Guía Completa para Ser Senior en Python

Esta guía cubre los conocimientos esenciales y ejemplos prácticos para dominar Python a nivel senior.

1. Fundamentos del Lenguaje

- **Tipado dinámico y fuerte**

```
x = 5
x = "Hola"  # Cambia de tipo
print(len(x))  # Funciona porque "Hola" es un string
# print(x + 2)  # Esto da error: no se puede sumar string con entero
```

- **Estructuras de datos integradas**

```
# Listas
numeros = [1, 2, 3, 4]
numeros.append(5)
print(numeros)

# Diccionarios
persona = {"nombre": "Ana", "edad": 30}
persona["profesion"] = "Ingeniera"
print(persona)
```

- **Comprehensions**

```
cuadrados = [x**2 for x in range(10)]
print(cuadrados)

dic = {"a": 1, "b": 2}
invertido = {v: k for k, v in dic.items()}
print(invertido)
```

- **Funciones**

```
def saludar(nombre="mundo", *args, **kwargs):
    print(f"Hola, {nombre}!")
    print("Argumentos adicionales:", args)
    print("Argumentos nombrados:", kwargs)

saludar("Juan", 1, 2, 3, edad=25, ciudad="Madrid")
```

- **Manejo de excepciones**

```
try:
    resultado = 10 / 0
except ZeroDivisionError:
    print("Error: No se puede dividir entre cero.")
finally:
    print("Fin del bloque try.")
```

2. Programación Orientada a Objetos

- **Clases y objetos**

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

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

persona = Persona("Juan", 30)
persona.saludar()
```

- **Decoradores de clases y métodos**

```
class MiClase:
    _contador = 0

    @classmethod
```

```
    def incrementar(cls):
        cls._contador += 1
        return cls._contador

print(MiClase.incrementar())  # 1
print(MiClase.incrementar())  # 2
```

3. Programación Funcional

- **Funciones de orden superior**

```
def operar(func, a, b):
    return func(a, b)

def sumar(x, y):
    return x + y

print(operar(sumar, 5, 3))  # 8
```

- **Generadores**

```
def generador():
    for i in range(5):
        yield i

for valor in generador():
    print(valor)
```

4. Módulos y Paquetes

- **Creación de módulos**

```
# archivo: modulo.py
def saludar():
    print("¡Hola desde el módulo!")

# archivo principal
```

```
import modulo
modulo.saludar()
```

5. Gestión de Datos

- **Manejo de archivos**

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

- **Serialización**

```
import json

datos = {"nombre": "Juan", "edad": 30}
json_string = json.dumps(datos)
print(json_string)
```

6. Programación Asíncrona

- **Async y await**

```
import asyncio

async def decir_hola():
    print("Hola")
    await asyncio.sleep(1)
    print("Adiós")

asyncio.run(decir_hola())
```

7. Buenas Prácticas

- **Pruebas unitarias**

```
import unittest

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

class TestSumar(unittest.TestCase):
    def test_suma(self):
        self.assertEqual(sumar(2, 3), 5)

if __name__ == "__main__":
    unittest.main()
```

8. Bibliotecas y Frameworks Clave

- **Uso de Flask**

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def home():
    return "¡Hola desde Flask!"

if __name__ == "__main__":
    app.run(debug=True)
```

9. Conocimientos Avanzados

- **Manejo de memoria**

```
import sys

x = [1, 2, 3]
print(sys.getsizeof(x))  # Tamaño en memoria
```

10. Desarrollo Profesional

- **Contribuir a proyectos Open Source**

  Explora GitHub y contribuye a proyectos relevantes para tu área.

¡Practica y experimenta con cada ejemplo para profundizar en tu conocimiento!