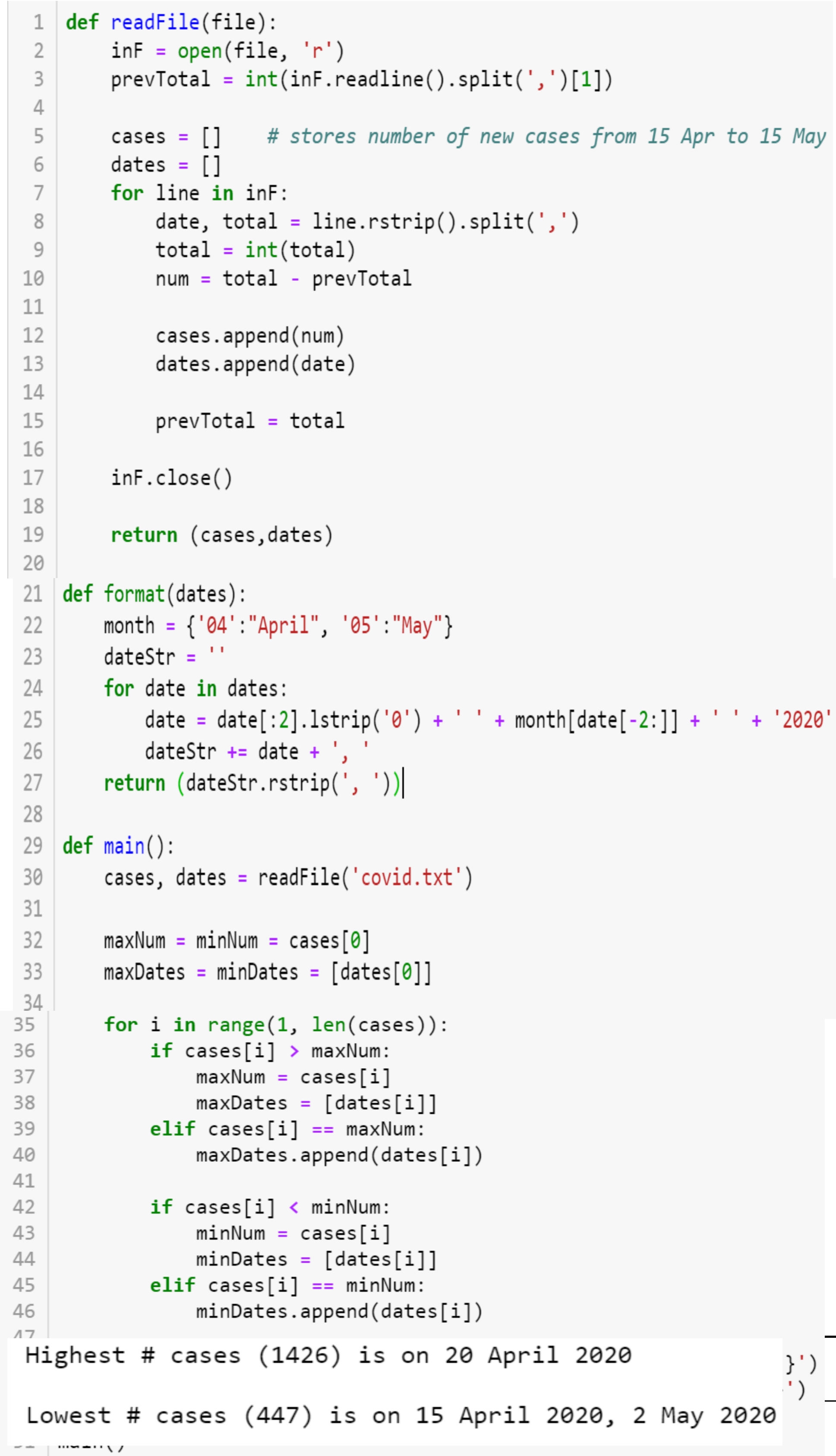
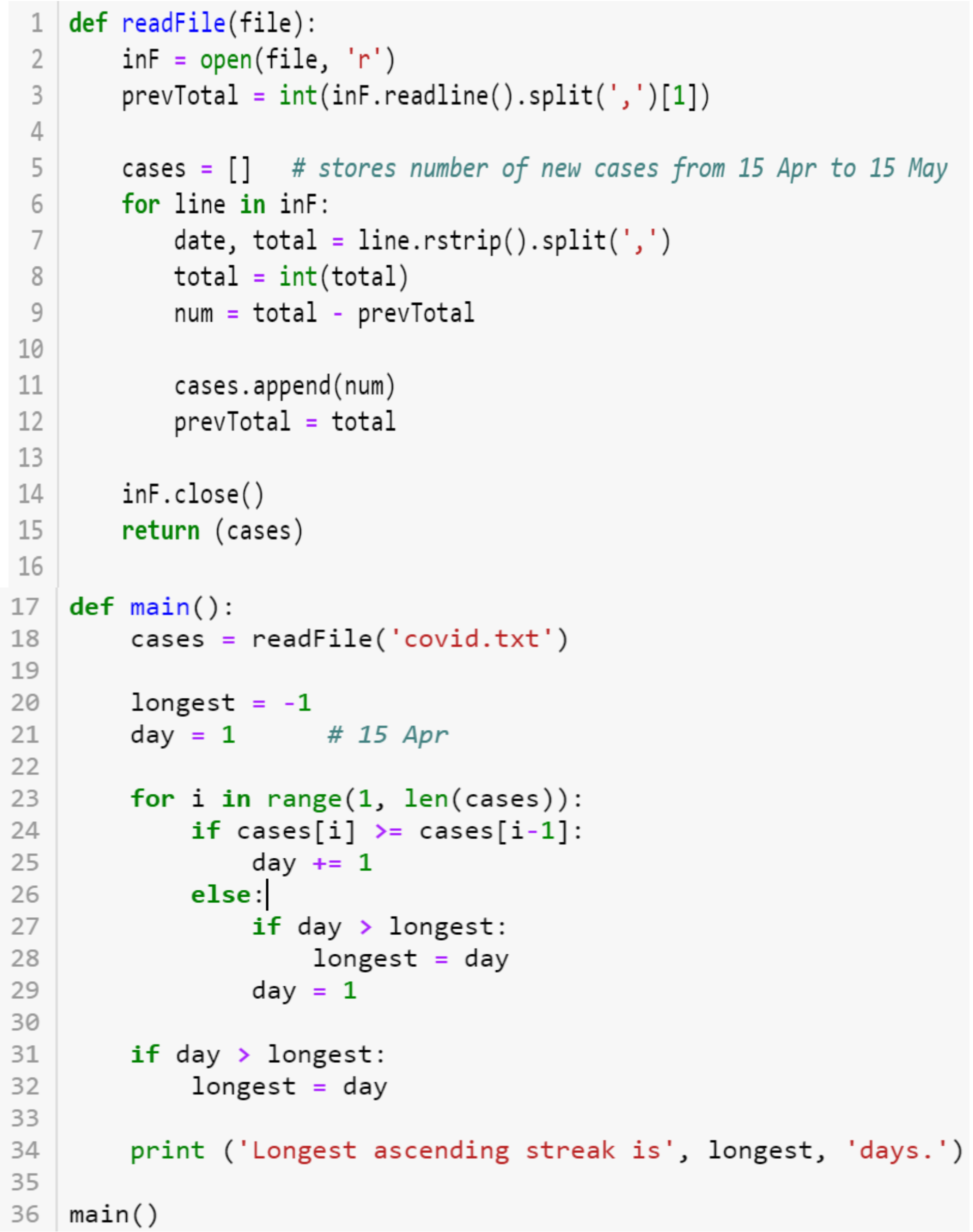
Task 1.1



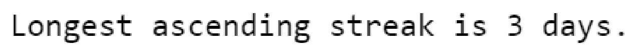
```python
def readFile(file):
    inF = open(file, 'r')
    prevTotal = int(inF.readline().split(',')[1])

    cases = []    # stores number of new cases from 15 Apr to 15 May
    dates = []
    for line in inF:
        date, total = line.rstrip().split(',')
        total = int(total)
        num = total - prevTotal

        cases.append(num)
        dates.append(date)

        prevTotal = total

    inF.close()

    return (cases,dates)

def format(dates):
    month = {'04':"April", '05':"May"}
    dateStr = ''
    for date in dates:
        date = date[:2].lstrip('0') + ' ' + month[date[-2:]] + ' ' + '2020'
        dateStr += date + ', '
    return (dateStr.rstrip(', '))

def main():
    cases, dates = readFile('covid.txt')

    maxNum = minNum = cases[0]
    maxDates = minDates = [dates[0]]

    for i in range(1, len(cases)):
        if cases[i] > maxNum:
            maxNum = cases[i]
            maxDates = [dates[i]]
        elif cases[i] == maxNum:
            maxDates.append(dates[i])

        if cases[i] < minNum:
            minNum = cases[i]
            minDates = [dates[i]]
        elif cases[i] == minNum:
            minDates.append(dates[i])
```

```
Highest # cases (1426) is on 20 April 2020

Lowest # cases (447) is on 15 April 2020, 2 May 2020
```


Task 1.2



```python
def readFile(file):
    inF = open(file, 'r')
    prevTotal = int(inF.readline().split(',')[1])

    cases = []   # stores number of new cases from 15 Apr to 15 May
    for line in inF:
        date, total = line.rstrip().split(',')
        total = int(total)
        num = total - prevTotal

        cases.append(num)
        prevTotal = total

    inF.close()
    return (cases)

def main():
    cases = readFile('covid.txt')

    longest = -1
    day = 1       # 15 Apr

    for i in range(1, len(cases)):
        if cases[i] >= cases[i-1]:
            day += 1
        else:
            if day > longest:
                longest = day
            day = 1

    if day > longest:
        longest = day

    print ('Longest ascending streak is', longest, 'days.')

main()
```

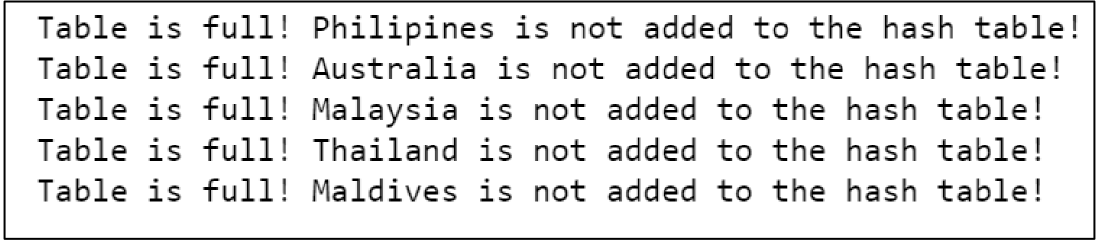


```
Longest ascending streak is 3 days.
```

```
# Task 2.1, function to create hash address, 3 marks
def HashKey(Country):
    country = Country.lower()
    total = 0
    for char in country:
        total += ord(char)
    return total % 30
```

```
# Task 2.2, create hash table with text file, 7 marks
size = 30
HashTable = [''] * size

with open('COUNTRY1.txt', 'r') as f:
    for line in f:
        country = line.strip()
        address = HashKey(country)
        if HashTable[address] == '':
            HashTable[address] = country
        else:
            index = address
            found = False
            tableFull = False

            while not found and not tableFull:
                if HashTable[index] == '':
                    found = True
                    HashTable[index] = country
                else:
                    index += 1
                    if index == size:
                        index = 0
                    if index == address:
                        tableFull = True

           if tableFull:
                print("Table is full!", country, "is not added to
the hash table!")
```

# 1 mark for 5 error message in output



```
Table is full! Philipines is not added to the hash table!
Table is full! Australia is not added to the hash table!
Table is full! Malaysia is not added to the hash table!
Table is full! Thailand is not added to the hash table!
Table is full! Maldives is not added to the hash table!
```

```
# Task 2.3 hash table search and test, 9 marks
def searchCountry(HashTable, country):
    address = HashKey(country)
    found = False
    exit = False
    index = address

    while not found and not exit:
        if HashTable[index] == country:
            found = True
        elif HashTable[index] == '':
            exit = True
        else:
            index += 1
            if index == size:
                index = 0
            if index == address:
                exit = True

    if found:
        print(country, "is found at address", index)
        print()
    else:
        print(country, "is not found in the hash table.")
        print()

# test cases, 3 mark
# locating by hash address, 'USA'found
searchCountry(HashTable, 'USA')
# locating by hash address, cell with different value, 'Spain'
found
searchCountry(HashTable, 'Spain')
# locating by hash address, cell with different value, 'Vietnam'
not found
searchCountry(HashTable, 'Vietnam')
```

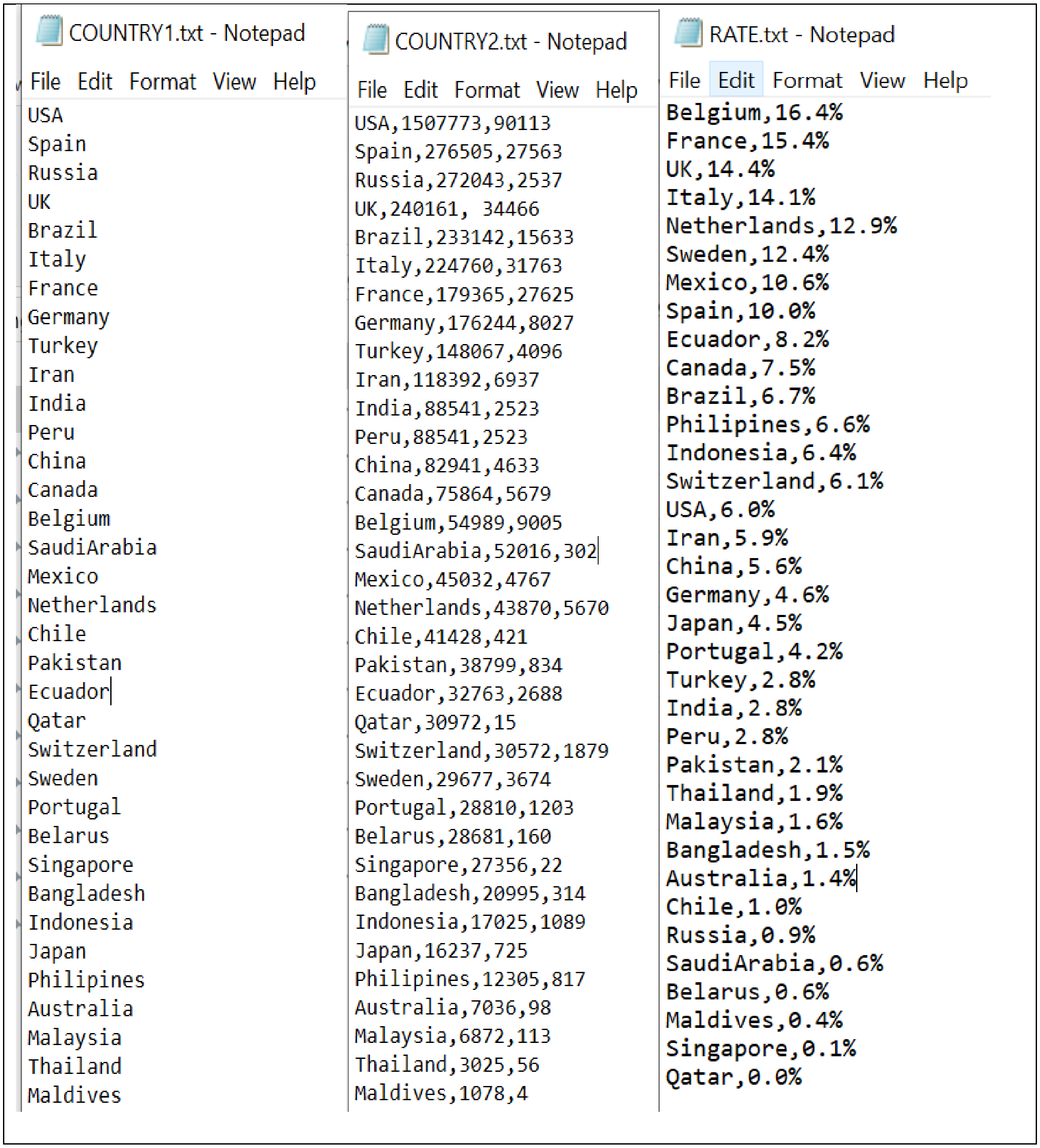
Found at Hash Address: USA [29], Russia [3], UK [14], Italy [7], France [23], Germany [5], Turkey [16], Iran [6], Peru [24], Belgium [21], SaudiArabia [2], Pakistan [19], Qatar [27

Found with collision:

Country	Hash Address	Hash Table Index	Country	Hash Address	Hash Table Index
Spain	29	0	Switzerland	7	12
Brazil	14	15	Sweden	16	18

India	7	8	Portugal	8	13
China	5	9	Belarus	0	4
Canada	0	1	Singapore	8	22
Mexico	15	17	Bangladesh	13	25
Netherlands	6	10	Indonesia	24	26
Chile	7	11	Japan	12	28
Ecuador	19	20			

```
#Task 2.4 Bubble Sort, 9 marks

# read text file, create list of rates, and
# dictionary of key: death rate, value: [country names]

with open('COUNTRY2.txt', 'r') as f:
    rateDict = {}
    rateList = []
    for line in f:
        line = line.strip()
        country, confirm, death = line.split(',')

        confirm = int(confirm)
        death = int(death)
        rate = round(death / confirm * 100, 1)

        if rate in rateDict:
            rateDict[rate].append(country)
        else:
            rateDict[rate] = [country]

        if rate not in rateList:
            rateList.append(rate)

# sort list
size = len(rateList)
for i in range(size, 1, -1):
    for j in range(i - 1):
        rate1 = rateList[j]
        rate2 = rateList[j + 1]
        if rate1 < rate2:
            rateList[j] = rate2
            rateList[j + 1] = rate1


with open('RATE.txt', 'w') as f:
    for rate in rateList:
        line = ''
        for country in rateDict[rate]:
            line = country + ',' + str(rate) + '%\n'
            f.write(line)
```


COUNTRY1.txt - Notepad
File Edit Format View Help
COUNTRY2.txt - Notepad
File Edit Format View Help
RATE.txt - Notepad
File Edit Format View Help

```
USA
Spain
Russia
UK
Brazil
Italy
France
Germany
Turkey
Iran
India
Peru
China
Canada
Belgium
SaudiArabia
Mexico
Netherlands
Chile
Pakistan
Ecuador
Qatar
Switzerland
Sweden
Portugal
Belarus
Singapore
Bangladesh
Indonesia
Japan
Philipines
Australia
Malaysia
Thailand
Maldives
```

```
USA,1507773,90113
Spain,276505,27563
Russia,272043,2537
UK,240161, 34466
Brazil,233142,15633
Italy,224760,31763
France,179365,27625
Germany,176244,8027
Turkey,148067,4096
Iran,118392,6937
India,88541,2523
Peru,88541,2523
China,82941,4633
Canada,75864,5679
Belgium,54989,9005
SaudiArabia,52016,302
Mexico,45032,4767
Netherlands,43870,5670
Chile,41428,421
Pakistan,38799,834
Ecuador,32763,2688
Qatar,30972,15
Switzerland,30572,1879
Sweden,29677,3674
Portugal,28810,1203
Belarus,28681,160
Singapore,27356,22
Bangladesh,20995,314
Indonesia,17025,1089
Japan,16237,725
Philipines,12305,817
Australia,7036,98
Malaysia,6872,113
Thailand,3025,56
Maldives,1078,4
```

```
Belgium,16.4%
France,15.4%
UK,14.4%
Italy,14.1%
Netherlands,12.9%
Sweden,12.4%
Mexico,10.6%
Spain,10.0%
Ecuador,8.2%
Canada,7.5%
Brazil,6.7%
Philipines,6.6%
Indonesia,6.4%
Switzerland,6.1%
USA,6.0%
Iran,5.9%
China,5.6%
Germany,4.6%
Japan,4.5%
Portugal,4.2%
Turkey,2.8%
India,2.8%
Peru,2.8%
Pakistan,2.1%
Thailand,1.9%
Malaysia,1.6%
Bangladesh,1.5%
Australia,1.4%
Chile,1.0%
Russia,0.9%
SaudiArabia,0.6%
Belarus,0.6%
Maldives,0.4%
Singapore,0.1%
Qatar,0.0%
```

Task 3.1

[1M] ProductCode is Primary Key for Product table

[1M] ProductCode is Foreign Key for Other 3 tables

[1M] Correct Data Types

[1M] Rest are all correct

```
CREATE TABLE "Product" (
     "ProductCode"  TEXT NOT NULL PRIMARY KEY,
     "Name"    TEXT,
     "Type"    TEXT,
     "Location" TEXT,
     "Price"   REAL
);
CREATE TABLE "Cake" (
     "ProductCode"  TEXT NOT NULL,
     "ServingSize"  INTEGER,
     "Shape"   TEXT,
     FOREIGN KEY("ProductCode") REFERENCES
"Product"("ProductCode")
);
CREATE TABLE "Loaf" (
     "ProductCode"  TEXT NOT NULL,
     "Weight"  REAL,
     FOREIGN KEY("ProductCode") REFERENCES
"Product"("ProductCode")
);
CREATE TABLE "Bun" (
     "ProductCode"  TEXT NOT NULL,
     "PiecesPerPackage"  INTEGER,
     FOREIGN KEY("ProductCode") REFERENCES
"Product"("ProductCode")
);
```

Task 3.2

```
import sqlite3
import csv

try:
      conn = sqlite3.connect("bakery.db")
      cur = conn.cursor()

      with open('CAKES.TXT', newline='') as csvfile:
            records = csv.reader(csvfile, delimiter=',', quotechar='"')
            for row in records:
                 cur.execute("Insert into Product(productcode, Name,
Type, Location, Price) Values(?,?,?,?,?)", (row[0],row[1], 'Cake',
row[2], float(row[3])))
                  cur.execute("Insert into Cake(productcode, ServingSize,
Shape) Values(?,?,?)", (row[0], row[4], row[5]))
                  conn.commit()

      with open('LOAVES.TXT', newline='') as csvfile:
            records = csv.reader(csvfile, delimiter=',', quotechar='"')
            for row in records:
                cur.execute("Insert into Product(productcode, Name, Type,
Location, Price) Values(?,?,?,?,?)", (row[0], row[1], 'Loaf', row[2],
float(row[3])))
```

```
                  cur.execute("Insert into Loaf(productcode, Weight)
Values(?,?)", (row[0],float(row[4])))
                  conn.commit()

      with open('BUNS.TXT', newline='') as csvfile:
            records = csv.reader(csvfile, delimiter=',', quotechar='"')
            for row in records:
                  cur.execute("Insert into Product(productcode, Name,
Type, Location, Price) Values(?,?,?,?,?)", (row[0],row[1], 'Bun', row[2],
row[3]))
                  cur.execute("Insert into Bun(productcode,
PiecesPerPackage) Values(?,?)", (row[0],float(row[4])))
                  conn.commit()
       conn.close()
 except Exception as err:
       print('Error: %s' % (str(err)))
 finally:
       conn.close()
```

Task 3.3

```
select p.ProductCode, p.Name,
p.Location, p.Price,
c.ServingSize from Product p inner join cake c on p.productcode
= c.productcode
and c.Shape='Circle'
```

Task 3.4

### /templates/index.html

```
<!DOCTYPE html>
<html>
<head><title>Bakery</title>
</head>
<body>
<form action="{{ url_for('index') }}" method="POST">
      <p>
      Location: <input type="text" value="" name="location"> &nbsp;&nbsp;
    <input type="submit">
      </p>
</form>
</html>
```

### /templates/result.html

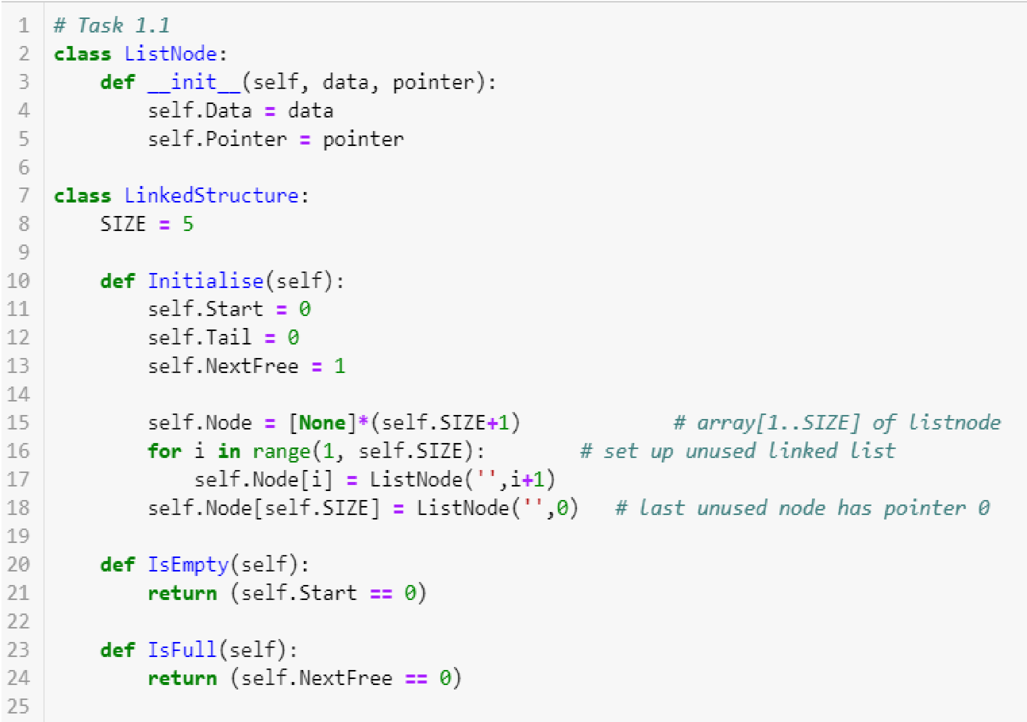
```
<!DOCTYPE html>
<html>
<head><title>Bakery</title>
</head>
<body>
```

```
<p>Listing</p>
      <table>
            <tr><th>Name</th><th>Type</th><th>Price</th><tr>
            {% if results|length > 0 %}
                  {% for item in results  %}
                  <tr>
                  <td>{{ item[0] }}</td><td>{{ item[1] }}</td><td>
{{ item[2] }}</td></tr>
                  {% endfor %}
            {%else%}
            <tr>
                  <td colspan="6">No Items</td>
            </tr>
            {%endif%}
      </table>
</body>
</html>
```

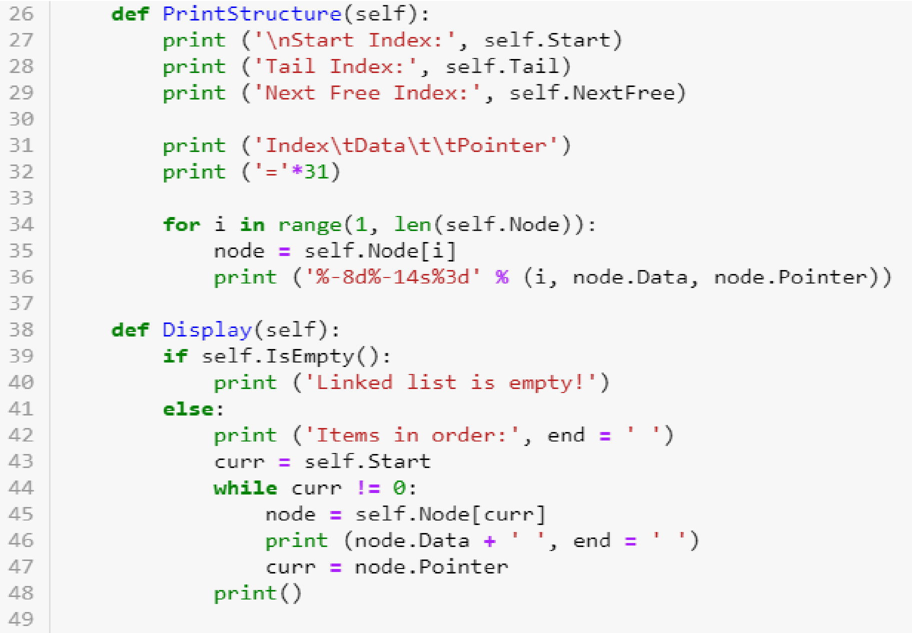
#### app.py

```
import flask, os, sqlite3
from flask import render_template, request

app = flask.Flask( __name__,  static_folder = './static', template_folder
= './templates')

@app.route('/', methods=['GET', 'POST'])
def index():
        if request.method == 'POST':
                location = request.form['location']
                conn = sqlite3.connect('bakery.db')
                cursor  =  conn.execute("select  name,  type,  price  from
product where                                         location    =    ?
order by price asc",(location,))
                all_rows = cursor.fetchall()
                cursor.close()
                conn.close()
                return  render_template(   'result.html',   results   =
all_rows)
        elif request.method == 'GET':
                return render_template('index.html')


if __name__ == '__main__':
    app.run()
```

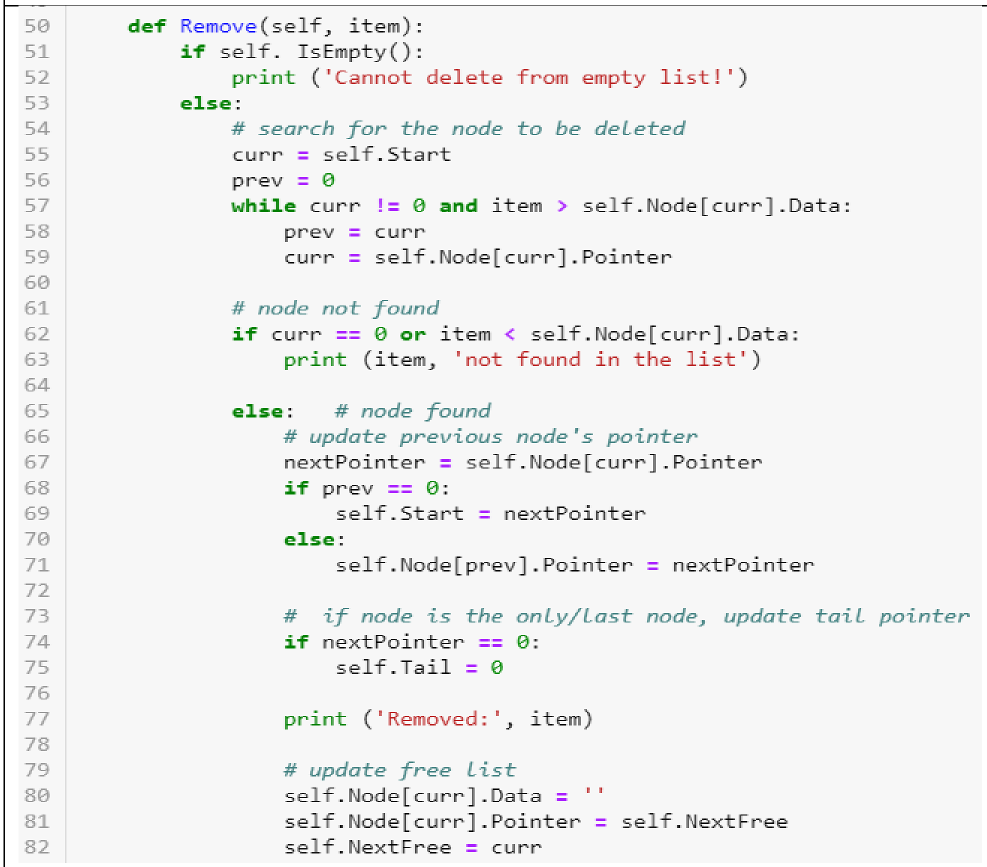
Task 4.1



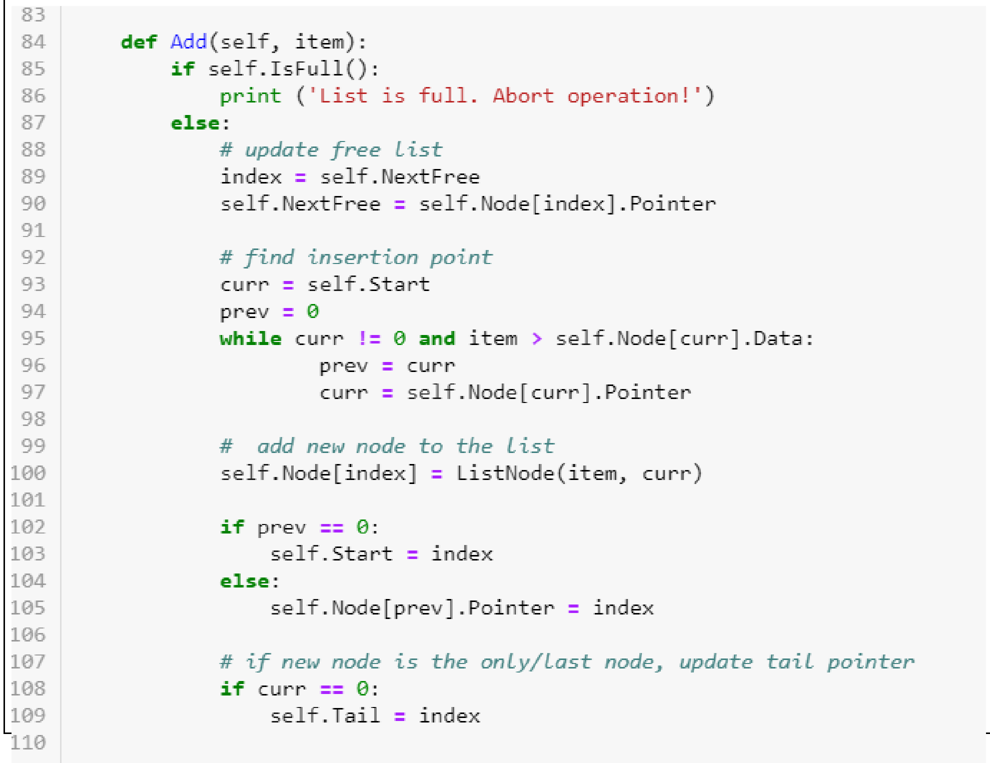
```python
# Task 1.1
class ListNode:
    def __init__(self, data, pointer):
        self.Data = data
        self.Pointer = pointer

class LinkedStructure:
    SIZE = 5

    def Initialise(self):
        self.Start = 0
        self.Tail = 0
        self.NextFree = 1

        self.Node = [None]*(self.SIZE+1)             # array[1..SIZE] of listnode
        for i in range(1, self.SIZE):        # set up unused linked list
            self.Node[i] = ListNode('',i+1)
        self.Node[self.SIZE] = ListNode('',0)   # last unused node has pointer 0

    def IsEmpty(self):
        return (self.Start == 0)

    def IsFull(self):
        return (self.NextFree == 0)

```



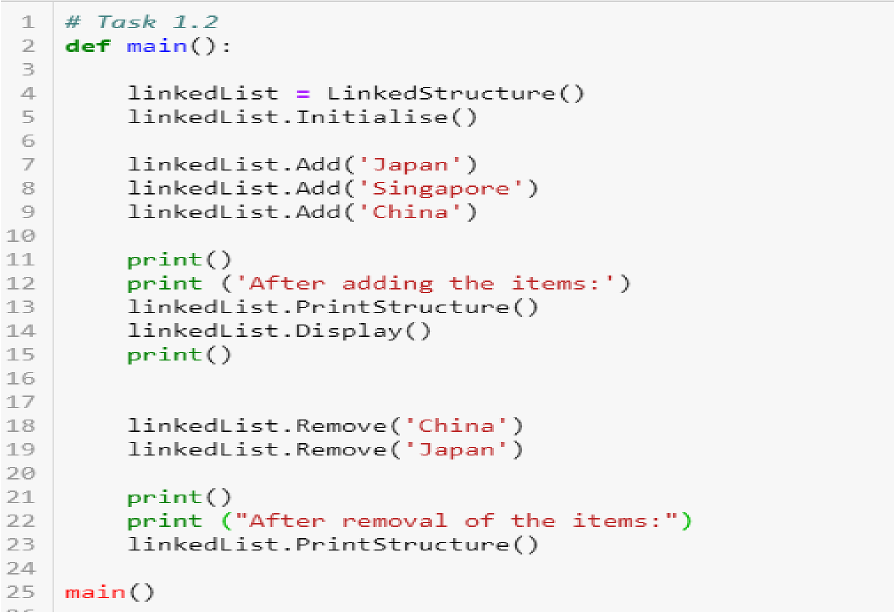
```python
    def PrintStructure(self):
        print ('\nStart Index:', self.Start)
        print ('Tail Index:', self.Tail)
        print ('Next Free Index:', self.NextFree)

        print ('Index\tData\t\tPointer')
        print ('='*31)

        for i in range(1, len(self.Node)):
            node = self.Node[i]
            print ('%-8d%-14s%3d' % (i, node.Data, node.Pointer))

    def Display(self):
        if self.IsEmpty():
            print ('Linked list is empty!')
        else:
            print ('Items in order:', end = ' ')
            curr = self.Start
            while curr != 0:
                node = self.Node[curr]
                print (node.Data + ' ', end = ' ')
                curr = node.Pointer
            print()

```



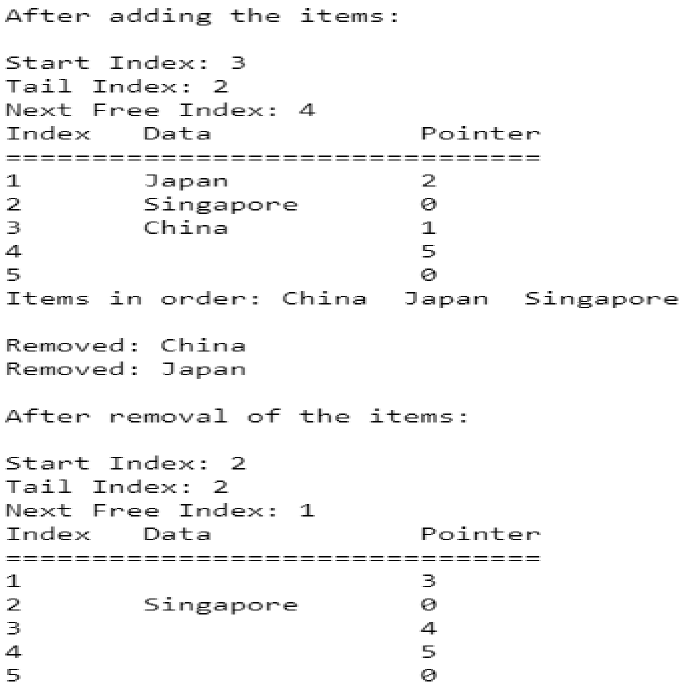
```python
    def Remove(self, item):
        if self. IsEmpty():
            print ('Cannot delete from empty list!')
        else:
            # search for the node to be deleted
            curr = self.Start
            prev = 0
            while curr != 0 and item > self.Node[curr].Data:
                prev = curr
                curr = self.Node[curr].Pointer

            # node not found
            if curr == 0 or item < self.Node[curr].Data:
                print (item, 'not found in the list')

            else:   # node found
                # update previous node's pointer
                nextPointer = self.Node[curr].Pointer
                if prev == 0:
                    self.Start = nextPointer
                else:
                    self.Node[prev].Pointer = nextPointer

                #  if node is the only/last node, update tail pointer
                if nextPointer == 0:
                    self.Tail = 0

                print ('Removed:', item)

                # update free list
                self.Node[curr].Data = ''
                self.Node[curr].Pointer = self.NextFree
                self.NextFree = curr
```



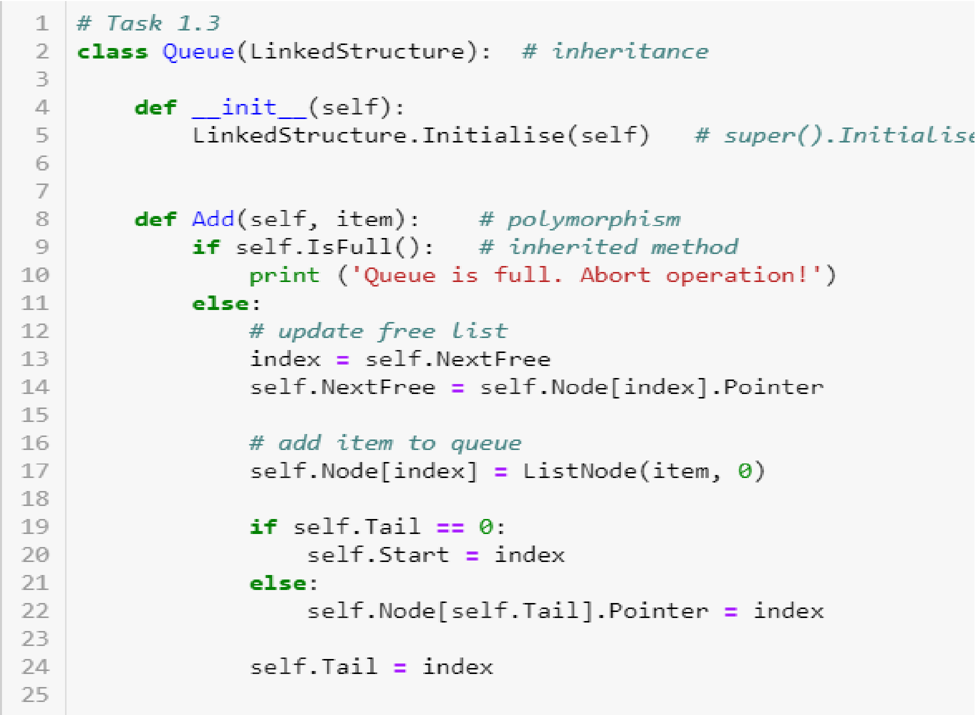
```python

    def Add(self, item):
        if self.IsFull():
            print ('List is full. Abort operation!')
        else:
            # update free list
            index = self.NextFree
            self.NextFree = self.Node[index].Pointer

            # find insertion point
            curr = self.Start
            prev = 0
            while curr != 0 and item > self.Node[curr].Data:
                    prev = curr
                    curr = self.Node[curr].Pointer

            #  add new node to the list
            self.Node[index] = ListNode(item, curr)

            if prev == 0:
                self.Start = index
            else:
                self.Node[prev].Pointer = index

            # if new node is the only/last node, update tail pointer
            if curr == 0:
                self.Tail = index

```



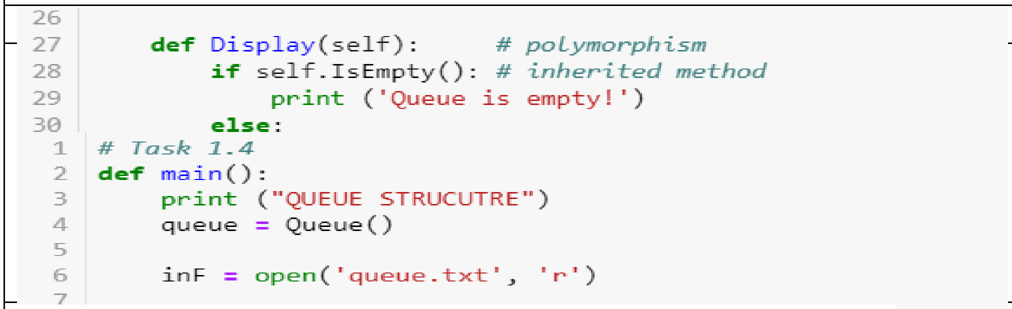
```python
# Task 1.2
def main():

    linkedList = LinkedStructure()
    linkedList.Initialise()

    linkedList.Add('Japan')
    linkedList.Add('Singapore')
    linkedList.Add('China')

    print()
    print ('After adding the items:')
    linkedList.PrintStructure()
    linkedList.Display()
    print()


    linkedList.Remove('China')
    linkedList.Remove('Japan')

    print()
    print ("After removal of the items:")
    linkedList.PrintStructure()

main()
```



```
After adding the items:

Start Index: 3
Tail Index: 2
Next Free Index: 4
Index   Data            Pointer
===============================
1       Japan           2
2       Singapore       0
3       China           1
4                       5
5                       0
Items in order: China  Japan  Singapore

Removed: China
Removed: Japan

After removal of the items:

Start Index: 2
Tail Index: 2
Next Free Index: 1
Index   Data            Pointer
===============================
1                       3
2       Singapore       0
3                       4
4                       5
5                       0
```

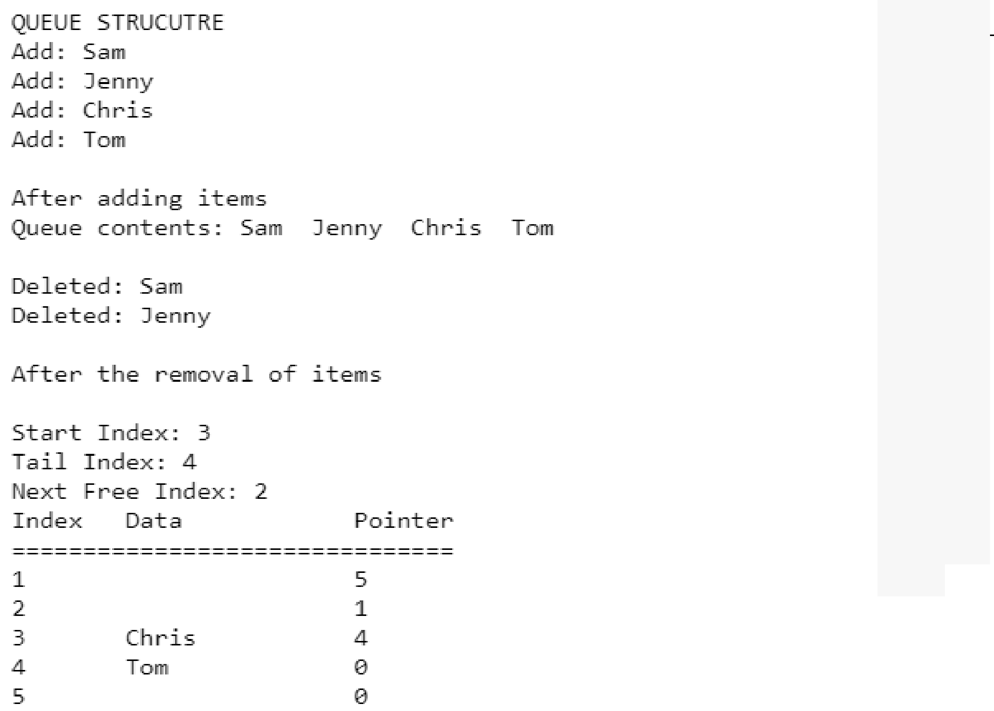


```python
# Task 1.3
class Queue(LinkedStructure):  # inheritance

    def __init__(self):
        LinkedStructure.Initialise(self)   # super().Initialise


    def Add(self, item):    # polymorphism
        if self.IsFull():   # inherited method
            print ('Queue is full. Abort operation!')
        else:
            # update free list
            index = self.NextFree
            self.NextFree = self.Node[index].Pointer

            # add item to queue
            self.Node[index] = ListNode(item, 0)

            if self.Tail == 0:
                self.Start = index
            else:
                self.Node[self.Tail].Pointer = index

            self.Tail = index

```



```python

    def Display(self):     # polymorphism
        if self.IsEmpty(): # inherited method
            print ('Queue is empty!')
        else:
```

```python
# Task 1.4
def main():
    print ("QUEUE STRUCUTRE")
    queue = Queue()

    inF = open('queue.txt', 'r')
```



```
QUEUE STRUCUTRE
Add: Sam
Add: Jenny
Add: Chris
Add: Tom

After adding items
Queue contents: Sam  Jenny  Chris  Tom

Deleted: Sam
Deleted: Jenny

After the removal of items

Start Index: 3
Tail Index: 4
Next Free Index: 2
Index   Data            Pointer
===============================
1                       5
2                       1
3       Chris           4
4       Tom             0
5                       0
```