

# Programación II

---

## Introducción a Python

- Características del Lenguaje
- Tipos básicos
- Operadores
- Funciones
- Entrada Estándar
- Condicionales
- Funciones Recursivas

# Qué es Python?

---

Python es un lenguaje de programación fácil de aprender (al menos, se considera que debería serlo), dinámico, orientado a objetos.

También se lo puede ver como un lenguaje de programación de propósito general y dispone de librerías para diferentes utilidades.



Python fue lanzado por primera vez en 1991, desarrollado inicialmente por Guido van Rossum.

Se inspiró en los lenguajes de programación ABC y Haskell.

Tiene la característica de ser un proyecto de código abierto, administrado por la Python Software Foundation. Hoy en día, Python es mantenido por un numeroso grupo de voluntarios en todo el mundo.

Su nombre es inspirado el la serie The Monty Python de la BBC de Londres.



La filosofía fundamental del lenguaje está resumida en el documento: The Zen of Python (PEP 20) el que incluye aforismos como:

- Beautiful is better than ugly
- Explicit is better than implicit
- Simple is better than complex
- Complex is better than complicated
- Readability counts

# Características del Lenguaje

---

Python es un lenguaje de alto nivel, multiplataforma e interpretado.

El propósito del diseño del lenguaje Python hace hincapié en la productividad del programador y legibilidad del código.

Frecuentemente se lo compara con Tcl, Perl, Scheme y Ruby.

Python puede usarse como lenguaje procedimental o como lenguaje orientado a objetos.

# Características del Lenguaje

---

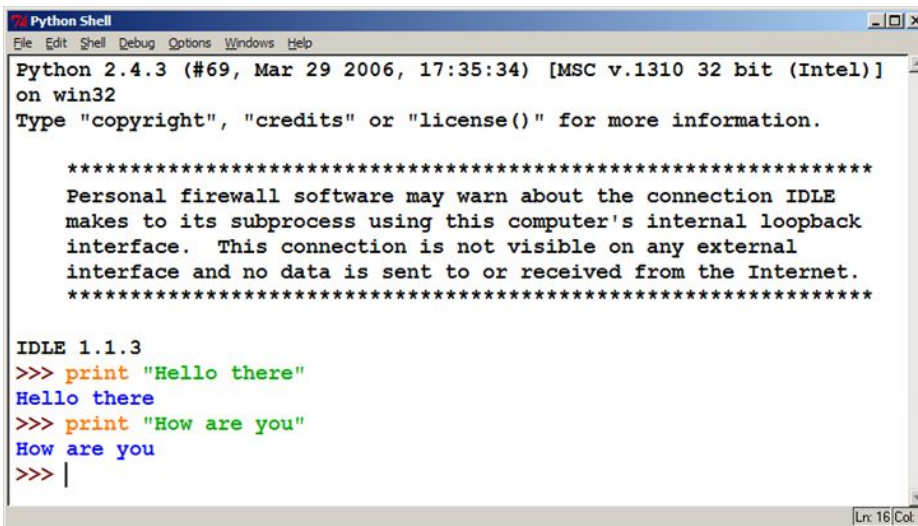
Python ofrece un entorno interactivo para pruebas y depuración.

El entorno de ejecución de Python detecta muchos de los errores de programación que escapan al control de los compiladores y proporciona información para detectarlos y corregirlos.

Posee estructuras de datos que se pueden manipular de modo sencillo.

# Características del Lenguaje

Como mencionamos, Python es un lenguaje interpretado:



```
Python Shell
File Edit Shell Debug Options Windows Help
Python 2.4.3 (#69, Mar 29 2006, 17:35:34) [MSC v.1310 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more information.

*****
Personal firewall software may warn about the connection IDLE
makes to its subprocess using this computer's internal loopback
interface. This connection is not visible on any external
interface and no data is sent to or received from the Internet.
*****

IDLE 1.1.3
>>> print "Hello there"
Hello there
>>> print "How are you"
How are you
>>> |
```

Sin embargo, Python tiene, no obstante, muchas de las características de los lenguajes compilados, por lo que se podría decir que es semi interpretado.





# Características del Lenguaje

---

¿Qué significa esto? En Python, como en Java y otros lenguajes, el código fuente se traduce a un pseudo código máquina intermedio llamado bytecode la primera vez que se ejecuta, generando archivos .pyc o .pyo (bytecode optimizado), que son los que se ejecutarán en sucesivas ocasiones. Esos archivos pueden llevarse a otras computadoras, que cuenten con Python instalado y correrlos sin depender del código fuente original ni del sistema operativo en que fue generado (esto depende de las librerías que se utilicen).

## Características del Lenguaje

---

Actualmente Python tiene dos ramas de desarrollo la rama 2.x y la rama 3.x, estas ramas están actualmente en las versiones Python 2.7.15 y Python 3.7.0.

Hace tiempo que se venía anunciando, desde Python, que la rama 2.x se iba a abandonar. Al día de hoy es prácticamente un hecho. El motivo por el que se fue dilatando la decisión es que ambas ramas son incompatibles entre sí, esto significa que si queríamos migrar un código escrito en la rama 2.x a la 3.x debíamos modificar el código para permitir que funcionara.

Para conocer más acerca de Python 3.x y sus diferencias frente a la rama 2.x podemos ingresar al siguiente sitio web:

<http://docs.python.org/release/3.0.1/whatsnew/3.0.html>

**Para la materia, y los ejemplos a continuación, se asume el uso de Python3.**



## Escribiendo código

---

Para escribir un programa en Python basta con abrir un editor de texto, escribir nuestro código y guardar el archivo con extensión **.py**.

Vamos a escribir hola mundo en Python:

```
print("Hola Mundo")
```

y lo guardamos con el nombre **Hola.py**.

En Linux, hay que agregar la línea, al comienzo, para que el sistema operativo abra el archivo .py con el intérprete adecuado:

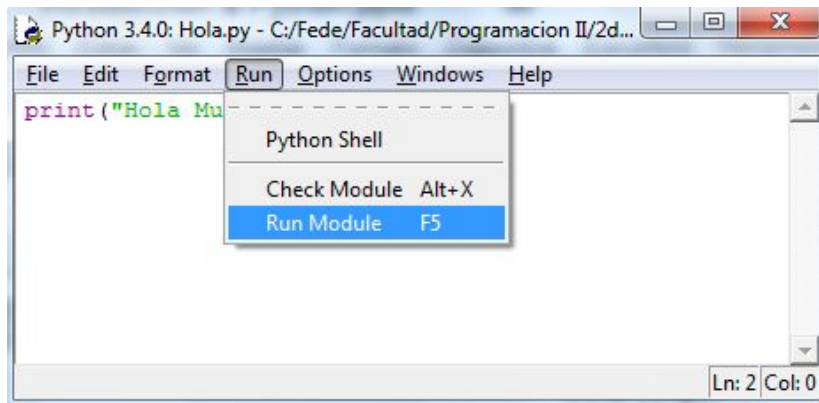
```
#!/usr/bin/python
```

# Escribiendo código

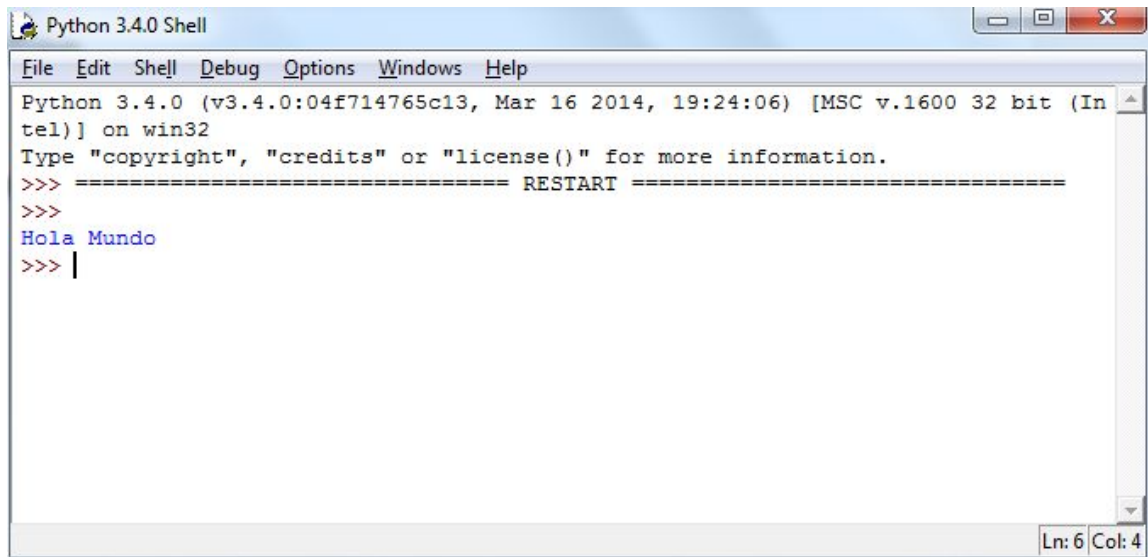
Para ejecutarlo sólo es preciso que en la terminal de Windows o Linux escribamos el comando:

```
python hola.py
```

o si estamos usando el IDLE de Python le digamos en la barra de menu: Run -> Run Module



Visualizando el resultado en el intérprete:



```
Python 3.4.0 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.0 (v3.4.0:04f714765c13, Mar 16 2014, 19:24:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
>>> Hola Mundo
>>> |
```

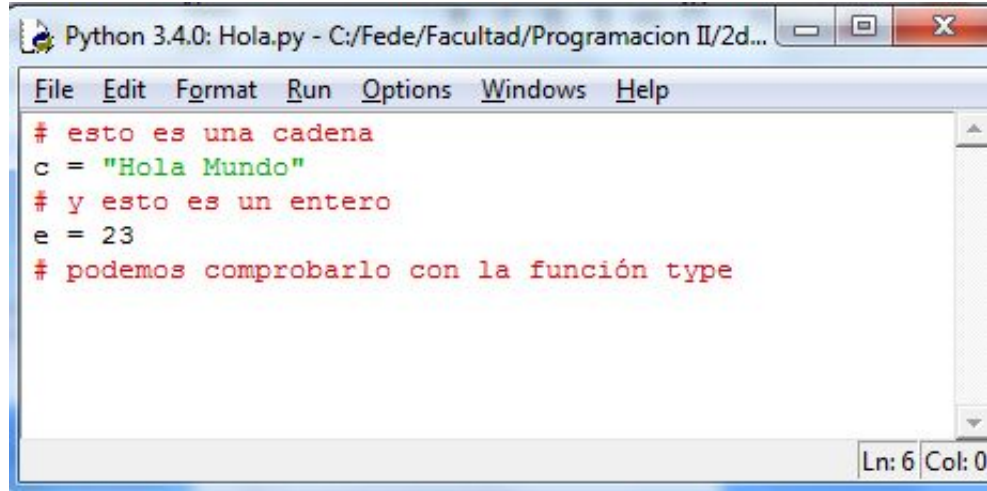
Ln: 6 Col: 4

En Python los tipos básicos se dividen en:

- Números:
  - 3 (int: entero),
  - 4L (long: entero largo)
  - 15.57 (float: de coma flotante)
  - 7 + 5j (complex: complejos)
- Cadenas de texto, como "Hola Mundo";
- Valores booleanos: true y false

# Tipos básicos

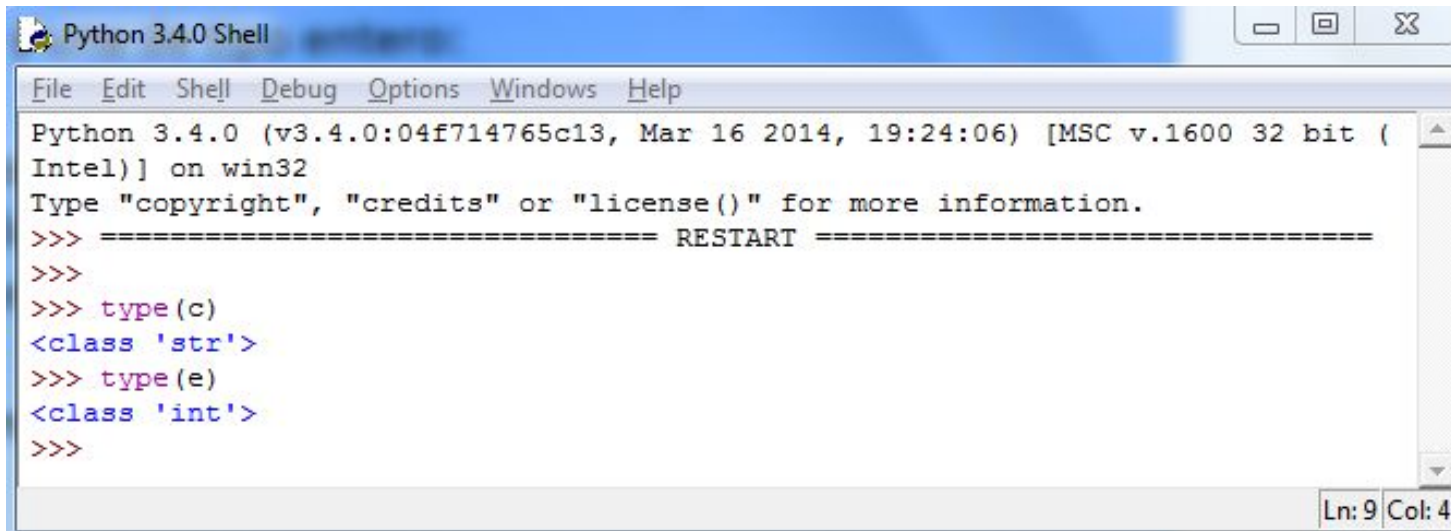
Vamos a crear un par de variables a modo de ejemplo, una de tipo String (cadena) y otra de tipo Int (entero).



```
Python 3.4.0: Hola.py - C:/Fede/Facultad/Programacion II/2d...
File Edit Format Run Options Windows Help
# esto es una cadena
c = "Hola Mundo"
# y esto es un entero
e = 23
# podemos comprobarlo con la función type
Ln: 6 Col: 0
```

# Tipos básicos

Ahora, validemos el tipo de cada una de esas variables usando la función `type`:



```
Python 3.4.0 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.0 (v3.4.0:04f714765c13, Mar 16 2014, 19:24:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
>>> type(c)
<class 'str'>
>>> type(e)
<class 'int'>
>>>
```

Como se puede ver, el comentario inline se antecede con `#`.



# Operadores aritméticos

Los operadores aritméticos para los tipos numéricos:

Operador	Descripción	Ejemplo
+	Suma	<code>r = 3 + 2</code> # r es 5
-	Resta	<code>r = 4 - 7</code> # r es -3
-	Negación	<code>r = -7</code> # r es -7
*	Multiplicación	<code>r = 2 * 6</code> # r es 12
**	Exponente	<code>r = 2 ** 6</code> # r es 64
/	División	<code>r = 3.5 / 2</code> # r es 1.75
//	División entera	<code>r = 3.5 // 2</code> # r es 1.0
%	Módulo	<code>r = 7 % 2</code> # r es 1

Cuando se mezclan tipos de números, Python convierte todos los operandos al tipo más complejo de entre los tipos de los operandos que forman la expresión.

Para usar estos comandos, al comienzo del programa se debe escribir:

```
from math import *
```

Function name	Description
<code>abs(value)</code>	Valor absoluto
<code>ceil(value)</code>	Redondea para arriba
<code>cos(value)</code>	Coseno, en radianes
<code>floor(value)</code>	Redondea, para abajo
<code>log10(value)</code>	Logaritmo, base 10
<code>max(value1, value2)</code>	Mayor de dos valores
<code>min(value1, value2)</code>	Mínimo de dos valores
<code>round(value)</code>	Entero más cercano
<code>sin(value)</code>	Seno, in radians
<code>sqrt(value)</code>	Raíz cuadrada

# Constantes matemáticas

---

Para usar estos comandos, al comienzo del programa se debe escribir:

```
from math import *
```

Constant	Description
<code>e</code>	2.7182818...
<code>pi</code>	3.1415926...

# Operadores Booleanos

Estos son los distintos tipos de operadores con los que podemos trabajar con valores booleanos, los llamados operadores lógicos o condicionales.

Operador	Descripción	Ejemplo
and	¿se cumple a y b?	<code>r = True and False # r es False</code>
or	¿se cumple a o b?	<code>r = True or False # r es True</code>
not	No a	<code>r = not True # r es False</code>

# Operadores Relacionales

Los valores booleanos son además el resultado de expresiones que utilizan operadores relacionales (comparaciones entre valores):

Operador	Descripción	Ejemplo
==	¿son iguales a y b?	<code>r = 5 == 3 # r es False</code>
!=	¿son distintos a y b?	<code>r = 5 != 3 # r es True</code>
<	¿es a menor que b?	<code>r = 5 &lt; 3 # r es False</code>
>	¿es a mayor que b?	<code>r = 5 &gt; 3 # r es True</code>
<=	¿es a menor o igual que b?	<code>r = 5 &lt;= 5 # r es True</code>
>=	¿es a mayor o igual que b?	<code>r = 5 &gt;= 3 # r es True</code>

Los Strings o cadenas no son más que texto encerrado entre comillas simples ('cadena') o dobles ("cadena"). Dentro de las comillas se pueden añadir caracteres especiales escapándolos con `\`, como `\n`, el carácter de nueva línea, o `\t`, el de tabulación.

Una cadena puede estar precedida por el carácter `r`, el cual indica que se trata de una cadena Raw(del inglés, cruda). Las cadenas raw se distinguen de las normales en que los caracteres escapados mediante la barra invertida (`\`) no se sustituyen por sus contrapartidas.

También es posible encerrar un String entre triples comillas (simples o dobles). De esta forma podremos escribir el texto en varias líneas, y al imprimir la cadena, se respetarán los saltos de línea que introdujimos sin tener que recurrir al carácter `\n`, así como las comillas sin tener que escaparlas.

# Operadores de Strings

## Indexar ([ ])

Los caracteres de un String son asociados a un índice, veamos un ejemplo de esto:

```
name = "P. Diddy"
```

lo que, internamente, se termina representando como:

index	0	1	2	3	4	5	6	7
character	P	.		D	<u>i</u>	d	d	y

por lo que:

```
>>> print(name, " comienza con ", name[0])  
P. Diddy comienza con P
```



# Operadores de Strings

---

De lo anterior podemos decir:

- Para acceder a un caracter individual del string se utiliza: `variable[index]`
- El usar `,` como separador en un string nos permite mostrar una cadena formada por diferentes datos, los cuales pueden ser de diferente tipo.

```
>>> print(" 3+4 da como resultado ", 7)
      3+4 da como resultado  7
```

# Operadores de Strings

---

## Concatenación (+)

sólo se pueden concatenar dos strings:

```
>>> saludo = "Hola " + "mundo"
>>> print(saludo)
Hola mundo
>>> cuenta = "3 + 4 = " + 7
Traceback (most recent call last):
  File "<pyshell#13>", line 1, in <module>
    cuenta = "3 + 4 = " + 7
TypeError: Can't convert 'int' object to str implicitly
```

# Operadores de Strings

---

## Multiplicación (\*)

sólo se pueden multiplicar un string y un entero:

```
>>> nombre = "Federico"  
>>> nombre*5  
'FedericoFedericoFedericoFedericoFederico'
```

En otros lenguajes de programación los bloques de código se determinan encerrándolos entre algún carácter o palabra.

Por ejemplo, en Racket se usan los paréntesis () para marcar el ámbito, en C se usan llaves, en Pascal las palabras begin y end.

En Python no se utilizan caracteres o palabras clave sino que se usa la indentación (usando tabulaciones) para marcar el ámbito de un bloque.

Es decir, si escribimos la definición de una función, las sentencias dentro de la función se escriben una tabulación dentro de la misma.

De esta forma se obliga a los programadores a indentar su código para que sea más sencillo de leer :)

Una función es un fragmento de código con un nombre asociado que realiza una serie de tareas y devuelve un valor. A los fragmentos de código que tienen un nombre asociado y no devuelven valores se les suele llamar procedimientos. En Python no existen los procedimientos, ya que cuando el programador no especifica un valor de retorno la función devuelve el valor **None** (nada).

En Python también existen lo que se conoce como funciones anónimas las cuales no cuentan con un nombre que las identifique, impidiendo su reutilización. A lo largo de las próximas slides veremos ejemplos de esto.

En Python las funciones se declaran de la siguiente forma:

```
def mi_funcion(param1, param2):  
    print param1  
    print param2
```

Es decir, la palabra clave **def** seguida del nombre de la función y entre paréntesis los argumentos separados por comas es la sintaxis para definir una función, luego aparecen los **:** que marcan el inicio del bloque de definición.

A continuación, en otra línea, tabulando tendríamos las líneas que conforman el código a ejecutar por la función.

También podemos encontrarnos con una cadena de texto como primera línea del cuerpo de la función. Estas cadenas se conocen con el nombre de **docstring** (cadena de documentación) y sirven, como su nombre indica, a modo de documentación de la función.

```
>>> def mi_funcion(param1, param2):  
    """Esta funcion imprime los dos valores pasados como parametros"""  
    print(param1)  
    print(param2)  
  
>>> mi_funcion(1, "hola")  
1  
hola  
>>> help(mi_funcion)  
Help on function mi_funcion in module __main__:  
  
mi_funcion(param1, param2)  
    Esta funcion imprime los dos valores pasados como parametros
```

Python, al igual que algunos lenguajes, permiten asignar valores por defecto para los parámetros, en caso que los mismos no sean pasados como argumento. Esto se hace situando un signo igual después del nombre del parámetro y, a continuación, el valor por defecto:

```
>>> def imprimir(texto, veces = 1):  
        print(vuces * texto)  
  
>>> imprimir("Hola", 2)  
HolaHola  
>>> imprimir("Hola")  
Hola  
>>>
```



Otra característica que tiene Python, no tan común, es la de poder identificar el valor de cada parámetro sin respetar el orden de la definición. Es decir, puedo hacer que el orden de los argumentos en la llamada sea diferente que el se encuentra definido. Esto se consigue indicando el argumento qué valor toma:

```
>>> def saludar(nombre, saludo="hola "):  
    print(saludo, nombre)  
  
>>> saludar(saludo= "Buen día", nombre= "Federico")  
Buen día Federico
```

Para que la función retorne un valor se debe utilizar la palabra clave **return**:

```
>>> def sumar(x, y):  
        return x + y
```

```
>>> sumar(2, 3)  
5
```

```
>>> z = sumar(2, 3)
```

```
>>> z  
5
```

Como vemos esta función, no hace otra cosa que sumar los valores pasados como parámetro y devolver el resultado como valor de retorno. Este valor, lo podemos almacenar en una variable para usarlo luego.

## Entrada estándar

La forma más sencilla de obtener información por parte del usuario es mediante la función **input**. Esta función toma como parámetro una cadena a usar como prompt (es decir, como texto a mostrar al usuario pidiendo la entrada) y devuelve una cadena con los caracteres introducidos por el usuario hasta que pulsó la tecla Enter. Veamos un pequeño ejemplo:

```
>>> def saludo():
    nombre = input("Como te llamas? ")
    print("Encantado, " + nombre)

>>> saludo()
Como te llamas? Federico
Encantado, Federico
...
```



## Entrada estándar

---

Si necesitáramos un entero como entrada en lugar de una cadena, por ejemplo, podríamos utilizar la función `int` para convertir la cadena a entero, aunque sería conveniente tener en cuenta que puede lanzarse una excepción si lo que introduce el usuario no es un número.



## Entrada estándar

```
>>> def cantidadDias():
    edad = input("Cuantos años tienes? ")
    dias = int(edad) * 365
    print("Has vivido " + str(dias) + " días")

>>> cantidadDias()
Cuantos años tienes? 43
Has vivido 15695 días
>>> cantidadDias()
Cuantos años tienes? cuarenta y tres
Traceback (most recent call last):
  File "<pyshell#53>", line 1, in <module>
    cantidadDias()
  File "<pyshell#51>", line 3, in cantidadDias
    dias = int(edad) * 365
ValueError: invalid literal for int() with base 10: 'cuarenta y tres'
```

# Condicionales

---

En Python existe, como en casi todos los lenguajes de programación una estructura de control donde, dependiendo del valor de una condición se ejecuta una (o varias) sentencias.

La sintaxis de esta estructura es:

```
if condición1:
    bloque si se cumple condición1
elif condición2:
    bloque si se cumple condición2
else:
    bloque en caso contrario
```

La opción **elif** y la **else**, es opcional y, es un equivalente a hacer otro if, en el caso que la condición anterior sea falsa.

Se puede poner 1 o más elif si se lo requiere la expresión pero, si existe, hay un único else.

En Python, para aquellos que conozcan otros lenguajes de programación, no existe la construcción switch que en otros lenguajes está presente.

Veamos algunos ejemplos de su uso:

```
>>> def tipoNumero(numero):  
    if numero < 0:  
        print("Negativo")  
    elif numero > 0:  
        print("Positivo")  
    else:  
        print("Cero")  
  
>>> tipoNumero(-1)  
Negativo  
>>> tipoNumero(0)  
Cero  
>>> tipoNumero(4)  
Positivo
```



También podríamos haber escrito:

```
>>> def tipoNumero(numero):  
    if numero < 0:  
        print("Negativo")  
    elif numero > 0:  
        print("Positivo")  
    elif numero == 0:  
        print("Cero")
```

```
>>> tipoNumero(-1)  
Negativo  
>>> tipoNumero(0)  
Cero  
>>> tipoNumero(4)  
Positivo
```

Usando sólo else:

```
>>> def tipoNumero(numero):  
    if numero < 0:  
        print("Negativo")  
    else:  
        if numero > 0:  
            print("Positivo")  
        else:  
            print("Cero")  
  
>>> tipoNumero(-1)  
Negativo  
>>> tipoNumero(0)  
Cero  
>>> tipoNumero(4)  
Positivo
```

# Funciones Recursivas

A continuación escribiremos algunas funciones recursivas como ejemplo.

La primera es el factorial, esta implementación de factorial que proponemos tiene el inconveniente que, si el número ingresado es negativo se va a romper (probarlo!!!!). Escriba una variante de esta función que no tenga ese problema.

```
>>> def factorial(n):  
    if n == 0:  
        return 1  
    else:  
        return n * factorial(n-1)  
  
>>> factorial(5)  
120
```

## Funciones Recursivas

---

Vamos a escribir una función que permita el ingreso de números enteros hasta que se ingrese un 0. Cuando se detecte el ingreso de un 0, la función debe mostrar la suma de los números que se ingresaron y la cantidad (sin incluir al último 0).

# Funciones Recursivas

Una solución posible es:

```
>>> def sumaNumeros(suma = 0, contador = 0):
    n = int(input("Ingrese un número o 0 para finalizar: "))
    if n == 0:
        print("La suma de los valores ingresados fue", suma, "y se ingresaron", contador, "numeros")
    else:
        sumaNumeros(suma+n, contador+1)

>>> sumaNumeros()
Ingrese un número o 0 para finalizar: 1
Ingrese un número o 0 para finalizar: 2
Ingrese un número o 0 para finalizar: 5
Ingrese un número o 0 para finalizar: 10
Ingrese un número o 0 para finalizar: 0
La suma de los valores ingresados fue 18 y se ingresaron 4 numeros
```