Estructuras Condicionales y Subproblemas

Índice

1. Estructura condicional

Una instrucción condicional contiene una expresión lógica que puede ser evaluada por verdadera o por falsa, y dos bloques de instrucciones adicionales designados en general como la salida o rama verdadera y la salida o rama falsa. En Python, una instrucción condicional típica sería:

```
if expresión lógica :
        Instrucciones de la rama verdadera
else :
        Instrucciones de la rama falsa
```

La secuencia de instrucciones que conforma la rama verdadera se escribe en la siguiente línea, respetando el encolumnado: Python identifica a las instrucciones que pertenecen a un mismo bloque de acuerdo a su encolumnado.

Una vez terminado de escribir el bloque de la rama verdadera, en la siguiente línea se escribe la palabra reservada else (en la misma columna que if) seguida de dos puntos (:) y en la siguiente línea se escribe la rama falsa respetando también el encolumnado.

Cuando se termina de escribir la rama falsa, se retorna a la misma columna que if y else para continuar escribiendo las instrucciones del programa indicando que se ha terminado de escribir el bloque condicional. Esta parte del código siempre se ejecutará pues es independiente del resultado de evaluación lógica de la condición.

2. Expresiones lógicas

En todo lenguaje existen operadores cuya acción no implica la obtención de un numero como resultado, sino calores lógicos de la forma verdadero o falso y algunos otros operadores entregaran resultados de otros tipos. En ese sentido, una expresión lógica es una expresión cuyo resultado esperado es un calor de verdad (True o False).

Para el planteo de expresiones lógicas, todo lenguaje de programación provee operadores que implican la obtención de un valor de verdad como resultado. Los más elementales son los llamados operadores relacionales u operadores de comparación, siendo en Python los siguientes:

Operador	Significado	Ejemplo	Funcionamiento
==	Igual a	a == b	Retorna ***True*** si A es igual a B
!=	Distinto de	a != b	Retorna ***True*** si A es distinto de B
<	Menor que	a < b	Retorna ***True*** si A es menor que B
>	Mayor que	a > b	Retorna ***True*** si A es mayor que B
<=	Menor o igual a	a <= b	Retorna ***True*** si A es menor o igual a B
>=	Mayor o igual a	a >= b	Retorna ***True*** si A es mayor o igual a B

Tabla - Operadores Relacionales

a. Operadores relacionales con cadenas de caracteres

Los operadores relacionales también permiten comparar de forma directa dos cadenas de caracteres: los operadores **==** & **!=** harán lo que se espera, determinar si 2 cadenas son iguales o distintas.

Si se usan los operadores **<** | **<=** | **>** | **>=** para comparar cadenas, Python hará lo que se conoce como una comparación lexicográfica, intentando ordenar alfabéticamente un conjunto de palabras, como si fuese un diccionario. *Cad1* será considerado menor que *cad2* si *cad1* estuviese antes en el diccionario que *cad2* :

Ana < Sergio < Tomás < Zaira

Para conseguir estos resultados, Python compara el primer carácter de cada cadena, y si fuesen diferentes, tomará como menor la cadena que comience con el carácter ASCII que aparezca antes (A) hasta el último (z).

Sergio < Zaira < ana < tomás

Si ambas cadenas tuviesen el mismo primer carácter, Python tomará el segundo de ambas (*cad1[1]* y *cad2[1]*) aplicando la misma regla, y así, hasta completar la comparación entre cadenas.

Manuel < Marcos < María < macarena < mariano

b. Conectores lógicos

El programador necesitará hacer varias comprobaciones al mismo tiempo, para ello, la forma más simple de hacer estas comprobaciones múltiples consiste en aplicar los llamados conectores lógicos.

Un conector lógico u operador booleano es un operador que permite encadenar la comprobación de dos o más expresiones lógicas y obtener un resultado único. Cada una de las expresiones lógicas encadenadas por un conector lógico se designa como una proposición lógica. Los conectores lógicos en Python son los siguientes:

Operador	Significado	Ejemplo
and	Conjunción lógica Y	a == b ***and*** y != x
or	Disyunción lógica O	a == 1 ***or*** a == 2
not	Negación lógica NO	***not*** a > 3

Tabla - Conectores Lógicos

Se puede mostrar la forma en que operan los conectores lógicos ***and***, ***or*** y ***not*** mediante las tablas de la verdad:

p *and* q → La salida solo será por la rama verdadera cuando ambas condiciones se cumplan (*True*), en caso de que 1 condición sea *False*, la salida será por la rama falsa.

p	q	p and q
True	True	True
True	False	False
False	True	False
False	False	False

Tabla de la verdad del conector AND

p *or* q → La salida en este caso será por la rama verdadera siempre que 1 condición sea *True*, en caso de que ambas condiciones sean *False*, la salida será por la rama falsa.

p	q	p or q
True	True	True
True	False	True
False	True	True
False	False	False

Tabla de la verdad del conector OR

***not* p** → La negación lógica se aplica sobre una sola proposición y su efecto es obtener el valor opuesto al de la proposición negada. La salida será por la rama verdadera cuando el valor de p sea *False*, y, por el contrario, la salida será por la rama falsa cuando el valor de p sea *True*.

p	not p
True	False
False	True

Tabla de la verdad del conector NOT

c. Precedencia en ejecución

Todos los operadores relacionales tienen la misma precedencia, y a su vez, esta es mayor que la de los conectores lógicos, pero menor que la de los operadores matemáticos. Obviamente, el uso de paréntesis permite cambiar esas precedencias según sus necesidades.

De esta forma, en cualquier expresión lógica sencilla o compleja el lenguaje agrupará los términos y resolverá:

1. Operadores aritméticos (**+**, **-**, *****, **/**, **//**, **%**, ******)
2. Operadores de comparación (***==***, ***!=***, **<**, **<=**, **>**, **>=**)
3. Conectores lógicos (***and***, ***or***, ***not***)

Tanto el conector ***and*** como el ***or*** en Python actúan de forma cortocircuitada, es decir, dependiendo del valor de la primera condición evaluada, la segunda puede llegar o no a ser evaluada dependiendo del resultado de la primera. Procede a la explicación:

- Dado el ejemplo (***a* > *b* *and* *a* > *c***), si ***a*** no es mayor que ***b***, nunca llegará a ejecutarse la evaluación de ***a* > *c*** ; por el contrario, si ***a*** es mayor que ***b***, entonces, se pasará a ejecutar la evaluación de si ***a*** es mayor que ***c***. En caso de que ***a*** sea mayor en ambas evaluaciones, se obtiene un *True*, por lo que se ejecutará la rama verdadera; por el contrario, si ***a*** no es mayor en ambos casos, se obtiene un *False*, por lo que se ejecutará la rama falsa.

- Por el contrario, dado el ejemplo (***a > b or a > c***), si ***a*** es mayor que ***b***, pasará a ejecutarse la evaluación de ***a > c*** ; al igual que, si ***a*** es menor que ***b***, pasará a ejecutarse la evaluación de si ***a > c***. En caso de que ***a*** sea mayor en una de las evaluaciones, se obtiene un *True*, por lo que se ejecutará la rama verdadera; por el contrario, si ***a*** es menor en ambos casos, se obtiene un *False*, por lo que se ejecutará la rama falsa.

3. Generar valores aleatorios

Hacer que un programa en Python genere número al azar es de mucha utilidad en aplicaciones que se basan en el desarrollo probabilístico (como un juego de PC).

En muchos problemas, se requiere poder pedirle al programa que almacene en una o más variables algún número seleccionado al azar. Esto se conoce como generación de números aleatorios (o random), y todo lenguaje de programación provee un cierto número de funciones predefinidas que permiten hacer esto en forma aceptable.

Estas funciones no generan realmente números de forma aleatoria, sino que se basan en algún algoritmo, que, partiendo de un valor inicial, es capaz de generar una secuencia de aspecto aleatorio para cualquiera que no conozca el valor inicial. Esto se conoce como generación de números pseudo-aleatorios.

a. import random

En Python existe lo que se conoce como módulo (o librería) llamado random, que consiste en las definiciones y funciones necesarias para poder acceder a la gestión de números pseudo-aleatorios.

Para poder usar el contenido del módulo random en un programa, debe usarse la instrucción import random al inicio del script en donde se requiera. Con esto, el programa podrá acceder a las funciones contenidas en ese modulo invocando a esas funciones, pero precediendo la invocación con el prefijo “random”.

A modo de ejemplo, este código genera un número aleatorio entre 0 y 1 sin llegar nunca a valer 1, pero si 0.99…:

```
import random
x = random.random()
print(x)
```

b. random.randrange(a,b)

La función ***random.randrange(a,b)*** sirve para generar un número entero (*int*) de forma aleatoria perteneciente al rango (***a,b***), siendo el número máximo (***n=b-1***).

Esto quiere decir que, si por ejemplo se desea sacar un número entero aleatorio entre 1 y 25, inclusive el 25, la función debería ser:

```
import random
x = random.randrange(1,26)
print(x)
```

c. random.randint(a,b)

La función ***random.randint(a,b)*** sirve para generar un número entero (*int*) de forma aleatoria perteneciente al rango (***a,b***), siendo b el número máximo como posible salida por pantalla.

Esto quiere decir que, si por ejemplo se desea sacar un número entero aleatorio entre 1 y 25, inclusive el 25, la función debería ser:

```
import random
x = random.randint(1,25)
print(x)
```

d. random.choice(sec1)

La función ***random.choice(sec)*** acepta como parámetro una secuencia y retorna un elemento cualquiera de esa secuencia elegido al azar.

Se puede obtener al azar un número cualquiera, así como una letra cualquiera de una cadena, una palabra de una cadena de palabras, y así con cualquier secuencia.

Por lo tanto, la función para sacar aleatoriamente una letra de la cadena ***A*** y un número de la cadena ***B*** sería la siguiente:

```
import random
a = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
b = 1, 2, 25, 34, 10, 27, 69, 87
sec1 = random.choice(a)
print(sec1)
sec2 = random.choice(b)
print(sec2)
```

4. Variantes de la instrucción condicional

La instrucción condicional suele ser designada como una condición doble. En Python, y en otros lenguajes, es perfectamente válido escribir una instrucción condicional que solo tenga la rama verdadera, omitiendo por completo la falsa.

Una instrucción condicional de ese tipo suele definirse como condición simple, y en ella no se especifica la rama else, pues la instrucción condicional termina cuando termina la rama verdadera.

Por otra parte, es posible que en la rama falsa y/o en la rama verdadera de una instrucción condicional se requiera plantear otra instrucción condicional. Es posible que, a su vez, cada nueva instrucción condicional incluya otras, y así, sucesivamente según vaya necesitando el programador. Esto se conoce como anidamiento de condiciones.

En Python se puede plantear una variante para la instrucción condicional que permite evitar el anidamiento excesivo de condiciones. Se trata de la variante ***elif*** (***else + if***), que puede usarse en forma combinada con el if-else normal. Equivale a un else que contenga a otra condición, sin tener que trabajar tanto en la indentación de la estructura anidada y simplificando el código fuente.

Cada línea que contiene un elif, continua en forma directa con la expresión lógica que se quería evaluar en la salida falsa de la condición anterior, sin tener que volver a escribir la palabra if.

Un ejemplo de uso sería:

```
if n == 1 :
        print('Opción 1')
elif n == 2 :
        print('Opción 2')
else :
        print('Opción no válida')
```

Python *no dispone de instrucciones condicionales múltiples especiales* como las instrucciones *switch* o *case* de otros lenguajes, pero, el uso de ***elif*** *permite plantear una estructura de condiciones anidadas* que equivale por completo a una condición múltiple.

Un ejemplo de uso sería:

```
n == int(input('Ingrese una opción: '))
if n == 1 :
        print('Opción 1')
elif n == 2 :
        print('Opción 2')
else :
        print('Opción no válida')
```

5. Subproblemas

La mayoría de los problemas son de estructura compuesta que pueden ser divididos en Subproblemas cada vez menores en complejidad, hasta llegar a Subproblemas que no necesiten nuevas subdivisiones.

Un Subproblemas es un problema incluido dentro de otro de estructura más compleja,

como, por ejemplo, en un ejercicio de Física te dan 2 datos y necesitas aplicar una fórmula que pide 3, y para ello necesitas sacar el tercer dato aplicando una formula sobre los 2 datos anteriores, este proceso mediante el cual se obtiene el tercer dato necesario para aplicar la fórmula, es un subproblema.

Un principio básico en el planteo de algoritmos consiste en tratar de identificar los Subproblemas simples de un problema compuesto, considerando que cada subproblema simple es también un problema que admite datos, desarrolla procesos y genera resultados.

6. Ejercicios propuestos

Ejercicio 1: Crear un fichero llamado if_Max.py en el cual se pidan introducir 2 números por teclado, dando igual si son decimales o enteros, pero deben ser comparados mediante un bloque condicional IF para conocer cuál de los dos es mayor.

Ejercicio 2: Crear un fichero llamado compNum.py que pida introducir 2 números en coma flotante por teclado y compare si estos son iguales, distintos, y en caso de ser distintos, cual es mayor de ambos.

Ejercicio 3: Crear un fichero llamado ordNombres.py en el cual se pida introducir 2 nombres y los ordene alfabéticamente, mostrando el resultado por pantalla, a 1 nombre por línea.

Ejercicio 4: Crear un fichero llamado log_And.py en el cual pida introducir por teclado un sueldo mensual en coma flotante y la antigüedad en la empresa en años como decimal, redondeando esta cifra a número entero. Para conceder el préstamo, deberán evaluarse si el empleado cobra más de 700€ al mes y tiene una antigüedad de más de 3 años. En caso de cumplir ambas condiciones, se procederá a conceder el préstamo, en caso contrario, mostrar un mensaje que indique porque no se concede (los euros/mes restantes hasta la cifra mínima o los años necesarios de antigüedad en la empresa).

Ejercicio 5: Crear un fichero llamado if_Or.py en el cual se pida introducir por teclado el sueldo mensual en euros, con 2 decimales, y las horas trabajadas esa semana. En caso de que el sueldo mensual sea inferior a 950€ o que las horas semanales superen las 40, emitir un mensaje indicando 'Te están explotando, cambia de curro'; si por el contrario, el usuario cobra más de 950€ y las horas semanales son menos de 40, emitir un mensaje que diga 'Buen curro!'.

Ejercicio 6: Crear un fichero llamado if_Not.py en el cual se pida introducir la edad por teclado y este compruebe si se tienen más de 18 años para poder acceder a sacarse el carnet, emitir un mensaje que indique si es apto o no.

Ejercicio 7: Crear un fichero llamado r_Range.py en el cual se genere un número en el rango 1-100, pero siendo el límite máximo 99. Mostrar por pantalla

Ejercicio 8: Crear un fichero llamado r_Int.py en el cual se generé un número perteneciente al rango 1-57, inclusive el 57, como si fuese una bola del euromillones. Mostrar el número extraído.

Ejercicio 9: Crear un fichero llamado r_Choice.py en el cual, se imprima por pantalla 3 palabras que estarán almacenadas en una variable y que será escogida 1 aleatoriamente, el usuario deberá introducirá la cadena que creerá correcta y el programa comprobará si acertó o no.

Ejercicio 10: Crear un fichero llamado if_if.py en el cual se pidan 3 números enteros por teclado y muestra el mayor, el menor y la media de los 3. Usar if anidados. Necesita elif.

Ejercicio 11 Crear un fichero llamado list.py en el cual se pida introducir un número entero por teclado entre 1-3, dependiendo del número introducido, mostrar el valor en letras. Usar elif.

Ejercicio 12: Crear un fichero llamado cuaCub_men.py en el cual se pida introducir 2 números por teclado y evalúe cual es el menor de ambos, y encontrado el menor, mostrar su cuadrado y su cubo.

Ejercicio 13: Crear un fichero llamado cuMx_cbMn.py en el cual se pida introducir 2 números y se ordenen de menor a mayor. Mostrar por pantalla el cuadrado del número mayor, el cubo del número menor.

Ejercicio 14: Crear un fichero llamado difCuCb.py en el cual se pidan 2 números por teclado y se halle cual es el mayor y el menor. Calcular el cuadrado y el cubo en ambos casos, mostrar por pantalla los resultados y por último, mostrar la diferencia entre ambos cuadrados y cubos.

Ejercicio 15: Crear un fichero llamado op_Mat.py en el cual se pidan introducir 2 números por teclado y se halle cual es el mayor y cuál es el menor. Usar el mayor como dividendo, base de potencia y num1 en las operaciones de suma, resta y producto, división entera, división real, resto. El número menor será el divisor, el exponente y el num2 en el resto de operaciones.

Ejercicio 16: Crear un fichero llamado ordLista.py en el cual se pida introducir 3 nombres y los ordene alfabéticamente. Mostrar por pantalla el resultado con 1 nombre por línea.

Ejercicio 17: Crear un fichero llamado calNotas.py en el cual se pida introducir las notas correspondientes a cada uno de los 3 trimestres del curso. Deberán redondearse a 1 decimal las notas de cada trimestre y calcularse la media aritmética final. Mostrar un mensaje por pantalla correspondiente a la calificación obtenida. 0-2.5 (muy deficiente); 2.5-4.5 (suspenso, ponte las pilas); 4.5-5 (Suspenso, un pelín de esfuerzo y apruebas); 5-6 (Suficiente); 6-7(bien); 7-8.5 (Notable); 8.5-9.5 (Sobresaliente); y 9.5-10 (Excelente).

Ejercicio 18: Crear un fichero llamado areas.py en el cual se carguen 3 números por teclado y se evalué cual es mayor, menor y el del medio. Sacar el área del triángulo, de un cuadrado, de un rectángulo y de una circunferencia, siendo la base el número mayor, la altura y el lado del cuadrado el numero mediano, y el radio de la circunferencia el menor de los números. Mostrar por pantalla cada uno de los datos (may= X, med=Y, men=Z) y posteriormente, el resultado de las áreas.

-->Soluciones a los ejercicios propuestos en PDF