

¿Que es Python?

Python es un lenguaje de programación interpretado, multiparadigma y de alto nivel. Es muy popular en el ámbito de la programación debido a su sintaxis sencilla y su facilidad de uso. En esta documentación, se cubrirán algunos de los conceptos básicos de Python y se proporcionarán ejemplos para ilustrar cada uno de ellos.

Instalación

- Descarga la última versión de Python desde la página oficial: [Descargar Aquí](#)
- Abre el archivo descargado y sigue las instrucciones del instalador.
- Asegúrate de marcar la opción "Agregar Python al PATH" durante la instalación para que puedas acceder a Python desde la línea de comandos.
- ¡Listo! Ya puedes abrir una terminal y escribir python para empezar a usar Python.

Tipos de datos

Los tipos de datos son fundamentales en cualquier lenguaje de programación, ya que son la base para la manipulación de la información. Por lo tanto, es importante aprender los tipos de datos disponibles en Python y cómo utilizarlos adecuadamente.

Python es un lenguaje de programación de tipado dinámico. Esto significa que no es necesario declarar el tipo de una variable antes de utilizarla. El tipo de una variable se determina automáticamente en tiempo de ejecución según el valor que se le asigne.

Los tipos de datos disponibles son:

- **Enteros (int):** Representan números enteros positivos o negativos, sin decimales.

```
a = 10  
b = -5
```

- **Flotantes (float):** Representan números decimales.

```
c = 3.14  
d = -0.5
```

- Cadenas de texto (str): Representan texto o caracteres y pueden definirse utilizando comillas simples o dobles.

```
texto1 = 'Hola'
texto2 = "mundo"
```

- Booleanos (bool): Representan valores lógicos True (verdadero) o False (falso).

```
verdadero = True
falso = False
```

- Listas (list): Representan colecciones ordenadas de elementos. Pueden contener cualquier tipo de datos, incluso otras listas.

```
lista1 = [1, 2, 3, 4, 5]
lista2 = ['a', 'b', 'c']
```

- Tuplas (tuple): Son similares a las listas, pero no se pueden modificar una vez que se crean.

```
tupla1 = (1, 2, 3)
tupla2 = ('x', 'y', 'z')
```

- Diccionarios (dict): Representan una colección de pares clave-valor. Cada elemento del diccionario se compone de una clave y su correspondiente valor.

```
diccionario = {'nombre': 'Jairo', 'edad': 35, 'ciudad': 'Madrid'}
```

Operadores

Los operadores son símbolos que se utilizan para realizar diversas operaciones. En Python, los operadores se dividen en varios tipos:

- **Operadores aritméticos:** se utilizan para realizar operaciones matemáticas como suma, resta, multiplicación, división, módulo y potencia.

```
# Suma (+): suma dos valores.
a = 5
b = 3
c = a + b
print(c) # salida: 8

# Resta (-): resta dos valores.
a = 5
b = 3
c = a - b
print(c) # salida: 2

# Multiplicación (*): multiplica dos valores.
a = 5
b = 3
c = a * b
print(c) # salida: 15
```

```
# División (/): divide el valor de la izquierda por el valor de la derecha.
a = 10
b = 3
c = a / b
print(c) # salida: 3.3333333333333335

# Módulo (%): devuelve el resto de la división de la izquierda por la derecha.
a = 10
b = 3
c = a % b
print(c) # salida: 1

# Potencia (**): eleva el valor de la izquierda a la potencia del valor de la derecha.
a = 2
b = 3
c = a ** b
print(c) # salida: 8
```

- **Operadores de comparación:** se utilizan para comparar dos valores y devolver un valor booleano (verdadero o falso). Los operadores de comparación son == (igual a), != (no igual a), < (menor que), > (mayor que), <= (menor o igual a) y >= (mayor o igual a).

```
# Variables a comparar
x = 5
y = 3
```

```
# >: mayor que
print(x > y) # True, ya que 5 es mayor que 3

# <: menor que
print(x < y) # False, ya que 5 no es menor que 3

# >=: mayor o igual que
print(x >= y) # True, ya que 5 es mayor o igual que 3
```

```
# <=: menor o igual que
print(x <= y) # False, ya que 5 no es menor o igual que 3

# ==: igual que
print(x == y) # False, ya que 5 no es igual que 3

# !=: distinto de
print(x != y) # True, ya que 5 es distinto de 3
```

- **Operadores lógicos:** se utilizan para combinar expresiones booleanas y devolver un valor booleano. Estos son: and (y), or (o) y not (no).

```
# and: Retorna True si ambos operandos son verdaderos.
a = 5
b = 10
c = 15
print(a < b and b < c) # True
print(a < b and c < b) # False
```

```
# not: Retorna True si el operando es falso.
op1 = True
op2 = False
op3 = 5 < 6
op4 = 5 > 6
print(not op1) # False
print(not op2) # True
print(not op3) # False
print(not op4) # True
```

```
# or: Retorna True si al menos uno de los operandos es verdadero.
d = 5
e = 10
f = 15
print(d < e or e < f) # True
print(d < 2 or e < d) # False
```

- **Operadores de asignación:** se utilizan para asignar valores a variables. Los operadores de asignación son: = (asignación simple), += (suma y asignación), -= (resta y asignación), *= (multiplicación y asignación), /= (división y asignación) y %= (módulo y asignación), entre otros.

```
# Operador de Asignacion =
x = 5

# Operador de asignacion y suma
x = 10
x += 5 # Ahora x vale 15

# Operador de asignacion y resta
x = 10
x -= 5 # Ahora x vale 5
```

```
# Operador de asignacion y
x = 5
x *= 2 # Ahora x vale 10

# Operador de asignacion y division
x = 10
x /= 2 # Ahora x vale 5

# Operador de asignacion y potenciacion
x = 2
x **= 3 # Ahora x vale 8 (2 elevado a la 3: 2 x 2 x 2 = 8)
```

Input y Print

`input()` y `print()` son dos funciones incorporadas en Python que se utilizan comúnmente para interactuar con el usuario y mostrar información en la consola.

- La función `input()` se utiliza para recibir datos ingresados por el usuario a través del teclado.
- la función `print()` se utiliza para mostrar información en la consola.

```
nombre = print("Ingresa su nombre: ") # Se ingresa Diego
print(f"¡Hola {nombre}!") # Salida: ¡Hola Diego!
```

Condicionales

Las estructuras condicionales permiten que el programa tome diferentes caminos de ejecución dependiendo de si una condición se cumple o no.

Las estructuras condicionales en Python son `if`, `if-else`, `if-elif-else` y Operador Ternario.

```
nacimiento = int(input("Ingresa tu fecha de nacimiento: "))

# Ejemplo 1
if 2023 - nacimiento > 25:
    print("Eres mayor de 25 años")
elif 2023 - nacimiento == 25:
    print("Tienes 25 años de edad")
else:
    print("Eres menor de 25 años")
```

```
# Operador Ternario
print("Eres de 25 años o mayor") if 2023 - nacimiento >= 25 else print("Eres menor de 25")
```

Loops

Los loops, también conocidos como bucles, son una estructura fundamental en la programación que permiten repetir un bloque de código varias veces hasta que se cumpla una condición.

En Python, existen dos tipos de loops: el bucle **while** y el bucle **for**.

```
i = 1
while i <= 10:
    print(i)
    i += 1
```

Ciclo While

```
for i in range(1,11):
    print(f"Numero: {i}")
```

Ciclo For

Funciones

Las funciones son bloques de código reutilizable que realizan una tarea específica. Se definen utilizando la palabra clave "def" seguida del nombre de la función, los parámetros de entrada (si los hay) y dos puntos.

El cuerpo de la función se encuentra indentado (tabulado) después de la definición. Las funciones pueden tomar cero o más parámetros y pueden o no devolver un valor.

```
# Ejemplo: Suma de 2 numeros
```

```
def suma(a, b):
    resultado = a + b
    print(f"Total: {resultado}")
```

```
# Para llamar a la función, simplemente
# escribimos su nombre y le pasamos los parametros
suma(10,20) # Total: 30
```

Clases

En la programación orientada a objetos, una clase es un conjunto de atributos y métodos que definen un objeto en particular. Los atributos son las características o propiedades del objeto, mientras que los métodos son las funciones o acciones que puede realizar el objeto. En Python, se definen las clases utilizando la palabra clave "class".

Dentro de la clase, se pueden definir los atributos y los métodos, y luego se puede crear uno o varios objetos de esa clase, los cuales tendrán los mismos atributos y métodos definidos en la clase. En pocas palabras, las clases pueden considerarse como plantillas o moldes a partir de los cuales se crean objetos.

Cada objeto creado a partir de una clase tendrá las mismas propiedades y métodos definidos en la clase(molde), pero también tendrá sus propios valores para esas propiedades.

```
class Persona:
    # self es una referencia al objeto actual que se está creando a partir de la clase.
    # Al utilizar self, se puede acceder a los atributos y métodos de la instancia de
    # la clase en la que se está trabajando. En pocas palabras permite acceder a los
    # valores entre los diferentes métodos de la clase.
    def __init__(self, nombre, edad):
        # __init__ es un método especial en Python que se usa para inicializar los atributos de una clase.
        # Es decir, es el constructor de la clase.
        self.nombre = nombre
        self.edad = edad

    def saludar(self):
        print(f"Hola, mi nombre es {self.nombre} y tengo {self.edad} años.")
```

```
# Creando una instancia
Usuario = Persona("Jean", 30)
print(Usuario.nombre) # Salida: Jean
print(Usuario.edad) # Salida: 30
Usuario.saludar() # Salida: Hola, mi nombre es Jean y tengo 30 años.
```

Try-Except

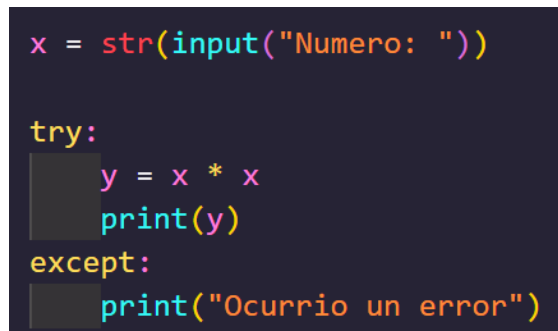
try-except es una estructura de control utilizada para manejar excepciones y errores en el código.

El bloque try permite probar un bloque de código en busca de errores, mientras que el bloque except permite manejar cualquier excepción que se produzca en el bloque try.

También se puede agregar un bloque else opcional que se ejecutará si no se produce ninguna excepción. Y, un bloque finally, también opcional, que se ejecutará siempre, independientemente de si se produce una excepción o no.

la sintaxis básica de try-except es:

```
try:  
    # Código a probar  
except Excepcion:  
    # Código a ejecutar en caso de que se produzca la excepción  
else:  
    # Código a ejecutar si no se produce ninguna excepción  
finally:  
    # Código a ejecutar siempre
```



```
x = str(input("Numero: "))  
  
try:  
    y = x * x  
    print(y)  
except:  
    print("Ocurrio un error")
```

En este ejemplo se está forzando una excepción TypeError

Existen varios tipos de excepciones que pueden ser lanzadas durante la ejecución de un programa:

- **TypeError:** se produce cuando se realiza una operación o función en un tipo de objeto que no es compatible con ella.
- **ValueError:** se produce cuando se llama a una función o método con un argumento que tiene un valor inesperado.
- **IndexError:** se produce cuando se intenta acceder a un elemento fuera del rango de índices de una secuencia (lista, tupla, etc.).

- **KeyError**: se produce cuando se intenta acceder a una clave que no está presente en un diccionario.
- **NameError**: se produce cuando se intenta utilizar una variable o nombre que no ha sido definido.
- **FileNotFoundError**: se produce cuando se intenta acceder a un archivo que no existe o no se puede abrir.

Es opcional definir la excepción que se espera manejar en una estructura try-except. Si no se especifica una excepción en la cláusula except, se capturará cualquier excepción que ocurra en el bloque try. Sin embargo, es buena práctica especificar la excepción para que el manejo de excepciones sea más preciso y para evitar capturar excepciones que no se espera manejar.

```
# Para obtener el tipo de error se puede utilizar la sentencia as seguida  
# de un nombre de variable en el bloque except  
try:  
    y = x * x  
    print(y)  
except TypeError as err:  
    print(f"Ocurrio el error: {err}")
```

En este ejemplo se está forzando una excepción TypeError, adicionalmente se captura y se muestra