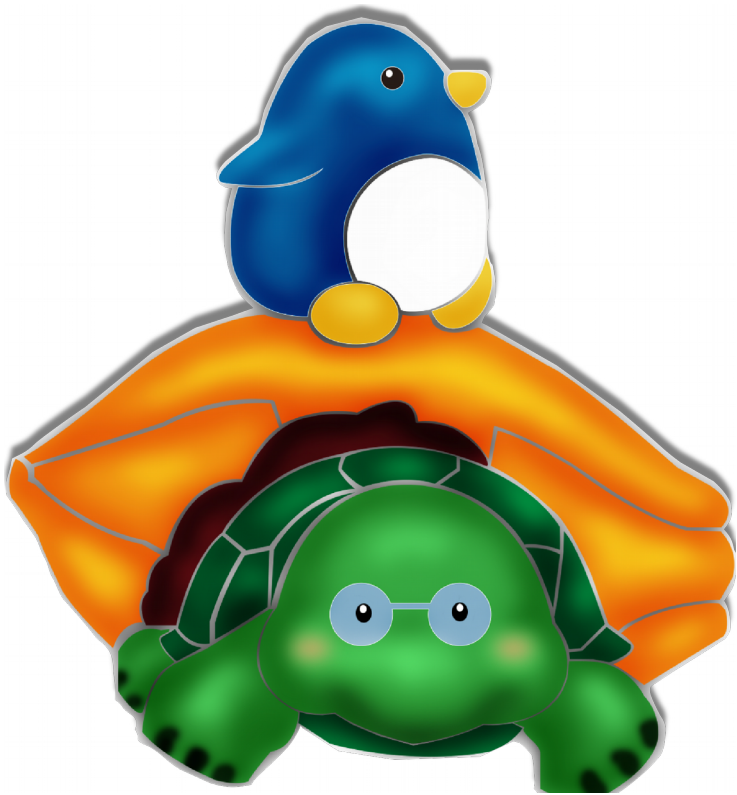


# TALLER DE PYTHON



***BIENVENIDOS....***

## Introducción

Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible.

Se trata de un lenguaje de programación multiparadigma

Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma.

## Función print()

```
print("hola mundo")
```

```
prueba1 = "hola mundo"
```

```
print(prueba1.center(50," "))
```

```
prueba2 = 45
```

```
print(prueba1,prueba2)
```

## Tipos de datos

Existen varios tipos de datos en python entre estos tenemos:

`dato_entero = 943`

`int`

`dato_flotante = 943.0`

`float`

`dato_completo = 943 + 943j`

`complex`

## input

La función `input()` permite obtener texto escrito por teclado. Al llegar a la función, el programa se detiene esperando que se escriba algo y se pulse la tecla Intro

```
cadena = input("Ingrese cadena: ")
```

```
entero = int(input("Ingrese entero: "))
```

```
flotante = float(input("Ingrese float: "))
```

## Operaciones básicas

Suma +

Multiplicación \*

División entera //

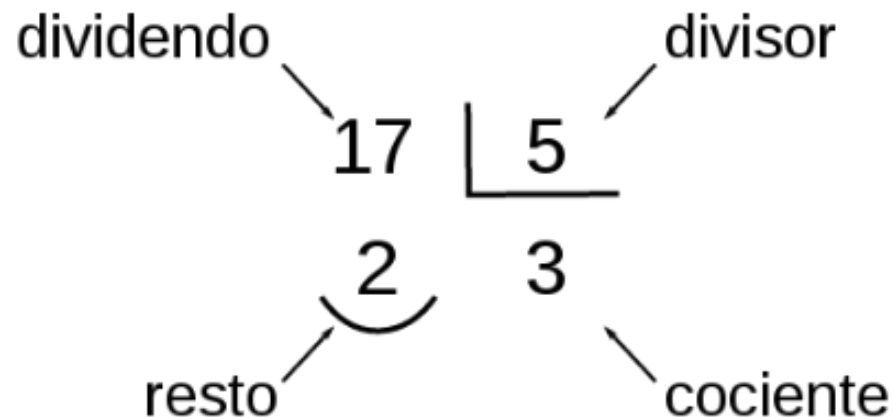
Potenciación \*\* o pow(x,y)

Resta -

División /

Módulo %

dividendo = divisor \* cociente + resto



## Ejemplo

Cree un programa que permita el ingreso de un número entero de dos dígitos. Su programa deberá dividir el número en dígitos y mostrarlos por pantalla.

```
Ingrese numero dos digitos: 25  
Unidad: 5  
Decena: 2
```

## Funciones para redondear

```
round(4.29)
```

```
>>4
```

```
round(8.87.1)
```

```
>>8.9
```

```
from math import floor
```

```
floor(5.89)
```

```
>>5
```

```
from math import ceil
```

```
ceil(9.11)
```

```
>>10
```



## Más tipos de datos

Otros tipos de datos nuevos son:

True, False

Boolean

cadena = "cursos de python"

String

Concatenar string

cadena = "cursos de python"

s = "aquí"

print(cadena+" "+s)

# Asignar valores de forma simultánea

```
a,b = 2,4
```

```
cadena = "hola"
```

```
flotante = 1.5
```

```
print ("Entero: %d" % entero)
```

```
print ("Cadena: %s" % cadena)
```

```
print ("Decimal: %f" % flotante)
```

```
print ("Decimal: %f, %s" % (flotante,cadena))
```

# Operaciones de String

Operaciones con string:

Concatenación de string (+) ej:

```
print("hola" + " mundo")
```

Repetición (\*):

```
hola = "hola"
```

```
print(hola*2) #holahola
```

# Operaciones de String

`a[1]` #Obtiene el carácter especificado en el paréntesis

`a[1:4]` obtiene el rango de caracter especificado en los corchetes

`in.-` Devuelve True si el carácter pertenece a la cadena 'h' in "hola"

## Funciones de string

<code>str(num)</code>	Transforma el argumento en string
<code>len(string)</code>	Retorna el tamaño del string
<code>String.lower()</code>	Retorna el string en minúscula
<code>String.upper()</code>	Retorna el string en mayúscula
<code>String.capitalize()</code>	Retorna la primera letra en mayúscula y el resto en minúscula
<code>str.title()</code>	Retorna el string hecho titulo.

## Funciones de string

String.isnumeric() Retorna True si la cadena es solo numeros

String.isalnum() Retorna True si son solo letras

String.islower() Retorna True si son minúscula

String.isupper() Retorna True si son mayúscula

String.istitle() Retorna True si cada palabra esta en mayuscula

## Funciones de string

`String.swapcase()` Invierte todos las mayúscula y minúsculas

`String.find(str)` Determina si un caracter es parte del string, y si lo es devuelve la posición, sino lo es devuelve -1

`String.max(str)` Retorna la mayor letra del alfabeto.

`String.min(str)` Retorna la menor letra del alfabeto.

## Funciones de string

`String.replace(old, new)` Reemplaza lo que está en old, con lo que está en new.

`String.split(str)` Retorna una lista de string, separados por el carácter puesto entre paréntesis

`String.splitlines()` Retorna una lista de string, separados por el salto de línea



# Listas

Las listas son conjuntos ordenados de elementos (números, cadenas, listas, etc). Las listas se delimitan por corchetes ([ ]) y los elementos se separan por comas.

Las listas pueden contener elementos del mismo tipo:

```
primos = [2, 3, 5, 7, 11, 13]
```

```
diasLaborables = ["Lunes", "Martes"]
```

# Listas



O pueden contener elementos de tipos distintos:

fecha = ["Lunes", 27, "Octubre", 1997]

O pueden contener listas:

peliculas = [ ["Senderos de Gloria", 1957],  
["Hannah y sus hermanas", 1986]]

# Listas

[koko<sup>a</sup>a]

## Concatenar listas

Las listas se pueden concatenar con el símbolo de la suma (+):

```
vocales = ["E", "I", "O"]
```

```
vocales = vocales + ["U"]
```

El operador suma (+) necesita que los dos operandos sean listas:

# Listas

[koko**a**]

También se puede utilizar el operador += para añadir elementos a una lista:

```
vocales = ["A"]
```

```
vocales += ["E"]
```

```
>>> vocales
```

```
['A', 'E']
```

Manipular elementos individuales de una lista

Cada elemento se identifica por su posición en la lista, teniendo en cuenta que se empieza a contar por 0.

```
>>>fecha = [27, "Octubre", 1997]
```

```
>>>fecha[0]
```

```
27
```

Se pueden utilizar números negativos (el último elemento tiene el índice -1 y los elementos anteriores tienen valores descendentes):

```
>>> fecha = [11, "Noviembre", 2016]
```

```
>>> fecha[-1]
```

```
2016
```

# Listas



Se puede modificar cualquier elemento de una lista haciendo referencia a su posición:

```
>>> fecha = [27, "Octubre", 1997]
```

```
>>> fecha[2] = 1998
```

## Manipular sublistas

De una lista se pueden extraer sublistas, utilizando la notación:

nombre[inicio:límite] donde inicio y límite hacen el mismo papel que en el tipo range(inicio, límite).

```
dias = ["Lunes", "Martes", "Miércoles",  
"Jueves", "Viernes", "Sábado", "Domingo"]
```

```
dias[1:4] # Se extrae una lista con los valores  
1, 2 y 3
```

```
['Martes', 'Miércoles', 'Jueves']
```



## Manipular sublistas

Se puede modificar una lista modificando sublistas. De esta manera se puede modificar un elemento o varios a la vez e insertar o eliminar elementos.

```
letras = ["A", "B", "C", "D", "E", "F", "G", "H"]
```

```
letras[1:4] = ["X"]      # Se sustituye la sublista  
['B','C','D'] por ['X']
```

Al definir sublistas, Python acepta valores fuera del rango, que se interpretan como extremos (al final o al principio de la lista).

```
>>> letras = ["D", "E", "F"]
```

```
>>> letras[3:3] = ["G", "H"]    # Añade ["G", "H"]  
al final de la lista
```

## La palabra reservada del

Permite eliminar un elemento o varios elementos a la vez de una lista, e incluso la misma lista.

```
>>> letras = ["A", "B", "C", "D", "E", "F", "G"]
```

```
>>> del letras[4] # Elimina la sublista ['E']
```

```
>>> del letras[1:3] # Elimina la sublista ['B', 'C']
```

```
>>> letras
```

```
['A', 'F', 'G']
```

# [koko]a

## Estructuras de Control

### Identación


En un lenguaje informático, la identación es lo que la sangría al lenguaje humano escrito (a nivel formal).

Una estructura de control, entonces, se define de la siguiente forma:

inicio de la estructura de control:

expresiones

# Condicionales

Símbolo	Significado	Ejemplo	Resultado
<code>==</code>	Igual que	<code>5 == 7</code>	<code>False</code>
<code>!=</code>	Distinto que	<code>rojo != verde</code>	<code>True</code>
<code>&lt;</code>	Menor que	<code>8 &lt; 12</code>	<code>True</code>
<code>&gt;</code>	Mayor que	<code>12 &gt; 7</code>	<code>True</code>
<code>&lt;=</code>	Menor o igual que	<code>12 &lt;= 12</code>	<code>True</code> 
<code>&gt;=</code>	Mayor o igual que	<code>4 &gt;= 5</code>	<code>False</code>

# Condicionales

[koko<sup>a</sup>]

Operador	Ejemplo	Explicación	Resultado
and	<code>5 == 7 and 7 &lt; 12</code>	False and False	False
and	<code>9 &lt; 12 and 12 &gt; 7</code>	True and True	True
and	<code>9 &lt; 12 and 12 &gt; 15</code>	True and False	False
or	<code>12 == 12 or 15 &lt; 7</code>	True or False	True
or	<code>7 &gt; 5 or 9 &lt; 12</code>	True or True	True
xor	<code>4 == 4 xor 9 &gt; 3</code>	True o True	False
xor	<code>4 == 4 xor 9 &lt; 3</code>	True o False	True

## If-elif-else

Las estructuras de control de flujo condicionales, se definen mediante el uso de tres palabras claves reservadas, del lenguaje: if (si), elif (sino, si) y else (sino).

## If-elif-else

1) Si semáforo esta en verde, cruzar la calle.  
Sino, esperar.

```
if semaforo == verde:  
    print "Cruzar la calle"  
else:  
    print "Esperar"
```



# Estructuras Iterativas

A diferencia de las estructuras de control condicionales, las iterativas (también llamadas cíclicas o bucles), nos permiten ejecutar un mismo código, de manera repetida, mientras se cumpla una condición.

En Python se dispone de dos estructuras cíclicas:

El bucle while

El bucle for

# Bucle while



Este bucle, se encarga de ejecutar una misma acción "mientras que" una determinada condición se cumpla. Ejemplo: Mientras que año sea menor o igual a 2012, imprimir la frase "Informes del Año año".

```
anio = 2001
```

```
while anio <= 2012:
```

```
    print "Informes del Año", str(anio)
```

```
    anio += 1
```

# Bucle while

[koko<sup>a</sup>]

```
while True:
```

```
    nombre = raw_input("Indique su nombre: ")
```

```
    if nombre:
```

```
        break
```

# Bucle for [kokoaa]

Un bucle for es un bucle que repite el bloque de instrucciones un número predeterminado de veces.

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

# Bucle for [koko<sup>a</sup>]

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

# `range(ini, fin)`

# [koko<sup>a</sup>]

El argumento que le vamos a pasar, será un entero llamado 'n'. Esta función creará una lista creciente de números desde el 0 hasta el (n-1). Por lo tanto, la función se llamaría de la siguiente manera:

`range(n)` Por ejemplo: `>> range(5)` `[0, 1, 2, 3, 4]`

**range(ini, fin)**

**[koko<sup>a</sup>]**

Lo escribiremos de la siguiente manera:

range(m, n)

Por ejemplo:

```
>> range(5, 10) [5, 6, 7, 8, 9]
```

**range(ini, fin)**

**[koko<sup>a</sup>]**

Lo escribiremos de la siguiente manera:

`range(m, n, k)`

Por ejemplo:

```
>> range(0, 10, 3) [0, 3, 6, 9]
```



# Contador [koko<sup>a</sup>]

Se entiende por contador una variable que lleva la cuenta del número de veces que se ha cumplido una condición.

```
print("Comienzo")
```

```
cuenta = 0
```

```
for i in range(1, 6):
```

```
    if i % 2 == 0:
```

```
        cuenta = cuenta + 1
```

```
print("Desde 1 hasta 5 hay", cuenta, "múltiplos de 2")
```

# Acumulador

[koko<sup>a</sup>]

Se entiende por acumulador una variable que acumula el resultado de una operación.

```
print("Comienzo")
```

```
suma = 0
```

```
for i in [1, 2, 3, 4]:
```

```
    suma = suma + i
```

```
print("La suma de los números de 1 a 4 es",  
suma)
```