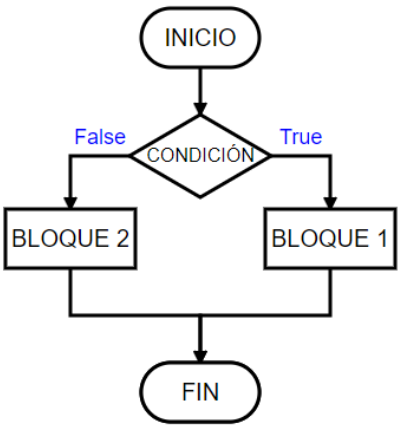
- El condicional:

Los condicionales son estructuras que permiten elegir entre la ejecución de una acción u otra. Son una condición, como bien indica su nombre, así que podemos pensar en ellos como si fueran el “si” condicional que usamos dentro de una frase.

Es decir, se trata de una situación a cumplir para que se ejecute una parte (true) u otra (false) del código posteriormente detallado. Suele trabajarse en la estructura de control if, la cual permite que un programa ejecute unas instrucciones cuando se cumplan una o más condiciones. Para poder trabajar con múltiples situaciones, tenemos como soporte elif (para otra situación específica posterior al if inicial) y else (para el resto de las situaciones no detalladas previamente).



Como ejemplo teórico tenemos la imagen de la derecha, y como ejemplo práctico la de abajo:

```python
a = 10
b = 20

if a < b:
    print(f"{a} es menor que {b}")
elif b < a:
    print(f"{a} es mayor que {b}")
else:
    print(f"{a} y {b} son iguales")
```

- Tipos de bucle en Python y sus utilidades:

En la gran mayoría de lenguajes nos solemos encontrar con 3 tipos de bucles: for, while, do while. Sin embargo, en Python sólo tenemos disponibles los 2 primeros.

La instrucción for permite repetir una instrucción o una instrucción compuesta un número especificado de veces. El cuerpo de una instrucción for se ejecuta cero o más veces hasta que una condición opcional sea false. A diferencia de bucle while (como ya veremos después), el bucle for, se aplica cuando el bloque a iterar tiene un número final definido.

```python
for i in range(1, 11):
    print(i)
```

El bucle while es otra estructura de control de flujo, concretamente lo que hace es repetir un código mientras dure una determinada condición. Se puede decir que el bucle while se utiliza para hacer algo repetidamente, bajo unas condiciones específicas, sin saber cuántas veces se repetirá.

Es decir, el bucle while, se aplica cuando la iteración no tiene un número definido. Esto puede llevar a bucles infinitos (Ctrl+C para detener la ejecución) como el siguiente:

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

Para evitar que se produzcan estos bucles infinitos, es necesario añadir una operación (de suma en el siguiente ejemplo) al condicionante del bucle, en este caso la variable a:

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

- Lista por comprensión en Python:

La comprensión de listas es una construcción en Python que nos permite crear listas a partir de otras listas, tuplas y cualquier iterable. Nos permite escribir en forma más concisa un algoritmo. Es una funcionalidad que nos permite crear listas avanzadas en una misma línea de código. Esto se ve mucho mejor en la práctica, por lo que ahí va el ejemplo:

```
# Método tradicional
lista = []
for letra in 'casa':
    lista.append(letra)
print(lista)

# ['c', 'a', 's', 'a']

# Con comprensión de listas
lista = [letra for letra in 'casa']
print(lista)

# ['c', 'a', 's', 'a']
```

Las compresiones de listas son una herramienta muy poderosa, que crea una lista basada en otra, en una única línea legible. Un ejemplo típico de esto es buscar el conjunto de elementos comunes en dos listas. Lo que se puede conseguir de con el siguiente código.

```
names_1 = ['Oralie' ,'Imojean' ,'Michele', 'Ailbert', 'Stevy']
names_2 = ['Jayson', 'Oralie' ,'Michele', 'Stevy', 'Alwyn']
common = [a for a in names_1 for b in names_2 if a == b]
# ['Oralie', 'Michele', 'Stevy']
```

- Los argumentos en Python:

Los argumentos en funciones se refieren a los valores que se pasan a una función para que realice una tarea específica. Como desarrolladores, es común encontrarnos con situaciones en las que necesitamos que una función tome distintos valores para realizar una tarea en particular.

A veces se le confunde con el parámetro. Sin embargo, la diferencia está en que se le llama parámetro a la variable que se declara al definir la función (en donde se recibe el valor). En cambio, se le llama argumento al valor que es enviado a la función cuando se invoca.

```
def my_function(parameter):#El parámetro
  print(parameter)

my_function("argument") #El argumento
```

En la mayoría de los casos, un procedimiento debe tener alguna información sobre las circunstancias en las que ha sido llamado. Un procedimiento que realiza tareas repetidas o compartidas usa información diferente en cada llamada. Esta información consta de variables, constantes y expresiones que se pasan al procedimiento al llamarlo.

Para comunicar esta información al procedimiento, este define un parámetro, mientras que el código de llamada pasa un argumento a ese parámetro. Puede imaginarse el parámetro como un espacio de estacionamiento y el argumento como un automóvil. Al igual que distintos automóviles pueden aparcar en un espacio de estacionamiento a distintas horas, el código de llamada puede pasar un argumento diferente al mismo parámetro cada vez que llama al procedimiento.

- La función Lambda en Python:

En Python, una función Lambda se refiere a una pequeña función anónima. Las llamamos “funciones anónimas” porque técnicamente carecen de nombre. Al contrario que una función normal, no la definimos con la palabra clave estándar def que utilizamos en Python.

El controlador de la función de Lambda es el método del código de la función que procesa eventos. Cuando se invoca una función, Lambda

ejecuta el método del controlador. La función se ejecuta hasta que el controlador devuelve una respuesta, se cierra o se agota el tiempo de espera.

Las expresiones lambda se usan idealmente cuando necesitamos hacer algo simple y estamos más interesados en hacer el trabajo rápidamente en lugar de nombrar formalmente la función.

```python
# Función Lambda para calcular el cuadrado de un número
square = lambda x: x ** 2
print(square(3)) # Resultado: 9

# Funcion tradicional para calcular el cuadrado de un numero
def square1(num):
  return num ** 2
print(square(5)) # Resultado: 25
```

- El paquete pip:

pip es el sistema de gestión de paquetes utilizado para instalar y administrar paquetes de software escritos en Python. Los paquetes de software son conjuntos de módulos y bibliotecas que pueden ser reutilizados en diferentes proyectos. Los paquetes pueden proporcionar funcionalidades que van desde operaciones matemáticas hasta acceso a bases de datos y análisis de datos, entre otros.

La ventaja de usar un gestor de paquetes como **pip** es que simplifica el proceso de instalación y actualización de paquetes, así como la gestión de dependencias entre ellos. Además, **pip** permite a los desarrolladores compartir sus propios paquetes con la comunidad y descargar paquetes creados por otros desarrolladores para usarlos en sus proyectos.

Algunas características clave de pip incluyen: instalación (**pip install nombre_del_paquete**), actualización (**pip install –upgrade nombre_del_paquete**), desinstalación (**pip uninstall nombre_del_paquete**) y listado de paquetes (**pip list**).