|  |  |
| --- | --- |
| ASSIGNMENT 1 | from collections import Counter  def CharWithFreq(string):  d = Counter(string)   for i in d:  print(i + " " +str(d[i]))  string = input('Enter a string :') CharWithFreq(string)  OUTPUT: |
| ASSIGNMENT 2 | num\_list = [10, 50, 80, 70, 49, 23, 11,4]  mean = sum(num\_list) / len(num\_list) variance = sum([((x - mean) \*\* 2) for x in num\_list]) / len(num\_list) res = variance \*\* 0.5  # Printing result print("Max of list is : " + str(max(num\_list))) print("Min of list is : " + str(min(num\_list))) print("Mean of list is : " + str(mean)) print("Variance of list is : " + str(variance)) print("Standard deviation of list is : " + str(res))  OUTPUT: |
| ASSIGNMENT 3 | 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 = [int(n) for n in input("Enter an array: ").split()]  len1 = len(a) print(maxArea(a, len1))  **OUTPUT:** |
| ASSIGMENT 4 | from itertools import combinations  values =[int(i) for i in input().split()] values.sort()  K = int(input('Enter K: '))  p=0 for i in range(1, len(values)+1):  com =list(set(combinations(values, i)))  for combination in com:  if sum(combination) == K:  p += 1 print(f"The count of all unique combinations is: {p}")  OUTPUT: |
| ASSIGNMENT 5 | Some of the basic inbuilt exceptions are:   1. exception ArithmeticError   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 ("This statement is raising an arithmetic exception.")  else:      print ("Success.")     1. exception LookupError   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 = [1, 2, 3]      print (a[3])  except LookupError:      print ("Index out of bound error.")     1. exception AttributeError   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. exception FloatingPointError   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. exception IndexError   An IndexError is raised when a sequence is referenced which is out of range.  array = [ 0, 1, 2 ]  print (array[3]) |
| ASSIGNMENT 8 | class Animal:  multicellular = True  eukaryotic = True   def breathe(self):  print("Class Animal- breathe")   def feed(self):  print("Class Animal- feed")  class Herbivorous(Animal):   def feed(self):  print("Class Herbivorous- feed")   herbi = Herbivorous() herbi.feed() herbi.breathe()  OUTPUT: |
| ASSIGNMENT 9 | 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: |