

# Parte 1. Fundamentos de la programación

Unai Pérez-Goya  
Área de Lenguajes y Sistemas Informáticos

Curso 2024/2025

# Parte 1.

## Fundamentos de la programación

### Sesión VI: Estructuras Iterativas

# Tabla de contenidos

Repaso

Programas iterativos

Estructura for

Estructura while

Ejercicios

# Repaso

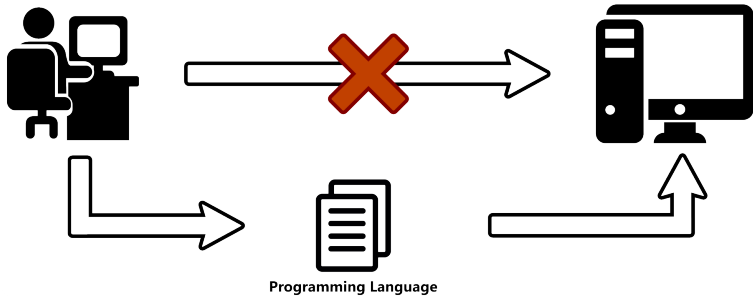
# Sesión I

## Lista de contenidos

1. Introducción a Python. Historia y características;
2. Lenguajes de programación. Concepto y uso;
3. Lenguajes interpretados vs. compilados. Programas ejecutables. Máquina intérprete;
4. Entorno de desarrollo integrado (IDE)

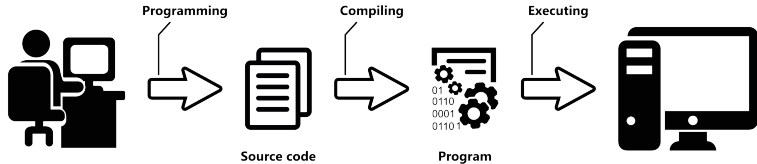
# Sesión I

## Lenguajes de programación



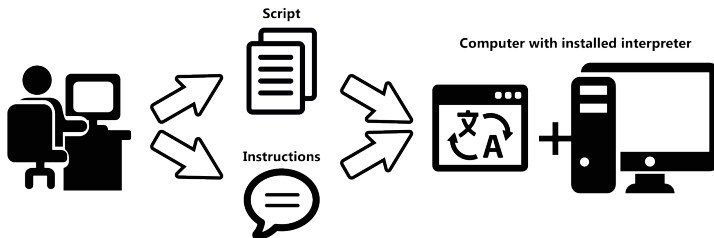
# Sesión I

## Proceso de compilación



# Sesión I

## Compilado vs. Interpretado



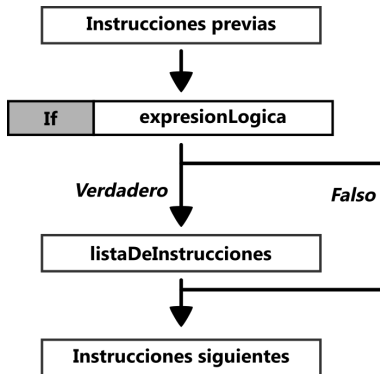
# Contenido de la sesión II

- ▶ Variables y uso básico en python
- ▶ Declaración y asignación;
- ▶ Tipos
  - ▶ Numéricos
  - ▶ Texto
  - ▶ Booleanos
- ▶ Constantes

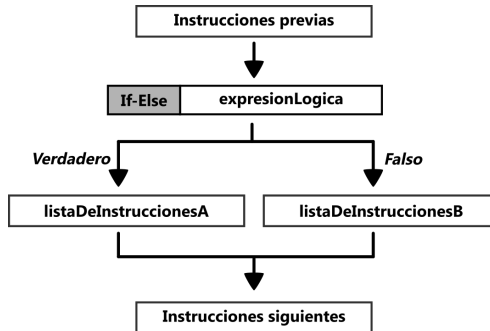
# Sesión III

- ▶ Estructura de los programas
- ▶ Estructuras de control
  1. Estructura `if`
  2. Estructura `if-else`
- ▶ Funciones, capsulación y ámbito de la variable
- ▶ Ejercicios.

# Sesión III



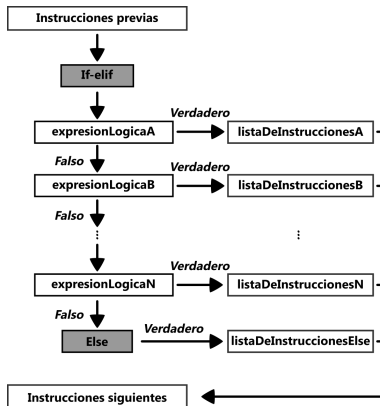
# Sesión III



# Sesión IV

- ▶ Interacción con el usuario;
- ▶ Estructura `if-elif`.
- ▶ Estructura `if-elif-else`.
- ▶ Funciones sobre variables de tipo `str`

# Sesión IV



# Contenido de la sesión V

- ▶ Datos estructurados
- ▶ Uso de `list`
- ▶ Recorrer listas con `for [each]`
- ▶ Ejercicios

# Resumen

Repaso

Programas iterativos

Estructura `for`

Estructura `while`

Ejercicios

# Programas iterativos

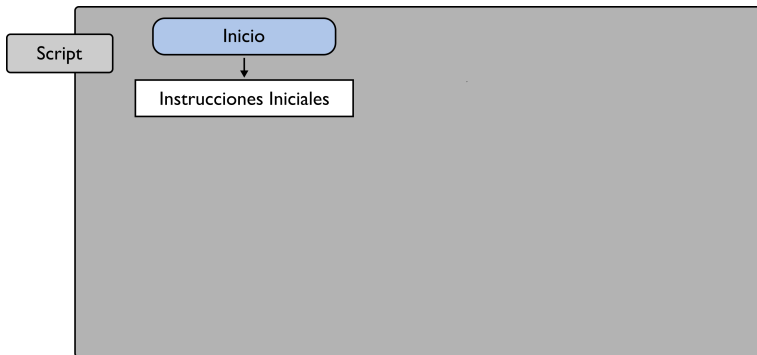


# Comparación con los programas vistos anteriormente

- ▶ Las estructuras alternativas están muy limitadas para muchos trabajos.
- **Ejemplo:** Cuenta atrás. Diseñar un programa que lea un número entero desde el teclado y realice la cuenta atrás hasta llegar al 0;
- **Ejemplo:** Leer dos números del teclado y multiplicarlos sin utilizar el operador `*`.

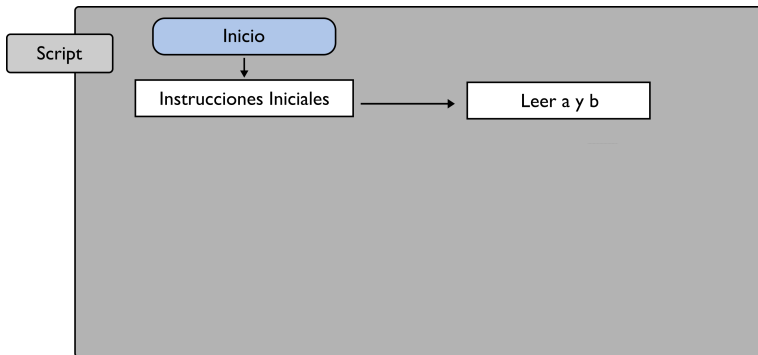
# Programas iterativos

- **Ejemplo:** Leer dos números del teclado y multiplicarlos sin utilizar el operador  $*$ .



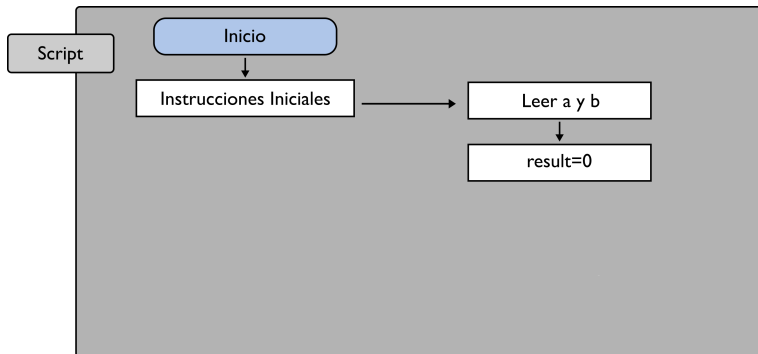
# Programas iterativos

- **Ejemplo:** Leer dos números del teclado y multiplicarlos sin utilizar el operador  $*$ .



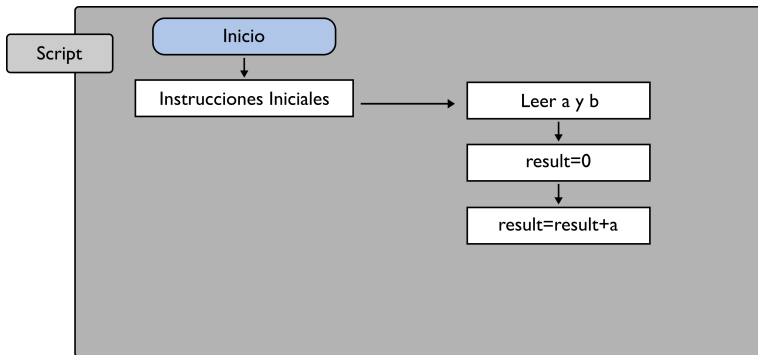
# Programas iterativos

- **Ejemplo:** Leer dos números del teclado y multiplicarlos sin utilizar el operador  $*$ .



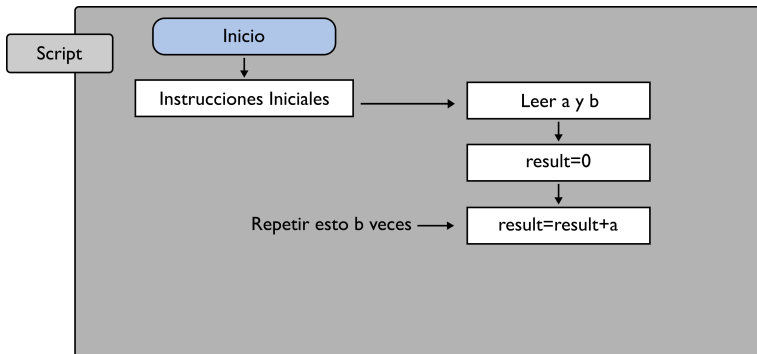
# Programas iterativos

- **Ejemplo:** Leer dos números del teclado y multiplicarlos sin utilizar el operador  $*$ .



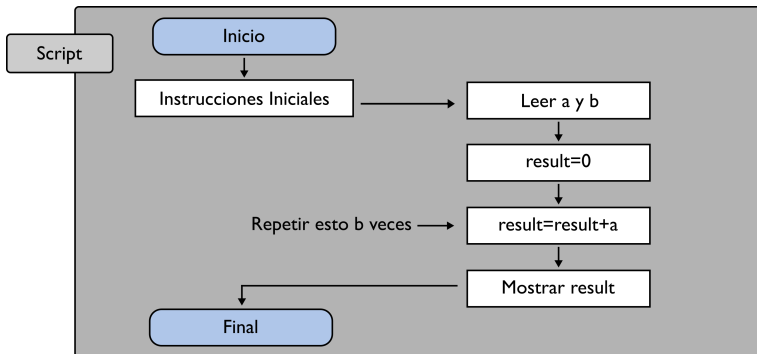
# Programas iterativos

- **Ejemplo:** Leer dos números del teclado y multiplicarlos sin utilizar el operador  $\star$ .



# Programas iterativos

- **Ejemplo:** Leer dos números del teclado y multiplicarlos sin utilizar el operador  $*$ .



# Programas iterativos

- ▶ Crearemos los programas con dos estructuras:
  - a) Conociendo de antemano el número de iteraciones a realizar;
  - b) Uniendo la repetición de las instrucciones a una expresión lógica;
- ▶ Las estructuras iterativas permiten ejecutar unas instrucciones tantas como queremos.
- ▶ De una manera, consiste en repetir la ejecución hasta conseguir el resultado deseado.
- ▶ Las dos estructuras que utilizaremos son:
  1. Estructura `for`;
  2. Estructura `while`;

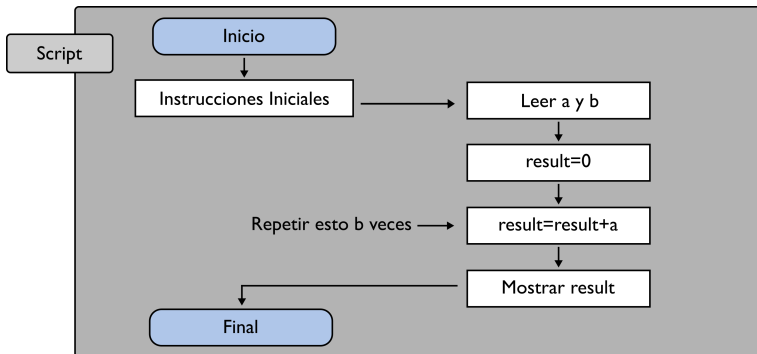
# Estructura **for**

# Repeticiones numeradas

- ▶ A veces tenemos un listado de instrucciones que queremos repetir.
- ▶ Antes de llegar a esa parte de código, sabemos el número de veces que deberíamos repetir la ejecución.
- **Ejemplo:** Leer dos números del teclado y multiplicarlos sin utilizar el operador `*`.

# Programas iterativos

- **Ejemplo:** Leer dos números del teclado y multiplicarlos sin utilizar el operador  $*$ .



# Repeticiones de un número determinado

- ▶ En informática se denomina a esta estructura de repetición `for`.
  - ▶ Aparece en la mayoría de los lenguajes de programación modernos, si no en todos.
- ▶ Se basa en la idea de un `contador`:
  - ▶ Es decir, una variable `cont` almacena el número de veces que se ejecutará el código.
  - ▶ En python, la estructura clásica de “`for`” no existe, pero se puede simular su funcionamiento con la función `range` .

# Estructura `for`

- ▶ En realidad, la función `range` genera una secuencia en un rango definido por el usuario.
- ▶ Su sintaxi es la siguiente:

- 👉 `range(start, stop, step)`
  - ▶ `start`: primer elemento de la secuencia.
  - ▶ `smallcodestop`: último elemento de la secuencia.
  - ▶ `step` en cuanto se incrementará cada elemento. Por defecto, 1.
  - ▶ Si `step=1` entonces el último elemento de la secuencia será `stop-1`.
- ▶ Si se ejecuta `range` con un único elemento, se definirá `start, step=0, 1` por defecto.

# Estructura `for`

- ▶ La estructura `for` se puede combinar con `range` para utilizarla como en cualquier otro lenguaje de programación.
- ▶ Ejecutará las instrucciones tantas veces como elementos en la secuencia generada por `range`:

```
 for x in range(start, stop):  
    instrucciones
```

- ▶ Como funciona:
  - ▶ en la primera iteración `x` tomará el valor de `start`.
  - ▶ Cada vez que se ejecute `instrucciones`, se ejecutará `x+step`.
  - ▶ Cuando `x` ten el mismo valor o mayor que `stop`, se detendrán las iteraciones.

# Estructura `for`

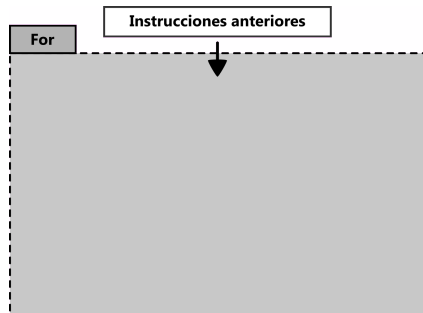


```
[Instrucciones  
anteriores]  
for x in range(start,  
stop):  
    instrucciones  
[Siguientes  
instrucciones]
```

**Instrucciones anteriores**

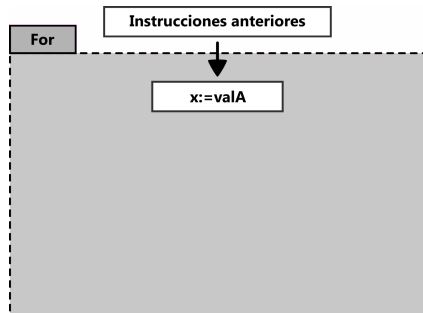
# Estructura for

👉 [Instrucciones  
anteriores]  
for x in range(start,  
stop):  
 instrucciones  
[Siguientes  
instrucciones]



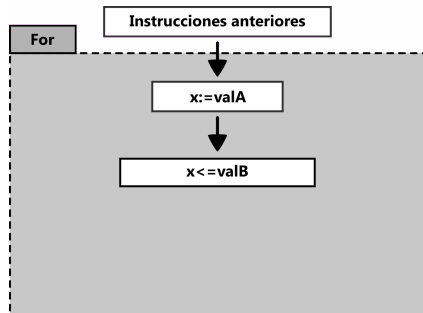
# Estructura for

👉 [Instrucciones  
anteriores]  
for x in range(start,  
stop):  
 instrucciones  
[Siguientes  
instrucciones]



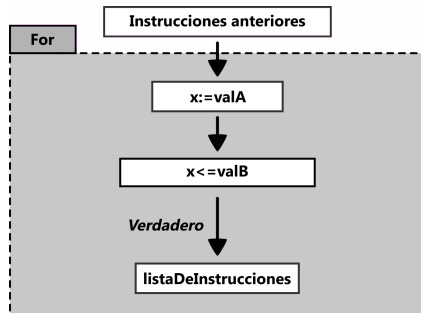
# Estructura for

👉 [Instrucciones  
anteriores]  
for x in range(start,  
stop):  
 instrucciones  
[Siguientes  
instrucciones]



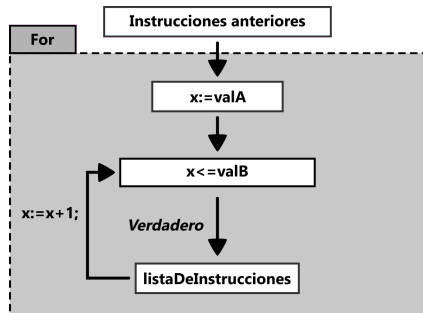
# Estructura for

👉 [Instrucciones  
anteriores]  
for x in range(start,  
stop):  
 instrucciones  
[Siguientes  
instrucciones]



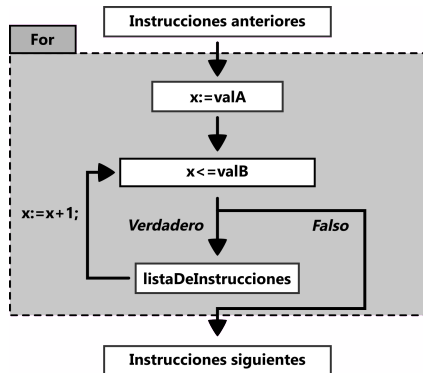
# Estructura for

👉 [Instrucciones  
anteriores]  
for x in range(start,  
stop):  
    instrucciones  
[Siguientes  
instrucciones]



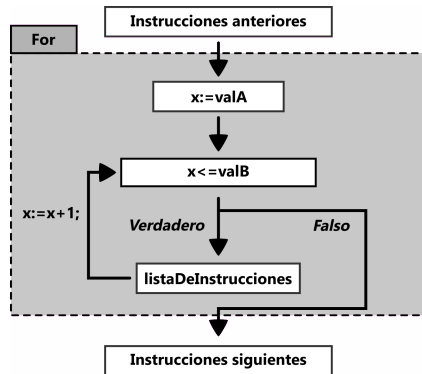
# Estructura for

👉 [Instrucciones  
anteriores]  
for x in range(start,  
stop):  
 instrucciones  
[Siguietes  
instrucciones]



# Estructura for

- Puede ocurrir que *instrucciones* no se ejecute nunca;
- Es necesario saber el número de veces que se van a ejecutar las instrucciones (*stop-start+1*, si no se cambia *step*).



# Estructura `for`

**P:** Entoces, ¿cómo puedo definir un rango?

**R:** Para empezar sólo con número enteros;

**R:** Más adelante podréis utilizar todo su pontencial con todas sus opciones.

**P:** Y, ¿Qué puedo poner en `instrucciones`?

- ▶ Cualquier cosa, incluidas todas las estructuras dadas hasta ahora como `for` eta/edo `while`.
- ▶ Igual que con todas las estructuras, **todas las instrucciones que se quieran repetir deben ir indentadas.**

# Ejemplo 1

- ★ **Ejemplo 1** Crea un programa en el que se visualizen los números desde 1 hasta un número escrito por el usuario.

# Escribe una secuencia de números

Ejemplo: Contador

```
1 # declarar variables
2 texto = input('Escribe un numero: ')
3 numero = int(texto)
4
5 for x in range(1,numero):
6     print(numero)
```

Código: Ejemplo:Contador.

## Ejemplo 2

- ★ **Ejemplo 2** Escribe el programa que muestre numeros desde 3 hasta un número escrito por el usuario.

# Escribe una secuencia de números

Ejemplo: Contador

```
1 # declarar variables
2 texto = input('Escribe un numero: ')
3 numero = int(texto)
4
5 for x in range(3,numero):
6     print(numero)
```

Código: Ejemplo:Contador.

## Ejemplo 3

- ★ **Ejemplo 3** Escribe un programa que realice un cuenta atrás desde un número escrito por el usuario hasta 0.

# Escribe una secuencia de números

Ejemplo: Contador

```
1 # declarar variables
2 texto = input('Escribe un numero: ')
3 numero = int(texto)
4
5 for x in range(numero,0,-1):
6     print(numero)
```

Código: Ejemplo:Contador.

## Ejemplo 4

- ★ **Ejemplo 4** Escribe un programa que visualice los números impares desde 1 hasta un número escrito por el usuario.

# Escribe una secuencia de números

Ejemplo: Contador

```
1 # declarar variables
2 texto = input('Escribe un numero: ')
3 numero = int(texto)
4
5 for x in range(1,numero,2):
6     print(numero)
```

Código: Ejemplo:Contador.

# Ejemplo 5

- ★ **Ejemplo 5** Escribe el programa que pida 5 números al usuario y los muestre por pantalla.

# Escribe una secuencia de números

Ejemplo: Contador

```
1 # declarar variables
2 numerok = list()
3
4 for x in range(1,5):
5     texto = input('Escribe un numero: ')
6     numeros = int(texto)
7     numeros.append(numero)
8
9 print(numeros)
```

Código: Ejemplo:Contador.

## Ejemplo 6

- ★ **Ejemplo 6** Escribe un programa que multiplique dos números solicitados al usuario.

*Nota:* El programa no puede utilizar el operador `*`.

# Biderketa kalkulatu

Ejemplo: biderketa

```
1 # declarar variables
2 texto = input('Escribe un numero: ')
3 a_numero = int(texto)
4 texto = input('Escribe un numero: ')
5 b_numero = int(texto)
6
7 resultado=0
8 for x in range(b):
9     resultado+=b_numero
10
11 print('Biderketaren resultado ',resultado, 'da')
```

Código: Ejemplo:biderketa.

# Estructura `while`

# Estructura `while`


- ▶ `while` es una estructura asociada a una expresión lógica. Mientras se cumpla la expresión lógica se ejecutará la lista de instrucciones:

```
👉 while expresion_logica:  
    instrucciones
```

- ▶ Mientras se cumpla `expresion_logica`, se ejecutará `instrucciones`.
  - ▶ En caso contrario saldrá de la estructura.

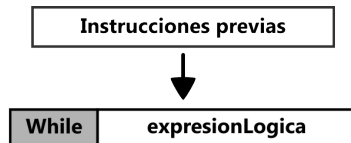
# Estructura while

## Instrucciones previas

 [Instrucciones anteriores]  
while expresion\_logica:  
 instrucciones  
[Siguietes instrucciones]

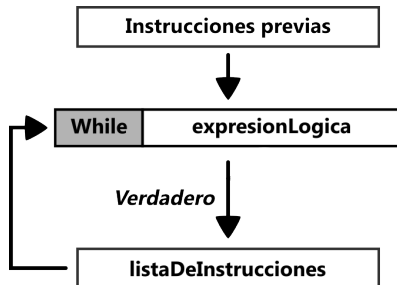
# Estructura while

👉 [Instrucciones  
anteriores]  
while expresion\_logica:  
 instrucciones  
[Siguietes  
instrucciones]



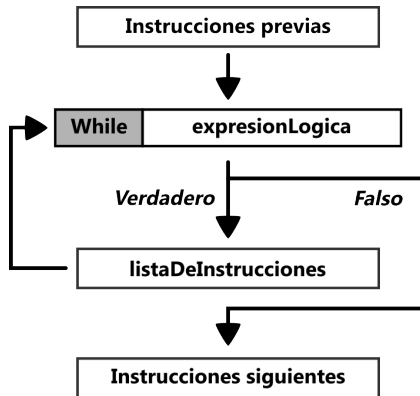
# Estructura while

👉 [Instrucciones anteriores]  
while expresion\_logica:  
 instrucciones  
[Siguietes instrucciones]



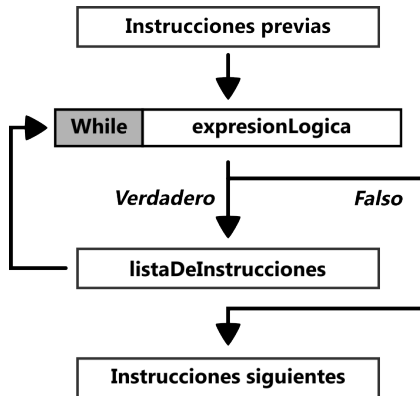
# Estructura while

👉 [Instrucciones anteriores]  
while expresion\_logica:  
 instrucciones  
[Siguietes instrucciones]



# Estructura while

- ▶ Puede pasar que `instrucciones` no se ejecute nunca;
- ▶ Puede pasar que `instrucciones` se ejecute eternamente;
- ▶ De modo que es necesario diseñar correctamente el programa para que `expresion logica` se deje de cumplir en algún momento.



# Estructura `while`

**P:** Entonces, que puedo poner en `expresion_logica`?

**R:** Un booleano o cualquier cosa que se evalúe como `True/False`.

**R:** Números, combinación de funciones, variables, operadores lógicos...

**P:** Y... que puedo poner en `instrucciones`?

- ▶ Cualquier cosa, nuevas estructuras iterativas, más `while`.
- ▶ Otra vez, **todas las instrucciones que se quieran repetir deben ir indentadas.**

# Ejemplo 1

- ★ **Ejemplo 1** Escribe un programa que requiera números enteros y positivos. Si el usuario introduce números negativos, el programa los solicitará una y otra vez hasta que el número introducido sea positivo.

# Escribe una secuencia de números

Ejemplo: Escribe una secuencia de números

```
1 # declarar variables
2 texto = input('Escribe un numero: ')
3 numero = int(texto)
4
5 while numero < 0:
6     texto = input('Escribe un numero: ')
7     numero = int(texto)
```

Código: Ejemplo: Escribe una secuencia de números.

## Ejemplo 2

- ★ **Ejemplo 2** Escribe el programa que pida al usuario un número entero impar. Si el usuario escribe un número par, el programa volverá a pedir un número.

# Obten un número impar

Ejemplo 2: pide un número impar

```
1 # declarar variables
2 texto = input('Escribe un numero: ')
3 numero = int(texto)
4
5 while numero%2==0:
6     texto = input('Escribe un numero: ')
7     numero = int(texto)
8
9 print('Lortutako zenbaki bakoitia ',numero)
```

Código: Ejemplo: Escribe una secuencia de números.

## Ejemplo 3

- ★ **Ejemplo 3** Idatzi zenbaki oso eta positibo bat eskatzen duen programa. Scripta 1-etik zenbaki horretaraino dauden zenbaki guztiak pantailaratu beharko ditu.

# crea una secuencia de números

Adibidea 2: crea una secuencia de números

```
1 # declarar variables
2 texto = input('Escribe un numero: ')
3 numero = int(texto)
4
5 kont=1
6 while kont<numero:
7     print(numero)
8     kont+=1
```

Código: Ejemplo: Escribe una secuencia de números.

# Ejemplo 4

## ★ Ejemplo 4

Crea un programa que pida un número entero y positivo. El script deberá visualizar todos los divisores de ese número.

# Zatitzaileak

## Ejemplo 4: divisores

```
1 # declarar variables
2 texto = input('Escribe un numero: ')
3 numero = int(texto)
4
5 kont=1
6 while kont<numero:
7     if numero%kont==0:
8         print(numero)
9     kont+=1
```

Código: Ejemplo: divisores.

## Ejemplo 5

- ★ **Ejemplo 5** Escribe un programa que pida números enteros hasta que el usuario introduzca 0. Cuando el usuario escriba el número 0, visualizará la suma de todos los números introducidos.

# Zatitzaileak

## Adibidea 5: Zenbakien sumatorio

- Zer dago gaizki ariketa honetan?

```
1 # declarar variables
2 texto = input('Escribe un numero: ')
3 numero = int(texto)
4
5 sumatorio=0
6 while not numero==0:
7     texto = input('Escribe un numero: ')
8     numero = int(texto)
9     sumatorio+=numero
```

Código: Ejemplo: Escribe una secuencia de números.

# Ejercicios

# Ejercicio 1:

- ★ **Ejercicio 1** Escribe una función que muestre números enteros entre 3 y 8 (incluidos 3 y 8). Escribe dos funciones, una utilizando la estructura `for` y otra utilizando la estructura `while`.

## Ejercicio 2:

- ★ **Ejercicio 2** Escribe un programa que pida desde el teclado dos números enteros `num1` y `num2` (los números deberán de cumplir `num1 <= num2`) y muestre todos los numeros en el rango `[num1.. num2]`. Los números deberán aparecer de pequeño a grande.

## Ejercicio 3:

- ★ **Ejercicio 3** Escribe un programa que pida desde el teclado dos números enteros `num1` y `num2` (los números deberán de cumplir `num1 <= num2`) y muestre todos los numeros en el rango `[num1.. num2]`. Los números deberán aparecer de grande a pequeño.

## Ejercicio 4:

- ★ **Ejercicio 4** Crea el programa que pida dos numeros enteros y positivos, los multiplique utilizando el operador `+`.

## Ejercicio 5:

- ★ **Ejercicio 5** Crea un programa que pida dos numeros enteros (ya sean positivos o negativos), y los multiplique utilizando el operador `+`.

## Ejercicio 6:

- ★ **Ejercicio 6** Escribe un programa que pida dos números positivos. Si el usuario introduce números negativos, el programa solicitará una asignación positiva alternativa a la misma.
  - ★ **Bonus:** Reescribe el programa para que como mucho pida 5 veces el número positivo. Si en la quinta ocasión la clave es negativa, la variable tomará el valor -1.

## Ejercicio 7:

- ★ **Ejercicio 7** Escribe un programa que pida al usuario un número entero y positivo e indique si se trata de un número primo (es decir, un número que sólo puede dividirse por 1 y por sí mismo).

# Resumen de la sesión

- ▶ Programas iterativos
- ▶ Uso de `for` con `range`
- ▶ Uso de `while`
- ▶ Ejercicios