😎 Ciclos o bucles

for

Un ciclo **for** es un ciclo que repite el bloque de instrucciones un número predeterminado de veces. El bloque de instrucciones que se repite se suele llamar cuerpo del ciclo y cada repetición se suele llamar iteración. La sintaxis de un ciclo for es la siguiente:

for variable in elemento_iterable (lista, cadena, range, etc.):

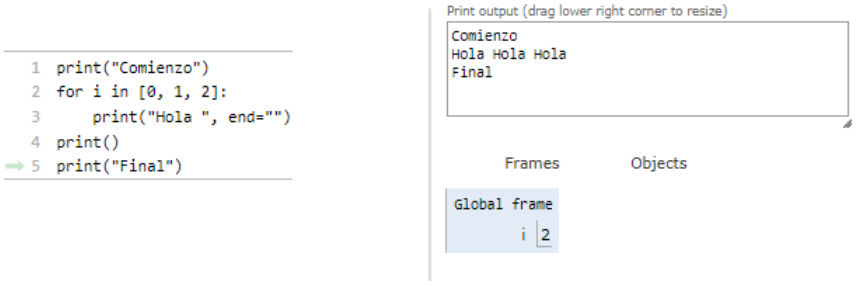
```
    cuerpo del ciclo
```

Ejemplos de ciclos for

1. En Python no es necesario definir la variable de control antes del ciclo, aunque se puede utilizar como variable de control una variable ya definida en el programa. El cuerpo del ciclo se ejecuta tantas veces como elementos tenga el elemento que se está recorriendo por ejemplo: elementos de una lista o de un range(), caracteres de una cadena, etc.). Por ejemplo:

In [1]:
```
print("Comienzo")
for i in [0, 1, 2]:
    print("Hola ", end="")
print()
print("Final")
```

```
Comienzo
Hola Hola Hola
Final
```



2. Si la lista está vacía, el ciclo no se ejecuta ninguna vez. Por ejemplo:

In [2]:
```
print("Comienzo")
for i in []:
    print("Hola ", end="")
print()
print("Final")
```

```
Comienzo

Final
```

3. Si la variable de control no se va a utilizar en el cuerpo del ciclo, como en los ejemplos anteriores, se puede utilizar el guion (_) en vez de un nombre de variable. Esta notación no tiene ninguna consecuencia con respecto al funcionamiento del programa, pero sirve de ayuda a la persona que esté leyendo el código fuente, que así sabe que los valores no se van a utilizar. Por ejemplo:

In [3]:
```
print("Comienzo")
for _ in [0, 1, 2]:
    print("Hola ", end="")
print()
print("Final")
```

```
Comienzo
Hola Hola Hola
Final
```

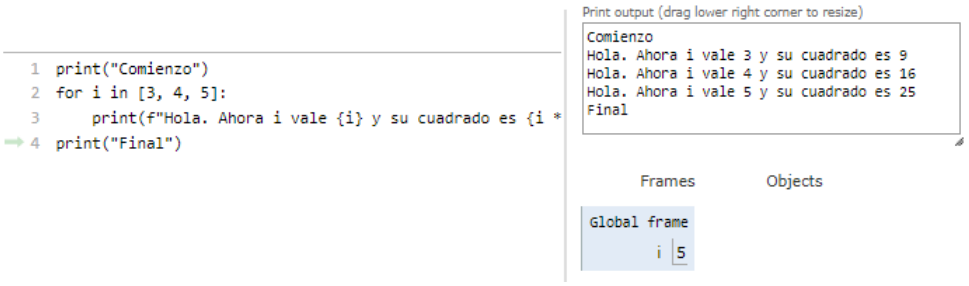
El indicador puede incluir cualquier número de guiones bajos (_, __, ___, ____, etc). Los más utilizados son uno o dos guiones (_ o __).

4. En el ejemplo anterior, la variable de control "i" no se utilizaba en el bloque de instrucciones, pero en muchos casos sí que se utiliza una variable. Cuando se utiliza, hay que tener en cuenta que la variable de control va tomando los valores del elemento que se recorre. Por

ejemplo:

In [4]:
```
print("Comienzo")
for i in [3, 4, 5]:
    print(f"Hola. Ahora i vale {i} y su cuadrado es {i ** 2}")
print("Final")
```

```
Comienzo
Hola. Ahora i vale 3 y su cuadrado es 9
Hola. Ahora i vale 4 y su cuadrado es 16
Hola. Ahora i vale 5 y su cuadrado es 25
Final
```



5. La lista puede contener cualquier tipo de elementos, no sólo números. El ciclo se repetirá siempre tantas veces como elementos tenga la lista y la variable irá tomando los valores de uno en uno. Por ejemplo:

In [5]:
```
print("Comienzo")
for i in ["Pepe", "Pepa", 47]:
    print(f"Hola. Ahora i vale {i}")
print("Final")
```

```
Comienzo
Hola. Ahora i vale Pepe
Hola. Ahora i vale Pepa
Hola. Ahora i vale 47
Final
```

6. La costumbre más extendida es utilizar la letra i como nombre de la variable de control, pero se puede utilizar cualquier otro nombre válido. Por ejemplo:

In [6]:
```
print("Comienzo")
for numero in [0, 1, 2, 3]:
    print(f"{numero} * {numero} = {numero ** 2}")
print("Final")
```

```
Comienzo
0 * 0 = 0
1 * 1 = 1
2 * 2 = 4
3 * 3 = 9
Final
```

7. La variable de control puede ser una variable empleada antes del ciclo. El valor que tuviese la variable no afecta a la ejecución del ciclo, pero cuando termina el ciclo, la variable de control conserva el último valor asignado:

In [7]:
```
i = 10
print(f"El ciclo no ha comenzado. Ahora i vale {i}")
for i in [0, 1, 2, 3, 4]:
    print(f"{i} * {i} = {i ** 2}")
print(f"El ciclo ha terminado. Ahora i vale {i}")
```

```
El ciclo no ha comenzado. Ahora i vale 10
0 * 0 = 0
1 * 1 = 1
2 * 2 = 4
3 * 3 = 9
4 * 4 = 16
El ciclo ha terminado. Ahora i vale 4
```

8. Cuando se escriben 2 o más ciclos seguidos, la costumbre es utilizar el mismo nombre de variable puesto que cada ciclo establece los valores de la variable sin importar los valores anteriores:

In [8]:
```
for i in [0, 1, 2]:
    print(f"{i} * {i} = {i ** 2}")
print()
```

```
0 * 0 = 0
1 * 1 = 1
2 * 2 = 4
```

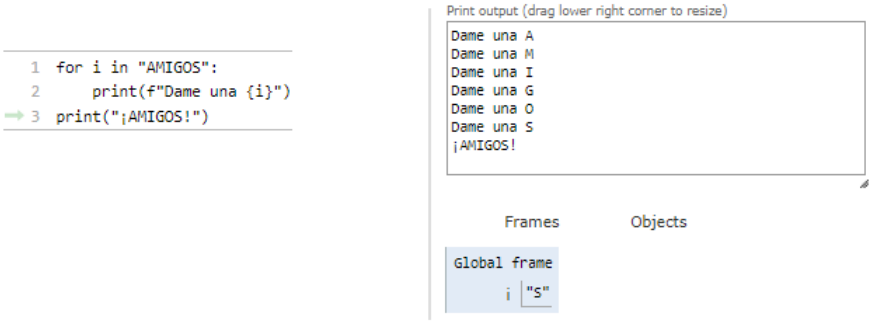
In [9]:
```
for i in [0, 1, 2, 3]:
    print(f"{i} * {i} * {i} = {i ** 3}")
```

```
0 * 0 * 0 = 0
1 * 1 * 1 = 1
2 * 2 * 2 = 8
3 * 3 * 3 = 27
```

8. Cuando se escriben 2 o más ciclos seguidos, la costumbre es utilizar el mismo nombre de variable puesto que cada ciclo establece los valores de la variable sin importar los valores anteriores:

In [10]:
```
for i in "AMIGOS":
    print(f"Dame una {i}")
print("¡AMIGOS!")
```

```
Dame una A
Dame una M
Dame una I
Dame una G
Dame una O
Dame una S
¡AMIGOS!
```



10. En los ejemplos anteriores se ha utilizado una lista para facilitar la comprensión del funcionamiento de los ciclos pero, -si es posible hacerlo-, se recomienda utilizar tipos range(), entre otros motivos porque durante la ejecución del programa ocupan menos memoria. El siguiente programa es equivalente al programa del ejemplo anterior:

In [11]:
```
print("Comienzo")
for i in range(3):
    print("Hola ", end="")
print()
print("Final")
```

```
Comienzo
Hola Hola Hola
Final
```

11. Otra de las ventajas de utilizar tipos range() es que el argumento del tipo range() controla el número de veces que se ejecuta el ciclo. En el ejemplo anterior era suficiente cambiar el argumento para que el programa salude muchas más veces.

In [12]:
```
print("Comienzo")
for i in range(0,10,2):
    print("Hola ", end="")
print()
print("Final")
```

```
Comienzo
Hola Hola Hola Hola Hola
Final
```

Esto permite que el número de iteraciones dependa del desarrollo del programa.

12. En el ejemplo siguiente es el usuario quien decide cuántas veces se ejecuta el ciclo:

In [13]:
```
veces = int(input("¿Cuántas veces quiere que le salude? "))
for i in range(veces):
    print("Hola ", end="")
print()
print("Adiós")
```

```
¿Cuántas veces quiere que le salude? 3
Hola Hola Hola
Adiós
```

Contadores, variables testigo (o bandera) y acumuladores

En muchos programas se necesitan variables que cuenten cuántas veces ha ocurrido algo (contadores) o que indiquen si simplemente ha ocurrido algo (testigo) o que acumulen valores (acumuladores).

Contador

Se entiende por contador una variable que lleva la cuenta del número de veces que se ha cumplido una condición. El ejemplo siguiente es un ejemplo de programa con contador (en este caso, la variable que hace de contador es la variable cuenta):

In [14]:
```
print("Comienzo")
cuenta = 0
for i in range(1, 6):
    if i % 2 == 0:
        cuenta = cuenta + 1
#         cuenta++ no se puede usar
print(f"Desde 1 hasta 5 hay {cuenta} múltiplos de 2")
```

```
Comienzo
Desde 1 hasta 5 hay 2 múltiplos de 2
```

Importante:

- **En cada iteración, el programa comprueba si i es múltiplo de 2.**
- **El contador se modifica sólo si la variable de control i es múltiplo de 2.**
- **El contador va aumentando de uno en uno.**
- **Antes del ciclo se debe inicializar el contador (en este caso, 0)**

Variables testigo o bandera

Se entiende por testigo una variable booleana que indica simplemente si una condición se ha cumplido o no. Es un caso particular de contador, pero se suele hacer con variables lógicas en vez de numéricas (en este caso, la variable que hace de testigo es la variable encontrado):

In [15]:
```
print("Comienzo")
encontrado = False
for i in range(1, 6):
    if i % 6 == 0:
        encontrado = True
if encontrado:
    print(f"Entre 1 y 5 hay al menos un múltiplo de 2.")
else:
    print(f"Entre 1 y 5 no hay ningún múltiplo de 2.")
```

```
Comienzo
Entre 1 y 5 no hay ningún múltiplo de 2.
```

Importante:

- **En cada iteración, el programa comprueba si i es múltiplo de 2.**
- **El testigo se modifica la primera vez que la variable de control i es múltiplo de 2.**
- **El testigo no cambia una vez ha cambiado.**
- **Antes del ciclo se debe dar un valor inicial al testigo (en este caso, False)**

Acumulador

Se entiende por acumulador una variable que acumula el resultado de una operación. El ejemplo siguiente es un ejemplo de programa con acumulador (en este caso, la variable que hace de acumulador es la variable suma):

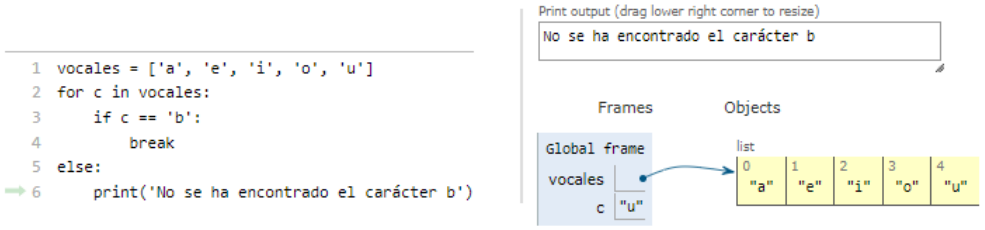
In [16]:
```
print("Comienzo")
suma = 0
for i in [1, 2, 3, 4]:
    suma = suma + i
print(f"La suma de los números de 1 a 4 es {suma}")
```

```
Comienzo
La suma de los números de 1 a 4 es 10
```

for con else

In [17]:
```
vocales = ['a', 'e', 'i', 'o', 'u']
for c in vocales:
    if c == 'b':
        break
else:
    print('No se ha encontrado el carácter b')
```

```
No se ha encontrado el carácter b
```



Ciclos anidados

Se habla de ciclos anidados cuando un ciclo se encuentra en el bloque de instrucciones de otro bloque. Al ciclo que se encuentra dentro del otro se le puede denominar ciclo interior o ciclo interno. El otro ciclo sería el ciclo exterior o ciclo externo. Los ciclos pueden tener cualquier nivel de anidamiento (un ciclo dentro de otro ciclo dentro de un tercero, etc.).

Ciclos anidados (variables independientes)

Se dice que las variables de los ciclos son independientes cuando los valores que toma la variable de control del ciclo interno no dependen del valor de la variable de control del ciclo externo. Por ejemplo:

In [18]:
```
for i in [0, 1, 2]:
    for j in [0, 1]:
        print(f"i vale {i} y j vale {j}")
```

```
i vale 0 y j vale 0
i vale 0 y j vale 1
i vale 1 y j vale 0
i vale 1 y j vale 1
i vale 2 y j vale 0
i vale 2 y j vale 1
```

En el ejemplo anterior, el ciclo externo (el controlado por i) se ejecuta 3 veces y el ciclo interno (el controlado por j) se ejecuta 2 veces por cada valor de i. Por ello la instrucción print() se ejecuta en total 6 veces (3 veces que se ejecuta el ciclo externo x 2 veces que se ejecuta cada vez el ciclo interno = 6 veces). El siguiente programa es equivalente al ejemplo anterior:

In [19]:
```
for i in range(3):
    for j in range(2):
        print(f"i vale {i} y j vale {j}")
```

```
i vale 0 y j vale 0
i vale 0 y j vale 1
i vale 1 y j vale 0
i vale 1 y j vale 1
i vale 2 y j vale 0
i vale 2 y j vale 1
```

Al escribir ciclos anidados, hay que prestar atención a la **indentación** de las instrucciones, ya que indica a Python si una instrucción forma parte de un bloque u otro. En los tres siguientes programas la única diferencia es el sangrado de la última instrucción:

1. En este caso, la última instrucción forma parte del cuerpo del ciclo interno. Por tanto el valor de i se escribe cada vez que se ejecuta el ciclo interno.

In [20]:
```
for i in [1, 2, 3]:
    for j in [11, 12]:
        print(i, end=" ")
        print(j, end=" ")
```

```
1 11 1 12 2 11 2 12 3 11 3 12
```

2. En este caso, la última instrucción forma parte del cuerpo del ciclo externo, pero no del interno. Por tanto el valor de i se escribe cada vez que se ha terminado de ejecutar el ciclo interno.

In [21]:
```
for i in [1, 2, 3]:
    for j in [11, 12]:
        print(j, end=" ")
    print(i, end=" ")
```

```
11 12 1 11 12 2 11 12 3
```

```
3. En este caso, la última instrucción no forma parte de ningún ciclo. Por tanto el valor de i se escribe una sola vez, al terminarse de ejecutar el ciclo externo.
```

In [22]:
```
for i in [1, 2, 3]:
    for j in [11, 12]:
        print(j, end=" ")
print(i, end=" ")
```

```
11 12 11 12 11 12 3
```

La costumbre más extendida es utilizar la letra "i" como nombre de la variable de control del ciclo externo y la letra "j" como nombre de la variable de control del ciclo interno (o "k" si hay un tercer nivel de anidamiento), pero se puede utilizar cualquier otro nombre válido.

4. En Python se puede incluso utilizar la misma variable en los dos ciclos anidados porque Python las trata como si fueran dos variables distintas. Por ejemplo:

In [23]:
```
for i in range(3):
    print(f"i (externa) vale {i}")
    for i in range(2):
        print(f"i (interna) vale {i}")
```

```
i (externa) vale 0
i (interna) vale 0
i (interna) vale 1
i (externa) vale 1
i (interna) vale 0
i (interna) vale 1
i (externa) vale 2
i (interna) vale 0
i (interna) vale 1
```

De todas formas, este uso no se recomienda porque da lugar a programas difíciles de leer y además no es habitual en otros lenguajes de programación. Se aconseja utilizar siempre nombres de variables distintos.

Ciclos anidados (variables dependientes)

Se dice que las variables de los ciclos son dependientes cuando los valores que toma la variable de control del ciclo interno dependen del valor de la variable de control del ciclo externo. Por ejemplo:

In [24]:
```
for i in [1, 2, 3]:
    for j in range(i):
        print(f"i vale {i} y j vale {j}")
```

```
i vale 1 y j vale 0
i vale 2 y j vale 0
i vale 2 y j vale 1
i vale 3 y j vale 0
i vale 3 y j vale 1
i vale 3 y j vale 2
```

En el ejemplo anterior, el ciclo externo (el controlado por i) se ejecuta 3 veces y el ciclo interno (el controlado por j) se ejecuta 1, 2 y 3 veces. Por ello la instrucción print() se ejecuta en total 6 veces.

La variable i toma los valores de 1 a 3 y la variable j toma los valores de 0 a i, por lo que cada vez el ciclo interno se ejecuta un número diferente de veces:

- **Cuando i vale 1, range(i) devuelve la lista [0] y por tanto el ciclo interno se ejecuta una sola vez y el programa escribe una sola línea en la que i vale 1 (y j vale 0).**
- **Cuando i vale 2, range(i) devuelve la lista [0, 1] y por tanto el ciclo interno se ejecuta dos veces y el programa escribe dos líneas en la que i vale 2 (y j vale 0 o 1 en cada una de ellas).**
- **Cuando i vale 3, range(i) devuelve la lista [0, 1, 2] y por tanto el ciclo interno se ejecuta tres veces y el programa escribe tres líneas en la que i vale 3 (y j vale 0, 1 o 2 en cada una de ellas).**

while

Un ciclo while permite repetir la ejecución de un grupo de instrucciones mientras se cumpla una condición (es decir, mientras la condición tenga el valor True). La sintaxis del ciclo while es la siguiente:

while condicion:

```
    cuerpo del ciclo
```

La ejecución de esta estructura de control while es la siguiente:
Python evalúa la condición:

> **si el resultado es True se ejecuta el cuerpo del ciclo. Una vez ejecutado el cuerpo del ciclo, se repite el proceso (se evalúa de nuevo la condición y, si es verdadera, se ejecuta de nuevo el cuerpo del ciclo) una y otra vez mientras la condición sea verdadera.**

> **si el resultado es False, el cuerpo del ciclo no se ejecuta y continúa la ejecución del resto del programa.**

La variable o las variables que aparezcan en la condición se llaman **variables de control**. Las variables de control deben definirse antes del ciclo while y modificarse en el ciclo while.

1. Por ejemplo, el siguiente programa escribe los números del 1 al 3:

In [25]:
```
i = 1
while i <= 3:
    print(i)
    i += 1
print("Programa terminado")
```

```
1
2
3
Programa terminado
```

2. El ejemplo anterior se podría haber programado con un ciclo for. La ventaja de un ciclo while es que la variable de control se puede modificar con mayor flexibilidad, como en el ejemplo siguiente:

In [26]:
```
i = 1
while i <= 50:
    print(i)
    i = 3 * i + 1
print("Programa terminado")
```

```
1
4
13
40
Programa terminado
```

3. Otra ventaja del ciclo while es que el número de iteraciones no está definida antes de empezar el ciclo, por ejemplo porque los datos los proporciona el usuario. El ejemplo siguiente ejemplo pide un número positivo al usuario una y otra vez hasta que el usuario lo haga correctamente:

In [27]:
```python
numero = int(input("Escriba un número positivo: "))
while numero < 0:
    print("¡Ha escrito un número negativo! Inténtelo de nuevo")
    numero = int(input("Escriba un número positivo: "))
print("Gracias por su colaboración")
```

```
Escriba un número positivo: -3
¡Ha escrito un número negativo! Inténtelo de nuevo
Escriba un número positivo: 4
Gracias por su colaboración
```

Ciclos infinitos

Si la condición del ciclo se cumple siempre, el ciclo no terminará nunca de ejecutarse y tendremos lo que se denomina un **ciclo infinito**, normalmente se deben a errores que se deben corregir.

Los ciclos infinitos no intencionados deben evitarse pues significan perder el control del programa.

Para interrumpir un ciclo infinito, hay que pulsar la combinación de teclas **Ctrl+C**. Al interrumpir un programa se mostrará un mensaje de error similar a éste:

Traceback (most recent call last):

File "ejemplo.py", line 3, in print(i)
KeyboardInterrupt

Estos algunos ejemplos de ciclos infinitos:

1. El programador ha olvidado modificar la variable de control dentro del ciclo y el programa imprimirá números 1 indefinidamente:

In [28]:
```python
i = 1
while i <= 10:
    print(i, end=" ")
#     i=i+1
```

```
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

2. El programador ha escrito una condición que se cumplirá siempre y el programa imprimirá números consecutivos indefinidamente:

In [29]:
```python
i = 1
while i > 0:
    print(i, end=" ")
    i += 1
```

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77
78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110
111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 13
8 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165
166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 19
3 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220
221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 24
8 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275
276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 30
3 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330
331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 35
8 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385
386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 41
3 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440
441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 46
8 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495
496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 52
3 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550
```

3. Se aconseja expresar las condiciones como desigualdades en vez de comparar valores. En el ejemplo siguiente, el programador ha escrito una condición que se cumplirá siempre y el programa imprimirá números consecutivos indefinidamente:

In [30]:
```
i = 1
while i != 100:
    print(i, end=" ")
    i += 2
```

```
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59 61 63 65 67 69 71 73 75 7
7 79 81 83 85 87 89 91 93 95 97 99 101 103 105 107 109 111 113 115 117 119 121 123 125 127 129 131 133 135 137
139 141 143 145 147 149 151 153 155 157 159 161 163 165 167 169 171 173 175 177 179 181 183 185 187 189 191 19
3 195 197 199 201 203 205 207 209 211 213 215 217 219 221 223 225 227 229 231 233 235 237 239 241 243 245 247
249 251 253 255 257 259 261 263 265 267 269 271 273 275 277 279 281 283 285 287 289 291 293 295 297 299 301 30
3 305 307 309 311 313 315 317 319 321 323 325 327 329 331 333 335 337 339 341 343 345 347 349 351 353 355 357
359 361 363 365 367 369 371 373 375 377 379 381 383 385 387 389 391 393 395 397 399 401 403 405 407 409 411 41
3 415 417 419 421 423 425 427 429 431 433 435 437 439 441 443 445 447 449 451 453 455 457 459 461 463 465 467
469 471 473 475 477 479 481 483 485 487 489 491 493 495 497 499 501 503 505 507 509 511 513 515 517 519 521 52
3 525 527 529 531 533 535 537 539 541 543 545 547 549 551 553 555 557 559 561 563 565 567 569 571 573 575 577
579 581 583 585 587 589 591 593 595 597 599 601 603 605 607 609 611 613 615 617 619 621 623 625 627 629 631 63
3 635 637 639 641 643 645 647 649 651 653 655 657 659 661 663 665 667 669 671 673 675 677 679 681 683 685 687
689 691 693 695 697 699 701 703 705 707 709 711 713 715 717 719 721 723 725 727 729 731 733 735 737 739 741 74
3 745 747 749 751 753 755 757 759 761 763 765 767 769 771 773 775 777 779 781 783 785 787 789 791 793 795 797
799 801 803 805 807 809 811 813 815 817 819 821 823 825 827 829 831 833 835 837 839 841 843 845 847 849 851 85
3 855 857 859 861 863 865 867 869 871 873 875 877 879 881 883 885 887 889 891 893 895 897 899 901 903 905 907
909 911 913 915 917 919 921 923 925 927 929 931 933 935 937 939 941 943 945 947 949 951 953 955 957 959 961 96
3 965 967 969 971 973 975 977 979 981 983 985 987 989 991 993 995 997 999 1001 1003 1005 1007 1009 1011 1013 1
015 1017 1019 1021 1023 1025 1027 1029 1031 1033 1035 1037 1039 1041 1043 1045 1047 1049 1051 1053 1055 1057 1
```

In [ ]:
```
```