|  |  |
| --- | --- |
| Candidate name: |  |
| Centre number: |  |
| Index number: |  |
| Programming language used: |  |

|  |
| --- |
| **Question 1**  **Evidence 1** |
| *Paste program code here*  def get\_highest\_freq(file):  f = open(file)  freq\_data = []  for line in f:  if line.strip().isalpha():  freq\_data.append([line.strip()])  elif line.strip().isdigit():  freq\_data[-1].append(int(line.strip()))  f.close()  freq\_data = quick\_sort(freq\_data, 1)  return freq\_data[0][0]  def quick\_sort(array, index):  if len(array) < 2:  return array  left = []  right = []  pivot = array[0]  for i in range(1, len(array)):  if array[i][index] > pivot[index]:  left.append(array[i])  else:  right.append(array[i])  return quick\_sort(left, index) + [pivot] + quick\_sort(right, index) |
| **Evidence 2** |
| *Paste screenshot here* |
| **Evidence 3** |
| *Paste program code here*  def amended\_get\_highest\_freq(file):  f = open(file)  freq\_data = []  for line in f:  if line.strip().isalpha():  freq\_data.append([line.strip()])  elif line.strip().isdigit():  freq\_data[-1].append(int(line.strip()))  f.close()  freq\_data = quick\_sort(freq\_data, 1)  for i in range(1, len(freq\_data)):  if freq\_data[i][1] < freq\_data[0][1]:  break  freq\_data = freq\_data[:i]  return [freq\_data[i][0] for i in range(len(freq\_data))] |
| **Evidence 4** |
| *Paste screenshot here* |
| **Question 2**  **Evidence 5** |
| *Paste program code here*  #First line of file, after being split with "  ##['EmployeeCode(1) = ', 'L001', ' : Surname(1) = ', 'Pollard', \  ##' : EmployeeCode(2) = ', 'L002', ' : Surname(2) = ', 'Wills', '']  #Index 1 is the first employee code, 3 is name for that person  #Index 5 is the second employee code, 7 is name for that person  def get\_file\_data(): #returns ARRAY of ARRAYS:[ID, Surname]  file\_data = []  f = open("EMPLOYEEDATA.TXT")  for line in f:  temp = line.strip().split('"')  file\_data.append([temp[1], temp[3]])  file\_data.append([temp[5], temp[7]])  f.close()  return file\_data  def search\_by\_surname():  while True:  target = input("Enter a valid Surname: ")  if target.isalpha():  return search(target, 1)[0]  print("Invalid Surname! Try again.")  def search\_by\_employee\_id():  while True:  target = input("Enter a valid Employee ID: ")  if len(target) == 4:  if target[0].isalpha() and target[1:].isdigit():  return search(target, 0)[1]  print("Invalid Employee ID! Try again.")  def search(target, index):  file\_data = get\_file\_data()  for i in range(len(file\_data)):  if file\_data[i][index].upper() == target.upper():  return file\_data[i]  return ['Surname Not Found', 'Employee ID Not Found'] #when not found |
| **Evidence 6** |
| *Paste annotated screenshots here*  **Abnormal Surname**  **Valid Employee ID and found**  **Valid Employee ID but not found**  **Abnormal Employee ID** |
| **Question 3**  **Evidence 7** |
| *Paste program code here*  class Node():  def \_\_init\_\_(self, Data, LeftP = 0, RightP = 0):  self.\_Data = Data  self.\_LeftP = LeftP  self.\_RightP = RightP  def get\_Data(self):  return self.\_Data  def set\_Data(self, new\_Data):  self.\_Data = new\_Data  def get\_LeftP(self):  return self.\_LeftP  def set\_LeftP(self, new\_LeftP):  self.\_LeftP = new\_LeftP  def get\_RightP(self):  return self.\_RightP  def set\_RightP(self, new\_RightP):  self.\_RightP = new\_RightP  class BST():  def \_\_init\_\_(self):  self.\_ThisTree = [None] + [Node('', i, 0) for i in range(2, 21)]\  + [Node('')]  self.\_Root = 0  self.\_NextFreePosition = 1  tree = BST() |
| **Evidence 8** |
| *Paste program code here*  def AddItemToBinaryTree(self, NewTreeItem):  if self.\_NextFreePosition == 0:  return False  self.\_ThisTree[self.\_NextFreePosition].set\_Data(NewTreeItem)    if self.\_Root == 0:  self.\_Root = self.\_NextFreePosition  else:  #traversse the tree to find the position for the new value  CurrentPosition = self.\_Root  LastMove = 'X'  while CurrentPosition != 0:  PreviousPosition = CurrentPosition  if NewTreeItem < self.\_ThisTree[CurrentPosition].get\_Data():  #move left  LastMove = 'L'  CurrentPosition = self.\_ThisTree[CurrentPosition]\  .get\_LeftP()  else:  #move right  LastMove = 'R'  CurrentPosition = self.\_ThisTree[CurrentPosition]\  .get\_RightP()  if LastMove == 'X':  raise IndexError  if LastMove == 'R':  self.\_ThisTree[PreviousPosition]\  .set\_RightP(self.\_NextFreePosition)  else:  self.\_ThisTree[PreviousPosition]\  .set\_LeftP(self.\_NextFreePosition)  temp = self.\_ThisTree[self.\_NextFreePosition]  self.\_NextFreePosition = self.\_ThisTree[self.\_NextFreePosition]\  .get\_LeftP()  temp.set\_LeftP(0)  return True |
| **Evidence 9** |
| *Paste program code here*  def OutputData(self):  print("Root:", self.\_Root)  print("NextFreePosition:", self.\_NextFreePosition)  print("{:<10}{:<20}{:<10}{}".format("Index", "Data", "LeftP", "RightP"))  for i in range(1, len(self.\_ThisTree)):  print("{:<10}{:<20}{:<10}{}".format(\  i,\  self.\_ThisTree[i].get\_Data(),\  self.\_ThisTree[i].get\_LeftP(),\  self.\_ThisTree[i].get\_RightP())) |
| **Evidence 10** |
| *Paste screenshot here* |
| **Evidence 11** |
| *Paste screenshot here* |
| **Evidence 12** |
| *Paste program code here*  def InOrderTraversal(self, index = None):  if index == None:  index = self.\_Root  if index != 0:  self.InOrderTraversal(self.\_ThisTree[index].get\_LeftP())  print(self.\_ThisTree[index].get\_Data())  self.InOrderTraversal(self.\_ThisTree[index].get\_RightP()) |
| **Evidence 13** |
| *Paste screenshot here* |
| **Question 4**  **Evidence 14** |
| *Paste program code here*  def get\_freq():  freq\_data = []  x\_value\_count = 0  while True:  while True:  x\_value = input("Next X value ... <ZZZ to END> ")  if x\_value == 'ZZZ':  return freq\_data  if x\_value\_count == 6:  print("No more than six X values are allowed. 'ZZZ' to END.")  else:  break  while True:  freq = input("Frequency ... ")  if freq.isdigit() and int(freq) in range(61):  freq = int(freq)  break  print("Invalid Frequency value! Try again.")  freq\_data.append([x\_value, freq])  x\_value\_count += 1  return freq\_data  def print\_freq(freq\_data):  #freq\_data: ARRAY[[X Value, Frequency] for each X Value]  print("\n\n" + "+" \* 40)  print("Frequency distribution")  print("+" \* 40 + "\n")  for i in range(len(freq\_data)):  print(" {:<8}{}".format(freq\_data[i][0], "@" \* freq\_data[i][1]))  print\_freq(get\_freq()) |
| **Evidence 15** |
| *Paste screenshot here* |
| **Evidence 16** |
| *Paste program code here*  #max 40 lines  #header takes 4 lines (including the empty line before the first x value  #36 lines allowed for max 6 values  #each line is thus allowed a max of 6 bar width  #'@' is replaced with '#' as it is easier to see    def amended\_print\_freq(freq\_data):  #freq\_data: ARRAY[[X Value, Frequency] for each X Value]  print("\n\n" + "+" \* 40)  print("Frequency distribution")  print("+" \* 40 + "\n")  for i in range(len(freq\_data)):  print(" {:<8}{}".format(freq\_data[i][0], "#" \* freq\_data[i][1]))  for j in range(5):  print("{:<10}{}".format("", "#" \* freq\_data[i][1]))  temp = get\_freq()  amended\_print\_freq(temp) |
| **Evidence 17** |
| *Paste screenshot here* |
| **Evidence 18** |
| *Paste program code here* |
| **Evidence 19** |
| *Paste screenshots here* |