**Python Practical Examination Question list**

| No. | Program Title |
| --- | --- |
| 1 | * Create a bar chart using Matplotlib that displays the distribution of different job roles in the bank marketing dataset. * Plot a histogram using Matplotlib to show the age distribution of the individuals in the dataset. * Create a scatter plot using Matplotlib to examine the relationship between account balance and the last contact duration. Use Bankmarketing.csv   import pandas as pd  import matplotlib.pyplot as plt  # Load the dataset  df = pd.read\_csv('Bankmarketing.csv')  # Task 1: Bar chart for job roles distribution  job\_counts = df['job'].value\_counts()  plt.figure(figsize=(10, 6))  job\_counts.plot(kind='bar', color='skyblue')  plt.title('Distribution of Job Roles')  plt.xlabel('Job Roles')  plt.ylabel('Count')  plt.xticks(rotation=45, ha='right')  plt.show()  # Task 2: Histogram for age distribution  plt.figure(figsize=(10, 6))  plt.hist(df['age'], bins=20, color='lightcoral', edgecolor='black')  plt.title('Age Distribution')  plt.xlabel('Age')  plt.ylabel('Count')  plt.show()  # Task 3: Scatter plot for account balance vs. last contact duration  plt.figure(figsize=(10, 6))  plt.scatter(df['balance'], df['duration'], alpha=0.5, color='green')  plt.title('Scatter Plot: Account Balance vs. Last Contact Duration')  plt.xlabel('Account Balance')  plt.ylabel('Last Contact Duration')  plt.show() |
| 2 | Create a list of 10 random players in a football team, display the list, sort it, and remove the first item with the specified value.  # List of football players  players = [      "Cristiano Ronaldo",      "Lionel Messi",      "Neymar Jr.",      "Kylian Mbappé",      "Robert Lewandowski",      "Mohamed Salah",      "Kevin De Bruyne",      "Sergio Ramos",      "Virgil van Dijk",      "Harry Kane"  ]  # Display the original list of players  print("Original List of Football Players:")  print(players)  players.sort()  print(players)  # Specify values to remove  values\_to\_remove = ["Neymar Jr." ]  # Remove specified values using the remove() method  for value in values\_to\_remove:      if value in players:          players.remove(value)          print("Removed player from the list: ",values\_to\_remove)      else:          print("player is not in the list.")  # Display the modified list of players  print("\nModified List of Football Players:")  print(players) |
| 3 | Write a program to randomly select 10 integer elements from the range 100 to 200 and find the smallest among all.  import random  # Randomly select 10 integer elements from the range 100 to 200  random\_elements = random.sample(range(100, 201), 10)  # Display the randomly selected elements  print("Randomly Selected Elements:")  print(random\_elements)  # Find the smallest element  smallest\_element = min(random\_elements)  # Display the smallest element  print("\nSmallest Element:")  print(smallest\_element) |
| 4 | Create a dictionary of 5 countries with their currency details and display them.  countries\_dict = {      "USA": {"Currency": "US Dollar", "Code": "USD"},      "Japan": {"Currency": "Japanese Yen", "Code": "JPY"},      "United Kingdom": {"Currency": "British Pound", "Code": "GBP"},      "Australia": {"Currency": "Australian Dollar", "Code": "AUD"},      "Brazil": {"Currency": "Brazilian Real", "Code": "BRL"}  }  print(countries\_dict) |
| 5 | Write a program that takes a sentence as input, replaces each blank with a hyphen, and prints the modified sentence.  original\_sentence = input("Enter a sentence: ")  # Replace blanks with hyphens  modified\_sentence = original\_sentence.replace(" ", "-")  # Print the modified sentence  print("Modified Sentence:")  print(modified\_sentence) |
| 6 | Python Program to Solve Quadratic Equations.  import math  # Function to solve quadratic equation  def solve\_quadratic(a, b, c):      # Calculate the discriminant      discriminant = b\*\*2 - 4\*a\*c        # Check if roots are real or complex      if discriminant >= 0:          # Calculate real roots          root1 = (-b + math.sqrt(discriminant)) / (2\*a)          root2 = (-b - math.sqrt(discriminant)) / (2\*a)          return root1, root2      else:          # Calculate complex roots          real\_part = -b / (2\*a)          imaginary\_part = math.sqrt(abs(discriminant)) / (2\*a)          root1 = "{real\_part:.2f} + {imaginary\_part:.2f}i"          root2 = "{real\_part:.2f} - {imaginary\_part:.2f}i"          return root1, root2  # Take user input for coefficients  a = float(input("Enter the coefficient a: "))  b = float(input("Enter the coefficient b: "))  c = float(input("Enter the coefficient c: "))  # Solve the quadratic equation  roots = solve\_quadratic(a, b, c)  # Display the roots  print("Roots of the quadratic equation:")  print("Root 1:", roots[0])  print("Root 2:", roots[1]) |
| 7 | Write a program to print a specified number of Fibonacci numbers.  def generate\_fibonacci(n):      fibonacci\_numbers = [0, 1]      while len(fibonacci\_numbers) < n:          next\_number = fibonacci\_numbers[-1] + fibonacci\_numbers[-2]          fibonacci\_numbers.append(next\_number)      return fibonacci\_numbers[:n]  # Take user input for the number of Fibonacci numbers  num\_fibonacci = int(input("Enter the number of Fibonacci numbers to generate: "))  # Generate and print the specified number of Fibonacci numbers  fibonacci\_sequence = generate\_fibonacci(num\_fibonacci)  print("Fibonacci Numbers (first {num\_fibonacci}):")  print(fibonacci\_sequence) |
| 8 | Create a NumPy array filled with all ones.  import numpy as np  # Create a NumPy array filled with all ones  ones\_array = np.ones((3, 4))  # Display the array  print("NumPy Array Filled with Ones:")  print(ones\_array) |
| 9 | Check whether a NumPy array contains a specified row.  import numpy as np  # Create a NumPy array  array = np.array([      [1, 2, 3],      [4, 5, 6],      [7, 8, 9]  ])  # Specify the row to check  specified\_row = np.array([4, 5, 6])  # Check if the specified row is in the array  contains\_row = np.any(np.all(array == specified\_row, axis=1))  # Display the result  if contains\_row:      print("The array contains the specified row.")  else:      print("The array does not contain the specified row.") |
| 10 | Compute mathematical operations on Array, Add & Multiply two matrices.  import numpy as np  # Create two 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]  ])  # Addition of matrices  matrix\_sum = matrix\_a + matrix\_b  # Multiplication of matrices  matrix\_product = np.dot(matrix\_a, matrix\_b)  # Display the original matrices  print("Matrix A:")  print(matrix\_a)  print("\nMatrix B:")  print(matrix\_b)  # Display the result of addition  print("\nMatrix Sum (A + B):")  print(matrix\_sum)  # Display the result of multiplication  print("\nMatrix Product (A \* B):")  print(matrix\_product) |
| 11 | Find the most frequent value in a NumPy array.  import numpy as np  # Create a NumPy array  my\_array = np.array([1, 2, 3, 4, 2, 2, 3, 1, 2, 4, 4, 5, 2, 2])  # Find unique values and their counts  unique\_values, counts = np.unique(my\_array, return\_counts=True)  # Find the index of the most frequent value  most\_frequent\_index = np.argmax(counts)  # Get the most frequent value  most\_frequent\_value = unique\_values[most\_frequent\_index]  # Display the array and the most frequent value  print("NumPy Array:")  print(my\_array)  print("\nMost Frequent Value:")  print(most\_frequent\_value) |
| 12 | Flatten a 2D numpy array into a 1D array.  import numpy as np  # Create a 2D NumPy array  my\_2d\_array = np.array([[1, 2, 3],                          [4, 5, 6],                          [7, 8, 9]])  # Flatten the 2D array into a 1D array  flattened\_array = my\_2d\_array.flatten()  # Display the original 2D array and the flattened 1D array  print("Original 2D Array:")  print(my\_2d\_array)  print("\nFlattened 1D Array:")  print(flattened\_array) |
| 13 | Calculate the sum of all columns in a 2D NumPy array.  import numpy as np  # Create a 2D NumPy array  my\_2d\_array = np.array([[1, 2, 3],                          [4, 5, 6],                          [7, 8, 9]])  # Calculate the sum of each column  column\_sums = np.sum(my\_2d\_array, axis=0)  # Display the original 2D array and the sum of each column  print("Original 2D Array:")  print(my\_2d\_array)  print("\nSum of Each Column:")  print(column\_sums) |
| 14 | Write a function to calculate the arithmetic mean of a variable number of values.  def calculate\_mean(\*values):      """      Calculate the arithmetic mean of a variable number of values.      Parameters:      \*values (float): Variable number of values.      Returns:      float: Arithmetic mean of the values.      """      if not values:          raise ValueError("At least one value is required to calculate the mean.")      mean = sum(values) / len(values)      return mean  # Example usage:  values\_to\_average = [5, 10, 15, 20]  result = calculate\_mean(\*values\_to\_average)  print(f"Arithmetic Mean: {result}") |
| 15 | Write a lambda function to find the largest of 2 numbers.  # Define a lambda function to find the largest of two numbers  find\_largest = lambda x, y: x if x > y else y  # Example usage  number1 = 10  number2 = 7  largest\_number = find\_largest(number1, number2)  print("The largest number between {number1} and {number2} is:")  print(largest\_number) |
| 16 | Write a program to find the factorial of a given number using recursion.  def factorial(n):      """Recursive function to calculate the factorial of a number."""      if n == 0 or n == 1:          return 1      else:          return n \* factorial(n - 1)  # Take user input for the number  number = int(input("Enter a number to calculate its factorial: "))  # Check if the number is non-negative  if number < 0:      print("Factorial is not defined for negative numbers.")  else:      # Calculate and display the factorial      result = factorial(number)      print("The factorial of number is: ")      print(result) |
|  | Implement a simple banking application with methods:  create Account(),  deposit()  , withdraw(),  computeInterest(),  displayBalance()  class BankAccount:      def \_\_init\_\_(self, account\_holder, initial\_balance=0, interest\_rate=0.05):          self.account\_holder = account\_holder          self.balance = initial\_balance          self.interest\_rate = interest\_rate      def deposit(self, amount):          if amount > 0:              self.balance += amount              print("Deposited ${amount}. New balance: ${self.balance}")          else:              print("Invalid deposit amount. Please enter a positive value.")      def withdraw(self, amount):          if 0 < amount <= self.balance:              self.balance -= amount              print("Withdrew ${amount}. New balance: ${self.balance}")          else:              print("Invalid withdrawal amount or insufficient funds.")      def compute\_interest(self):          interest\_amount = self.balance \* self.interest\_rate          self.balance += interest\_amount          print("Interest computed: ${interest\_amount}. New balance: ${self.balance}")      def display\_balance(self):          print("Account holder: {self.account\_holder}")          print("Current balance: ${self.balance}")  # Example usage:  # Create an account  account1 = BankAccount("John Doe", initial\_balance=1000, interest\_rate=0.1)  # Display initial balance  account1.display\_balance()  # Deposit and withdraw  account1.deposit(500)  account1.withdraw(200)  # Compute interest  account1.compute\_interest()  # Display final balance  account1.display\_balance() |
| 17 | Make an "Invalid Marks" exception class that is thrown when marks obtained by a student exceed 100.  class InvalidMarksException(Exception):      def \_\_init\_\_(self, marks):          super().\_\_init\_\_(f"Invalid marks: {marks}. Marks cannot exceed 100.")  def check\_marks(marks):      if marks > 100:          raise InvalidMarksException(marks)      else:          print(f"The marks obtained are valid: {marks}")  # Example usage:  try:      student\_marks = 10      check\_marks(student\_marks)  except InvalidMarksException as e:      print(f"Error: {e}") |
| 18 | Create a user-defined exception to display a proper message when the value of (b\*d) is zero.  class ZeroProductError(Exception):      def \_\_init\_\_(self, message="Product of b and d cannot be zero."):          self.message = message          super().\_\_init\_\_(self.message)  def calculate\_product(b, d):      if b \* d == 0:          raise ZeroProductError      else:          return b \* d  # Example usage:  try:      b\_value = int(input("Enter the value of b: "))      d\_value = int(input("Enter the value of d: "))      result = calculate\_product(b\_value, d\_value)      print(f"The product of b and d is: {result}")  except ZeroProductError as e:      print(f"Error: {e}")  except ValueError:      print("Invalid input. Please enter valid integers.") |
| 19 | Make use of the assert statement to catch AssertionError.  class InvalidMarksException(Exception):      def \_\_init\_\_(self, marks):          super().\_\_init\_\_(f"Invalid marks: {marks}. Marks cannot exceed 100.")  def check\_marks(marks):      assert marks <= 100, InvalidMarksException(marks)      print(f"The marks obtained are valid: {marks}")  # Example usage:  try:      student\_marks = 105      check\_marks(student\_marks)  except AssertionError as e:      print(f"AssertionError: {e}")  except InvalidMarksException as e:      print(f"Custom Exception: {e}") |
| 20 | * Determine the shape of the data after dropping the feature "Unnamed: 0", missing values, and duplicated values in a dataframe. * Calculate the average age of the clients who have subscribed to a deposit in a given dataset.Use bank\_marketting.csv   import pandas as pd  # Load the dataset  df = pd.read\_csv('bank\_marketing.csv')  # Drop the "Unnamed: 0" column  df = df.drop(columns=['Unnamed: 0'])  # Drop rows with missing values  df = df.dropna()  # Drop duplicate rows  df = df.drop\_duplicates()  # Determine the shape of the data after dropping columns and rows  shape\_after\_dropping = df.shape  print(f"Shape after dropping 'Unnamed: 0', missing values, and duplicates: {shape\_after\_dropping}")  # Calculate the average age of clients who have subscribed to a deposit  average\_age\_subscribed = df[df['y'] == 'yes']['age'].mean()  print(f"Average age of clients who have subscribed to a deposit: {average\_age\_subscribed:.2f}") |

| **No.** | **Question** |
| --- | --- |
| 21 | Write a program to calculate the factorial of a number using loops.  def factorial(n):      """Calculate the factorial of a number using a loop."""      if n < 0:          return "Factorial is not defined for negative numbers."      elif n == 0 or n == 1:          return 1      else:          result = 1          for i in range(2, n + 1):              result \*= i          return result  # Take user input for the number  number = int(input("Enter a number to calculate its factorial: "))  # Calculate and display the factorial  result = factorial(number)  print(f"The factorial of {number} is: {result}") |
| 22 | Create a script to reverse a string entered by the user.  # Take user input for a string  user\_input = input("Enter a string: ")  # Reverse the string  reversed\_string = user\_input[::-1]  # Display the reversed string  print("Reversed String:", reversed\_string) |
| 23 | Develop a program to find the largest and smallest elements in an array.  def find\_largest\_smallest(arr):      if not arr:          return None, None  # Return None for both largest and smallest for an empty array      # Initialize variables to track the largest and smallest elements      largest = smallest = arr[0]      for element in arr:          if element > largest:              largest = element          elif element < smallest:              smallest = element      return largest, smallest  # Example usage:  # Create an array  my\_array = [12, 5, 7, 23, 9, 16, 3]  # Find the largest and smallest elements  largest\_element, smallest\_element = find\_largest\_smallest(my\_array)  # Display the results  print("Array:", my\_array)  print("Largest Element:", largest\_element)  print("Smallest Element:", smallest\_element) |
| 24 | Write a function that checks whether a passed string is a palindrome or not.  def is\_palindrome(input\_string):      # Convert the input string to lowercase for case-insensitive comparison      cleaned\_string = input\_string.lower()      # Remove spaces from the cleaned string      cleaned\_string = ''.join(cleaned\_string.split())      # Check if the cleaned string is equal to its reverse      return cleaned\_string == cleaned\_string[::-1]  # Example usage:  # Take user input for a string  user\_input = input("Enter a string: ")  # Check if the input string is a palindrome  result = is\_palindrome(user\_input)  # Display the result  if result:      print(f"{user\_input} is a palindrome.")  else:      print(f"{user\_input} is not a palindrome.") |
| 25 | Develop a Python function that merges two dictionaries into one.  def merge\_dictionaries(dict1, dict2):      """      Merge two dictionaries into one.      Parameters:      - dict1 (dict): First dictionary      - dict2 (dict): Second dictionary      Returns:      dict: Merged dictionary      """      merged\_dict = dict1.copy()      merged\_dict.update(dict2)      return merged\_dict  # Example usage:  dictionary1 = {'a': 1, 'b': 2}  dictionary2 = {'b': 3, 'c': 4}  merged\_dictionary = merge\_dictionaries(dictionary1, dictionary2)  print("Dictionary 1:", dictionary1)  print("Dictionary 2:", dictionary2)  print("Merged Dictionary:", merged\_dictionary) |
| 26 | Implement a Python class representing a Rectangle with methods to calculate area and perimeter.  class Rectangle:      def \_\_init\_\_(self, length, width):          self.length = length          self.width = width      def calculate\_area(self):          """Calculate the area of the rectangle."""          return self.length \* self.width      def calculate\_perimeter(self):          """Calculate the perimeter of the rectangle."""          return 2 \* (self.length + self.width)  # Example usage:  # Create a Rectangle object  rectangle1 = Rectangle(5, 8)  # Calculate and display the area and perimeter  area = rectangle1.calculate\_area()  perimeter = rectangle1.calculate\_perimeter()  print(f"Rectangle with length {rectangle1.length} and width {rectangle1.width}:")  print(f"Area: {area}")  print(f"Perimeter: {perimeter}") |
| 27 | Create a Numpy program to compute the inverse of a matrix.  import numpy as np  # Create a 2x2 matrix  matrix = np.array([[4, 7],                     [2, 6]])  # Compute the inverse of the matrix  inverse\_matrix = np.linalg.inv(matrix)  # Display the original matrix and its inverse  print("Original Matrix:")  print(matrix)  print("\nInverse Matrix:")  print(inverse\_matrix) |
| 28 | Develop a program to merge two Pandas DataFrames.  import pandas as pd  # Create two DataFrames  df1 = pd.DataFrame({'ID': [1, 2, 3], 'Name': ['John', 'Alice', 'Bob']})  df2 = pd.DataFrame({'ID': [2, 3, 4], 'Age': [25, 30, 22]})  # Merge DataFrames on the 'ID' column  merged\_df = pd.merge(df1, df2, on='ID', how='inner')  # Display the original DataFrames and the merged DataFrame  print("DataFrame 1:")  print(df1)  print("\nDataFrame 2:")  print(df2)  print("\nMerged DataFrame:")  print(merged\_df) |
| 29 | Develop a script to find the GCD (Greatest Common Divisor) of two numbers.  def gcd(a, b):      """Compute the Greatest Common Divisor (GCD) of two numbers using the Euclidean Algorithm."""      while b:          a, b = b, a % b      return abs(a)  # Example usage:  num1 = int(input("Enter the first number: "))  num2 = int(input("Enter the second number: "))  result = gcd(num1, num2)  print(f"The GCD of {num1} and {num2} is: {result}") |
| 30 | Write a Python function to find the sum of digits of a number using recursion.  def sum\_of\_digits(n):      """Calculate the sum of digits of a number using recursion."""      if n == 0:          return 0      else:          return n % 10 + sum\_of\_digits(n // 10)  # Example usage:  number = int(input("Enter a number: "))  result = sum\_of\_digits(number)  print(f"The sum of digits of {number} is: {result}") |
| 31 | Create a Python program to catch multiple exceptions in a single block.  def divide\_numbers(a, b):      """Divide two numbers and handle multiple exceptions."""      try:          result = a / b          print(f"The result of {a} / {b} is: {result}")      except (ZeroDivisionError, TypeError, ValueError) as e:          print(f"Error: {e}")  # Example usage:  try:      num1 = int(input("Enter the numerator: "))      num2 = int(input("Enter the denominator: "))      divide\_numbers(num1, num2)  except ValueError:      print("Invalid input. Please enter valid integers.")  except KeyboardInterrupt:      print("\nProgram interrupted by the user.")  except Exception as e:      print(f"An unexpected error occurred: {e}") |
| 32 | Create a Python program to sort an array of integers using Numpy.  import numpy as np  # Create an array of integers  array\_of\_integers = np.array([5, 2, 8, 1, 3])  # Sort the array  sorted\_array = np.sort(array\_of\_integers)  # Display the original and sorted arrays  print("Original Array:", array\_of\_integers)  print("Sorted Array:", sorted\_array) |