

Task 1 Soln

```
# Task 1.1

import random

rows = 6
cols = 6

# for loop to display
def displayMaze(maze):
    for i in range(rows):
        for j in range(cols):
            print(maze[i][j], end = ' ')
        print()
    print()

# open and close file
# create 2D array
def getMaze(maze):
    inFile = open('MAZE.txt', 'r')
    for line in inFile:
        line = list(line.rstrip())
        maze.append(line)
    inFile.close()

# generate exit position
# update the maze
def createExit(maze):
    col = random.randint(1, cols-2)    # from 2nd col to 2nd last col
    maze[rows-1][col] = "."
    exitPos = {'y':rows-1, 'x':col}
    return exitPos

def main():
    maze = []
    getMaze(maze)
    exitPos = createExit(maze)
    displayMaze(maze)

main()
```

```
# # # # T #
# . # . . #
# . . . # #
# # . # . #
# . . . . #
# . # # # #
```

```

# Task 1.2

def getDirection():
    newD = input("Enter direction ('U','D','L','R',''): ").upper()
    valid = ['U','D','L','R','']
    while newD not in valid:
        print('Invalid direction! Try again!')
        newD = input("Enter user direction ('U','D','L','R',''): ").upper()
    return newD

def moveRobot(maze, robot, newD, steps):
    newY, newX = robot['y'], robot['x']
    if newD == 'L':
        newX -= 1
    elif newD == 'R':
        newX += 1
    elif newD == 'U' and newY > 0:
        newY -= 1
    elif newD == 'D':
        newY += 1
    # if newD == '', do nothing

    if maze[newY][newX] == '.':
        steps += 1
        maze[robot['y']][robot['x']] = 'X'
        maze[newY][newX] = 'T'
        robot['y'], robot['x'] = newY, newX
    else:
        print ("Can't go there!\n")

    return steps

def main():
    maze = []
    getMaze(maze)
    exitPos = createExit(maze)
    displayMaze(maze)

    # initialization
    robot = {'y':0, 'x':4}
    atExit = False
    steps = 0
    prevD = ""

    while not atExit:
        newD = getDirection()
        if newD == "":
            newD = prevD

        steps = moveRobot(maze, robot, newD, steps)

```

```

        if robot == exitPos:
            atExit = True
        else:
            prevD = newD

        displayMaze(maze)

    print (f"The robot takes {steps} moves to exit the maze.")

main()

```

```

# # # # T #
# . # . . #
# . . . # #
# # . # . #
# . . . . #
# # # # . #

Enter direction ('U','D','L','R',''): D
# # # # X #
# . # . T #
# . . . # #
# # . # . #
# . . . . #
# # # # . #

Enter direction ('U','D','L','R',''): R
Can't go there!

# # # # X #
# . # . T #
# . . . # #
# # . # . #
# . . . . #
# # # # . #

Enter direction ('U','D','L','R',''): L
# # # # X #
# . # T X #
# . . . # #
# # . # . #
# . . . . #
# # # # . #

Enter direction ('U','D','L','R',''): D
# # # # X #
# . # X X #
# . . T # #
# # . # . #
# . . . . #
# # # # . #

```

Enter direction ('U','D','L','R',''): L

```
# # # # X #
# . # X X #
# . T X # #
# # . # . #
# . . . . #
# # # # . #
```

Enter direction ('U','D','L','R',''): D

```
# # # # X #
# . # X X #
# . X X # #
# # T # . #
# . . . . #
# # # # . #
```

Enter direction ('U','D','L','R',''):

```
# # # # X #
# . # X X #
# . X X # #
# # X # . #
# . T . . #
# # # # . #
```

Enter direction ('U','D','L','R',''): R

```
# # # # X #
# . # X X #
# . X X # #
# # X # . #
# . X T . #
# # # # . #
```

Enter direction ('U','D','L','R',''):

```
# # # # X #
# . # X X #
# . X X # #
# # X # . #
# . X X T #
# # # # . #
```

Enter direction ('U','D','L','R',''): D

```
# # # # X #
# . # X X #
# . X X # #
# # X # . #
# . X X X #
# # # # T #
```

The robot takes 9 moves to exit the maze.

***Marker's Comments:**

Task 1.1

- Some students did not close the file after reading it or comment that file automatically closes if use Python's 'with' keyword for opening.
- Most of the students, who did not perform well, are not familiar with the syntax of creating a 2-D array (or list of lists).
- When displaying the maze, some students attempt incorrectly by outputting the list directly instead of writing code to output each value individually as shown in the question.
- The most common error in generating the random number for exit was to include the random numbers 0 and 5. Random number generated must be from 1 to 4. If not, the maze is unsolvable.

Task 1.2

- A significant number of students did not validate the input of direction as required.
- The most common error was not to check if the new position is out of bounds (e.g. robot cannot move 'U' when it is in its initial position at $y = 0$ and $x = 4$).
- Some students did not recognise when an empty string was input, and the previous move had to be repeated. Those that did handle this situation often used unnecessarily repeated code, with many solutions deploying significant amounts of copying and pasting. Students should use functions to make their code shorter and more readable.

Task 2 Soln:

```
#Task2_1

class Person:
    def __init__(self, name, age):
        self._name=name
        self._age=age

    def getName(self):
        return self._name

    def getAge(self):
        return self._age

    def print(self):
        print (f"{self._name}, {self._age}")
```

```
#Task2_2

def task2_2(filename):
    lst=[]

    f = open(filename, 'r')
    for line in f:
        name, age= line.split(',')
        lst.append(Person(name.strip(), int(age.strip())))
    f.close()

    return lst

#print(task2_2('person.txt'))

list_of_person = task2_2('PERSON.txt')
for person in list_of_person:
    person.print()
```

```
Alice, 18
Bob, 20
Charlie, 17
David, 16
Emily, 19
Austin, 19
Cole, 20
Adam, 16
Benjamin, 16
Chloe, 19
Daniel, 19
Eva, 20
Bailey, 18
Daisy, 18
Amelia, 17
Brian, 19
Catherine, 18
Dylan, 17
Eleanor, 16
```

Bella, 17
Caleb, 16
Delilah, 20
Ethan, 17
Ella, 18
Arthur, 20

Task2_3 insertion sort

```
def task2_3(list_of_person, key, order):  
    n=len(list_of_person)  
  
    for i in range(1, n):  
  
        target = list_of_person[i]  
        j = i-1  
  
        if key == "name":  
            if order == 'asc':  
                while j >= 0 and  
                    (target.getName() < list_of_person[j].getName()):  
                    list_of_person[j+1] = list_of_person[j]  
                    j -= 1  
            else:  
                while j >= 0 and  
                    (target.getName() > list_of_person[j].getName()):  
                    list_of_person[j+1] = list_of_person[j]  
                    j -= 1  
  
        elif key == "age":  
            if order == 'asc':  
                while j >= 0 and  
                    (target.getAge() < list_of_person[j].getAge()):  
                    list_of_person[j+1] = list_of_person[j]  
                    j -= 1  
            else:  
                while j >= 0 and  
                    (target.getAge() > list_of_person[j].getAge()):  
                    list_of_person[j+1] = list_of_person[j]  
                    j -= 1  
  
        list_of_person[j+1] = target  
  
list_of_person=task2_2('PERSON.txt')  
task2_3(list_of_person, 'name', 'asc')  
for person in list_of_person:  
    person.print()
```

Adam, 16
Alice, 18
Amelia, 17
Arthur, 20
Austin, 19
Bailey, 18

```
Bella, 17
Benjamin, 16
Bob, 20
Brian, 19
Caleb, 16
Catherine, 18
Charlie, 17
Chloe, 19
Cole, 20
Daisy, 18
Daniel, 19
David, 16
Delilah, 20
Dylan, 17
Eleanor, 16
Ella, 18
Emily, 19
Ethan, 17
Eva, 20
```

```
# Task2_4
#Quicksort

def quicksort(list_of_person, low, high, key, order):
    # perform a recursive quicksort
    # sorting the range [low,high], inclusive of both ends

    if low < high:    # list has more than one element
        # partition into two sublists, pos is partitioning index
        pos = partition(list_of_person, low, high, key, order)

        # separately sort elements before partition and after partition
        quicksort(list_of_person, low, pos-1, key, order)
        quicksort(list_of_person, pos+1, high, key, order)

    # else list has 0 or 1 element and requires no sorting

def partition(list_of_person, low, high, key, order):
    # partition list into two sublists
    # re-arrange the list so that the pivot is properly partitioned

    i = low    # boundary index
    pivot = list_of_person[high]

    for j in range(low, high):

        # if current element is smaller than the pivot, move it to the left
        if key == 'name':
            if order == 'asc':
                if list_of_person[j].getName() < pivot.getName():
                    list_of_person[i], list_of_person[j]
                    = list_of_person[j], list_of_person[i]
                #swap cur element with the element at boundary index
                i = i + 1    # increment boundary index
```



```

        else:
            if list_of_person[j].getName() > pivot.getName():
                list_of_person[i], list_of_person[j]
                = list_of_person[j], list_of_person[i]
                #swap cur element with the element at boundary index
                i = i + 1          # increment boundary index

    elif key == 'age':
        if order == 'asc':
            if list_of_person[j].getAge() < pivot.getAge():
                list_of_person[i], list_of_person[j]
                = list_of_person[j], list_of_person[i]
                #swap cur element with the element at boundary index
                i = i + 1          # increment boundary
        else:
            if list_of_person[j].getAge() > pivot.getAge():
                list_of_person[i], list_of_person[j]
                = list_of_person[j], list_of_person[i]
                # swap cur element with the element at boundary index
                i = i + 1          # increment boundary index

    # end of FOR loop; place pivot in the correct position at index i
    list_of_person[i], list_of_person[high]
    = list_of_person[high], list_of_person[i]
    return i          # final position of pivot

def task2_4(list_of_person, key, order):
    quicksort(list_of_person, 0, len(list_of_person)-1, key, order)

list_of_person=task2_2('PERSON.txt')
task2_4(list_of_person, 'age', 'desc')
for person in list_of_person:
    person.print()

```

```

Arthur, 20
Delilah, 20
Bob, 20
Eva, 20
Cole, 20
Emily, 19
Chloe, 19
Daniel, 19
Austin, 19
Brian, 19
Alice, 18
Ella, 18
Bailey, 18
Daisy, 18
Catherine, 18
Charlie, 17
Amelia, 17
Dylan, 17

```

```
Bella, 17
Ethan, 17
Benjamin, 16
Caleb, 16
David, 16
Eleanor, 16
Adam, 16
```

```
#Task 2.5
```

```
def task2_5(list_of_person, method, key, order):
    if method=='insertion sort':
        task2_3(list_of_person, key, order)
    elif method=='quick sort':
        task2_4(list_of_person, key, order)

list_of_person=task2_2('PERSON.txt')
task2_5(list_of_person, 'quick sort', 'name', 'desc')
for person in list_of_person:
    person.print()
```

```
Eva, 20
Ethan, 17
Emily, 19
Ella, 18
Eleanor, 16
Dylan, 17
Delilah, 20
David, 16
Daniel, 19
Daisy, 18
Cole, 20
Chloe, 19
Charlie, 17
Catherine, 18
Caleb, 16
Brian, 19
Bob, 20
Benjamin, 16
Bella, 17
Bailey, 18
Austin, 19
Arthur, 20
Amelia, 17
Alice, 18
Adam, 16
```

***Marker's Comments**

Task2.2

- Some students did not close the file after reading it or comment that file automatically closes if use Python's 'with' keyword for opening.
- Some students did not execute the task2_2 as instructed in the question and did not include the expected output in their submitted file.

Task2.3

- Some students uses swapping of adjacent elements instead of shifting the element to the adjacent index and insert the target to the correct position.
- Some students failed to execute task2_3 with the argument values as instructed in the question, and did not include the expected output in their submitted files.
- Student must verify the accuracy of all combinations of keys and orders for task2_3, rather than focusing solely on the specific one mentioned in the question.
- Using reverse() function or list reversal via slicing is not in line with the question's expectations, as it necessitates that the student demonstrate a sound understanding of the insertion sort logic.

Many students' code lacks sufficient comments to clarify certain statements or contains unnecessary comments. Incorporating a consistent practice of offering insightful and purposeful comments can greatly assist readers in understanding the coder's thought process, eliminating the need for guesswork when deciphering the code.

\$

Task 3 Soln:

```
# Task 3.1

class Player:
    def __init__(self, name, elo, pointer):
        self.name = name
        self.elo = elo
        self.ptr = pointer

class PlayerList:
    def __init__(self, n):
        self.head = -1
        self.free = 0
        self.data = [None] * n
        for i in range(n-1):
            self.data[i] = Player('-', -1, i+1)
        self.data[n-1] = Player('-', -1, -1)

    def size(self):
        counter = 0
```

```

    ptr = self.head
    while ptr != -1:
        counter += 1
        ptr = self.data[ptr].ptr
    return counter

def register(self, name, elo):
    if self.free == -1:
        print(f"Teams are full, unable to register {name}.")
        return

    elo = int(elo)
    curr = self.head
    prev = -1

    while curr != -1 and self.data[curr].elo > elo:
        prev = curr
        curr = self.data[curr].ptr

    new = self.free
    self.free = self.data[self.free].ptr
    self.data[new].name = name
    self.data[new].elo = elo

    if prev == -1:    # adding to front
        self.head = new
    else:
        self.data[prev].ptr = new

    self.data[new].ptr = curr    # adding general case

def withdraw(self, name):
    ptr = self.head
    prev = -1

    while ptr != -1 and self.data[ptr].name != name:
        prev = ptr
        ptr = self.data[ptr].ptr

    if ptr == -1:
        print(f"{name} not found.")
    else:
        self.data[ptr].name = "-"
        self.data[ptr].elo = -1
        if prev == -1:
            self.head = self.data[ptr].ptr    # removing head
        else:
            self.data[prev].ptr = self.data[ptr].ptr    # for general case
        self.data[ptr].ptr = self.free
        self.free = ptr    # manage free space

def display(self):

```

```

        print(f"Head: {self.head}, Free: {self.free}")
        print(f"idx {'player name':^13} {'elo':^5} {'ptr':^3}")
        cap = len(self.data)
        for i in range(cap):
            print(f"{i:>2}: {self.data[i].name:^13} {self.data[i].elo:>5} {self.data[i].ptr:>3}")

```

Task 3.2

```

import csv

cteam = PlayerList(7)

f = open('CHESS.csv','r')
data = csv.reader(f)
for person in data:
    name, elo = person
    cteam.register(name,int(elo))
f.close()

print()
cteam.display()
print()
cteam.withdraw('Taylor')
print('Size:',cteam.size())
print()
cteam.display()

```

Teams are full, unable to register Kim.
Teams are full, unable to register Adele.

```

Head: 4, Free: -1
idx  player name    elo  ptr
0:    Nicki         1250  3
1:    Lisa          1337  0
2:    Iggy           828   5
3:    Taylor        1109  6
4:    Missy         1437  1
5:    Megan          745  -1
6:    Cardi          962   2

```

Size: 6

```

Head: 4, Free: 3
idx  player name    elo  ptr
0:    Nicki         1250  6
1:    Lisa          1337  0
2:    Iggy           828   5

```

3:	-	-1	-1
4:	Missy	1437	1
5:	Megan	745	-1
6:	Cardi	962	2

***Marker's Comments**

Task3.1

- In `PlayerList` class, the `__init__()` method should have 3 attributes as given in the question, using the variable names given. Many students created `self.CAPACITY` and `self.size`
- `size()` method is trying to test student understanding on simple traversal and keeping a counter. Students need to think about what the examiner is trying to assess in the question. Using `self.size` attribute is not allowed as the question mentioned there are the 3 given attributes only.
- `register()` method: When the `self.data` array is full, returning a string is not as useful as the error message does not show. `PRINT` is recommended for error messages.
- `display()` method: contents of the `data` array is a `Player` object that has 3 attributes, but some students did not print `ptr`

Task3.2

- While many students cannot get the desired output, it is important to take note that `name` is a string and `elo` is an integer. Hence, the data type must be corrected converted when registering.
- Many students submitted outputs with poor readability. Do consider labeling your output and using whitespace. eg. printing 6 (for `cteam.size()`) without the `size:` label and cluttered in the middle is discouraged.

Task 4 soln:

```
# Task 4.1

import sqlite3
conn = sqlite3.connect('STORE.db')

# for debugging
conn.execute('DROP TABLE IF EXISTS Donut')
conn.execute('DROP TABLE IF EXISTS Member')
conn.execute('DROP TABLE IF EXISTS Sale')

conn.execute("CREATE TABLE Donut ( \
    DonutID INTEGER UNIQUE PRIMARY KEY, \
    DonutName TEXT, \
    UnitPrice REAL)")
conn.execute("CREATE TABLE Member ( \
    MemberNumber INTEGER UNIQUE PRIMARY KEY, \
    MemberName TEXT, \
    Phone TEXT)")
conn.execute("CREATE TABLE Sale ( \
    SaleID INTEGER UNIQUE PRIMARY KEY, \
    MemberNumber INTEGER, \
    DonutID INTEGER, \
    Date TEXT, \
    Quantity INTEGER, \
    FOREIGN KEY(MemberNumber) REFERENCES Member(MemberNumber), \
    FOREIGN KEY(DonutID) REFERENCES Donut(DonutID))")

conn.commit()
conn.close()

# Task 4.2

import sqlite3
conn = sqlite3.connect('STORE.db')

f = open('DONUT.txt', 'r')
for line in f:
    DonID, DonName, Price = line.strip().split(',')
    conn.execute("INSERT INTO Donut(DonutID, DonutName, UnitPrice) \
        VALUES (?, ?, ?)", (int(DonID), DonName, float(Price)))
f.close()

f = open('MEMBER.txt', 'r')
for line in f:
    MemNum, MemName, Phone = line.strip().split(',')
    conn.execute("INSERT INTO Member(MemberNumber, MemberName, Phone) \
        VALUES (?, ?, ?)", (int(MemNum), MemName, Phone))
f.close()

f = open('SALE.txt', 'r')
```

```

for line in f:
    SaleID, MemNum, DonID, Date, Quantity = line.strip().split(',')
    conn.execute("INSERT INTO Sale(SaleID, MemberNumber, DonutID, \
        Date, Quantity) VALUES (?, ?, ?, ?, ?)", \
        (int(SaleID), int(MemNum), int(DonID), Date, int(Quantity)))
f.close()
conn.commit()
conn.close()

```

Task 4.3

```

import sqlite3
conn = sqlite3.connect('STORE.db')

number = input("Please enter member's number:")
query = "SELECT MemberName FROM Member WHERE MemberNumber = ?"
cursor = conn.execute(query, (number, ))
print('Orders by', cursor.fetchone()[0])

query = "SELECT Donut.DonutName, Sale.Date, Sale.Quantity \
        FROM Sale INNER JOIN Donut ON Donut.DonutID = Sale.DonutID \
        INNER JOIN Member ON Member.MemberNumber = Sale.MemberNumber \
        WHERE Member.MemberNumber = ? "
cursor = conn.execute(query, (number,))

print('Donut Name \t Date \t Quantity')
for result in cursor:
    print(f"{result[0]} \t {result[1]} \t {result[2]}")
conn.close()

```

Please enter member's number:104
Orders by Calvin

Donut Name	Date	Quantity
Ping Strawberry	20230720	3
Ping Classic	20230721	2
Plain Cruller	20230721	1
Ping Strawberry	20230723	3
Sugar Cruller	20230726	3
Plain Cruller	20230726	1

Task 4.4

```

import sqlite3, flask
from flask import render_template, request
app = flask.Flask(__name__)

@app.route('/', methods = ['GET', 'POST'])
def index():
    if request.method == 'GET':
        return render_template('form.html')
    else:
        date = request.form['date']
        conn = sqlite3.connect('STORE.db')
        cursor = conn.execute("SELECT Donut.DonutName, SUM(Sale.Quantity) \
                                FROM Sale INNER JOIN Donut \
                                ON Sale.DonutID = Donut.DonutID \
                                WHERE Sale.Date = ? \
                                GROUP BY Donut.DonutID \

```



```

ORDER BY SUM(Sale.Quantity) DESC", (date, ))

results = []
for result in cursor:
    results.append(result)
conn.close()
return render_template('display.html', results = results)

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

```

```

<!DOCTYPE html>
<html>
  <head><title>Summary of Order by Date</title></head>
  <body>
    <table border = 1px>
      <tr>
        <th>Donut Name</th>
        <th>Quantity</th>
      </tr>

      {% for row in results %}
        <tr>
          <td>{{row[0]}}</td>
          <td>{{row[1]}}</td>
        </tr>
      {% endfor %}
    </table>
  </body>
</html>

```

```

<!DOCTYPE html>
<html>
  <head><title>Order Form</title></head>
  <body>
    <form method = 'post'>
      <h1>Please enter the date in numerical form of yyyyymmdd:</h1>
      <h3><i>For example, enter 20230701 for 2023 July 1</i></h3>
      <p><input name = 'date'></p>
      <p><input type = 'submit'></p>
    </form>
  </body>
</html>

```

HTML Output

Donut Name	Quantity
Ping Classic	7
Black Chocolate	4
Plain Cruller	3

Marker's Comments

SQL statements can be very long, use line breaks '\n' to make your codes more readable.

We can use the **quick reference guide and DB Browser** to write and test SQL statements.

Task 4.1 we need to adjust the **data type** in real life context (price is REAL, not TEXT or INTEGER), or as required by the question (Date is TEXT). FK of one table comes from PK of another table. MemberNumber in Sale is not unique since one member can make multiple orders, DonutID in sale is not unique since one donut can be ordered many times.

Task 4.2 **Text files must be closed after reading**. Students using 'with' methods must write comments that the file is closed automatically.

Task 4.3 input from the user is the member number, and we use SQL to find the member name.

Task 4.4 folder and files management is done poorly. Folder 'Task 4_4' has subfolder 'templates' containing the two HTML template files, and Python file 'Task 4_4.py' to run the Flask, and database file 'STORE.db'. Markers should be able to run your web application (Python file) in the folder without any changes, and hence may not mark any files in the wrong location. Many students are not aware of the use of SUM, GROUP BY and ORDER BY.