Programa lectivo
Python learning

Índice:

Temas

Conclusión: Empieza con lo básico y avanza a temas más complejos a medida que te sientas cómodo. La práctica es clave, así que trabaja en proyectos pequeños y busca siempre aprender algo nuevo. ¡Buena suerte en tu carrera como programador en Python! Si tienes alguna pregunta específica, no dudes en preguntar.

Tema 1: Fundamentos de Programación

Conceptos Básicos: Variables, tipos de datos (enteros, flotantes, cadenas, listas, diccionarios).

Ejemplo

```python
nombre = "Juan"
edad = 25
altura = 1.75
gustos = ["fútbol", "cine", "música"]
datos = {"nombre": nombre, "edad": edad}
```

Por qué se utiliza: Los fundamentos son la base de cualquier lenguaje de programación. Entender cómo funcionan las variables y los tipos de datos es crucial para manipular información en tus programas.

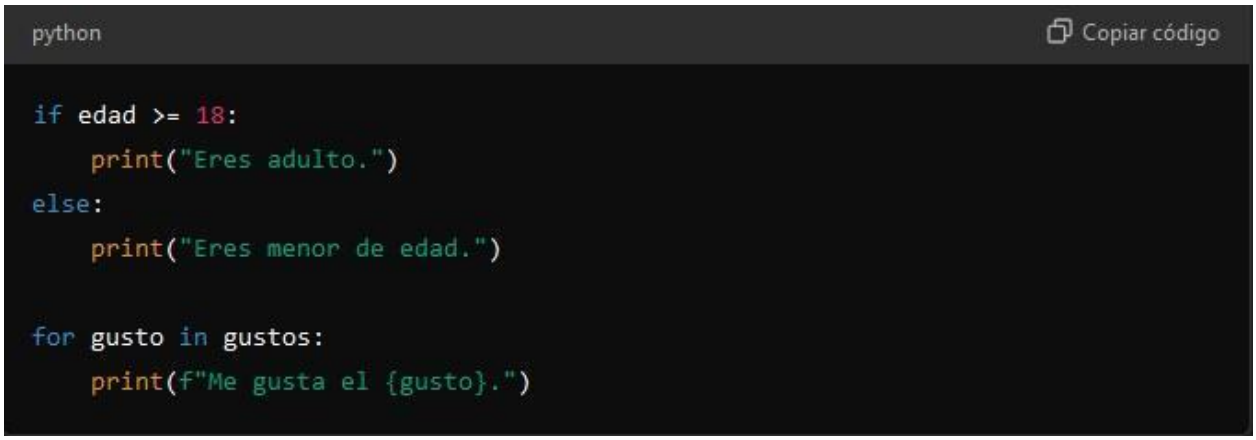
Función:

- **Variables**: Almacenan datos que puedes usar y modificar.
- **Tipos de Datos**: Determinan el tipo de información que puedes almacenar (números, texto, colecciones).

Tema 2: Control de Flujo / Bucles

Condicionales: `if`, `elif`, `else`.

Bucles: `for`, `while`.



Por qué se utiliza: Permite que tu programa tome decisiones y ejecute diferentes bloques de código según ciertas condiciones.

Función:

- **Condicionales**: Evalúan expresiones booleanas para ejecutar diferentes acciones (por ejemplo, decidir si un usuario es mayor de edad).
- **Bucles**: Repite un bloque de código varias veces, útil para procesar listas o realizar tareas repetitivas.

Tema 3: Funciones

Definición y llamada de funciones.

Parámetros y retorno de valores.

```python
def saludar(nombre):
    return f"Hola, {nombre}!"

print(saludar("María"))
```

Por qué se utiliza: Facilitan la organización y reutilización del código. Puedes definir una acción una vez y llamarla múltiples veces.

Función: Agrupan un conjunto de instrucciones que realizan una tarea específica y pueden recibir parámetros y devolver valores.

Tema 4: Estructura de Datos

Listas, tuplas, conjuntos y diccionarios.

```python
# Lista
frutas = ["manzana", "banana", "naranja"]
# Diccionario
persona = {"nombre": "Carlos", "edad": 30}
```

Por qué se utiliza: Permiten almacenar y organizar datos de manera eficiente. La elección de la estructura de datos adecuada puede afectar el rendimiento de tu programa.

Función:

- **Listas**: Almacenan colecciones de elementos ordenados y mutables.
- **Diccionarios**: Almacenan pares clave-valor, permitiendo acceso rápido a los valores mediante sus claves.

TEMA 5: Módulos y Paquetes

- Importar y utilizar módulos.
- Crear tus propios módulos.

```python
import math
print(math.sqrt(16))  # Salida: 4.0
```

Por qué se utiliza: Permiten organizar el código en diferentes archivos y reutilizar funciones sin tener que reescribirlas.

Función:

- **Módulos**: Archivos que contienen definiciones de funciones y variables.
- **Paquetes**: Colecciones de módulos organizados en directorios.

Tema 6: Manejo de Errores

- Uso de `try`, `except`.

```python
try:
    resultado = 10 / 0
except ZeroDivisionError:
    print("No se puede dividir por cero.")
```

Por qué se utiliza: Ayuda a manejar situaciones inesperadas sin que el programa se bloquee, mejorando la experiencia del usuario.

Función: Proporciona una manera de anticipar y reaccionar ante errores que pueden ocurrir durante la ejecución del programa.

Tema 7: Programación Orientada a Objetos (OOP)

- Clases y objetos, herencia, encapsulamiento.

```python
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."

juan = Persona("Juan", 25)
print(juan.saludar())
```

Por qué se utiliza: Facilita la modelación de problemas del mundo real, promoviendo la reutilización y la organización del código.

Función:

- **Clases**: Plantillas para crear objetos, que pueden contener atributos y métodos.
- **Objetos**: Instancias de clases que tienen sus propios datos y comportamientos.

Tema 8: Bibliotecas y Frameworks Comunes

- Familiarizarte con bibliotecas como NumPy, Pandas, Flask, Django.

Ejemplo usando biblioteca Flack

```python
from flask import Flask

app = Flask(__name__)

@app.route("/")
def home():
    return "¡Hola, Mundo!"

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

Por qué se utiliza: Extienden las capacidades de Python y permiten realizar tareas complejas con menos código y mayor eficiencia.

Función:

- **Bibliotecas**: Conjuntos de funciones predefinidas para realizar tareas específicas (por ejemplo, matemáticas, manipulación de datos).Las bibliotecas son **FUNCIONES IMPLEMENTADAS**
- **Frameworks**: Estructuras más grandes que proporcionan un conjunto de herramientas y convenciones para desarrollar aplicaciones (como web o científicas).

Tema 9: Control de Versiones

- Uso de Git para el control de versiones.
- Crear repositorios, realizar commits, branches.

```bash
git init
git add .
git commit -m "Primer commit"
```

Por qué se utiliza: Permite llevar un seguimiento de los cambios en el código, facilitando la colaboración y la recuperación de versiones anteriores.

Función:

- **Git**: Herramienta que ayuda a gestionar versiones del código y a colaborar con otros desarrolladores.

Tema 10: Práctica Continua

- *Resuelve problemas en plataformas como* **LeetCode, HackerRank o Codewars.**
- Contribuye a proyectos de código abierto en GitHub.

Por qué se utiliza: La programación es una habilidad que se desarrolla con la práctica. Resolver problemas y trabajar en proyectos mejora tu comprensión y habilidades.

Función: Te ayuda a aplicar lo aprendido y a familiarizarte con diferentes tipos de desafíos y tecnologías.

Recursos Recomendados

- **Libros**: "Automate the Boring Stuff with Python" de Al Sweigart.
- **Cursos en línea**: Coursera, Udemy, Codecademy.
- **Documentación oficial**: Python.org.

Estructura de programación en python

1. Comentarios

- **Uso inicial**: Siempre es buena práctica comenzar con comentarios que expliquen el propósito del código.
- **Por qué**: Facilitan la comprensión del código para ti y para otros que lo lean más tarde.

2. Declaración de Variables

- **Uso**: Definir variables que serán utilizadas en el programa.
- **Por qué**: Almacenan datos que necesitarás a lo largo de tu código. Establecer estas variables al inicio ayuda a comprender qué información se manejará.

3. Funciones

- **Uso**: Definir funciones que encapsulan bloques de código reutilizables.
- **Por qué**: Organizar tu código en funciones mejora la legibilidad y la reutilización. También te permite dividir tareas complejas en partes más manejables.

4. Estructuras de Control

- **Uso**: Implementar condicionales (`if`, `else`) y bucles (`for`, `while`).
- **Por qué**: Permiten que tu programa tome decisiones y repita acciones basadas en ciertas condiciones. Esto es esencial para la lógica del programa.

5. Clases y Objetos

- **Uso**: Definir clases para modelar entidades del mundo real o conceptos en tu programa.
- **Por qué**: La programación orientada a objetos (OOP) ayuda a organizar y estructurar el código, especialmente en programas más complejos. Te permite crear objetos que pueden interactuar entre sí.

6. Módulos y Paquetes

- **Uso**: Importar módulos y bibliotecas que contengan funciones o clases que desees utilizar.
- **Por qué**: Las bibliotecas permiten aprovechar el trabajo de otros desarrolladores y agregar funcionalidades sin necesidad de reinventar la rueda.

7. Manejo de Excepciones

- **Uso**: Implementar bloques `try` y `except` para manejar posibles errores.
- **Por qué**: Proporciona robustez a tu código, permitiendo que el programa maneje errores de manera controlada en lugar de cerrarse abruptamente.

8. Entrada y Salida

- **Uso**: Obtener información del usuario y mostrar resultados.
- **Por qué**: La interacción con el usuario es fundamental. `input()` permite recibir datos, mientras que `print()` muestra la salida, facilitando la comunicación con el usuario.

9. Pruebas y Depuración

- **Uso**: Probar y depurar tu código, asegurando que funciona como se espera.
- **Por qué**: Es esencial verificar que tu código realiza las tareas correctamente y manejar cualquier posible error que no haya sido anticipado.

10. Ejecución Final

- **Uso**: Ejecutar el programa y observar su comportamiento en la práctica.
- **Por qué**: Después de haber escrito y probado el código, la ejecución es el último paso para ver si todo funciona como se espera.

python Copiar código

```python
# 1. Comentarios
# Este programa calcula la suma de dos números.

# 2. Declaración de Variables
num1 = 5
num2 = 10

# 3. Función
def sumar(a, b):
    return a + b

# 4. Estructura de Control
if num1 > 0 and num2 > 0:
    # 6. Llamada a Función
    resultado = sumar(num1, num2)
    # 8. Salida
    print(f"La suma de {num1} y {num2} es {resultado}.")
else:
    print("Por favor, ingrese números positivos.")

# 5. (Si fuese necesario) Definir clases y objetos aquí.

# 7. Manejo de Excepciones (en caso de entradas no válidas, por ejemplo)
try:
    num1 = int(input("Ingrese el primer número: "))
    num2 = int(input("Ingrese el segundo número: "))
except ValueError:
    print("Por favor, ingrese solo números.")
```

Ejemplo

1. Características Avanzadas del Lenguaje

Decoradores y Generadores:

Creación y uso de decoradores para modificar funciones o clases.

Utilización de generadores con yield para manejar secuencias de datos eficientemente.

Context Managers:

Uso de with para gestionar recursos.

Creación de context managers personalizados con __enter__, __exit__ o @contextmanager.

Metaprogramación y Metaclases:

Modificación de clases y objetos en tiempo de ejecución.

Creación de metaclases para controlar la creación de clases.

Programación Funcional:

Uso de funciones de orden superior, inmutabilidad y expresiones lambda.

Herramientas como functools e itertools.

2. Concurrencia y Paralelismo

Hilos y Procesos:

Uso de los módulos threading y multiprocessing para tareas concurrentes.

Programación Asíncrona:

Implementación con asyncio para operaciones de I/O eficientes.

3. Desarrollo y Calidad de Código

Testing y TDD:

Escritura de pruebas unitarias con unittest, pytest o nose.

Adopción del Desarrollo Basado en Pruebas (TDD).

Optimización y Perfilado:

Identificación de cuellos de botella con cProfile, timeit y line_profiler.

Técnicas para mejorar la eficiencia del código.

•Buenas Prácticas y Patrones de Diseño:

Principios SOLID, DRY, KISS.

Implementación de patrones como Singleton, Factory, Observer, etc.

Tipado Estático:

Uso de anotaciones de tipo y herramientas como mypy.

4. Desarrollo Web y APIs

Frameworks Avanzados:

Profundización en Django o Flask, incluyendo autenticación y optimización.

Desarrollo de APIs:

Diseño y construcción de APIs RESTful y GraphQL.

Uso de FastAPI para APIs rápidas y eficientes.

5. Ciencia de Datos y Machine Learning

Bibliotecas Específicas:

Dominio de NumPy, pandas, scikit-learn, TensorFlow, PyTorch.

Aplicaciones Avanzadas:

Análisis de datos, modelado predictivo y despliegue de modelos.

6. Gestión de Proyectos y Herramientas

Control de Versiones y CI/CD:

Configuración de pipelines con GitHub Actions, Jenkins o GitLab CI.

Automatización de pruebas y despliegues.

Entornos Virtuales y Dependencias:

Uso de venv, virtualenv, conda, pipenv o poetry.

Desarrollo de Paquetes:

Creación y distribución con setuptools y pip.

Publicación en PyPI.

7. Interoperabilidad y Extensiones

Integración con Otros Lenguajes:

Uso de Cython para compilar Python a C.

Integración con C/C++ para mejorar el rendimiento.

8. Seguridad y Manejo de Datos

Prácticas de Seguridad:

Gestión segura de contraseñas, encriptación y manejo de datos sensibles.

9. Bases de Datos Avanzadas

ORMs y Bases de Datos NoSQL:

Uso de SQLAlchemy o Django ORM.

Exploración de bases de datos como MongoDB.

10. Automatización y Scripting Avanzado

Automatización de Tareas:

Creación de scripts complejos para tareas repetitivas.

Uso de selenium para automatizar interacciones web o fabric para despliegues.

11. Contribución y Comunidad

Proyectos de Código Abierto:

Participación en proyectos comunitarios para ganar experiencia y colaborar.

Documentación y Comunicación:

Escritura de documentación clara con herramientas como Sphinx.

Participación en foros y eventos para mantenerte actualizado.

Recomendaciones Finales:

Proyectos Personales: Desarrolla proyectos complejos que integren múltiples áreas para consolidar tus conocimientos.

Participación Activa: Únete a comunidades y contribuye a proyectos de código abierto para enriquecer tu experiencia.

thanks for reading :)