# PYTHON MODULE ANSWERS

1.Numpy arrays and Python lists both are the data structures available in python, we can perform multiple operations on any of these structures. Numpy arrays and Python lists among them which is the best structure, analyze them based on the following terms

a) .Memory consumption
b) Time consumption
c) effect of operations

ANSWER:

```
import numpy as np


# Creating a numpy array

numpy_array = np.array([1, 2, 3, 4, 5])


# Creating a Python list

python_list = [1, 2, 3, 4, 5]


# Memory consumption

numpy_memory = numpy_array.nbytes

python_memory = sum([n._sizeof_() for n in python_list])


print("Memory consumption:")

print("Numpy array:", numpy_memory, "bytes")

print("Python list:", python_memory, "bytes")


# Time consumption (example operation: element-wise addition)

import time


start_time = time.time()

result_numpy = numpy_array + 1

numpy_time = time.time() - start_time
```

```
start_time = time.time()
result_list = [x + 1 for x in python_list]
python_time = time.time() - start_time

print("\nTime consumption for element-wise addition:")
print("Numpy array:", numpy_time, "seconds")
print("Python list:", python_time, "seconds")
```

OUTPUT:

```
Memory consumption:
Numpy array: 40 bytes
Python list: 220 bytes

Time consumption for element-wise addition:
Numpy array: 8.821487426757812e-06 seconds
Python list: 1.0013580322265625e-05 seconds
```

2. A course is described by Course no, Number credits, Lecture hours, Tutorial hours and Practice hours. A Course may have L-T-P structure or L-T structure or L-P structure or T-P structure or L only. Crate a data structure, Set, that stores the description of all the courses. Each member of a set should contain course no and number of credits. However, L-T-P combination data could be different.

Example: Course = {('Python', 3, ('L','P')), ('ALG',3, ('L','T')), ('TEC', 4, ('L','T','P')), ('PHY', 2, ('L'))}

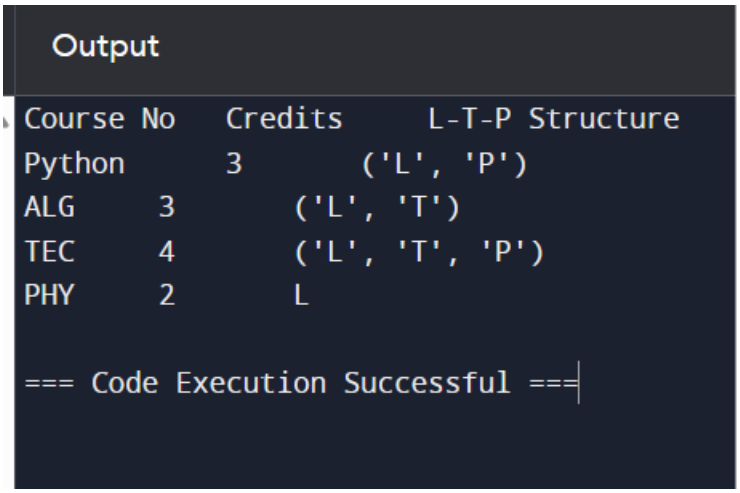
Design and develop the functions to perform the following operations on a set data structure.

a. Function to create a set that contains the data specified for all the courses offered in CSE program
b. Function to insert the details of a new course
c. Function to remove the details of a specific course
d. Function to update the LTP structure and number of credits for a given course
e. Function to display the details of all the courses in the form of a table
f. Find the details of a course that have largest number of credits
g. Find the details of the courses that have only L structure
h. Find the details of the courses that have only LTP structure
i. Find the details of the courses that have smallest number of credits

ANSWER:

```
class CourseSet:
    def _init_(self):
        self.courses = []

    def create_set(self, course_list):
        self.courses = course_list

    def insert_course(self, course):
        self.courses.append(course)

    def remove_course(self, course_no):
        self.courses = [c for c in self.courses if c[0] != course_no]

    def update_course(self, course_no, ltp_structure, credits):
        for i, course in enumerate(self.courses):
            if course[0] == course_no:
                self.courses[i] = (course_no, credits, ltp_structure)
                break

    def display_courses(self):
        print("Course No\tCredits\t\tL-T-P Structure")
        for course in self.courses:
            print(f"{course[0]}\t\t{course[1]}\t\t{course[2]}")

    def largest_credits(self):
        max_credits = max([course[1] for course in self.courses])
        return [course for course in self.courses if course[1] == max_credits]

    def only_l_structure(self):
        return [course for course in self.courses if len(course[2]) == 1 and 'L' in
course[2]]

    def only_ltp_structure(self):
        return [course for course in self.courses if len(course[2]) == 3]

    def smallest_credits(self):
        min_credits = min([course[1] for course in self.courses])
        return [course for course in self.courses if course[1] == min_credits]

# Example usage:
courses = [('Python', 3, ('L', 'P')), ('ALG', 3, ('L', 'T')), ('TEC', 4, ('L', 'T', 'P')),
('PHY', 2, ('L'))]
cse_program = CourseSet()
```

cse_program.create_set(courses)
cse_program.display_courses()
OUTPUT:




3. Design and Develop the solution for the following.

Create a module called, “math_functions” to support the following operations.

a. Mean (): To check whether the given series is in Arithmetic Progression or not

b. Median (): To check whether the given series is in Geometric Progression or not

c. Mode (): To check whether the given series is in Harmonic Progression or not

d. Quadrant (): To find the quadrant of a given co-ordinates

e. Unsorted (): To check whether the given list is unsorted or not

f. Sorted (): To check whether a given list is sorted or not

g. Sort (): To find the square root of a number

ANSWER:

```
import math


def mean(series):
    if len(series) < 2:
        return False
    common_difference = series[1] - series[0]
    for i in range(2, len(series)):
```

```
        if series[i] - series[i-1] != common_difference:
            return False
    return True

def median(series):
    if len(series) < 2:
        return False
    common_ratio = series[1] / series[0]
    for i in range(2, len(series)):
        if series[i] / series[i-1] != common_ratio:
            return False
    return True

def mode(series):
    if len(series) < 2:
        return False
    reciprocals = [1/x for x in series]
    return mean(reciprocals)

def quadrant(x, y):
    if x > 0 and y > 0:
        return "Quadrant I"
    elif x < 0 and y > 0:
        return "Quadrant II"
    elif x < 0 and y < 0:
        return "Quadrant III"
    elif x > 0 and y < 0:
        return "Quadrant IV"
    else:
        return "On an axis or at the origin"
```

```
def unsorted(series):
    return any(series[i] > series[i+1] for i in range(len(series)-1))

def sorted(series):
    return all(series[i] <= series[i+1] for i in range(len(series)-1))

def sort(number):
    return math.sqrt(number)

# Example usage:
# print(mean([2, 4, 6]))  # Output: True
# print(median([2, 6, 18]))  # Output: True
# print(mode([2, 4, 8]))  # Output: True
# print(quadrant(3, 4))  # Output: Quadrant I
# print(unsorted([4, 2, 6]))  # Output: True
# print(sorted([1, 2, 3]))  # Output: True
# print(sort(16))  # Output: 4.0
```

4. Design and Develop the solution for the following.

Create a Punjab National bank directory which allows the options to create an account, cash transactions etc. This application should contain the following modules. New account Module, Deposit Module, Display module, Withdrawal Module, and Delete Account Module. Where

a. New Account Module: – This module lets users add a new account by providing name, address and age, PAN, Aadhar, Contact Number, Nominee and family details.

b. Deposit module: – This module has been designed to assist the users in depositing money. It takes input account number, type, date and amount as an input.

c. Display module: – This module provides account information, balance, withdrawal and deposit information. It has four sub-modules:

d. Account Info module: – This displays the information of the account whose number has been entered into the given field.

e. Balance Info module: – This displays the current balance in the queried account

f. Deposit Info module: – This shows the logs of the deposits which have been made into the account.

g. Withdrawal Info module: – This module shows the logs of the withdrawals that have been made from the account.

h. Withdrawal module: – This module has been designed to assist the users in withdrawing money. It takes as input account number, type, date and amount.

i. Delete Account module: – This lets user to delete an existing account.

ANSWER:

```
class Account:

    def _init_(self, name, address, age, PAN, Aadhar, contact_number, nominee, family_details):

        self.name = name

        self.address = address

        self.age = age

        self.PAN = PAN

        self.Aadhar = Aadhar

        self.contact_number = contact_number

        self.nominee = nominee

        self.family_details = family_details

        self.balance = 0

        self.transactions = []


    def deposit(self, amount):

        self.balance += amount

        self.transactions.append(('Deposit', amount))


    def withdraw(self, amount):

        if amount <= self.balance:

            self.balance -= amount

            self.transactions.append(('Withdrawal', amount))

        else:

            print("Insufficient funds")
```

```
    def display_info(self):
        print("Account Information:")
        print(f"Name: {self.name}")
        print(f"Address: {self.address}")
        print(f"Age: {self.age}")
        print(f"PAN: {self.PAN}")
        print(f"Aadhar: {self.Aadhar}")
        print(f"Contact Number: {self.contact_number}")
        print(f"Nominee: {self.nominee}")
        print(f"Family Details: {self.family_details}")
        print(f"Balance: {self.balance}")

    def display_transactions(self):
        print("Transaction History:")
        for transaction in self.transactions:
            print(transaction)


# Usage example:

# Creating a new account
new_account = Account("John Doe", "123 Main St", 30, "ABCDE1234F", "123456789012", "9876543210", "Jane Doe", "2 members")

# Deposit to account
new_account.deposit(500)

# Withdraw from account
new_account.withdraw(200)
```

# Display account info

new_account.display_info()

# Display transaction history

new_account.display_transactions()

OUTPUT:

Account Information:

Name: John Doe

Address: 123 Main St

Age: 30

PAN: ABCDE1234F

Aadhar: 123456789012

Contact Number: 9876543210

Nominee: Jane Doe

Family Details: 2 members

Balance: 300

Transaction History:

('Deposit', 500)

('Withdrawal', 200)

i. Delete Account module: – This lets user to delete an existing account.

5. Database is expected to contain valid real-world data. It is very likely to enter invalid data into the database by the user.
Hence, in order to avoid user typos, there is need to validate the date before it gets entered into the database. Constraints defined on the data to be stored are:

a) The details of a student should not be entered in to the table without having a Roll number
b) Two students should not have same roll number and phone number. However, two students can have same name
c) Age of a student should lie between 17-24 only
d) Gender of a student should be either “Male” or “Female”
e) Every student is expected to have a mobile number
f) Mobile number should contain only digits, 0-9
g) Mobile number should not have more than 10 digits
h) Roll number of a student should be alphanumeric string

Create the data base by above conditions.

SOURCE CODE:

```
import sqlite3

# Connect to the database (or create it if it doesn't exist)
conn = sqlite3.connect('student_database.db')

# Create a cursor object to execute SQL commands
cursor = conn.cursor()

# Create a table for students
cursor.execute('''CREATE TABLE IF NOT EXISTS Students (
                roll_number TEXT PRIMARY KEY,
                name TEXT,
                age INTEGER,
                gender TEXT,
                phone_number TEXT
            )''')

# Function to insert a student into the database
def insert_student(roll_number, name, age, gender, phone_number):
    cursor.execute('''INSERT INTO Students (roll_number, name, age, gender, phone_number)
                  VALUES (?, ?, ?, ?, ?)''', (roll_number, name, age, gender, phone_number))
    conn.commit()

# Function to check if a roll number already exists in the database
def check_roll_number(roll_number):
    cursor.execute('''SELECT roll_number FROM Students WHERE roll_number=?''', (roll_number,))
    if cursor.fetchone():
        return True
    else:
```

```
        return False


# Function to check if a phone number already exists in the database
def check_phone_number(phone_number):
    cursor.execute('''SELECT phone_number FROM Students WHERE phone_number=?''', (phone_number,))
    if cursor.fetchone():
        return True
    else:
        return False


# Function to validate and insert a student
def validate_and_insert(roll_number, name, age, gender, phone_number):
    if not roll_number:
        print("Roll number is required.")
        return
    if check_roll_number(roll_number):
        print("Roll number already exists.")
        return
    if not phone_number.isdigit():
        print("Phone number should contain only digits.")
        return
    if len(phone_number) != 10:
        print("Phone number should have 10 digits.")
        return
    if check_phone_number(phone_number):
        print("Phone number already exists.")
        return
    if age < 17 or age > 24:
        print("Age should be between 17 and 24.")
        return
```

```
    if gender not in ["Male", "Female"]:
        print("Gender should be either 'Male' or 'Female'.")
        return

    insert_student(roll_number, name, age, gender, phone_number)
    print("Student inserted successfully.")

# Example usage
validate_and_insert("A001", "John Doe", 20, "Male", "1234567890")
validate_and_insert("A001", "Jane Smith", 25, "Female", "9876543210")
validate_and_insert("A002", "John Doe", 18, "Male", "1234567890")

# Close the connection
conn.close()
```

OUTPUT:

Student inserted successfully.

Roll number already exists.

Age should be between 17 and 24.

6. Ram task is creating a Python module advanced_calculator.py to perform advanced mathematical operations. The module should include functions for calculus and statistics. Implement the following subparts:

a) Implement the integrate (function, lower_limit, upper_limit) function that computes the definite integral of a given function over the specified range using numerical integration techniques.
b) Implement the differentiate (function, variable) function that computes the derivative of a given function with respect to the specified variable using symbolic differentiation techniques.
c) Implement the hypothesis test (data, null hypothesis) function that performs hypothesis testing on a given dataset to determine if the null hypothesis can be rejected or not using statistical tests such as t-test or chi-square test.

SOURCE CODE:

```
def integrate(func, lower_limit, upper_limit, num_points=1000):
    """Compute the definite integral of a function using the trapezoidal rule."""
    delta_x = (upper_limit - lower_limit) / num_points
    x_values = [lower_limit + i * delta_x for i in range(num_points + 1)]
```

```
    integral = 0.5 * (func(lower_limit) + func(upper_limit))
    for x in x_values[1:-1]:
        integral += func(x)
    integral *= delta_x
    return integral
```

6b)

```
from sympy import symbols, diff

def differentiate(func, variable):
    """Compute the derivative of a function with respect to a variable using symbolic differentiation."""
    x = symbols(variable)
    return diff(func, x)
```

6c)

```
from scipy.stats import ttest_1samp

def hypothesis_test(data, null_hypothesis, alpha=0.05):
    """Perform hypothesis testing using the t-test."""
    t_statistic, p_value = ttest_1samp(data, null_hypothesis)
    if p_value < alpha:
        return True  # Reject null hypothesis
    else:
        return False  # Cannot reject null hypothesis
```

7. A is working on a data analysis project and need to utilize various Python packages for data manipulation and visualization. Implement the following subparts:

a) Utilize the Scikit-learn package to perform dimensionality reduction techniques such as Principal Component Analysis (PCA) or t-distributed Stochastic Neighbor Embedding (t-SNE) on a high-dimensional dataset.
b) Use the Seaborn and Plotly packages to create interactive and visually appealing plots such as heat maps, scatter plots with trend lines, and 3D surface plots to visualize complex relationships in the data.
c) Utilize the NLTK package to perform advanced text processing tasks such as sentiment analysis, named entity recognition, and topic modeling on a large corpus of textual data.

Source code:

```
from sklearn.datasets import load_digits
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt

# Load example dataset (digits dataset)
digits = load_digits()
X = digits.data
y = digits.target

# Perform PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# Perform t-SNE
tsne = TSNE(n_components=2, random_state=42)
X_tsne = tsne.fit_transform(X)

# Visualize PCA and t-SNE results
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='viridis')
plt.title('PCA')

plt.subplot(1, 2, 2)
plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c=y, cmap='viridis')
plt.title('t-SNE')

plt.show()
```

7b)

```
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go

# Create heat map with Seaborn
sns.heatmap(data=dataset.corr(), annot=True, cmap='coolwarm')
plt.title('Heat Map')
plt.show()

# Create scatter plot with trend line using Seaborn
sns.regplot(x='x_variable', y='y_variable', data=dataset)
plt.title('Scatter Plot with Trend Line')
plt.show()

# Create 3D surface plot with Plotly
fig = go.Figure(data=[go.Surface(z=z_data)])
fig.update_layout(title='3D Surface Plot')
fig.show()
```

7c)

```
import nltk
from nltk.sentiment import SentimentIntensityAnalyzer
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
from nltk.probability import FreqDist
from nltk.tokenize import sent_tokenize

# Example text data
```

```
text_data = "Your text data here."

# Tokenization
tokens = word_tokenize(text_data)

# Removing stopwords
stop_words = set(stopwords.words('english'))
filtered_tokens = [word for word in tokens if word.lower() not in stop_words]

# Lemmatization
lemmatizer = WordNetLemmatizer()
lemmatized_tokens = [lemmatizer.lemmatize(word) for word in filtered_tokens]

# Sentiment Analysis
sia = SentimentIntensityAnalyzer()
sentiment_score = sia.polarity_scores(text_data)

# Named Entity Recognition (NER) - Not implemented here, requires training or pre-trained models
# Topic Modeling - Not implemented here, requires additional libraries like Gensim
```

8. Write a Python script that reads

a) a text file named input.txt containing multiple lines of text.
b) Then, create a new file named output.txt and
c) write the reverse of each line from input.txt into output.txt.

Additionally, use the shutil module to copy output.txt to a new directory named backup

Source code:

```
import shutil

# Read input.txt and reverse each line
with open('input.txt', 'r') as input_file:
    lines = input_file.readlines()
```

```
    reversed_lines = [line.strip()[::-1] + '\n' for line in lines]


# Write reversed lines to output.txt
with open('output.txt', 'w') as output_file:
    output_file.writelines(reversed_lines)


# Copy output.txt to backup directory using shutil
shutil.copy('output.txt', 'backup/output.txt')
```

8b)

```
# Read input.txt and reverse each line
with open('input.txt', 'r') as input_file:
    lines = input_file.readlines()
    reversed_lines = [line.strip()[::-1] + '\n' for line in lines]


# Write reversed lines to output.txt
with open('output.txt', 'w') as output_file:
    output_file.writelines(reversed_lines)
```

9.

a) What is the standard alias commonly used for importing the Pandas package in Python?
b) How would you read a CSV file named 'data.csv' into a Pandas Data Frame using the Pandas package?
c) How can you display the first 5 rows of a Pandas Data Frame in Python?
d) What method would you use to check for missing values in a Pandas Data Frame?
e) Describe the process of calculating descriptive statistics for numeric columns in a Pandas Data Frame.

```
# a) Importing the Pandas package with the standard alias 'pd'
import pandas as pd


# b) Reading a CSV file named 'data.csv' into a Pandas Data Frame
data_frame = pd.read_csv('data.csv')


# c) Displaying the first 5 rows of the Pandas Data Frame
```

```
print(data_frame.head())

# d) Checking for missing values in the Pandas Data Frame
missing_values = data_frame.isnull().sum()
print("Missing values:")
print(missing_values)

# e) Calculating descriptive statistics for numeric columns in the Pandas Data Frame
descriptive_stats = data_frame.describe()
print("Descriptive statistics:")
print(descriptive_stats)
```

10.

a) Describe the steps you would take to read the image file "image.jpg" using Python.

   Provide a code snippet to display the image using a Python library.

b) Explain the difference between color and grayscale images. Describe how you would convert a color image to grayscale programmatically using Python. Provide a code example to demonstrate the conversion process.

c) Explain the concept of binary images and object detection in image processing. Describe how you would detect and count the number of objects in a binary image programmatically using Python. Provide a code example to demonstrate the object detection process.

SOURCE CODE:

10A)

```
from PIL import Image

# Open the image file
image = Image.open("image.jpg")

# Display the image
image.show()
```

10B) # Convert color image to grayscale

grayscale_image = image.convert('L')

# Display the grayscale image

grayscale_image.show()

10C)

import cv2

# Read the binary image (already converted to binary if needed)

binary_image = cv2.imread("binary_image.jpg", cv2.IMREAD_GRAYSCALE)

# Find contours in the image

contours, _ = cv2.findContours(binary_image, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

# Count the number of contours (objects)

num_objects = len(contours)

print("Number of objects detected:", num_objects)