encuentra al igual que ella dentro de determinada función, cierto? Si es así, ¿cuantos niveles puede escalar 'nonlocal'?



3



Ana Lima · hace 11 meses

¡Hola Ismael! Pareciera que no tiene un limite de niveles para usar, pero no es lo recomendable, te recomiendo que declares global y no nonlocal en cada método en que vayas a usar.

1



Perla Godinez Castillo Estudiante · el año pasado

<https://www.programiz.com/python-programming/iterator>



1



Introducción a Click



1



Jesús Maximiliano Manchuk Estudiante • el año pasado

<https://www.youtube.com/watch?v=QVdf0LgmICw>

complementa la información dada. (Está en ingles)



2



alvamar zapata • hace 5 meses

gracias!

1



elbarbero400 Estudiante • el año pasado



24



Adolam • el año pasado



Introducción a Click



Neiro13 · el año pasado

Gracias, bien grafico.

1



u.mitchel · hace 7 meses

Incredible como una simple imagen aclara tanto, muchas gracias por tu aporte.

1



zkmark9999 · hace 4 meses

buen ejemplo

1

[Ver más respuestas](#)

elbarbero400 Estudiante · el año pasado

```
some_var_in_other_scope = 10
c = 10
```

```
defecho(value):
    return value
```

```
defouter_function(some_local_name):
    definner_function(other_local_name):
        # Tiene acceso a la built-in function print y al nombre local some_local_name
        print(some_local_name)
        # También tiene acceso a su scope local
        print(other_local_name)
```

```
defsome_function():
    global some_var_in_other_scope
    some_var_in_other_scope += 1
```

```
defouter_function_2():
    global c
    c = 20
    definner_function():
```

[Introducción a Click](#)

```
inner_function()
print('c =',c)

if __name__ == '__main__':
    a = echo
    a = a('Billy') # 3
    print(a)

    b = outer_function
    print(b(some_var_in_other_scope))

    some_function()
    print(some_var_in_other_scope)

outer_function_2()
print('c =',c)````
```



ingkstr Estudiante · el año pasado

Igualmente me sale desordenado el código 😞



Facundo Nicolás García Martoni · hace 6 meses

¡Hola @ingkstr! ¿Podrías compartirme una captura de pantalla? 🤔

1



Luis Mercado Estudiante · el año pasado

Deben corregir como se muestra el código, se ve muy desorganizado y algunas palabras juntas.



Nestor Cepeda · el año pasado

Estaba pensando exactamente lo mismo.

```
defouter_function(some_local_name):
```



Introducción a Click



4



Kevin Lozoya Giner · el año pasado

Si después de las primeras ``` al editar pones python se soluciona (```python)

```
defouter_function(some_local_name):
```

2



Facundo Nicolás García Martoni · hace 6 meses

¡Hola @lemf0209! Esto es un error en el editor de comentarios de Platzi, quiero que sepas que ya está reportado y el equipo se encuentra trabajando en su solución 🐛

1



Jorge Mayorga · hace 4 meses

```
defouter_function(some_local_name):
```

1



Jorge Mayorga · hace 4 meses

Está arreglado

1

[Ver más respuestas](#)

victor0402 Estudiante · hace 2 años

Solo por incluir algo a lo ya mencionado:

En Python, la palabra clave **global** permite modificar la variable fuera del alcance actual. Se utiliza para crear una variable global y realizar cambios en la variable en un contexto local.

### Reglas de palabra clave global

Las reglas básicas para la palabra clave **global** en Python son:

- Cuando creamos una variable dentro de una función, es local por defecto.
- Cuando definimos una variable fuera de una función, es global por defecto. No tienes que usar la

[Introducción a Click](#)

En lo referente a la palabra **nonlocal**, actúa de manera similar, solo que orientada a lo que sería un alcance de funciones y funciones anidadas, a continuación un ejemplo:

```
def method():  
  
    def method2():  
        # este método no tiene acceso a la  
        # variable value, por cuanto se usa  
        # nonlocal para poder acceder.  
        nonlocal value  
        value = 100  
  
    # Variable local.  
    value = 10  
    method2()
```



48



Rhonal Velasco · hace 2 años

Buen aporte

7



daniel-medina137 · el año pasado

Gracias por el aporte

1



LuisEdsonQC · el año pasado

Por fin entendí la funcionalidad del keyword 'global'  
Gracias!!!

1



Matias Molina · el año pasado

buen aporte muchas gracias !

1

[Ver más respuestas](#)

Introducción a Click



Introducción a Click