¿Qué es un condicional?

Los condicionales en python te permiten tomar decisiones en base al resultado de comparaciones o el valor de variables. Normalmente se usan dentro de declaraciones “if” y en bucles.

Lista de condicionales:

- Igual: x == y
- Desigual: x != y
- Menor que: x < y
- Menor o igual que: x <= y
- Mayor que: x > y
- Mayor o igual que: x >= y

Ejemplos de uso:

```
x = 5
y= 10
If y > x:
   print(“y es mayor que x”) Tienes que usar la indentación o recibirás un error
```

Si solo tienes una condición puedes ponerlo todo en la misma línea

```
if y > x: print(“y es mayor que x”)
```

Elif

La clave **elif** se usa para aplicar una condición cuando las demás no sean verdaderas.

```
x = 5
y= 5
If y > x:
   print(“y es mayor que x”)
elif x == y
   print(“x es igual a y”)
```

Else

La clave **else** se utiliza para todo lo demás que no se encuentre dentro de **if** o **elif.** En el siguiente ejemplo las dos primeras condiciones no son válidas, por lo que se utilizara la condición **else**.

```
x = 10
y= 5
If y > x:
   print(“y es mayor que x”)
elif x == y
   print(“x es igual a y”)
else
   print(“x es mayor que y”)
```

Si solo tienes una condicion para **if** y otra para **else** puedes ponerlo todo en la misma linea.

print ("X") if x > y else print ("Y")

And

La palabra clave **and** se puede usar para combinar dos condiciones.

```
x = 5
Y = 3
Z = 1
If x > y and x > z
   print("x es mayor que z e y")
```

Or

La palabra clave or se utiliza para combinar dos declaraciones y que solo una tenga que ser verdadera.

```
x = 5
Y = 3
Z = 36
If x > y or x > z
   print("al menos una es verdadera")
```

Not

Not se utiliza cuando quieres dar por valido lo contrario a la condición que has utilizado

```
x = 5
y= 10
If not x > y:
   print("x NO es mayor que y")
```

Tambien puede haber declaraciones if dentro de otras declaraciones if, esto se llama declaraciones if anidadas.

```
x = 20
Y = 10
Z = 30
If x > y:
   print("x es mayor que y")
   If x > z:
     print("y tambien mayor que z")
   else:
      print("pero no mayor que z")
```

¿Cuáles son los diferentes tipos de bucles en Python? ¿Por qué son útiles?

Python dispone de dos tipos de bucles, Los **For in** y los **While**

For in:

Los bucles for in se utilizan cuando se quiere repetir un bloque de código un número limitado de veces. Siempre se utilizan con un objeto iterable, como una lista, una tupla, un rango de números...

La declaración for itera sobre los componentes de una secuencia en orden, ejecutando el bloque de codigo cada vez.

Ejemplo: Muestra en la consola cada nombre dentro de nombres

```
nombre = ["miguel", "ana", "patricia"]
for nombre in nombres:
  print(nombre)
```

El bucle itera por cada letra de “patata” y la imprime en la consola:

```
for letra in “patata”:
  print(letra)
```

La función Range()

La función range() permite usar el bucle un numero especifico de veces. Esta devuelve una secuencia de números, empezando por el 0 por defecto, y aumentando en incrementos de 1, y termina en el número especificado.

```
for x in range(11):
  print(x)
```

En este bloque de código se imprimirán en la consola los números del 1 al 10. Hay que tener en cuenta que range(11) no significa del 1 al 11, sino del 1 al 10.

```
for x in range(3, 11):
  print(x)
```

En este bloque de código hemos añadido un argumento de inicio, por lo que empezara a imprimir en la consola empezando por el 3.

La función range() incrementa la secuencia por 1 cada vez, sin embargo, podemos cambiar esto añadiendo un tercer argumento.

```
for x in range(10, 31, 5):
  print(x)
```

Este bloque de código imprimirá los números en la consola, empezando por el 10 y aumentando en incrementos de 5 hasta llegar a 30.

Continue y Break

Con las declaraciones continue y break podemos parar la iteración actual y pasar a la siguiente y parar el bucle completo respectivamente.

Ejemplo: Cuando llega al nombre “ana” continua a la siguiente parte sin llegar a imprimir el nombre

```
nombre = ["miguel", "ana", "patricia"]
for nombre in nombres:
   If nombre == “ana:
      continue
   print(nombre)
```

Rompe el bucle después de llegar e imprimir en la consola “ana”

```
nombre = ["miguel", "ana", "patricia"]
for nombre in nombres:
  print(nombre)
   If nombre == “ana:
      break
```

Bucles anidados

Los bucles anidados son bucles dentro de otros bucles.
En un bucle anidado el bucle interior se ejecutará una vez por cada iteración del bucle exterior.

```
colores = ["rojo", "gris", "azul"]
coches = [“audi”, “toyota”, “seat”]

for color in colores:
  for coche in coches:
     print(color, coche)
```

While:

Con los bucles while podemos ejecutar un bloque de código mientras una condición sea verdadera

Ejemplo:

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

Este bloque de código imprimirá en la consola x y lo incrementará por uno por cada iteración, siempre que x sea menor o igual a 10

También podemos utilizar **break** para parar el bucle, aunque la condición sea verdadera.

```
x = 1
while x <= 10:
   print(x)
   If x = = 5:
      break
   x += 1
```

O podemos utilizar **continue** para pasar a la siguiente iteración.

```
x = 1
while x <= 10:
   X +=1
   If x = 6:
      continue
   Print(x)
```

Else

Podemos utilizar la declaración **else** para ejecutar un bloque de código una vez la condicion no sea verdadera

```
x = 1
while x <= 10:
   print(x)
   x += 1
else:
   print("has alcanzado el máximo")
```

¿Qué es una lista por comprensión en Python?

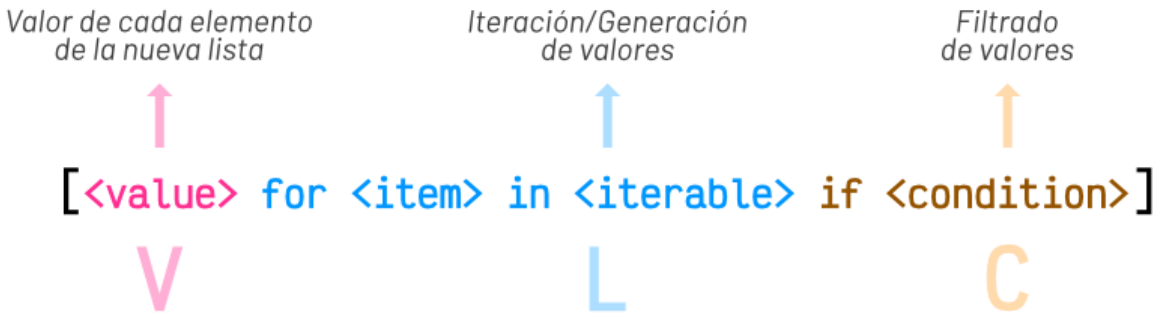
La comprensión de listas te permite utilizar una sintaxis más corta cuando quieres crear una lista usando los valores de otra lista existente.

Si no usas la comprensión de listas, tendrías que utilizar esta sintaxis:

```
Coches = ["audi","bmw","seat","toyota","ford"]
nueva_lista = [ ]

for coche in coches
   if "a" in coche:
      newlist.append(coche)
print(nueva_lista)
```

En cambio, usando la comprensión de listas, podemos hacer todo es en una sola línea de código, la sintaxis es la siguiente:



coches = [“audi”,”bmw”,”seat”,”toyota”,”ford”]

nueva_lista = [coche for coche in coches if “a” in coche]

print(nueva_lista)

¿Qué es un argumento en Python?

Los argumentos son los valores o información que se pasan a las funciones para que realicen una tarea. Los argumentos se especifican después del nombre de la función, entre paréntesis. Puedes especificar cuantos argumentos quieras, siempre que estén separados por una coma.

```
def nombre_completo(nombre, apellido):
    print(nombre + “ “ + apellido)
```

Por defecto, una función debe declararse con el numero correcto de argumentos. Si como en el ejemplo de arriba, una función espera 2 argumentos, debes usar 2 argumentos, ni mas ni menos, o no funcionara.

También podemos usar cualquier tipo de dato como argumento (string, numero, lista, diccionario, etc.)

```
def función(vehículos)
for vehículo in vehículos:
    print(coche)

coches = [“Audi”, “BMW”, “Toyota”, “Seat” ]

función(coches)
```

Argumento por defecto

Podemos asignar un valor por defecto para que se use cuando no se use ningún argumento a la hora de ejecutar la función.

```
def saludo(nombre = “invitado”):
    print(“Saludos “ + nombre)
```

Argumentos arbitrarios (*args)

Si no sabes cuantos argumentos necesitaras para tu función, puedes usar los argumentos arbitrarios escribiendo * justo antes del nombre del argumento.

```
def lista(*persona):
    print("El último de la lista es “ + persona[-1])

lista(“Juan”,”Marta”,”Arturo”)
```

Argumentos con palabra clave

Puedes usar los argumentos con palabra clave para asignar una palabra clave como valor del argumento, de esta manera el orden de los argumentos es irrelevante.

```
def lista(persona1, persona2, persona3):
    print("El último de la lista es “ + persona3)

lista(persona1 = “Juan”, persona2 = “Marta”, persona3 = “Arturo”)
```

*Argumentos arbitrarios con palabra clave (**kwargs)*

*Si no sabes cuantos argumentos de palabra clave necesitaras para tu función, añade dos asteriscos ** antes del nombre del argumento.*

```
def funcion(**persona):
    print(“Su apellido es “ + persona[“apellido”])

función(nombre = “John”, apellido = “Smith”)
```

¿Qué es una función Lambda en Python?

Una función lambda es una pequeña función que no requiere de nombre, en su lugar se utiliza la palabra clave **lambda** en lugar de la clasica **def**. Puede usar cualquier número de argumentos, pero solo puede tener una expresión.

```
def a(x, y):
    return x + y

b = lambda x, y: x + y
```

resultado = lambda x, y, z : x + y + z

print(resultado(1,2,3)

Las funciones lambda están mejor utilizadas cuando las usas como función anónima dentro de otra función.

def potencia(y):
* return lambda x : x * y*

potencia_de_2 = potencia(2)
potencia_de_3 = potencia(3)

print(potencia_de_2(5))
print(potencia_de_3(5))

¿Qué es un paquete PIP

PIP es un administrador para módulos de Python. Los módulos son librerías de código de Python que puedes importar para usar código creado por otras personas en tus proyectos.

PIP viene instalado desde la version de Python 3.4 en adelante, de no ser asi puedes instalarlo en este link:

https://pypi.org/project/pip/

Puedes comprobar si tienes instalado PIP usando este comando en la consola:

```
C:\Users\Your Name\AppData\Local\Programs\Python\Python36-32\Scripts>pip –version
```

Descargar e Instalar/Desinstalar paquetes

Puedes descargar un paquete de forma muy sencilla. Entra en la consola de comandos y navega hasta el directorio Scripts de Python. Desde allí puedes usar *install + nombre del paquete* para descargarlo.

Ejemplo con el paquete **camelcase**:

```
C:\Users\Your Name\AppData\Local\Programs\Python\Python36-32\Scripts>pip install camelcase
```

De la misma forma, puedes eliminar un paquete que no necesites usando el *uninstall + nombre del paquete* en el mismo directorio.

```
C:\Users\Your Name\AppData\Local\Programs\Python\Python36-32\Scripts>pip uninstall camelcase
```

Después de introducir el comando se te pedirá una confirmación para la eliminación del paquete, pulsa y para eliminar el paquete

Ejemplo de uso

Para usar un paquete descargado primero tendrás que importarlo primero.

Para importar el paquete math tendremos que escribir lo siguiente:

import math

De la misma manera, si queremos importar una función específica del paquete, podemos hacerlo de la siguiente manera:

from math import sqrt

También podemos importar un paquete dándole un alias, de esta forma podemos usar el alias en lugar del nombre del paquete al escribir el código:

Import math as m

math.sqrt(5) *-->* *m.sqrt(5)*