

Introducción a Python

Python es un lenguaje de programación de alto nivel, interpretado y multipropósito. Es conocido por su sintaxis clara y legible, lo que lo hace ideal tanto para principiantes como para programadores experimentados.

Programa Básico de Python En Python, un programa básico puede consistir en imprimir un mensaje en la consola o realizar operaciones matemáticas simples.

```
print("¡Hola, mundo!");
```

```
print("Mi nombre es Fernando. Vamos los millo")
```

```
print("¡Aguante Luján Joven!")
```

Salida Fija o Salida Variable

La salida en Python puede ser fija, como en el ejemplo anterior, o variable, dependiendo de los datos que se manipulen en el programa.

Salida Fija:

```
# Salida fija
print("¡Esta es una salida fija y viva Messi!")
```

En este ejemplo, la salida siempre será **"¡Esta es una salida fija y viva Messi!"** cuando se ejecute el programa.

Salida Variable:

```
nombre = "Lionel"
print(nombre)
```

En este caso nosotros declaramos la variable "nombre" e inmediatamente le asignamos un valor(str). Pero también podemos hacerlo más interactivo e ingresar el valor de la variable a través del metodo input().

```
# Salida variable
nombre = input("Por favor, introduce tu nombre: ")
```

```
print("¡Hola,", nombre, "!")
```

En este caso, la salida depende de la entrada del usuario. El programa solicita al usuario que introduzca su nombre y luego imprime un saludo personalizado con el nombre ingresado.

La salida puede variar dependiendo del nombre que el usuario introduzca en la consola. Por ejemplo, si el usuario introduce "Juan", la salida será "¡Hola, Juan!", y si introduce "María", la salida será "¡Hola, María!", y así sucesivamente. En este caso en la función `print()` concatenamos o unimos una frase que va entre comillas "" y luego de una coma ingresamos el nombre de la variable que valga la redundancia se llama "nombre".

Sentencia `input()`

La función `input()` se utiliza para recibir entrada del usuario a través de la consola. Retorna una cadena (string) que puede ser asignada a una variable.

```
nombre = input("Por favor, introduce tu nombre: ")  
print("¡Hola,", nombre, "!")
```

La función `input()` la explicamos en el punto anterior cuando ingresábamos las salidas variables.

Tipos de Variables Básicas

En Python, las variables pueden contener diferentes tipos de datos, como enteros, flotantes, cadenas, listas, diccionarios, entre otros.

```
entero = 10  
flotante = 3.14  
cadena = "Hola, mundo!"  
lista = [1, 2, 3, 4, 5]  
diccionario = {"clave": "valor"}
```

La función `type()` en Python es una función incorporada que devuelve el tipo de un objeto dado. Puede aceptar un argumento y devuelve el tipo de ese argumento. Aquí hay una descripción más detallada de la función `type()`:

Sintaxis: `type(objeto)`

Argumento: objeto: Es el objeto del cual deseas conocer el tipo.

Valor de retorno: Devuelve el tipo de dato del objeto dado como argumento.

Uso común: La función `type()` es comúnmente utilizada en Python para la depuración de código, para verificar el tipo de objeto antes de realizar operaciones específicas en él. También se utiliza en situaciones donde necesitas determinar el tipo de dato de un objeto dinámicamente durante la ejecución del programa.

Ejemplo de uso:

```
x = 10
y = "Hola, mundo!"
z = [1, 2, 3]

print(type(x)) # Salida: <class 'int'>
print(type(y)) # Salida: <class 'str'>
print(type(z)) # Salida: <class 'list'>
```

o como en el caso anterior:

```
entero = 10
flotante = 3.14
cadena = "Hola, mundo!"
lista = [1, 2, 3, 4, 5]
diccionario = {"clave": "valor"}

print(type(entero))      # Salida: <class 'int'>
print(type(flotante))    # Salida: <class 'float'>
print(type(cadena))      # Salida: <class 'str'>
print(type(lista))       # Salida: <class 'list'>
print(type(diccionario)) # Salida: <class 'dict'>
```

Esto imprimirá los tipos de datos de las variables entero, flotante, cadena, lista y diccionario. La función `type()` devuelve el tipo de dato de una variable en Python.

Estructuras de Control

Python ofrece varias estructuras de control para controlar el flujo de ejecución del programa:

if, elif, else: Permite tomar decisiones basadas en condiciones. **while:** Se utiliza para ejecutar un bloque de código mientras se cumpla una condición. **for:** Se utiliza para iterar sobre una secuencia (como listas, tuplas o cadenas). **microframework Flask:** Flask es un microframework para construir aplicaciones web en Python.

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

numero = 1
while numero <= 10:
    print(numero)
    numero += 1

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

IF

La sentencia if permite evaluar una condición y realizar acciones si la condición es verdadera. La sintaxis básica es: IF - ELIF - ELSE

Por ejemplo:

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

¿Qué crees que se mostrará por consola?

Esta parte del código utiliza una estructura de control if-else para tomar decisiones basadas en una condición. Si la variable edad es mayor o igual a 18, se imprimirá "Eres mayor de edad.". De lo contrario, se imprimirá "Eres menor de edad.".

WHILE (Ciclo o Bucle)

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

Aquí, se utiliza un bucle while para imprimir los números del 1 al 10. El bucle continuará ejecutándose mientras el valor de la variable numero sea menor o igual a 10. En cada iteración del bucle, se imprime el valor de numero y luego se incrementa en 1.

FOR (Ciclo o Bucle)

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

En esta sección, se utiliza un bucle for para iterar sobre una secuencia de números generados por la función `range()`. La función `range(1, 6)` genera una secuencia de números desde 1 hasta 5 (no incluyendo el límite superior, es decir, hasta 6 - 1). En cada iteración del bucle, el valor de `i` será uno de los números en la secuencia generada por `range()`, y se imprimirá ese valor. En este caso, imprimirá los números del 1 al 5.

Operadores Aritméticos:

Suma (+): Se utiliza para sumar dos valores.


Resta (-): Se utiliza para restar el segundo valor del primero.

Multiplicación (*): Se utiliza para multiplicar dos valores.

División (/): Se utiliza para dividir el primer valor por el segundo. Devuelve un resultado de punto flotante.

División entera (//): Se utiliza para dividir el primer valor por el segundo y obtener el resultado entero.

Módulo (%): Devuelve el resto de la división del primer valor por el segundo.

Exponenciación () ** * Calcula el primer valor elevado a la potencia del segundo.

Ejemplo de operadores aritméticos:

```
a = 10
b = 3

suma = a + b          # 13
resta = a - b          # 7
multiplicacion = a * b # 30
division = a / b       # 3.33333...
division_entera = a // b # 3
modulo = a % b         # 1
exponenciacion = a ** b # 1000
```

Operador Módulo (%):

El operador módulo % devuelve el resto de la división del primer número por el segundo. En otras palabras, si divides el primer número por el segundo, el operador módulo devuelve el residuo de esa división.

Ejemplo:

```
a = 10
b = 3

modulo = a % b # 1

# Explicación: Cuando divides 10 por 3, el resultado es 3 con un residuo de 1.
# Por lo tanto, el valor de 'modulo' es 1. IMPAR
```

El operador **módulo %** se puede utilizar para determinar si un número es par o impar. Si un número es divisible por 2 sin dejar residuo, entonces es par; de lo contrario, es impar.

Aquí tienes un ejemplo de cómo puedes usar el operador módulo para buscar números pares:

```
# Definimos una lista de números
numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Iteramos sobre cada número en la lista
for num in numeros:
```

```
# Comprobamos si el número es par utilizando el operador módulo
if num % 2 == 0:
    print(num, "es un número par")
else:
    print(num, "es un número impar")
```

Este código recorre la lista de números y utiliza el operador módulo para determinar si cada número es par o impar. Si el resultado de la operación `num % 2` es igual a cero, entonces el número es par y se imprime un mensaje correspondiente. De lo contrario, se imprime que es impar.

Operador Exponenciación (**):

El operador exponenciación `**` se utiliza para elevar un número a una potencia determinada.

Ejemplo:

```
a = 10
b = 3

exponenciacion = a ** b    # 1000

# Explicación: 10 elevado a la potencia de 3 es igual a 1000.
```

Estos operadores son muy útiles en matemáticas y en la programación para realizar cálculos y operaciones específicas.

Operadores Lógicos:

AND (y): Devuelve True si ambos valores son True.

OR (o): Devuelve True si al menos uno de los valores es True.

NOT (no): Devuelve True si el valor es False y viceversa.

Ejemplo de operadores lógicos:

```
a = True
b = False

resultado_and = a and b    # False
resultado_or = a or b      # True
resultado_not = not a      # False
```

Estos son algunos de los operadores más comunes en Python. Puedes utilizarlos para realizar diversas operaciones y comparaciones en tus programas.

Y recuerda ... Lionel Messi y Dibu Martinez están orgullosos de vos.