# Introducción a Python para Principiantes

Guía paso a paso con ejemplos sencillos para quienes nunca han programado.

## 1. Introducción.

- ¿Qué es Python?

  Es un lenguaje de programación de código abierto y de uso gratuito que se caracteriza por tener una sintaxis sencilla y fácil de comprender. Su facilidad de aprendizaje permite que personas con distintos perfiles profesionales puedan utilizarlo para crear soluciones, automatizar tareas o procesos y trabajar con análisis de datos, incluso sin tener experiencia previa como desarrolladores de software.

- ¿Para qué se utiliza?

  Python tiene múltiples aplicaciones, entre las más comunes se encuentran el desarrollo de aplicaciones web, la automatización de tareas o procesos, el análisis y procesamiento avanzado de datos principalmente en la Inteligencia de Negocios (BI) y en el desarrollo de soluciones relacionadas con la Inteligencia Artificial.

- ¿Por qué Python es una buena opción para comenzar a programar?

  Porque es una excelente opción para comenzar a programar porque su sintaxis es relativamente sencilla y permite concentrarse en aprender los conceptos fundamentales de la programación sin tener que lidiar inicialmente con una sintaxis demasiado compleja.

  Además, Python puede ayudarnos a automatizar procesos y tareas repetitivas dentro de nuestro entorno laboral, así como a desarrollar soluciones que faciliten determinadas actividades y permitan ofrecer respuestas más eficientes a nuestros clientes.

- ¿Qué aprenderás en esta guía?

  En esta guía aprenderás los conceptos fundamentales de Python y la sintaxis necesaria para comenzar a programar desde cero.

  A través de ejemplos sencillos y ejercicios prácticos, aprenderás a:

  - ✓ Escribir y ejecutar tus primeros programas en Python.
  - ✓ Utilizar variables y diferentes tipos de datos.
  - ✓ Trabajar con operadores y realizar operaciones matemáticas.
  - ✓ Recibir información proporcionada por el usuario.
  - ✓ Utilizar condiciones para tomar decisiones.

- ✓ Crear estructuras repetitivas mediante bucles.
- ✓ Trabajar con listas y otros tipos de datos.
- ✓ Crear y utilizar funciones.
- ✓ Manejar errores básicos.
- ✓ Integrar los conocimientos aprendidos en pequeños proyectos prácticos.

No es necesario tener experiencia previa en programación. La guía está diseñada para avanzar progresivamente desde los conceptos más básicos hasta la creación de pequeños programas.

- ¿Qué necesitas para empezar?

Para comenzar solo necesitas:

- ✓ Una computadora con acceso a Internet.
- ✓ Instalar Python en tu computadora.
- ✓ Un editor de código o entorno de desarrollo donde puedas escribir y ejecutar tus programas.
- ✓ Disposición para practicar y experimentar con los ejemplos de esta guía.

## 2. Preparando nuestro entorno

- ✓ Instalar Python. Puedes hacer la descarga desde la url oficial Download Python | Python.org
- ✓ Comprobar que Python funciona accediendo a Python desde el botón de inicio de la computadora.
- ✓ Elegir dónde escribir código (si utilizas otra herramienta).

## 3. Primeros conceptos de programación

- ¿Que es una variable?

Es un mecanismo (nombre) que utilizamos para almacenar un valor en memoria, por ejemplo:
nombre = “Juan”
edad = 25

Puedes cambiar su valor e imprimir el resultado (verlo en pantalla).
La idea de guardar información en una variable significa asignarle un dato a un nombre para poder usarlo después.

Te muestro cómo realizarlo:

Entra a tu herramienta de Python y escribe lo siguiente:
nombre = “Ana” → **Tu variable**
print(“Hola”, nombre) → **print para ver el resultado en pantalla y “Hola” para colocar un texto delante de tu variable.**
**Resultado: Hola Ana**

```
>>> nombre = "Ana"
>>> print("Hola", nombre)
Hola Ana
```

Otro ejemplo con número

```
>>> edad = 18
>>> print("Tengo", edad)
Tengo 18
```

También

```
>>> edad = 25
>>> print ("Tengo", edad, "años")
Tengo 25 años
```

## 4. Tipos de datos básicos

- **Texto (str):** se utiliza para guardar palabras o frases. Ejemplo: nombre = “Ana”
- **Números enteros (int):** se utilizar para almacenar números enteros, puede ser número positivo o negativo. Ejemplos: edad = 20 / temperatura = -5
- **Decimales (float):** tipo de dato utilizado para trabajar con valores en decimales. Ejemplos: precio = 100.98 / altura = 1.75
- **Verdadero/Falso (boolean):** los booleanos solo pueden almacenar dos valores, true (verdadero) o false (falso). Ejemplos: es_estudiante = true / es_empleado = false

## 5. Recibir información del usuario

A diferencia de los ejemplos vistos anteriormente, en los que nosotros proporcionamos los datos, en esta sección solicitamos la información para que sea el usuario quien la introduzca.

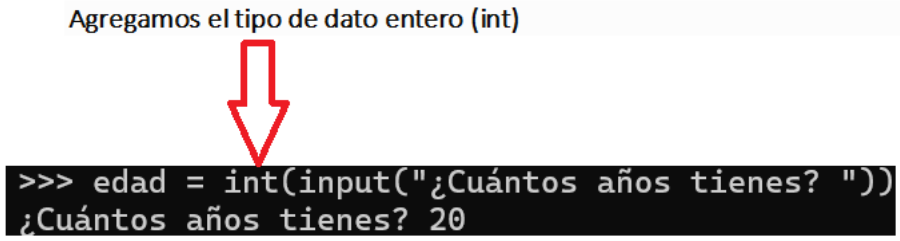
Para este caso utilizamos la instrucción input(). Ejemplo:

```
>>> nombre = input("¿Cuál es tu nombre? ")
¿Cuál es tu nombre? Juan
```

Es importante tomar en cuenta que input() almacena como texto los datos que escribimos, así sea un número, ejemplo:

```
>>> edad = input("¿Cuántos años tienes? ")
¿Cuántos años tienes? 20
```

Por tanto, si queremos que nuestro resultado sea exactamente un valor numérico, debemos especificarlo de la siguiente manera:




Para fortalecer lo aprendido, te invito ha realizar el siguiente ejercicio práctico:

 Crea un programa que pregunte el nombre y la edad del usuario y muestre un mensaje personalizado.

## 6. Operaciones matemáticas

En Python podemos realizar operaciones matemáticas básicas muy fácil. Por ejemplo:

Suma

```
>>> a = 20
>>> b = 10
>>> print(a + b)
30
```

Resta

```
>>> a = 20
>>> b = 15
>>> print(a - b)
5
```

Multiplicación

```
>>> a = 5
>>> b = 3
>>> print(a * b)
15
```

División

```
>>> a = 10
>>> b = 3
>>> print(a / b)
3.3333333333333335
```

Si queremos realizar una división solo con enteros utilizamos doble /.

```
>>> a = 10
>>> b = 3
>>> print(a // b)
3
```

También podemos utilizar el operador % para obtener el resto de una división, como en el ejemplo anterior que 10/3 es igual a 3, pero sobra 1.

```
>>> a = 10
>>> b = 3
>>> print(a % b)
1
```

Otro operador es el de potencia **, si queremos elevar un numero a una potencia x, también lo podemos realizar en este lenguaje.  Ejemplo:

```
>>> print(2 ** 3)
8
```

Significa 2 x 2 x 2 = 8

Algo importante a tomar en cuenta es que Python respeta el orden matemático en las operaciones, por ejemplo:

```
>>> resultado = 2 + 3 * 4
>>> print(resultado)
14
```

Acá Python realiza la multiplicación primero 3 * 4 y luego le suma el 2, por tanto, el resultado es 14. Si queremos que el resultado sea lo contrario, debemos realizarlo de la siguiente manera:

```
>>> resultado = (2 + 3) * 4
>>> print(resultado)
20
```

Nota: mayormente en las operaciones matemáticas utilizamos variables. Ejemplos:

```
>>> precio = 100
>>> cantidad = 2
>>> total = precio * cantidad
>>> print(total)
200
```

```
>>> precio = 100
>>> descuento = 20
>>> precio_final = precio - (precio * descuento / 100)
>>> print(precio_final)
80.0
```

 Ejercicio para calcular el precio de una compra:

Una persona compra:

- 3 camisetas a 150 pesos cada una.
- 2 pantalones a 550 pesos cada uno.
- La tienda aplica un 10% de descuento sobre el total.

Escribe un programa en Python que calcule:

1. El precio de las camisetas.
2. El precio de los pantalones.
3. El precio total antes del descuento.
4. El valor del descuento.
5. El precio final.

Si quieres fortalecer un poco más sobre aplicar lo aprendido en las operaciones matemáticas, te invito a realizar el siguiente mini proyecto.

 Crea una calculadora sencilla.

## 7. Trabajar con condiciones

En Python la condición **if** se utiliza para ejecutar un bloque de código cuando una condición es verdadera. Por ejemplo:

```
>>> edad = 18
>>> if edad >= 18:
...  print("Eres mayor de edad")
...
Eres mayor de edad
```

Para el caso de **if...else** se utiliza cuando tenemos dos posibilidades. Por ejemplo:

```
>>> edad = 12
>>> if edad >= 18:
...  print("Eres mayor de edad")
... else:
...  print("Eres menor de edad")
...
Eres menor de edad
```

Con la condición **if...elif...else** la utilizamos para comprobar varias condiciones. Por ejemplo:

```
>>> nota = 85
>>> if nota >= 90:
...  print("Excelente")
... elif nota >= 70:
...  print("Aprobado")
... else:
...  print("Reprobado")
...
Aprobado
```

## 8. Operadores de comparación

Son utilizados para comparar dos valores y conocer el resultado si es verdadero (true) o falso (false). Veamos cada uno de ellos con un ejemplo.

### Igual que (==)

```
>>> a = 10
>>> b = 10
>>> print(a == b)
True
```

Diferente de (!=)

```
>>> a = 10
>>> b = 12
>>> print(a != b)
True
```

Mayor que (>)

```
>>> edad = 22
>>> print(edad > 30)
False
```

Menor que (<)

```
>>> numero = 5
>>> print(numero < 1)
False
```

Mayor o igual que (>=)

```
>>> nota = 70
>>> print(nota >= 70)
True
```

Menor o igual que (<=)

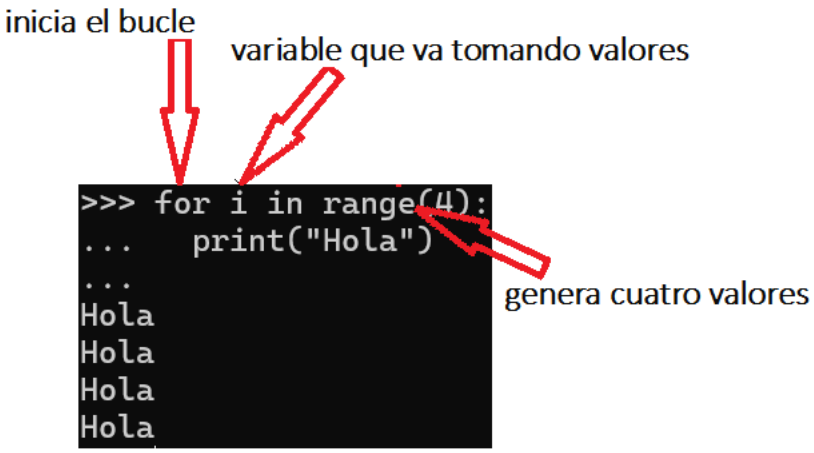
```
>>> nota = 80
>>> print(nota <= 70)
False
```

## 9. Repetir acciones con bucles

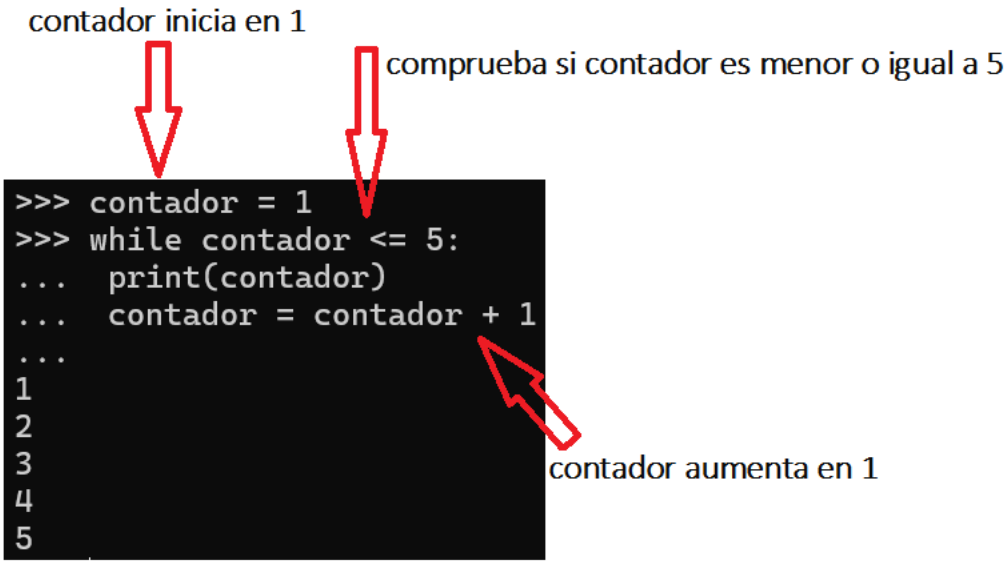
Los bucles sirven para repetir acciones varias veces sin tener que escribir el mismo código varias veces.

Existen dos tipos principales de bucles:

- ✓ **for:** Repite una acción una cantidad determinada de veces o recorre una secuencia. Ejemplo:




✓ **while:** Repite una acción mientras una condición sea verdadera. ejemplo:




Un ejemplo cuando utilizamos condiciones dentro de un bucle.

```
>>> for numero in range(1, 11):
...  if numero % 2 == 0:
...   print(numero)
...
2
4
6
8
10
```

En este caso, el operador % obtiene el residuo de una división. Si la variable numero % 2 == 0, el número es par.

Para reforzar el uso de los bucles, te invito a que pruebes realizar estos ejercicios.

Imprime con for el mensaje: “Python is easy” cinco veces.

Utiliza while para imprimir los números del 1 al 10.

## 10. Trabajando con listas

- ¿Qué es una Lista?
  En Python una lista es una estructura de datos que permite almacenar varios elementos en una sola variable. Puedes guardar números, textos u otros tipos de datos. Estas se escriben entre corchetes [] separados por coma. Ejemplo:

```
>>> nombres = ["Ana", "Jose", "Ricardo"]
>>> print(nombres)
['Ana', 'Jose', 'Ricardo']
```

- Índices en una lista
  Los índices son las posiciones que permiten acceder a los elementos de una lista. Comienzan en cero (0). Ejemplo:

```
>>> nombres = ["Ana", "Jose", "Ricardo"]
>>> print(nombres[0])
Ana
>>> print(nombres[1])
Jose
>>> print(nombres[2])
Ricardo
```

  También se pueden utilizar con números negativos para iniciar desde el final. Ejemplo:

```
>>> nombres = ["Ana", "Jose", "Ricardo"]
>>> print(nombres[-1])
Ricardo
>>> print(nombres[-2])
Jose
>>> print(nombres[-3])
Ana
```

- Añadir elementos

  Para añadir elementos a una lista, podemos utilizar los siguientes métodos:

  - ✓ append(): Añade un elemento al final.
  - ✓ insert(): Añade un elemento en una posición específica.
  - ✓ extend(): Añade varios elementos de otra lista.

  Ejemplos:

```
>>> colores=["rojo", "negro"]
>>> colores.append("azul")
>>> print(colores)
['rojo', 'negro', 'azul']
```

```
>>> colores=["rojo", "negro", "azul"]
>>> colores.insert(1, "naranja")
>>> print(colores)
['rojo', 'naranja', 'negro', 'azul']
```

- Eliminar elementos

  Python ofrece diferentes métodos para eliminar elementos de una lista:

  - ✓ remove(): Elimina un elemento por su valor.
  - ✓ pop(): Elimina un elemento por su índice y lo devuelve.
  - ✓ del: Elimina un elemento por su índice.

  Ejemplos:

```
>>> edades = [20, 15, 10]
>>> edades.remove(15)
>>> print(edades)
[20, 10]
```

```
>>> edades = [20, 15, 10]
>>> edades.pop(0)
20
>>> print(edades)
[15, 10]
```

```
>>> edades = [20, 15, 10]
>>> del edades[1]
>>> print(edades)
[20, 10]
```

- Recorrer lista

  Esto significa leer o utilizar cada uno de sus elementos, normalmente mediante un ciclo for.

  Ejemplos:

```
>>> frutas = ["fresas", "manzanas", "uvas"]
>>> for fruta in frutas:
...  print(fruta)
...
fresas
manzanas
uvas
```

```
>>> frutas = ["fresas", "manzanas", "uvas"]
>>> for i in range(len(frutas)):
...  print(frutas[i])
...
fresas
manzanas
uvas
```

  En este ejemplo, utilizamos **range** para generar posiciones y **len** que devuelve la cantidad de elementos. En la impresión frutas[i] permitirá acceder a cada elemento de nuestra lista.

 Ejercicio práctico: crea una lista de números del 1 al 10 y muestra solamente los números pares.

## 11. Crear nuestras propias funciones

- ¿Qué es una función propia?

  Una función propia es un bloque de código que creamos para realizar una tarea específica. Podemos ejecutarla cada vez que la necesitemos, sin tener que escribir el mismo código varias veces. En Python, utilizamos la palabra reservada **def** para crear una función.

- Sintaxis de una función

```
>>> def nombre_funcion():
...     instrucciones
```

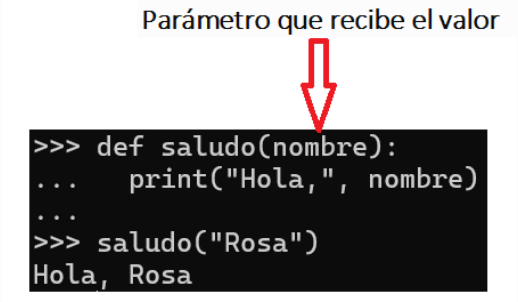
- ✓ **def:** Indica que vamos a definir una función.
- ✓ **nombre_funcion:** Es el nombre de la función que le damos.
- ✓ **():** Contiene los parámetros, si los hay.
- ✓ **::** Indica el inicio del bloque de instrucciones.
- ✓ **Espacios al inicio, sangría o indentación:** Indica qué instrucciones pertenecen a la función.

Ejemplo:

```
>>> def saludo():
...     print("Bienvenidos a la guia de Python para principiantes")
...
>>> saludo()
Bienvenidos a la guia de Python para principiantes
```

- **Funciones con parámetros**

Los parámetros permiten enviar datos a una función para que trabaje con ellos. Ejemplo:




- **Funciones que devuelven resultados**

Para devolver un resultado en este caso utilizamos la palabra **return**. Ejemplo:

```
>>> def suma(a, b):
...   return a + b
...
>>> resultados = suma(4, 2)
>>> print(resultados)
6
```

- **Crear una función para trabajar con listas**

En este caso podemos crear funciones propias para realizar operaciones con listas. Ejemplo:

```
>>> def mostrar_colores(lista):
...     for elemento in lista:
...         print(elemento)
...
>>> colores = ["Rojo", "Rosado", "Verde"]
>>> mostrar_colores(colores)
Rojo
Rosado
Verde
```

 Ejercicio práctico: crea una función llamada calcular_promedio que reciba tres notas, calcule el promedio y lo muestre en pantalla.

## 12. Manejo de errores

El manejo de errores permite controlar los problemas que pueden ocurrir cuando se ejecuta un programa, evitando que este se detenga inesperadamente.

En Python utilizamos principalmente las instrucciones **try** y **except** para detectar y manejar errores.

- **try:** Contiene el código que podría generar un error.
- **except:** Contiene las instrucciones que se ejecutan si ocurre un error.

Ejemplos:

```
>>> try:
...     numero = int(input("Introduce un numero: "))
...     print("El numero es:", numero)
... except:
...     print("Error: debes introducir un numero valido.")
...
Introduce un numero: 24
El numero es: 24
```

```
>>> try:
...     numero = int(input("Introduce un numero: "))
...     print("El numero es:", numero)
... except:
...     print("Error: debes introducir un numero valido.")
...
Introduce un numero: hola
Error: debes introducir un numero valido.
```

## 13. Desafío

Aplicando lo aprendido, ahora te propongo a que realices el siguiente ejercicio para reforzar lo aprendido.



### Juego de adivinar el número

El programa debe realizar lo siguiente:

- Elige un número.
- Pide al usuario que intente adivinarlo.
- Indica si debe elegir un número mayor o menor.
- Repite hasta acertar.
- Cuenta los intentos.

## 14. ¿Qué aprender después?

A continuación, te listo las siguientes rutas que puede seguir para ampliar tu conocimiento en este mundo:

- ✓ Programación Orientada a Objetos.
- ✓ Módulos y Paquetes.
- ✓ Archivos.
- ✓ APIs.
- ✓ Automatización.
- ✓ Desarrollo Web.
- ✓ Análisis de Datos.
- ✓ Inteligencia Artificial.