

Sentencias básicas del lenguaje Python

M2: FUNDAMENTOS DE PROGRAMACIÓN
PYTHON PARA INGENIEROS DE DATOS

|AE1: CODIFICAR UNA RUTINA UTILIZANDO SENTENCIAS BÁSICAS
PARA RESOLVER UN PROBLEMA DE MEDIANA COMPLEJIDAD ACORDE
AL LENGUAJE PYTHON.

Introducción

Python es un lenguaje de programación de alto nivel, creado en 1991 por Guido van Rossum, cuyo objetivo principal era proporcionar una herramienta de programación que fuera fácil de aprender, leer y escribir. Desde su creación, Python ha crecido de manera exponencial en popularidad gracias a su versatilidad, que le permite ser utilizado en una amplia gama de aplicaciones, desde el desarrollo web hasta la ciencia de datos y la inteligencia artificial. Python se destaca por su filosofía de "baterías incluidas", lo que significa que viene con una biblioteca estándar extensa que permite a los desarrolladores realizar tareas comunes sin necesidad de recurrir a paquetes externos.

Esta lección está diseñada para aquellos que desean aprender los fundamentos de Python y cómo aplicarlo en situaciones prácticas. A lo largo del contenido, exploraremos desde los conceptos más básicos como variables y tipos de datos, hasta la ejecución de scripts y la creación de programas sencillos. Además, cada sección incluirá ejemplos prácticos, tablas comparativas y ejercicios para poner en práctica lo aprendido. Python es una herramienta poderosa que, gracias a su simplicidad y facilidad de uso, es accesible para principiantes y suficientemente robusta para desarrolladores experimentados que buscan soluciones eficientes y escalables.

Aprendizajes esperados

Cuando finalices la lección serás capaz de:

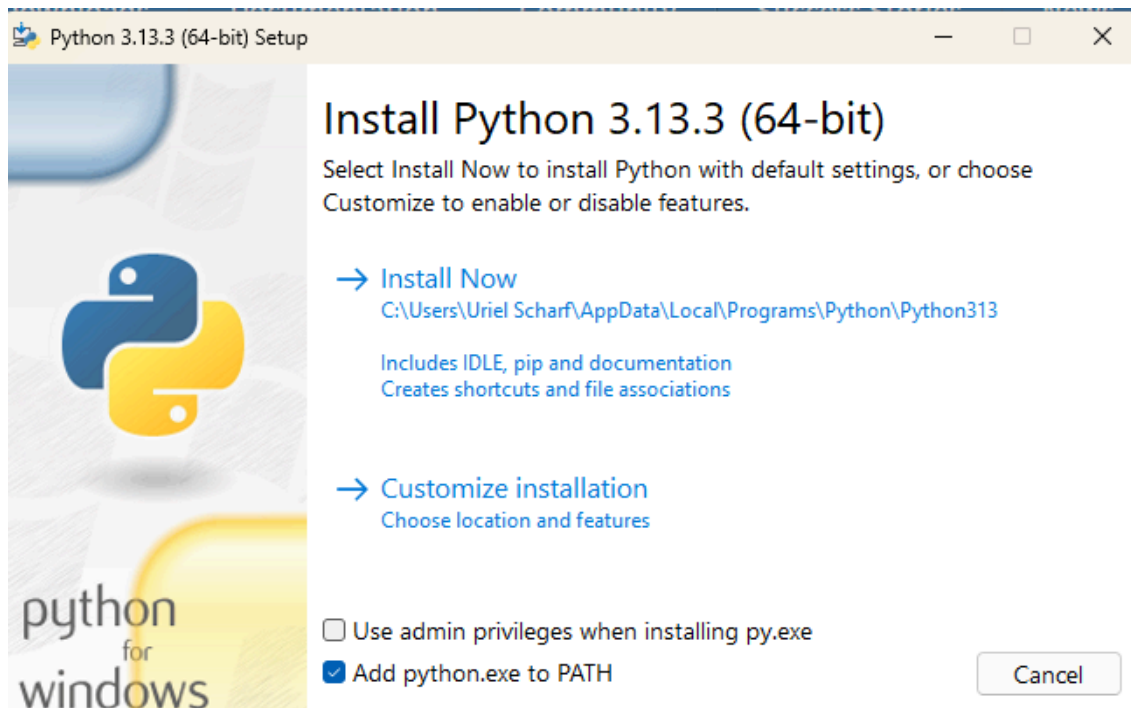
- Comprender y utilizar los tipos de datos fundamentales en Python, incluyendo enteros, decimales, cadenas de caracteres, booleanos y números complejos.
- Realizar conversiones de tipo de datos en Python y entender cómo estas conversiones afectan las operaciones entre variables.
- Aplicar operadores y expresiones aritméticas y lógicas en Python para realizar cálculos y evaluaciones condicionales.
- Utilizar sentencias condicionales (`if`, `elif`, `else`) para crear flujos de control en tus programas.
- Crear y ejecutar scripts de Python en la consola y entender el flujo de ejecución de un programa.

- Realizar operaciones de entrada y salida de datos a través de la consola, interactuando con el usuario mediante `input()` y `print()`.
- Desarrollar scripts básicos que utilicen las funcionalidades de Python para resolver problemas de forma sencilla y efectiva.
- Comprender la estructura y organización de un programa en Python, preparándote para abordar problemas más complejos.

Instalación y Primer Script en Python con VS Code

Paso 1: Instalar Python

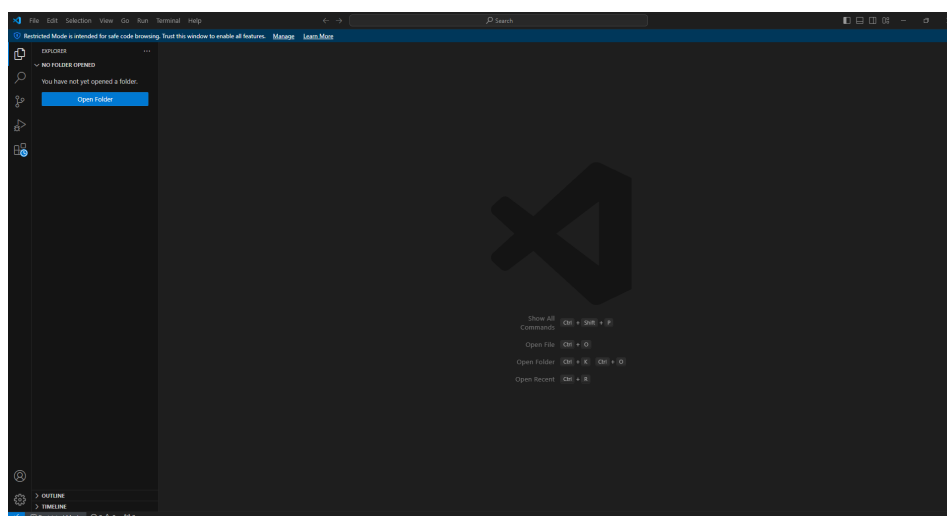
1. Ingresá a: <https://www.python.org/downloads/>
2. Hacé clic en el botón amarillo "**Download Python**" correspondiente a tu sistema operativo (Windows/Mac/Linux).
3. Ejecutá el instalador descargado.
 - **IMPORTANTE:** Asegurate de **tildar** la opción "**Add Python to PATH**" antes de hacer clic en "Install Now".
 - **Captura sugerida:** Pantalla del instalador con el checkbox "Add to PATH" marcado.



Paso 2: Instalar Visual Studio Code


1. Ingresá a: <https://code.visualstudio.com/>
2. Hací clic en **Download for Windows / macOS / Linux** (según tu sistema).
3. Ejecutá el instalador y seguí los pasos por defecto.
4. Abrí Visual Studio Code una vez finalizada la instalación.

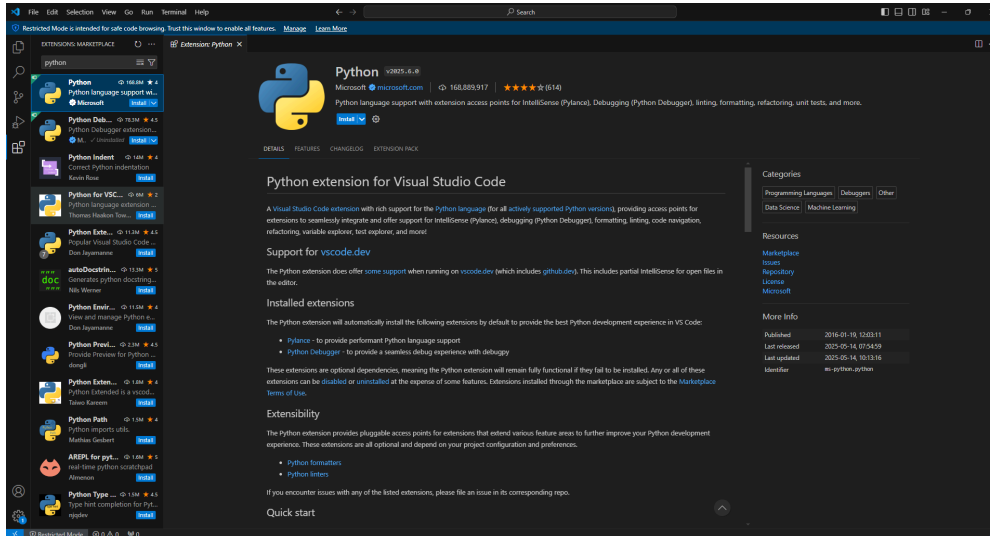
 Captura sugerida: Pantalla de inicio de VS Code.



Paso 3: Instalar la extensión de Python en VS Code

1. Dentro de VS Code, hacé clic en el icono de **extensiones** (🔌) en la barra lateral izquierda.
2. Buscá “Python” (editor oficial: Microsoft).
3. Hacé clic en **Install**.

 Captura sugerida: Resultado de búsqueda con la extensión oficial seleccionada.

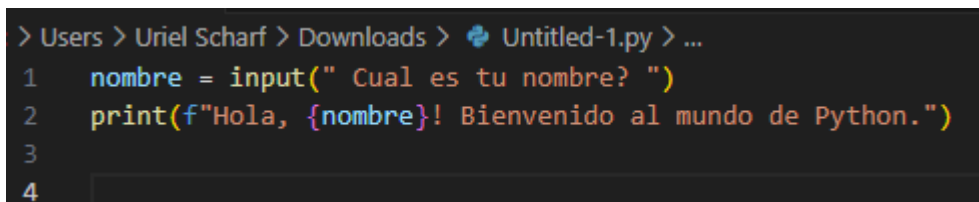


Paso 4: Crear tu primer archivo en Python


1. En el menú de VS Code, andá a **File > New File** y guardalo como **hola.py**.
2. Escribí este código:

```
nombre = input("¿Cuál es tu nombre? ")
print(f"Hola, {nombre}! Bienvenido al mundo de Python.")
```

 Captura sugerida: Editor con el archivo **hola.py** abierto y el código escrito.




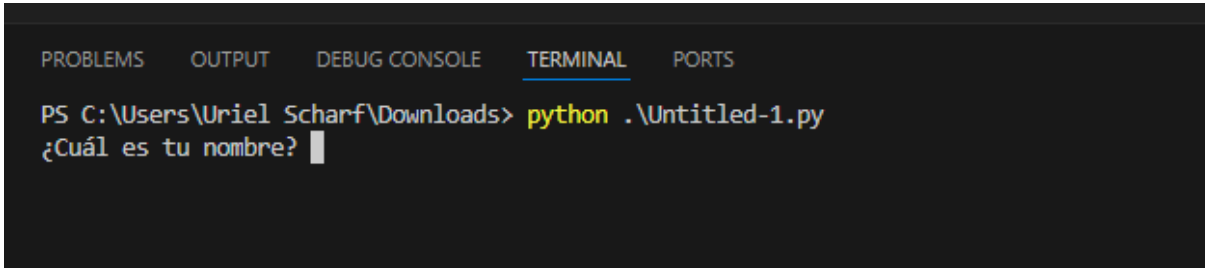
Paso 5: Ejecutar el script

1. Hacé clic en el botón de  arriba a la derecha o usá el menú contextual

con clic derecho > **"Run Python File in Terminal"**.

2. Ingresá tu nombre cuando te lo pida el terminal y verificá que aparezca el mensaje personalizado.

 *Captura sugerida:* Terminal mostrando el mensaje de entrada y salida correctamente.



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Uriel Scharf\Downloads> python .\Untitled-1.py
¿Cuál es tu nombre? █
```

Confirmación del entorno listo

- Si lograste ver tu mensaje en pantalla, ¡ya estás listo para avanzar al siguiente módulo!
- Si algo no funciona, revisá si:
 - Python está bien instalado.
 - La extensión de Python está activa.
 - El archivo se guardó con extensión **.py**.

Variables y Tipos de Dato Fundamentales en Python

Introducción al Concepto de Variables

En Python, una variable es un espacio en la memoria donde se almacena información que puede ser utilizada y manipulada a lo largo del código. A diferencia de otros lenguajes de programación, Python no requiere declarar el tipo de dato de una variable explícitamente; el lenguaje infiere el tipo según el valor asignado. Esto hace que Python sea un lenguaje de tipado dinámico, lo que facilita la creación y manipulación de variables de manera intuitiva.

Las variables en Python se crean simplemente asignando un valor a un nombre, utilizando el operador de asignación `=`. Este nombre representa el valor

almacenado y puede utilizarse en cálculos, impresiones, o como parte de estructuras de control. El nombre de una variable debe cumplir ciertas reglas: debe comenzar con una letra o guion bajo (_), y no puede contener caracteres especiales ni espacios. Es importante elegir nombres descriptivos para las variables, lo que facilita la legibilidad del código.

Tipos de Datos Fundamentales

Python proporciona varios tipos de datos fundamentales que son ampliamente utilizados para representar diferentes tipos de información. A continuación, describimos los tipos de datos más comunes, con ejemplos prácticos para cada uno.

1. Enteros (**int**)

- Los enteros en Python son números sin decimales, y pueden ser positivos, negativos o cero. Python permite trabajar con enteros de gran tamaño sin perder precisión, lo que es especialmente útil para cálculos matemáticos complejos.

Ejemplo:

```
edad = 25
poblacion = 1000000
print("Edad:", edad)
print("Población:", poblacion)
```

2. Decimales (**float**)

- Los números de punto flotante o decimales representan números con una parte decimal. Son útiles para cálculos que requieren precisión, como mediciones científicas o valores financieros.

Ejemplo:

```
precio = 19.99
altura = 1.75
print("Precio:", precio)
print("Altura:", altura)
```

Cadenas de caracteres (**str**)

- Las cadenas de caracteres son secuencias de texto y se representan en Python entre comillas simples (') o dobles ("). Las cadenas pueden contener letras, números y caracteres especiales, y se utilizan ampliamente para almacenar y manipular información textual.

Ejemplo:

```
nombre = "Juan"
saludo = "Hola, ¿cómo estás?"
print("Nombre:", nombre)
print("Saludo:", saludo)
```

3. Booleanos (**bool**)

- Los booleanos representan valores de verdad, es decir, **True** o **False**. Este tipo de dato es esencial en la programación para realizar decisiones y condiciones en el código.

Ejemplo:

```
es_mayor = True
es_menor = False
print("Es mayor:", es_mayor)
print("Es menor:", es_menor)
```

Números complejos (**complex**)

- Python también permite trabajar con números complejos, que tienen una parte real y una imaginaria. Este tipo de dato es útil en cálculos científicos y matemáticos avanzados.

Ejemplo:

```
numero_complejo = 3 + 4j
print("Número complejo:", numero_complejo)
print("Parte real:", numero_complejo.real)
print("Parte imaginaria:", numero_complejo.imag)
```

Tabla Comparativa de Tipos de Datos Fundamentales

La siguiente tabla resume los tipos de datos fundamentales en Python, sus características y ejemplos:

Tipo de Dato	Descripción	Ejemplo	Uso Común
<code>int</code>	Número entero sin parte decimal	<code>42, -10, 0</code>	Contadores, índices, ID's
<code>float</code>	Número con punto decimal	<code>3.14, -2.5</code>	Cálculos científicos, precios
<code>str</code>	Cadena de caracteres o texto	<code>"Python", "A"</code>	Nombres, descripciones, mensajes
<code>bool</code>	Valor booleano (<code>True</code> o <code>False</code>)	<code>True, False</code>	Condiciones y decisiones
<code>complex</code>	Número con parte real e imaginaria	<code>3 + 4j</code>	Cálculos avanzados, ciencia

Ejemplos Avanzados de Uso de Variables y Tipos de Datos

Cálculo de Área de un Círculo

Utilizando variables de tipo `float` para realizar un cálculo matemático:

```
radio = 5.0
area = 3.1416 * (radio ** 2)
print("El área del círculo es:", area)
```

Concatenación de Cadenas

Usar cadenas (`str`) para combinar texto y crear mensajes personalizados:

```
nombre = "Ana"
edad = 30
mensaje = "Hola, " + nombre + ". Tienes " + str(edad) + " años."
print(mensaje)
```

Uso de Booleanos en Condiciones

Usar booleanos para determinar si una persona es mayor de edad:

```
edad = 20
es_mayor = edad >= 18
if es_mayor:
    print("Es mayor de edad")
else:
    print("Es menor de edad")
```

Buenas Prácticas en el Uso de Variables

1. **Nombres descriptivos:** Utiliza nombres que describan claramente el propósito de la variable. En lugar de `x`, `y`, o `z`, es mejor usar `edad`, `altura`, o `nombre`.
2. **Uso de minúsculas con guiones bajos:** En Python, se recomienda escribir los nombres de las variables en minúsculas y con guiones bajos entre palabras para mayor legibilidad (`edad_usuario` en lugar de `EdadUsuario`).
3. **Evita palabras reservadas:** No uses palabras reservadas de Python (como `print`, `for`, `if`) como nombres de variables, ya que esto puede causar errores en el código.

Ejercicio Práctico

Escribe un programa en Python que reciba el nombre, la edad y la altura de una persona, y luego imprima un mensaje personalizado. Utiliza las conversiones de tipo si es necesario.

Ejemplo de Ejercicio:

```
nombre = input("Ingresa tu nombre: ")
edad = int(input("Ingresa tu edad: "))
altura = float(input("Ingresa tu altura en metros: "))

mensaje = f"Hola, {nombre}. Tienes {edad} años y mides {altura} metros."
print(mensaje)
```

Resumen del Tema

- Las **variables** permiten almacenar información y se crean mediante la asignación de un valor a un nombre.
- Python admite varios **tipos de datos fundamentales**: enteros (`int`), decimales (`float`), cadenas (`str`), booleanos (`bool`), y números complejos (`complex`).
- Cada tipo de dato tiene características y usos específicos, desde el almacenamiento de texto hasta el cálculo de valores matemáticos complejos.
- Utilizar **nombres descriptivos** para las variables y **evitar palabras reservadas** son buenas prácticas que mejoran la legibilidad y mantenimiento del código.

Conclusión

Entender las variables y tipos de datos fundamentales es esencial en Python, ya que permiten organizar y manipular la información de manera eficiente. Este conocimiento es la base para trabajar con datos en Python y desarrollar aplicaciones que respondan de forma dinámica a diferentes condiciones y entradas.

Conversiones de Tipo en Python

Introducción al Concepto de Conversión de Tipo

En Python, la conversión de tipo es el proceso de cambiar el tipo de dato de una variable a otro tipo. Este proceso es útil cuando se necesita realizar operaciones entre tipos de datos diferentes, como concatenar una cadena con un número o realizar cálculos entre enteros y decimales. Python permite realizar conversiones de tipo de dos maneras: **conversión implícita** y **conversión explícita**.

- **Conversión implícita**: Python realiza esta conversión automáticamente cuando es necesario y seguro. Por ejemplo, al sumar un número entero (`int`) y un número decimal (`float`), Python convierte automáticamente el entero en decimal para evitar pérdida de precisión.
- **Conversión explícita**: En algunos casos, Python requiere que el programador indique la conversión de tipo de forma explícita utilizando funciones de conversión, como `int()`, `float()`, `str()`, entre otras. Esta técnica

es conocida también como "casting" y es útil cuando el tipo de dato necesita ser cambiado para una operación específica.

Tipos de Conversiones y Funciones de Conversión

Python proporciona varias funciones para realizar conversiones de tipo. A continuación, se describen las funciones de conversión más comunes con ejemplos prácticos.

1. Conversión a Entero (**int()**)

- La función **int()** convierte un valor en un número entero. Si el valor es un número decimal, se trunca la parte decimal. También puede convertir cadenas numéricas a enteros.

Ejemplo:

```
decimal = 10.7
entero = int(decimal) # Resultado: 10
print("Entero:", entero)

cadena = "123"
numero = int(cadena) # Convierte cadena a entero
print("Número:", numero)
```

2. Conversión a Decimal (**float()**)

- La función **float()** convierte un valor en un número de punto flotante (decimal). Es útil cuando se necesita realizar cálculos con decimales para obtener resultados más precisos.

Ejemplo:

```
entero = 5
decimal = float(entero) # Convierte entero a decimal
print("Decimal:", decimal)

cadena = "3.14"
numero_decimal = float(cadena) # Convierte cadena a decimal
print("Número decimal:", numero_decimal)
```

3. Conversión a Cadena (**str()**)

- La función **str()** convierte un valor en una cadena de caracteres. Esta conversión es útil cuando se necesita mostrar un mensaje que combine texto y valores numéricos.

Ejemplo:

```
edad = 25
mensaje = "Tienes " + str(edad) + " años."
print(mensaje)
```

4. Conversión a Booleano (**bool()**)

- La función **bool()** convierte un valor en un valor booleano (**True** o **False**). Los valores numéricos **0** y **0.0**, y la cadena vacía **" "**, se convierten en **False**. Cualquier otro valor se convierte en **True**.

Ejemplo:

```
numero = 0
estado = bool(numero) # Resultado: False
print("Estado:", estado)

texto = "Python"
estado_texto = bool(texto) # Resultado: True
print("Estado de texto:", estado_texto)
```

5. Conversión a Complejo (**complex()**)

- La función **complex()** convierte un valor en un número complejo. Se puede especificar la parte real y la imaginaria, y Python creará el número complejo correspondiente.

Ejemplo:

```
real = 5
imaginario = 3
numero_complejo = complex(real, imaginario) # Resultado: (5+3j)
print("Número complejo:", numero_complejo)
```

Tabla Comparativa de Funciones de Conversión

La siguiente tabla muestra algunas de las funciones de conversión más comunes en Python y su uso.

Función	Descripción	Ejemplo	Resultado
<code>int()</code>	Convierte a entero	<code>int("5.9")</code>	5
<code>float()</code>	Convierte a decimal	<code>float("3")</code>	3.0

<code>str()</code>	Convierte a cadena	<code>str(100)</code>	"100"
<code>bool()</code>	Convierte a booleano	<code>bool(0)</code>	False
<code>complex()</code>	Convierte a número complejo	<code>complex(4, 5)</code>	(4+5j)

Ejemplos Prácticos de Conversión de Tipo

A continuación, se presentan algunos ejemplos que ilustran cómo utilizar las conversiones de tipo en Python para resolver problemas comunes.

1. Concatenación de Texto y Números

- A veces, se necesita combinar texto con números para crear un mensaje personalizado. En este caso, es necesario convertir los números a cadenas.

```
nombre = "Carlos"

edad = 30
mensaje = "Hola, " + nombre + ". Tienes " + str(edad) + " años."
print(mensaje)
```

2. Promedio de Calificaciones

- Supongamos que tenemos una lista de calificaciones como cadenas de texto y queremos calcular el promedio. Primero, necesitamos convertir las cadenas a números.

```
calificaciones = ["85", "90", "78", "92"]

suma = sum([int(nota) for nota in calificaciones])

promedio = suma / len(calificaciones)
print("Promedio:", promedio)
```

3. Validación de Entrada de Usuario

- Para recibir un número entero del usuario, usamos `input()` y luego convertimos la entrada a entero con `int()`. Esto asegura que el valor ingresado sea un número.

```
python
Copiar código
entrada = input("Ingresa un número: ")
numero = int(entrada)
print("Número ingresado:", numero)
```

Buenas Prácticas al Realizar Conversiones de Tipo

1. **Verificar los Datos Antes de la Conversión:** Antes de convertir una cadena a un número, asegúrate de que la cadena contiene un valor numérico válido. Esto evita errores de ejecución.
2. **Utilizar Conversiones Temporales:** En operaciones complejas, realiza conversiones temporales solo para el cálculo específico y luego convierte el resultado al tipo final necesario.

Manejo de Errores en Conversiones: Usa un bloque `try-except` para capturar posibles errores durante la conversión. Esto es útil cuando se trabaja con datos de entrada del usuario.

```
try:
    entrada = input("Ingresa un número entero: ")
    numero = int(entrada)
    print("Número ingresado:", numero)
except ValueError:
    print("Error: Debes ingresar un número entero válido.")
```

Ejercicio Práctico

Escribe un programa que reciba el nombre, la edad y el salario de una persona, y luego muestre un mensaje en el siguiente formato:

"Hola, [nombre]. Tienes [edad] años y tu salario es \${salario}."

Asegúrate de que el salario se muestre con dos decimales.

Ejemplo de Solución:

```
nombre = input("Ingresa tu nombre: ")
edad = int(input("Ingresa tu edad: "))
salario = float(input("Ingresa tu salario: "))

mensaje = f"Hola, {nombre}. Tienes {edad} años y tu salario es ${salario:.2f}."
print(mensaje)
```

Resumen del Tema

- Las **conversiones de tipo** permiten cambiar el tipo de dato de una variable a otro, lo cual es útil para realizar operaciones entre diferentes tipos de datos.

- Python permite realizar conversiones **implícitas** (automáticas) y **explícitas** (usando funciones de conversión como `int()`, `float()`, `str()`, `bool()`, `complex()`).
- Usar **conversiones explícitas** es necesario cuando se necesitan resultados específicos en una operación, como concatenar texto con números o calcular promedios a partir de datos en formato de cadena.
- Es una buena práctica verificar los datos antes de convertirlos y manejar los posibles errores en las conversiones.

Conclusión

Las conversiones de tipo en Python son una herramienta fundamental para manejar diferentes tipos de datos en el código. Comprender cuándo y cómo realizar estas conversiones te permitirá manipular información de manera efectiva y evitar errores comunes al operar entre tipos incompatibles.

Operadores y Expresiones en Python

Introducción a los Operadores

En Python, los operadores son símbolos o palabras que permiten realizar operaciones sobre variables y valores. Estos operadores son esenciales para construir expresiones que llevan a cabo cálculos y evaluaciones, como sumar números, comparar valores o combinar condiciones lógicas. Python soporta varios tipos de operadores que se dividen en categorías, cada una diseñada para un propósito específico.

Las expresiones en Python son combinaciones de variables, operadores y valores que se evalúan para producir un resultado. Estas expresiones son fundamentales para crear lógicas y estructuras de control en el programa, ya que permiten definir condiciones y realizar operaciones complejas.

Tipos de Operadores en Python

1. Operadores Aritméticos

- Se utilizan para realizar operaciones matemáticas básicas como suma, resta, multiplicación, división, etc.
- **Ejemplos de Operadores Aritméticos:**

Operador	Descripción	Ejemplo	Resultado
+	Suma	3 + 2	5
-	Resta	5 - 3	2
*	Multiplicación	4 * 2	8
/	División	10 / 2	5.0
//	División entera	10 // 3	3
%	Módulo	10 % 3	1
**	Exponenciación	2 ** 3	8

Ejemplo de Uso:

```

a = 10
b = 3
suma = a + b
division_entera = a // b
potencia = a ** b
print("Suma:", suma)
print("División entera:", division_entera)
print("Potencia:", potencia)

```

Operadores de Asignación

- Estos operadores se utilizan para asignar valores a las variables. Además del operador de asignación simple `=`, Python ofrece operadores de asignación compuestos que combinan una operación aritmética con la asignación.
- **Ejemplos de Operadores de Asignación:**

Operador	Descripción	Ejemplo	Equivalente
=	Asignación simple	x = 5	x = 5
+=	Suma y asignación	x += 3	x = x + 3

<code>-=</code>	Resta y asignación	<code>x -= 2</code>	<code>x = x - 2</code>
<code>*=</code>	Multiplicación y asignación	<code>x *= 4</code>	<code>x = x * 4</code>
<code>/=</code>	División y asignación	<code>x /= 5</code>	<code>x = x / 5</code>
<code>//=</code>	División entera y asignación	<code>x //= 2</code>	<code>x = x // 2</code>
<code>%=</code>	Módulo y asignación	<code>x %= 3</code>	<code>x = x % 3</code>
<code>**=</code>	Potencia y asignación	<code>x **= 2</code>	<code>x = x ** 2</code>

Ejemplo de Uso:

```
x = 10
x += 5
x **= 2
print("Valor de x:", x)
```

2. Operadores de Comparación

- Estos operadores permiten comparar dos valores y devuelven un valor booleano (**True** o **False**). Son fundamentales para la creación de condiciones en sentencias de control como **if** y **while**.
- Ejemplos de Operadores de Comparación:**

Operador	Descripción	Ejemplo	Resultado
<code>==</code>	Igual a	<code>5 == 5</code>	True
<code>!=</code>	Distinto de	<code>5 != 3</code>	True
<code>></code>	Mayor que	<code>7 > 5</code>	True
<code><</code>	Menor que	<code>3 < 4</code>	True
<code>>=</code>	Mayor o igual que	<code>5 >= 5</code>	True
<code><=</code>	Menor o igual que	<code>4 <= 5</code>	True

Ejemplo de Uso:

```
a = 8
b = 5
resultado = a > b
print("¿A es mayor que B?", resultado)
```

3. Operadores Lógicos

- Los operadores lógicos (**and**, **or**, **not**) permiten combinar expresiones booleanas y son especialmente útiles en condiciones compuestas.

○ Ejemplos de Operadores Lógicos:

Operador	Descripción	Ejemplo	Resultado
and	Devuelve True si ambas condiciones son True	(a > 5) and (b < 10)	True
or	Devuelve True si al menos una condición es True	(a > 10) or (b < 10)	True
not	Invierte el valor booleano	not(a > 10)	True

Ejemplo de Uso:

```
edad = 20
es_adulto = edad >= 18
tiene_identificacion = True
puede_entrar = es_adulto and tiene_identificacion
print("¿Puede entrar?", puede_entrar)
```

4. Operadores de Identidad

- Los operadores de identidad (**is**, **is not**) se utilizan para verificar si dos variables apuntan al mismo objeto en la memoria.

Ejemplos de Operadores de Identidad:

```
x = [1, 2, 3]
y = x
z = [1, 2, 3]
```

```
print(x is y)          # True, porque 'y' apunta a la misma lista
                        # que 'x'
print(x is z)          # False, porque 'z' es una lista diferente
                        # aunque con el mismo contenido
```

5. Operadores de Pertenencia

- Los operadores de pertenencia (**in**, **not in**) permiten verificar si un elemento está presente en una secuencia, como una lista, una tupla o una cadena de texto.

Ejemplo de Uso:

```
lista = [1, 2, 3, 4, 5]
print(3 in lista)      # True
print(6 not in lista)  # True
```

Ejemplos de Uso de Expresiones en Python

1. Cálculo de Descuento

- Calcular el precio final aplicando un descuento solo si el precio es mayor a 100.

```
precio = 120
descuento = 0.1
if precio > 100:
    precio -= precio * descuento
print("Precio final:", precio)
```

2.

3. Evaluación de Edad para Entrada a un Evento

- Usar operadores de comparación y lógicos para verificar si una persona puede entrar a un evento.

```
edad = 20

es_vip = True
if edad >= 18 or es_vip:
    print("Puede entrar al evento.")
else:
    print("No puede entrar al evento.")
```

4. Suma Condicional en una Lista

- Sumar solo los números pares de una lista.

```
numeros = [1, 2, 3, 4, 5, 6]

suma_pares = sum([num for num in numeros if num % 2 == 0])
print("Suma de pares:", suma_pares)
```

Buenas Prácticas al Usar Operadores y Expresiones

1. **Usar Paréntesis en Expresiones Complejas:** Cuando las expresiones son complejas y combinan múltiples operadores, usa paréntesis para mejorar la legibilidad y asegurar que se evalúan en el orden deseado.
2. **Evitar Comparaciones Innecesarias:** En lugar de `if x == True`, usa `if x`. En lugar de `if x == False`, usa `if not x`.
3. **Asignaciones Compuestas:** Aprovecha los operadores de asignación compuestos (`+=`, `-=`, etc.) para simplificar el código y evitar redundancias.

Ejercicio Práctico

Escribe un programa que reciba una lista de edades e imprima la cantidad de personas mayores de edad (18 años o más) en la lista.

Ejemplo de Solución:

```
edades = [15, 20, 17, 30, 22, 13, 18]
mayores = sum([1 for edad in edades if edad >= 18])
print("Número de personas mayores de edad:", mayores)
```

Resumen del Tema

- Los **operadores** en Python son herramientas fundamentales para realizar cálculos, comparaciones y evaluaciones lógicas en el código.
- Python soporta **operadores aritméticos, de asignación, de comparación, lógicos, de identidad y de pertenencia**, cada uno diseñado para una tarea específica.
- Las **expresiones** son combinaciones de variables, valores y operadores que se evalúan para producir un resultado, permitiendo crear lógicas complejas y condiciones en el programa.

- Es importante seguir **buenas prácticas** como usar paréntesis en expresiones complejas y operadores compuestos para simplificar el código y hacerlo más legible.

Conclusión

Dominar el uso de operadores y expresiones en Python es esencial para cualquier programador, ya que son la base de los cálculos y las condiciones en el código. Estos conocimientos te permitirán crear programas más dinámicos y funcionales, adaptados a diferentes necesidades y requisitos.

Operadores y Expresiones Aritméticas en Python

Introducción a los Operadores Aritméticos

Los operadores aritméticos en Python son símbolos que permiten realizar operaciones matemáticas básicas y avanzadas sobre números. Estos operadores incluyen suma, resta, multiplicación, división, entre otros. Son fundamentales en cualquier programa que implique cálculos numéricos, desde aplicaciones financieras hasta análisis científicos. Python es capaz de trabajar con números enteros (`int`), decimales (`float`) y complejos (`complex`), y permite combinar estos tipos de datos en expresiones aritméticas.

Lista de Operadores Aritméticos en Python

Python proporciona una serie de operadores aritméticos que se describen a continuación:

1. Suma (+)

- Se utiliza para sumar dos o más valores numéricos.

Ejemplo:

```
resultado = 5 + 3
print("Suma:", resultado) # Resultado: 8
```

2. Resta (-)

- Se utiliza para restar un valor de otro.

Ejemplo:

```
resultado = 10 - 4  
print("Resta:", resultado) # Resultado: 6
```

3. Multiplicación (*)

- Multiplica dos valores.

Ejemplo:

```
resultado = 7 * 3  
print("Multiplicación:", resultado) # Resultado: 21
```

4. División (/)

- Realiza una división entre dos valores y devuelve un resultado decimal (float).

Ejemplo:

```
resultado = 15 / 4  
print("División:", resultado) # Resultado: 3.75
```

5. División Entera (//)

- Realiza una división y devuelve solo la parte entera del resultado, eliminando los decimales.

Ejemplo:

```
resultado = 15 // 4  
print("División Entera:", resultado) # Resultado: 3
```

6. Módulo (%)

- Devuelve el residuo de una división entre dos valores. Es útil para operaciones que requieren divisibilidad, como verificar si un número es par o impar.

Ejemplo:

```
resultado = 10 % 3  
print("Módulo:", resultado) # Resultado: 1
```

7. Exponenciación (**)

- Eleva un número a la potencia de otro.

Ejemplo:

```
resultado = 2 ** 3
print("Exponenciación:", resultado) # Resultado: 8
```

Tabla Resumen de Operadores Aritméticos

Operador	Nombre	Descripción	Ejemplo	Resultado
+	Suma	Suma de dos números	5 + 3	8
-	Resta	Resta de un número a otro	10 - 4	6
*	Multiplicación	Multiplicación de dos números	7 * 3	21
/	División	División que retorna un valor decimal	15 / 4	3.75
//	División Entera	División que retorna solo la parte entera	15 // 4	3
%	Módulo	Resto de la división de dos números	10 % 3	1
**	Exponenciación	Eleva un número a la potencia de otro	2 ** 3	8

Ejemplos Prácticos de Uso de Operadores Aritméticos

1. Cálculo de Área de un Círculo

- Usando el operador de exponenciación para elevar al cuadrado el radio y el operador de multiplicación para calcular el área.

```
radio = 5
pi = 3.1416
area = pi * (radio ** 2)
print("Área del círculo:", area)
```

2. Conversión de Temperaturas

- Convertir una temperatura de grados Celsius a Fahrenheit usando operaciones aritméticas.

```
celsius = 25
fahrenheit = (celsius * 9/5) + 32
print("Temperatura en Fahrenheit:", fahrenheit)
```

3. Divisibilidad

- Verificar si un número es par o impar usando el operador módulo.

```
numero = 8
if numero % 2 == 0:
    print("El número es par")
else:
    print("El número es impar")
```

4. Cálculo de Interés Compuesto

- Calcular el valor final de una inversión utilizando el operador de exponenciación para aplicar la fórmula del interés compuesto.

```
principal = 1000 # Capital inicial
tasa_interes = 0.05 # Tasa de interés anual
periodos = 10 # Años
monto_final = principal * ((1 + tasa_interes) ** periodos)
print("Monto final después de 10 años:", monto_final)
```

5. Cálculo de Edad en Días

- Calcular la edad en días, multiplicando el número de años por la cantidad de días en un año (365).

```
edad_anos = 30
edad_dias = edad_anos * 365
print("Edad en días:", edad_dias)
```

Buenas Prácticas al Usar Operadores Aritméticos

Usar Paréntesis para Prioridad: Aunque Python sigue las reglas de precedencia de operadores, es recomendable usar paréntesis en expresiones complejas para mejorar la legibilidad y evitar errores.

```
resultado = (5 + 3) * 2
```

Evitar División por Cero: La división por cero (`/ 0` o `// 0`) produce un error en Python. Si existe la posibilidad de que un divisor sea cero, es una buena práctica verificar antes de realizar la operación.

```
divisor = 0
if divisor != 0:
    resultado = 10 / divisor
else:
    print("Error: división por cero.")
```

1. **Utilizar el Operador Módulo para Divisibilidad:** El operador módulo (%) es útil para verificar si un número es divisible por otro, como en el caso de números pares e impares.

Asignación de Resultados a Variables: Para evitar recalcular una operación repetidamente, asigna el resultado a una variable y reutilízalo cuando sea necesario.

```
subtotal = 100
descuento = 10
total = subtotal - descuento
print("Total con descuento:", total)
```

Ejercicio Práctico

Escribe un programa en Python que reciba la base y la altura de un triángulo, y luego calcule e imprima el área del triángulo.

Ejemplo de Solución:

```
base = float(input("Ingresa la base del triángulo: "))
altura = float(input("Ingresa la altura del triángulo: "))
area = (base * altura) / 2
print("El área del triángulo es:", area)
```

Resumen del Tema

- Los **operadores aritméticos** en Python permiten realizar operaciones matemáticas básicas como suma, resta, multiplicación, división, entre otras.

- Las expresiones aritméticas combinan operadores y valores para realizar cálculos y obtener resultados específicos.
- Es importante seguir **buenas prácticas** como usar paréntesis en expresiones complejas y verificar divisiones por cero para evitar errores en el código.
- Los operadores aritméticos son esenciales para muchas aplicaciones de programación, desde cálculos simples hasta fórmulas científicas y financieras.

Conclusión

Comprender y utilizar operadores aritméticos en Python es fundamental para cualquier programador, ya que son necesarios para una gran variedad de aplicaciones que involucran cálculos matemáticos. La práctica con estos operadores te permitirá crear programas más dinámicos y efectivos en la resolución de problemas numéricos.

Operadores y Expresiones Lógicas en Python

Introducción a los Operadores Lógicos

Los operadores lógicos en Python son herramientas que permiten realizar evaluaciones basadas en condiciones. Estos operadores son esenciales para crear expresiones complejas y condiciones compuestas que determinan el flujo de un programa, como decisiones en estructuras condicionales (`if`, `elif`, `else`) o bucles (`while`). Python cuenta con tres operadores lógicos principales: `and`, `or` y `not`. Estos operadores trabajan con valores booleanos (`True` y `False`) y devuelven resultados que se pueden utilizar para tomar decisiones en el código.

Tipos de Operadores Lógicos

1. Operador `and` (y)

- El operador `and` evalúa dos condiciones y devuelve `True` solo si ambas son verdaderas. Si una de las condiciones es `False`, el resultado es `False`.
- **Tabla de Verdad del Operador `and`:**

Condición A	Condición B	A <code>and</code> B
True	True	True

True	False	False
False	True	False
False	False	False

Ejemplo:

```
edad = 25
```

```
tiene_identificacion = True
```

```
puede_entrar = (edad >= 18) and tiene_identificacion
```

```
print("¿Puede entrar?", puede_entrar) # Resultado: True
```

2. Operador **or** (o)

- El operador **or** evalúa dos condiciones y devuelve **True** si al menos una de las condiciones es verdadera. Solo devuelve **False** cuando ambas condiciones son **False**.

- Tabla de Verdad del Operador **or**:**

Condición A	Condición B	A or B
True	True	True
True	False	True
False	True	True
False	False	False

Ejemplo:

```
edad = 16
```

```
es_vip = True
```

```
puede_entrar = (edad >= 18) or es_vip
```

```
print("¿Puede entrar?", puede_entrar) # Resultado: True
```

3. Operador **not** (no)

- El operador **not** invierte el valor de una expresión. Si la expresión es **True**, el resultado de **not** será **False**, y viceversa.

- **Tabla de Verdad del Operador not:**

Condición	not Condición
True	False
False	True

Ejemplo:

```
es_menor = False  
  
print("¿Es menor?", not es_menor) # Resultado: True
```

Ejemplos Prácticos de Uso de Operadores Lógicos

1. Verificación de Edad y Documentación para Entrada

- Usar operadores lógicos para verificar si una persona cumple con ambas condiciones: tener 18 años o más y tener identificación.

```
edad = 20  
  
tiene_identificacion = True  
  
puede_entrar = (edad >= 18) and tiene_identificacion  
  
print("¿Puede entrar?", puede_entrar) # Resultado: True
```

2. Permiso para Descuento en una Compra

- Permitir un descuento si el cliente es miembro o si su compra supera un monto mínimo.

```
es_miembro = False  
  
total_compra = 120  
  
tiene_descuento = es_miembro or (total_compra > 100)  
  
print("¿Tiene descuento?", tiene_descuento) # Resultado: True
```

3. Validación de Entrada para un Sistema

- Verificar si un usuario tiene permisos o si es un administrador. También invertir el valor booleano de la condición para verificar que el usuario no esté bloqueado.

```
es_admin = False

tiene_permisos = True

esta_bloqueado = False

puede_acceder = (es_admin or tiene_permisos) and not
esta_bloqueado

print("¿Puede acceder?", puede_acceder) # Resultado: True
```

4. Verificación de Rango de Edad

- Usar operadores lógicos para verificar si una persona está dentro de un rango de edad específico.

```
edad = 30

es_joven_adulto = (edad >= 18) and (edad <= 35)

print("¿Es joven adulto?", es_joven_adulto) # Resultado: True
```

5. Sistema de Autorización para Empleados

- Un empleado puede acceder a ciertas áreas si tiene una autorización específica o si pertenece a una categoría de empleados que no requiere dicha autorización.

```
tiene_autorizacion = True

es_empleado_temporal = False

puede_acceder = tiene_autorizacion or not es_empleado_temporal

print("¿Puede acceder al área?", puede_acceder) # Resultado:
True
```

Buenas Prácticas al Usar Operadores Lógicos

1. **Evitar Condiciones Redundantes:** Usa condiciones simples y evita repetir expresiones que pueden simplificarse. Por ejemplo, en lugar de `if (x == True)`, usa simplemente `if x`.

Agrupar Expresiones con Paréntesis: Cuando trabajes con expresiones lógicas complejas, utiliza paréntesis para agrupar y mejorar la claridad de la condición. Esto también ayuda a controlar el orden de evaluación.

```
puede_entrar = (edad >= 18) and (tiene_identificacion or es_vip)
```

- 2.
3. **Priorizar la Legibilidad:** Asegúrate de que las expresiones sean fáciles de leer, especialmente si combinan varios operadores lógicos. Divide expresiones largas en varias líneas si es necesario.
4. **Evitar el Uso de `not` en Condiciones Complejas:** Aunque `not` es útil para invertir valores booleanos, su uso en expresiones complejas puede dificultar la lectura del código. Intenta estructurar la lógica para minimizar el uso de `not` cuando sea posible.

Ejercicio Práctico

Escribe un programa que verifique si un usuario puede acceder a un servicio de membresía. El acceso se concede si el usuario es mayor de 21 años y tiene suscripción activa, o si es miembro VIP sin importar su edad.

Ejemplo de Solución:

```
edad = int(input("Ingresa tu edad: "))

tiene_suscripcion = input("¿Tienes suscripción activa? (s/n): ")
                    .lower() == "s"

es_vip = input("¿Eres miembro VIP? (s/n): ").lower() == "s"

puede_acceder = (edad >= 21 and tiene_suscripcion) or es_vip
print("¿Puede acceder al servicio?", puede_acceder)
```

Resumen del Tema

- Los **operadores lógicos** (**and**, **or**, **not**) permiten realizar evaluaciones y crear condiciones complejas en Python.
- El operador **and** devuelve **True** solo si ambas condiciones son verdaderas, mientras que el operador **or** devuelve **True** si al menos una condición es verdadera.
- El operador **not** invierte el valor booleano de una condición, convirtiendo **True** en **False** y viceversa.
- Los operadores lógicos se usan comúnmente en expresiones condicionales y estructuras de control para definir la lógica de los programas.

Conclusión

El dominio de los operadores y expresiones lógicas es fundamental para desarrollar programas en Python que respondan a condiciones y criterios específicos. Estos operadores son especialmente útiles en la toma de decisiones y en el control de flujo del programa, lo que permite crear aplicaciones que pueden reaccionar de manera inteligente a diferentes escenarios.

Sentencias Condicionales (if, elif, else) en Python

Introducción a las Sentencias Condicionales

Las sentencias condicionales son estructuras fundamentales en cualquier lenguaje de programación, y Python no es la excepción. Estas sentencias permiten al programa tomar decisiones basadas en condiciones. Dependiendo del resultado de una expresión lógica (verdadera o falsa), el programa puede ejecutar diferentes bloques de código. En Python, las sentencias condicionales incluyen **if**, **elif** y **else**, y juntas permiten construir flujos de control complejos y adaptativos.

La estructura básica de una sentencia condicional en Python se compone de la palabra clave **if**, seguida de una condición y un bloque de código que se ejecuta si la condición es verdadera. Python también proporciona **elif** (abreviatura de

"else if") para verificar condiciones adicionales y **else** para manejar todos los demás casos cuando las condiciones previas son falsas.

Estructura de las Sentencias Condicionales

1. Condicional Simple (**if**)

- La sentencia **if** permite ejecutar un bloque de código solo si una condición es verdadera.

Ejemplo:

```
edad = 20
if edad >= 18:
    print("Eres mayor de edad.")
```

2. Condicional Completo (**if - else**)

- Combina **if** con **else** para ejecutar un bloque de código si la condición es falsa.

Ejemplo:

```
temperatura = 30
if temperatura > 25:
    print("Hace calor.")
else:
    print("El clima es fresco.")
```

3. Condicional Múltiple (**if - elif - else**)

- Usa **elif** para evaluar condiciones adicionales si las condiciones anteriores son falsas. Se puede utilizar un número ilimitado de sentencias **elif**.

Ejemplo:

```
nota = 85
if nota >= 90:
    print("Excelente")
elif nota >= 80:
    print("Muy bien")
elif nota >= 70:
    print("Bien")
```

```
else:  
    print("Necesita mejorar")
```

Ejemplos Prácticos de Uso de Sentencias Condicionales

1. Verificar Edad para Votar

- Un programa simple para verificar si una persona es elegible para votar.

```
edad = int(input("Ingresa tu edad: "))  
if edad >= 18:  
    print("Puedes votar.")  
else:  
    print("No puedes votar.")
```

2. Clasificación de Temperatura

- Clasificar la temperatura en frío, templado o caliente.

```
temperatura = int(input("Ingresa la temperatura actual: "))  
if temperatura < 15:  
    print("Hace frío.")  
elif 15 <= temperatura <= 25:  
    print("El clima es templado.")  
else:  
    print("Hace calor.")
```

3. Calificación de un Examen

- Determinar el rango de una calificación y mostrar el resultado.

```
calificacion = int(input("Ingresa tu calificación: "))  
if calificacion >= 90:  
    print("Sobresaliente")  
elif calificacion >= 70:  
    print("Aprobado")  
else:  
    print("Reprobado")
```

4. Descuento en una Compra

- Calcular el precio final aplicando un descuento si el total de la compra supera los \$100.

```
total_compra = float(input("Ingresa el total de tu compra: "))
```

```
if total_compra > 100:
    descuento = total_compra * 0.1
    total_compra -= descuento
    print(f"Se aplicó un descuento de $ {descuento:.2f}")
else:
    print("No se aplica descuento.")
print(f"Total a pagar: $ {total_compra:.2f}")
```

5. Verificación de Acceso a un Sistema

- Comprobar si el usuario y la contraseña son correctos para conceder acceso.

```
usuario = input("Ingresa tu nombre de usuario: ")

contrasena = input("Ingresa tu contraseña: ")
if usuario == "admin" and contrasena == "12345":
    print("Acceso concedido")
else:
    print("Acceso denegado")
```

Buenas Prácticas al Usar Sentencias Condicionales

1. **Escribir Condiciones Claras y Sencillas:** Las condiciones deben ser fáciles de leer y comprender. Utiliza nombres descriptivos en tus variables para mejorar la legibilidad.
2. **Evitar Anidar Condiciones en Exceso:** Las condiciones anidadas pueden hacer que el código sea difícil de leer y mantener. Siempre que sea posible, intenta simplificar las condiciones o dividir el código en funciones.
3. **Usar `elif` en Lugar de Múltiples `if`:** Usar `elif` en lugar de varios `if` independientes mejora el rendimiento, ya que detiene la evaluación de condiciones una vez que encuentra una verdadera.
4. **Utilizar Comparaciones Claras:** En lugar de escribir condiciones redundantes, como `if x == True`, usa `if x`. De manera similar, para `False`, usa `if not x`.
5. **Usar `else` solo cuando sea Necesario:** Si el `else` no es necesario, es mejor omitirlo para evitar lógica innecesaria.

Ejercicio Práctico

Escribe un programa que reciba la edad de una persona y muestre un mensaje indicando si es un niño (menor de 12 años), un adolescente (entre 12 y 18 años), un adulto (entre 18 y 65 años) o un adulto mayor (mayor de 65 años).

Ejemplo de Solución:

```
edad = int(input("Ingresa tu edad: "))
if edad < 12:
    print("Eres un niño.")
elif edad < 18:
    print("Eres un adolescente.")
elif edad < 65:
    print("Eres un adulto.")
else:
    print("Eres un adulto mayor.")
```

Resumen del Tema

- Las **sentencias condicionales** en Python (**if**, **elif**, **else**) permiten ejecutar bloques de código en función de condiciones.
- La sentencia **if** ejecuta código solo si la condición es verdadera, **elif** permite evaluar condiciones adicionales, y **else** ejecuta código si todas las condiciones anteriores son falsas.
- Es importante seguir **buenas prácticas** como evitar anidaciones excesivas y escribir condiciones claras para mejorar la legibilidad y mantenibilidad del código.

Conclusión

Las sentencias condicionales son esenciales para controlar el flujo de un programa y tomar decisiones en función de condiciones. Con ellas, puedes crear programas más dinámicos y adaptativos que respondan a diferentes entradas y situaciones. Practicar con estas estructuras te permitirá desarrollar lógica condicional sólida, lo cual es fundamental para el desarrollo en Python y en cualquier otro lenguaje de programación.

Ejecución de Scripts en Python

Introducción a la Ejecución de Scripts

Un script en Python es un archivo de texto que contiene código escrito en el lenguaje Python y se ejecuta de forma secuencial, línea por línea, desde el principio hasta el final. Estos scripts se guardan con la extensión `.py` y se ejecutan en un entorno de Python, ya sea desde un terminal, un IDE, o un entorno interactivo como Jupyter Notebooks. La ejecución de scripts permite automatizar tareas, realizar cálculos, y ejecutar programas completos.

Cómo Crear y Ejecutar un Script en Python

1. Crear el Archivo Script

- Usa un editor de texto (como Notepad, VS Code, PyCharm, etc.) para escribir el código Python.
- Guarda el archivo con la extensión `.py`, por ejemplo, `mi_script.py`.

2. Ejecutar el Script en la Terminal

- Abre la terminal o consola de comandos.
- Navega hasta la ubicación del archivo guardado.
- Escribe el comando `python nombre_del_archivo.py` para ejecutar el script.

Ejemplo:

```
python mi_script.py
```

3. Ejecución en un IDE

- La mayoría de los IDEs y editores de código (como PyCharm o VS Code) tienen opciones para ejecutar scripts directamente presionando un botón de "Run" o ejecutando con teclas rápidas.

4. Ejemplo de un Script Simple

Código en `mi_script.py`:

```
nombre = input("Ingresa tu nombre: ")
print(f"Hola, {nombre}. Bienvenido a Python.")
```

- Ejecutar:

- En la terminal: `python mi_script.py`

Buenas Prácticas para Ejecución de Scripts

- **Comentarios:** Agrega comentarios para explicar partes clave del código.

- **Organización del Código:** Divide el código en funciones para hacerlo más modular y fácil de mantener.
- **Pruebas antes de Ejecutar en Producción:** Asegúrate de probar los scripts en un entorno seguro antes de ejecutarlos en producción.

Impresión en Consola

Introducción a la Impresión en Consola

La función `print()` en Python se utiliza para mostrar resultados y mensajes en la consola. Es fundamental para la depuración y la interacción con el usuario, ya que permite verificar el estado de las variables y la lógica del programa.

Uso de la Función `print()`

1. Sintaxis Básica

- La función `print()` imprime cualquier valor o mensaje que se pase como argumento.

Ejemplo:

```
print("Hola, mundo")
```

2. Concatenación de Valores

- Puedes concatenar cadenas y valores de variables utilizando el operador `+` o separándolos con comas en `print()`.

Ejemplo:

```
nombre = "Ana"
print("Hola, " + nombre)
print("Edad:", 25)
```

3. Uso de F-Strings

- Desde Python 3.6, las f-strings permiten formatear cadenas de manera fácil e intuitiva.

Ejemplo:

```
edad = 30
print(f"Tienes {edad} años.")
```

4. Formateo Avanzado

- La función `print()` también permite formatear números y texto con precisión.

Ejemplo:

```
precio = 49.987
print(f"El precio es ${precio:.2f}")
```

Ejemplo Completo de Impresión

```
nombre = "Luis"
edad = 28
salario = 1000.5

print(f"Nombre: {nombre}")
print(f"Edad: {edad}")
print(f"Salario mensual: ${salario:.2f}")
```

Entrada de Datos en Consola

Introducción a la Entrada de Datos

La función `input()` en Python se utiliza para recibir datos ingresados por el usuario desde la consola. Esta función siempre retorna un valor de tipo `str` (cadena de caracteres), por lo que es común realizar conversiones de tipo si se requieren otros formatos.

Uso de la Función `input()`

1. Sintaxis Básica

- `input()` muestra un mensaje y espera a que el usuario ingrese un valor, que luego es almacenado en una variable.

Ejemplo:

```
nombre = input("¿Cuál es tu nombre? ")
print("Hola, ", nombre)
```

2. Conversión de Tipos de Entrada

- Como `input()` devuelve una cadena, es común convertir el valor a tipos como `int` o `float`.

Ejemplo:

```
edad = int(input("¿Cuál es tu edad? "))  
print(f"Tienes {edad} años.")
```

3. Ejemplo Completo con Entradas y Conversiones

```
nombre = input("¿Cuál es tu nombre? ")  
edad = int(input("¿Cuántos años tienes? "))  
altura = float(input("¿Cuál es tu altura en metros? "))  
  
print(f"Nombre: {nombre}")  
print(f"Edad: {edad} años")  
print(f"Altura: {altura} metros")
```

Buenas Prácticas para la Entrada de Datos

- **Manejo de Errores:** Usa `try-except` para manejar errores de conversión.
- **Mensajes Claros:** Proporciona mensajes que indiquen claramente qué tipo de dato debe ingresar el usuario.

```
try:  
    edad = int(input("Ingresa tu edad (en números enteros):  
"))  
    print(f"Tienes {edad} años.")  
except ValueError:  
    print("Error: Debes ingresar un número entero.")
```


Cierre

A lo largo de este manual, hemos cubierto conceptos fundamentales de programación en Python, desde tipos de datos y operadores, hasta estructuras de control y manejo de entrada y salida en la consola. Estos conocimientos son esenciales para cualquier programador que desee construir aplicaciones en Python. La práctica de estos conceptos permitirá dominar los fundamentos y construir una base sólida para proyectos más complejos.

La ejecución de scripts y el uso de `print()` y `input()` son herramientas clave para interactuar con los usuarios y obtener datos esenciales para nuestras aplicaciones. A medida que avances, comprenderás cómo usar estos elementos de manera eficiente y en combinación con estructuras de control, funciones y otros componentes avanzados de Python. Con una sólida base en estos fundamentos, estarás listo para abordar desafíos de programación y desarrollar soluciones en Python.

Referencias

1. **Documentación Oficial de Python**
<https://docs.python.org/es/3/>
2. **Automate the Boring Stuff with Python - Al Sweigart**
Un libro gratuito y accesible en línea que cubre temas fundamentales y ejemplos prácticos.
3. **Python para Todos - Charles Severance**
Un recurso accesible para principiantes que introduce los fundamentos de Python con ejemplos claros y ejercicios.
4. **Real Python**
<https://realpython.com> - Una excelente fuente de tutoriales prácticos y avanzados sobre Python y sus aplicaciones.
5. **W3Schools Python Tutorial**
<https://www.w3schools.com/python/> - Un recurso introductorio que cubre todos los aspectos básicos de Python con ejemplos breves y prácticos.
6. **GeeksforGeeks**
<https://www.geeksforgeeks.org/python-programming-language/> - Ofrece una amplia variedad de artículos y ejemplos sobre temas específicos de Python.

¡Muchas gracias!

Nos vemos en la próxima lección