

Task 1.1

```
1 def readFile(file):
2     inF = open(file, 'r')
3     prevTotal = int(inF.readline().split(',')[1])
4
5     cases = []    # stores number of new cases from 15 Apr to 15 May
6     dates = []
7     for line in inF:
8         date, total = line.rstrip().split(',')
9         total = int(total)
10        num = total - prevTotal
11
12        cases.append(num)
13        dates.append(date)
14
15        prevTotal = total
16
17    inF.close()
18
19    return (cases, dates)
20
21 def format(dates):
22     month = {'04': "April", '05': "May"}
23     dateStr = ''
24     for date in dates:
25         date = date[:2].lstrip('0') + ' ' + month[date[-2:]] + ' ' + '2020'
26         dateStr += date + ', '
27     return (dateStr.rstrip(', '))
28
29 def main():
30     cases, dates = readFile('covid.txt')
31
32     maxNum = minNum = cases[0]
33     maxDates = minDates = [dates[0]]
34
35     for i in range(1, len(cases)):
36         if cases[i] > maxNum:
37             maxNum = cases[i]
38             maxDates = [dates[i]]
39         elif cases[i] == maxNum:
40             maxDates.append(dates[i])
41
42         if cases[i] < minNum:
43             minNum = cases[i]
44             minDates = [dates[i]]
45         elif cases[i] == minNum:
46             minDates.append(dates[i])
47
48     print (f'Highest # cases ({maxNum}) is on {format(maxDates)}')
49     print (f'Lowest # cases ({minNum}) is on {format(minDates)}')
50
51 main()
```

Highest # cases (1426) is on 20 April 2020

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

Task 1.2

```
1 def readFile(file):
2     inF = open(file, 'r')
3     prevTotal = int(inF.readline().split(',')[1])
4
5     cases = [] # stores number of new cases from 15 Apr to 15 May
6     for line in inF:
7         date, total = line.rstrip().split(',')
8         total = int(total)
9         num = total - prevTotal
10
11         cases.append(num)
12         prevTotal = total
13
14     inF.close()
15     return (cases)
16
17 def main():
18     cases = readFile('covid.txt')
19
20     longest = -1
21     day = 1 # 15 Apr
22
23     for i in range(1, len(cases)):
24         if cases[i] >= cases[i-1]:
25             day += 1
26         else:
27             if day > longest:
28                 longest = day
29             day = 1
30
31     if day > longest:
32         longest = day
33
34     print ('Longest ascending streak is', longest, 'days.')
35
36 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!")
```

```
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,
# 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,

```
# 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	COUNTRY2.txt - Notepad	RATE.txt - Notepad
File Edit Format View Help	File Edit Format View Help	File Edit Format View Help
USA	USA,1507773,90113	Belgium,16.4%
Spain	Spain,276505,27563	France,15.4%
Russia	Russia,272043,2537	UK,14.4%
UK	UK,240161, 34466	Italy,14.1%
Brazil	Brazil,233142,15633	Netherlands,12.9%
Italy	Italy,224760,31763	Sweden,12.4%
France	France,179365,27625	Mexico,10.6%
Germany	Germany,176244,8027	Spain,10.0%
Turkey	Turkey,148067,4096	Ecuador,8.2%
Iran	Iran,118392,6937	Canada,7.5%
India	India,88541,2523	Brazil,6.7%
Peru	Peru,88541,2523	Philippines,6.6%
China	China,82941,4633	Indonesia,6.4%
Canada	Canada,75864,5679	Switzerland,6.1%
Belgium	Belgium,54989,9005	USA,6.0%
SaudiArabia	SaudiArabia,52016,302	Iran,5.9%
Mexico	Mexico,45032,4767	China,5.6%
Netherlands	Netherlands,43870,5670	Germany,4.6%
Chile	Chile,41428,421	Japan,4.5%
Pakistan	Pakistan,38799,834	Portugal,4.2%
Ecuador	Ecuador,32763,2688	Turkey,2.8%
Qatar	Qatar,30972,15	India,2.8%
Switzerland	Switzerland,30572,1879	Peru,2.8%
Sweden	Sweden,29677,3674	Pakistan,2.1%
Portugal	Portugal,28810,1203	Thailand,1.9%
Belarus	Belarus,28681,160	Malaysia,1.6%
Singapore	Singapore,27356,22	Bangladesh,1.5%
Bangladesh	Bangladesh,20995,314	Australia,1.4%
Indonesia	Indonesia,17025,1089	Chile,1.0%
Japan	Japan,16237,725	Russia,0.9%
Philippines	Philippines,12305,817	SaudiArabia,0.6%
Australia	Australia,7036,98	Belarus,0.6%
Malaysia	Malaysia,6872,113	Maldives,0.4%
Thailand	Thailand,3025,56	Singapore,0.1%
Maldives	Maldives,1078,4	Qatar,0.0%

Task 3.1

```
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">   <br>
        <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

```
1 # Task 1.1
2 class ListNode:
3     def __init__(self, data, pointer):
4         self.Data = data
5         self.Pointer = pointer
6
7 class LinkedStructure:
8     SIZE = 5
9
10    def Initialise(self):
11        self.Start = 0
12        self.Tail = 0
13        self.NextFree = 1
14
15        self.Node = [None]*(self.SIZE+1)          # array[1..SIZE] of Listnode
16        for i in range(1, self.SIZE):             # set up unused linked list
17            self.Node[i] = ListNode('',i+1)
18        self.Node[self.SIZE] = ListNode('',0)      # Last unused node has pointer 0
19
20    def IsEmpty(self):
21        return (self.Start == 0)
22
23    def IsFull(self):
24        return (self.NextFree == 0)
25
26    def PrintStructure(self):
27        print ('\nStart Index:', self.Start)
28        print ('Tail Index:', self.Tail)
29        print ('Next Free Index:', self.NextFree)
30
31        print ('Index\tData\t\tPointer')
32        print ('='*31)
33
34        for i in range(1, len(self.Node)):
35            node = self.Node[i]
36            print ('%-8d%-14s%3d' % (i, node.Data, node.Pointer))
37
38    def Display(self):
39        if self.IsEmpty():
40            print ('Linked list is empty!')
41        else:
42            print ('Items in order:', end = ' ')
43            curr = self.Start
44            while curr != 0:
45                node = self.Node[curr]
46                print (node.Data + ' ', end = ' ')
47                curr = node.Pointer
48            print()
49
```

```

50     def Remove(self, item):
51         if self.IsEmpty():
52             print ('Cannot delete from empty list!')
53         else:
54             # search for the node to be deleted
55             curr = self.Start
56             prev = 0
57             while curr != 0 and item > self.Node[curr].Data:
58                 prev = curr
59                 curr = self.Node[curr].Pointer
60
61             # node not found
62             if curr == 0 or item < self.Node[curr].Data:
63                 print (item, 'not found in the list')
64
65             else: # node found
66                 # update previous node's pointer
67                 nextPointer = self.Node[curr].Pointer
68                 if prev == 0:
69                     self.Start = nextPointer
70                 else:
71                     self.Node[prev].Pointer = nextPointer
72
73                 # if node is the only/last node, update tail pointer
74                 if nextPointer == 0:
75                     self.Tail = 0
76
77                 print ('Removed:', item)
78
79                 # update free list
80                 self.Node[curr].Data = ''
81                 self.Node[curr].Pointer = self.NextFree
82                 self.NextFree = curr
83
84     def Add(self, item):
85         if self.IsFull():
86             print ('List is full. Abort operation!')
87         else:
88             # update free list
89             index = self.NextFree
90             self.NextFree = self.Node[index].Pointer
91
92             # find insertion point
93             curr = self.Start
94             prev = 0
95             while curr != 0 and item > self.Node[curr].Data:
96                 prev = curr
97                 curr = self.Node[curr].Pointer
98
99             # add new node to the list
100            self.Node[index] = ListNode(item, curr)
101
102            if prev == 0:
103                self.Start = index
104            else:
105                self.Node[prev].Pointer = index
106
107            # if new node is the only/last node, update tail pointer
108            if curr == 0:
109                self.Tail = index
110

```

Task 4.2

```
1 # Task 1.2
2 def main():
3
4     linkedList = LinkedStructure()
5     linkedList.Initialise()
6
7     linkedList.Add('Japan')
8     linkedList.Add('Singapore')
9     linkedList.Add('China')
10
11     print()
12     print('After adding the items:')
13     linkedList.PrintStructure()
14     linkedList.Display()
15     print()
16
17
18     linkedList.Remove('China')
19     linkedList.Remove('Japan')
20
21     print()
22     print("After removal of the items:")
23     linkedList.PrintStructure()
24
25 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

Task 4.3

```
1  # Task 1.3
2  class Queue(LinkedStructure): # inheritance
3
4      def __init__(self):
5          LinkedStructure.Initialise(self) # super().Initialise
6
7
8      def Add(self, item): # polymorphism
9          if self.IsFull(): # inherited method
10             print('Queue is full. Abort operation!')
11         else:
12             # update free list
13             index = self.NextFree
14             self.NextFree = self.Node[index].Pointer
15
16             # add item to queue
17             self.Node[index] = ListNode(item, 0)
18
19             if self.Tail == 0:
20                 self.Start = index
21             else:
22                 self.Node[self.Tail].Pointer = index
23
24             self.Tail = index
25
26
27      def Display(self): # polymorphism
28          if self.IsEmpty(): # inherited method
29             print('Queue is empty!')
30         else:
31             print('Queue contents:', end = ' ')
32             curr = self.Start
33             while curr != 0:
34                 node = self.Node[curr]
35                 print(node.Data + ' ', end = ' ')
36                 curr = node.Pointer
37             print()
38
39
40      def Remove(self):
41          if self.IsEmpty():
42             return('Cannot delete from empty queue!')
43         else:
44             returnValue = self.Node[self.Start].Data
45
46             # remove item from queue
47             index = self.Start
48             self.Start = self.Node[index].Pointer
49             if self.Start == 0:
50                 self.Tail = 0
51
52             # update free list
53             self.Node[index].Data = ''
54             self.Node[index].Pointer = self.NextFree
55             self.NextFree = index
56
57             return returnValue
```

Task 4.4

```
1  # Task 1.4
2  def main():
3      print ("QUEUE STRUCUTRE")
4      queue = Queue()
5
6      inF = open('queue.txt', 'r')
7
8      # Add fie contents to queue
9      for item in inF:
10         item = item.rstrip()
11         print ("Add:", item)
12         queue.Add(item)
13
14     inF.close()
15
16     # output queue and array contents
17     print ('\nAfter adding items ')
18     #queue.PrintStructure()
19     queue.Display()
20     print ()
21
22     # remove and output two items from queue
23     for i in range(2):
24         print ('Deleted:', queue.Remove())
25
26     print ('\nAfter the removal of items')
27     queue.PrintStructure()
28     #queue.Display()
29
30 main()
```

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