**PROGRAMMING ASSIGNMENT**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **SUBJECT CODE:** | CE259 | **SEMESTER:** | 4 | **ACADEMIC YEAR:** | | 2021-22 |
| **SUBJECT:** | PROGRAMMING IN PYTHON | | | | | |
| **NAME:** | Jiya Shah | | **ROLL NO:** | | 20CS083 | |

|  |  |
| --- | --- |
| **Practical 1** | **You are given a string. Your task is to count the frequency of letters of the string and print the letters in descending order of frequency.**  **If the following string is given as input to the program:**  **abbbccde**  **Then, the output of the program should be:**  **b 3**  **a 2**  **c 2**  **d 1**  **e 1** |
| **Code** | word = input()  dct = {}  for i in word:  dct[i] = dct.get(i,0) + 1  dct = sorted(dct.items(),key=lambda x: (-x[1],x[0]))  for i in dct:  print(i[0],i[1]) |
| **Output** |  |
| **Practical 2** | **Write a procedure to find min, max, mean, standard deviation, variance of number list.**  **Example**  **Input: 10 50 80 70 49 23 11 4**  **Output: 4 80 37. 13 27. 25 848. 70** |
| **Code** | import statistics  import pandas as pd  sr = pd.Series([10, 50, 80, 70, 49, 23, 11, 4])  mean = sr.mean()  median = sr.median()  mode = sr.mode()  range1 = sr.max() - sr.min();  stdeviation = sr.std(axis=0, skipna=True)  print(mean)  print(median)  print(mode)  print(range1)  print(stdeviation)  print("Variance is %s"  % (statistics.variance(sr))) |
| **Output** |  |
| **Practical 3** | **You are given an integer array height of length n. There are n vertical lines drawn such that the two endpoints of the ith line are (i, 0) and (i, height[i]).Find two lines that together with the x-axis form a container, such that the container contains the most water.**  **Return the maximum amount of water a container can store.**  **Notice that you may not slant the container**  **Input: height = [ 1, 8, 6, 2, 5, 4, 8, 3, 7]**  **Output: 49**  **Explanation: The above vertical lines are represented by array [1, 8, 6, 2, 5, 4, 8, 3, 7]**  **In this case, the max area of water (blue section) the container can contain is 49.**  **Example 2:**  **Input: height = [1,1]**  **Output: 1** |
| **Code** | def maxArea(A, Len) :  area = 0  for i in range(Len) :  for j in range(i + 1, Len) :  area = max(area, min(A[j], A[i]) \* (j - i))  return area  a = [1, 8, 6, 2, 5, 4, 8, 3, 7]  b = [1, 1]  len1 = len(a)  print(maxArea(a, len1))  len2 = len(b)  print(maxArea(b, len2)) |
| **Output** |  |
| **Practical 4** | **Given a list of integers, write a program to print the count of all possible unique combinations of numbers whose sum is equal to K** |
| **Code** | from itertools import combinations  numbers = [int(n) for n in input().split()]  k = int(input())  count = 0  for i in range(1, len(numbers)+1):  for c in combinations(numbers, i):  if sum(c) == k:  count += 1  print(count) |
| **Output** |  |
| **Practical 5** | **Explain about the different types of Exceptions in Python with suitable example** |
| **Code** | Some of the basic inbuilt exceptions are:   1. **ArithmeticError Exception**   This class is the base class for those built-in exceptions that are raised for various arithmetic errors such as:   * OverflowError * ZeroDivisionError * FloatingPointError  |  | | --- | | try:  a = 10/0  print (a)  except ArithmeticError:  print ("Arithmetic exception raised")  else:  print ("Success") |  1. **LookupError Exception**   This is the base class for those exceptions that are raised when a key or index used on mapping or sequence is invalid or not found. The exceptions raised are:   * KeyError * IndexError  |  | | --- | | try:  a = 10/0  print (a)  except ArithmeticError:  print ("Arithmetic exception raised")  else:  print ("Success") |  1. **AttributeError Exception**   An AttributeError is raised when an attribute reference or assignment fails such as when a non-existent attribute is referenced   |  | | --- | | class Attributes(object):  pass  object = Attributes()  print(object.attribute) |  1. **FloatingPointError Exception**   A FloatingPointError is raised when a floating point operation fails. This exception is always defined, but can only be raised when Python is configured with the–with-fpectl option, or the WANT\_SIGFPE\_HANDLER symbol is defined in the pyconfig.h file   |  | | --- | | import math  print(math.exp(1000)) |  1. **IndexError Exception**   An IndexError is raised when a sequence is referenced which is out of range   |  | | --- | | array = [ 0, 1, 2]  print (array[3]) | |
| **Output** | 1. **ArithmeticError Exception**      1. **LookupError Exception**      1. **AttributeError Exception**      1. **FloatingPointError Exception**      1. **IndexError Exception** |
| **Practical 6** | **Complete django tutorial (part 1 to part 7) from the official document** |
| **Practical 7** | **Write a django code to send an email with attachment** |
| **Code** | import smtplib  from email.mime.multipart import MIMEMultipart  from email.mime.text import MIMEText  from email.mime.base import MIMEBase  from email import encoders  fromaddr = "light.ryuk.37@gmail.com"  password = "Educ@tion6601"  toaddr = "education6601@gmail.com"  # instance of MIMEMultipart  msg = MIMEMultipart()  # storing the senders email address  msg['From'] = fromaddr  msg['To'] = toaddr  msg['Subject'] = "Check mail"  # string to store the body of the mail  body = "Please find the attached test image."  # attach the body with the msg instance  msg.attach(MIMEText(body, 'plain'))  # open the file to be sent  filename = "indian\_flag.jpg"  attachment = open(filename, "rb")  # instance of MIMEBase and named as p  p = MIMEBase('application', 'octet-stream')  # To change the payload into encoded form  p.set\_payload((attachment).read())  # encode into base64  encoders.encode\_base64(p)  p.add\_header('Content-Disposition', "attachment; filename= %s" % filename)  # attach the instance 'p' to instance 'msg'  msg.attach(p)  # creates SMTP session  server = smtplib.SMTP('smtp.gmail.com', 587)  server.starttls()  server.login(fromaddr, password)  # Converts the Multipart msg into a string  text = msg.as\_string()  server.send\_message(msg)  server.quit() |
| **Output** | **Sender:**  **Receiver:** |
| **Practical 8** | **Program to demonstrate the Overriding of the Base Class method in the Derived Class** |
| **Code** | class P1\_class():  def show(self):  print("Inside Parent Class 1")  class P2\_class():  def display(self):  print("Inside Parent Class 2")  class Child\_class(P1\_class, P2\_class):  def show(self):  print("Inside Child Class")  obj = Child\_class()  obj.show()  obj.display() |
| **Output** |  |
| **Practical 9** | **Write Python code to create a function named move\_rectangle() that takes an object of Rectangle class and two numbers named dx and dy. It should change the location of the Rectangle by adding dx to the x coordinate of corner and adding dy to the y coordinate of corner** |
| **Code** | class Point(object):  pass  class Rectangle(object):  pass  rectangle = Rectangle()  bottom\_left = Point()  bottom\_left.x = 8.0  bottom\_left.y = 3.0  top\_right = Point()  top\_right.x = 9.0  top\_right.y = 6.0  rectangle.corner1 = bottom\_left  rectangle.corner2 = top\_right  dx = 15.0  dy = 16.0  def move\_rectangle(rectangle, dx, dy):  print(f"The rectangle started with bottom left corner at ({rectangle.corner1.x},{rectangle.corner1.y})"  f"\nTop right corner at ({rectangle.corner2.x},{rectangle.corner2.y})"  f"\ndx is {dx} and dy is {dy}")  rectangle.corner1.x = rectangle.corner1.x + dx  rectangle.corner2.x = rectangle.corner2.x + dx  rectangle.corner1.y = rectangle.corner1.y + dy  rectangle.corner2.y = rectangle.corner2.y + dy  print(f"It ended with a bottom left corner at ({rectangle.corner1.x},{rectangle.corner1.y})"  f"\nTop right corner at ({rectangle.corner2.x},{rectangle.corner2.y})")  move\_rectangle(rectangle, dx, dy) |
| **Output** |  |