1. Create a class STORE to keep track of products (Product Code, Name and price). Display the menu of all products to the user. Generate bill as per order

class Product:

def \_\_init\_\_(self, code, name, price):

self.code = code

self.name = name

self.price = price

class Store:

def \_\_init\_\_(self):

self.products = []

def add\_product(self, product):

self.products.append(product)

def display\_menu(self):

print("Product Menu:")

print("Code\tName\t\tPrice")

for product in self.products:

print(f"{product.code}\t{product.name}\t\t${product.price:.2f}")

def generate\_bill(self, order):

total\_cost = 0

print("\nYour Bill:")

print("Code\tName\t\tPrice")

for product\_code, quantity in order.items():

product = next((p for p in self.products if p.code == product\_code), None)

if product:

cost = product.price \* quantity

total\_cost += cost

print(f"{product.code}\t{product.name}\t\t${product.price:.2f}\tQty: {quantity}\tCost: ${cost:.2f}")

print("\nTotal Bill: ${:.2f}".format(total\_cost))

# Example usage:

# Create product instances

product1 = Product(code="P001", name="Laptop", price=800.0)

product2 = Product(code="P002", name="Mouse", price=20.0)

product3 = Product(code="P003", name="Keyboard", price=50.0)

# Create a store and add products

my\_store = Store()

my\_store.add\_product(product1)

my\_store.add\_product(product2)

my\_store.add\_product(product3)

# Display the menu

my\_store.display\_menu()

# User places an order (product code and quantity)

user\_order = {"P001": 2, "P002": 1}

# Generate and display the bill

my\_store.generate\_bill(user\_order)

Write a Pandas program to add, subtract, multiple and divide two Pandas Series.

Sample Series: [2, 4, 6, 8, 10], [1, 3, 5, 7, 9]

import pandas as pd

# Sample Series

series1 = pd.Series([2, 4, 6, 8, 10])

series2 = pd.Series([1, 3, 5, 7, 9])

# Addition

addition\_result = series1 + series2

# Subtraction

subtraction\_result = series1 - series2

# Multiplication

multiplication\_result = series1 \* series2

# Division

division\_result = series1 / series2

# Display the results

result\_df = pd.DataFrame({

'Series 1': series1,

'Series 2': series2,

'Addition': addition\_result,

'Subtraction': subtraction\_result,

'Multiplication': multiplication\_result,

'Division': division\_result

})

print(result\_df)

1. Write a Pandas program to sort a given Series.

Original Data Series:

0 100

1 200

2 python

3 300.12

4 400

dtype: object

import pandas as pd

# Original Data Series

data\_series = pd.Series(['100', '200', 'python', '300.12', '400'])

# Convert the Series to numeric values, ignoring errors for non-numeric elements

numeric\_series = pd.to\_numeric(data\_series, errors='coerce')

# Sort the numeric Series

sorted\_series = numeric\_series.sort\_values()

# Display the original and sorted Series

result\_df = pd.DataFrame({

'Original Series': data\_series,

'Sorted Series': sorted\_series

})

print(result\_df)

1. Write a Pandas program to select the 'name' and 'score' columns from the following DataFrame.

Sample Python dictionary data and list labels:

exam\_data = {'name': ['Anastasia', 'Dima', 'Katherine', 'James', 'Emily', 'Michael', 'Matthew', 'Laura', 'Kevin', 'Jonas'],

'score': [12.5, 9, 16.5, np.nan, 9, 20, 14.5, np.nan, 8, 19],

'attempts': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],

'qualify': ['yes', 'no', 'yes', 'no', 'no', 'yes', 'yes', 'no', 'no', 'yes']}

labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']

import pandas as pd

import numpy as np

# Sample Python dictionary data and list labels

exam\_data = {

'name': ['Anastasia', 'Dima', 'Katherine', 'James', 'Emily', 'Michael', 'Matthew', 'Laura', 'Kevin', 'Jonas'],

'score': [12.5, 9, 16.5, np.nan, 9, 20, 14.5, np.nan, 8, 19],

'attempts': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],

'qualify': ['yes', 'no', 'yes', 'no', 'no', 'yes', 'yes', 'no', 'no', 'yes']

}

labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']

# Create a DataFrame

df = pd.DataFrame(exam\_data, index=labels)

# Select the 'name' and 'score' columns

selected\_columns = df[['name', 'score']]

# Display the result

print(selected\_columns)

Write a Python class named Rectangle constructed by a length and width and a method which will compute the area of a rectangle.

class Rectangle:

def \_\_init\_\_(self, length, width):

self.length = length

self.width = width

def compute\_area(self):

area = self.length \* self.width

return area

# Example usage:

# Create an instance of the Rectangle class

rectangle\_instance = Rectangle(length=5, width=3)

# Compute and print the area

area\_result = rectangle\_instance.compute\_area()

print(f"The area of the rectangle is: {area\_result}")

Write a Python program to add, multiply two matrices using numpy

import numpy as np

def add\_matrices(matrix1, matrix2):

result = np.add(matrix1, matrix2)

return result

def multiply\_matrices(matrix1, matrix2):

result = np.dot(matrix1, matrix2)

return result

# Example matrices

matrix\_a = np.array([[1, 2, 3],

[4, 5, 6],

[7, 8, 9]])

matrix\_b = np.array([[9, 8, 7],

[6, 5, 4],

[3, 2, 1]])

# Add matrices

sum\_result = add\_matrices(matrix\_a, matrix\_b)

print("Sum of matrices:")

print(sum\_result)

# Multiply matrices

product\_result = multiply\_matrices(matrix\_a, matrix\_b)

print("\nProduct of matrices:")

print(product\_result)

Sort following NumPy array

Case 1: Sort array by the second row

Case 2: Sort the array by the second column

sampleArray = numpy.array([[34,43,73],[82,22,12],[53,94,66]])

import numpy as np

# Given NumPy array

sampleArray = np.array([[34, 43, 73],

[82, 22, 12],

[53, 94, 66]])

# Case 1: Sort array by the second row

sorted\_array\_by\_second\_row = sampleArray[:, sampleArray[1, :].argsort()]

# Case 2: Sort the array by the second column

sorted\_array\_by\_second\_column = sampleArray[sampleArray[:, 1].argsort()]

# Display the results

print("Original Array:")

print(sampleArray)

print("\nCase 1: Sort array by the second row:")

print(sorted\_array\_by\_second\_row)

print("\nCase 2: Sort the array by the second column:")

print(sorted\_array\_by\_second\_column)

Write a Python program that calculates basic statistical operations, including mean, median, and standard deviation, on a given NumPy array

import numpy as np

def calculate\_statistics(numpy\_array):

# Calculate mean, median, and standard deviation

mean\_value = np.mean(numpy\_array)

median\_value = np.median(numpy\_array)

std\_deviation = np.std(numpy\_array)

# Display the results

print("Original Array:")

print(numpy\_array)

print("\nStatistics:")

print(f"Mean: {mean\_value:.2f}")

print(f"Median: {median\_value:.2f}")

print(f"Standard Deviation: {std\_deviation:.2f}")

# Example usage:

# Replace this array with your own data

example\_array = np.array([23, 45, 67, 12, 89, 34, 56, 78, 91, 43])

# Calculate statistics for the example array

calculate\_statistics(example\_array)

Create a result array by adding the following two NumPy arrays. Next, modify the result array by calculating the square of each element. Also generate the transposition of an array using the tool numpy

import numpy as np

# Two NumPy arrays

array1 = np.array([[1, 2, 3], [4, 5, 6]])

array2 = np.array([[7, 8, 9], [10, 11, 12]])

# Add the arrays

result\_array = np.add(array1, array2)

# Calculate the square of each element in the result array

squared\_result = np.square(result\_array)

# Generate the transposition of the result array

transposed\_result = np.transpose(result\_array)

# Display the original arrays, result array, squared result, and transposition

print("Original Array 1:")

print(array1)

print("\nOriginal Array 2:")

print(array2)

print("\nResult Array (Sum of Arrays 1 and 2):")

print(result\_array)

print("\nSquared Result Array:")

print(squared\_result)

print("\nTransposition of Result Array:")

print(transposed\_result)

Write a python program to check how many times a given number can be divided by 3 before it is less than or equal to 10.

def count\_divisions\_by\_three(number):

count = 0

while number > 10:

number /= 3

count += 1

return count

# Example usage:

given\_number = 81

divisions\_count = count\_divisions\_by\_three(given\_number)

print(f"The given number {given\_number} can be divided by 3 {divisions\_count} times before it becomes less than or equal to 10.")