Capítulo 3
Pruebas y lógica buleana

1. Las pruebas y condiciones

1.1 Principio

En el anterior capítulo ha podido familiarizarse con las expresiones utilizando los operadores, tanto de cálculo, como de comparación (igualdad por ejemplo) o buleanos. Estos operadores y expresiones tienen todo su sentido cuando se usan en las condiciones (que llamamos también uniones condicionales). Una expresión evaluada es verdadera (el resultado es diferente de cero) o falsa. Dependiendo de este resultado, el algoritmo va a realizar una acción u otra. Es el principio de condición.

Gracias a los operadores buleanos, la expresión puede ser compuesta: varias expresiones están relacionadas entre ellas con ayuda de un operador buleano, eventualmente agrupadas con los paréntesis, para modificar la prioridad.

```
(a=1 O (b*3=6)) Y c>10
```

Es una expresión totalmente válida. Esta será verdadera si cada uno de sus componentes respeta las condiciones impuestas. Esta expresión es verdadera si a vale 1 y c es superior a 10 o si b vale 2 (2*3=6) y c es superior a 10.

Retome el algoritmo del capítulo anterior, que calculaba los dos resultados posibles de una ecuación de segundo grado. El enunciado simplificado decía que por razones prácticas, solo funcionará el caso en el que la ecuación tiene dos soluciones. Dicho de otra manera, el algoritmo no es falso en este supuesto, pero es incompleto. Faltan las condiciones para probar el valor del determinante: ¿es positivo, negativo o nulo? Y en estos casos, ¿qué hacer y cómo hacerlo?

Imagine un segundo algoritmo que permite ir de un punto A a un punto B. En realidad no va a hacerlo aquí, porque es muy complejo en una gran red de carreteras. Hay numerosos sitios en Internet que le permiten establecer un trayecto con sus indicaciones. Es el resultado lo que es interesante. Las indicaciones son sencillas: continuar todo recto, girar a la derecha en el próximo cruce, recorra tres kilómetros y en la rotonda, tome la tercera salida dirección a B. En la mayor parte de los casos, si sigue este trayecto llegará a buen puerto. Pero, ¿qué sucede con las situaciones inesperadas? ¿Por donde va a ir si la carretera de la derecha en el próximo cruce se ha convertido en vía de sentido prohibido (algunas veces sucede, también con un GPS, tenga cuidado) o hay obras que impiden tomar la tercera salida en la rotonda?

Retome el trayecto: vaya todo recto. Si en el próximo cruce la carretera de la derecha es de sentido prohibido: continúe recto y después gire a la derecha en el siguiente cruce y después a la izquierda en dos kilómetros hasta la rotonda. En caso contrario: gire a la derecha y recorra tres kilómetros hasta la rotonda. En la rotonda, si la salida hacia B está libre, tome esta salida. En caso contrario, tome la salida hacia C y después trescientos metros más adelante, gire a la derecha hacia B.

Este pequeño recorrido no solo pone de manifiesto la complejidad de un trayecto en caso de desvíos, sino también las numerosas condiciones que permiten establecer un trayecto en caso de problema. Si tiene algún navegador por GPS estos cuentan con posibilidades de itinerarios Bis, recorridos que evitan una sección concreta e incluso peajes. ¿Puede ahora imaginar el número de expresiones a evaluar en todos estos supuestos, además de la velocidad de cada carretera para optimizar la hora de llegada?

1.2 ¿Qué probar?

Los operadores se aplican sobre casi todos los tipos de datos, incluidas las cadenas de caracteres, al menos en pseudo-código algorítmico (normalmente será necesario utilizar instrucciones especiales del lenguaje de programación para comparar cadenas de caracteres). Por lo tanto, puede probar prácticamente todo. Probar incluye evaluar una expresión que es una condición. Por lo tanto, una condición es el acto de realizar pruebas para que, en función del resultado de estas, realizar unas acciones u otras.

Por lo tanto, una condición es una afirmación: el algoritmo y el programa determinan si es verdadera o falsa.

Una condición que devuelve VERDADERO o FALSO, tiene como resultado un **buleano**.

Una condición normalmente es una comparación. Como recordatorio, una comparación es una expresión compuesta de tres elementos:

– un primer valor: variable o escalar.
– un operador de comparación.
– un segundo valor: variable o escalar.

Los operadores de comparación son:

– La igualdad: **=**
– La diferencia: **!=** o **<>**
– Inferior: **<**
– Inferior o igual: **<=**
– Superior: **>**
– Superior o igual: **>=**

También es posible que la condición sea una expresión unaria: un valor con o sin un operador. Una variable pueda ser verdadera o falsa, por lo tanto puede ser suficiente con evaluar la condición dada.

El pseudo-código algorítmico no prohíbe comparar cadenas de caracteres. Usted se encargará de comparar solo las variables de tipos compatibles. En una condición, independientemente del resultado de una expresión, esta siempre será evaluada como verdadera o falsa.

Observación

En una condición, también se puede utilizar el operador de asignación. En este caso, si asigna 0 a una variable la expresión será falsa y si asigna cualquier otro valor, será verdadera.

En lenguaje normal, es posible que diga "seleccione un número entre 1 y 10". En matemáticas, escribe esto como sigue:

```
1 ≤ numero ≤ 10
```

Si escribe esto en su algoritmo, espere resultados sorprendentes cuando transforme su programa en algo real. En efecto, como sabe, los operadores de comparación tienen una prioridad, pero la expresión que componen también se evalúa normalmente de izquierda a derecha. Si la variable `numero` contiene el valor 15, a continuación se muestra qué sucede:

– Se evalúa la expresión 1 ≤ 15: es verdadera.
– ¿Y ahora? Todo va a depender del lenguaje de programación utilizado, pero para la mayor parte de ellos la expresión verdadero ≤ 10 es verdadera: "verdadero" es aquí el resultado de la expresión 1 ≤ 15. Verdadero vale normalmente 1. Por lo tanto, 1 ≤ 10 es verdadera, lo que no es el resultado esperado.
– El resultado es terrible: el programa considera la expresión como verdadera y el código correspondiente se va a ejecutar.

Probablemente esto no es lo que esperaba. Por lo tanto, debe evitar esta forma de expresión.

A continuación se muestra lo que conviene en este caso:

```
numero>=1 Y numero<=10
```

O incluso

```
1<=numero Y numero<=10
```

1.3 Pruebas SI

1.3.1 Forma sencilla

En algoritmia hay una única instrucción de prueba: "**Si**", que sin embargo tiene dos formas: una sencilla y una compleja. La prueba SI permite ejecutar un bloque de instrucciones si la condición (la o las expresiones que la componen), es verdadera. La forma sencilla es la siguiente:

```
Si buleano Entonces
  Bloque de instrucciones
FinSi
```

Observe aquí que el buleano es la condición (o expresión). Como se ha indicado anteriormente, la condición también se puede representar con una única variable. Si contiene 0, representa el buleano FALSO, en caso contrario el buleano VERDADERO.

¿Qué sucede si la condición es verdadera? Se ejecuta el bloque de instrucciones situado después de "**Entonces**". Su tamaño (el número de instrucciones) no tiene ninguna importancia: desde una hasta n líneas, sin límite. En caso contrario, el programa continúa por la siguiente instrucción, es decir el "**FinSi**". El siguiente ejemplo muestra cómo obtener el valor absoluto de un número con este método.

```
PROGRAMA ABS
VAR
  numero:entero
INICIO
  numero←-15
  Si numero<0 Entonces
    numero←- numero
  FinSi
  Mostrar numero
FIN
```

En Python, se debe utilizar "if" con la expresión buleana entre paréntesis. La sintaxis es la siguiente:

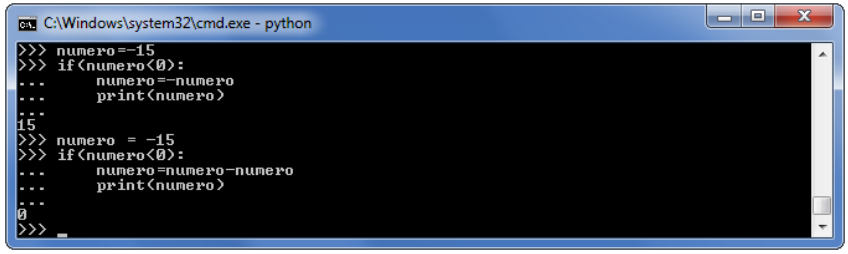
```
if(boolean):
/*código */
```

Una noción importante aquí es que Python, al contrario que muchos lenguajes tales que C o Java, no utiliza los llaves para determinar un bloque de instrucción sino la indentación.

Si observa el código anterior, tenemos la prueba if con la condición que termina por ":".

La siguiente línea está indentada, es decir, hemos pulsado la tecla [Tabulación] antes de empezar la línea. Todo el código posterior que está tabulado, pertenecerá al if.

```
    numero=-15
    if(numero <0):
       numero=- numero
       print(numero)
```



Una condición con un valor buleano, se hace de la misma manera. El siguiente código solo tiene una función pedagógica: implementar un flag que permita indicar si se ha comprobado la condición. Si el flag es verdadero, entonces el número es negativo. A continuación se prueba el flag para mostrar el resultado opuesto.

```
PROGRAMA ABS
VAR
  numero:entero

  flag:buleano
INICIO

  flag←FALSO
 numero←-15
  Si numero<0 Entonces
    flag←VERDADERO
  FinSi
```