Guía esencial de Python

1. Variables y Tipos de Datos

Variables: Se crean asignando un valor con el operador `=`.

```
 x = 10      # Entero
y = 3.14    # Flotante
name = "Juan"  # Cadena
is_valid = True  # Booleano
```

- **Tipos de Datos Comunes:**
    - `int`: Enteros
    - `float`: Números decimales
    - `str`: Cadenas de texto
    - `bool`: Valores booleanos (`True` o `False`)
    - `list`: Listas
    - `tuple`: Tuplas (inmutables)
    - `dict`: Diccionarios
    - `set`: Conjuntos

---

2. Condicionales (if, elif, else)

Estructura básica:

```
 edad = 18
if edad >= 18:
    print("Eres mayor de edad")
elif edad == 17:
    print("Casi eres mayor de edad")
else:
    print("Eres menor de edad")
```

-

---

3. Bucles

3.1 Bucle `for`

Se utiliza para iterar sobre secuencias como listas o rangos.

```
for i in range(5):  # range(5) genera [0, 1, 2, 3, 4]
    print(i)

nombres = ["Ana", "Luis", "Pedro"]
for nombre in nombres:
    print(nombre)
```

3.2 Bucle `while`

Repite mientras una condición sea verdadera.

```
contador = 0
while contador < 5:
    print(contador)
    contador += 1
```

---

4. Switch-Case (Simulado)

Python no tiene `switch-case`, pero se puede simular con diccionarios.

```
def switch_case(opcion):
    opciones = {
        1: "Opcion 1",
        2: "Opcion 2",
        3: "Opcion 3"
    }
    return opciones.get(opcion, "Opcion no válida")

print(switch_case(2))
```

---

5. Funciones

5.1 Definir Funciones

Se usan para encapsular código reutilizable.

```
def saludar(nombre):
```

```
    return f"Hola, {nombre}!"

print(saludar("María"))
```

5.2 Argumentos por Defecto

```
def saludar(nombre, mensaje="¿Cómo estás?"):
    return f"Hola, {nombre}. {mensaje}"

print(saludar("Juan"))
```

---

6. Listas y Comprehensions

6.1 Operaciones con Listas

```
numeros = [1, 2, 3, 4]
numeros.append(5)  # Agregar elemento
numeros.remove(2)  # Eliminar elemento
print(numeros)
print(len(numeros))
```

```
digits = []
for symbol in s:
    if '1234567890'.find(symbol) != -1:
        digits.append(int(symbol))
print(digits)
```

6.2 List Comprehension

```
cuadrados = [x**2 for x in range(5)]  # [0, 1, 4, 9, 16]
print(cuadrados)
```

---

7. Manejo de Errores

Se usa `try-except` para manejar excepciones.

```
try:
    numero = int(input("Introduce un número: "))
    resultado = 10 / numero
except ValueError:
```

```
    print("Error: Debes ingresar un número válido.")
except ZeroDivisionError:
    print("Error: No puedes dividir entre cero.")
```

e

```
-   except ValueError:
-       print("Introduce un numero entero")
-   except Exception as e:
-       print("Error:- " , e)
```

-

---

8. Clases y Objetos

8.1 Crear una Clase

```
class Persona:
    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad

    def saludar(self):
        return f"Hola, me llamo {self.nombre} y tengo {self.edad} años."

p1 = Persona("Ana", 25)
print(p1.saludar())
```

8.2 Herencia entre Clases

La herencia permite que una clase herede propiedades y métodos de otra, lo que facilita la reutilización de código.

```
class Animal:
    def __init__(self, nombre):
        self.nombre = nombre

    def hablar(self):
        return "El animal hace un sonido."
```

```python
class Perro(Animal):  # Perro hereda de Animal
    def __init__(self, nombre, raza):
        super().__init__(nombre)  # Llama al constructor de la clase
base
        self.raza = raza

    def hablar(self):  # Sobrescribe el método hablar
        return "El perro dice: ¡Guau!"

# Crear una instancia de Perro
mi_perro = Perro("Rex", "Pastor Alemán")
print(mi_perro.nombre)  # Accede al atributo heredado
print(mi_perro.raza)  # Accede al atributo de la clase hija
print(mi_perro.hablar())  # Llama al método sobrescrito
```

---

9. Archivos

9.1 Leer Archivos

```
with open("archivo.txt", "r") as archivo:
    contenido = archivo.read()
    print(contenido)
```

9.2 Escribir Archivos

```
with open("archivo.txt", "w") as archivo:
    archivo.write("Hola, mundo!")
```

---

10. Módulos

10.1 Importar Módulos

```
import math
print(math.sqrt(16))

from math import pi
print(pi)
```

10.2 Crear un Módulo Propio

Archivo `mimodulo.py`:

```
def sumar(a, b):
    return a + b
```

Usarlo:

```
from mimodulo import sumar
print(sumar(3, 4))
```

---

12. Pythonic Tips

Operador ternario:

```
 edad = 20
mensaje = "Mayor" if edad >= 18 else "Menor"
print(mensaje)
```

-

Unpacking:

```
 a, b, c = [1, 2, 3]
print(a, b, c)
```

---

4. Diccionarios

Los diccionarios almacenan pares clave-valor y son muy útiles para búsquedas rápidas.

4.1 Crear un Diccionario

```
mi_diccionario = {
    "nombre": "Juan",
    "edad": 25,
    "ciudad": "Madrid"
}
```

4.2 Operaciones Comunes con Diccionarios

```python
# Acceder a un valor
print(mi_diccionario["nombre"])  # "Juan"

# Agregar un nuevo par clave-valor
mi_diccionario["profesión"] = "Ingeniero"

# Modificar un valor
mi_diccionario["edad"] = 26

# Eliminar una clave
del mi_diccionario["ciudad"]

# Iterar sobre un diccionario
for clave, valor in mi_diccionario.items():
    print(f"{clave}: {valor}")
```

STRINGS:

Usar `find` y `rfind`

- `str.find(subcadena)`: Devuelve la **primera posición** de la subcadena. Si no se encuentra, devuelve `-1`.
- `str.rfind(subcadena)`: Devuelve la **última posición** de la subcadena. Si no se encuentra, devuelve `-1`.

Ejemplo:

python
CopiarEditar

```python
cadena = "abracadabra"
letra = "a"

primera_posicion = cadena.find(letra)
ultima_posicion = cadena.rfind(letra)
```

CONCATENAR:

1. Usando `+` (Conversión manual con `str()`)

python
CopiarEditar

```
nombre = "Juan"
edad = 25
altura = 1.75

print("Nombre: " + nombre + ", Edad: " + str(edad) + ", Altura: " +
str(altura) + "m")
```

- **Problema:** Si intentas concatenar un `int` o `float` sin `str()`, obtendrás un error.

---

2. Usando `f-strings` (Recomendado ✅)

python
CopiarEditar

```
print(f"Nombre: {nombre}, Edad: {edad}, Altura: {altura}m")
```

✅ **Ventaja:** Más limpio, legible y no necesitas convertir tipos de datos manualmente.

---

3. Usando `format()`

python
CopiarEditar

```
print("Nombre: {}, Edad: {}, Altura: {}m".format(nombre, edad,
altura))
```

✅ **Ventaja:** Similar a `f-strings`, pero menos intuitivo.

---

4. Usando `join()` (Solo para strings)

Si tienes una lista de strings, `join()` es muy útil:

python
CopiarEditar

```
palabras = ["Hola", "mundo", "!"]
print(" ".join(palabras))  # Salida: Hola mundo !
```

- **Nota:** Solo funciona con strings, debes convertir números antes de usarlo.

---

5. Usando `,` en `print()` (Automática, pero con espacios)

python
CopiarEditar

```
print("Nombre:", nombre, "Edad:", edad, "Altura:", altura, "m")
```

ejemplos:

```
def calcularPromedio(numeros: list[int]) -> float:
    suma = 0
    if len(numeros) > 0:  # Usamos len() para comprobar si la lista tiene elementos
        for num in numeros:
            suma += num
        promedio = suma / len(numeros)  # Dividimos entre la cantidad de números
        return promedio
    else:
        print("La lista está vacía")
        return None  # Retornamos None si la lista está vacía

print("Introduce un número o 'fin' para terminar")
```

```python
n = input()
numeros = []

while n != "fin":
    if n.isdigit():  # Verificamos si la entrada es un número entero
        numeros.append(int(n))  # Convertimos a entero antes de agregar
    else:
        print("Entrada no válida, introduce un número entero")
    print("Introduce un número o 'fin' para terminar")
    n = input()

promedio = calcularPromedio(numeros)

if promedio is not None:
    print("El promedio es:", promedio)
```