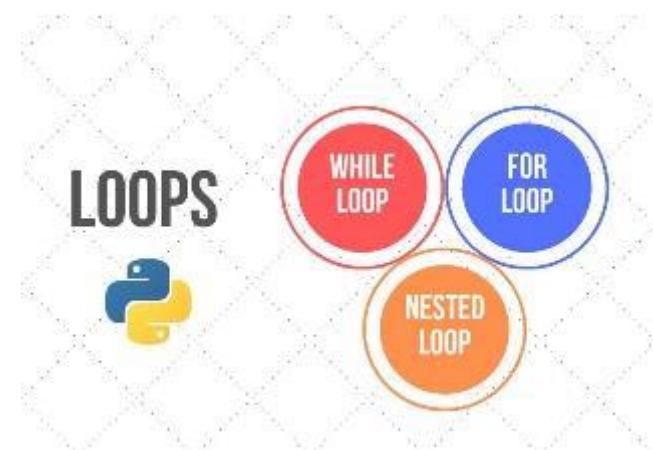
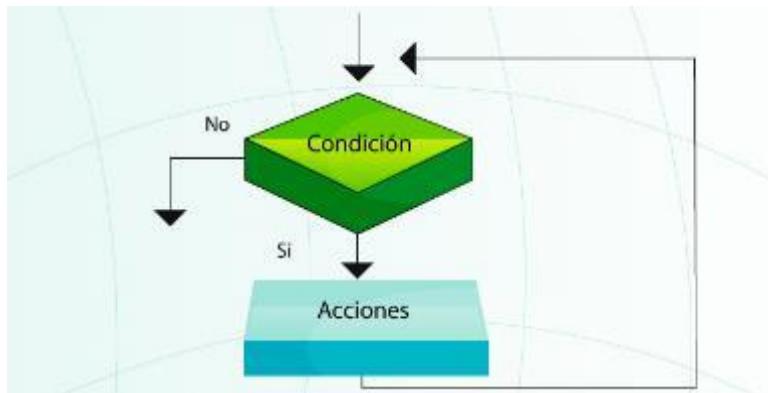




Estructuras de Repetición

Cátedra de Fundamentos de Programación



¿Qué veremos hoy?

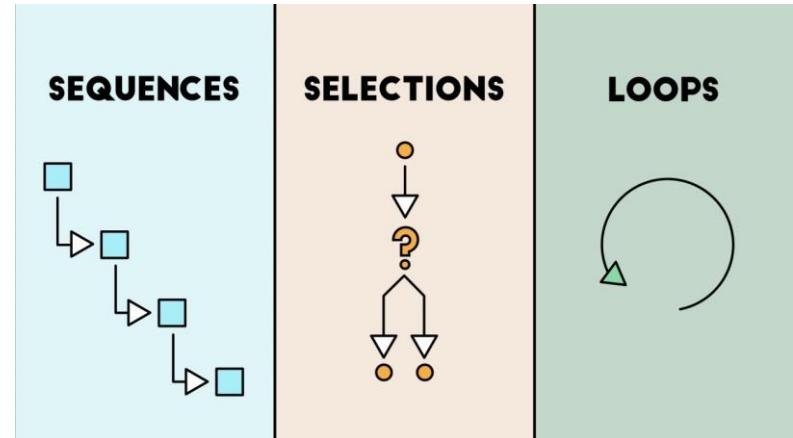
- Necesidad de estructuras de repetición
- Definición de ciclo/bucle e iteración

- Bucle **while**

- Bucle **for**

- Uso de las instrucciones **break** y **continue**

- Acumuladores, contadores y banderas.



Estructuras repetitivas

Hasta ahora se ha trabajado con instrucciones de entrada, salida, expresiones y operadores; asignaciones, instrucciones secuenciales y de selección. Hay una gran variedad de situaciones que requieren que una o varias instrucciones se repitan varias veces, ya sean cálculos u otro tipo de instrucciones. Las estructuras repetitivas abren la posibilidad de realizar una secuencia de instrucciones más de una vez.

Ejemplo: calcular las calificaciones de los 100 (o N) alumnos de una clase.

Puntaje	Calificación
Entre 90 y 100	5
Entre 80 y 89	4
Entre 70 y 79	3
Entre 60 y 69	2
Menos de 60	1

Estructuras repetitivas

Ejemplos de problemas que requieren algún tipo de repetición:

- Obtener la suma de una serie de números leídos del teclado.
- Calcular el promedio de edad de los alumnos de la clase de Algoritmo.
- Contar el número de veces de intentos fallidos para acceder a un sitio web.
- Determinar el máximo y el mínimo de una lista de números.
- etc, etc..

Estructuras repetitivas

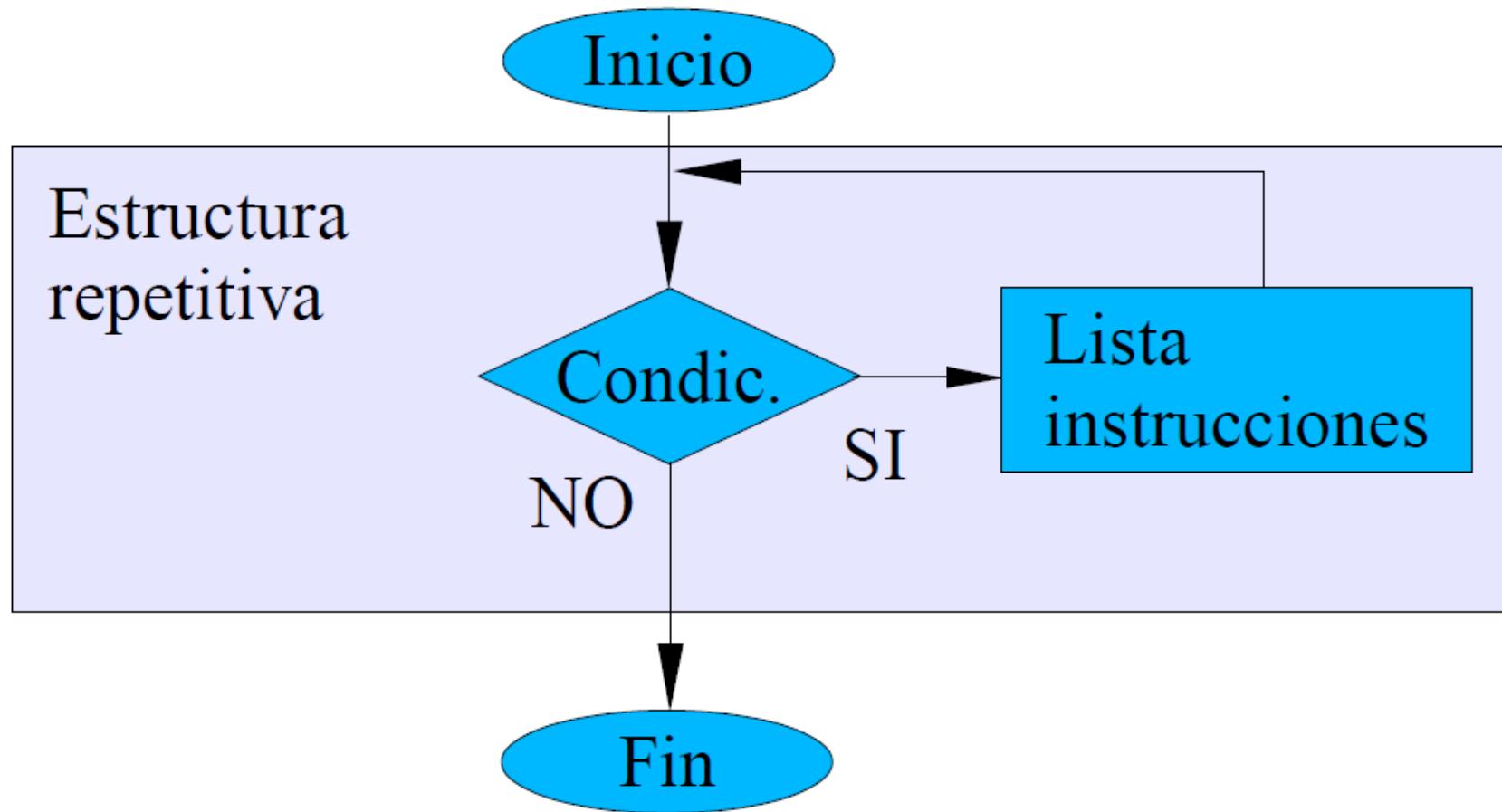
Por ejemplo, si queremos imprimir todos los valores que van de **N** a **N+4**:

```
N = int(input("Ingrese un numero: "))
print(N)
N=N+1
print(N)
N=N+1
print(N)
N=N+1
print(N)
N=N+1
print(N)
```

```
Ingrese un numero: 10
10
11
12
13
14
>>>
```

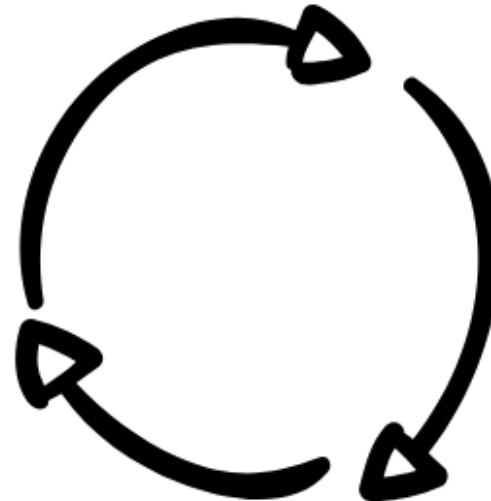
Estructuras repetitivas

Propósito: Repetir varias veces la ejecución de una serie de instrucciones bajo cierto criterio.



Estructuras repetitivas

- Mientras (**while**)
- Repetir (**do-while**)
- Desde (**for**)



Las estructuras que repiten una secuencia de instrucciones un número determinado de veces se denominan **bucles** o **lazos**.

Se denomina **iteración** al hecho de repetir la ejecución de una secuencia de acciones. La iteración se asocia a un número entero que indica el número de veces que se repite un trozo de código (o serie de instrucciones).

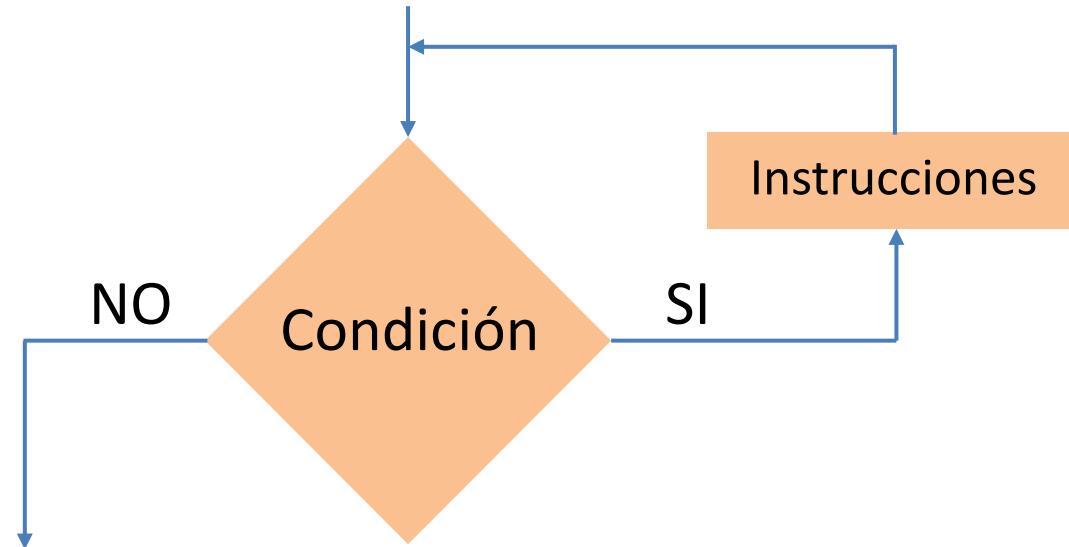
Principales preguntas para el diseño de un bucle:

- ¿Qué contiene el bucle?
- ¿Cuántas veces se debe repetir?

Bucle **while** (*mientras*)

El bucle **while** es el tipo de bucle más sencillo. Admite la siguiente sintaxis:

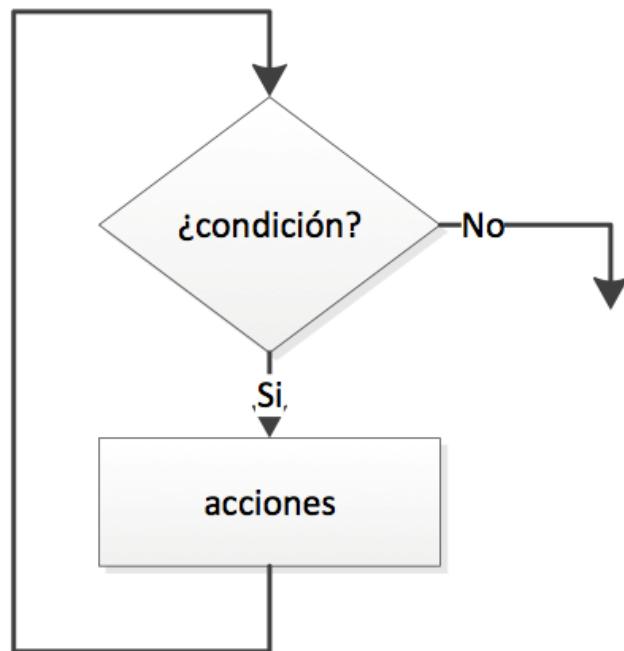
```
while expresión:  
    instrucción1  
    instrucción2  
    ...
```



El bucle **while** comienza por evaluar la **expresión** (condición). Si es verdadera, se ejecuta las instrucciones. Luego se vuelve a evaluar la expresión. De nuevo, si es verdadera, se vuelve a ejecutar las instrucciones.

Este proceso continúa hasta que el resultado de evaluar la expresión sea falso. Por esto esta expresión está relacionada a la **condición de salida**.

Bucle **while** (*mientras*)



Seudocódigo

```
mientras <condición> hacer  
    <acciones>  
fin_mientras
```

```
N = 1  
mientras N < 10 hacer  
    escribir("Num.:", N)  
    N = N + 1  
fin_mientras
```

Python

```
while condición:  
    acción1  
    acción2  
    acción3
```

```
N = 1  
while N < 10:  
    print("Num.:", N)  
    N = N + 1
```

Bucle while

Normalmente, en las instrucciones del bucle **while** se coloca alguna instrucción que modifique la expresión de control. Por ejemplo:

```
variable=5
while variable>5:
    print("la variable vale:",variable)
    variable=variable-1; #modifica la expresión de control
    print("valor tras decrementar la variable:",variable)
```

```
la variable vale: 5
valor tras decrementar la variable: 4
la variable vale: 4
valor tras decrementar la variable: 3
la variable vale: 3
valor tras decrementar la variable: 2
la variable vale: 2
valor tras decrementar la variable: 1
>>> |
```

Acumuladores y contadores

Contador:

- Es una variable que se incrementará en una unidad cada vez que se ejecute el proceso.
- El contador se utiliza para llevar la cuenta de determinadas acciones durante la ejecución del programa.
- Un contador debe ser inicializado, lo cual consiste en asignarle un valor inicial (generalmente cero).

`contador = contador+1 ó contador+=1`

Acumulador:

- La principal diferencia con el contador es que el incremento (o decremento) de cada suma es variable en lugar de constante.
- Puede ser un acumulador aditivo o multiplicativo.
- Un acumulador también debe ser inicializado (0 para la suma y 1 para la multiplicación).

`acumulador = acumulador+x ó acumulador+=x`

Bucle **while**

Ejemplo 1: Desarrolle un algoritmo que permita realizar la división entre dos números enteros mediante restas sucesivas.

```
dividendo = int(input("Ingrese el dividendo: "))
divisor = int(input("Ingrese el divisor: "))
cociente = 0 #inicialización del contador
while dividendo>=divisor:
    dividendo = dividendo - divisor
    cociente+=1
resto = dividendo
print("El cociente es:",cociente)
print("El resto es:",resto)
```

Bucle **while**

Ejemplo 2

Leer por teclado las calificaciones (de 0 a 100) de los alumnos de una clase.

Imprimir los siguiente:

- Cantidad de calificaciones recibidas
- La mejor calificación
- La peor calificación
- El promedio de calificaciones

El programa termina cuando se recibe alguna calificación fuera del rango (0 a 100). Las calificaciones son valores enteros (no hace falta validar esto último).

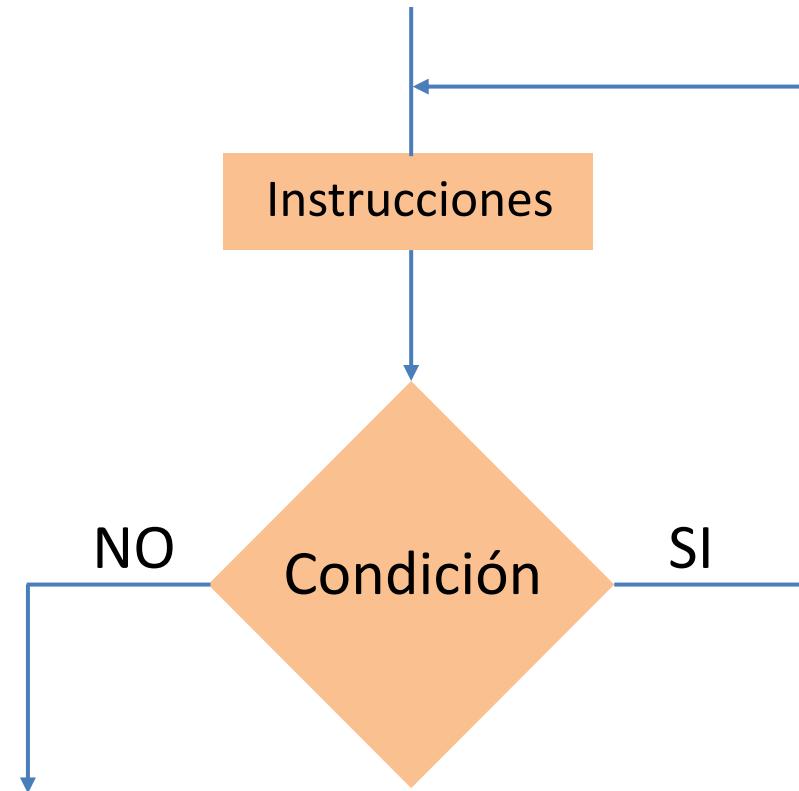
Bucle while

```
sum=0 #Inicialización del acumulador
cont=0 #Inicialización del contador
calif = int(input("Ingrese la calificación: "))
mayor = -1 #porque la peor calificacion es 0
menor = 101 #porque la mejor calificacion es 100
while calif>=0 and calif<=100:
    sum = sum + calif #acumulador
    cont = cont + 1 #contador
    if mayor<calif:
        mayor = calif
    if menor>calif:
        menor = calif
    calif = int(input("Ingrese la calificación: "))
if cont>0:
    print("La cantidad de calificaciones válidas es:",cont)
    prom = sum/cont
    print(f"El promedio de calificaciones es: {prom:.2f}")
    print("La mayor calificación es:",mayor)
    print("La menor calificación es:",menor)
else:
    print("No se ingresó al menos una calificación válida")
```

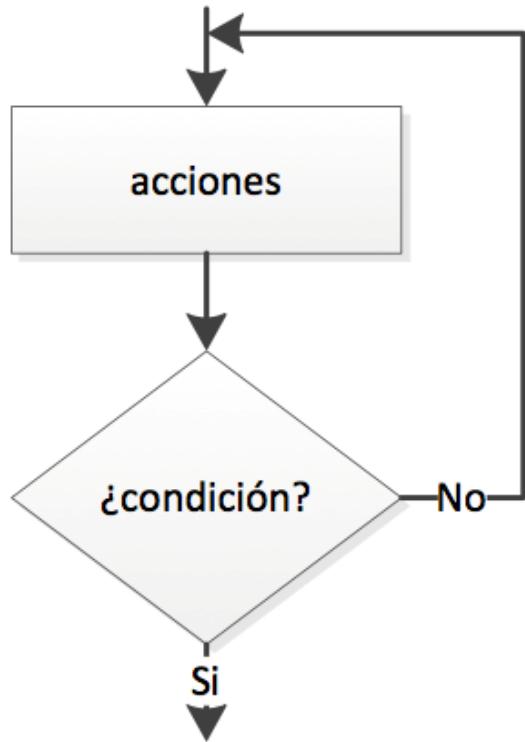
Bucle **do-while** (*hacer-mientras*)

Su funcionamiento es análogo al del bucle **while**, salvo que la expresión de control se evalúa al final del bucle. Esto nos garantiza que el bucle **do-while** se ejecuta al menos una vez. Es menos habitual que el bucle **while**.

En otros lenguajes, como C, existe el **do-while**. En Python no existe una estructura, aunque puede adaptarse el **while** para obtener el mismo funcionamiento.



Bucle **do-while** (*hacer-mientras*)



Python

Seudocódigo

```
repetir
    <acciones>
hasta_que <condición>
```

```
N = 1
repetir
    escribir("Num.:", N)
    N = N + 1
hasta_que N = 10
```

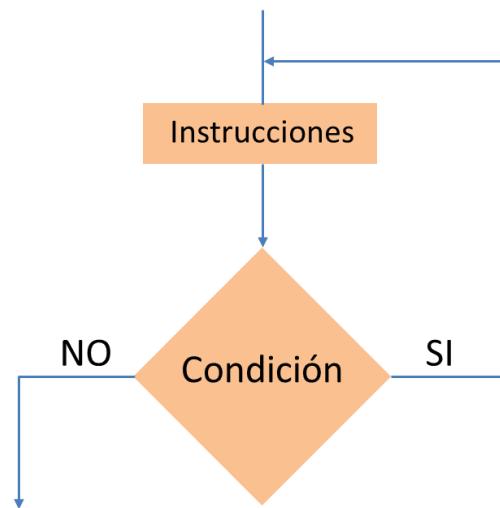
No existe una estructura igual a esta.

Siempre puede utilizar la construcción **while**

Uso común: validación de datos

Ejemplo 1: – Ingresar un número entero y positivo

```
n = float(input("Ingrese un numero entero y positivo: "))
while n!=int(n) or n<=0:
    print("Error! El numero debe ser entero y positivo")
    n = float(input("Ingrese otro numero: "))
n = int(n)
print("El numero ingresado es:",n)
```



Bucle do-while

Ejemplo 2: Calcular el factorial de un número n. Nota: usaremos un acumulador multiplicativo.

```
n = int(input("Ingrese el valor de n: "))
i=1 #Inicialización del contador
factorial=1 #Inicialización del acumulador multiplicativo
i+=1 #contador
while i<=n:
    factorial = factorial * i #acumulador multiplicativo
    i+=1 #contador
print(f"El factorial de {n} es: {factorial}")
```

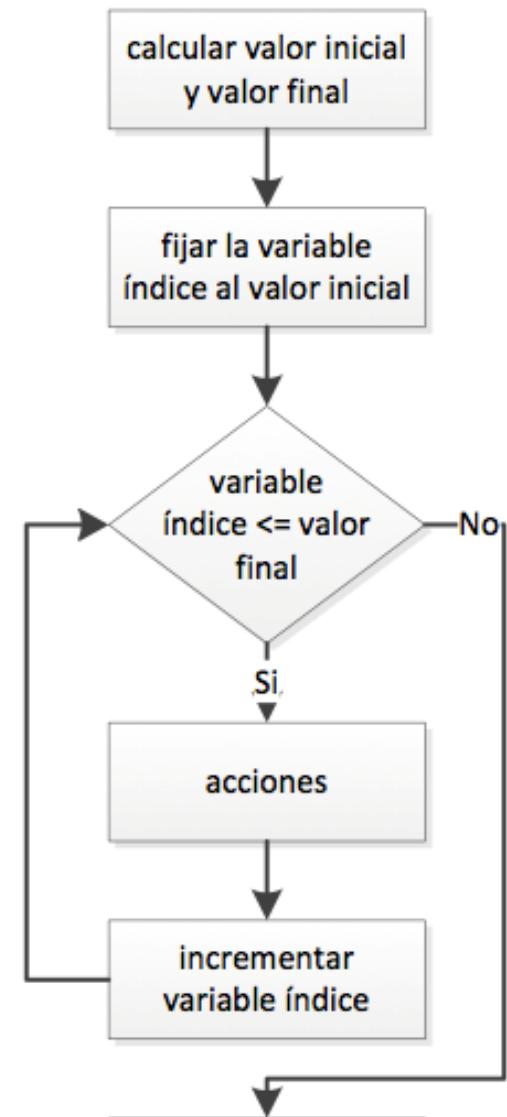
Bucle **for** (*desde/para*)

La estructura repetitiva **for** (*desde o para* en seudocódigo) ejecuta las acciones del cuerpo del bucle un número especificado de veces, y de modo automático controla el número de iteraciones o pasos a través del cuerpo del bucle.

```
desde N = 1 hasta 10 hacer
    escribir("Num.:", N)
fin_desde
```

Sintaxis en Python:

```
for <var> in range(<inicio>,<fin>,<paso>):
    acción
    accion
```



Uso de `range()` en Python

Permite generar una secuencia de números consecutivos (en realidad es un objeto).

Su formato es:

`range(<inicio>, <fin>, <paso>)`

Donde:

`<inicio>`: es el valor inicial de la secuencia. Por defecto es cero.

`<fin>`: es el valor final de la secuencia menos 1.

`<paso>`: es el incremento que se hará para el siguiente número. Por defecto es 1.

`range()` debe tener al menos el argumento `<fin>`.

Ejemplos:

`range(10)` : genera la secuencia `0, 1, 2, 4, ..., 9`

`range(1, 10)` : genera la secuencia `1, 2, 3, 4, ..., 9`

`range(1, 10, 2)`: genera la secuencia `1, 3, 5, 7, 9`

Bucle for

Ejemplo 1 - Mostrar en pantalla los cuadrados de los números del 1 al 10

```
for num in range(1,11):
    print("El cuadrado de",num,"es:",num**2)
```

```
El cuadrado de 1 es: 1
El cuadrado de 2 es: 4
El cuadrado de 3 es: 9
El cuadrado de 4 es: 16
El cuadrado de 5 es: 25
El cuadrado de 6 es: 36
El cuadrado de 7 es: 49
El cuadrado de 8 es: 64
El cuadrado de 9 es: 81
El cuadrado de 10 es: 100
```

>>>

Bucle for

Ejemplo 2 - Extraído de <https://wiki.c2.com/?FizzBuzzTest>

Imprimir en pantalla los números comprendidos entre 1 y 100. Pero para los múltiplos de 3, imprimir “Fizz” en lugar del número, mientras que para los múltiplos de 5 se imprime “Buzz” en lugar del número. Si el número es múltiplo de 3 y de 5, mostrar “Fizzbuzz” en lugar del número.

```
print("Numeros del 1 al 100")
for i in range(1,101):
    if i%3==0 and i%5==0:
        print("FizzBuzz")
    elif i%3==0:
        print("Fizz")
    elif i%5==0:
        print("Buzz")
    else:
        print(i)
```

Uso de banderas o interruptores

Una **bandera**, **interruptor**, o *switch* es una variable que puede tomar los valores falso (cero) o verdadero (distinto de cero) a lo largo de la ejecución de un programa, comunicando así información de una parte a otra del mismo. Pueden ser utilizados para el control de bucles.

Ejemplo:

```
contador=0  
bandera=True  
while bandera:  
    contador = contador + 1  
    if contador==10:  
        bandera=False
```



Uso de banderas

Ejemplo 3 – Escribir un programa que determine si un número ingresado por teclado es un número primo o no.

```
n = int(input("Ingrese un numero entero y mayor que 1: "))
esPrimo=True #bandera
for k in range(2,n):
    if n%k==0: #k es divisor de n?
        esPrimo=False #Si lo es, entonces no es primo

if esPrimo:
    print(f"El numero {n} es primo")
else:
    print(f"El numero {n} no es primo")
```

Uso de **break**

En Python, así como en otros lenguajes, existe la instrucción de salto **break** (interrumpir).

La misma se usa para interrumpir (romper) la ejecución normal de un bucle. Así, el control del programa se transfiere (*salta*) a la primera instrucción después del bucle.

```
while condición: #puede ser también un for
```

```
    ---
```

```
    ---
```

```
        if condiciónSalida:  
            break #sale del while
```

```
    ---
```

```
    ---
```

Uso de **break**

Ejemplo 3-2 – Escribir un programa que determine si un número ingresado por teclado es un número primo o no.

```
n = int(input("Ingrese un numero entero y mayor que 1: "))
esPrimo=True #bandera
for k in range(2,n):
    if n%k==0: #k es divisor de i?
        esPrimo=False #Si lo es, entonces no es primo
        break #Podemos salir del ciclo al encontrar un div
if esPrimo:
    print(f"El numero {n} es primo")
else:
    print(f"El numero {n} no es primo")
```

Uso de `continue`

En Python, así como en otros lenguajes, existe la instrucción de salto `continue` (*continuar*).

La misma se usa para interrumpir (romper) la ejecución normal de un bucle. Sin embargo, el control del programa no se transfiere a la primera instrucción después del bucle (como `break`); sino que finaliza la *iteración en curso*, transfiriéndose el control del programa a la condición de salida del bucle, para decidir si se debe realizar una nueva iteración o no.

```
while condición:  
    ---  
    ---  
    if condición_salto:  
        continue #no se ejecuta el resto de la iteración  
    ---  
    ---
```

Uso de `continue`

```
a = 10
while a<=20:
    if a==15: # Salta esta iteración
        a=a+1
        continue #Vuelve al inicio del bucle
    print("Valor de a:",a)
    a=a+1
```

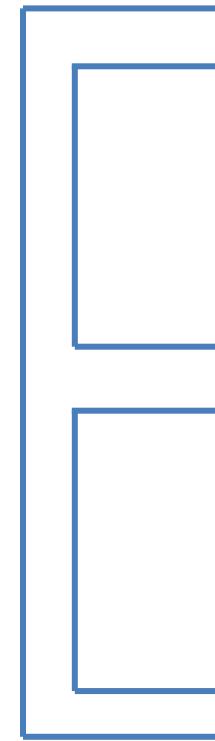
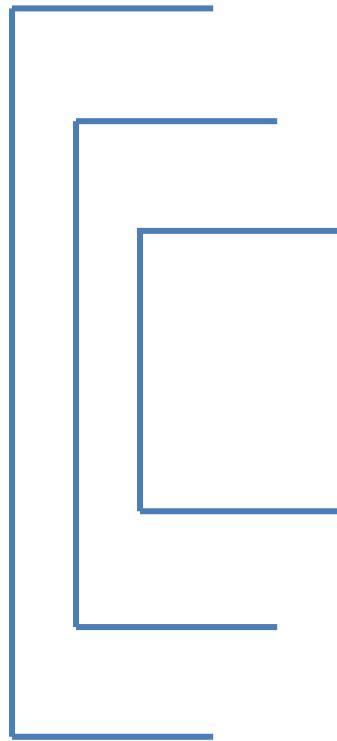
En este ejemplo se imprimen los valores del 10 al 20, excepto el 15.

```
Valor de a: 10
Valor de a: 11
Valor de a: 12
Valor de a: 13
Valor de a: 14
Valor de a: 16
Valor de a: 17
Valor de a: 18
Valor de a: 19
Valor de a: 20
```

>>>

Anidamientos de ciclos repetitivos

(Bucles en cascada)



Observación: las sangrías definen los anidamientos

Ejemplo de anidamiento

Imprimir todos los números primos comprendidos entre 2 y n .

```
n = int(input("Ingrese un numero entero y mayor que 1: "))
print("Los numeros primos del 2 al n son:")
for i in range(2,n+1):
    esPrimo=True #bandera
    for k in range(2,i):
        if i%k==0: #k es divisor de i?
            esPrimo=False #Si lo es, entonces no es primo
            break
    if esPrimo:
        print(i)
```

Ingrese un numero entero y mayor que 1: 20
Los numeros primos del 2 al 20 son:

2
3
5
7
11
13
17
19

>>>

Formato general para un bucle for en Python

- La sintaxis de un bucle for es la siguiente:

```
for variable in elemento iterable (lista, cadena, range, etc.):  
    cuerpo del bucle
```

- No es necesario definir la variable de control antes del bucle, aunque se puede utilizar como variable de control una variable ya definida en el programa.
- El cuerpo del bucle se ejecuta tantas veces como elementos tenga el elemento recorrible (elementos de una lista o de un range(), caracteres de una cadena, etc.).

Ejemplos

- Ejemplo de bucle for 1

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

```
Comienzo
Hola Hola Hola
Final
```

- Ejemplo de bucle for 2

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

```
Comienzo

Final
```

Si la lista está vacía, el bucle no se ejecuta ninguna vez.

En el primer ejemplo, los valores que toma la variable no son importantes, lo que importa es que la lista tiene tres elementos y por tanto el bucle se ejecuta tres veces. El siguiente programa produciría el mismo resultado que el anterior:

- **Ejemplo de bucle for 3**

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

```
Comienzo
Hola Hola Hola
Final
```

Si la variable de control no se va a utilizar en el cuerpo del bucle, 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 sabe así que los valores no se van a utilizar.

- **Ejemplo de bucle for 4**

```
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 _).

En los ejemplos anteriores, la variable de control "i" no se utilizaba en el bloque de instrucciones, pero en muchos casos sí que se utiliza. Cuando se utiliza, hay que tener en cuenta que la variable de control va tomando los valores del elemento recorrible.

- **Ejemplo de bucle for 5**

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

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

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

- **Ejemplo de bucle for 6**

```
print("Comienzo")
for i in ["Alba", "Benito", 27]:
    print(f"Hola. Ahora i vale {i}")
print("Final")
```

```
Comienzo
Hola. Ahora i vale Alba
Hola. Ahora i vale Benito
Hola. Ahora i vale 27
Final
```

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

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

```
i = 10
print(f"El bucle no ha comenzado. Ahora i vale {i}")

for i in [0, 1, 2, 3, 4]:
    print(f"{i} * {i} = {i ** 2}")

print(f"El bucle ha terminado. Ahora i vale {i}")
```

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

La variable de control se puede modificar en el interior del bucle, pero en cada iteración Python asigna a la variable de control el valor que le corresponde de acuerdo con el elemento iterable que define el bucle:

```
i = 10
print(f"El bucle no ha comenzado. Ahora i vale {i}")

for i in [0, 1, 2]:
    print(f"Ahora i vale {i}")
    i = 100
    print(f"Ahora i vale {i}")

print(f"El bucle ha terminado. Ahora i vale {i}")
```

```
El bucle no ha comenzado. Ahora i vale 10
Ahora i vale 0
Ahora i vale 100
Ahora i vale 1
Ahora i vale 100
Ahora i vale 2
Ahora i vale 100
El bucle ha terminado. Ahora i vale 100
```

En los ejemplos anteriores se ha utilizado una lista para facilitar la comprensión del funcionamiento de los bucles pero, si es posible hacerlo, se recomienda utilizar tipos `range()` entre otros motivos porque durante la ejecución del programa ocupan menos memoria en el ordenador.

El siguiente programa es equivalente al programa del ejemplo 1:

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

```
Comienzo
Hola Hola Hola
Final
```

Otra de las ventajas de utilizar tipos `range()` es que el argumento del tipo `range()` controla el número de veces que se ejecuta el bucle.

En el ejemplo anterior basta cambiar el argumento para que el programa salude muchas más veces.

```
print("Comienzo")
for i in range(10):
    print("Hola ", end="")
print()
print("Final")
```

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

Esto permite que el número de iteraciones dependa del desarrollo del programa. En el ejemplo siguiente es el usuario quien decide cuántas veces se ejecuta el bucle:

```
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? 6
Hola Hola Hola Hola Hola Hola
Adiós
```

Cuando se escriben dos o más bucles seguidos, la costumbre es utilizar el mismo nombre de variable puesto que cada bucle establece los valores de la variable sin importar los valores anteriores:

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

print()

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

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

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

Bucle for a través de cadenas

Hemos utilizado bucles for con listas, ¿qué tal con cadenas? Recuerde que las cadenas son una secuencia, en cuyo caso la variable de control va tomando como valor cada uno de los caracteres:

```
for i in "AMIGO":
    print(f"Dame una {i}")
print("¡AMIGO!")
```

```
Dame una A
Dame una M
Dame una I
Dame una G
Dame una O
¡AMIGO!
```

Bucle for con otros elementos iterables como tuplas y diccionarios, veremos mas adelante

Desafíos – Estructuras de repetición

Tema 1: Escribir un programa que invierta los dígitos de un número entero introducido por teclado.

Entrada

-123456



Salida

-654321

Tema 2: Escribir un programa que muestre el máximo común divisor (MCD) de dos números A y B (usando el algoritmo de Euclides).

Tema 3: Dado un valor *limite* como entrada, calcular el menor valor de n tal que se cumpla:

$$\sum_{i=1}^n \frac{2^i}{i} = \frac{2^1}{1} + \frac{2^2}{2} + \frac{2^3}{3} + \cdots + \frac{2^{n-1}}{n-1} + \frac{2^n}{n} \leq \text{limite}$$

Gracias por la atención

