Task 1 Soln

```
# Task 1.1
f = open('MINEFIELD.txt', 'r')

n = int(f.readline().strip())

field = []
for i in range(n):
    field.append(['.'] * n)
field[n // 2][n // 2] = 'S'

mine_list = []
line = f.readline()
while line != '':
    x, y = line.split(',')
    x = int(x)
    y = int(y)
    mine_list.append((x, y))
    field[x][y] = 'M'
    line = f.readline()

f.close()

print("Mine Field:")
for i in range(n):
    for j in range(n):
        print(field[i][j], end = ' ')
    print()
```

```
Mine Field:
. . M . . . .
. . . M . . .
. . . . . . M
. . . S M . .
. . . . . . .
. . . M . . .
. . . . . . .
```

```
# Task 1.2
import random
x = n // 2
y = n // 2
stop = False
win = False
steps = ''
moves = ['UP', 'DOWN', 'LEFT', 'RIGHT']
```

```
while not stop:
    move = random.randint(0, 3)

    if move == 0: # move up
        x -= 1
    elif move == 1: # move down
        x += 1
    elif move == 2: # move left
        y -= 1
    else: # move right
        y += 1

    steps = steps + moves[move] + ' '
    if field[x][y] == 'M': # 1 mark on condition to lose
        stop = True
    elif x == 0 or x == n - 1 or y == 0 or y == n - 1:
        stop = True
        win = True
        field[x][y] = 'P'
    else:
        field[x][y] = 'P'

print('STEPS:', steps)
for i in range(n):
    for j in range(n):
        print(field[i][j], end = ' ')
    print()
if win:
    print("WIN! You walked to the boundary!")
else:
    print('LOSE! You stepped onto the mine!')
```

```
STEPS: LEFT UP DOWN DOWN RIGHT RIGHT RIGHT DOWN RIGHT
. . M . . . .
. . . M . . .
. . P . . . M
. . P S M . .
. . P P P P .
. . . M . P P
. . . . . . .
WIN! You walked to the boundary!
```

Task 2 Soln:

```
# Task 2.1
def read_csv(filename):
    books_file = open(filename, "r")
    book_str = books_file.read()
    book_list = book_str.split("\n")
    array = []
    for book in book_list:                          # loop to extract
        title, author, year = book.split(",")       # split the comma
        array.append([title, author, year])
    books_file.close()
    return array

books_array = read_csv("booklist.csv")
print(len(books_array))
```

```
import csv
# Task 2.1 (alternative by csv package)
def read_csv(filename):
    books_file = open(filename, "r", encoding="utf-8")
    book_list = csv.reader(books_file, delimiter=",")
    array = []
    for book in book_list:          # loop to extract
        title, author, year = book      # split the comma
        array.append([title, author, year])
    books_file.close()
    return array

books_array = read_csv("booklist.csv")
print(len(books_array))
print(books_array)
```

```
18
[['White Fang', 'Jack London', '1906'], ['The Wind in the Willows',
'Kenneth Grahame', '1908'], ['Moby Dick', 'Herman Melville', '1851'],
['Jane Eyre', 'Charlotte Bronte', '1847'], ['The Picture of Dorian
Gray', 'Oscar Wilde', '1890'], ['The Three Musketeers', 'Alexandre
Dumas', '1844'], ['Persuasion', 'Jane Austen', '1817'], ['Dream of the
Red Chamber', 'Cao Xueqin', '1791'], ['Little Women', 'Louisa May
Alcott', '1868'], ['The Phantom of the Opera', 'Gaston Leroux', '1909'],
['Water Margin', 'Shi Naian', '1450'], ['A Christmas Carol', 'Charles
Dickens', '1843'], ['One Hundred Years of Solitude', 'Gabriel Garcia
Marquez', '1967'], ['Nineteen Eighty-Four', 'George Orwell', '1949'],
['Journey to the West', 'Wu Chengen', '1592'], ['Romance of the Three
Kingdoms', 'Luo Guanzhong', '1522'], ['Fahrenheit 451', 'Ray Bradbury',
'1953'], ['War and Peace', 'Leo Tolstoy', '1867']]
```

```
# Task 2.2
def bubble(array, sort_key):
    sort_dict = {"title": 0, "author": 1, "year": 2}
    if sort_key not in sort_dict:
        return -1                    # -1 return if invalid
    s = sort_dict[sort_key]          # convert sort_key to index
    length = len(array)
    for i in range(length-1,0,-1):
        for j in range(i):           # nested loop for bubble
            if array[j][s] > array[j+1][s]: # compare adjacent
                array[j], array[j+1] = array[j+1], array[j]
    return array

print(bubble(books_array,"title"))
print(bubble(books_array,"ISBN"))
```

```
[['A Christmas Carol', 'Charles Dickens', '1843'], ['Dream of the Red
Chamber', 'Cao Xueqin', '1791'], ['Fahrenheit 451', 'Ray Bradbury',
'1953'], ['Jane Eyre', 'Charlotte Bronte', '1847'], ['Journey to the
West', 'Wu Chengen', '1592'], ['Little Women', 'Louisa May Alcott',
'1868'], ['Moby Dick', 'Herman Melville', '1851'], ['Nineteen Eighty-
Four', 'George Orwell', '1949'], ['One Hundred Years of Solitude',
'Gabriel Garcia Marquez', '1967'], ['Persuasion', 'Jane Austen',
'1817'], ['Romance of the Three Kingdoms', 'Luo Guanzhong', '1522'],
['The Phantom of the Opera', 'Gaston Leroux', '1909'], ['The Picture of
Dorian Gray', 'Oscar Wilde', '1890'], ['The Three Musketeers',
'Alexandre Dumas', '1844'], ['The Wind in the Willows', 'Kenneth
Grahame', '1908'], ['War and Peace', 'Leo Tolstoy', '1867'], ['Water
Margin', 'Shi Naian', '1450'], ['White Fang', 'Jack London', '1906']]
-1
```

```
# Task 2.3
def merge(array, sort_key):
    sort_dict = {"title": 0, "author": 1, "year": 2}
    if sort_key not in sort_dict:
        return -1
    s = sort_dict[sort_key]

    if len(array)<2:
        return array


    mid = len(array) // 2
    left = merge(array[:mid],sort_key)
    right = merge(array[mid:],sort_key)
    # split the array in half
    # run merge sort on each recursively

    merged = []
    while len(left) and len(right): # repeat until 1 empty
        if left[0][s] <= right[0][s]: # take the smaller item
            merged = merged + [left.pop(0)]
```

```
        else:
            merged = merged + [right.pop(0)]

    merged = merged + left + right
    # merge after either L/R empty
    for i in range(len(array)):
        array[i] = merged[i]
    return array

print(merge(books_array,"author"))
print(merge(books_array,"year"))
```

```
[['The Three Musketeers', 'Alexandre Dumas', '1844'], ['Dream of the Red
Chamber', 'Cao Xueqin', '1791'], ['A Christmas Carol', 'Charles
Dickens', '1843'], ['Jane Eyre', 'Charlotte Bronte', '1847'], ['One
Hundred Years of Solitude', 'Gabriel Garcia Marquez', '1967'], ['The
Phantom of the Opera', 'Gaston Leroux', '1909'], ['Nineteen Eighty-
Four', 'George Orwell', '1949'], ['Moby Dick', 'Herman Melville',
'1851'], ['White Fang', 'Jack London', '1906'], ['Persuasion', 'Jane
Austen', '1817'], ['The Wind in the Willows', 'Kenneth Grahame',
'1908'], ['War and Peace', 'Leo Tolstoy', '1867'], ['Little Women',
'Louisa May Alcott', '1868'], ['Romance of the Three Kingdoms', 'Luo
Guanzhong', '1522'], ['The Picture of Dorian Gray', 'Oscar Wilde',
'1890'], ['Fahrenheit 451', 'Ray Bradbury', '1953'], ['Water Margin',
'Shi Naian', '1450'], ['Journey to the West', 'Wu Chengen', '1592']]

[['Water Margin', 'Shi Naian', '1450'], ['Romance of the Three
Kingdoms', 'Luo Guanzhong', '1522'], ['Journey to the West', 'Wu
Chengen', '1592'], ['Dream of the Red Chamber', 'Cao Xueqin', '1791'],
['Persuasion', 'Jane Austen', '1817'], ['A Christmas Carol', 'Charles
Dickens', '1843'], ['The Three Musketeers', 'Alexandre Dumas', '1844'],
['Jane Eyre', 'Charlotte Bronte', '1847'], ['Moby Dick', 'Herman
Melville', '1851'], ['War and Peace', 'Leo Tolstoy', '1867'], ['Little
Women', 'Louisa May Alcott', '1868'], ['The Picture of Dorian Gray',
'Oscar Wilde', '1890'], ['White Fang', 'Jack London', '1906'], ['The
Wind in the Willows', 'Kenneth Grahame', '1908'], ['The Phantom of the
Opera', 'Gaston Leroux', '1909'], ['Nineteen Eighty-Four', 'George
Orwell', '1949'], ['Fahrenheit 451', 'Ray Bradbury', '1953'], ['One
Hundred Years of Solitude', 'Gabriel Garcia Marquez', '1967']]
```

```
# Task 2.4
def reverse(array):
    length = len(array)
    mid = length // 2
    for i in range(mid): # using a loop
        array[i], array[length-1-i] = array[length-1-i], array[i]
        # swap to reverse
    return array

print(reverse([1,3,5,2,4]))
print(reverse([1,9,6,4]))
```

```
[4, 2, 5, 3, 1]
[4, 6, 9, 1]
```

```
# Task 2.5
arr = read_csv("newbooks.csv")
merge(arr,"year") # bubble/merge using year as key
reverse(arr) # reverse AFTER sorting

new_csv = open("YEAR_name_ct.csv", "w")
# open with "w" (must close at end)

book_str = []
for book in arr:
    book_str.append(",".join(book))
# re-combine with commas
ret_str = "\n".join(book_str)
new_csv.write(ret_str)
new_csv.close()
```

```
# evidence from csv:
Animal Farm,George Orwell,1945
Of Mice and Men,John Steinbeck,1937
To Kill a Mockingbird,Harper Lee,1960
The Catcher in the Rye,J. D. Salinger,1951
The Adventures of Tom Sawyer ,Mark Twain,1876
Monty Python's Big Red Book,Graham Chapman,1971
The Strange Case of Dr. Jekyll & Mr. Hyde,Robert Louis Stevenson,1886
The War of the Worlds,H. G. Wells,1898
Wuthering Heights,Emily Bronte,1847
Dracula,Bram Stoker,1897
Pride & Prejudice,Jane Austen,1813
The Great Gatsby,F. Scott Fitzgerald,1925
```

Task 3 Soln:

```
# Task 3.1
class Node:
    def __init__(self, data, next):
        self.data = data
        self.next = next

class LinkedList:
    def __init__(self):
        self.head = None
        self.size = 0

    def to_String(self):
        items = []
        probe = self.head
        while probe != None:
            items.append(probe.data)
            probe = probe.next
        return ', '.join(items)

    def insert(self, word, p):
        if p == 1 or self.size == 0: # condition to add at the front
            self.head = Node(word, self.head)
        else:
            if p > self.size: # special case
                p = self.size + 1
            probe = self.head

            for i in range(1, p - 1):
                probe = probe.next
            probe.next = Node(word, probe.next)
        self.size += 1

    def delete(self, p):
        if p == 1 or self.size == 1: # condition to delete at the front
            self.head = self.head.next
        else:
            if p > self.size: # special case
                p = self.size
            probe = self.head
            for i in range(1, p - 1):
                probe = probe.next
            probe.next = probe.next.next
        self.size -= 1

    def search(self, word):
        found = False
        probe = self.head

        while not found and probe != None:
            if probe.data == word: # correct case when found
```

```
                found = True
            else:
                probe = probe.next

        return found

# test design with inserting at the front, normal p value, p > size

ll = LinkedList()
ll.insert('apple', 5) # add to an empty linked list, and p > size
ll.insert('durian', 3) # add to the end of the linked list
ll.insert('pear', 2) # add item in between
print('items:',ll.to_String())

# test for found and not found
print(ll.search('apple'))
print(ll.search('carrot'))
```

```
items: apple, pear, durian
True
False
```

```
# Task 3.2
class Stack(LinkedList):
    def push(self, word):
        Stack.insert(self, word, 1)

    def pop(self):
        Stack.delete(self, 1)

s = Stack()
s.push('apple')
s.push('pear')
s.push('carrot')
s.pop()
print(s.to_String())
```

```
pear, apple
```

```
# Task 3.3
class Queue(LinkedList):
    def enqueue(self, word):
        Queue.insert(self, word, self.size + 1)

    def dequeue(self):
        Queue.delete(self, 1)

q = Queue()
q.enqueue('apple')
q.enqueue('pear')
q.enqueue('carrot')
q.dequeue()
print(q.to_String())
```

```
pear, carrot
```

Task 4 soln:

```
<!--Task4_1.htm -->

<!DOCTYPE html>
<html>
<head><title>Menu</title>
<link rel="stylesheet" type="text/css"
href="{{ url_for('static', filename='styles.css') }}">
</head>
<body>
<p>Menu</p>
<p><a href="{{url_for("task4_2")}}">Student health records</a></p>
<p><a href="{{url_for("task4_3")}}">Health record statistics</a></p>
<p><a href="{{url_for("task4_4")}}">Add health record</a></p>

</body>
</html>
```

```
### Task4_1
@app.route('/', methods=['GET'])
def task4_1():
    return render_template('task4_1.html')

@app.route('/task4_2', methods=['GET'])
def task4_2():
    pass

@app.route('/task4_3', methods=['GET'])
def task4_3():
    pass
```

Menu

List All Student Health Records

Health Record Statistic

Add Health Record

```
#Task4_2.sql

SELECT student.name, student.gender, StudentHealthRecord.weight,
StudentHealthRecord.height
FROM student LEFT OUTER JOIN StudentHealthRecord
ON student.studentID = StudentHealthRecord.studentid
ORDER BY student.gender, student.name DESC
```

```
<!--Task4_2.htm -->

<!DOCTYPE html>
<html>
<head><title>Student Health Records</title>
<link rel="stylesheet" type="text/css"
href="{{ url_for('static', filename='styles.css') }}">
</head>
<body>
<p>Student Health Records</p>
<table>
<tr><th>Name</th><th>Gender</th><th>Weight</th><th>Height</th></tr>
{% if results|length > 0 %}
       {% for item in results  %}
    <tr>
        <td>{{ item.name }}</td>
             <td>{{ item.getGender() }}</td>
             <td>{{ item.getWeight() }}</td>
             <td>{{ item.getHeight() }}</td>
    </tr>
       {% endfor %}
{%else%}
    <tr>
        <td colspan="2">No logs</td>
    </tr>
{%endif%}
</table>
<p><a href="{{url_for("task4_1")}}">Back to Menu</a></p>
</body>
</html>

### Task4_2
@app.route('/all')
def task4_2():
       sql="select  student.name, student.gender,
StudentHealthRecord.weight,StudentHealthRecord.height from student left outer join
StudentHealthRecord on student.studentID = StudentHealthRecord.studentid  order by
name"
       db = sqlite3.connect('students.db')
       db.row_factory = sqlite3.Row
       cursor = db.execute(sql)
       all_rows = cursor.fetchall()
       cursor.close()
       db.close()
       listx=[]
       for row in all_rows:
            s=Student(row["name"], row["gender"], row["weight"],row["height"])
            listx.append(s)
       return render_template('task4_2.html', results=listx)
```

Student Health Records

Name	Gender	Weight	Height
Alex	M	51.0	1.75
Arlo	M	55.0	1.65
Ella	F	46.0	1.7
Isla	F	48.0	1.68
June	F	50.0	1.75
Kai	M	None	None
Leo	M	60.0	1.73
Nyla	F	None	None
Vera	F	50.0	1.8
Zane	M	None	None

Back to Menu

#Task4_3.sql

```
SELECT  gender, COUNT(*), AVG(weight), AVG(height) FROM student
INNER JOIN StudentHealthRecord ON
student.StudentID=StudentHealthRecord.StudentID
GROUP BY gender
```

Alternatively …

```
SELECT  COUNT(*) FROM student where gender='M'
SELECT  COUNT(*) FROM student where gender='F'
SELECT  AVG(weight) FROM student INNER JOIN StudentHealthRecord ON
student.StudentID=StudentHealthRecord.StudentID
WHERE gender='M'
SELECT  AVG(weight) FROM student INNER JOIN StudentHealthRecord ON
student.StudentID=StudentHealthRecord.StudentID
WHERE gender='F'
SELECT  AVG(height) FROM student INNER JOIN StudentHealthRecord ON
student.StudentID=StudentHealthRecord.StudentID
WHERE gender='M'
SELECT  AVG(height) FROM student INNER JOIN StudentHealthRecord ON
student.StudentID=StudentHealthRecord.StudentID
WHERE gender='F'
```

```
<!--Task4_3.htm -->

<!DOCTYPE html>
<html>
<head><title>Health Record Statistics</title>
<link rel="stylesheet" type="text/css"
href="{{ url_for('static', filename='styles.css') }}">
</head>
<body>
<p>Health Record Statistics</p>
<table>
<tr><th>Attributes</th><th>Male</th><th>Female</th></tr>
{% if results|length > 0 %}
       {% for item in results  %}
    <tr>
        <td>{{ item.getAttribute() }}</td>
             <td>{{ item.getMale() }}</td>
             <td>{{ item.getFemale() }}</td>
    </tr>
       {% endfor %}
{%else%}
    <tr>
        <td colspan="3">No logs</td>
    </tr>
{%endif%}
</table>
<p><a href="{{url_for("task4_1")}}">Back to Menu</a></p>
</body>
</html>

### Task4_3
@app.route('/statistics', methods=['GET'])
def task4_3():
       db = sqlite3.connect('students.db')
       db.row_factory = sqlite3.Row
       sql="select  gender as gender, count(*) as cnt, avg(weight) as wt,
avg(height) as ht from student left outer join StudentHealthRecord on
student.StudentID=StudentHealthRecord.StudentID group by gender"
       cursor = db.execute(sql)
       all_rows = cursor.fetchall()
       cursor.close()
       db.close()
       listx=[]
       numberRec = Record("Number")
       weightRec = Record("Avg Weight")
       heightRec = Record("Avg Height")

       for row in all_rows:
             if row["gender"]=="M":
                    numberRec.setMale(row["cnt"])
                    weightRec.setMale(row["wt"])
                    heightRec.setMale(row["ht"])
             else:
                    numberRec.setFemale(row["cnt"])
                    weightRec.setFemale(row["wt"])
                    heightRec.setFemale(row["ht"])
       listx.append(numberRec)
       listx.append(weightRec)
       listx.append(heightRec)
       return render_template('task4_3.html', results=listx)
```

Health Record Statistics

Attributes	Male	Female
Number	5	5
Avg Weight	55.33	48.50
Avg Height	1.71	1.73

Back to Menu

```
#Task4_4.sql

INSERT INTO Student(Name, Gender) VALUES('Helen','F')
##Assumming the studentID is 12
INSERT INTO StudentHealthRecord (StudentID, Weight, Height) VALUES (12, 48.7, 1.72)
```

```
<!--Task4_4.html -->

<!DOCTYPE html>
<html>
<head><title>Add Health Record</title>
<link rel="stylesheet" type="text/css"
href="{{ url_for('static', filename='styles.css') }}">
</head>
<body>
<p>Add Health Record</p>
<form method="POST" action="/add" >
    <p><label for="name" >Name: </label><input type="text" value="" name="name"
id="name" ></p>
       <p><label for="gender" >Gender: </label><input type="radio" value="M"
name="gender" id="gender" >Male</input><input type="radio" value="F" name="gender"
id="gender" >Female</input></p>
       <p><label for="weight" >Weight: </label><input type="text" value=""
name="weight" id="weight" ></p>
       <p><label for="height" >Height: </label><input type="text" value=""
name="height" id="height" ></p>
    <p><input type="submit" name="action" value="Add" ></p>
</form>
<p><a href="{{url_for("task4_1")}}">Back to Menu</a></p>
</body>
</html>

### Task4_4
@app.route('/add', methods=['GET', 'POST'])
def task4_4():
       if request.method=='GET':
              return render_template('task4_4.html')
       if 'action' in request.form:
              action = request.form['action']
              name = request.form['name']
              gender = request.form['gender']
              weight = request.form['weight']
              height = request.form['height']
       if action == 'Add':
              try:
                     db = sqlite3.connect('students.db')
                     cur = db.cursor()
```

```
                    cur.execute("Insert into Student(Name, Gender)
values(?,?)",(name,gender))
                    studentID = cur.lastrowid
                    cur = db.execute("INSERT INTO StudentHealthRecord (StudentID,
Weight, Height) VALUES (?, ?,?)",(studentID,weight,height))
                    db.commit()
                    cur.close()
                    db.close()
                    return render_template('Task4_4k.html', msg="Added
successfully")
             except:
                    if db:
                           db.close()
                           return render_template('Task4_4k.html',  msg="Add Error")
                    else:
                           return redirect(url_for('task4_1'))
      else:
             result_msg=''
             return redirect(url_for('task4_1'))
```

```
<!--Task4_5.htm not required -->

<!DOCTYPE html>
<html>
<head><title>Add Health Record</title>
<link rel="stylesheet" type="text/css"
href="{{ url_for('static', filename='styles.css') }}">
</head>
<body>
<p>Add Health Record</p>
<p> Record added successfully </p>
<p><a href="{{url_for("task4_1")}}">Back to Menu</a></p>
</body>
</html>
```

```
## Student.py
##class Student – Helper class used to contain student particular for Task4_2

class Student:
      def __init__(self, name, gender, weight, height, id=0 ):
             self._studentID = id
             self._name=name
             self._gender =gender
             self._weight=weight
             self._height=height

      def getStudentID(self):
             return self._studentID

      def setStudentID(self, id):
             self._studentID=id

      def getName(self):
             return self._name

      def setName(self, name):
             self._name=name

      def getGender(self):
             return self._gender

      def setGender(self, gender):
```

```
            self._gender=gender

      def getWeight(self):
            return self._weight

      def setWeight(self, weight):
            self._weight=f'{weight:.2f}'

      def getHeight(self):
            return self._height

      def setHeight(self, height):
            self._height=f'{height:.2f}'
```

```
## HealthRecord.py
## class Record - Helper class used to contain health statistic for Task4_3

class Record:
      def __init__(self, attribute):
            self._attribute = attribute
            self._male=0
            self._female=0

      def getAttribute(self):
            return self._attribute

      def setAttribute(self, attribute):
            self._attribute=attribute

      def getMale(self):
            return self._male

      def setMale(self, male):
            if self._attribute=="Number":
                  self._male= f'{male}'
            else:
                  self._male= f'{male:.2f}'

      def getFemale(self):
            return self._female

      def setFemale(self, female):
            if self._attribute=="Number":
                  self._female=f'{female}'
            else:
                  self._female=f'{female:.2f}'
```

```
## Task4.py

import flask, os, sqlite3
from Student import Student
from HealthRecord import Record
from flask import render_template, request, redirect, url_for

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


### Task4_1
@app.route('/', methods=['GET'])
def task4_1():
    return render_template('task4_1.html')

### Task4_2
@app.route('/all')
```

```
def task4_2():
	sql="select  student.name, student.gender,
StudentHealthRecord.weight,StudentHealthRecord.height from student left outer join
StudentHealthRecord on student.studentID = StudentHealthRecord.studentid  order by
name"
	db = sqlite3.connect('students.db')
	db.row_factory = sqlite3.Row
	cursor = db.execute(sql)
	all_rows = cursor.fetchall()
	cursor.close()
	db.close()
	listx=[]
	for row in all_rows:
		s=Student(row["name"], row["gender"], row["weight"],row["height"])
		listx.append(s)
	return render_template('task4_2.html', results=listx)

### Task4_3
@app.route('/statistics', methods=['GET'])
def task4_3():
	db = sqlite3.connect('students.db')
	db.row_factory = sqlite3.Row
	sql="select  gender as gender, count(*) as cnt, avg(weight) as wt,
avg(height) as ht from student left outer join StudentHealthRecord on
student.StudentID=StudentHealthRecord.StudentID group by gender"
	cursor = db.execute(sql)
	all_rows = cursor.fetchall()
	cursor.close()
	db.close()
	listx=[]
	numberRec = Record("Number")
	weightRec = Record("Avg Weight")
	heightRec = Record("Avg Height")

	for row in all_rows:
		if row["gender"]=="M":
			numberRec.setMale(row["cnt"])
			weightRec.setMale(row["wt"])
			heightRec.setMale(row["ht"])
		else:
			numberRec.setFemale(row["cnt"])
			weightRec.setFemale(row["wt"])
			heightRec.setFemale(row["ht"])
	listx.append(numberRec)
	listx.append(weightRec)
	listx.append(heightRec)
	return render_template('task4_3.html', results=listx)

### Task4_4
@app.route('/add', methods=['GET', 'POST'])
def task4_4():
	if request.method=='GET':
		return render_template('task4_4.html')
	if 'action' in request.form:
		action = request.form['action']
		name = request.form['name']
		gender = request.form['gender']
		weight = request.form['weight']
		height = request.form['height']
	if action == 'Add':
		try:
			db = sqlite3.connect('students.db')
			cur = db.cursor()
```

```
                    cur.execute("Insert into Student(Name, Gender)
values(?,?)",(name,gender))
                    studentID = cur.lastrowid
                    cur = db.execute("INSERT INTO StudentHealthRecord (StudentID,
Weight, Height) VALUES (?, ?,?)",(studentID,weight,height))
                    db.commit()
                    cur.close()
                    db.close()
                    return render_template('Task4_4k.html', msg="Added
successfully")
             except:
                    if db:
                           db.close()
                           return render_template('Task4_4k.html',  msg="Add Error")
                    else:
                           return redirect(url_for('task4_1'))
      else:
             result_msg=''
             return redirect(url_for('task4_1'))



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

Task4_1

Menu

List All Student Health Records

Health Record Statistic

Add Health Record

Task4_2

Student Health Records

Name	Gender	Weight	Height
Alex	M	51.0	1.75
Arlo	M	55.0	1.65
Ella	F	46.0	1.7
Isla	F	48.0	1.68
June	F	50.0	1.75
Kai	M	None	None
Leo	M	60.0	1.73
Nyla	F	None	None
Vera	F	50.0	1.8
Zane	M	None	None

Back to Menu

Task4_3

Health Record Statistics

Attributes	Male	Female
Number	5	5
Avg Weight	55.33	48.50
Avg Height	1.71	1.73

Back to Menu

Task4_4

Add Health Record

Name:

Gender: Male Female

Weight:

Height:

Add

Back to Menu