

Bloque 1

Estructuras de Programa
E/S por consola, Comentarios, Cadenas,
Instrucciones selectivas e iterativas

Fundamentos de Programación
Departamento de Lenguajes y Sistemas Informáticos



Estructura de nuestros programa y sentencia import

1. En general las funciones que haya que desarrollar estarán dentro de una carpeta **src** en un archivo o módulo independiente con la extensión “.py”.

xxxxxxxx.py

2. Para probar las funciones haremos otro archivo o módulo también dentro de la carpeta **src** cuya denominación empezará por “*test_....py*”.

test_ xxxxxxxx.py

3. Para que el módulo test (el del punto 2) reconozca *todas* las funciones que están en el módulo de funciones (el del punto 1) deberá contener, antes de invocarlas, la sentencia:

*from xxxxxxxx import ** (donde *xxxxxxxx* es la denominación del módulo)

Nota.- Si no se quiere importar todas las funciones se puede sustituir el * por los nombres exactos de las funciones que se deseen importar separadas por “,”

from xxxxxxxx import función1, función2,



Entrada por la consola (input)

La función *input*("texto") permite la captura de textos por teclado, devolviendo lo que se ha escrito por teclado en un *tipo str* (string).

Ejemplo para introducir texto:

```
nombre= input("Introduce tu nombre: ")
```

Si se escribe: María José <<Enter>>, la variable *nombre* almacenará "María José"

Ejemplo para introducir números:

Para datos numéricos se aplican las funciones *int* para enteros y *float* para reales.

```
edad= int(input("teclea tu edad: "))
```

```
estatura= float(input("teclea tu estatura en metros: "))
```

los decimales se separan con el punto

Si primero se teclea: 22 <<Enter>> y después: 1.83<<Enter>>, las variables *edad* y *estatura* almacenarán 22 y 1.83 respectivamente. Al ser valores numéricos se pueden realizar operaciones aritméticas con dichas variables.



Salida por la consola (print)

La función **print**(variable o texto) permite visualizar texto o el contenido que almacena una variable.

Por ejemplo: Si suponemos *edad* es numérico de tipo **int** que almacena 19).

print (edad) correcto: *Visualiza el único parámetro numérico que recibe la función print*

Resultado: 19

print ("Tu edad es: "+ edad) da error: *porque no se concatena una cadena con un número*

Resultado: ----

print ("Tu edad es: "+ **str**(edad)) correcto: **str** *convierte un valor numérico a cadena y el "+" concatena*

Resultado: Tu edad es:19

print ("Tu edad es:", edad) correcto: *Visualiza el 1er. parámetro (una cadena) y el 2do. (un número)*

Resultado: Tu edad es: 19 (observar que hay un blanco entre los : y el 19)

Nota: *Más adelante se enseñará como formatear los valores a visualizar.*



Comentarios

Los lenguajes de programación permiten escribir líneas de comentario que ayudan a la comprensión de los algoritmos que se implementan. *No interfieren con las instrucciones o sentencias*

En Python hay dos formas de realizar comentarios

- Mediante el carácter **#** y a continuación, en la misma línea, se escribe el texto que se desea
- Mediante **"""** y **"""** para escribir comentarios que abarquen más de una línea

Por ejemplo:

```
# esto es un comentario escrito en una línea
```

```
'''
```

```
Este comentario que ocupa  
más de  
una línea, Concretamente tres.
```

```
'''
```



Trabajando con cadenas –String- (str)

Son literales que permiten trabajar con textos “alfanuméricos”. También pueden almacenar un único carácter.

Sintaxis:

Se encierran entre apóstrofes, dobles comillas o tres apóstrofes (en este último caso, permite literales de más de una línea).

Ejemplos:

- *‘Esto es un literal, también llamado cadena o String, construido entre apóstrofes’*
- *“Esto también es otro literal, salvo que para construirlo se ha usado las dobles comillas”*
- *“Este es también un literal, pero tiene la versatilidad de poder escribirse en más de una línea.*

A la hora de visualizarse por la consola, dado que se ha escrito en cuatro líneas también se visualiza en cuatro líneas ””

- *‘2’, “2”, “”2”” o ‘A’, “a”, “”=”” → también son literales*



Operaciones habituales con cadenas –String- (str)

Si cadena=“Fundamentos de Programación”

- **len**(cadena) devuelve el número de caracteres que tiene → **27** (incluye espacios en blanco)
- Se accede a una/s posiciones concretas con el operador **[]** (operador slicing)

¡Ojo! El primer elemento es el **0**. Así que:

- cadena[0] → **F**
- cadena[1] → **u**
- cadena[-1] → **n** (índices negativos cuentan desde el final)
- cadena[-4] → **c**
- cadena[3:9] → **dament** (observar que no llega a la posición 9. ¡Se queda en la 8!)
- cadena[10:-10] → **s de Pr**

Se puede omitir el primero o el segundo de los valores de operador para acceder desde el primero o hasta el último, respectivamente:

- cadena[:9] → **Fundament**
- cadena[20:] → **amación**
- cadena[-5:] → **ación**



Trabajando con cadenas –String- (str)

Operaciones de concatenación

- Operador **+**: concatena las cadenas de su izquierda y derecha:
 - ‘Funda’ + “ment” + ‘os de Pr’ + “ogramación” → *Fundamentos de Programación*
- Operador *****: concatena la cadena tantas veces consigo mismo como indique el número que sigue al operador. Si cadena=“Betis – Sevilla”
 - cadena*3 → *Betis – SevillaBetis – SevillaBetis – Sevilla*



Operadores relacionales con cadenas –String- (str)

Operaciones relacionales para cadenas.

- Operadores relacionales (<, <=, >, >=, ==).

Permiten compara dos cadenas por su léxico, no por su longitud:

- 1. Se comparan los dos primeros caracteres y una será menor que la otra si su primer carácter está antes que el primer carácter de la otra en el diccionario.*
- 2. Si son iguales, se comparan los segundos caracteres y así sucesivamente hasta que encontrar una pareja en una posición en que no sean iguales, el carácter que esté antes en el alfabeto será el que haga que la cadena sea la menor. Si una de ellas se acaba antes que la otra, esta será la menor.*
- 3. Si tienen la misma longitud y contienen los mismos caracteres y en el mismo orden son iguales (==)*
 - ‘Ana’ < ‘Alfonso’ → False (la **ele** está antes que la **ene** en el alfabeto).*
 - ‘Ana’ < ‘alfonso’ → True (las **mayúsculas** están antes que las **minúsculas**)*



Caracteres especiales –String- (str)

Secuencias de escape.

- Existen algunos caracteres que tienen un significado especial anteponiéndoles \ (son un carácter, aunque se escriban con dos o más pulsaciones de teclado).

Las más comunes son:

- **\n**: Salta una línea
- **\t**: Inserta una tabulación
- **\'**: Inserta un apostrofe (') en una cadena encerrada entre apóstrofes ' '
'Eres un \'listillo\' que te aprovechas de otros' → Eres un 'listillo' que te aprovechas de otros
- **\"**: Inserta unas comillas (") en una cadena encerrada entre comillas " "
"Eres un \"listillo\" que te aprovechas de otros" → Eres un "listillo" que te aprovechas de otros

Ejemplo:

```
print("Hola Mundo!\nAdiós Mundo!")
```

Hola Mundo!
Adiós Mundo!



Función range

La función **range** (*m,n,p*) permite generar números enteros desde *m* hasta *n-1* (es decir, no incluye a n) con un incremento de “*p en p*”



Ejemplos

- **range(1,10,1)** o **range(1,10)** → 1, 2, 3, 4, 5, 6, 7, 8, 9 (si se omite *p*, toma por defecto el valor 1)
- **range(1,18,3)** → 1, 4, 7, 10, 13, 16
- **range(18,3,-3)** → 18, 15, 12, 9, 6
- **range(5, -5, -2)** → 5, 3, 1, -1, -3
- **range(1,18,-3)** → no genera ningún valor porque no es posible ir del 1 al 18 con incrementos negativos.



Sentencia if-else

Permite *evaluar una condición* y ejecutar un conjunto o bloque de sentencias (**las azules**) si el resultado de evaluar la condición es *True* (cierta o verdadera) o el otro bloque (**las amarillas**) si el resultado de evaluar la condición es *False* (falsa)

Sintaxis:

if condición:

sentencia 11
sentencia 12

...

sentencia 1n

} Bloque 1

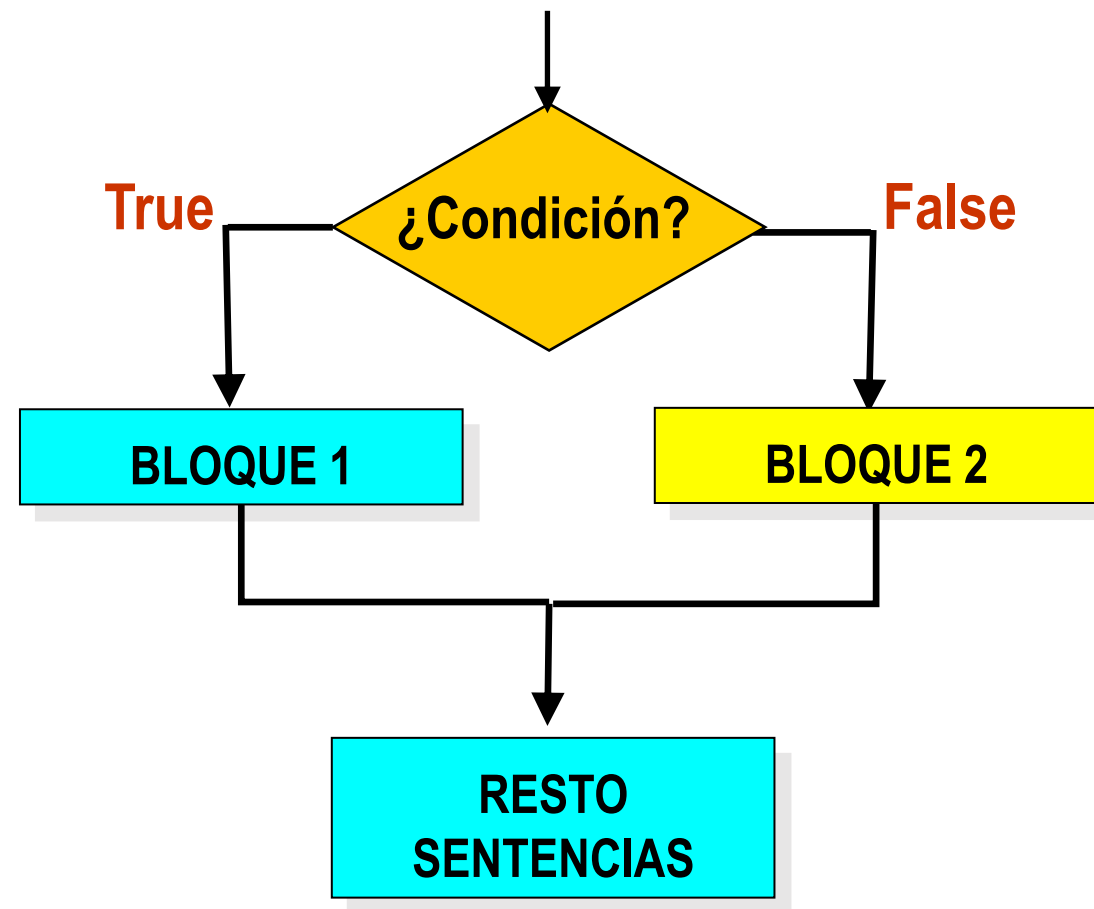
else:

sentencia 21
sentencia 22

...

sentencia 2m

} Bloque 2





Sentencia if (sin else)

Permite *evaluar una condición* y ejecutar un conjunto o bloque de sentencias si el resultado de evaluar la condición es *True* (cierta o verdadera) o no ejecutar nada si el resultado es *False* (falso)

Sintaxis:

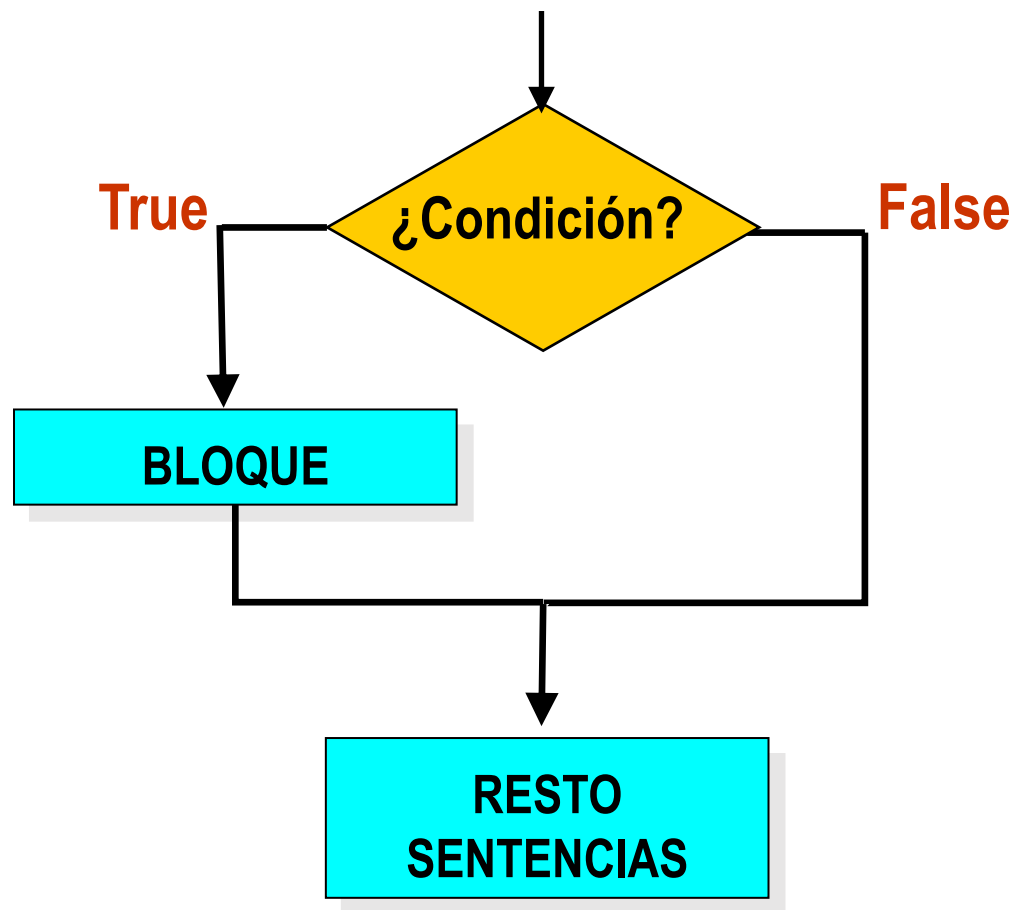
if condición:

sentencia 1
sentencia 2

...

sentencia n

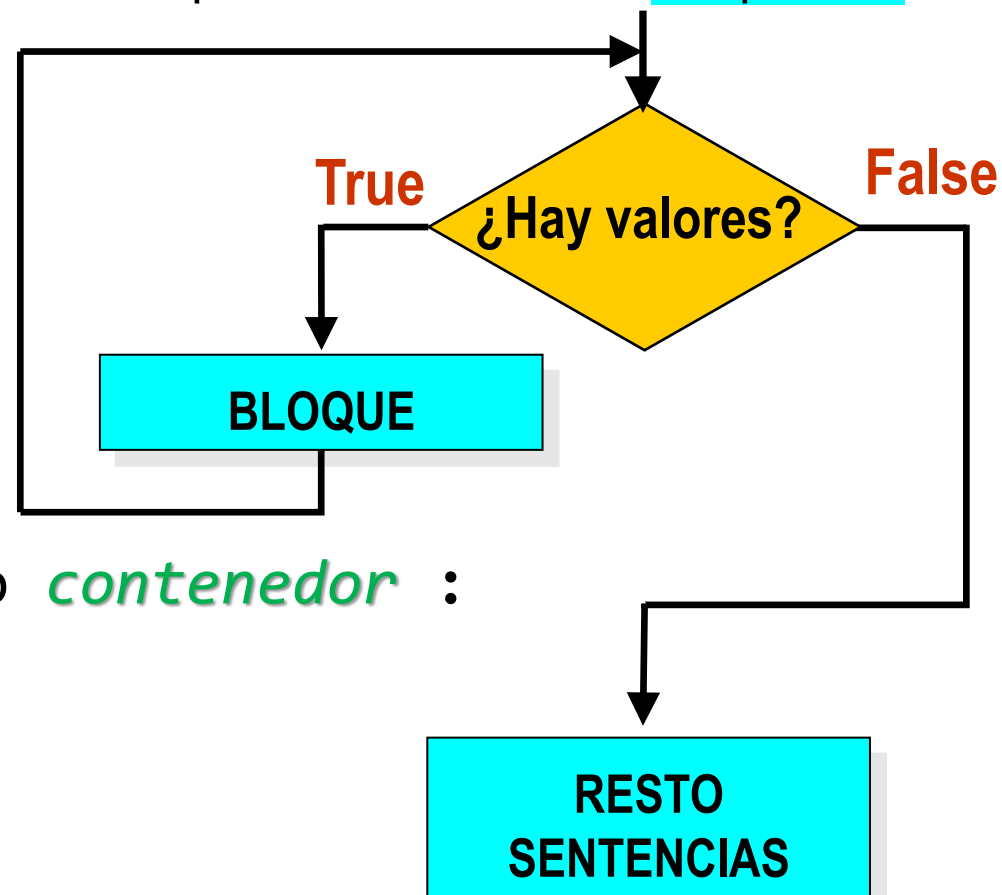
} Bloque





Sentencia for

La sentencia **for** permite recorrer cada uno de los valores de una **secuencia de valores** o los valores contenidos en un “**contenedor**” (*ya estudiaremos que es un contenedor y los 4 tipos que vamos a manejar*), desde el primero al último, ejecutando para cada valor el **bloque de sentencias**



Sintaxis:

for **variable** **in** **secuencia de valores** o **contenedor** :

sentencia 1
sentencia 2

...

sentencia n

} Bloque



Ejemplo de for que recorre los elementos generados por range



+



Ejemplo

```
for número in range (1,20,4):  
    print (número)
```



Salida por la consola

1
5
9
13
17



Sentencia while

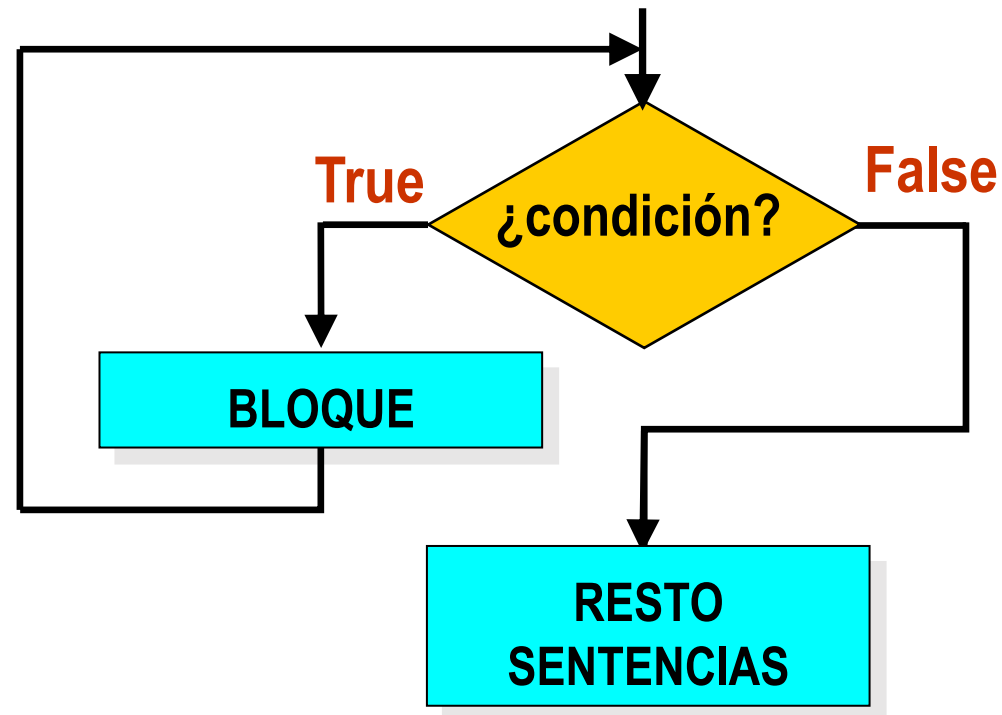
Permite, al igual que la sentencia for, ejecutar bucles sobre un bloque de sentencias mientras la condición evalúe a *True*

Sintaxis:

while *condición*:

```
sentencia 1  
sentencia 2  
...  
sentencia n
```

Bloque



Observaciones importantes:

- Se ejecuta de manera *indefinida* el bloque de sentencias, mientras la *condición* es *True* por lo que debe haber entre las *sentencias*, al menos una, que cambie la *condición* a *False* en alguna iteración del bucle.
- Si la *condición* es *False* la primera vez, *no se ejecuta el bloque de sentencias*



Sentencia while (ejemplo)

Ejemplo:

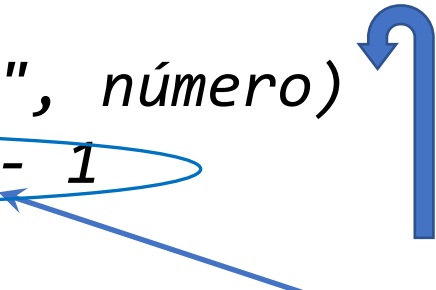
```
número= 4
```

```
while número > 0:
```

```
    print ("Número =", número)
```

```
    número = número - 1
```

```
print ("sacabó")
```



Esta sentencia va cambiando la condición, restando una unidad positiva al valor la variable número

Resultado por la consola

Número = 4

Número = 3

Número = 2

Número = 1

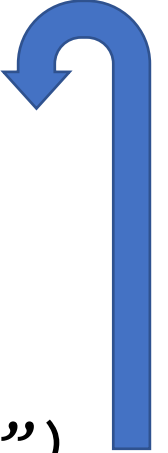
sacabó



Sentencia while (ejemplo)

Ejemplo típico de introducción de datos hasta que se teclea un dato que finaliza :

```
año=int(input("Teclea un año (o 0 para terminar: "))
while año!=0:
    if es_bisiesto(año)==True:
        print("--->El año es bisiesto")
    else:
        print("--->El año no es bisiesto")
    año=int(input("Teclea un año (o 0 para terminar): "))
print("fin")
```



Se presupone que la función denominada *es_bisiesto* recibe como parámetro dado un número entero y devuelve *True* o *False* según ese número represente un año bisiesto o no



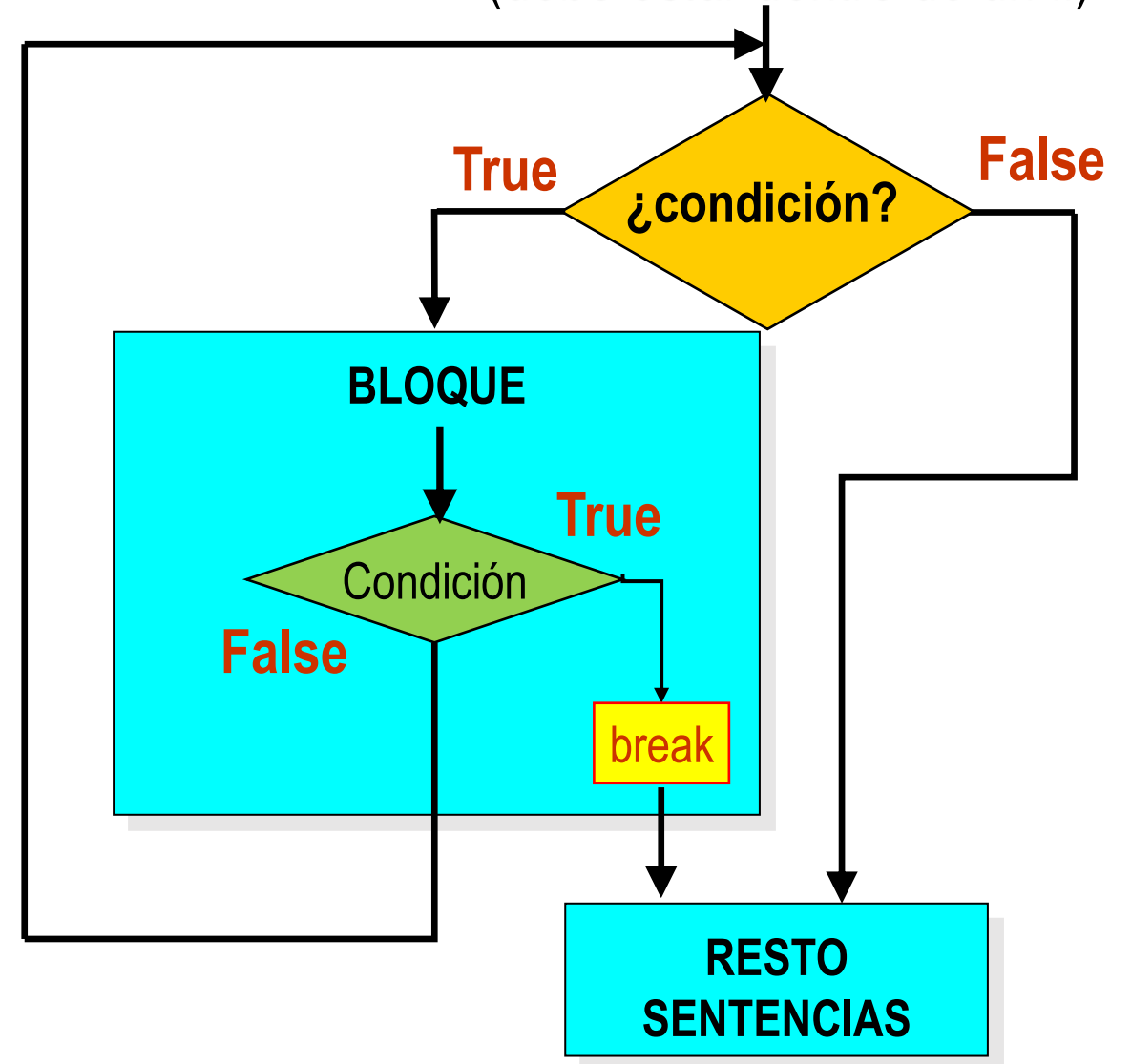
Sentencia break

Permite finalizar anticipadamente la ejecución de un for o un while (*debe estar dentro de un if*)

Sintaxis :

```
for variable in ...:  
    if condición:  
        ...  
        break  
    sentencias-n  
    ...  
    sentencia-m
```

} Bloque





Ejemplo de recorrido de cadenas-String- (str)

Recorrido de cadenas.

Se puede recorrer una cadena mediante un for con las dos siguientes:

Sintaxis:

a) Recorrido **como contenedor**

```
grado="Ing. Software"
```

```
for letra in grado:
```

```
    print(letra)
```



b) Recorrido **por posición**

```
grado="Ing. Software"
```

```
for posición in range(0,len(grado)):
```

```
    print(grado[posición])
```



I
n
g
.
S
o
f
t
w
a
r
e



Ejercicio (para clase)

Se trata de hacer un proyecto “*T_03_AñoBisiesto*”.

Cree una carpeta *src* y en ella dos módulos .py:

- El primer módulo “*es_bisiesto.py*” contendrá la función “*es_bisiesto*” que recibiendo un número entero como parámetro devolverá *True* o *False* según represente un año bisiesto o no.

Regla: *Un año es bisiesto si:*

a) es divisible por 400, o

b) si es divisible por 4 pero no por 100

- El segundo archivo “*test_es_bisiesto.py*” contendrá las instrucciones necesarias para que, usando la función *es_bisiesto*, pida un año (*de tipo entero*), o 0, y visualice el año introducido y a continuación : “es bisiesto” o “no es bisiesto”. El proceso finaliza cuando introduce el año 0

Juego de ensayo:

- 2000 es bisiesto → porque es divisible por 400
- 1992 es bisiesto → porque es divisible por 4 pero no por 100
- 2100 no es bisiesto → porque no es divisible por 400 y aunque es divisible por 4 también lo es por 100.
- 2021 no es bisiesto → porque no es divisible por 400 y tampoco por 4



Ejercicio (para casa)

Realizar un proyecto *T04_SumaDeSecuencias*

Cree una carpeta *src* y en ella dos módulos .py:

- El primer módulo “*suma_números.py*” contendrá la función “*sumar_números*” que recibiendo tres números *n1*, *n2* y *n3*, devuelva la suma de los números comprendido entre *n1* y *n2* que sean múltiplos de *n3*.
- El segundo archivo “*test_suma_números.py*” que pida por teclado tres números enteros **a**, **b** y **c**, y usando la función “*sumar_números*” visualice, por ejemplo:

La suma de los números entre 15 y 88 múltiplos de 7 es: 525

La suma de los números entre 12 y 39 múltiplos de 3 es: 255

Importante: comprobar que **a** es menor o igual que **b**, si no es así se visualiza **Error en los datos** y se finaliza el programa.



Ejercicio (para casa)

Se trata de hacer un proyecto Python “*T05_Número_Combinatorio*” con dos archivos “.py” dentro de la carpeta *src*

- El primer archivo “*combinatoria.py*” contendrá dos funciones:
 - “*factorial*” que recibiendo como parámetro *un número entero* no negativo devuelva el factorial de dicho número. *Recuerda que : $n! = 1 \times 2 \times 3 \times \dots \times (n - 1) \times n$ y que $0! = 1$.*
 - “*número_combinatorio*” que recibiendo como parámetro *dos números* no negativos devuelva el número combinatorio. *Recuerda que: $\binom{m}{n} = \frac{m!}{n! \times (m-n)!}$*
- El segundo archivo “*test_combinatoria.py*” contendrá las instrucciones necesarias para pedir por teclado dos números y probar la función “*número_combinatorio*”. Se debe comprobar que si el numerador es menor que el denominador visualice directamente “*El numerador es más pequeño que el denominador*”.

Juego de Ensayo: Si se teclea 5 y 0 debe visualizar: *5 sobre 0 es 1*

Valores para probar: $\binom{5}{0} \rightarrow 1$; $\binom{5}{1} \rightarrow 5$; $\binom{5}{2} \rightarrow 10$; $\binom{5}{3} \rightarrow 10$; $\binom{5}{4} \rightarrow 5$; $\binom{5}{5} \rightarrow 1$