CONTROL DE FLUJO

En esta lección vamos a ver los condicionales y los bucles.

Sentencias condicionales

Si un programa no fuera más que una lista de órdenes a ejecutar de forma secuencial, una por una, no tendría mucha utilidad. Los condicionales nos permiten comprobar condiciones y hacer que nuestro programa se comporte de una forma u otra, que ejecute un fragmento de código u otro, dependiendo de esta condición.

Aquí es donde cobran su importancia el tipo booleano y los operadores lógicos y relacionales que aprendimos en el capítulo sobre los tipos básicos de Python.

if

La forma más simple de un estamento condicional es un `if` (del inglés si) seguido de la condición a evaluar, dos puntos (`:`) y en la siguiente línea e indentado, el código a ejecutar en caso de que se cumpla dicha condición.

```
fav = "mundogeek.net"
# si (if) fav es igual a "mundogeek.net"
if fav == "mundogeek.net":
    print "Tienes buen gusto!"
    print "Gracias"
```

Como veis es bastante sencillo.

Eso si, aseguraros de que indentáis el código tal cual se ha hecho en el ejemplo, es decir, aseguraros de pulsar Tabulación antes de las dos órdenes `print`, dado que esta es la forma de Python de saber que vuestra intención es la de que los dos `print` se ejecuten sólo en el caso de que

se cumpla la condición, y no la de que se imprima la primera cadena si se cumple la condición y la otra siempre, cosa que se expresaría así:

```
if fav == "mundogeek.net":
    print "Tienes buen gusto!"
print "Gracias"
```

En otros lenguajes de programación los bloques de código se determinan encerrándolos entre llaves, y el indentarlos no se trata más que de una buena práctica para que sea más sencillo seguir el flujo del programa con un solo golpe de vista. Por ejemplo, el código anterior expresado en Java sería algo así:

```
String fav = "mundogeek.net";
if (fav.equals("mundogeek.net")){
    System.out.println("Tienes buen gusto!");
    System.out.println("Gracias");
}
```

Sin embargo, como ya hemos comentado, en Python se trata de una obligación, y no de una elección. De esta forma se obliga a los programadores a indentar su código para que sea más sencillo de leer :)

if … else

Vamos a ver ahora un condicional algo más complicado. ¿Qué haríamos si quisiéramos que se ejecutaran unas ciertas órdenes en el caso de que la condición no se cumpliera? Sin duda podríamos añadir otro if que tuviera como condición la negación del primero:

```
if fav == "mundogeek.net":
    print "Tienes buen gusto!"
    print "Gracias"

if fav != "mundogeek.net":
    print "Vaya, que lástima"
```

pero el condicional tiene una segunda construcción mucho más útil:

```
if fav == "mundogeek.net":
    print "Tienes buen gusto!"
    print "Gracias"
else:
    print "Vaya, que lástima"
```

Vemos que la segunda condición se puede sustituir con un `else` (del inglés: si no, en caso contrario). Si leemos el código vemos que tiene bastante sentido: "si fav es igual a mundogeek.net, imprime esto y esto, si no, imprime esto otro".

if … elif … elif … else

Todavía queda una construcción más que ver, que es la que hace uso del `elif`.

```
if numero < 0:
    print "Negativo"
elif numero > 0:
    print "Positivo"
else:
    print "Cero"
```

`elif` es una contracción de *else if*, por lo tanto `elif numero > 0` puede leerse como "si no, si numero es mayor que 0". Es decir, primero se evalúa la condición del `if`. Si es cierta, se ejecuta su código y se continúa ejecutando el código posterior al condicional; si no se cumple, se evalúa la condición del `elif`. Si se cumple la condición del `elif` se ejecuta su código y se continua ejecutando el código posterior al condicional; si no se cumple y hay más de un `elif` se continúa con el siguiente en orden de aparición. Si no se cumple la condición del `if` ni de ninguno de los `elif`, se ejecuta el código del `else`.

A if C else B

También existe una construcción similar al operador `?` de otros lenguajes, que no es más que una forma compacta de expresar un `if else`. En esta construcción se evalúa el predicado C y se devuelve A si se cumple o B si no se cumple: *A* `if` *C* `else` *B*. Veamos un ejemplo:

```
var = "par" if (num % 2 == 0) else "impar"
```

Y eso es todo. Si conocéis otros lenguajes de programación puede que esperarais que os hablara ahora del `switch`, pero en Python no existe esta construcción, que podría emularse con un simple diccionario, así que pasemos directamente a los bucles.

Bucles

Mientras que los condicionales nos permiten ejecutar distintos fragmentos de código dependiendo de ciertas condiciones, los bucles nos permiten ejecutar un mismo fragmento de código un cierto número de veces, mientras se cumpla una determinada condición.

while

El bucle `while` (mientras) ejecuta un fragmento de código mientras se cumpla una condición.

```
edad = 0
while edad < 18:
    edad = edad + 1
    print "Felicidades, tienes " + str(edad)
```

La variable `edad` comienza valiendo 0. Como la condición de que `edad` es menor que 18 es cierta (0 es menor que 18), se entra en el bucle. Se aumenta `edad` en 1 y se imprime el mensaje informando de que el usuario ha cumplido un año. Recordad que el operador + para las cadenas funciona concatenando ambas cadenas. Es necesario utilizar la función `str` (de *string*, cadena) para crear una cadena a partir del número, dado que no podemos concatenar números y cadenas, pero ya comentaremos esto y mucho más en próximos capítulos.

Ahora se vuelve a evaluar la condición, y 1 sigue siendo menor que 18, por lo que se vuelve a ejecutar el código que aumenta la edad en un año e imprime la edad en la pantalla. El bucle continuará ejecutándose hasta que `edad` sea igual a 18, momento en el cual la condición dejará de cumplirse y el programa continuaría ejecutando las instrucciones siguientes al bucle.

Ahora imaginemos que se nos olvidara escribir la instrucción que aumenta la edad. En ese caso nunca se llegaría a la condición de que `edad` fuese igual o mayor que 18, siempre sería 0, y el bucle continuaría indefinidamente escribiendo en pantalla `Has cumplido 0`.

Esto es lo que se conoce como un bucle infinito.

Sin embargo hay situaciones en las que un bucle infinito es útil. Por ejemplo, veamos un pequeño programa que repite todo lo que el usuario diga hasta que escriba `adios`.

```
while True:
    entrada = raw_input("> ")
    if entrada == "adios":
        break
    else:
        print entrada
```

Para obtener lo que el usuario escriba en pantalla utilizamos la función `raw_input`. No es necesario que sepais qué es una función ni cómo funciona exactamente, simplemente aceptad por ahora que en cada iteración del bucle la variable `entrada` contendrá lo que el usuario escribió hasta pulsar Enter.

Comprobamos entonces si lo que escribió el usuario fue `adios`, en cuyo caso se ejecuta la orden `break` o si era cualquier otra cosa, en cuyo caso se imprime en pantalla lo que el usuario escribió.

La palabra clave `break` (romper) sale del bucle en el que estamos.

Este bucle se podría haber escrito también, no obstante, de la siguiente forma:

```
salir = False
while not salir:
    entrada = raw_input()
    if entrada == "adios":
        salir = True
    else:
        print entrada
```

pero nos ha servido para ver cómo funciona `break`.

Otra palabra clave que nos podemos encontrar dentro de los bucles es `continue` (continuar). Como habréis adivinado no hace otra cosa que pasar directamente a la siguiente iteración del bucle.

```
edad = 0
while edad < 18:
```

```
edad = edad + 1
if edad % 2 == 0:
    continue
print "Felicidades, tienes " + str(edad)
```

Como veis esta es una pequeña modificación de nuestro programa de felicitaciones. En esta ocasión hemos añadido un `if` que comprueba si la edad es par, en cuyo caso saltamos a la próxima iteración en lugar de imprimir el mensaje. Es decir, con esta modificación el programa sólo imprimiría felicitaciones cuando la edad fuera impar.

for ... in

A los que hayáis tenido experiencia previa con según que lenguajes este bucle os va a sorprender gratamente. En Python `for` se utiliza como una forma genérica de iterar sobre una secuencia. Y como tal intenta facilitar su uso para este fin.

Este es el aspecto de un bucle `for` en Python:

```
secuencia = ["uno", "dos", "tres"]
for elemento in secuencia:
    print elemento
```

Como hemos dicho los `for` se utilizan en Python para recorrer secuencias, por lo que vamos a utilizar un tipo secuencia, como es la lista, para nuestro ejemplo.

Leamos la cabecera del bucle como si de lenguaje natural se tratara: "para cada elemento en secuencia". Y esto es exactamente lo que hace el bucle: para cada elemento que tengamos en la secuencia, ejecuta estas líneas de código.

Lo que hace la cabecera del bucle es obtener el siguiente elemento de la secuencia `secuencia` y almacenarlo en una variable de nombre `elemento`. Por esta razón en la primera iteración del bucle `elemento` valdrá `"uno"`, en la segunda `"dos"`, y en la tercera `"tres"`.

Fácil y sencillo.

En C o C++, por ejemplo, lo que habríamos hecho sería iterar sobre las

posiciones, y no sobre los elementos:

```
int mi_array[] = {1, 2, 3, 4, 5};
int i;
for(i = 0; i < 5; i++) {
    printf("%d\n", mi_array[i]);
}
```

Es decir, tendríamos un bucle `for` que fuera aumentando una variable `i` en cada iteración, desde 0 al tamaño de la secuencia, y utilizaríamos esta variable a modo de índice para obtener cada elemento e imprimirlo.

Como veis el enfoque de Python es más natural e intuitivo.

Pero, ¿qué ocurre si quisiéramos utilizar el `for` como si estuviéramos en C o en Java, por ejemplo, para imprimir los números de 30 a 50? No os preocupéis, porque no necesitaríais crear una lista y añadir uno a uno los números del 30 al 50. Python proporciona una función llamada `range` (rango) que permite generar una lista que vaya desde el primer número que le indiquemos al segundo. Lo veremos después de ver al fin a qué se refiere ese término tan recurrente: las funciones.