PROGRAMACIÓN LÓGICA Y FUNCIONAL

Condiciones

Sentencia If (Si)

Permite dividir el flujo de un programa en diferentes caminos. El if se ejecuta siempre que la expresión que comprueba devuelva True

In [1]:
```
if True:  # equivale a if not False
    print("Se cumple la condición")
    print("También se muestre este print")
```

```
Se cumple la condición
También se muestre este print
```

Podemos encadenas diferentes If

In [2]:
```
a = 5
if a == 2:
    print("a vale 2")
if a == 5:
    print("a vale 5")
```

```
a vale 5
```

O también anidar If dentro de If

In [3]:
```
a = 5
b = 10
if a == 5:
    print("a vale",a)
    if b == 10:
        print("y b vale",b)
```

```
a vale 5
y b vale 10
```

Como condición podemos evaluar múltiples expresiones, siempre que éstas devuelvan True o False

In [4]:
```
if a==5 and b == 10:
    print("a vale 5 y b vale 10")
```

```
a vale 5 y b vale 10
```

Sentencia Else (Sino)

Se encadena a un If para comprobar el caso contrario (en el que no se cumple la condición).

In [5]:
```
n = 11
if n % 2 == 0:
    print(n,"es un número par")
else:
    print(n,"es un número impar")
```

```
11 es un número impar
```

Sentencia Elif (Sino Si)

Se encadena a un if u otro elif para comprobar múltiples condiciones, siempre que las anteriores no se ejecuten.

In [6]:
```
comando = "OTRA COSA"
if comando == "ENTRAR":
    print("Bienvenido al sistema")
elif comando == "SALUDAR":
    print("Hola, espero que te lo estés pasando bien aprendiendo Python")
elif comando == "SALIR":
    print("Saliendo del sistema...")
else:
    print("Este comando no se reconoce")
```

```
Este comando no se reconoce
```

In [8]:
```
nota = float(input("Introduce una nota: "))
if nota >= 9:
    print("Sobresaliente")
elif nota >= 7:
    print("Notable")
elif nota >= 6:
    print("Bien")
elif nota >= 5:
    print("Suficiente")
else:
    print("Insuficiente")
```

```
Introduce una nota: 10
Sobresaliente
```

Es posible simular el funcionamiento de elif con if utilizando expresiones condicionales

In [9]:
```
nota = float(input("Introduce una nota: "))
if nota >= 9:
    print("Sobresaliente")
if nota >= 7 and nota < 9:
    print("Notable")
if nota >= 6 and nota < 7:
    print("Bien")
if nota >= 5 and nota < 6:
    print("Suficiente")
if nota < 5:
    print("Insuficiente")
```

```
Introduce una nota: 8
Notable
```

Instrucción Pass

Sirve para finalizar un bloque, se puede utilizar en un bloque vacío.

In [10]:
```
if True:
    pass
```

Iteraciones

Iterar significa realizar una acción varias veces. Cada vez que se repite se denomina iteración.

Sentencia While (Mientras)

Se basa en repetir un bloque a partir de evaluar una condición lógica, siempre que ésta sea True.

Queda en las manos del programador decidir el momento en que la condición cambie a False para hacer que el While finalice.

In [1]:
```
c = 0
while c <= 5:
    c+=1
    print("c vale",c)
```

```
c vale 1
c vale 2
c vale 3
c vale 4
c vale 5
c vale 6
```

Sentencia Else en bucle While

Se encadena al While para ejecutar un bloque de código una vez la condición ya no devuelve True (normalmente al final).

In [2]:
```
c = 0
while c <= 5:
    c+=1
    print("c vale",c)
else:
    print("Se ha completado toda la iteración y c vale",c)
```

```
c vale 1
c vale 2
c vale 3
c vale 4
c vale 5
c vale 6
Se ha completado toda la iteración y c vale 6
```

Instrucción Break

Sirve para "romper" la ejecución del While en cualquier momento. No se ejecutará el Else, ya que éste sólo se llama al finalizar la iteración.

In [4]:
```
c = 0
while c <= 5:
    c+=1
    if (c==4):
        print("Rompemos el bucle cuando c vale",c)
        break
    print("c vale",c)
else:
    print("Se ha completado toda la iteración y c vale",c)
```

```
c vale 1
c vale 2
c vale 3
Rompemos el bucle cuando c vale 4
```

Instrucción Continue

Sirve para "saltarse" la iteración actual sin romper el bucle.

In [9]:
```
c = 0
while c <= 5:
    c+=1
    if c==3 or c==4:
        # print("Continuamos con la siguiente iteración",c)
        continue
    print("c vale",c)
else:
    print("Se ha completado toda la iteración y c vale",c)
```

```
c vale 1
c vale 2
c vale 5
c vale 6
Se ha completado toda la iteración y c vale 6
```

Creando un menú interactivo

In [10]:
```
print("Bienvenido al menú interactivo")
while(True):
    print("""¿Qué quieres hacer? Escribe una opción
    1) Saludar
    2) Sumar dos números
    3) Salir""")
    opcion = input()
    if opcion == '1':
        print("Hola, espero que te lo estés pasando bien")
    elif opcion == '2':
        n1 = float(input("Introduce el primer número: "))
        n2 = float(input("Introduce el segundo número: "))
        print("El resultado de la suma es: ",n1+n2)
    elif opcion =='3':
        print("¡Hasta luego! Ha sido un placer ayudarte")
        break
    else:
        print("Comando desconocido, vuelve a intentarlo")
```

```
Bienvenido al menú interactivo
¿Qué quieres hacer? Escribe una opción
    1) Saludar
    2) Sumar dos números
    3) Salir
1
Hola, espero que te lo estés pasando bien
¿Qué quieres hacer? Escribe una opción
    1) Saludar
    2) Sumar dos números
    3) Salir
2
Introduce el primer número: 10
Introduce el segundo número: 5
El resultado de la suma es:  15.0
¿Qué quieres hacer? Escribe una opción
    1) Saludar
    2) Sumar dos números
    3) Salir
kdjsk
Comando desconocido, vuelve a intentarlo
¿Qué quieres hacer? Escribe una opción
    1) Saludar
    2) Sumar dos números
    3) Salir
3
¡Hasta luego! Ha sido un placer ayudarte
```

Recorriendo los elementos de una lista utilizando While

In [1]:
```
numeros = [1,2,3,4,5,6,7,8,9,10]
indice = 0
while indice < len(numeros):
    print(numeros[indice])
    indice+=1
```

```
1
2
3
4
5
6
7
8
9
10
```

Sentencia For (Para) con listas

In [2]:
```
for numero in numeros:  # Para [variable] en [lista]
    print(numero)
```

```
1
2
3
4
5
6
7
8
9
10
```

Modificar ítems de la lista al vuelo

Para asignar un nuevo valor a los elementos de una lista mientras la recorremos, podríamos intentar asignar al número el nuevo valor:

In [3]:
```
for numero in numeros:
    numero *= 10
```

In [4]:
```
numeros
```

Out[4]: `[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`

Sin embargo, ésto no funciona. La forma correcta de hacerlo es haciendo referencia al índice de la lista en lugar de la variable:

In [5]:
```
indice = 0
numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
for numero in numeros:
    numeros[indice] *= 10
    indice+=1
numeros
```

Out[5]: [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

Podemos utilizar la función enumerate() para conseguir el índice y el valor en cada iteración fácilmente:

In [6]:
```
numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
for indice,numero in enumerate(numeros):
    numeros[indice] *= 10
numeros
```

Out[6]: [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

For con cadenas

In [7]:
```
cadena = "Hola amigos"
for caracter in cadena:
    print(caracter)
```

```
H
o
l
a

a
m
i
g
o
s
```

Pero debemos recordar que las cadenas son inmutables:

In [9]:
```
for i,c in enumerate(cadena):
    cadena[i] = "*"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-9-8ba888c46579> in <module>()
      1 for i,c in enumerate(cadena):
----> 2     cadena[i] = "*"

TypeError: 'str' object does not support item assignment
```

Sin embargo siempre podemos generar una nueva cadena:

In [13]:
```
cadena2 = ""
for caracter in cadena:
    cadena2 += caracter * 2
```

In [11]:
```
cadena
```

Out[11]:
```
'Hola amigos'
```

In [14]:
```
cadena2
```

Out[14]:
```
'HHoollaa  aammiiggooss'
```

La función range()

Sirve para generar una lista de números que podemos recorrer fácilmente, pero no ocupa memoria porque se interpreta sobre la marcha:

In [15]:
```
for i in range(10):
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```

In [16]:
```
range(10)
```

Out[16]:
```
range(0, 10)
```

In [17]:
```
for i in [0,1,2,3,4,5,6,7,9]:
    print(i)
```

```
0
1
2
3
4
5
6
7
9
```

Si queremos conseguir la lista literal podemos transformar el range a una lista:

In [18]:
```
list(range(10))
```

Out[18]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Ejemplo de cabecera

In [ ]:

```
n = 0
while n < 10:
    if (n % 2) == 0:
        print(n,'es un número par')
    else:
        print(n,'es un número impar')
    n = n + 1
```