Introducción al Python

> Esta es **una muy breve introducción al Python** como recordatorio de algunas instrucciones si ya has utilizado este lenguaje.
>
> Si es la primera vez, te recomendamos que visites nuestro curso **PYTHON PARA TODOS**
>
> **Python for everybody** por Charles R. Severance licencia CC-BY-NCSA que empieza desde cero.

Lenguajes, intérpretes y compiladores

Python es un lenguaje **de alto nivel** destinado a ser relativamente sencillo para que los humanos lean y escriban y para que los ordenadores lean y procesen. Otros lenguajes de alto nivel incluyen Java, C ++, PHP, Ruby, Basic, Perl, JavaScript y muchos más. El hardware real dentro de la Unidad Central de Procesamiento (CPU) no comprende ninguno de estos lenguajes de alto nivel.

La CPU entiende un idioma que llamamos **lenguaje de máquina**. El lenguaje de máquina es muy simple y francamente muy tedioso de escribir porque está representado en ceros y unos:

El lenguaje de máquina parece bastante simple en la superficie, dado que solo hay ceros y unos, pero su sintaxis es aún más compleja y mucho más compleja que Python. Muy pocos programadores escriben lenguaje de máquina. En su lugar, creamos varios traductores para permitir que los programadores escriban en lenguajes de alto nivel como Python o JavaScript y estos traductores convierten los programas al lenguaje de máquina para su ejecución real por parte de la CPU.

Estos traductores de lenguaje de programación se dividen en dos categorías generales: (1) intérpretes y (2) compiladores.

Un **intérprete** lee el código fuente del programa como está escrito por el programador, analiza el código fuente e interpreta las instrucciones sobre la marcha. Python es un intérprete y cuando ejecutamos Python de forma interactiva, podemos escribir una línea de Python (una oración) y Python la procesa de inmediato y está lista para que escribamos otra línea de Python.

```
>>> x = 6
>>> print(x)
6
```

Copyright 2026

```
>>> y = x * 7
>>> print(y)
42
>>>
```

Está en la naturaleza de un **intérprete** poder tener una conversación interactiva como se muestra arriba. A un **compilador** debemos entregarle todo el programa en un archivo, y luego ejecuta un proceso para traducir el código fuente de alto nivel al lenguaje de máquina y luego el compilador coloca el lenguaje de máquina resultante en un archivo para su posterior ejecución.

Variables

Las variables son como cajas que puedes meter valores. Y los valores pueden ser de varios **tipos** :

- **int** si son enteros
- **float** si tienen decimales
- **binario**Deben comenzar por 0b. Por ejemplo: 0b110, 0b11
- **string** son frases, son "***cadenas***" de caracteres entre "
- **bool** Solamente hay dos literales booleanos True o False
- **lista** Se pueden declarar variables que son conjuntos por ejemplo Colores = ["verde", "rojo", "naranja"]

Para crear una variable puedes usar cualquier palabra, x, y, z o Nombre_alumno ... pero algunas palabras no puedes usar, ver

Para visualizar variables puedes usar la **instrucción print** poniendo entre paréntesis el valor o variable que quieres visualizar.

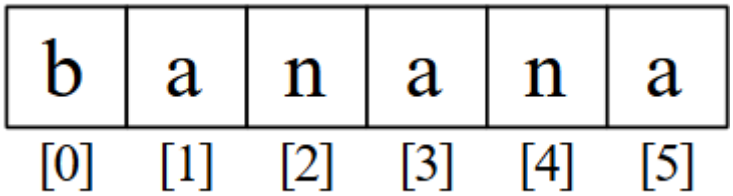
> Modifica los valores como quieras, es un **intérprete**, juega y dale al play para ver el resultado

Como puedes ver se ha introducido un operador el + que realiza la suma del valor de x original (43) y se le incrementa una unidad resultando en la impresión un 44.

Cadenas

Cadenas son secuencias de caracteres, por ejemplo la palabra "banana"

Copyright 2026



fuente 'Python for Everybody' por Charles R. Severance

Se puede obtener su longitud con la función len, o obtener un carácter ...

Operadores

Este apartado de operadores es adaptado de Federico Coca Guia de Trabajo de Microbit CC-BY-SA

Los **operadores aritméticos** se utilizan para realizar operaciones matemáticas como sumas, restas, multiplicaciones, etc.

Operador	Descripción	Ejemplo
+	Suma o concatenación en textos	`5+3=8`, `"Hola" + "Mundo" = "Hola Mundo"`
-	Diferencia	`6-3=3`
*	Multiplicación	`3*3=9`
/	División	`6/2=3`
//	Parte entera de un cociente	`10//3=3`
%	Resto de un cociente	`10%3=1`
**	Potenciación	`5**2=25`

Los **operadores de asignación** se utilizan para asignar valores a variables.

Operador	Descripción	Ejemplo
=	Asignación	`x=4`, `a = a + 1`
+=	Suma y asignación	`x+=1` equivale a `x = x + 1`
-=	Diferencia y asignación	`x-=1` equivale a `x = x - 1`
*=	Multiplicación y asignación	`x*=3` equivale a `x = x * 3`

Copyright 2026

Operador	Descripción	Ejemplo
/=	División y asignación	`x/=3` equivale a `x = x / 3`
%=	Asignación de restos	`x%=3` equivale a `x = x % 3`
**=	Asignación de exponentes	`x**=3` equivale a `x = x ** 3`

Los **operadores de comparación** comparan dos valores/variables y devuelven un resultado booleano: Verdadero o Falso `True` o `False`.

Operador	Descripción	Ejemplo
==	Igual a	`2==3` retorna `False`
!=	Distinto de	`2!=3` retorna `True`
<	Menor que	`2<3` retorna `True`
>	Mayor que	`2>3` retorna `False`
<=	Menor o igual que	`2<=3` retorna `True`
>=	Mayor o igual que	`2>=3` retorna `False`

Los **operadores lógicos** se utilizan para comprobar si una expresión es Verdadera o Falsa. Se utilizan en la toma de decisiones.

Operador	Descripción	Ejemplo
and	AND lógica	`a and b #True si a y b son ciertos`
or	OR lógica	`a or b #True si a o b son ciertos`
not	NOT lógica	`not a #True si el operador a es falso`
in	pertenencia	Devuelve True si pertenece
no int	no pertenencia	Devuelve True si no pertenece
is	identidad	Devuelve True si son iguales
is not	no identidad	Devuelve True si no son inguales

Los **operadores bit a bit** o bitwise actúan sobre los operandos como si fueran cadenas de dígitos binarios. Operan bit a bit:

Operador	Descripción	Ejemplo

Copyright 2026

&	AND bit a bit	`5&6 # 101 & 110 = 110 = 4`
|	OR bit a bit	`5 | 6 # 101 | 110 = 111 = 7`
~	NOT bit a bit	`~3 # ~011 = 100 = -4`
^	XOR bit a bit	`5^3 # 101^011 = 110 = 6`
<<	Desplazamiento izquierda	`4<<1 # 100 << 1 = 1000 = 8`
>>	Desplazamiento derecha	`4 >> 1 # 100 >> 1 = 010 = 2`

Comentarios en Python
Una sola línea : Escribiendo el símbolo almohadilla (#) delante del comentario.
Multilínea: Escribiendo triple comillas dobles (“””) al principio y al final del comentario.

Entradas de teclado

Ya hemos visto salidas por pantalla con **print**, pero ahora con input puede leer variables del teclado.
Fíjate que hay que poner las líneas **x = float (x)** e **y = float(y)** para convertirlos a números decimales, en caso contrario las interpreta string y no puede multiplicar en Resultado, pero en el siguiente ejemplo **no es necesario en la variable cel** (celsius) pues se multiplica por números decimales 32.0 5.0 y 9.0

try y **except** son dos funciones que son *un seguro para el programador* por si el usuario en vez de teclear un número, mete un string o carácter

La sangría es importante en Python
La sangría se refiere a los espacios al comienzo de una línea de código. Mientras que en otros lenguajes de programación la sangría en el código es solo para facilitar la lectura, la sangría en Python es muy importante ya que se usa para indicar un bloque de código.

Condicionales

Las instrucciones **if: else:** son las que nos permiten realizar operaciones según las condiciones puestas. *Ojo con la sangría*

\n es un carácter especial que significa "Salto de página"

Copyright 2026

Bucles

- **while** ejecuta lo contenido en la sangría mientras sea verdadero la condición
- **for** ejecuta lo contenido en la sangría mientras y va recorriendo la variable dentro del rango creado

Para verlo mejor vamos a ver estos ejemplos

- EJEMPLO BUCLE WHILE
  - mientras n sea positivo va ejecutando : imprime n y lo decrementa
  - al decrementar llega un momento que deja de ser positivo y finaliza el bucle
- EJEMPLO BUCLE WHILE INFINITO
  - Es muy típico en robótica, todo el rato hace el bucle (en robótica para que lea los sensores y realice cosas en los actuadores) pero este ejemplo no esta en un robot sino en tu pc y no queremos que se quede "colgado" luego al teclear "fin" acaba gracias a la instrucción **break**
  - Fíjate que hay una instrucción **continue** para que pase a la siguiente iteración provocando que no imprime lo tecleado
- EJEMPLO BUCLE FOR FRIENDS
  - Va recorriendo la variable friend dentro del cojunto lista friends
  - como puedes ver la diferencia entre for y while es que for además recorre la variable
- EJEMPLO BUCLE FOR
  - mientras n este en el rango de 0 a 5 se ejecuta

Funciones

No vamos a entrar en detalle, pero observa el siguiente código

- **FUNCIONES PREDEFINIDAS** Si observas, la primera línea llama a importar una librería externa, **import math** donde math es un fichero que tienen funciones predefinidas, vamos a utilizar una de ellas, la raiz cuadrada **sqrt** luego para llamar a esa función que esta definida dentro de math se hace con la instrucción **math.sqrt**
- **FUNCIONES DEFINIDAS POR TI** em este caso, se utiliza la palabra **def** para crear una función, que le vamos a pasar tres argumentos a, b y c y para finalizar la función usamos **return** para devolver el valor que queremos obtener

Para saber más de Python

CURSO PYTHON FOR EVERYBODY en español	ver
Curso completo de Python 222pag pdf (*)	Descargar

Copyright 2026

Curso completo de Python 422pag (*)	Descargar
Curso completo de Python desde 0 (*)	Ver
Curso de Python desde 0 (*)	Ver
Manual de referencia Python (*)	Ver
Programación en Python (*)	Ver
Trabajando con ficheros en Python (*)	Ver
Programación orientada a objeto en Python (*)	Ver
un manual para aquellos usuarios con previos conocimientos de Python, como la programación modular y orientada a objetos. También algunos conocimientos de las librerías **tkinter** (Para crear interfaces gráficos y **SQlite3** (para gestionar bases de datos). (*)	Descargar

(*) Agradecimientos a Pere Manel http://peremanelv.com

Revision #1
Created 2026-11-17 11:34:22 CET by Javier Quintana
Updated 2026-11-17 11:34:36 CET by Javier Quintana

Copyright 2026