#Task 1.1

```
def linear_search_outliers(Data, Maximum):
    outliers = []
    for i in range(len(Data)):
        if Data[i] > Maximum:
            outliers.append(i)
    return outliers
```

#Task 1.2

```
# Copy and paste the code from Task2_2.txt to initialize the data list
Data = [
    51.1, 77.3, 82.4, 97.5, 104.6, 69.8, 105.2, 95.7, 62.3, 109.1,
    76.9, 81.5, 68.2, 53.9, 59.6, 88.4, 85.0, 55.6, 67.7, 86.3,
    89.9, 75.0, 79.2, 52.4, 99.8, 92.1, 92.3, 91.2, 93.7, 103.0,
    107.5, 94.6, 60.1, 100.9, 73.5, 103.5, 98.4, 51.6, 78.7, 74.2,
    101.4, 106.8, 63.7, 72.8, 87.6, 58.8, 66.4, 56.1, 84.3, 61.9
]

maximum_value = 90.0
outlier_indices = linear_search_outliers(Data, maximum_value)

print(outlier_indices)
print()

filtered_data_list = []
for i in range(len(Data)):
    if i not in outlier_indices:
        filtered_data_list.append(Data[i])

print(filtered_data_list)
```

#Task 1.3

```
def quicksort(Data):
    if len(Data) > 1:
        pivot = Data[0] #fisrt item as pivot
        smaller = []
        larger = []

        for item in Data[1:]:
            if item < pivot:
                smaller.append(item)
            else:
                larger.append(item)

        return quicksort(smaller) + [pivot] + quicksort(larger)
    else:
        if len(Data) == 1:
            return [Data[0]]
        else:
            return []
```

#Task 1.4

```
sorted_data = quicksort(filtered_data_list)

# Display the sorted numerical data
print("Sorted Data without Outliers:", sorted_data)
```

```
#task2.1
def hash_function(ISBN):
    total = 0
    for char in ISBN:
        total += ord(char)
    remainder = total % 53
    return remainder
print(hash_function("0205080057"))

#task 2.2
class Book_Record():
    def __init__(self,ISBN,Title,Author,Due_Date):
        self.ISBN = ISBN
        self.Title = Title
        self.Author = Author
        self.Due_Date = Due_Date

    def Get_ISBN(self):
        return self.ISBN

    def Get_Title(self):
        return self.Title

    def Get_Author(self):
        return self.Author

    def Get_Due_Date(self):
        return self.Due_Date

    def Set_Due_Date(self,new_due_date):
        self.Due_Date = new_due_date
```

```
     def to_string(self):
        return self.Get_ISBN() +', '+ self.Get_Title() + ', ' +
self.Get_Author()+', '+self.Get_Due_Date()


#task2.3
hta = [Book_Record('','','','') for i in range(53)]
#array that stores up to 53 Book_Record objects


file = open("Task2_3.txt",'r')
for line in file:
    line = line.strip().split(',')
    hash_value = hash_function(line[0])
    index = hash_value
    full = False

    while hta[index].Get_ISBN() != '':
        index  = (index+1) % len(hta)

        if index == hash_value: #hash table is full
            full = True
            break

    if full == False: #add to hash table array if not full
        hta[index] = Book_Record(line[0],line[1],line[2],line[3])
```

```
#task2.4
def search_book_record(hta):
    isbn = input("enter ISBN: ")
    hash_value = hash_function(isbn)
    index = hash_value

    while True:
        if hta[index].Get_ISBN() == isbn: #found
            return hta[index].to_string()

        if hta[index].Get_ISBN() == '': #not full and not found
            return "Book not on loan"

        index = (index+1) % len(hta)

        if index == hash_value: #full and not found
            return "Book not on loan"

#task2.5
print(search_book_record(hta))
print(search_book_record(hta))
```

task2.6

```
def update_book_record(hta):
    isbn = input("enter ISBN: ")
    due_date = input("new due date: ")
    index = hash_function(isbn)

    while True:
        if hta[index].Get_ISBN() == isbn:
            hta[index].Set_Due_Date(due_date)
            break
        else:
            index = (index+1) % len(hta)
```

#Task 2.7

```
update_book_record(hta)


def display(hta):
    index = 0

    print("Index".ljust(8) + "ISBN".ljust(12) + "Title".ljust(40) +
"Author".ljust(30) + "Due_Date")

    for record in hta:
        if record is not None:
            print(str(index).ljust(8) + record.Get_ISBN().ljust(12) +
record.Get_Title().ljust(40) + record.Get_Author().ljust(30) +
record.Get_Due_Date())
        else:
            print(str(index))
        index += 1


display(hta)
```

Task 3.1

```
class Node:  # [2]
    def __init__(self, data):
        self.data = data
        self.next = None


class Stack:
    def __init__(self):  #[1]
        self.top = None # pointer to Node object

    def push(self, data):  #[2]
        temp = self.top
        self.top = Node(data)
        self.top.next = temp

    def pop(self):   #[2]
        temp = self.top
        self.top = self.top.next
        return temp

    def to_string(self):  #[3]
        result = []
        curr = self.top
        while curr!=None:
            result.append(curr.data)
            curr = curr.next
        return ", ".join(result)
```

```
Output:
ship, yacht, train, car, bus, plane
ship
yacht
train
```

```
# Task 3.2
lst = ['plane','bus','car','train','yacht','ship']

stack = Stack()
for ele in lst:
    stack.push(ele)   #1
print(stack.to_string())  #1
print(stack.pop().data)
print(stack.pop().data)
print(stack.pop().data)  #1

# Task 3.3
class Queue:
    def __init__(self):  #1
        self.head = None # pointer to Node object

    def enqueue(self, data): # add to end of queue
        if self.head == None: #queue is empty
            self.head = Node(data)  # 1

        else: # add to end
            prev = self.head
            curr = self.head.next
            while curr!=None:
                prev = curr
                curr = curr.next     #2

            prev.next = Node(data)    #1

    def dequeue(self):  # remove from front of queue
        temp = self.head
```

```
        self.head = self.head.next
        return temp    #2

    def to_string(self):
        result = []
        curr = self.head  #1

        while curr!=None:
            result.append(curr.data)
            curr = curr.next    #1

        return ", ".join(result)

# Task 3.4
lst = ['plane','bus','car','train','yacht','ship']
q = Queue()
for ele in lst:
    q.enqueue(ele)   #1
print(q.to_string())   #1
print(q.dequeue().data)
print(q.dequeue().data)
print(q.dequeue().data)  #1
```

Output:

```
plane, bus, car, train, yacht, ship
plane
bus
car
```

```
# Task 4.1 #[5]
import sqlite3
connection = sqlite3.connect("MerlionThemePark.db")  #1
sql = '''CREATE TABLE Ticket (
    tDate TEXT PRIMARY KEY,
    dayOfWeek TEXT,
    unitPrice INTEGER,
    totQuan INTEGER,
    availQuan INTEGER )'''  #2
connection.execute(sql)

sql2 = '''CREATE TABLE "Sale" (
    sID INTEGER PRIMARY KEY AUTOINCREMENT,
    tDate TEXT,
    quan INTEGER,
    totalPrice INTEGER,
    FOREIGN KEY(tDate) REFERENCES Ticket(tDate) )'''  #2
connection.execute(sql2)
connection.close()

# Task 4.2 #[5]
import sqlite3
connection = sqlite3.connect("MerlionThemePark.db")

infile = open("TICKET.txt")
lines = infile.readlines()  #1

for line in lines:
    line = line.strip().split(',') #1
    connection.execute("INSERT INTO Ticket(tDate, dayOfWeek, unitPrice,
totQuan, availQuan) " + "VALUES(?,?,?,?,?)",
(line[0],line[1],line[2],line[3],line[4])) #2
```

```
connection.commit()
connection.close() #1

# Task 4.3 #[5]+[1]
import sqlite3
connection = sqlite3.connect("MerlionThemePark.db")

cursor = connection.execute("SELECT * FROM Ticket")
rows = cursor.fetchall() #1

month = input("Enter month:")
#2023-11-01,Wednesday,40,100,100
print("{:<14} {:<14} {:>6} {:>6} {:>10}".format("Date","Day of
Week","Price","Total","Available"))  #1

for row in rows:
    date = row[0]
    date = date.split('-')  #1
    if date[1]==month: #1
        print("{:<14} {:<14} {:>6} {:>6}
{:>10}".format(row[0],row[1],row[2],row[3],row[4]))  #1

Correct Output shown   #1
```

```
# Task 4.4 #[14]+[1]
from flask import *
import sqlite3

app = Flask(__name__)  # Flask class constructor

@app.route('/')
def index():
    return render_template('index.html')  #1

@app.route('/ticket', methods=["POST"])
def ticket():
    month = request.form['month']

    connection = sqlite3.connect("MerlionThemePark.db")
    cursor = connection.execute("SELECT * FROM ticket")
    rows = cursor.fetchall()  #1
    result = []

    for row in rows:
        date = row[0]
        date = date.split('-')  # (yr, mth, day)

        if date[1]==month:
            result.append(row)  #1

    connection.close()
    return render_template('ticket.html',result=result, month=month) #1

@app.route('/buy', methods=["POST"])
def buy():
    month = request.form['month']
```

```
    day = request.form['day']
    quan = request.form['quan']

    connection = sqlite3.connect("MerlionThemePark.db")
    cursor = connection.execute("SELECT * FROM ticket")
    rows = cursor.fetchall()
    total = 0
    remain = 0
    date = ""

    # 2023-11-01,Wednesday,40,100,100
    # 2023-08-07,2023-11-01,10,400
    for row in rows:
        date = row[0]
        yr, mth, d = date.split('-')  # (yr, mth, day)

        if mth==month and int(d) == int(day): #1

            if int(quan) <= int(row[4]): # valid quan
                total += int(row[2]) * int(quan)
                remain = int(row[4]) - int(quan)  #1
                break
            else:
                return "<h1>Insufficient quantity. Transaction
unsuccessfully.</h1>"

    date = "2023-" + month + "-" + day
    connection.execute("INSERT INTO Sale (tDate, quan, totalPrice) VALUES
(?,?,?)",(date,quan,total))  #1

    # UPDATE table_name
    # SET column1 = value1, column2 = value2, ...
```

```
    # WHERE condition;
    sql = "UPDATE Ticket SET availQuan = ? WHERE tDate = ?"
    connection.execute(sql, (remain, date))  #1

    connection.commit()
    connection.close()

    msg = "<p>Ticket date " + date + ", quantity " + quan + ', total price
$' + str(total)
    return '<h1>Your transaction is successful.</h1>' + msg  #1

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

 # index.html #[2]

```
<h1>Merlion Theme Park Ticket</h1>
<form action="/ticket" method = "POST">
     Date:
     Month (in number): <input type="TEXT" name = "month" size="2">
     <p> <input type="SUBMIT">
</form>
```

 # ticket.html #[4]

```
<html>
<head>
<style>
     table, th, td {
       border: 1px solid black;
       border-collapse: collapse;
     }
</style>
</head>
```

```
<body>
<h1>Merlion Theme Park Ticket</h1>

<form action="/buy" method="POST">
Date<br>
Year: 2023<br>
Month: <input type="TEXT" name = "month" size="2">
Enter Day: <input type="TEXT" name = "day" size="2">
<p>
Quantity of tickets to buy: <input type="TEXT" name = "quan" size="3"><br>
<input type="SUBMIT" value = "BUY" ><br>
<p>
<table>
      <tr><th>Date</th><th>Day in week</th><th>Unit price
      S($)</th><th>Available</th><tr>
      {% for row in result %}
            <tr>
            <td>{{ row[0] }}</td>
            <td>{{ row[1] }}</td>
            <td>{{ row[2] }}</td>
            <td>{{ row[4] }}</td>
            </tr>
      {% endfor %}
</table></form></body></html>
```