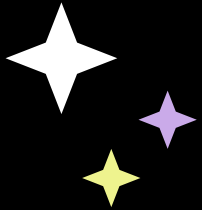




# Estructuras de control en Python



AED 2025





## En la carpeta del módulo 2 encontrarás:

### 1. Guía Modulo 2 (presentación)

En esta presentación te contamos qué material hay disponible y hacemos un repaso de los temas del módulo.

### 2. Colab con ejemplos

Separados por bloques de temas, puedes ejecutar online los ejemplos que preparamos.

[3\\_Condicionales e Iterativas Python](#)

### 3. Guía teórica

Material teórico detallado de lo que necesitas saber de condicionales y bucles, como es la sintaxis y la comparación con pseudocódigo


### 4. Cuestionario

Cuestionario de auto-evaluación para medir si adquirieron los conocimientos esperados para este módulo

[Accede al cuestionario en este link](#)

# Sobre Google Colab y su uso de IA para asistir en el desarrollo

Al ejecutar código que tiene un error da un mensaje de sugerencia



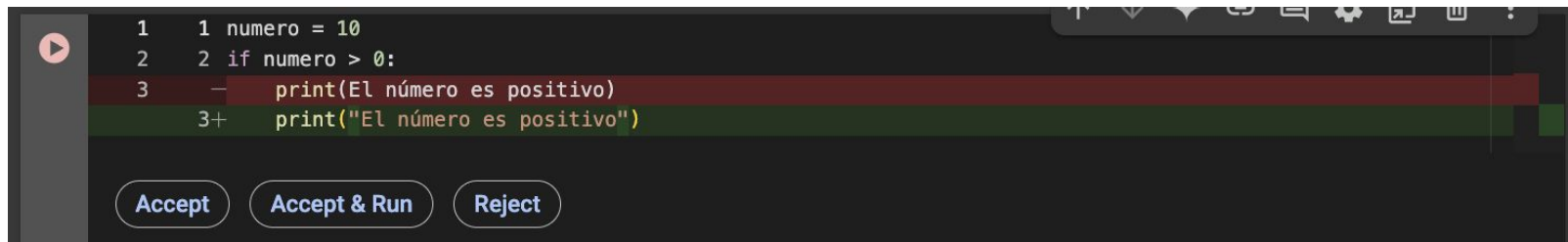
```
1 numero = 10
2 if numero > 0:
3     print(El número es positivo)
```

File "<ipython-input-4-23ee0d955a27>", line 3  
print(El número es positivo)  
^  
SyntaxError: invalid syntax. Perhaps you forgot a comma?

Next steps: [Fix error](#)

Gemini

Si hacemos click en **Fix Error** propone una solución, asumiendo que es un mensaje y nos olvidamos las comillas “



```
1 1 numero = 10
2 2 if numero > 0:
3 - print(El número es positivo)
3+ print("El número es positivo")
```

[Accept](#) [Accept & Run](#) [Reject](#)

# Estructuras de control

Señalan el orden en que tienen que sucederse los pasos de un algoritmo.

Secuencia	Decisión	Repetición
Expresiones que permiten calcular valores	if	while
Llamados a funciones o procedimientos	if - else	do - while
	elif	for

# Enfoque secuencial al programar



La estructura secuencial es aquella en la que una **acción (instrucción)** sigue a otra en secuencia.

Las tareas se suceden de tal modo que **la salida de una es la entrada de la siguiente** y así sucesivamente hasta el fin del proceso.

# Secuencial

```
X = 5  
X = x * 2  
print( x + 6)  
X = "hola"  
print(x)
```

La estructura secuencial es aquella en la que una **acción (instrucción)** sigue a otra en secuencia.

Las tareas se suceden de tal modo que **la salida de una es la entrada de la siguiente** y así sucesivamente hasta el fin del proceso.

# Secuencial

**Si un programa sólo ejecuta instrucciones planas unas tras otras no servirían de mucho.**

Que las acciones se ejecuten secuencialmente implica que **nunca se ejecutará más de una acción al mismo tiempo**

Por suerte, ahí es donde aparecen las estructuras de control, las cuales van a permitir que el **flujo del programa** se adapte y sepa cómo actuar ante determinadas situaciones e incluso **repetir** una tarea si es necesario.

# Estructuras de control

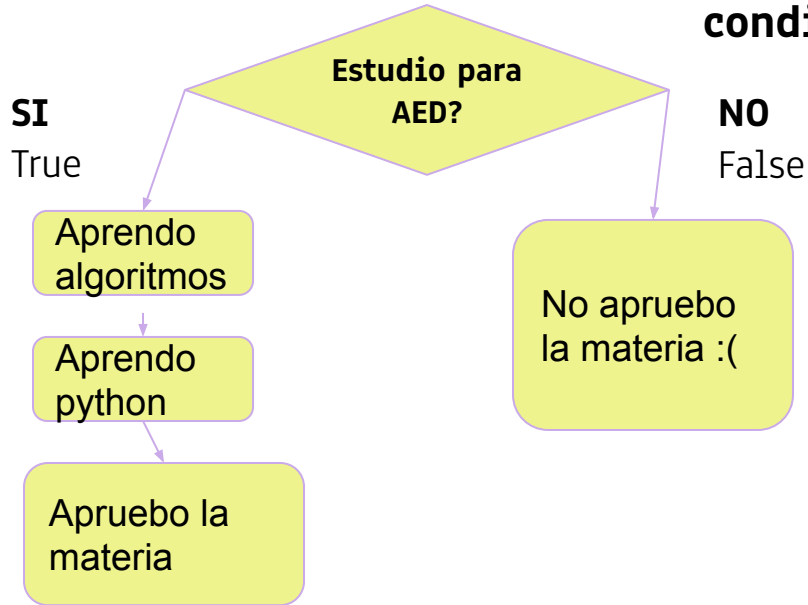
Señalan el orden en que tienen que sucederse los pasos de un algoritmo.

Secuencia	Decisión	Repetición
Expresiones que permiten calcular valores	if	while
Llamados a funciones o procedimientos	if - else	do - while
	elif	for



# Condicionales

**Cuando hay que tomar una decisión aparecen las estructuras condicionales.**

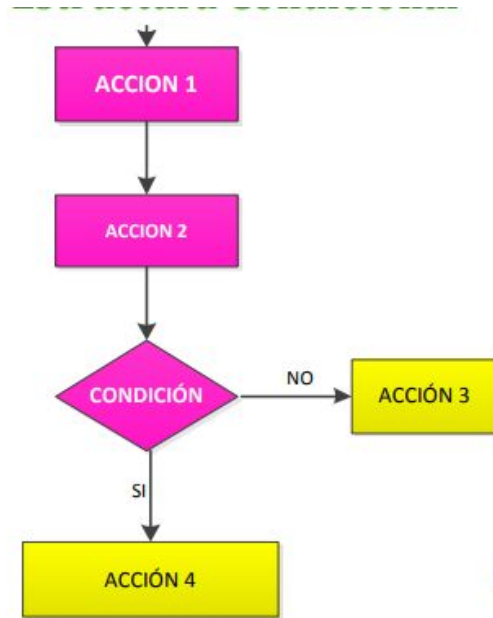


En nuestra vida diaria se nos presentan situaciones donde debemos decidir.

- ¿Aprendo a programar en python o js?
- ¿Me pongo este pantalón?
- Para ir a la facu ¿elijo el camino A o el camino B?
- Al comenzar un curso ¿elijo el turno mañana, tarde o noche?

**Las acciones que haga despues varían del resultado de la decisión**

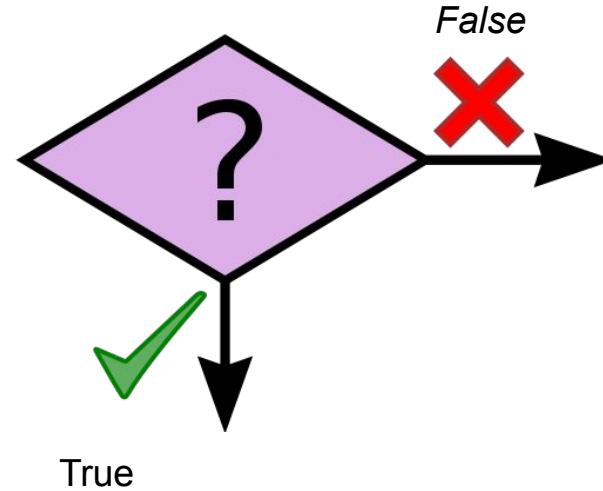
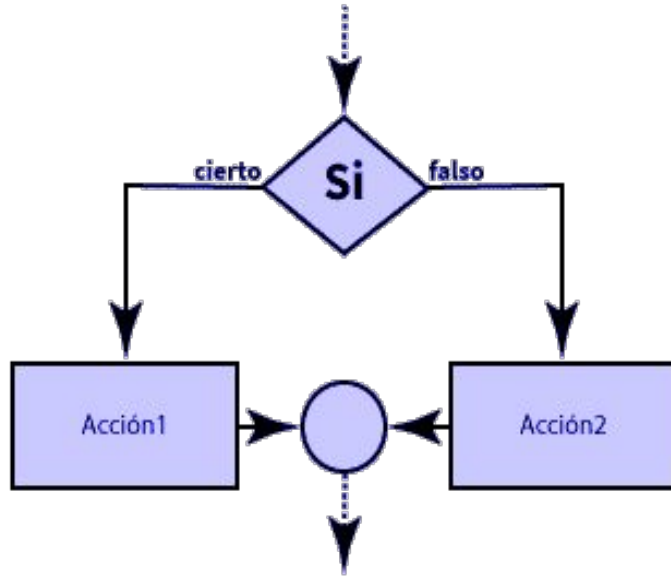
# Condicionales



Las estructuras selectivas se utilizan para tomar decisiones lógicas; de ahí que se suelen denominar también estructuras de decisión o **condicionales**

**En estas estructuras, se realiza una evaluación de una condición y de acuerdo al resultado, el algoritmo realiza una determinada acción. Las condiciones son especificadas utilizando expresiones lógicas.**

# Condicionales



# Python If:

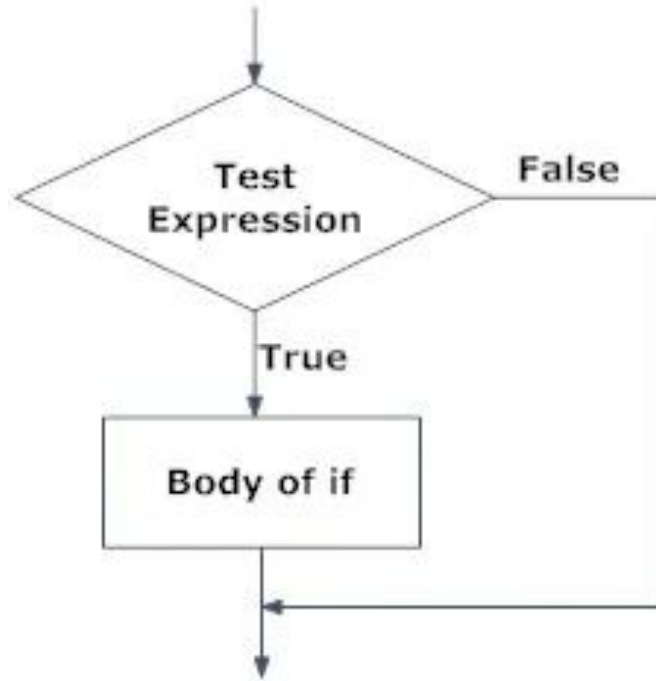


Fig: Operation of if statement

# ¿Como lo hago en Python?

## Sentencias **if**

```
if [condicion]:  
    Sentencias #INDENTACION  
otrasSentencias
```

# Condicionales en Python

## Código

```
if True: # equivale a if not False
    print("Se cumple la condición")
    print("También se muestre este print")
```

## Resultado

```
Se cumple la condición
También se muestre este print
```

- Si se verifica que una determinada condición se cumple, se puede ejecutar una serie de instrucciones y luego seguir adelante.
- Si la condición NO se cumple, NO se ejecutan dichas instrucciones y se sigue adelante.

# Condicionales en Python

## Código

```
num = 3
if num > 0:
    print(num, "Es un numero positivo.")
print("Esta linea siempre se imprime.")

num = -1
if num > 0:
    print(num, "Es un numero positivo.")
print("Esta linea tambien siempre se imprime.")
```

## Resultado

```
"Es          un          numero          positivo"
"Esta linea siempre se imprime."
"Esta linea tambien siempre se imprime."
```

# Condicional Alternativo Simple

- Si se verifica que una determinada condición se cumple, se puede ejecutar una serie de instrucciones y luego seguir adelante.
- Si la condición NO se cumple, NO se ejecutan dichas instrucciones y se ejecutan **otras instrucciones**

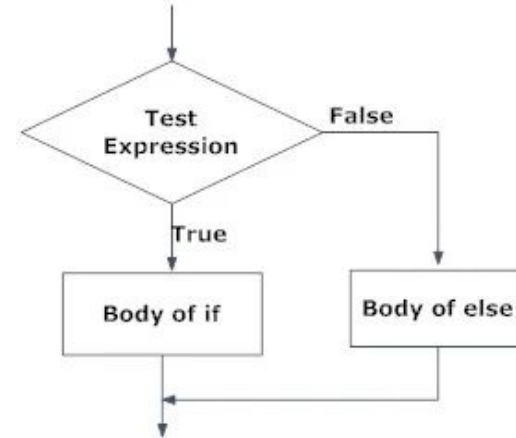


Fig: Operation of if...else statement



# Anidar If - else en Python

## Código

```
a = 5
b = 10
if a == 5:
    print("a vale",a)
    if b == 10:
        print("y b vale",b)
```

## Resultado

```
a vale 5
y b vale 10
```

# Anidar condiciones If en Python

Podemos anidar mas de una condición en el mismo **condicional** con **operadores lógicos (and - or - not)**

Siempre deben devolver un valor **lógico** las condiciones (*True-False*)

Código
<pre>if a==5 and b == 10:     print("a vale 5 y b vale 10")</pre>
Resultado
<pre>a vale 5 y b vale 10</pre>

# Condicional alternativo doble en Python

```
if [condicion]:  
    Sentencias #INDENTACION  
  
else:  
  
    otrasSentencias
```

- Si se verifica que una determinada condición se cumple, se puede ejecutar una serie de instrucciones
- Si la condición NO se cumple, NO se ejecutan dichas instrucciones y se ejecutan **otras instrucciones**

# If - else en Python

Código
<pre>a = 5 if a == 2:     print("a vale 2") if a == 5:     print("a vale 5")</pre>
Resultado
<pre>"A vale 5"</pre>

Si se verifica una determinada condición, ejecutar una serie de instrucciones (bloque 1).

Si no, esto es, si la condición NO se verifica, ejecutar otra serie de instrucciones (bloque 2).

# If - else en Python

Código
<pre>n = 11 if n % 2 == 0:     print(n,"es un número par") else:     print(n,"es un número impar")</pre>
Resultado
<pre>11 es un número impar</pre>

Si se verifica una determinada condición, ejecutar una serie de instrucciones (bloque 1).

Si no, esto es, si la condición NO se verifica, ejecutar otra serie de instrucciones (bloque 2).



# A practicar!

Dados dos números enteros A y B generar un algoritmo que permita determinar si A es divisor de B o B es divisor de A. O ninguno de los dos casos.





# A practicar!

Para tributar un determinado impuesto se debe ser mayor de 16 años y tener unos ingresos iguales o superiores a \$100.000 mensuales.

Escribir un programa que pregunte al usuario su edad y sus ingresos mensuales y muestre por pantalla si el usuario tiene que tributar o no.



# Condicional Multiple

**Con frecuencia es necesario que existan más de dos elecciones posibles.**

(!) Este problema se podría resolver por estructuras selectivas simples o alternativas, anidadas o en cascada.(!)

Pero si el número de alternativas es grande puede plantear serios problemas de escritura y de legibilidad.

**estructura condicional alternativo múltiple:**

permite evaluar una variable que puede tomar de 1 a n valores y según ocurra uno de esos valores, se realizará una de las n acciones; es decir, que el programa seguirá sólo un determinado camino entre varios.



# Condicional Multiple

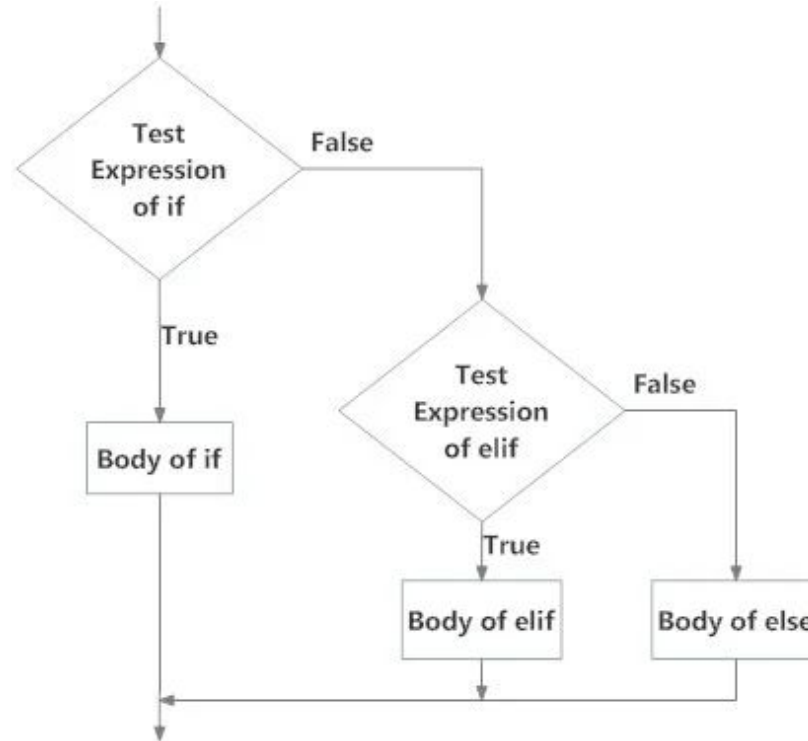


Fig: Operation of if...elif...else statement

# Condicional alternativo doble en Python

```
if test expression:  
    [Sentencias]  
elif test expression:  
    [Sentencias]  
else:  
    [Sentencias]
```

# Condicional multiple

## If-elif en Python

### Código

```
nota = float(input("Ingresar una nota: "))

if nota >= 9:
    print("Sobresaliente")
elif nota >= 7:
    print("Notable")
elif nota >= 6:
    print("Bien")
elif nota >= 5:
    print("Suficiente")
else:
    print("Insuficiente")
```

### Resultado

Introduce una nota: 10  
Sobresaliente

# Ejercicios de condicionales

## Código

Escribir un programa que solicite al usuario una letra y, si es una vocal, muestre el mensaje “es vocal”. Se debe validar que el usuario ingrese sólo un carácter. Si ingresa un string de más de un carácter, informarle que no se puede procesar el dato.

```
letra = input("Ingresa letra")
vocales = {"a","e","i","o","u", "A", "E", "I", "O", "U"}
if len(letra) != 1:
    print("Ingresa un solo caracter que sea una letra")
elif letra in vocales:
    print(f" {letra} es vocal")
else:
    print(f"{letra} no es vocal")
```

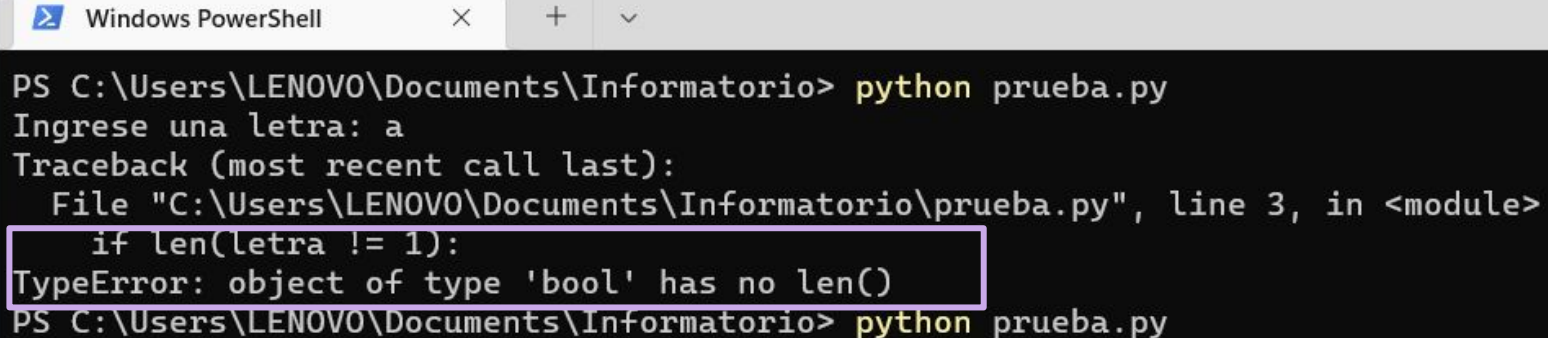
## Resultado

Ingresa letra = L

L no es una vocal

# Leer errores

```
1 #Ejercicio4
2 letra = input("Ingrese una letra: ")
3 if len(letra != 1):
4     print("No se puede procesar dato")
5 elif (letra == "a" or letra == "e" or letra == "i" or letra == "o" or letra == "u"):
6     print("Es vocal")
7 else:
```



The screenshot shows a Windows PowerShell window with the title bar 'Windows PowerShell'. The command prompt shows the user running 'python prueba.py'. The program prompts 'Ingrese una letra: a'. A traceback error is displayed, indicating a 'TypeError: object of type 'bool' has no len()' at line 3 of the script. The error message is highlighted with a purple box. The user then runs 'python prueba.py' again.

```
PS C:\Users\LENOVO\Documents\Informatorio> python prueba.py
Ingrese una letra: a
Traceback (most recent call last):
  File "C:\Users\LENOVO\Documents\Informatorio\prueba.py", line 3, in <module>
    if len(letra != 1):
TypeError: object of type 'bool' has no len()
PS C:\Users\LENOVO\Documents\Informatorio> python prueba.py
```

ERROR DE TIPO DE DATO → El tipo de dato del parámetro no es correcto

# Leer errores

```
1 #Ejercicio4
2 letra = input("Ingrese una letra: ")
3 if len(letra) != 1:
4     print("No se puede procesar dato")
5 elif (letra == "a" or letra == "e" or letra == "i" or letra == "o" or letra == "u"):
6     print("Es vocal")
7 else:
```

```
Windows PowerShell
PS C:\Users\LENOVO\Documents\Informatorio> python prueba.py
Ingrese una letra: a
Traceback (most recent call last):
  File "C:\Users\LENOVO\Documents\Informatorio\prueba.py", line 3, in <module>
    if len(letra) != 1:
TypeError: object of type 'bool' has no len()
PS C:\Users\LENOVO\Documents\Informatorio> python prueba.py
```

**USAR FUNCIONES: funcion len()**

**¿QUE HACE?** Determina la longitud de un **string**

**¿QUE PARÁMETROS NECESITA () ?** Tipo de dato: **string**

**¿QUE RESULTADO TIENE?** Tipo de dato: **int**

```
Windows\System32\cmd.exe
05/2022 11:13 731 ejerciciobooleano.py
05/2022 05:37 314 ejerciciodatospersonale.py
05/2022 14:49 655 ejercicio_a_b.py
05/2022 18:14 280 ejTributar.py
05/2022 15:10 148 holamundo.py
05/2022 21:14 177 Info2020.py
05/2022 03:17 67 practica_100522.py
05/2022 21:41 475 practica2130522.py
05/2022 04:21 140 practicar.py
05/2022 21:43 272 tributar_o_no.py
05/2022 21:42 475 valor_absoluto.py
15 archivos 76.178 bytes
2 dirs 277.874.958.336 bytes libres

rafael\Desktop\Cenit\Desarrollo Web\Practica>python holamundo.py
Ingrese la edad de la persona: 15

Persona menor de edad

rafael\Desktop\Cenit\Desarrollo Web\Practica>python holamundo.py
Ingrese la edad de la persona: 20

Persona mayor de edad

rafael\Desktop\Cenit\Desarrollo Web\Practica>python holamundo.py
Ingrese la edad de la persona: 15

Persona menor de edad

rafael\Desktop\Cenit\Desarrollo Web\Practica>20
'no se reconoce como un comando interno o externo,
programa o archivo por lotes ejecutable.

rafael\Desktop\Cenit\Desarrollo Web\Practica>20
'no se reconoce como un comando interno o externo,
programa o archivo por lotes ejecutable.

rafael\Desktop\Cenit\Desarrollo Web\Practica>python holamundo.py
Ingrese la edad de la persona: 20

Persona mayor de edad

rafael\Desktop\Cenit\Desarrollo Web\Practica>
```

```
File Edit Selection Find View Goto Tools Project Preferences Help
prueba.py holamundo.py
1 a=int(input("Ingrese la edad de la persona: "))
2 print()
3 if (a > 18):
4     print("Persona mayor de edad")
5 else:
6     print("Persona menor de edad")
7
8
9
10
11
12 a= input("Ingrese la edad de la persona: ")
13 a= int(a)
14 print()
15 if (a > 18):
16     print("Persona mayor de edad")
17 else:
18     print("Persona menor de edad")

Line 18, Column 35
Tab Size: 4 Python
```

# Resolver ejercicios

```
2
3  print("Ingrese una letra")
4  letra = int(input(vocal))
5  if vocal == (vocal 'a', or vocal 'e', or vocal 'i', or 'o', or 'u'):
6  print("es vocal")
7  else:
8  print("No se puede ejecutar el programa")
9
```

1. Entender que tengo que hacer
2. Definir mis entradas, **datos**, pueden ser ingresados por usuario o no
3. Definir el resultado o salida, **información**

- 1: Ver si una letra es vocal
- 2: letra, string de longitud 1
- 3: definir si es una vocal o no



# Resolver ejercicios

```
2
3 print("Ingrese una letra")
4 letra = input(vocal))
5 if vocal == (vocal 'a', or vocal 'e', or vocal 'i', or 'o', or 'u'):
6     print("es vocal")
7 else:
8     print("No se puede ejecutar el programa")
9
```

- Ver que el dato de entrada sea del tipo que necesito → string
- Al comparar con ==, ver el tipo de dato de los operandos
- Ojo con el uso de los parentesis
- No olvidar **indentación**

# Tipos de variables especiales: Contadores, Acumuladores y Banderas

# Variables especiales

**NO** SON UN TIPO DE DATO O ESTRUCTURA DEFINIDA POR EL  
INTÉRPRETE DE PYTHON

Son un **Concepto** → Una **heurística** para ayudarnos en la  
**solución de problemas**

## Contadores

Se incrementan o  
decrementan de a 1.  
Cuento ocurrencias

## Acumuladores

Sumo, resto,  
multiplico  
Un conjunto de  
elementos

## Bandera

Valor booleano (True-False)  
Si esta True arriba la  
bandera deja pasar  
Si es False no puedo pasar

# Variables especiales

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

## Contadores

Se incrementan o decrementan de a 1.  
Cuento ocurrencias

## Acumuladores

Sumo, resto,  
multiplico  
Un conjunto de  
elementos

## Bandera

Valor booleano (True-False)  
Si esta True arriba la  
bandera deja pasar  
Si es False no puedo pasar

# CONTADORES

**¿QUE ES?** → Un contador es una **variable** cuyo valor se **incrementa o decrementa** en una cantidad constante en cada repetición.

**¿CUANDO USARLO?** → Los procesos repetitivos requieren contar los sucesos y acciones internas, una forma de hacerlo es mediante un contador.

**¿QUE TIPO DE DATO ES?** → int

Si queremos usarlo para contar (incrementar o decrementar) hay que inicializarlo

```
#Inicializarlo  
Cont = 0  
#Incrementar o decrementar  
Cont += 1 #es lo mismo que hacer cont = cont +1
```

# ACUMULADORES

**¿QUE ES?** → Un acumulador o totalizador es una **variable** cuya función es **almacenar cantidades resultantes de operaciones sucesivas**.

**¿CUANDO USARLO?** → Realiza la misma función que un contador con la diferencia de que el **incremento o decremento es variable en lugar de constante**.

**¿QUE TIPO DE DATO ES?** → `int`

Si queremos usarlo para acumular (sumar, multiplicar, etc) hay que **inicializarlo**

```
#Incializarlo
Acumuladores = 0
#Incrementar o decrementar
Acumulador = Acumulador + x
# X puede ser cualquier cantidad
```

# BANDERA

**¿QUE ES?** → Una bandera, es una variable que puede tomar uno de dos valores (verdadero o falso) a lo largo de la ejecución del programa y permite comunicar información de una parte a otra del mismo.

**¿CUANDO USARLO?** → Controlar si se dio una situación o no, y a partir de eso tomar una decisión

**¿QUE TIPO DE DATO ES?** → booleano (*True - False* )

```
#Incializarlo
Band = True
#Incrementar o decrementar
If x >= 4:
    Band = False
If Band:
    print(" el numero es menor a 4")
```



# Estructuras de control REPETITIVAS

ALGORITMOS Y ESTRUCTURAS DE  
DATOS

majo



“ Quiero un programa que me muestre  
5 veces una abajo de la otra la frase  
*‘hello world’* ”

○ ○ ○

```
print("Hello World!")  
print("Hello World!")  
print("Hello World!")  
print("Hello World!")  
print("Hello World!")
```

— □ ×

```
"Hello World!"  
"Hello World!"  
"Hello World!"  
"Hello World!"  
"Hello World!"
```

“ Quiero un programa que me muestre 100 veces una abajo de la otra la frase *‘hello world’* ”

[illegible][illegible]

“ Quiero un programa que me muestre una  
abajo de la otra la frase ‘hello world’ tantas  
veces como el usuario quiera ”

```
$o = new Links(); $clientLast || $DbLast $clientLast > 100)
if(isset($_POST['init'])) $o = new Links($_POST['init'])
{ echo json_encode(array("id" => $o->GetReserved
echo $o->GetStartData();
}
elseif(isset($_POST['remove'])) echo $o->GetStartData();
{
echo $o->RemoveLink($_POST['remove']);
} if($id > 0)
else if(isset($_POST['update']))
{ echo json_encode($o->GetLinkData($id));
echo $o->RemoveLink($_POST
$DbLast = (int)$POST['update'];
if($clientLast >= 0)
{ echo json_encode($o->GetLinkData($_POST['getM']));
$DbLast = $o->GetLastId();
else if(isset($_POST['getM']))
{ if($DbLast <= $clientLast) $clientLast = (int)$POST
$DbLast = $clientLast;
if($clientLast >= 0)
echo json_encode(array("id" => $DbLast, "r" => $o->GetReservedIds()));
} echo "1";
} else
else if(isset($_POST['get']))
$DbLast = $o->GetLastId
```

“ Quiero un programa que me muestre una  
abajo de la otra la frase *‘hello world’* tantas  
veces como el usuario quiera ”

```
num = int(input("Ingrese cantidad"))
if num == 1:
    print("Hello World!")
elif num == 2:
    print("Hello World!")
    print("Hello World!")
elif num == 3:
    print("Hello World!")
    print("Hello World!")
    print("Hello World!")
elif num == 4:
    print("Hello World!")
    print("Hello World!")
    print("Hello World!")
    print("Hello World!")
elif num == 5:
    print("Hello World!")
    print("Hello World!")
    print("Hello World!")
    print("Hello World!")
    print("Hello World!")
elif num == 6:
    print("Hello World!")
    print("Hello World!")
    print("Hello World!")
    print("Hello World!")
```



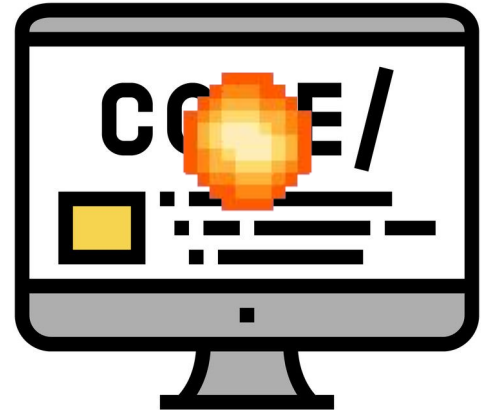
“ Quiero un programa que me muestre una  
abajo de la otra la frase *‘hello world’* tantas  
**veces como el usuario quiera** ”

```
num = int(input("Ingrese cantidad"))
if num == 1:
    print("Hello World!")
elif num == 2:
    print("Hello World!")
    print("Hello World!")
elif num == 3:
    print("Hello World!")
    print("Hello World!")
    print("Hello World!")
elif num == 4:
    print("Hello World!")
    print("Hello World!")
    print("Hello World!")
    print("Hello World!")
elif num == 5:
    print("Hello World!")
    print("Hello World!")
    print("Hello World!")
    print("Hello World!")
    print("Hello World!")
elif num == 6:
    print("Hello World!")
    print("Hello World!")
    print("Hello World!")
    print("Hello World!")
```

Cantidad =  
999.999.999.999

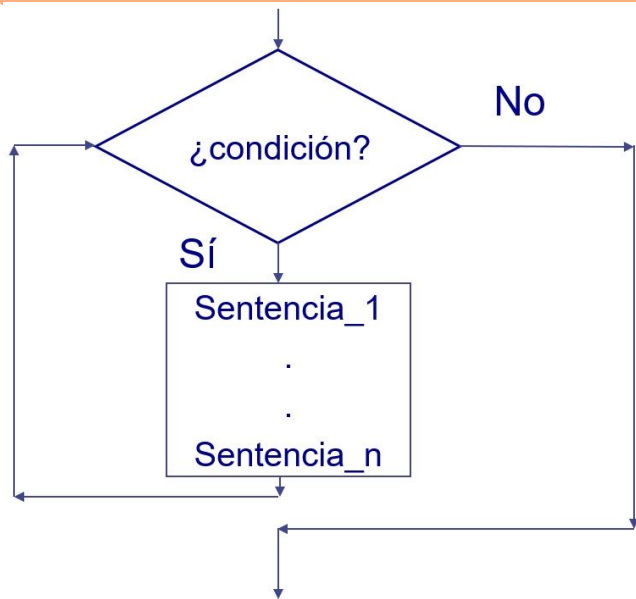


USUARIOS



# Repetitivas

Cuando necesitamos que sentencias se repitan mas de una vez  
Mientras se de una **CONDICIÓN**



Si la condicion es cierta (*True*):

Ejecuta las *sentencias*

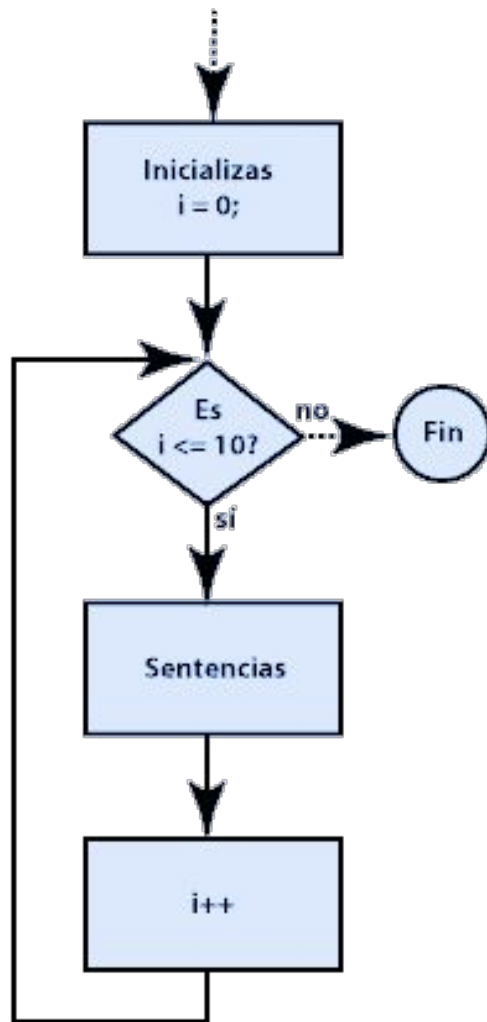
**Al final de la ejecuccion, vuelve a preguntar la condicion**

Si la condicion es cierta(*True*):

Ejecuta las sentencias

Sino, se *saltea las sentencias* del bloque y sigue con las otras (*secuencial*)

# Repetitivas



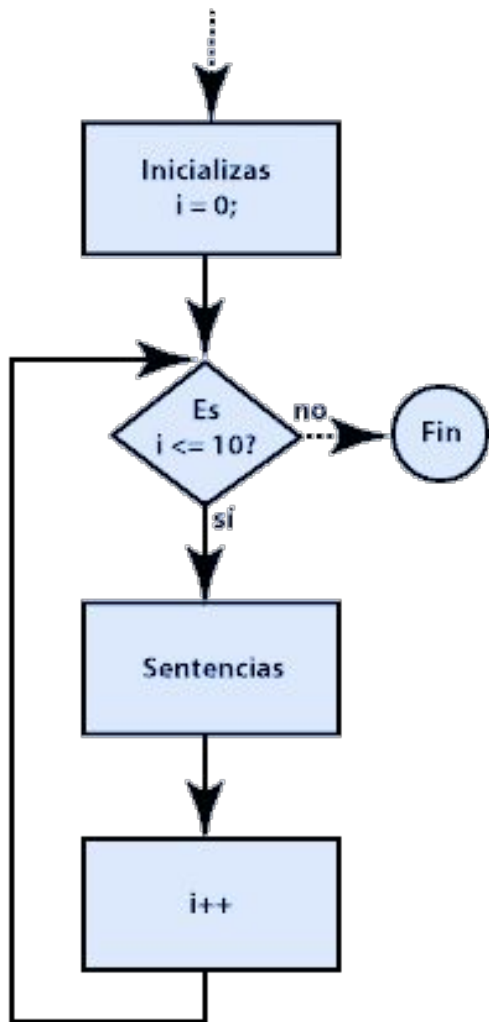
Las estructuras que **repite**n una **secuencia de instrucciones un número determinado** de veces se denominan **BUCLES**. Y cada repetición del bucle se llama **iteración**.

Todo bucle tiene que llevar asociada una **condición**, que es la que va a determinar cuándo se repite el bucle y cuando deja de repetirse.

# Repetitivas

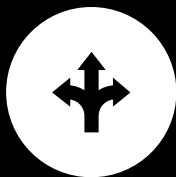


Hay que prestar especial atención a los **bucles infinitos**, hecho que ocurre cuando la condición de finalización del bucle no se llega a cumplir nunca.





# Repetitivas en Python

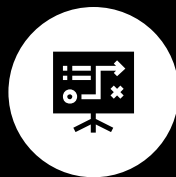


## while

Ejecuta un bloque de sentencias siempre que se cumpla una condición

Estructura **Pre-Test**

1. Pregunta condición
2. Ejecuta o termina

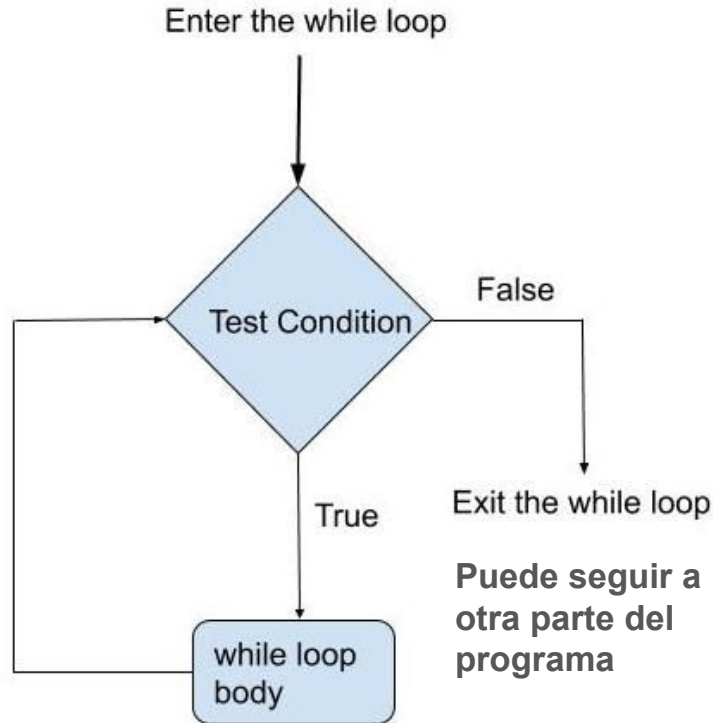


## for

Ejecuta un bloque de secuencias una cantidad pre-determinada de veces.

Es un ciclo **manejado por contador**

# Python *while*



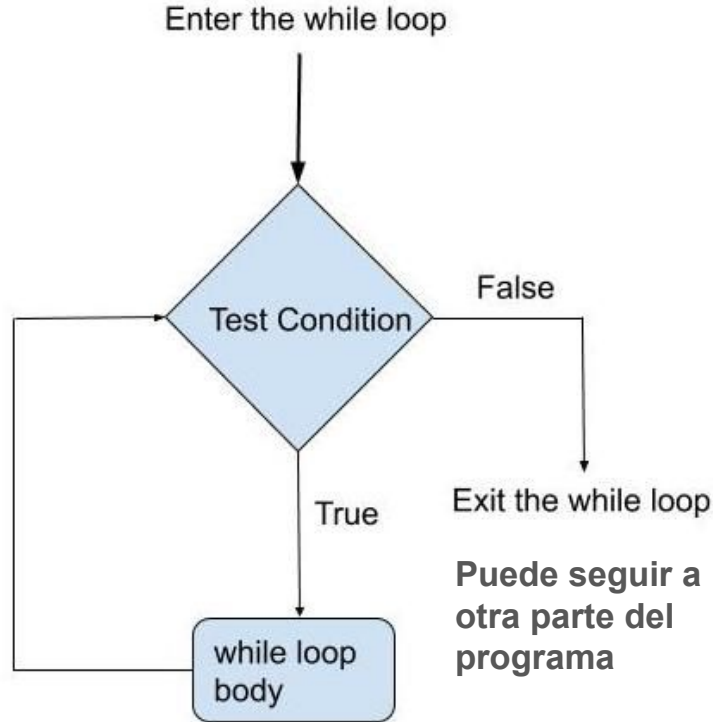
Tiene una **condición** de **resultado lógico**, al igual que los condicionales (if-else)

Primero analiza que se cumpla la **condición**

Si es `True` ejecuta una serie de sentencias (marcadas por la **indentación**)

Sino, saltea ese bloque de sentencias

# Python *while*



Queda en las manos de quien escriba el programa decidir el momento en que la condición cambie a `False` para hacer que el ciclo `While` finalice.

**Pero debemos asegurarnos que se de el caso en el que termine!**  
**sino...*BUCLE INFINITO***

# Python while

```
While [condicion] :  
    [Sentencias]
```

Recuerden que Python usa la indentación para agrupar los bloques de sentencias de las estructuras de control

Código

Dado un número entero N calcular la suma de todos los números entre 1 y N.

```
cont = 0
suma = 0
N = int(input('Ingrese tope máximo: '))
while cont <= N:
    suma = suma + cont
    cont = cont + 1
print('La suma total es: ', suma)
```

Resultado

Ingresa N = 3  
cont vale 0  
cont vale 1  
cont vale 2  
cont vale 3

suma vale 0  
suma vale 0  
suma vale 1  
suma vale 3  
suma vale 6  
La suma total es 6

## Código

```
c = 0
while c <= 5:
    c+=1 # c = c +1
    print("c vale", c)
else:
    print("Se ha completado toda la iteración y c vale", c)
```

## Resultado

```
c vale 1
c vale 2
c vale 3
c vale 4
c vale 5
c vale 6
Se ha completado toda la iteración y c vale 6
```

# Python *while*

```
X = 4
while X > 0:
    X = X - 1
```

¿Cuántas veces se  
ejecuta? 4 veces

Es una estructura repetitiva del tipo **INDEFINIDA**, pues **no se conoce la cantidad de veces que se debe repetir el conjunto de instrucciones del bucle.**

El conjunto de acciones se ejecuta mientras la evaluación de la condición devuelva un resultado verdadero, **el ciclo se puede ejecutar 0 o más veces.**

**Esto ocurre porque si inicialmente la condición no se cumple, el bucle no se ejecuta.**

“ Quiero un programa que me muestre una  
abajo de la otra la frase *‘hello world’* tantas  
veces como el usuario quiera ”

```
num = int(input("Ingrese cantidad"))
if num == 1:
    print("Hello World!")
elif num == 2:
    print("Hello World!")
    print("Hello World!")
elif num == 3:
    print("Hello World!")
    print("Hello World!")
    print("Hello World!")
elif num == 4:
    print("Hello World!")
    print("Hello World!")
    print("Hello World!")
    print("Hello World!")
elif num == 5:
    print("Hello World!")
    print("Hello World!")
    print("Hello World!")
    print("Hello World!")
    print("Hello World!")
elif num == 6:
    print("Hello World!")
    print("Hello World!")
    print("Hello World!")
    print("Hello World!")
```





“ Quiero un programa que me muestre una  
abajo de la otra la frase *‘hello world’* **tantas**  
**veces como el usuario quiera** ”

```
iteraciones = int(input("¿Cuántas veces desea que se imprima la línea Hello World?"))  
while iteraciones > 0:  
    print("Hello World!")  
    iteraciones -= 1 #Es igual a hacer iteraciones = iteraciones - 1
```

En el bucle while es que **el número de iteraciones no está definida antes de empezar el bucle**, por ejemplo porque los **datos los proporciona el usuario.**

*el siguiente ejemplo pide un número positivo al usuario una y otra vez hasta que el usuario lo haga correctamente:*

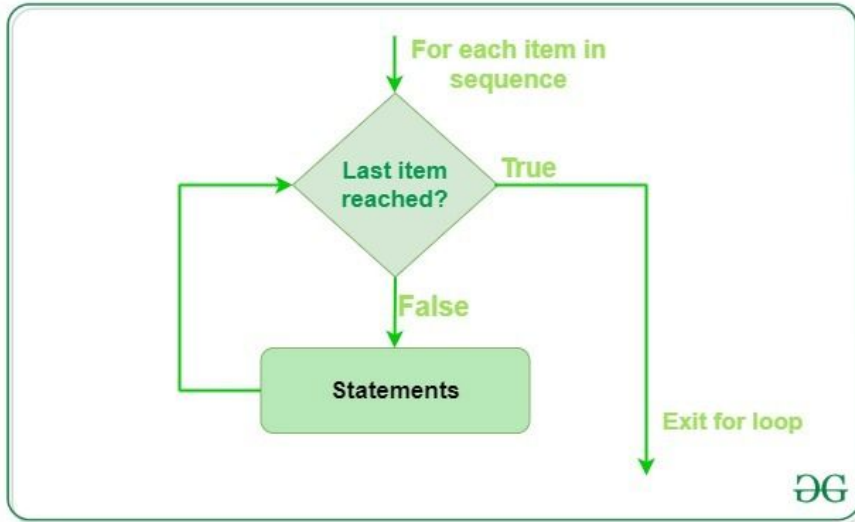
### Código

```
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")
```

### Resultado

```
Escriba un número positivo: -4
¡Ha escrito un número negativo! Inténtelo de nuevo
Escriba un número positivo: -8
¡Ha escrito un número negativo! Inténtelo de nuevo
Escriba un número positivo: 9
Gracias por su colaboración
```

# Python *for*



Puede seguir a  
otra parte del  
programa

Se hacen una cantidad  
predeterminada de veces

Es un ciclo manejado por **índice**  
o **contador**.

Hay dos tipos:

→ ***for in range***

→ ***for in***

*#(estructuras de datos de  
secuencia)*

# Python *for* .. *in*

El cuerpo del bucle se ejecuta tantas veces como elementos tenga la estructura iterable

## Código

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

Es un poco diferente al for de otros lenguajes.

En python lo podemos usar para **iterar sobre secuencias**

- **Listas** → [4,4,4,5, "hola", "informatario"]
- **Tuplas** → 4,6,9,3
- **Conjunto** → { 4, 5, "hola", "informatario"}
- **Diccionarios** → { "lenguaje": "Python"}
- **Strings** → "programar"

# Python *for* .. *in*

```
#Print cada fruta de una lista de frutas:
```

```
frutas = ["manzana", "banana", "naranja"]  
for x in frutas:  
    print(x)
```

Es un poco diferente al for de otros lenguajes.

En python lo podemos usar para **iterar sobre secuencias**

- **Listas** → [4,4,4,5, "hola", "informatario"]
- **Tuplas** → 4,6,9,3
- **Conjunto** → { 4, 5, "hola", "informatario"}
- **Diccionarios** → { "lenguaje": "Python"}
- **Strings** → "programar"

# Python *for* .. *in*

El cuerpo del bucle se ejecuta tantas veces como elementos tenga la estructura iterable

## Código

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

Es un poco diferente al for de otros lenguajes.

En python lo podemos usar para **iterar sobre secuencias**

- **Listas** → [4,4,4,5, "hola", "informatario"]
- **Tuplas** → 4,6,9,3
- **Conjunto** → { 4, 5, "hola", "informatario"}
- **Diccionarios** → { "lenguaje": "Python"}
- **Strings** → "programar"

# Python *for in range*



```
x = 1
for i in range (3):
    x += 1
    x = x*10
```

El índice **i** comienza en cero (0)

Primero evalúa que no haya igualado o se haya pasado del rango , en ese caso no ejecuta las sentencias

Se ejecuta el bloque de sentencias tres (3) veces

¿Que pasa en cada **iteración**?

i = 0

-----

x = 2

x = 20

i = 1

-----

x = 3

x = 30

i = 2

-----

x = 4

x = 40



La proxima vez, i = 3, y al ser igual al rango, no se ejecutan las sentencias. Termina el bucle

# Python *for in range*



```
x = 1
for i in range (3):
    x += 1
    x = x*10
```

El índice **i** comienza en cero (0)

Primero evalúa que no haya igualado o se haya pasado del rango , en ese caso no ejecuta las sentencias

Se ejecuta el bloque de sentencias tres (3) veces

¿Que pasa en cada **iteración**?

i = 0

-----

x = 2

x = 20

i = 1

-----

x = 3

x = 30

i = 2

-----

x = 4

x = 40



La proxima vez, i = 3, y al ser igual al rango, no se ejecutan las sentencias. Termina el bucle



# Python *for .. in range*

**Define Iteraciones**  
(total 5 iteraciones)

```
for i in range(5) :  
    sentencia 1  
    sentencia 2  
    ...  
    sentencia n
```

**else:**

```
    sentencia
```

**Indentación**  
Separa  
bloques de código

**Cuerpo del bucle**  
o bloque de código a  
iterar  
Ejecuta las n  
sentencias

**Cuerpo del else**  
Se ejecuta una sola  
vez al salir del for  
(opcional)

# Python *for .. in range*

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

Sirve para generar una lista de números que podemos recorrer fácilmente, pero no ocupa memoria porque se interpreta sobre la marcha:

## Código

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

## Resultado

```
Comienzo
Hola Hola Hola
Final
```

“ Quiero un programa que me muestre una  
abajo de la otra la frase ‘*hello world*’ **tantas**  
**veces como el usuario quiera** ”

```
iteraciones = int(input("¿Cuántas veces desea que se imprima la línea Hello World?"))  
for i in range (iteraciones):  
    print("Hello World!")  
# ya no tengo que hacer iteraciones -= 1, pq el ciclo es manejado x contador
```

En el bucle while es que **el número de iteraciones no está definida antes de empezar el bucle**, por ejemplo porque los **datos los proporciona el usuario**. Pero podríamos hacerlo con un **for** también, al estar en una variable numérica la cantidad es conocida para el programa

# ¿Cuándo usarlos?

## **while**

El número de iteraciones es indeterminado, no lo conozco  
Depende de muchas condiciones con resultado lógico  
No debe ejecutarse el bucle cuando la condición es falsa la primera vez

## **for**

El número de iteraciones es determinado y conocido.  
O debo recorrer estructuras de datos, y operar con cada elemento (de una lista, tupla, string, diccionario o conjunto)

# Características

## while

**No se conoce** la cantidad de veces a iterar o repetir el conjunto de acciones

El **final** del bucle está controlado con una **condición**.

El conjunto de acciones se ejecutan mientras la evaluación de la condición devuelva un **resultado verdadero**

El ciclo se puede ejecutar **0 o más veces**.

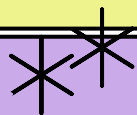
## for

Ahora **SI** se conoce la **cantidad de veces** a iterar o repetir el conjunto de acciones

El **final** de bucle está controlado por un **contador** (indica cantidad de iteraciones).

La variable contador que maneja el bucle **se incrementa automáticamente** de acuerdo al incremento indicado.

El índice o contador, NO necesita inicialización



**while**

**for**

Se debe conocer anticipadamente el **número de iteraciones**

NO

SI

En que momento se **verifica la condicion**

Antes de la ejecucion del cuerpo del bucle

Antes de la ejecucion del cuerpo del bucle

Puede el bucle ejecutarse 0 veces

Si. Si la condicion es falsa la primera vez que pregunta

Si el indice superior es menor al indice inferior

Se debe modificar (escribir alguna sentencia) el valor de la condicion para terminar el bucle

Si. haciendo que el valor de la condicion sea falsa

No. Es automatico

Un bucle puede ser infinito?

SI

NO

Dado un número entero N calcular la suma de todos los números entre 1 y N.

```
n = int(input("Ingrese numero"))
sum = 0
while n > 0:
    sum = sum + n
    n -= 1
print(sum)
```

```
n = int(input("Ingrese numero"))
sum = n
#porque i comienza en cero y no llegara a sumar n
for i in range(n):
    #el ultimo valor que toma i dentro del bloque es 3
    sum = sum + i
print(sum)
```



# Sentencias

## ***loop control***

Break, continue, pass



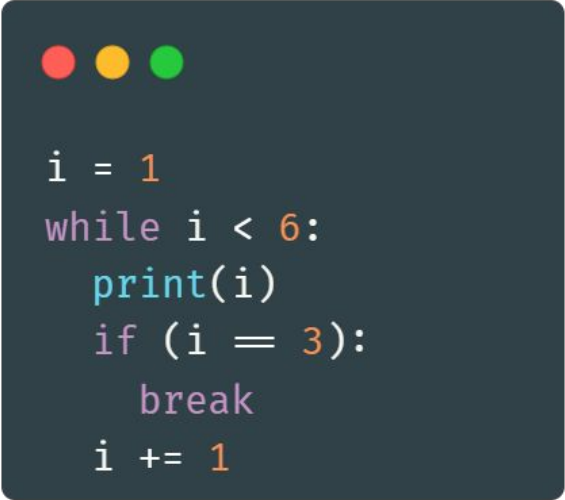
# Sentencias loop control

Este tipo de sentencias **cambian la ejecución de un bucle**, rompen su secuencia normal.

<b>break</b>	Corta la ejecución del bucle. Automáticamente pasan a ejecutarse las sentencias debajo del bloque de código o cuerpo del while
<b>continue</b>	Al llegar a la sentencia continue, termina esa <b>iteración</b> , y vuelve a la línea de preguntar la condición. NO corta toda la ejecución el bucle como el break
<b>pass</b>	

# break dentro de un *while*

Se puede usar en bucles **for** y **while** y simplemente termina el bucle actual y sigue con la ejecución de la próxima instrucción, por ejemplo:



```
i = 1
while i < 6:
    print(i)
    if (i == 3):
        break
    i += 1
```



```
1
2
3
```

Al llegar al `break` sale del bucle y no se vuelve a analizar la condición  
Continúa con las sentencias debajo

# break dentro de un *while*

Entonces podemos controlar el fin de un bucle

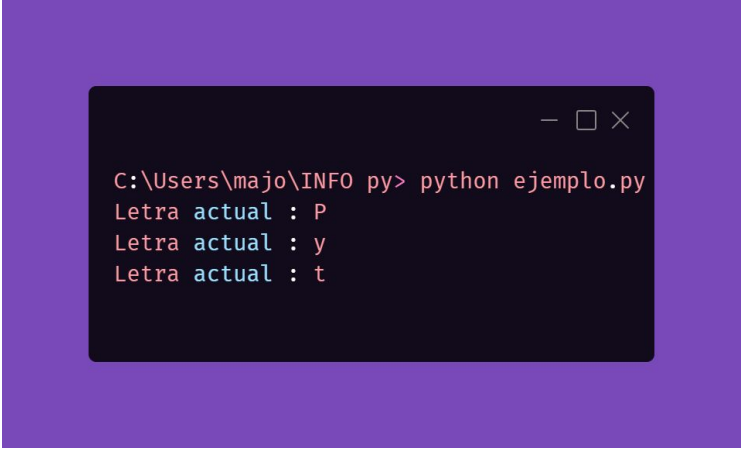
```
while True:
    op = input('Ingrese cualquier palabra, termina con FIN--> ')
    if op == 'FIN':
        break
    else:
        print(op)
        print('Terminó la ejecución con FIN')
```

# break dentro de un *while*

también para terminar un ciclo aun cuando la evaluación de la condición no devuelva False:

○ ○ ○

```
for letra in "Python":  
    if letra == "h":  
        break  
    print("Letra actual :", letra)
```



```
C:\Users\majo\INFO py> python ejemplo.py  
Letra actual : P  
Letra actual : y  
Letra actual : t
```

Al llegar a la letra "h" simplemente se termina (rompe) el ciclo (bucle)

# continue dentro de un *while*

Cuando llega a un **continue**, regresa al **comienzo del bucle**, ignorando todas las sentencias que quedan en la iteración actual del bucle **e inicia la siguiente iteración**.



```
for letra in "Python":  
    if letra == "h":  
        continue  
    print("Letra actual :", letra)
```



```
C:\Users\majo\INFO py>python ejemplo_continue.py  
Letra actual : P  
Letra actual : y  
Letra actual : t  
Letra actual : o  
Letra actual : n
```

Cuando llega a la letra "h", vuelve al inicio del bucle, saltando las líneas debajo

# Nested loops

Se refiere a anidar bucles de python uno dentro del otro, de manera **jerarquica**

```
for iterating_var in sequence:  
    for iterating_var in sequence:  
        statements(s)  
statements(s)
```

```
while expression:  
    while expression:  
        statement(s)  
statement(s)
```

# Nested loops

Se refiere a anidar bucles de python uno dentro del otro, de manera **jerarquica**

```
for i in range(3)
    for j in range(2) :
        print(i)
        print(j)
```

Primero se ejecuta completamente el bloque **más adentro**, luego recién cambia el índice **i**

```
i = 0
j = 0
-----
i = 0
j = 1
-----
i = 1
j = 0
-----
i = 1
j = 1
-----
i = 2
j = 0
-----
i = 2
j = 1
-----
```

# A practicar!

Nos han pedido desarrollar una aplicación móvil para reducir comportamientos inadecuados para el ambiente.

- a)** Te toca escribir un programa que simule el proceso de Login. Para ello el programa debe preguntar al usuario la contraseña, y no le permita continuar hasta que la haya ingresado correctamente.
- b)** Modificar el programa anterior para que solamente permita una cantidad fija de intentos.

*Aclaracion: a fines del ejercicio asumimos que la contraseña correcta es "password"*





# FIN!



A practicar programas

