**WEEK 2**

**PYTHON INTERMEDIATE TOPICS**

***Selected Intermediate Topics:***

Python Classes and Object-Oriented Programming (OOP)

File Handling and Input/Output (I/O)

Exception Handling1) Python Classes and Object-Oriented Programming (OOP)

File Handling and Input/Output (I/O)

Exception Handling

***1) Python Classes and Object-Oriented Programming (OOP)***

**Key Points:**

Classes in Python are blueprints for creating objects, which encapsulate data (attributes) and behavior (methods).

Inheritance allows a class to inherit attributes and methods from another class, enabling code reuse and hierarchical organization.

Encapsulation ensures that the internal representation of an object is hidden from the outside, promoting modularity and reusability.

Polymorphism allows objects to be treated as instances of their parent class, enabling flexibility in method invocation.

**Code Examples:**

# Define a simple class

class Person:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

def greet(self):

return f"Hello, my name is {self.name} and I'm {self.age} years old."

# Create an instance of the Person class

person1 = Person("Alice", 30)

# Access attributes and call methods

print(person1.name) # Output: Alice

print(person1.greet()) # Output: Hello, my name is Alice and I'm 30 years old.

1. **File Handling and Input/Output (I/O):**

**Key Points:**

Python provides built-in functions (open(), read(), write(), close()) for file handling.

Use the open() function to open a file in various modes (e.g., read mode 'r', write mode 'w', append mode 'a').

Context managers (with statement) ensure proper resource management and automatically close files after use.

File objects support methods like read(), write(), readline(), readlines(), and seek() for file manipulation.

Code Examples:

# Writing to a file

with open("example.txt", "w") as f:

f.write("Hello, world!\n")

f.write("This is a test.\n")

# Reading from a file

with open("example.txt", "r") as f:

contents = f.read()

print(contents)

1. **Exception Handling:**

***Key Points:***

Exceptions are events that occur during the execution of a program that disrupts the normal flow of instructions.

Use try, except, else, and finally blocks to handle exceptions gracefully and provide fallback mechanisms.

Catch specific exception types or use a generic except block to handle all exceptions.

The else block is executed if no exceptions occur within the try block, while the finally block is always executed, regardless of whether an exception occurs.

Code Examples:

# Exception handling example

try:

num = int(input("Enter a number: "))

result = 10 / num

except ValueError:

print("Please enter a valid number.")

except ZeroDivisionError:

print("Cannot divide by zero.")

else:

print("Result:", result)

finally:

print("End of program.")

# Exception handling example

try:

num = int(input("Enter a number: "))

result = 10 / num

except ValueError:

print("Please enter a valid number.")

except ZeroDivisionError:

print("Cannot divide by zero.")

else:

print("Result:", result)

finally:

print("End of program.")

These topics provide a deeper understanding of fundamental concepts in Python programming, including object-oriented programming, file handling, and exception handling. Understanding these topics is essential for building robust and maintainable Python applications.