**ROBOTICS & ARTIFICIAL INTELLIGENCE DEPARTMENT**

# Total Marks: 04

**Obtained Marks:**

Programming for Artificial Intelligence

**Assignment # 01**

**Last date of Submission: 3rd March 2024**

# Submitted To: Mr. Saad Irfan Khan

**Student Name: Adin Abbas**

# Reg. Number:

**ROBOTICS & ARTIFICIAL INTELLIGENCE DEPARTMENT**

***Instructions****: Copied or shown assignments will be marked zero. Late submissions are not entertained in any case.*

**CLO 1 – PLO A, E – C2**

**Question 01.** **(4 Marks)**

Consider yourself working as a developer for a large data analytics firm and you have been tasked with developing a Python program for analyzing sales data in some retail company. To complete this task, you are first required to:

1. Explain how Python's basic constructs like loops, if-else statements, and functions can be used to manipulate and analyze the data effectively.

Loops

Data iteration: Loops let you handle each element of a dataset effectively. A for loop, for instance, can iterate through a list of numbers and determine the maximum or average value.

If-then clauses:

Making decisions based on data: Conditional statements let you evaluate data and take particular actions in response to the findings. If a value satisfies a set of requirements, an if-else statement can determine this and carry out any necessary calculations or manipulations.

Uses:

Modularity and reusability: Functions let you write reusable code segments that carry out particular data manipulation operations. This encourages efficiency and organization in the code, particularly when working with intricate data analysis procedures.

1. Discuss the role of object-oriented programming concepts such as classes, inheritance, and encapsulation in organizing and managing the data processing tasks.

1. Classes:

Data Blueprints: Classes act as blueprints for creating objects that represent real-world entities or functionalities related to data processing. A class defines the attributes (data) and methods (functions) that objects of that class will possess.

Example:

Create a DataProcessor class with attributes like data (to hold the data) and methods like calculate\_average() and clean\_data().

2. Inheritance:

Code Reusability: Inheritance allows you to create new classes (subclasses) that inherit attributes and methods from existing classes (superclasses). This promotes code reuse, reduces redundancy, and allows you to build upon existing functionalities for specific data processing tasks.

Example:

Create a StatisticsCalculator class that inherits from DataProcessor. It can inherit the data attribute and add methods like calculate\_standard\_deviation() specific to statistical analysis.

3. Encapsulation:

Data Protection and Modularity: Encapsulation promotes data protection and code modularity. It allows you to control access to an object's attributes by defining methods that manage how data is accessed and modified. This ensures data integrity and prevents unintended modifications.

Example:

In the DataProcessor class, make the data attribute private and provide public methods like get\_data() and set\_data() to control access and modification of the data.

Benefits of using OOP for Data Processing:

Organization and Maintainability: By using classes and inheritance, you can organize data processing tasks into well-defined modules, making code easier to understand, maintain, and modify.

Extensibility: Inheritance allows you to easily extend functionalities by creating subclasses with specialized methods for specific data processing needs.

Data Reusability: Encapsulation promotes data security and allows you to reuse data structures and processing methods across different parts of your program.

1. Explore how advanced Python topics like generators and decorators can enhance the efficiency of the program.

Generators:

Generators are special functions that return an iterator object. Unlike regular functions that return a single value, generators yield multiple values one at a time. This allows them to be more memory-efficient when dealing with large datasets. They "pause" execution between each yield statement, only calculating the next value when requested.

Decorators:

Decorators are functions that modify the behavior of other functions. They are applied using the @ symbol before the function definition. Decorators can add functionalities like logging, error handling, timing, or authentication to existing functions without altering their core logic. This promotes code reusability and reduces redundancy.

1. Provide a brief outline of how you would structure and implement such a program.

Code:

import pandas as pd

# Function to load data from CSV file (replace 'data.csv' with your actual file path)

def load\_sales\_data(filepath):

try:

data = pd.read\_csv(filepath)

return data

except FileNotFoundError:

print("Error: File not found!")

return None

# Function to calculate total sales by product category (assuming a 'category' column)

def analyze\_sales\_by\_category(data):

if data is not None:

category\_totals = data.groupby('category')['sales\_amount'].sum()

print("Total Sales by Category:")

print(category\_totals)

# Load data from CSV

sales\_data = load\_sales\_data('data.csv')

# Analyze data if loading was successful

if sales\_data is not None:

analyze\_sales\_by\_category(sales\_data)

data = [5, 12, 8, 19, 2]

total = 0

for element in data:

total += element

average = total / len(data)

print(f"The average of the data is: {average}")

data = [10, 25, 18, 32, 7]

for element in data:

if element > 20:

print(f"{element} is greater than 20")

else:

print(f"{element} is less than or equal to 20")

def calculate\_standard\_deviation(data):

# Function to calculate standard deviation

# ... (implementation details)

return standard\_deviation

data = [8, 14, 5, 10, 17]

std\_dev = calculate\_standard\_deviation(data)

print(f"The standard deviation of the data is: {std\_dev}")