Funciones (1)

En esta lección se trata la creación de funciones en Python y el uso de variables en las funciones. Otros conceptos relativos a las funciones se tratan en la lección Funciones (2):

Definir funciones en Python

Las funciones se pueden crear en cualquier punto de un programa, escribiendo su definición.

La primera línea de la definición de una función contiene:

- la palabra reservada def
- el nombre de la función (la guía de estilo de Python recomienda escribir todos los caracteres en minúsculas separando las plabras por guiones bajos)
- paréntesis (que pueden incluir los argumentos de la función, como se explica más adelante)

Las instrucciones que forman la función se escriben con sangría con respecto a la primera línea.

Por comodidad, se puede indicar el final de la función con la palabra reservada return (más adelante se explica el uso de esta palabra reservada), aunque no es obligatorio.

Para poder utilizar una función en un programa se tiene que haber definido antes. Por ello, normalmente las definiciones de las funciones se suelen escribir al principio de los programas

El ejemplo siguiente muestra un programa que contiene una función y el resultado de la ejecución de ese programa.

Ejemplo de función

```
def licencia():
    print("Copyright 2013 Bartolomé
Sintes Marco")
    print("Licencia CC-BY-SA 4.0")
    return

print("Este programa no hace nada
interesante.")
licencia()
print("Programa terminado.")
```

```
Este programa no hace nada interesante.
Copyright 2013 Bartolomé Sintes Marco
Licencia CC-BY-SA 4.0
Programa terminado.
```

Puede ver la ejecución paso a paso de este programa utilizando los iconos de avance y retroceso situados abajo a la derecha.

El ejemplo siguiente muestra un programa incorrecto que intenta utilizar una función antes de haberla definido.

Ejemplo de función incorrecta

```
print("Este programa no hace nada
interesante.")
licencia()
print("Programa terminado.")

def licencia():
    print("Copyright 2013 Bartolomé
Sintes Marco")
    print("Licencia CC-BY-SA 4.0")
    return
```

```
Este programa no hace nada interesante.
Traceback (most recent call last):
  File "ejemplo.py", line 2, in <module>
    licencia()
NameError: name 'licencia' is not defined
```

Puede ver la ejecución paso a paso de este programa utilizando los iconos de avance y retroceso situados abajo a la derecha.

Variables en funciones

Conflictos de nombres de variables

Como se ha comentado antes, una de las principales ventajas de las subrutinas es que permiten reutilizar el código. Pero copiar y pegar subrutinas de un programa a otro puede producir lo que se llama un **conflicto de nombres de variables**. En efecto, si la subrutina que pegamos en un programa utiliza alguna variable auxiliar para algún cálculo intermedio y resulta que el programa ya utilizaba una variable con el mismo nombre que esa variable auxiliar, los cambios en la variable que se hagan en la subrutina podrían afectar al resto del programa de forma imprevista.

Para resolver el problema de los conflictos de nombres, los lenguajes de programación limitan lo que se llama el alcance o el ámbito de las variables. Es decir, que los lenguajes de programación permiten que una variable exista únicamente en el interior de una subrutina y no afecte a otras variables de mismo nombre situadas fuera de esa subrutina. Como las subrutinas pueden contener a su vez subrutinas, se suele hablar de niveles: el nivel más alto sería el programa principal, el siguiente nivel serían las subrutinas incluidas en el programa principal y cada vez que hay una subrutina incluida dentro de otra estaríamos bajando un nivel.

El problema es más complicado de lo que parece a primera vista, porque a menudo también nos interesará que una subrutina pueda modificar variables que estén definidas

en otros puntos del programa. Así que los lenguajes de programación tienen que establecer mecanismos para aislar las variables y evitar los conflictos de nombres, pero al mismo tiempo deben permitir el acceso a las variables en los casos que así lo quiera el programador.

Aunque cada lenguaje tiene sus particularidades, el mecanismo más habitual se basa en los siguientes principios:

- cada variable pertenece a un ámbito determinado: al programa principal o a una subrutina.
- las variables son completamente inaccesibles en los ámbitos superiores al ámbito al que pertenecen
- las variables pueden ser accesibles o no en ámbitos inferiores al ámbito al que pertenecen (el lenguaje puede permitir al programador elegir o no esa accesibilidad)
- en cada subrutina, las variables que se utilizan pueden ser entonces:
  - variables locales: las que pertenecen al ámbito de la subrutina (y que pueden ser accesibles a niveles inferiores)
  - variables libres: las que pertenecen a ámbitos superiores pero son accesibles en la subrutina

---

Conflictos de nombres de variables en Python

Python sigue estos principios generales, pero con algunas particularidades:

- En los lenguajes tipificados, como se debe declarar las variables que se utilizan, la declaración se aprovecha para indicar si la variable pertenece a la subrutina o procede de un ámbito superior. Pero como Python no es un lenguaje tipificado, el ámbito de pertenencia de la variable debe deducirse del programa siguiendo unas reglas que se detallan más adelante (aunque Python también permite declarar explícitamente el ámbito en los casos en que se quiera un ámbito distinto al determinado por las reglas).
- Python distingue tres tipos de variables: las variables locales y dos tipos de variables libres (globales y no locales):
  - variables locales: las que pertenecen al ámbito de la subrutina (y que pueden ser accesibles a niveles inferiores)
  - variables globales: las que pertenecen al ámbito del programa principal.
  - variables no locales: las que pertenecen a un ámbito superior al de la subrutina, pero que no son globales.

  Si el programa contiene solamente funciones que no contienen a su vez funciones, todas las variables libres son variables globales. Pero si el programa contiene una función que a su vez contiene una función, las variables libres de esas "subfunciones" pueden ser globales (si pertenecen al programa principal) o no locales (si pertenecen a la función).

- Para identificar explícitamente las variables globales y no locales se utilizan las palabras reservadas global y nonlocal. Las variables locales no necesitan identificación. La palabra reservada nonlocal se introdujo en Python 3 (PEP 3104).

A continuación, se detallan las reglas y situaciones posibles, acompañadas de ejemplos.

Variables locales

- Si no se han declarado como globales o no locales, las variables **a las que se asigna valor** en una función se consideran variables **locales**, es decir, sólo existen en la propia función, incluso cuando en el programa exista una variable con el mismo nombre, como muestra el siguiente ejemplo:

Ejemplo de variable local 1

```
def subrutina():
    a = 2
    print(a)
    return

a = 5
subrutina()
print(a)
```

```
2
5
```

Puede ver la ejecución paso a paso de este programa utilizando los iconos de avance y retroceso situados abajo a la derecha.

- Las variables **locales** sólo existen en la propia función y no son accesibles desde niveles superiores, como puede verse en el siguiente ejemplo:

Ejemplo de variable local 2

```
def subrutina():
    a = 2
    print(a)
    return

subrutina()
print(a)
```

```
2
Traceback (most recent call last):
  File "ejemplo.py", line 7, in
<module>
    print(a)
NameError: name 'a' is not defined
```

Puede ver la ejecución paso a paso de este programa utilizando los iconos de avance y retroceso situados abajo a la derecha.

- Si en el interior de una función **se asigna valor** a una variable que no se ha declarado como global o no local, esa variable es **local** a todos los efectos. Por ello el siguiente programa da error:

Ejemplo de variable local 3

```
def subrutina():
    print(a)
    a = 2
    print(a)
    return

a = 5
subrutina()
print(a)
```

```
Traceback (most recent call last):
  File "ejemplo.py", line 8, in
<module>
    subrutina()
  File "ejemplo.py", line 2, in
subrutina
    print(a)
UnboundLocalError: local variable 'a'
referenced before assignment
```

Puede ver la ejecución paso a paso de este programa utilizando los iconos de avance y retroceso situados abajo a la derecha.

Variables libres globales o no locales

Si a una variable **no se le asigna valor** en una función, Python la considera **libre** y busca su valor en los niveles superiores de esa función, empezando por el inmediatamente superior y continuando hasta el programa principal. Si a la variable se le asigna valor en algún nivel intermedio la variable se considera **no local** y si se le asigna en el programa principal la variable se considera **global**, como muestran los siguientes ejemplos:

- En el ejemplo siguiente, la variable libre "a" de la función subrutina() se considera global porque obtiene su valor del programa principal:

Ejemplo de variable libre global

```
def subrutina():
    print(a)
    return

a = 5
subrutina()
print(a)
```

```
5
5
```

Puede ver la ejecución paso a paso de este programa utilizando los iconos de avance y retroceso situados abajo a la derecha.

- En el ejemplo siguiente, la variable libre "a" de la función sub_subrutina() se considera no local porque obtiene su valor de una función intermedia:

Ejemplo de variable libre no local

```
def subrutina():
    def sub_subrutina():
        print(a)
        return

    a = 3
    sub_subrutina()
    print(a)
    return

a = 4
subrutina()
print(a)
```

```
3
3
4
```

Puede ver la ejecución paso a paso de este programa utilizando los iconos de avance y retroceso situados abajo a la derecha.

- Si a una variable que Python considera libre (porque no se le asigna valor en la función) tampoco se le asigna valor en niveles superiores, Python dará un mensaje de error, como muestra el programa siguiente:

Ejemplo de variable libre no definida

```
def subrutina():
    print(a)
    return

subrutina()
print(a)
```

```
Traceback (most recent call last):
  File "ejemplo.py", line 5, in
<module>
    subrutina()
  File "ejemplo.py", line 2, in
subrutina
    print(a)
NameError: global name 'a' is not
defined
```

Puede ver la ejecución paso a paso de este programa utilizando los iconos de avance y retroceso situados abajo a la derecha.

Variables declaradas global o nonlocal

Si queremos asignar valor a una variable en una subrutina, pero no queremos que Python la considere local, debemos declararla en la función como global o nonlocal, como muestran los ejemplos siguientes:

- En el ejemplo siguiente la variable se declara como global, para que su valor sea el del programa principal:

Ejemplo de variable declarada global

```
def subrutina():
    global a
    print(a)
    a = 1
    return

a = 5
subrutina()
print(a)
```

```
5
1
```

Puede ver la ejecución paso a paso de este programa utilizando los iconos de avance y retroceso situados abajo a la derecha.

- En el ejemplo siguiente la variable se declara como nonlocal, para que su valor sea el de la función intermedia:

Ejemplo de variable declarada nonlocal

```
def subrutina():
    def sub_subrutina():
        nonlocal a
        print(a)
        a = 1
        return

    a = 3
    sub_subrutina()
    print(a)
    return

a = 4
subrutina()
print(a)
```

```
3
1
4
```

Puede ver la ejecución paso a paso de este programa utilizando los iconos de avance y retroceso situados abajo a la derecha.

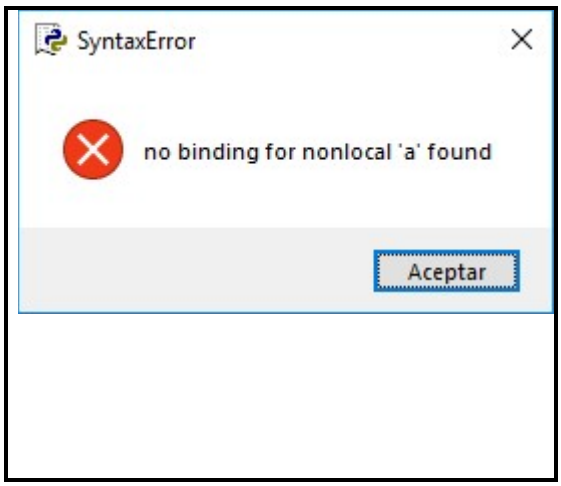
- Si a una variable declarada global o nonlocal en una función no se le asigna valor en el nivel superior correspondiente, Python dará un error de sintaxis, como muestra el programa siguiente:

Ejemplo de variable declarada nonlocal no definida

```
def subrutina():
    def sub_subrutina():
        nonlocal a
        print(a)
        a = 1
        return

    sub_subrutina()
    print(a)
    return

a = 4
subrutina()
print(a)
```



Puede ver la ejecución paso a paso de este programa utilizando los iconos de avance y retroceso situados abajo a la derecha.

Última modificación de esta página: 20 de marzo de 2016