Programación 1

Tecnicatura Universitaria en Inteligencia Artificial

2022

Apunte 3

1. Estructuras de Control

De no ser por las estructuras de control, el código en cualquier lenguaje de programación sería ejecutado secuencialmente hasta terminar. Un código, no deja de ser un conjunto de instrucciones que son ejecutadas unas tras otra. Gracias a las estructuras de control, podemos cambiar el flujo de ejecución de un programa, haciendo que ciertos bloques de código se ejecuten sí y sólo sí si se dan unas condiciones particulares. Vamos a introducir la primera estructura de control, las decisiones.

2. Decisiones

Vamos a introducir la idea de decisiones a través de problemas que iremos complejizando.

Problema 1

Debemos leer un número y, si el número es positivo, debemos escribir en pantalla el cartel "Número positivo".

Solución. Especificamos nuestra solución: se deberá leer un número x. Si x >0 se escribe el mensaje `"Número positivo"`.

Diseñamos nuestra solución:

1. Solicitar al usuario un número, guardarlo en `x`.
2. Si x >0, imprimir `"Número positivo"`

Es claro que la primera línea se puede traducir como

```
x = int(input("Ingrese un número: "))
```

Sin embargo, con las instrucciones que vimos hasta ahora no podemos tomar el tipo de decisiones que nos planteamos en la segunda línea de este diseño.

Para resolver este problema introducimos una nueva instrucción que llamaremos **condicional** y tiene la siguiente forma:

```
if <condición>:
    <hacer algo si se da la condición>
```

Donde `if` es una palabra reservada.

¿Qué es la condición que aparece luego de la palabra reservada `if`? La condición debe ser una expresión lógica o booleana. Una expresión booleana es toda expresión cuyo resultado devuelve `True` o `False`. Para recordar este tema puede releer el **Apunte 2**, donde se presentan los tipos de datos y los operadores que nos permiten construir estas expresiones.

2.1. Comparaciones simples

La instrucción `if` que acabamos de introducir, tiene entonces el siguiente significado: se evalúa `<condición>` (que debe ser una expresión lógica), y si el resultado es `True` (verdadero) se ejecutan las acciones indicadas como `<hacer algo si se da la condición>`.

Podríamos entonces, completar la solución al **Problema 1** con el siguiente código:

```
x = int(input("Ingrese un número: "))
if x > 0:
    print("Número positivo")
```

Veremos ahora una modificación al problema planteado.

Problema 2

Necesitamos además un mensaje `"Número no positivo"` cuando no se cumple la condición.

Modificamos la especificación consistentemente y modificamos el diseño:

1. Solicitar al usuario un número, guardarlo en `x`.
2. Si `x >0`, imprimir `"Número positivo"`
3. En caso contrario, imprimir `"Número no positivo"`

La negación de $x > 0$ es : $\neg(x > 0)$ que se traduce en Python como `not x >0`, por lo que implementamos nuestra solución en Python como:

```
x = int(input("Ingrese un número: "))
if x > 0:
    print("Número positivo")
if not x > 0:
    print("Número no positivo")
```

Sin embargo hay algo que nos preocupa: si ya averiguamos una vez, en la segunda línea del cuerpo, si `x >0`, ¿Es realmente necesario volver a preguntarlo en la cuarta?. Existe una construcción alternativa para la estructura de decisión:

Si se da la condición C, hacer S, de lo contrario, hacer T. Esta estructura tiene la forma:

```
if <condición >:
    <hacer algo si se da la condición>
else:
    <hacer otra cosa si no se da la condición>
```

Donde `if` y `else` son palabras reservadas.

Su significado es el siguiente: se evalúa `<condición>`, si el resultado es `True` se ejecutan las acciones indicadas como `<hacer algo si se da la condición>`, y si el resultado es `False` se ejecutan las acciones indicadas como `<hacer otra cosa si no se da la condición>`.

Volvemos a nuestro diseño:

1. Solicitar al usuario un número, guardarlo en `x`.
2. Si `x >0`, imprimir `"Número positivo"`
3. En caso contrario, imprimir `"Número no positivo"`

Este diseño se implementa como:

```
x = int(input("Ingrese un número: "))
if x > 0:
    print("Número positivo")
else:
    print("Número no positivo")
```

Es importante destacar que, en general, negar la condición del `if` y poner else no son intercambiables, porque no necesariamente producen el mismo efecto en el programa. Notar qué sucede en los dos programas que se transcriben a continuación.

```
x = int(input("Ingrese un nro: "))
if x > 0:
    print("Número positivo")
    x = -x
if x < 0:
    print("Número no positivo")
```

```
x = int(input("Ingrese un nro: "))
if x > 0:
    print("Número positivo")
    x = -x
else:
    print("Número no positivo")
```

¿Producen los mismos resultados? Pruebe ambos programas con una entrada positiva y analice los resultados.

2.2. Múltiples decisiones consecutivas

La decisión de incluir una decisión en un programa, parte de una lectura cuidadosa de la especificación. En nuestro caso la especificación nos decía:

Si el número es positivo escribir un mensaje `"Número positivo"`, de lo contrario escribir un mensaje `"Número no positivo"`.

Veamos qué se puede hacer cuando se presentan tres o más alternativas:

Problema 3

Si el número es positivo escribir un mensaje `"Número positivo"`, si el número es igual a 0 un mensaje `Ïgual a 0"`, y si el número es negativo escribir un mensaje `"Número negativo"`.

Una posibilidad es considerar que se trata de una estructura con dos casos como antes, sólo que el segundo caso es complejo (es nuevamente una alternativa):

1. Solicitar al usuario un número, guardarlo en x.
2. Si x >0, imprimir "Número positivo"
3. De lo contrario:
   a) Si x = 0, imprimir Ïgual a 0"
   b) De lo contrario, imprimir "Número no positivo"

Este diseño se implementa como:

```
x = int(input("Ingrese un número: "))
if x > 0:
    print("Número positivo")
else:
    if x == 0:
        print("Igual a 0")
    else:
        print("Número negativo")
```

Esta estructura se conoce como de *alternativas anidadas* ya que dentro de una de las ramas de la alternativa (en este caso la rama del `else`) se anida otra alternativa.

Pero ésta no es la única forma de implementarlo. Existe otra construcción, equivalente a la anterior pero que no exige sangrías cada vez mayores en el texto. Se trata de la estructura de *alternativas encadenadas*, que tiene la forma

```
if <condición_1 >:
    <hacer algo_1 si se da la condición_1 >
elif <condición_2 >:
    <hacer algo_2 si se da la condición_2 >
...
...
elif <condición_n >:
    <hacer algo_n si se da la condición_n >
else:
    <hacer otra cosa si no se da ninguna
    de las condiciones anteriores>
```

Donde `if`, `elif` y `else` son palabras reservadas.

En nuestro ejemplo:

```
x = int(input("Ingrese un número: "))
if x > 0:
    print("Número positivo")
elif x == 0:
    print("Igual a 0")
else:
    print("Número negativo")
```

Se evalúa la primera alternativa, si es verdadera se ejecuta su cuerpo. De lo contrario se evalúa la segunda alternativa, si es verdadera se ejecuta su cuerpo, etc. Finalmente, si todas las alternativas anteriores fallaron, se ejecuta el cuerpo del else.

Si vienes de otros lenguajes de programación, sabrás que el `switch` es una forma alternativa de `elif`, sin embargo en Python esta cláusula no existe.

2.3. Operador ternario

El operador ternario o `ternary operator` es una herramienta muy potente que muchos lenguajes de programación tienen. En Python es un poco distinto a lo que sería en C, pero el concepto es el mismo. Se trata de una cláusula `if`, `else` que se define en una sola línea y puede ser usado por ejemplo, dentro de un `print()`.

```
x = 5
print("Es 5" if x == 5 else "No es 5") #Es 5
```

Existen tres partes en un operador ternario, que son exactamente iguales a los que había en un `if else`. Tenemos la condición a evaluar, el código que se ejecuta si se cumple, y el código que se ejecuta si no se cumple. En este caso, tenemos los tres en la misma línea.

```
# [código si se cumple] if [condición] else [código si no se cumple]
```

Es muy útil y permite ahorrarse algunas líneas de código, además de aumentar la rapidez a la que escribimos. Si por ejemplo tenemos una variable a la que queremos asignar un valor en función de una condición, se puede hacer de la siguiente manera. Siguiendo el ejemplo anterior, en el siguiente código intentamos dividir `a` entre `b`. Si `b` es diferente a cero, se realiza la división y se almacena en `c`, de lo contrario se almacena -1. Ese -1 podría ser una forma de indicar que ha habido un error con la división.

```
a = 10
b = 5
c = a/b if b!=0 else -1
print(c) #2
```

2.4. Ejemplos if

```
# Verifica si un número es par o impar
x = 6
if not x%2:
    print("Es par")
else:
    print("Es impar")
```

```
# Decrementa x en 1 unidad si es mayor que cero
x = 5
x-=1 if x>0 else x
print(x)
```

3. Bibliografía

- Apunte de la materia Algoritmos y Programación 1, primera materia de programación de las carreras de Ingeniería en Informática y Licenciatura en Análisis de Sistemas dela Facultad de Ingeniería de la UBA: http://materias.fi.uba.ar/7501/apunte%20PYTHON.pdf

- Libro digital: https://ellibrodepython.com/