Prólogo

Este libro, "Python en 10 días", ha sido desarrollado con el objetivo de proporcionar una guía clara y accesible para quienes desean aprender Python desde cero hasta un nivel intermedio. A través de explicaciones concisas y ejemplos prácticos, el lector podrá desarrollar una comprensión sólida de los conceptos fundamentales del lenguaje.

El aprendizaje de Python es una puerta de entrada a múltiples áreas del desarrollo y la tecnología, incluyendo inteligencia artificial, ciencia de datos, desarrollo web y automatización. Este material está diseñado para ser útil tanto para principiantes como para aquellos que buscan reforzar sus conocimientos.

Espero que este libro sea de gran ayuda en tu viaje de aprendizaje y que encuentres en Python una herramienta poderosa y versátil para tus proyectos y aspiraciones.

Licencia Creative Commons

Este libro está licenciado bajo la **Licencia Creative Commons Atribución-CompartirIgual 4.0 Internacional (CC BY-SA 4.0)**. Esto significa que eres libre de:

- **Compartir** — copiar y redistribuir el material en cualquier medio o formato.
- **Adaptar** — remezclar, transformar y construir sobre el material para cualquier propósito, incluso comercialmente.

Bajo las siguientes condiciones:

- **Atribución** — Debes dar el crédito adecuado, proporcionar un enlace a la licencia e indicar si se han realizado cambios. Puedes hacerlo de cualquier manera razonable, pero no de una forma que sugiera que el licenciante te respalda a ti o a tu uso.
- **CompartirIgual** — Si remezclas, transformas o creas a partir del material, debes distribuir tus contribuciones bajo la misma licencia que el original.

Para más información, visita: https://creativecommons.org/licenses/by-sa/4.0/

Autor: Oscar de la Cuesta Campillo
@oscardelacuesta

Python en 10 días: De Cero a Programador

Día 1: Introducción a Python

- ¿Qué es Python? Historia y características principales.
- Instalación de Python y configuración del entorno.
- Primer programa: "Hola, Mundo".
- Uso de `print()`, comentarios y la consola interactiva.

Día 2: Variables y Tipos de Datos

- Números (`int`, `float`, `complex`), texto (`str`).
- Booleanos (`True`, `False`) y operadores lógicos.
- Conversión entre tipos (`int()`, `float()`, `str()`).
- Entrada del usuario con `input()`.

Día 3: Estructuras de Control

- Condicionales (`if`, `elif`, `else`).
- Bucles (`for`, `while`).
- Uso de `break` y `continue`.

Día 4: Listas, Tuplas y Diccionarios

- Listas: creación, acceso, modificación y métodos (`append()`, `pop()`, `sort()`, etc.).
- Tuplas: inmutabilidad y uso.
- Diccionarios: claves, valores y métodos (`keys()`, `values()`, `items()`).

Día 5: Funciones y Módulos

- Definición y uso de funciones con `def`.
- Parámetros y valores de retorno.
- Importación de módulos (`math`, `random`, `os`).
- Creación de módulos propios.

Día 6: Manejo de Archivos

- Lectura y escritura de archivos (`open()`, `read()`, `write()`).
- Manejo de excepciones con `try`, `except`, `finally`.
- Archivos CSV y JSON.

Día 7: Programación Orientada a Objetos (POO)

- Clases y objetos.
- Métodos y atributos.
- Herencia y polimorfismo.

Día 8: Librerías Útiles y Manipulación de Datos

- `numpy` para cálculos numéricos.
- `pandas` para manejo de datos.
- `matplotlib` para gráficos básicos.

Día 9: Desarrollo de una Aplicación Básica

- Uso de `tkinter` para interfaces gráficas.
- Creación de una calculadora o una app simple.
- Organización del código en módulos.

Día 10: Introducción a Web y APIs

- Uso de `requests` para consumir APIs.
- Introducción a `Flask` para crear una API web.
- Proyecto final: API simple con Flask.

Glosario de términos

Versiones Actuales, IDEs, Herramientas y Librerías de Interés

Python en 10 días: Día 1 - Introducción a Python

¿Qué es Python?

Python es un lenguaje de programación de alto nivel, interpretado y de propósito general. Es conocido por su sintaxis sencilla y clara, lo que lo hace ideal para principiantes. Su versatilidad permite su uso en desarrollo web, ciencia de datos, inteligencia artificial, automatización y mucho más.

Características principales de Python

- Sintaxis simple y fácil de aprender.
- Lenguaje interpretado (no requiere compilación).
- Gran comunidad y amplia documentación.
- Soporte para múltiples paradigmas (orientado a objetos, funcional, imperativo).
- Extensa biblioteca estándar y compatibilidad con librerías de terceros.

Instalación de Python

1. Descarga Python desde el sitio oficial: https://www.python.org/downloads/
2. Sigue las instrucciones para instalarlo según tu sistema operativo (Windows, macOS, Linux).
3. Verifica la instalación ejecutando en la terminal o consola:
4. `python --version`

   o en algunos sistemas:

   ```
   python3 --version
   ```

Primer Programa: "Hola, Mundo"

Una tradición en la programación es empezar con un simple programa que imprime "Hola, Mundo" en la pantalla.

Código:

```
print("Hola, Mundo")
```

Explicación:

- `print()` es una función incorporada en Python que muestra texto en la pantalla.
- El texto debe ir entre comillas dobles o simples.

Uso de la Consola Interactiva

Python permite ejecutar código línea por línea en su consola interactiva. Para abrirla, escribe `python` o `python3` en la terminal y prueba:

```
2 + 3
```

Esto devolverá `5`, ya que Python puede usarse como calculadora.

Comentarios en Python

Los comentarios se utilizan para hacer anotaciones en el código y no afectan su ejecución.

```
# Esto es un comentario
print("Esto se ejecutará")  # Comentario en la misma línea
```

Conclusión

Hoy hemos aprendido qué es Python, sus características, cómo instalarlo y ejecutar nuestro primer programa. En el próximo día, exploraremos las variables y los tipos de datos en Python.

Python en 10 días: Día 2 - Variables y Tipos de Datos

¿Qué son las Variables?

En Python, una variable es un espacio en la memoria donde se almacena un valor. No es necesario declarar el tipo de la variable, ya que Python lo detecta automáticamente.

Declaración de Variables:

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

- `nombre` es una cadena de texto (str).
- `edad` es un número entero (int).
- `altura` es un número decimal (float).
- `es_estudiante` es un valor booleano (bool).

Tipos de Datos en Python

Python tiene varios tipos de datos incorporados. Algunos de los más comunes son:

1. Números

Python soporta diferentes tipos de números:

```
entero = 10        # int
flotante = 3.14    # float
complejo = 2 + 3j  # complex
```

2. Cadenas de Texto (Strings)

Las cadenas se pueden definir con comillas simples o dobles.

```
saludo = "Hola, mundo"
nombre = 'Python'
```

Se pueden concatenar y manipular:

```
mensaje = saludo + " desde " + nombre
print(mensaje)  # Salida: Hola, mundo desde Python
```

3. Booleanos

Los valores booleanos representan `True` o `False`.

```
es_mayor = True
es_menor = False
```

Estos valores se usan en expresiones lógicas:

```
print(5 > 3)  # Devuelve True
print(10 == 5)  # Devuelve False
```

4. Conversión entre Tipos

Se pueden convertir entre tipos usando funciones incorporadas:

```
num_str = "10"
num_int = int(num_str)  # Convierte a entero
num_float = float(num_str)  # Convierte a flotante
print(num_int + num_float)  # Salida: 20.0
```

5. Entrada del Usuario

Python permite recibir datos del usuario con `input()`.

```
nombre = input("Introduce tu nombre: ")
print("Hola, " + nombre)
```

Por defecto, `input()` devuelve una cadena, pero se puede convertir:

```
edad = int(input("Introduce tu edad: "))
print("El próximo año tendrás", edad + 1)
```

Conclusión

Hoy aprendimos sobre variables y tipos de datos en Python. En el siguiente día exploraremos las estructuras de control como condicionales y bucles.

Python en 10 días: Día 3 - Estructuras de Control

Introducción

Las estructuras de control permiten dirigir el flujo de ejecución de un programa según ciertas condiciones. En este día aprenderemos sobre condicionales y bucles en Python.

Condicionales en Python

Los condicionales permiten ejecutar bloques de código según una condición dada.

Sentencia `if`

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

Si la condición es `True`, el bloque de código dentro de `if` se ejecuta.

Sentencia `if-else`

```
edad = 16
if edad >= 18:
    print("Eres mayor de edad")
else:
    print("Eres menor de edad")
```

Si la condición en `if` es `False`, se ejecuta el bloque en `else`.

Sentencia `if-elif-else`

```
nota = 85
if nota >= 90:
    print("Sobresaliente")
elif nota >= 70:
    print("Aprobado")
else:
    print("Reprobado")
```

Se pueden encadenar múltiples condiciones con `elif`.

Bucles en Python

Los bucles permiten repetir un bloque de código varias veces.

Bucle `while`

```
contador = 1
while contador <= 5:
    print("Iteración", contador)
    contador += 1
```

El bucle `while` se ejecuta mientras la condición sea `True`.

Bucle `for`

```
for i in range(5):
    print("Valor de i:", i)
```

El bucle `for` se ejecuta sobre un rango o una secuencia de elementos.

Uso de `break` y `continue`

```
for i in range(10):
    if i == 5:
        break  # Detiene el bucle
    print(i)
for i in range(10):
    if i % 2 == 0:
        continue  # Salta la iteración actual
    print(i)
```

Conclusión

Hoy aprendimos sobre condicionales y bucles en Python. En el siguiente día exploraremos listas, tuplas y diccionarios.

Python en 10 días: Día 4 - Listas, Tuplas y Diccionarios

Introducción

Las estructuras de datos permiten almacenar y manipular colecciones de elementos. En este día aprenderemos sobre listas, tuplas y diccionarios en Python.

Listas en Python

Las listas son estructuras ordenadas y mutables que pueden contener diferentes tipos de datos.

Creación de listas

```
numeros = [1, 2, 3, 4, 5]
nombres = ["Juan", "Ana", "Luis"]
mixta = [1, "Python", 3.14, True]
```

Acceso a elementos

```
print(numeros[0])  # Accede al primer elemento
print(nombres[-1])  # Accede al último elemento
```

Modificación de listas

```
numeros[0] = 10  # Modifica el primer elemento
print(numeros)
```

Métodos comunes de listas

```
numeros.append(6)  # Agrega un elemento al final
numeros.insert(2, 99)  # Inserta un elemento en una posición específica
numeros.pop()  # Elimina el último elemento
numeros.remove(99)  # Elimina el primer elemento con el valor dado
print(numeros)
```

Tuplas en Python

Las tuplas son estructuras similares a las listas, pero son inmutables (no pueden modificarse después de su creación).

Creación de tuplas

```
tupla = (1, 2, 3, "Python")
```

Acceso a elementos

```
print(tupla[1])  # Accede al segundo elemento
```

Desempaquetado de tuplas

```
a, b, c, d = tupla
print(a, d)
```

Diccionarios en Python

Los diccionarios son estructuras de datos que almacenan pares clave-valor.

Creación de diccionarios

```
persona = {"nombre": "Juan", "edad": 30, "ciudad": "Madrid"}
```

Acceso a valores

```
print(persona["nombre"])  # Devuelve "Juan"
```

Modificación de diccionarios

```
persona["edad"] = 31  # Modifica un valor
persona["profesión"] = "Ingeniero"  # Agrega una nueva clave-valor
```

Métodos comunes de diccionarios

```
print(persona.keys())  # Devuelve las claves
print(persona.values())  # Devuelve los valores
print(persona.items())  # Devuelve los pares clave-valor
```

Conclusión

Hoy aprendimos sobre listas, tuplas y diccionarios en Python. En el siguiente día exploraremos funciones y módulos para estructurar mejor nuestro código.

Python en 10 días: Día 5 - Funciones y Módulos

Introducción

Las funciones y módulos permiten estructurar mejor el código, facilitando su reutilización y mantenimiento. Hoy aprenderemos cómo definir funciones y usar módulos en Python.

Funciones en Python

Las funciones permiten encapsular bloques de código reutilizables.

Definición de una función

```
def saludar():
    print("Hola, bienvenido a Python")
```

Llamada a una función

```
saludar()
```

Funciones con parámetros

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

resultado = sumar(5, 3)
print("Resultado:", resultado)
```

Parámetros con valores por defecto

```
def potencia(base, exponente=2):
    return base ** exponente

print(potencia(3))  # Usa el valor por defecto de exponente
print(potencia(3, 3))  # Exponente definido manualmente
```

Argumentos y parámetros variables

```
def sumar_todo(*numeros):
    return sum(numeros)

print(sumar_todo(1, 2, 3, 4, 5))
```

Módulos en Python

Los módulos permiten organizar el código en archivos reutilizables.

Importación de módulos

```
import math
print(math.sqrt(16))  # Calcula la raíz cuadrada de 16
```

Importar funciones específicas

```
from math import pi, sin
print(pi)
print(sin(0))
```

Creación de un módulo propio

1. Crea un archivo `operaciones.py` con el siguiente código:

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

def restar(a, b):
    return a - b
```

2. Importa y usa el módulo en otro archivo:

```
import operaciones
print(operaciones.sumar(5, 3))
```

Conclusión

Hoy aprendimos sobre funciones y módulos en Python. En el siguiente día exploraremos la manipulación de archivos y el manejo de excepciones.

Python en 10 días: Día 6 - Manejo de Archivos y Excepciones

Introducción

El manejo de archivos permite leer y escribir datos en archivos, mientras que la gestión de excepciones nos ayuda a manejar errores de manera segura en nuestros programas.

Manejo de Archivos en Python

Python permite interactuar con archivos mediante la función `open()`.

Apertura y lectura de archivos

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

- `"r"` indica que abrimos el archivo en modo lectura.
- `with` garantiza el cierre correcto del archivo.

Lectura línea por línea

```
with open("archivo.txt", "r") as archivo:
    for linea in archivo:
        print(linea.strip())
```

Escritura en archivos

```
with open("archivo.txt", "w") as archivo:
    archivo.write("Hola, este es un nuevo archivo\n")
```

- `"w"` sobrescribe el archivo si ya existe.

Agregar contenido a un archivo

```
with open("archivo.txt", "a") as archivo:
    archivo.write("Nueva línea agregada\n")
```

- `"a"` permite agregar contenido sin borrar lo existente.

Manejo de Excepciones

Las excepciones permiten manejar errores de manera controlada para evitar que el programa se detenga abruptamente.

Bloque `try-except`

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

Captura de múltiples excepciones

```
try:
    numero = int(input("Introduce un número: "))
    resultado = 10 / numero
except ValueError:
    print("Error: Debes ingresar un número entero")
except ZeroDivisionError:
    print("Error: No se puede dividir entre cero")
```

Uso de `finally`

El bloque `finally` se ejecuta siempre, haya o no una excepción.

```
try:
    archivo = open("archivo.txt", "r")
    contenido = archivo.read()
finally:
    archivo.close()
```

Conclusión

Hoy aprendimos a leer y escribir archivos en Python, así como a manejar excepciones. Mañana veremos programación orientada a objetos (POO).

Python en 10 días: Día 7 - Programación Orientada a Objetos (POO)

Introducción

La Programación Orientada a Objetos (POO) es un paradigma que organiza el código en clases y objetos, facilitando la reutilización y estructuración del código.

Clases y Objetos

En Python, una **clase** es un modelo para crear objetos, que representan entidades con propiedades y comportamientos.

Definición de una clase

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

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

Creación de un objeto

```
persona1 = Persona("Juan", 30)
print(persona1.saludar())
```

Herencia

La herencia permite que una clase hija herede atributos y métodos de una clase padre.

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

    def info(self):
        return f"Soy {self.nombre} y estudio en el curso {self.curso}."
```

Uso de la herencia

```
estudiante1 = Estudiante("Ana", 22, "Matemáticas")
print(estudiante1.info())
```

Encapsulamiento

El encapsulamiento protege los atributos de una clase para que no sean modificados directamente.

```
class CuentaBancaria:
    def __init__(self, titular, saldo):
        self.titular = titular
        self.__saldo = saldo  # Atributo privado

    def mostrar_saldo(self):
        return f"Saldo disponible: {self.__saldo}"
```

Acceso seguro a atributos privados

```
cuenta = CuentaBancaria("Pedro", 1000)
print(cuenta.mostrar_saldo())
```

Polimorfismo

El polimorfismo permite que métodos con el mismo nombre puedan comportarse de manera diferente según la clase que los implemente.

```
class Animal:
    def hacer_sonido(self):
        pass

class Perro(Animal):
    def hacer_sonido(self):
        return "Guau!"

class Gato(Animal):
    def hacer_sonido(self):
        return "Miau!"
```

Uso del polimorfismo

```
animales = [Perro(), Gato()]
for animal in animales:
    print(animal.hacer_sonido())
```

Conclusión

Hoy aprendimos los conceptos fundamentales de la Programación Orientada a Objetos en Python. Mañana exploraremos librerías útiles y la manipulación de datos.

Python en 10 días: Día 8 - Librerías útiles y Manipulación de Datos

Introducción

Python cuenta con una gran cantidad de librerías que facilitan el desarrollo en diferentes ámbitos. Hoy exploraremos algunas de las más útiles para la manipulación de datos.

`numpy` - Cálculo numérico

`numpy` es una librería utilizada para operaciones matemáticas eficientes con arreglos.

Instalación

```
pip install numpy
```

Uso básico

```
import numpy as np

arr = np.array([1, 2, 3, 4])
print(arr * 2)  # Multiplica cada elemento por 2
```

`pandas` - Manipulación de Datos

`pandas` permite manejar datos en estructuras como DataFrames y Series.

Instalación

```
pip install pandas
```

Uso básico

```
import pandas as pd

df = pd.DataFrame({"Nombre": ["Ana", "Luis"], "Edad": [25, 30]})
print(df)
```

`matplotlib` - Gráficos y Visualización

`matplotlib` permite generar gráficos de datos de manera sencilla.

Instalación

```
pip install matplotlib
```

Creación de un gráfico básico

```
import matplotlib.pyplot as plt

x = [1, 2, 3, 4]
y = [10, 20, 25, 30]

plt.plot(x, y)
plt.xlabel("X Axis")
plt.ylabel("Y Axis")
plt.title("Gráfico de Líneas")
plt.show()
```

`seaborn` - Gráficos Avanzados

`seaborn` mejora la visualización de datos con gráficos estilizados.

Instalación

```
pip install seaborn
```

Gráfico de distribución

```
import seaborn as sns
import numpy as np

sns.histplot(np.random.randn(1000), bins=30)
plt.show()
```

Conclusión

Hoy exploramos librerías esenciales para manipulación de datos y visualización. Mañana aprenderemos a desarrollar una aplicación básica en Python.

Python en 10 días: Día 9 - Desarrollo de una Aplicación Básica

Introducción

Hoy aprenderemos a desarrollar una aplicación básica en Python utilizando `tkinter`, una biblioteca para interfaces gráficas de usuario (GUI). Crearemos una calculadora simple.

Instalación de `tkinter`

`tkinter` viene preinstalado con Python, por lo que no es necesario instalarlo. Podemos importarlo directamente en nuestro código.

Creación de la Calculadora

Estructura básica

```
import tkinter as tk

def on_click(boton):
    if boton == "=":
        try:
            resultado.set(eval(entrada.get()))
        except:
            resultado.set("Error")
    elif boton == "C":
        entrada.set("")
        resultado.set("")
    else:
        entrada.set(entrada.get() + boton)

# Configuración de la ventana
ventana = tk.Tk()
ventana.title("Calculadora")
entrada = tk.StringVar()
resultado = tk.StringVar()

# Campo de entrada
pantalla = tk.Entry(ventana, textvariable=entrada, font=("Arial", 20), bd=10,
insertwidth=2, width=14, justify="right")
pantalla.grid(row=0, column=0, columnspan=4)
```

```
# Botones de la calculadora
botones = [
    ["7", "8", "9", "/"],
    ["4", "5", "6", "*"],
    ["1", "2", "3", "-"],
    ["0", "C", "=", "+"]
]

for i, fila in enumerate(botones):
    for j, boton in enumerate(fila):
        tk.Button(ventana, text=boton, padx=20, pady=20, font=("Arial", 20),
command=lambda b=boton: on_click(b)).grid(row=i+1, column=j)

ventana.mainloop()
```

Explicación del Código

- Se crea una ventana con `tk.Tk()`.
- Se define un campo de entrada para mostrar los números y operaciones.
- Se generan botones de manera dinámica.
- La función `on_click()` maneja los eventos de los botones.
- `eval()` se usa para calcular la expresión introducida.

Conclusión

Hoy construimos una aplicación básica con `tkinter`. Mañana aprenderemos a trabajar con APIs y a desarrollar servicios web en Python.

Python en 10 días: Día 10 - Introducción a Web y APIs

Introducción

Hoy aprenderemos a trabajar con APIs y a desarrollar un servicio web básico en Python utilizando `Flask`.

¿Qué es una API?

Una API (Interfaz de Programación de Aplicaciones) permite la comunicación entre aplicaciones a través de peticiones HTTP.

Instalación de `Flask`

```
pip install flask
```

Creación de una API con Flask

Estructura básica de una API en Flask

```
from flask import Flask, jsonify, request

app = Flask(__name__)

data = [
    {"id": 1, "nombre": "Producto 1", "precio": 10},
    {"id": 2, "nombre": "Producto 2", "precio": 20}
]

@app.route("/productos", methods=["GET"])
def obtener_productos():
    return jsonify(data)

@app.route("/productos", methods=["POST"])
def agregar_producto():
    nuevo_producto = request.json
    data.append(nuevo_producto)
    return jsonify({"mensaje": "Producto agregado"}), 201

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

Explicación del Código

- Se crea una instancia de Flask.
- Se define una lista `data` que simula una base de datos.
- Se implementa un endpoint `GET /productos` para obtener productos.
- Se define un endpoint `POST /productos` para agregar nuevos productos.
- `jsonify()` convierte datos a formato JSON.
- `request.json` obtiene datos enviados en formato JSON.

Probando la API

Para ejecutar la API, usa el comando:

```
python api.py
```

Luego, abre Postman o usa `curl` para probar:

```
curl -X GET http://127.0.0.1:5000/productos
```

Para agregar un producto:

```
curl -X POST http://127.0.0.1:5000/productos -H "Content-Type:
application/json" -d '{"id":3, "nombre":"Producto 3", "precio":30}'
```

Conclusión

Hoy aprendimos a crear una API REST con Flask. Con esto hemos finalizado nuestro curso de 10 días sobre Python. ¡Felicitaciones por completar el aprendizaje!

Glosario de Términos

Este glosario contiene definiciones de términos clave utilizados en el libro *Python en 10 días*.

API (Interfaz de Programación de Aplicaciones)

Un conjunto de reglas y definiciones que permiten a diferentes programas comunicarse entre sí. En Python, se utilizan librerías como `requests` para interactuar con APIs.

Argumento

Un valor que se pasa a una función cuando se llama. Puede ser posicional o con nombre.

Biblioteca (Librería)

Colección de módulos y funciones predefinidas que pueden utilizarse en un programa. Ejemplos incluyen `math`, `pandas` y `flask`.

Booleano

Un tipo de dato que solo puede tener dos valores: `True` o `False`. Se usa comúnmente en estructuras de control.

Bucle

Estructura de control que permite ejecutar repetitivamente un bloque de código. En Python, se usan `for` y `while`.

Clase

Plantilla para crear objetos. Define atributos y métodos comunes para sus instancias.

Diccionario

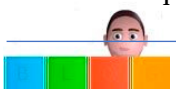
Estructura de datos en Python que almacena pares clave-valor. Se define con llaves `{}`.

Entorno Virtual

Un espacio aislado donde se pueden instalar paquetes sin afectar el sistema global. Se crea con `venv` o `virtualenv`.

Excepción

Un error que interrumpe la ejecución de un programa. Se maneja con `try-except`.



Función

Bloque de código reutilizable que se define con `def` y puede recibir argumentos y devolver valores.

IDE (Entorno de Desarrollo Integrado)

Software que facilita la programación con herramientas como resaltado de sintaxis, depuración y ejecución. Ejemplos: PyCharm, VS Code.

Lista

Estructura de datos ordenada y mutable en Python. Se define con corchetes `[]`.

Módulo

Archivo Python que contiene funciones y clases reutilizables. Se importa con `import`.

Objeto

Instancia de una clase que contiene atributos y métodos.

PIP

Gestor de paquetes de Python que permite instalar bibliotecas externas.

POO (Programación Orientada a Objetos)

Paradigma de programación basado en clases y objetos.

Repositorio

Lugar donde se almacenan y gestionan proyectos de código, como en GitHub o GitLab.

Tupla

Estructura de datos similar a una lista, pero inmutable. Se define con `()`.

Variable

Espacio en memoria donde se almacena un valor. No requiere declaración de tipo en Python.

Este glosario servirá como referencia rápida para los conceptos clave tratados en el libro.

Anexo: Versiones Actuales, IDEs, Herramientas y Librerías de Interés

Versiones Actuales de Python

Python evoluciona constantemente con nuevas versiones que mejoran rendimiento y seguridad. Para verificar tu versión instalada, usa:

```
python --version
```

Algunas versiones recientes son:

- **Python 3.12** (Última versión estable)
- **Python 3.11** (Optimización de rendimiento y mejoras en `match-case`)
- **Python 3.10** (Mejoras en tipado y sintaxis)

Se recomienda utilizar siempre la versión más reciente compatible con tus proyectos.

IDEs y Entornos de Desarrollo

Un buen entorno de desarrollo facilita la programación en Python. Algunas opciones recomendadas son:

IDEs Populares

- **PyCharm** (Profesional y Community Edition)
- **Visual Studio Code** (Extensión oficial de Python)
- **Jupyter Notebook** (Ideal para ciencia de datos y machine learning)
- **Spyder** (Optimizado para análisis científico)
- **Thonny** (Recomendado para principiantes)

Consolas y Entornos Interactivos

- **IPython** (Shell interactivo mejorado)
- **REPL** (`python` en terminal)

Herramientas Útiles

Administradores de Paquetes

- **pip** (`pip install paquete`) - Instalador de paquetes estándar.
- **conda** (`conda install paquete`) - Útil para entornos científicos.

Entornos Virtuales

- **venv** (`python -m venv mi_entorno`) - Manejo de dependencias.
- **virtualenv** (Alternativa a `venv`, más flexible).

Control de Versiones

- **Git** - Administración de código fuente.
- **GitHub/GitLab/Bitbucket** - Hosting de repositorios.

Librerías de Interés

Desarrollo Web

- **Flask** - Framework web ligero.
- **Django** - Framework web robusto y escalable.
- **FastAPI** - Framework moderno para APIs.

Ciencia de Datos y Machine Learning

- **NumPy** - Cálculos numéricos eficientes.
- **Pandas** - Manipulación de datos en estructuras tabulares.
- **Matplotlib / Seaborn** - Visualización de datos.
- **Scikit-learn** - Machine Learning.
- **TensorFlow / PyTorch** - Deep Learning.

Automatización y Scripting

- **Selenium** - Automatización de navegación web.
- **Requests** - Manejo de peticiones HTTP.
- **BeautifulSoup** - Scraping web.

Desarrollo de Aplicaciones

- **Tkinter** - Interfaz gráfica estándar de Python.
- **PyQt / PySide** - Desarrollo de interfaces modernas.
- **Kivy** - Aplicaciones multiplataforma.

Conclusión

Este anexo proporciona una referencia sobre herramientas y librerías útiles para diferentes aplicaciones en Python. Mantenerse actualizado con las últimas versiones y herramientas puede mejorar significativamente la eficiencia y la calidad del código.