1000 PROGRAMADORES

Introducción a la
Programación en Python

MÓDULO 4

Sentencia if. Sentencia for. Sentencia while.
Expresiones anidadas.

Ministerio de Educación
Cultura, Ciencia y Tecnología
Gobierno de Salta

Ministerio de Economía
y Servicios Públicos
Gobierno de Salta

Universidad
Nacional de Salta

INTRODUCCIÓN

Al igual que otros lenguajes de programación, Python incorpora una serie de sentencias de control. Las alternativas o condicionales y las repetitivas o cíclicas. Veremos de qué se trata y cómo se usa cada una.

Los condicionales if, else, elif se utilizan para ejecutar una instrucción en caso de que una o más condiciones se cumplan. Un condicional es el momento en que se debe tomar una decisión en nuestro programa o script. Dependiendo la decisión que se tome ocurrirá una cosa u otra, o ninguna.
La comprensión de los condicionales es un elemento clave en la programación pues determinará que un programa sea dinámico y cambie según diferentes condiciones..

SENTENCIAS CONDICIONALES

SENTENCIA *IF*

La sentencia **`if`** se utiliza para ejecutar un bloque de código si, y sólo si, se cumple una determinada condición. La estructura básica es la siguiente:

```
if condición:
      bloque de código
```

Es decir, solo si **`condición`** se evalúa a **`True`**, se ejecutarán las sentencias que forman parte de **`bloque de código`**. En caso de que se evalúe a **`False`** no se ejecutará ninguna sentencia perteneciente a **`bloque de código`**.

`Condición` puede ser un literal, el valor de una variable, el resultado de una expresión o el valor devuelto por una función.

En las expresiones es muy común usar los operadores booleanos y de comparación.

IMPORTANTE: El cuerpo del bloque está indicado con una sangría o identación mayor. Dicho bloque termina cuando se encuentre la primera línea con un sangrado menor.
***Sangría**, **identación** y **sangrado** se usan como sinónimos y representa el espacio en blanco que dejamos desde el margen izquierdo al comenzar una sentencia.*

Por ejemplo:

```
x = 17
if x < 20:
      print('x es menor que 20')
```

```
x es menor que 20
```

---

salida

En el código anterior la variable x toma el valor 17. En la línea 2, la condición de la sentencia **`if`** evalúa si x es menor que 20. Como el valor devuelto es **`True`**, se ejecuta

el **bloque** del **if**, mostrando por pantalla la cadena x es menor que 20.

Veamos otros ejemplos:

```
valores = [1, 3, 4, 8]
if 5 in valores:
      print('está en valores')
print('fin')
```

```
fin
```

salida

```
valores = [1, 3, 4, 8]
if 3 in valores:
      print('está en valores')
print('fin')
```

```
está en valores
fin
```

salida

En ambos casos se define una lista de valores y comprobamos si en ella está un número determinado. En el primer caso el if comprueba si el número 5 se encuentra entre estos valores. Como la expresión devuelve como resultado False, porque 5 no está entre los valores simplemente mostrará por pantalla la cadena fin. En el segundo caso, comprobando si 3 se encuentra entre los valores observamos que se muestra ambas cadenas de texto.

SENTENCIA *IF-ELSE*

Hay ocasiones en que la sentencia **if** básica no es suficiente y es necesario ejecutar un conjunto de instrucciones o sentencias cuando la condición se evalúa a **False**.

Para ello se utiliza la estructura **if...else...** Esta es estructura es como sigue:

```
if condición:
   bloque de código (cuando condición es True)
```

```
else:
   bloque de código 2 (cuando condición es False)
```

Imagina un programa en el que necesitas realizar la división de dos números. Si divides un número entre 0, el programa lanzará un error y terminará. Para evitar esto, podemos añadir una sentencia **`if ... else...`** como en el ejemplo siguiente:

```
resultado = None
x = 10
y = 2
if y != 0:
   resultado = x / y
else:
   resultado = f'No se puede dividir {x} entre {y}'
print(resultado)
```

```
5.0
```

---

salida

Si ejecutamos el caso que quisimos salvar, se vería:

```
resultado = None
x = 10
y = 0
if y != 0:
   resultado = x / y
else:
   resultado = f'No se puede dividir {x} entre {y}'
print(resultado)
```

```
No se puede dividir 10 entre 0
```

---

salida

SENTENCIA *IF-ELIF-ELSE*

También es posible que haya situaciones en que una decisión dependa de más de una condición. En estos casos se usa una sentencia if compuesta, cuya estructura es como se indica a continuación:

```
if cond1:
   bloque cond1 (sentencias si se evalúa la cond1 a True)
elif cond2:
   bloque cond2 (sentencias si cond1 es False pero cond2 es True)
....
....        #podría haber más sentencias elif
else:
   bloque else (sentencias si cond1 y cond2 se evalúan a False)
```

Es decir, en esta ocasión, se comprueba la condición cond1. Si se evalúa a True, se ejecuta el bloque cond1 y luego el flujo continúa después del bloque if.

Luego, si cond1 se evalúa a False, se comprueba la condición cond2. Si ésta se evalúa a True, se ejecuta el bloque de sentencias bloque cond2. En caso contrario, pasaría a comprobar la condición del siguiente elif y así sucesivamente hasta que llegue al else, que se ejecuta si ninguna de las condiciones anteriores se evaluaron a True.

Veamos un ejemplo. Imagina que queremos comprobar si un número es menor que 0, igual a 0 o mayor que 0 y en cada situación debemos ejecutar un grupo de sentencias diferente. Por ejemplo:

```
x = 28
if x < 0:
   print(f'{x} es menor que 0')
elif x > 0:
   print(f'{x} es mayor que 0')
else:
   print('x es 0')
```

```
28 es mayor que 0
```

---

salida

En el ejemplo anterior x toma el valor 28. La primera condición de la sentencia if se

evalúa a False y pasa a la siguiente (a la del bloque elif). En esta ocasión, la expresión se evalúa a True, mostrándose por pantalla la cadena 28 es mayor que 0.

SENTENCIA *IF* ANIDADAS

A cualquiera de los bloques de sentencias anteriores se le puede volver a incluir una sentencia if, o if … else … o if … elif … else …

Podemos reescribir el último ejemplo de la siguiente manera:

```
x = 28
if x < 0:
   print(f'{x} es menor que 0')
else:
   if x > 0:
       print(f'{x} es mayor que 0')
   else:
       print('x es 0')
```

SENTENCIAS ITERATIVAS

Un bucle o ciclo en programación es la ejecución continua de un determinado bloque de código mientras una condición asignada se cumple.

SENTENCIA *FOR*

El bucle for se utiliza para recorrer los elementos de un objeto iterable (lista, tupla, conjunto, diccionario) y ejecutar un bloque de código. En cada paso de la iteración se tiene en cuenta a un único elemento del objeto iterable, sobre el cuál se pueden aplicar una serie de operaciones.

Su sintaxis es la siguiente:

```
for <elem> in <iterable>:
   <sentencias>
```

Aquí, **`elem`** es la variable que toma el valor del elemento dentro del **`iterador`** en cada paso del bucle. Este finaliza su ejecución cuando se recorren todos los elementos.

Si tenemos una lista de números y queremos mostrarlos por consola podríamos hacer:

```
nums = [4, 78, 9, 84]
for n in nums:
   print(n)
```

```
4
78
9
84
```

---

salida

Ahora, si tenemos una lista de números y queremos mostrar por consola solo aquellos que son pares, podríamos hacer:

```
#Creación de la lista con números
numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9,10]
#En la variable num se almacenan los ítem de la lista
for num in numeros:
      if num % 2 == 0: #Si el resto de la división por dos es cero
            print (num)
```

```
2
4
6
8
10
```

---

salida

Iteración en listas

En el siguiente ejemplo definiremos una lista de palabras y un ciclo para iterarla, luego a cada ítem le calculamos su longitud, es decir la cantidad de caracteres de cada palabra, usando la función **`len`**, además mostramos esta información.

```
Palabras = ['Peine', 'Pelo', 'Ventana', 'Dentista', 'Asesino']
for caracteres in Palabras:
      print ((caracteres), ('( Longitud:'), (len(caracteres)),')')
```

```
Peine ( Longitud: 5 )
Pelo ( Longitud: 4 )
Ventana ( Longitud: 7 )
Dentista ( Longitud: 8 )
Asesino ( Longitud: 7 )
```

---

salida

Iteración en diccionarios

Un caso es especial de bucle for se da al recorrer los elementos de un diccionario. Dado que un diccionario está compuesto por pares `clave/valor`, hay distintas formas de iterar sobre ellas.

Podemos iterar sobre las claves del diccionario, usando las sentencias básicas que vimos anteriormente:

```
Agenda = {
      'Marcelo' : '3443456993',
      'Gaston' : '3443456992',
      'Lucas' : '3443456991',
      }
for k in Agenda:
      print (k)
```

```
Marcelo
Gaston
Lucas
```

---

salida

Podemos iterar sobre los valores del diccionario, usando la función predefinida **`values()`**. Observen como se la implementa, usando un punto luego del nombre de la variable en la que se definió el diccionario:

```
Agenda = {
      'Marcelo' : '3443456993',
      'Gaston' : '3443456992',
      'Lucas' : '3443456991',
      }
for v in Agenda.values():
      print (v)
```

```
3443456993
3443456992
3443456991
```

---

salida

Por último, podemos iterar a la vez sobre la clave y el valor de cada uno de los elementos del diccionario usando dos variables **k** y **v** e implementando la función

predefinida **`items()`**.

```
Agenda = {
      'Marcelo' : '3443456993',
      'Gaston' : '3443456992',
      'Lucas' : '3443456991',
      }
for (k, v) in Agenda.items():
      print (k, v)
```

```
Marcelo 3443456993
Gaston 3443456992
Lucas 3443456991
```

---

salida

Iteración en cadenas de texto

Veremos su uso desarrollando el siguiente ejemplo, donde definimos una cadena determinada y contamos cuántas letras 'a' posee la misma.

```
entrada = "mi mamá me mima" #Cadena de texto a analizar
contador = 0
cuentalaletra = 'a' #Almacenamos en la variable la letra a contar
for letra in entrada:
   if letra == cuentalaletra: #Comparamos cada letra de la cadena
            contador =contador + 1
print ((cuentalaletra), (':'), (contador))
```

```
a : 2            #Notar que no cuenta la a acentuada de mamá
```

---

salida

La clase range

Una de las iteraciones más comunes que se realizan, es la de iterar un número, por ejemplo, entre 0 y n. Supongamos que queremos iterar una variable **`i`** de 0 a 3.

Haciendo uso de lo que hemos visto anteriormente, podríamos hacer lo siguiente:

```
for i in (0, 1, 2, 3):
   print(i)
```

```
0
1
2
3
```

salida

La clase **`range`** se usa para implementar y/o simular el ciclo **`for`** basado en una secuencia numérica. El constructor de esta clase, **`range(max)`**, devuelve un iterable cuyos valores van desde 0 hasta max - 1.

Por lo tanto el ejemplo anterior usando esta nueva clase sería:

```
for i in range(3): #i tomará los valores 0, 1 y 2
  print(i)
```

```
0
1
2
```

salida

El tipo de datos **`range`** se puede invocar con uno, dos e incluso tres parámetros:

- **`range(max)`**: Un iterable de números enteros consecutivos que empieza en 0 y acaba en max - 1
- **`range(min, max)`**: Un iterable de números enteros consecutivos que empieza en min y acaba en max - 1
- **`range(min, max, step)`**: Un iterable de números enteros consecutivos que empieza en min acaba en max - 1 y los valores se van incrementando de step en step. Este último caso simula el bucle for con variable de control.

Veamos un ejemplo de cada caso:

```
#todos los enteros entre 0 y 3
for i in range(4):
 print(i)
```

```
0
1
2
3
```

---

salida

```
#todos los enteros entre 0 y 3
for i in range(0,4):
 print(i)
```

```
0
1
2
3
```

---

salida

```
#todos los enteros PARES entre 0 y 3
for i in range(0,4,2):
 print(i)
```

```
0
2
```

---

salida

1000 PROGRAMADORES

Introducción a la **Programación en Python**

Ministerio de Educación
Cultura, Ciencia y Tecnología
Gobierno de Salta

Ministerio de Economía
y Servicios Públicos
Gobierno de Salta

Universidad
Nacional de Salta

SENTENCIA *WHILE*

La sentencia o bucle while es una sentencia de control de flujo que se utiliza para ejecutar un bloque de instrucciones de forma continuada mientras se cumpla una condición determinada.

El cuerpo del ciclo se ejecutará mientras una condición determinada sea verdadera. Es decir, si la condición se cumple se ejecutará el cuerpo de dicho ciclo y al finalizar se volverá a comprobar la condición, si continúa siendo verdadera pues se ejecutara nuevamente. Cuando se deje de cumplir, se saldrá del ciclo y se continuará la ejecución normal. Llamaremos **iteración** a una ejecución completa del bloque de código.

La estructura de esta sentencia es la siguiente:

```
while (condición):
   bloque de código
```

Mientras `condición` se evalúe a `True`, se ejecutarán las instrucciones y sentencias de `bloque de código`. Aquí, `condición` puede ser un literal, el valor de una variable, el resultado de una expresión o el valor devuelto por una función.

```
x = 3
while x > 0:
   x -=1
   print(x)
```

```
2
1
0
```

---

salida

La sentencia *while* se puede usar en multitud de ocasiones. Por ejemplo: Comprobar si existe un elemento en una secuencia.

Si tenemos la lista de valores [5, 1, 9, 2, 7, 4] y queremos saber si el número 2 está contenido en dicha lista. La estructura típica de bucle while para ello es como sigue:

```
valores = [5, 1, 9, 2, 7, 4]
encontrado = False
indice = 0
longitud = len(valores)
while not encontrado and indice < longitud:
   valor = valores[indice]
   if valor == 2:
       encontrado = True
   else:
       indice += 1
if encontrado:
   print(f'El número 2 ha sido encontrado en el índice {indice}')
else:
   print('El número 2 no se encuentra en la lista de valores')
```

```
El número 2 ha sido encontrado en el índice 3
```

salida

En este ejemplo observamos que se utilizan 3 variables de control:

- **encontrado**: Indica si el número 2 ha sido encontrado en la lista.
- **indice**: Contiene el índice del elemento de la lista que va a ser evaluado.
- **longitud**: Indica el número de elementos de la lista.

En esta ocasión, la condición de continuación del bucle *while* es que no se haya encontrado el número 2 y que el índice del elemento a evaluar sea menor que la longitud de la lista (es posible acceder a un elemento de una lista a partir del índice de su posición, comenzando por 0).

Por tanto, el bucle finaliza bien cuando se haya encontrado el elemento o bien cuando se haya evaluado el último elemento de la lista. Si se ha encontrado el número 2, se muestra un mensaje indicando el índice en el que está. En caso contrario, se muestra un mensaje indicando que el número 2 no se encuentra en la lista.

El anterior es solo un ejemplo para ver cómo funciona la sentencia while. En realidad, en Python se puede usar el operador ***in*** *para comprobar de forma automática si un elemento está contenido en una secuencia.*

Veamos cómo crear un menú de opciones muy sencillo:

```
b = True
while b:
    opcion = (input("""Elige una fruta para tu desayuno:
    1- Manzanas
    2- Bananas
    3- Nada
    """))
    if opcion == '1':
        print ("Has seleccionado manzanas")
    elif opcion == '2':
        print ("Has seleccionado bananas")
    elif opcion == '3':
        print ("Has seleccionado nada")

    rta = (input("Ingresa 'chau' para terminar "))
    if rta == "chau":
        b = False
print("FIN")
```

```
Elige una fruta para tu desayuno:
    1- Manzanas
    2- Bananas
    3- Nada
    1                              #el usuario elige 1
Has seleccionado manzanas
Ingresa 'chau' para terminar e     #el usuario elige NO terminar
Elige una fruta para tu desayuno:
    1- Manzanas
    2- Bananas
    3- Nada
    1                              #el usuario elige 1
Has seleccionado manzanas
Ingresa 'chau' para terminar chau  #el usuario elige terminar
```

```
FIN
```

---

salida

En la primera parte del código se puede ver que creamos el ciclo debajo, en el cuerpo del mismo colocamos un **input** para mostrarle al usuario las opciones y permitirle elegir una para almacenarla en la variable “opcion”.

Dentro del cuerpo del ciclo, de acuerdo al ingreso de usuario se ejecuta una u otra tarea, en este caso simplemente mostrar un mensaje.

También dentro del cuerpo del ciclo, pero hacia el final, usamos otro input para darle la posibilidad al usuario de detener la ejecución o no del ciclo.

EXPRESIONES ANIDADAS

Las estructuras o flujos de control pueden anidarse, es decir, la ejecución de alguna de ellas se encuentra dentro del **bloque** o **cuerpo de código** de otra y por lo tanto su ejecución dependerá de la ejecución de esa otra.

Desarrollaremos el siguiente ejemplo: "Determinar cuáles son los 5 primeros números primos de 2 cifras"

```
#Se sugiere leer los comentarios desde el último hasta el primero.
x = 9
contador = 0
while (contador<5):          #este ciclo controla la estructura for
que a su vez controla la estructura de alternativa.
 x = x+1
 cant = 0
 for divisor in range(1,x+1):#el ciclo for se repite mientras la
condición del contador sea verdadera
   if ((x%divisor)==0):      #la pregunta se realiza las veces que
se repite el ciclo for
     cant=cant+1
 if (cant == 2):
   contador = contador + 1
   print("El",contador,"° primo de 2 cifras es:",x)
```

```
El 1 ° primo de 2 cifras es: 11
El 2 ° primo de 2 cifras es: 13
El 3 ° primo de 2 cifras es: 17
El 4 ° primo de 2 cifras es: 19
El 5 ° primo de 2 cifras es: 23
```

---

salida

BIBLIOGRAFÍA

[1] Python para todos. R. Gonzalez Duque. Licencia Creative Commons Reconocimiento 2.5 España.
Descarga gratuita de versión más reciente en http://mundogeek.net/tutorial-python/
[2] Python 3 al descubierto 2da Edición. A. Fernández Montoro. Alfaomega. Año 2013
[3] Tutoriales Web varios.