Bucles, o el arte de repetirse una y otra vez

¿De qué nos ocuparemos?
Cómo utilizar bucles para evitar el retipeo. Diferentes tipos de bucles y cuándo utilizar cada uno de ellos.

En el último ejercicio imprimimos una parte de la tabla del doce, pero tuvimos que escribir bastante. Incluso si quisiéramos extenderla, nos llevaría más tiempo y esfuerzo. Afortunadamente tenemos una mejor manera de realizar esta tarea, lo que a su vez nos permite conocer el verdadero poder que un lenguaje de programación nos brinda.

Bucles FOR

Haremos que el lenguaje de programción efectúe todas las repeticiones, sustituyendo una variable que se incrementa con cada repetición. En Python deberemos usar:

```
>>>for i in range(1,13):
...     print "%d x 12 = %d" % (i, i*12)
```

Nota 1: Necesitamos que `range(1,13)` especifique el valor 13 ya que `range()` genera una serie de valores a partir del primer número y hasta el segundo, pero sin incluirlo. Esto puede parecer un tanto extraño, pero obedece a una razón y uno pronto se acostumbra.

Nota 2: El operador `for` en Python es en realidad un operador *foreach* ya que aplica la secuencia de código siguiente a cada miembro de una colección. En este caso, la colección es la lista de números generados por `range()`. Esto puede comprobarse si tipeas `print range(1,13)` en la línea de comando de Python y te fijás qué aparece en la pantalla.

Pero, ¿cómo funciona el programa? Vayamos paso por paso.

En primer lugar, Python usa la función `range` para crear una lista de números del 1 al 12.

Luego Python hace que `i` sea igual al primer valor de la lista, en este caso, 1. Luego ejecuta las instrucciones tabuladas con una sangría utilizando el valor de `i = 1`:

```
   print "%d x 12 = %d" % (1, 1*12)
```

Ahora Python vuelve a la línea del `for` y asigna a la variable `i` el próximo valor en la lista, esta vez equivalente a 2. Nuevamente ejecuta el código sangrado, pero ahora con el valor de `i = 2`:

```
   print "%d x 12 = %d" % (2, 2*12)
```

Repite esta secuencia hasta que ha asignado todos los valores de la lista a la variable `i`. Cuando esto se produce, continúa con el siguiente comando que está en una línea **no** sangrada - en este caso no hay ninguna instrucción por lo cual el programa finaliza.

Veamos el mismo bucle en BASIC:

```
FOR I = 1 to 12
    PRINT I, " x 12 = ", I*12
NEXT I
```

Esto es mucho más explícito y sencillo de entender. Sin embargo, la versión de Python es más flexible ya que nos permite iterar sobre una serie de números, items en una lista o incluso caracteres en una cadena.

Y en Tcl:

Tcl usa una construcción del bucle `for` que es muy común en varios lenguajes, modelada sobre C. Su aspecto es el siguiente:

```
for {set i 1} {$i <= 12} {incr i} {
    puts [format "%d x 12 = %d" $i [expr $i*12]]
    }
```

Nota: La construcción tiene tres partes:

- una *inicialización*: `{set i 1}` que se ejecuta una sola vez al principio de todo el bloque,
- una *comprobación*: `{$i <= 12}` que se ejecuta previa a cada iteración, y
- el *incremento*: `{incr i}` que se ejecuta después de cada iteración.

El *bloque del bucle* solamente se ejecuta si la comprobación es verdadera. Cada una de estas partes puede contener código arbitrariamente, pero la comprobación debe evaluarse mediante un valor booleano (lo cual en Tcl implica cero o no cero).

Tcl también posee una construcción `foreach` que puede aplicarse a listas.

Bucles WHILE

Los bucles `FOR` no son el único tipo de iteraciones de que disponen los lenguajes. Y esto es muy útil, ya que para utilizar un bucle `FOR` necesitamos saber de antemano o al menos poder calcular el número de repeticiones que queremos realizar. ¿Qué ocurre cuando deseamos que se repita una tarea específica hasta que algo suceda, sin saber exactamente cuándo se producirá este hecho? Por ejemplo, podríamos querer procesar una serie de datos de un archivo, pero no sabemos de antemano qué cantidad de datos contiene el archivo.

Simplemente deseamos seguir procesando los datos hasta que lleguemos al final del archivo. Esto es posible con un bucle FOR, pero a la vez es bastante complicado.

Para resolver este problema disponemos de otro tipo de bucle: el *WHILE*. En BASIC:

```
J = 1
WHILE J <= 12
    PRINT J, " x 12 = ", J*12
    J = J + 1
WEND
```

Esto produce el mismo resultado que antes pero utiliza un bucle `while` en lugar de un `for`. Notá que la estructura está formada por el `while` seguido por una *expresión* que se evalúa como verdadera o falsa (¿recordás?). Si la expresión es verdadera, se ejecuta el código enmarcado dentro del bucle.

Examinaremos como alternativa la versión de Tcl:

```
set j  1
while {$j <= 12} {
   puts [format "%d x 12 = %s"  $j [expr $j*12]]
   set j [expr $j + 1]
}
```

Como te darás cuenta, la estructura es muy similar, a excepción de las llaves que están en lugar del WEND que se utiliza en BASIC. ¿Pero qué es ese lío dentro del bucle? ¿Te acordás de las cadenas de formato en Python? Bueno, `format` es el equivalente en Tcl. El signo `$j` significa el valor de j (y no la letra 'j'!) y `expr` expresa "calcular el próximo bit como una expresión". Los corchetes le indican a Tcl qué bits debe procesar primero. Tcl es una lenguaje inusual en tanto que intenta interpretar el código en un solo movimiento, por lo cual sin los corchetes imprimiría la palabra "expr" y al encontrarse con una serie de valores no sabría qué hacer y daría un mensaje de error. Por esta razón debemos aclararle que deseamos realizar una suma, luego formatear la cadena y por último imprimir el resultado. ¿Confundido? No te preocupes. Como dije antes, Tcl es un lenguaje muy extaño pero con unos cuantos puntos a favor.

Veamos qué ocurre con Python:

```
>>> j = 1
>>> while j <= 12:
...     print "%d x 12 = %d" % (j, j*12)
...     j = j + 1
```

Llegados a este punto, el código debería resultarte lo suficientemente claro y comprensible. Sólo quiero remarcar una cosa: ¿notaste los dos puntos (:) al final del comando `while` y del `for` unas líneas más arriba? Esta convención le indica a Python que a la instrucción le sigue una porción de código (un *bloque*). La mayor parte de los lenguajes presentan un marcador de fin de bloque (como el WEND de BASIC o las llaves en Tcl) pero Python usa el sangrado para indicar la estructura. Esto significa que es muy importante sangrar todas las líneas con el tabulador dentro de un bucle. De todos modos, esta es una buena práctica ya que facilita la lectura del programa.

Bucles más flexibles

Volvamos a nuestra tabla del doce al comienzo de esta sección. El bucle que utilizamos allí funciona bien para imprimir dicha tabla, pero ¿qué pasa si deseamos utilizar otros valores? ¿Podemos modificar el bucle para que realice la tabla del siete? Para lograr esto debemos escribir:

```
>>> for j in range(1,13):
...     print "%d x 7 = %d" % (j,j*7)
```

Esto significa que debemos cambiar el 12 por el 7 en dos lugares. Y con cada valor nuevo que querramos introducir deberemos cambiar el programa. Pero, ¿no sería mejor ingresar el multiplicador deseado? Esta es una mejor solución: reemplazamos el valor en la cadena del `print` por una variable a la que asignaremos el valor deseado antes de comenzar el bucle:

```
>>> multiplicador = 12
>>> for j in range(1,13):
...     print "%d x %d = %d" % (j, multiplicador, j*multiplicador)
```

Este es nuestra vieja amiga, la tabla del doce. Ahora, si queremos producir la tabla del siete, sólo tenemos que asignar el valor 7 a la variable "multiplicador".

Fijate que aquí combinamos una secuencia de comandos con un bucle. Primero tenemos un comando, `multiplicador = 12` seguido *secuencialmente* por un bucle `for`.

Bucles con bucles

Llevemos nuestro pequeño programa un paso más adelante. Suponé que deseamos imprimir todas las tablas de multiplicación desde el 2 al 12 (la del uno es muy fácil, así que ni nos preocupamos por ella). Lo que debemos hacer entonces es colocar la variable multiplicador como parte del bucle:

```
>>> for multiplicador in range(2,13):
...     for j in range(1,13):
...         print "%d x %d = %d" % (j,multiplicador,j*multiplcador)
```

Fijate que la sección sangrada dentro del primer bucle es exactamente el bucle que estuvimos utilizando con anterioridad. Esto funciona así:
Asignamos a multiplicador el primer valor (2) y entramos en el segundo bucle.
Luego asignamos el próximo valor (3) a la variable multiplicador y entra otra vez en el bucle interno,
y así siguiendo. Esta técnica se conoce como bucles *anidados*.

Un pequeño problema es que las tablas se juntan y quedan un tanto desprolijas. Podemos solucionar esto colocando una línea de separación después del primer bucle:

```
>>> for multiplicador in range(2,13):
...     for j in range(1,13):
...         print "%d x %d = %d" % (j,multiplicador,j*multiplicador)
...     print "-------------------- "
```

Fijate que el segundo print se alinea con el segundo for, en realidad es la segunda instrucción del bucle secuencial. Recordá que el sangrado y las tabulaciones son muy importantes en Python.

Tarea para el hogar: ¿cómo harías para que en el separador se indique cuál es la tabla que continúa? Una ayuda: necesitarás usar la variable multiplicador y una cadena de formato.

Otros bucles

Algunos lenguajes presentan otros tipos de bucles, pero en general se basan en una mezcla de `for` y `while`. (Modula 2 y Oberon sólo proveen de un bucle `while` ya que pueden simular los bucles `for`, como vimos más arriba. Podés encontrar otros bucles como:

DO-WHILE
: Igual que while , pero la comprobación se realiza siempre al final del bucle, por lo que el bloque se ejecuta al menos una vez.

REPEAT-UNTIL
: Similar al anterior, pero la lógica de la comprobación se invierte.

GOTO, JUMP, LOOP etc
: Están presentes en los lenguajes más antiguos: colocan un marcador en el código hacia el cual saltan directamente.

Para recordar

- Los bucles `FOR` repiten un conjunto de comandos durante un número fijo de iteraciones.
- Los bucles `WHILE` repiten un bloque de comandos hasta que se produce una condición terminal. El código puede no ser ejecutado nunca si la condición terminal se evalúa como falsa.
- Existen otros tipos de bucles, pero casi siempre vamos a encontrar los bucles `FOR` y `WHILE` en prácticamente todos los lenguajes.
- Los bucles `FOR` en Python son en realidad bucles `FOREACH` ya que operan sobre una lista de items.
- Los bucles pueden anidarse unos dentro de otros.

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