Conceptos Básicos y Creación de Scripts Simples

PYTHON: Introducción a la Programación

INTRODUCCIÓN

Python es un lenguaje de programación de alto nivel, interpretado y de propósito general.

Fue creado por Guido van Rossum y lanzado por primera vez en 1991. Python se destaca por su sintaxis clara y legible, lo que lo hace ideal para principiantes

Ventajas de aprender Python

- Simplicidad: Fácil de aprender y leer.

- Versatilidad: Usado en desarrollo web, ciencia de datos, inteligencia artificial y más.

- Comunidad: Gran comunidad de usuarios y desarrolladores.

Objetivos del tutorial

En este tutorial, aprenderás los conceptos básicos de Python y cómo crear scripts simples que te ayudarán a comprender los fundamentos de la programación.

CAPÍTULO 1: INSTALACIÓN Y CONFIGURACIÓN

1. Instalación de Python

1. Visita la página oficial de Python: https://www.python.org/

2. Descarga la versión más reciente compatible con tu sistema operativo.

3. Sigue las instrucciones de instalación.

2. Configuración del Entorno de Desarrollo

- IDLE: El entorno de desarrollo integrado que viene con Python.

- Visual Studio Code (VS Code): Un editor de código popular. Se recomienda instalar la extensión de Python.

3. Primer Script "Hola Mundo"

```
print("¡Hola, Mundo!")
```

Guarda el archivo como hola_mundo.py y ejecútalo para ver el resultado.

CAPÍTULO 2: FUNDAMENTOS DE PYTHON

Sintaxis Básica

Python es conocido por su sintaxis simple y clara, lo que lo convierte en uno de los lenguajes de programación más accesibles para principiantes. Este capítulo cubrirá los fundamentos de la sintaxis de Python, incluyendo comentarios, variables, operadores y estructuras básicas de código.

Comentarios

Los comentarios en Python son líneas de texto que el intérprete ignora, utilizados para explicar el código. Hay dos tipos principales de comentarios:

- Comentarios de una línea: Se escriben con el símbolo #.

```
# Esto es un comentario de una línea
print("Hola, Mundo")  # También se puede usar al final de una línea de código
```

- Comentarios de múltiples líneas: Se utilizan comillas triples (''' o """).

```
"""
Esto es un comentario
de múltiples líneas
"""
```

Variables y Tipos de Datos

- Variables: Son espacios de memoria donde se almacenan valores. En Python, no es necesario declarar el tipo de variable explícitamente.

```
nombre = "Juan"  # Variable tipo string
edad = 25        # Variable tipo entero
altura = 1.75    # Variable tipo float
```

- Tipos de datos básicos:
  - Enteros (int): Números sin parte decimal.

```
numero = 10
```

  - Flotantes (float): Números con parte decimal.

```
precio = 19.99
```

  - Cadenas (str): Secuencias de caracteres.

```
texto = "Hola"
```

  - Booleanos (bool): Valores True o False.

```
es_mayor = True
```

- Operadores
  - Aritméticos:

    Suma (+): a + b

    Resta (-): a - b

    Multiplicación (*): a * b

    División (/): a / b

    Módulo (%): a % b (resto de la división)

    Potencia (**): a ** b

  - Comparación:

    Igual (==): a == b

    Diferente (!=): a != b

Mayor que (>): a > b

Menor que (<): a < b

Mayor o igual que (>=): a >= b

Menor o igual que (<=): a <= b

- Lógicos:

Y (and): True and False

O (or): True or False

No (not): not True

CAPÍTULO 3: FUNDAMENTOS DE PYTHON

Introducción

Las estructuras de control son fundamentales en la programación, ya que permiten dirigir el flujo de ejecución de un programa. En Python, estas estructuras incluyen condicionales y bucles, que se utilizan para tomar decisiones y repetir acciones. Este capítulo se centra en la implementación de estas estructuras y su uso en la creación de programas eficientes y dinámicos.

Condicionales

Las declaraciones condicionales permiten que un programa ejecute ciertas líneas de código solo si se cumple una condición específica. En Python, se utilizan las palabras clave if, elif y else para crear estructuras condicionales.

- Sintaxis básica del if:

  if condición:

  # código a ejecutar si la condición es verdadera

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

- Uso de elif para múltiples condiciones:

  if condición1:

  # código a ejecutar si condición1 es verdadera

  elif condición2:

  # código a ejecutar si condición2 es verdadera

  else:

  # código a ejecutar si ninguna condición anterior es verdadera

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

Bucles

- Bucle for: Se utiliza para iterar sobre una secuencia (como una lista, tupla o cadena).

  Sintaxis básica: **for** variable **in** secuencia (código a ejecutar en cada iteración)

```python
for i in range(5):
    print(i)
```

- Bucle while: Repite un bloque de código mientras una condición sea verdadera.

  Sintaxis básica: **while** condición (código a ejecutar mientras la condición sea verdadera)

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

Control de Bucles

- Declaración break: Termina el bucle prematuramente

```python
for i in range(10):
    if i == 5:
        break
    print(i)
```

Este bucle se detendrá cuando i sea igual a 5.

- Declaración continue: Salta a la siguiente iteración del bucle.

```
for i in range(10):
    if i % 2 == 0:
        continue
    print(i)
```

Este bucle imprimirá solo los números impares del 0 al 9.

- Declaración else en bucles: Se ejecuta si el bucle termina sin una interrupción (break)

```
for i in range(5):
    print(i)
else:
    print("Bucle completado")
```

Ejemplos Prácticos

Comprobador de Números Primos:

```
num = int(input("Introduce un número: "))
es_primo = True

if num < 2:
    es_primo = False
else:
    for i in range(2, num):
        if num % i == 0:
            es_primo = False
            break

if es_primo:
    print(f"{num} es un número primo")
else:
    print(f"{num} no es un número primo")
```

Contador de Vocales en una Cadena:

```python
cadena = input("Introduce una cadena: ")
vocales = "aeiouAEIOU"
contador = 0

for letra in cadena:
    if letra in vocales:
        contador += 1

print(f"Hay {contador} vocales en la cadena.")
```

Generador de Tablas de Multiplicar:

```python
numero = int(input("Introduce un número: "))

for i in range(1, 11):
    print(f"{numero} x {i} = {numero * i}")
```

CAPÍTULO 4: FUNCIONES Y MÓDULOS

Introducción

Las funciones y los módulos son pilares fundamentales en la programación con Python. Permiten estructurar el código de manera modular, facilitando la reutilización, la organización y el mantenimiento del mismo. Este capítulo explora cómo definir y utilizar funciones, así como la importación y uso de módulos en Python.

Funciones

Las funciones en Python son bloques de código reutilizables diseñados para realizar una tarea específica. Ayudan a organizar el código en piezas más manejables y facilitan la reutilización.

- Definición de Funciones:Las funciones se definen usando la palabra clave **def**, seguida del nombre de la función y paréntesis (). El bloque de código dentro de la función está indentado.

  def nombre_de_la_funcion(parametros):

  # Cuerpo de la función

  return resultado

```
def saludar():
    print("Hola, Mundo")

saludar()  # Llamada a la función
```

- Parámetros y Argumentos:Las funciones pueden aceptar parámetros, que son variables que se pasan a la función.

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

resultado = sumar(5, 3)
print(resultado)  # 8
```

- Valores por Defecto: Puedes definir valores por defecto para los parámetros. Si no se proporciona un argumento, se utiliza el valor por defecto.

```
def saludar(nombre="Mundo"):
    print(f"Hola, {nombre}")

saludar()          # Hola, Mundo
saludar("Juan")    # Hola, Juan
```

- Funciones Lambda: Son funciones anónimas definidas con la palabra clave lambda. Son útiles para operaciones simples.

```
suma = lambda a, b: a + b
print(suma(5, 3))  # 8
```

- Funciones Anidadas: Puedes definir funciones dentro de otras funciones.

```
def funcion_externa():
    print("Esta es la función externa")

    def funcion_interna():
        print("Esta es la función interna")

    funcion_interna()

funcion_externa()
```

- Funciones como Argumentos:Las funciones pueden ser pasadas como argumentos a otras funciones.

```
def aplicar_operacion(a, b, operacion):
    return operacion(a, b)

resultado = aplicar_operacion(5, 3, sumar)
print(resultado)  # 8
```

Módulos

Los módulos en Python son archivos que contienen definiciones y declaraciones de Python, como funciones, clases y variables. Facilitan la organización del código en partes reutilizables.

- Importación de Módulos: Puedes importar un módulo utilizando la palabra clave import.

```
import math
print(math.sqrt(16))  # 4.0
```

- Importación Específica: Puedes importar funciones o variables específicas de un módulo.

```
from math import sqrt, pi
print(sqrt(25))  # 5.0
print(pi)        # 3.1415926535897793
```

- Alias para Módulos:Puedes asignar un alias a un módulo para simplificar su uso

```
import numpy as np
array = np.array([1, 2, 3])
print(array)
```

- Creación de Módulos Propios:Puedes crear tus propios módulos guardando funciones y variables en un archivo .py y luego importándolos.
  - Archivo mimodulo.py:

```
def saludar(nombre):
    print(f"Hola, {nombre}")
```

  - Uso del módulo

```
import mimodulo
mimodulo.saludar("Ana")  # Hola, Ana
```

- Exploración de Módulos:Utiliza dir() para listar los atributos y funciones de un módulo.

```
import math
print(dir(math))
```

Ejemplos Prácticos

1. Calculadora con Funciones y Módulos:

   Archivo calculadora.py:

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

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

def multiplicar(a, b):
    return a * b

def dividir(a, b):
    if b != 0:
        return a / b
    else:
        return "División por cero no permitida"
```

Uso del módulo calculadora.py:

```
import calculadora

a = 10
b = 5
print(calculadora.sumar(a, b))          # 15
print(calculadora.restar(a, b))         # 5
print(calculadora.multiplicar(a, b))    # 50
print(calculadora.dividir(a, b))        # 2.0
```

2. Generador de Números Aleatorios:

```
import random

def generar_numero_aleatorio(inicio, fin):
    return random.randint(inicio, fin)

print(generar_numero_aleatorio(1, 100))
```

3. Uso de Módulo os para Manejo de Archivos:

```
import os

# Crear un nuevo directorio
os.mkdir("nuevo_directorio")

# Cambiar de directorio
os.chdir("nuevo_directorio")

# Mostrar el directorio actual
print(os.getcwd())
```

CAPÍTULO 5: MANEJO DE ARCHIVOS

Introducción

El manejo de archivos es una habilidad crucial en la programación, permitiendo a los desarrolladores leer, escribir y manipular archivos en el sistema operativo. Python proporciona una amplia variedad de funciones para interactuar con archivos de manera eficiente. Este capítulo se centrará en cómo abrir, leer, escribir y cerrar archivos, así como en el manejo de excepciones relacionadas con operaciones de archivos.

Apertura y Cierre de Archivos

Apertura de Archivos: La función **open**() se utiliza para abrir archivos. Puede aceptar diferentes modos de apertura, como lectura, escritura y adición.

```
archivo = open("nombre_del_archivo.txt", "modo")
```

- Modos comunes:

r: Lectura (predeterminado)

w: Escritura (crea un archivo nuevo o sobrescribe uno existente)

a: Añadir (escribe al final del archivo sin borrar el contenido existente)

b: Binario (añadir a los modos anteriores para archivos binarios)

- Ejemplos:

```
archivo_lectura = open("archivo.txt", "r")
archivo_escritura = open("archivo.txt", "w")
archivo_añadir = open("archivo.txt", "a")
```

- Cierre de Archivos: Es importante cerrar los archivos después de usarlos para liberar recursos del sistema.

```
archivo = open("archivo.txt", "r")
# Operaciones con el archivo
archivo.close()
```

- Uso de with para Manejar Archivos: El uso de with asegura que el archivo se cierre automáticamente después de que el bloque de código se ejecute.

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

Lectura de Archivos

- Leer Todo el Contenido:

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

- Leer Línea por Línea:

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

- Leer con readline():Lee una sola línea a la vez.

```
with open("archivo.txt", "r") as archivo:
    linea = archivo.readline()
    while linea:
        print(linea, end='')
        linea = archivo.readline()
```

- Leer Todas las Líneas en una Lista:

```
with open("archivo.txt", "r") as archivo:
    lineas = archivo.readlines()
    for linea in lineas:
        print(linea, end='')
```

Escritura de Archivos

- Escribir en un Archivo:

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

- Añadir Contenido a un Archivo:

```
with open("archivo.txt", "a") as archivo:
    archivo.write("Añadiendo esta línea al final del archivo.\n")
```

- Escribir Varias Líneas con writelines():

```
lineas = ["Primera línea\n", "Segunda línea\n", "Tercera línea\n"]
with open("archivo.txt", "w") as archivo:
    archivo.writelines(lineas)
```

Manejo de Archivos Binarios

- Lectura de Archivos Binarios

```
with open("imagen.png", "rb") as archivo:
    contenido_binario = archivo.read()
```

- Escritura en Archivos Binarios

```
with open("imagen_copia.png", "wb") as archivo:
    archivo.write(contenido_binario)
```

Manejo de Excepciones

Es esencial manejar excepciones para gestionar errores que puedan ocurrir durante las operaciones con archivos, como archivos no encontrados o errores de permisos.

- Uso de try y except:

```
try:
    with open("archivo_no_existe.txt", "r") as archivo:
        contenido = archivo.read()
except FileNotFoundError:
    print("El archivo no fue encontrado.")
```

- Manejo de Excepciones Múltiples:

```
try:
    with open("archivo.txt", "r") as archivo:
        contenido = archivo.read()
except FileNotFoundError:
    print("El archivo no existe.")
except PermissionError:
    print("No tienes permiso para leer el archivo.")
```

- Ejemplos Prácticos
    a. Copiar el Contenido de un Archivo a Otro:

```
with open("archivo_origen.txt", "r") as origen:
    with open("archivo_destino.txt", "w") as destino:
        for linea in origen:
            destino.write(linea)
```

b. Contador de Palabras en un Archivo:

```
def contar_palabras(archivo):
    with open(archivo, "r") as f:
        contenido = f.read()
        palabras = contenido.split()
        return len(palabras)

print(contar_palabras("archivo.txt"))
```

c. Buscar y Reemplazar Texto en un Archivo:

```
def buscar_reemplazar(archivo, buscar, reemplazar):
    with open(archivo, "r") as f:
        contenido = f.read()

    contenido = contenido.replace(buscar, reemplazar)

    with open(archivo, "w") as f:
        f.write(contenido)

buscar_reemplazar("archivo.txt", "viejo", "nuevo")
```

CAPÍTULO 6: PROYECTOS SIMPLES

Introducción

Los proyectos simples son una excelente manera de aplicar los conceptos aprendidos y consolidar tus habilidades en programación con Python. Este capítulo presenta tres proyectos prácticos: una calculadora básica, un generador de contraseñas y un contador de palabras en un archivo de texto. Cada proyecto incluye una explicación detallada y el código completo, proporcionando una base sólida para que puedas expandir y adaptar estos ejemplos a tus necesidades específicas.

Proyecto 1: Calculadora Básica

Este proyecto implica crear una calculadora que realice operaciones aritméticas básicas: suma, resta, multiplicación y división.

- Objetivos:
    - Implementar funciones para cada operación.
    - Manejar la entrada del usuario.
    - Validar entradas para evitar errores.
- Código del Proyecto:

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

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

def multiplicar(a, b):
    return a * b
```

```
def dividir(a, b):
    if b == 0:
        return "Error: División por cero"
    return a / b

def calculadora():
    print("Seleccione una operación:")
    print("1. Sumar")
    print("2. Restar")
    print("3. Multiplicar")
    print("4. Dividir")

    opcion = input("Ingrese su elección (1/2/3/4): ")

    num1 = float(input("Ingrese el primer número: "))
    num2 = float(input("Ingrese el segundo número: "))

    if opcion == '1':
        print(f"Resultado: {sumar(num1, num2)}")
    elif opcion == '2':
        print(f"Resultado: {restar(num1, num2)}")
    elif opcion == '3':
        print(f"Resultado: {multiplicar(num1, num2)}")
```

```
    elif opcion == '4':
        print(f"Resultado: {dividir(num1, num2)}")
    else:
        print("Opción no válida")

calculadora()
```

- Explicación:

  Se definen funciones para cada operación aritmética.

  La función calculadora() gestiona la interacción con el usuario y llama a la función correspondiente según la elección del usuario.

Proyecto 2: Generador de Contraseñas

Este proyecto genera una contraseña aleatoria con una longitud específica, combinando letras mayúsculas, minúsculas, dígitos y caracteres especiales.

- Objetivos:
  - Utilizar la biblioteca random.
  - Generar contraseñas seguras.
- Código del Proyecto:

```
import random
import string

def generar_contrasena(longitud):
    caracteres = string.ascii_letters + string.digits + string.punctuation
    contrasena = ''.join(random.choice(caracteres) for i in range(longitud))
    return contrasena
```

```
longitud = int(input("Introduce la longitud de la contraseña: "))
print(f"Contraseña generada: {generar_contrasena(longitud)}")
```

- Explicación:

  Se utiliza string.ascii_letters para letras, string.digits para dígitos y string.punctuation para caracteres especiales.

  La función generar_contrasena() crea una contraseña aleatoria de la longitud especificada.

Proyecto 3: Contador de Palabras

Este proyecto cuenta el número de palabras en un archivo de texto proporcionado por el usuario.

- Objetivos:
  - Leer archivos de texto.
  - Manipular cadenas de texto.
- Código del Proyecto:

```
def contar_palabras(archivo):
    try:
        with open(archivo, 'r') as f:
            contenido = f.read()
            palabras = contenido.split()
            return len(palabras)
    except FileNotFoundError:
        return "El archivo no fue encontrado."
    except Exception as e:
```

```
        return f"Error: {e}"


nombre_archivo = input("Introduce el nombre del archivo: ")

print(f"Número de palabras: {contar_palabras(nombre_archivo)}")
```

- Explicación:
    - Se abre el archivo en modo lectura y se lee todo el contenido.
    - split() se utiliza para dividir el texto en palabras.
    - Se maneja la excepción FileNotFoundError para casos en que el archivo no exista.