- ¿Qué es un condicional?

Un condicional nos permite agregar condiciones y aplicar distintos bloques de código en función de si estas se cumplen o no, creando así una programación más dinámica.

Para ello utilizamos la estructura 'if'. Esta solo ejecutará el bloque de código si la condición que haya después de 'if' se cumple (es decir que el valor booleano sea de 'True'). Hay que añadir dos puntos(':') tras la condición y cabe destacar la importancia de una indentación correcta a la hora de escribir el código para evitar errores.

```
edad = 44

if edad > 20:
   print(¡Acceso autorizado!)
```

También existe la opción de añadir la sentencia 'else' que ejecutará el código si la condición del if anterior no se cumple(tiene un valor booleano de 'false').

```
if edad > 20:
   print((¡Acceso autorizado!)
else:
   print(¡Acceso denegado!)
```

Asimismo podemos utilizar la sentencia 'elif' para encadenar varias condiciones después de la 'if' inicial. Solo la primera condición en cumplirse será la que se ejecute.

```
if edad < 20:
   print(¡Acceso denegado!)
elif edad > 80:
   print(¡Acceso denegado!)
else:
   print(¡Acceso autorizado!)
```

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

Hay dos tipos de bucles en Python los 'For' y los 'While'. Ambos nos permiten implementar iteraciones sobre una colección(Tupla, diccionario, lista, etc.) y ejecutar código repetidas veces. Su utilidad consiste en automatizar un proceso que de otra forma tardaría mucho en programarse.

Bucle ‘For’

Este bucle ejecuta código repetidamente en función de la cantidad de elementos que contenga el objeto iterable, se detendrá una vez no queden más elementos.

Sintaxis: For variable iteradora(puede ser cualquier nombre) in colección: bloque de código

También es importante tener una indentación correcta.

```
fruits = [‘pineapple’, ‘orange’, ‘banana’, ‘lemon’]

for fruit in fruits:
   print(fruit)
```

Bucle ‘While’

Los bucles ‘While’ ejecutan los bloques de código siempre y cuando la condición del bucle sea verdadera. Solo se detendrá cuando la condición sea falsa, para ello es necesaria la implementación de un valor centinela. Este valor es de vital importancia, ya que sin él, la condición podría ser verdadera para siempre y crear un bucle infinito.

Sintaxis: While condición: bloque de código

```
num = 1
while num < 6:
  print(num)
  num += 1
```

- ¿Qué es una lista por comprensión en Python?

Una lista por comprensión nos permite crear nuevas listas utilizando valores de colecciones preexistentes, para ello establecemos bucles ‘for’ en una única línea de código, creando así listas de forma dinámica y con una sintaxis más concisa.

Para usarlas, primero creamos una nueva lista y la asignamos de la siguiente forma: Lista = [expresión for elemento in iterable].

Expresión:La expresión es la operación que se repetirá en cada iteración.
Elemento: Cada uno de los elementos que contenga la colección.

Iterable: la lista, tupla, o cadena original.
Es necesario el uso de corchetes(‘[]’) puesto que el objetivo es generar una lista.

lista = range(1, 11)

cuadrados = [num ** 2 for num in lista]

También existe la opción de añadir una condicional, por lo que solo se añadirán a la lista los elementos que cumplan determinadas condiciones.

lista = range(1, 11)

pares = [num for num in lista if num % 2 == 0]

- ¿Qué es un argumento en Python?

Los argumentos son información que podemos añadir a las funciones, los argumentos se especifican después del nombre de la función entre paréntesis’()’. Se pueden añadir más de un argumento a una función, pero, al llamarla debemos utilizar la misma cantidad de argumentos que cuando fue definida.

```
def suma(a, b):
   total = a + b
   print(total)

suma(2, 3)
```

Existe la opción de añadir argumentos por defecto mediante el uso del signo igual (‘=’), así no será necesario el utilizar argumentos al llamar la función.

```
def suma(a = 6, b = 2):
   total = a + b
   print(total)

suma()
```

La posición del argumento es la que define el orden de los parámetros, pero podemos hacer uso de argumentos con nombre al llamar la función para ignorar el orden.

```
def full_name(first, last):
    print(f'{first} {last}')

full_name(last = 'García', first = 'Pablo')
```

Si no estamos seguros de la cantidad de argumentos que nuestra función va a necesitar podemos emplear argumentos arbitrarios, estos nos permiten usar una cantidad variante de argumentos al llamar la función. La sintaxis consiste en usar un asterisco (*) al definir la función. La función recibe una tupla de argumentos.

```
def my_function(*kids):
    print("The youngest child is " + kids[2])

my_function("Emil", "Tobias", "Linus")
```

Por último, mencionar que podemos combinar los argumentos arbitrarios y argumentos con nombre para crear los argumentos arbitrarios clave. La sintaxis es ahora de dos asteriscos (**). La función recibe un diccionario de argumentos.

```
def my_function(**kid):
    print("His last name is " + kid["lname"])

my_function(fname = "Tobias", lname = "Refsnes")
```

Todos estos tipos de argumentos pueden usarse al mismo tiempo.

- ¿Qué es una función Lambda en Python?

La función Lambda es una forma corta de declarar funciones pequeñas y anónimas (es decir, que carecen de nombre). Lambda nos permite almacenar un proceso, e incorporarlo a otras funciones de forma fácil y rápida.

Sintaxis: lambda argumentos: expresión

Las funciones Lambda pueden tener cualquier número de argumentos, pero solo una expresión.

cubo = lambda x: x ** 3
print(cubo(2))

- ¿Qué es un paquete pip?

Un paquete pip es una librería o módulo con diversas funcionalidades. Estos paquetes no se hallan en python de base, por lo que primero debemos descargarlos online o mediante el uso de la herramienta pip e importarlos. Cualquier usuario puede distribuir o utilizar estos paquetes, de modo que podemos utilizar el trabajo de otros desarrolladores en nuestros propios proyectos, ahorrándonos así tiempo ,esfuerzo y formando una comunidad saludable.