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).

Por ejemplo:

nombre= *input*(“Introduce tu nombre: ”)

Si se escribe “María José” <<Enter>>, la variable *nombre almacenará “María José”*

Si se necesita que los datos sean 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*

Por ejemplo:

Si primero se teclea 22 <<Enter>> y después 1.83<<Enter>>, las variables *edad* y *estatura* almacenarán 22 y 1.83 respectivamente como valores numéricos para realizar operaciones aritméticas.

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.*

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,….

Comentarios

Todos los lenguajes de programación permiten escribir líneas de comentario que ayudan a la compresión de los algoritmos que se implementan.

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.
””
```

Sentencia if-else

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

Sintaxis:

if condición:
    sentencia 11
    sentencia 12
    …
    sentencia 1n

Bloque obligatorio

else:
    sentencia 21
    sentencia 22
    …
    sentencia 2m

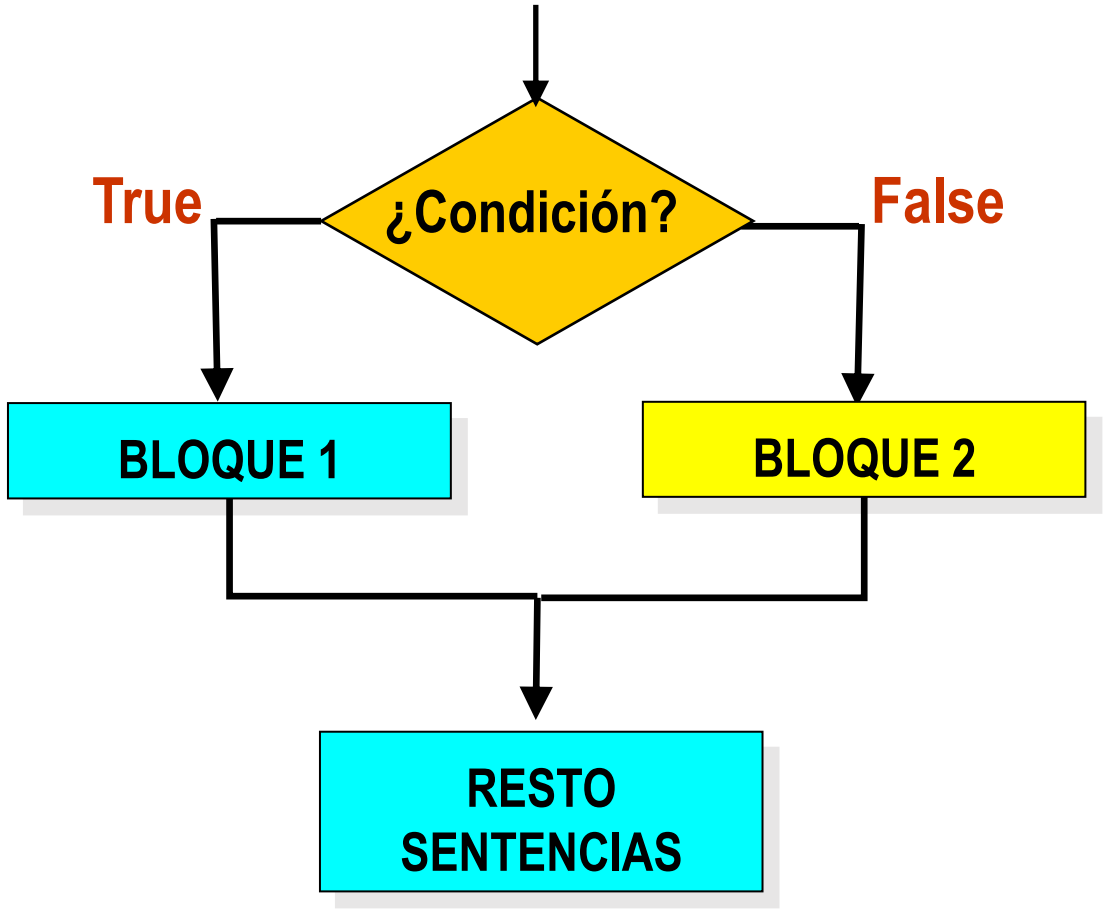
La sentencia *else* y el bloque se puede omitir con lo que si la evaluación de la condición es *False* no se hace nada.

Sentencia if-else

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

Sintaxis:

if condición:
    sentencia 11
    sentencia 12
    …
    sentencia 1n
else:
    sentencia 21
    sentencia 22
    …
    sentencia 2m



Ejercicio:

Se trata de hacer un programa Python que calcule el perímetro y el área de un triángulo conociendo sus tres lados.:

1. Nombre del proyecto: *T_04_Area_Triangulo_Herón*
2. Cree dentro del proyecto la carpeta *src*
3. Dentro de *src* cree los módulos: *triangulo.py* y *test_triangulo.py*
4. El módulo triangulo.py debe contener dos funciones
   - *perímetro* que recibe como parámetros de entrada los tres lados y devuelve el perímetro.
   - *área_por_herón* que recibe como parámetros de entrada los tres lados y devuelve el área. Utilizar la fórmula de Herón:

     Si “p” es el perímetro $p = a + b + c$ y “s" el semiperímetro $s = \frac{p}{2}$ el

     $$área = \sqrt{s(s-a)(s-b)(s-c)}$$
5. El módulo test_triangulo.py debe pedir por teclado los tres lados y visualizar los resultados.

Ejercicio para casa:

Modifique en el proyecto anterior *T_04_Triangulo* la función *área_por_herón* para que sólo calcule el área si realmente los datos que se introducen por teclado permitirían forman un triángulo:

- Si alguno de los lados es negativo cero devuelvan *None*.
- Si la suma de dos lados es menor que el otro devuelva *None*

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*"

Por ejemplo:

- *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 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 (las amarillas)

Sintaxis (tiene varios formatos que iremos aprendiendo):

```
for variable in secuencia de valores o contenedor :
        sentencia-1
        ...
        sentencia-n
```

Ejemplo:

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

Salida por el terminal

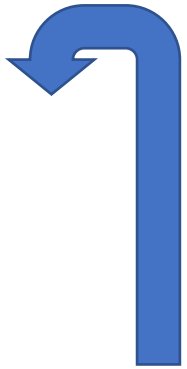
```
1
5
9
13
17
```

Sentencia while

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

- *Sintaxis:*

```
while condición:
    sentencia 1
    sentencia 2
    ...
    sentencia n
```



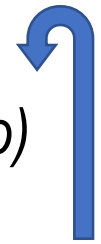
- *Observaciones importantes:*
  - *Se ejecuta de manera indefinida* el bloque de sentencias, mientras la *condición* es *True* por lo que debe haber entre las sentencias, al menos una, que cambie la *condición* durante en alguna iteración del bucle a *False*
  - Si la *condición* es *False* la primera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

Sentencia while (ejemplo)

Ejemplo 1:

```
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

Número = 4
Número = 3
Número = 2
Número = 1
sacabó

Sentencia while (ejemplo)

Ejemplo 2:

```
año=int(input("Teclea un año: ")
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): ")
```

Ejercicio: Se trata de hacer un proyecto "*T_05_Año_Bisiesto*" con dos módulos .py

- El primer módulo "*es_bisiesto.py*" contendrá la función "*esbisiesto*" 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 pedir un año y probar la función *esbisiesto*, visualice los resultados para el siguiente juego de ensayo:

Juego de ensayo:

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

Ejercicio para casa:

Modifique en el proyecto anterior *T_05_Año_Bisiesto* la función *test_es_bisiesto.py* para que esté pidiendo años mientras no se introduzca un *año negativo*