Python Programming

Módulo 1

Sentencias de control

Condicional IF

Un código no deja de ser un conjunto de instrucciones que son ejecutadas unas tras otra. Gracias a las estructuras de control `IF`, podemos cambiar el flujo de ejecución de un programa, haciendo que ciertas partes del código se ejecuten si se dan condiciones particulares.

IF

Es muy importante tener en cuenta que la sentencia IF debe ir terminada por : y el bloque de código a ejecutar debe estar indentado.

```
edad = 16
if edad < 18:
      print("Sos menor de edad.")
```

IF - ELSE

Es posible que no solo queramos hacer algo si una determinada condición se cumple, sino que además queramos hacer algo, en caso contrario. Por eso usamos la cláusula else.

```
edad = 30
if edad < 18:
      print("Sos menor de edad.")
else:
      print("Sos mayor de edad.")
```

Notar que ambos bloques de código son excluyentes, se hace uno u otro, pero nunca se ejecutarán los dos.

Condicional IF

IF - ELIF - ELSE

Hasta ahora hemos visto como ejecutar un bloque de código si se cumple una condición, o ejecutar otro bloque si no se cumple, pero hay veces que no es suficiente. En muchos casos, podemos tener varias condiciones con diferentes posibilidades y para cada una queremos un código distinto.

```
edad = 70

if edad < 18:
      print("Sos menor de edad.")
elif edad < 65:
      print("Sos mayor de edad.")
else:
      print("Sos Jubilado.")
```

Cuando ninguna de las condiciones se cumple (es decir, todas retornan `False`), se ejecuta el bloque de código luego de `else`.

Este lenguaje no tiene la sentencia de control switch, con lo antes visto podemos reemplazar sin problema.

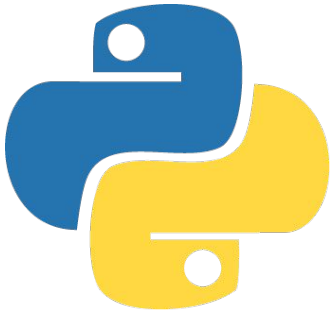
Condicional IF

Operador Ternario

El operador ternario es una herramienta muy potente que muchos lenguajes de programación emplean. Se trata de una cláusula if-else que se define en una sola línea y puede ser usado dentro de un `print()`.

```
valor = 10
print("Valor es 10" if valor==10 else "Valor no es 10")

out: Valor es 10
```

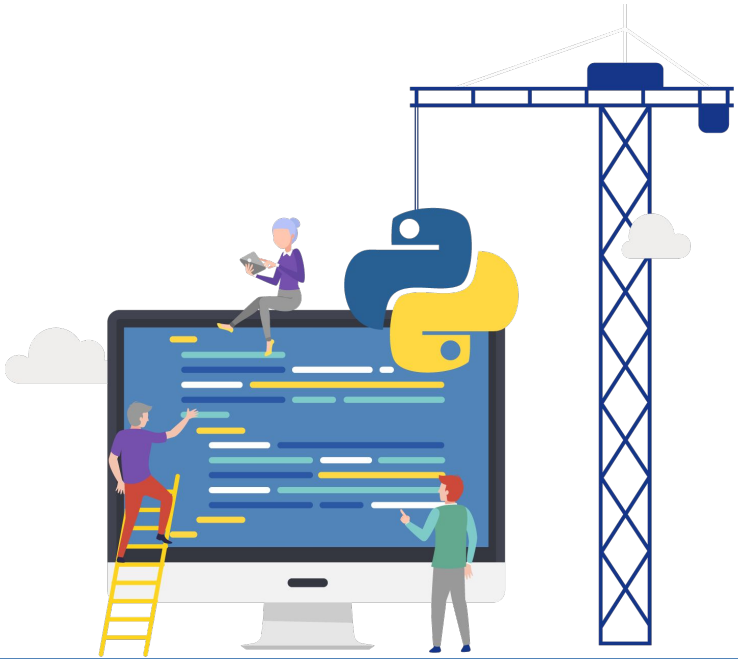


Condicional IF

Para tener en cuenta:

Los números en Python equivalen a `True` siempre que sean distintos a cero. Las colecciones (listas, tuplas, diccionarios e incluso las cadenas) equivalen a `True` cuando no están vacías (es decir, cuando `len(colección) > 0`).

```
a = [1, 2, 3, 4]
if a:
      print("La lista no está vacía.")
else:
      print("La lista está vacía.")
```



Bucle WHILE

`While` nos permite ejecutar una sección de código repetidas veces. El código se ejecutará mientras una condición se cumpla. Cuando se deje de cumplir, se sale del bucle y se continuará la ejecución normal.

Suele llamarse *iteración* a una ejecución completa del bloque de código.

```
a = 1
while a < 10:
      print(a)
      a += 1
```

El código imprime en pantalla los números del 1 al 9 ya que, según la condición, el bucle debe ejecutarse siempre que a sea menor a 10. En cada ejecución utiliza la función incorporada `print()` para enviar un mensaje a la pantalla y la sentencia `a += 1` para incrementar el valor de a en 1 (es equivalente a `a = a + 1`). Si hubiésemos omitido esta última línea, el código imprimiría 1 en pantalla infinitamente.

Bucle FOR

A diferencia de otros lenguajes, en Python el bucle **“for”** se utiliza para recorrer una colección de objetos (una lista, tupla, diccionario, etc.). Se dice que un objeto es “iterable” cuando puede ser “recorrido” (estrictamente hablando el concepto de objeto iterable es un tanto más complejo, pero para nuestro propósito vamos a obviar los detalles).

En el for no existe condición, sino un iterable que define las veces que se ejecutará el código. En el siguiente ejemplo vemos un bucle for que se ejecuta 4 veces.

Ejemplo:

```
datos = [10, 20, 30, 40]
for aux in datos:
      print(aux)
```

Este código imprime en pantalla cada uno de los elementos en la lista datos. Le estamos indicando a Python que “para cada elemento de la lista datos, ejecute el siguiente código (`print(aux)`), luego de haber colocado el valor de dicho elemento en `aux`.

Break y Continue

Break

La sentencia `break` nos permite alterar el comportamiento de los bucles `while` y `for`. Concretamente, permite terminar con la ejecución del bucle.

Esto significa que una vez se encuentra la palabra `break`, el bucle será terminado independientemente si la condición es verdadera o falsa.

Continue

El uso de continue nos permite modificar el comportamiento de de los bucles `while` y `for`.

Concretamente el continue saltea todo el código restante en la iteración actual y vuelve al principio en el caso de que aún queden iteraciones por completar.

La diferencia entre el `break` y `continue` es que el `continue` no rompe el bucle, si no que pasa a la siguiente iteración saltando el código pendiente para terminar la vuelta actual.

¡Sigamos trabajando!