Basic Concepts and Creation of Simple Scripts

PYTHON: Introduction to Programming

INTRODUCTION

Python is a high-level, interpreted, general-purpose programming language. It was created by Guido van Rossum and first released in 1991. Python stands out for its clear and readable syntax, making it ideal for beginners.

Advantages of learning Python

- Simplicity: Easy to learn and read.
- Versatility: Used in web development, data science, artificial intelligence, and more.
- Community: Large community of users and developers.

Tutorial Objectives

In this tutorial, you will learn the basics of Python and how to create simple scripts that will help you understand the fundamentals of programming.

CHAPTER 1: INSTALLATION AND SETUP

1. Installation of Python

Visit the official Python website: python.org

Download the latest version compatible with your operating system.

Follow the installation instructions.

2. Development Environment Setup

- IDLE: The integrated development environment that comes with Python.
- Visual Studio Code (VS Code): A popular code editor. It is recommended to install the Python extension.

3. First Script "Hola Mundo"

```
print("¡Hola, Mundo!")
```

Save the file as hello_world.py and run it to see the result.

CHAPTER 2: PYTHON FUNDAMENTALS

Basic Syntax

Python is known for its simple and clear syntax, making it one of the most accessible programming languages for beginners. This chapter covers the basics of Python syntax, including comments, variables, operators, and basic code structures.

Comments

Comments in Python are lines of text ignored by the interpreter, used to explain the code. There are two main types of comments:

- Single-line comments: Written with the # symbol.

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

- Multi-line comments: Use triple quotes (''' or """).

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

Variables and Data Types

- Variables: Memory spaces where values are stored. In Python, it is not necessary to declare the variable type explicitly.

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

- Basic data types:
  - Integers (int): Numbers without a decimal part.

```
numero = 10
```

  - Floats (float): Numbers with a decimal part.

```
precio = 19.99
```

  - Strings (str): Sequences of characters.

```
texto = "Hola"
```

  - Booleans (bool): True or False values.

```
es_mayor = True
```

- Operators
  - Arithmetic:

    Addition (+): a + b

    Subtraction (-): a - b

    Multiplication (*): a * b

    Division (/): a / b

    Modulus (%): a % b (remainder of the division)

    Exponentiation (**): a ** b

  - Comparison:

    Equal (==): a == b

    Not equal (!=): a != b

Greater than (>): a > b

Less than (<): a < b

Greater or equal to (>=): a >= b

Less or equal to (<=): a <= b

- Logical:

  And (and): True and False

  Or (or): True or False

  Not (not): not True

CHAPTER 3: CONTROL STRUCTURES

Introduction

Control structures are fundamental in programming as they allow directing the flow of a program's execution. In Python, these structures include conditionals and loops, used to make decisions and repeat actions. This chapter focuses on implementing these structures and their use in creating efficient and dynamic programs.

Conditionals

Conditional statements allow a program to execute certain lines of code only if a specific condition is met. In Python, the if, elif, and else keywords are used to create conditional structures.

- Basic if syntax:

  if condición:

  # code to execute if the condition is true

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

- Using elif for multiple conditions:

  if condición1:

  # code to execute if condition1 is true

  elif condición2:

  # code to execute if condition2 is true

  else:

  # code to execute if none of the previous conditions are true

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

Loops

- **for** Loop: Used to iterate over a sequence (like a list, tuple, or string)

  Basic syntax: **for** variable **in** sequence (code to execute in each iteration)

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

- while Loop: Repeats a block of code while a condition is true.

  Basic syntax: **while** condition (code to execute while the condition is true)

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

Loop Control

- break Statement: Terminates the loop prematurely

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

This loop will stop when i equals 5.

- continue Statement: Skips to the next iteration of the loop.

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

This loop will print only the odd numbers from 0 to 9.

- **else Statement in loops: Executes if the loop terminates without a break

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

Example

prime number checker:

```
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")
```

Vowel Counter in a String:

```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.")
```

Multiplication Tables Generator:

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

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

CHAPTER 4: FUNCTIONS AND MODULES

Introduction

Functions and modules are fundamental pillars in programming with Python. They allow structuring code in a modular way, facilitating reuse, organization, and maintenance. This chapter explores how to define and use functions, as well as the import and use of modules in Python.

Functions

Functions in Python are reusable blocks of code designed to perform a specific task. They help organize code into more manageable pieces and facilitate reuse.

- Definition of Functions: Functions are defined using the def keyword, followed by the function name and parentheses (). The block of code inside the function is indented.

  def function_name(parameters):

  # Function body

  return result

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

saludar()  # Llamada a la función
```

- Parameters and Arguments: Functions can accept parameters, which are variables passed to the function.

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

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

- Default Values: You can define default values for parameters. If no argument is provided, the default value is used.

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

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

- Lambda Functions: These are anonymous functions defined with the lambda keyword. They are useful for simple operations.

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

- Nested Functions: You can define functions within other functions.

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

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

    funcion_interna()

funcion_externa()
```

- Functions as Arguments: Functions can be passed as arguments to other functions.

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

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

Modules

Modules in Python are files containing Python definitions and statements, such as functions, classes, and variables. They facilitate organizing code into reusable parts.

- Importing Modules: You can import a module using the import keyword.

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

- Specific Import: You can import specific functions or variables from a module.

```
from math import sqrt, pi
print(sqrt(25))  # 5.0
print(pi)        # 3.141592653589793
```

- Alias for Modules: You can assign an alias to a module to simplify its usage.

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

- Creating Your Own Modules: You can create your own modules by saving functions and variables in a .py file and then importing them.
-
    - File mimodulo.py:

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

    - Using the module:

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

- Exploring Modules: Use dir() to list the attributes and functions of a module.

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

Examples

1. Calculator with Functions and Modules:

   File 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"
```

Module use 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. Random number generator:

```
import random

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

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

3. Using the os Module for File Management:

```
import os

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

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

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

CHAPTER 5: FILE HANDLING

Introduction

File handling is a crucial skill in programming, allowing developers to read, write, and manipulate files in the operating system. Python provides a wide variety of functions to interact with files efficiently. This chapter will focus on how to open, read, write, and close files, as well as handle exceptions related to file operations.

Opening and Closing Files

Opening Files: The open() function is used to open files. It can accept different modes of opening, such as reading, writing, and appending.

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

- Basic syntax:

r: Read (default)

w: Write (creates a new file or overwrites an existing one)

a: Append (writes at the end of the file without deleting existing content)

b: Binary (add to the previous modes for binary files)

- Examples:

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

- Closing Files: It is important to close files after using them to free up system resources.

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

- Using with to Manage Files: Using with ensures the file is closed automatically after the code block is executed.

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

Reading Files

- Read All Content:

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

- Read Line by Line:

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

- Read with readline(): Reads one line at a time..

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

- Read All Lines into a List:

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

Writing to Files

- Write to a File:

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

- Append Content to a File:

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

- Write Multiple Lines with writelines():

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

Handling Binary Files

- Reading Binary Files

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

- Writing to Binary Files:

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

Exception Handling

It is essential to handle exceptions to manage errors that may occur during file operations, such as file not found or permission errors.

- Using try and except:

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

- Handling Multiple Exceptions:

```
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.")
```

- Practical Examples
    a. Copy File Content to Another File:

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

b. Word Counter in a File:

```python
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. Find and Replace Text in a File:

```python
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")
```

CHAPTER 6: SIMPLE PROJECTS

Introduction

Simple projects are an excellent way to apply learned concepts and consolidate your programming skills with Python. This chapter presents three practical projects: a basic calculator, a password generator, and a word counter in a text file. Each project includes a detailed explanation and complete code, providing a solid foundation for you to expand and adapt these examples to your specific needs.

Project 1: Basic Calculator

This project involves creating a calculator that performs basic arithmetic operations: addition, subtraction, multiplication, and division.

- Objectives:
    - Implement functions for each operation.
    - Handle user input.
    - Validate inputs to avoid errors.
- Project Code:

```
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()
```

- Explanation:

  Functions are defined for each arithmetic operation.

  The calculator() function manages user interaction and calls the corresponding function based on the user's choice.

Project 2: Password Generator

This project generates a random password with a specific length, combining uppercase letters, lowercase letters, digits, and special characters.

- Objetives:
  - Use the random library..
  - Generate secure passwords.
- Project Code:

```
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)}")
```

- Explanation:

  string.ascii_letters is used for letters, string.digits for digits, and string.punctuation for special characters.

  The generate_password() function creates a random password of the specified length.

Project 3: Word Counter

This project counts the number of words in a text file provided by the user.

- Objectives:
  - Read text files.
  - Manipulate text strings.
- Project Code:

```
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)}")
```

- Explanation:
    - The file is opened in read mode, and the entire content is read.
    - split() is used to divide the text into words