Alcance de Nombres (namespaces)

Introducción

¿Qué es el *alcance de nombres*? escucho que preguntas. Bueno, no es algo muy sencillo de explicar. No por que sea especialmente complicado, sino porque cada lenguaje los maneja en una manera diferente. El concepto es realmente sencillo: el alcance de un nombre es el lugar o espacio dentro de un programa en el cual un nombre (variable, clase, etc.) es válido.

Este concepto surgió debido a que los primitivos lenguages de programación (como el BASIC) sólo tenían *Variables Globales*, es decir, variables que podían ser utilizadas en absolutamente cualquier sección del programa, inclusive dentro de funciones. Esto provocó que el mantenimiento de los programas largos fuera muy complicado, ya que resultaba posible que en un pequeño sector del programa una variable fuera modificada sin que las otras partes se dieran cuenta. Este efecto podía provocar serios problemas en la depuración de los programas. Para solucionar este problema, los lenguajes posteriores (incluido el BASIC moderno) incluyen el concepto de "alcance de nombres" (namespaces). El lenguaje C++ ha llevado esta idea al extremo al permitir al programador crear sus propios alcances dentro del mismo programa. Esto es muy útil para los creadores de bibliotecas ya que les permite mantener sus nombres de las funciones sin mezclarlos con los de otras bibliotecas provistas por otro programador.

El modelo de Python

En Python cada módulo crea su propio entorno de nombres. Para acceder a dichos nombres debemos precederlos con el nombre del módulo más un punto o importar explícitamente los nombres que deseamos utilizar. Hasta aquí nada nuevo: es lo que ya hemos hecho con los módulos `sys` y `string`. En cierto sentido, la definición de una clase también crea su propio entorno de nombres. Por esta razón, si deseamos acceder a un método o a una propiedad de la clase, primero debemos utilizar el nombre de la variable de instancia o el nombre de la clase.

En Python hay sólo tres entornos (o *alcances*):

1. Alcance local - nombres definidos dentro de una función o método
2. Alcande de módulo - nombres definidos dentro de un archivo
3. Alcance interno - nombres definidos dentro de Python (están siempre disponibles)

Hasta aquí todo muy lindo. Pero ¿qué sucede cuando dos variables que tienen distintos alcances llevan el mismo nombre, o cuando se hace referencia a un nombre que no está en el entorno actual? Veamos la primera situación: si una función hace referencia a una variable X y ya existe una variable X dentro de la función (alcance local) entonces Python utilizará dicha variable. Es deseable evitar los conflictos de nombres tales como que una variable local y una variable de módulo del mismo nombre sean requeridas en una misma función. En ese caso, la variable local tapará a la global.

En general, se debe minimizar el uso de variables globales: es mejor pasar la variable como parámetro y luego devolver la variable modificada.

En el segundo caso, cuando se hace referencia a un nombre que no existe en el entorno actual, Python lo resuelve de la siguiente manera: la función buscará en su entorno local; si no lo encuentra allí, lo busca en el entorno de módulo y si tampoco está allí, lo busca en el entorno interno de Python. El único problema con esto es cuando deseamos asignar un valor a la variable externa. Normalmente esto producirá la creación de una nueva variable local, algo que nosotros no deseamos, por lo cual debemos especificar que se trata de una variable global para evitar que sea incluida como local dentro de la función.

Veamos cómo funciona todo esto en un ejemplo:

```
# variables con alcance de módulo
W = 5
Y = 3

#los parámetros son como variables de la función
#por eso X tiene alcance local
def spam(X):

   #decirle a la función que busque en el nivel de módulo y que no cree su propia variable W
   global W

   Z = X*2 # crea nueva variable local Z
   W = X+5 # usa la variable W del módulo (global)

   if Z > W:
      # print pertenece al alcance interno de Python
      print "2 x X es mayor que X + 5"
      return Z
   else:
      return Y # no exite variable Y local, por lo que usa la variable Y del módulo
```

Cuando importamos un módulo como `sys` lo ponemos a disposión en forma local y entonces podemos acceder a los nombres dentro del módulo `sys`, calificándolos con el identificador y el punto. Pero si utilizamos:

```
from sys import exit
```

sólo importamos la función `exit` al entorno local y no podemos utilizar ningún otro nombre de `sys`, ni siquiera el mismo `sys`.

Y con BASIC también

BASIC ha tomado el camino inverso al de Python ya que convierte todas las variables por defecto en globales (por compatibilidad con versiones anteriores de BASIC) pero permite al programador crear sus propias variables LOCALES dentro de las funciones.

Tcl

Por lo que sé, no hay manera de controlar los alcances de las variables en Tcl, quizás por la extraña manera en que Tcl interpreta los programas. En todo caso parece que todas las variables son locales en sus inmediaciones. Las variables a nivel de archivo son visibles para los comandos dentro del propio archivo y las variables de procedimiento son sólo visibles dentro del procedimiento. Para comunicar dos entornos distintos, es necesario pasar los valores por medio de parámetros.

Anterior Próxima Contenido

Si tenés sugerencias o dudas podés enviar un email en inglés a: alan.gauld@btinternet.com o en español a: manilio@zoommail.com