HACETTEPE UNIVERSITY DEPARTMENT OF COMPUTER ENGINEERING

BBM103 – ASSİGNMENT 2 REPORT

Gazi Kağan Soysal

2210356050

23/11/2022

CONTENTS

1. PROBLEM

Health is the most important issue in life for us. If we lose our health, life becomes very difficult for us. For this, we have to be very careful about our health. In other words, we shouldn't be sick. If we are sick, we must do whatever it takes to get well. But in doing so, we may be doing something unnecessarily. The most important reason for this is misdiagnosis. If we consider cancer, the most important disease of our time, tests are being done about it. However, the accuracy rate of these tests is not 100%. Even though we don't have cancer if we live as if it is cancer, we will be using drugs unnecessarily. We are also psychologically worn out because we think we are sick.

2. DESİGN

We will set up a data system. In this system, our patients will be informed, and their opportunities and suggestions will be explained to us. To enter data, we will use the input.txt file located in the same folder as our code. At the beginning of the code, we will write some code to read this file line by line. Thus, each row will be a separate data entry for us. We will also need to edit some details so that the written texts can be perceived correctly. We will define a list called "patient_list" to keep the information of all patients together.

When we write "create" to this text file, our program will understand that patient information will be created. After typing "Create", the patient's name, diagnosis accuracy rate, name of the disease, incidence rate, treatment name, and risk ratio of the treatment will be written next to the same line. These should be written with a space between them. When this information is written, the patient will be registered to the system thanks to the "create()" function. If the patient is registered in the system, it sends a "patient already registered" warning message.

When we want to remove a patient from the system, after typing "remove" in a new line of the text file, the name of the patient to be removed will be written. The rest of the information is not needed because when we write a name, the system will establish the connection between that patient's name and the information. Thanks to the "remove()" function, the patient in the system will be removed from the system. If there is no patient with this name in the system, a "no patient to be removed" warning is sent.

When creating and removing patients, when we want to see existing patients, we can see a list of existing patients by typing "list" into the file.

When we want to see the probability that a patient in the system has the disease, we can see this probability after typing "probability" and typing the patient's name. If the patient is not in the system, a "probability not calculated" warning will be sent.

When you want to get advice about whether the current patient should be treated or not, after typing "recommendation" and typing the patient's name, a message "the system recommends or does not recommend treatment" is displayed by the system thanks to the "recommendation()" function. If the patient is not in the system, the message "Unable to advise because the patient is not in the system" is sent.

All these operations can be done in batches and after all of them are written, the result of each operation will be printed in the "outputs.txt" file in the same folder, thanks to the "saving_output_file" function.

3. Programmer's Catalogue

3.1)

```
patient_list = []

all_inputs = open("doctors_aid_inputs.txt", "r")

all_outputs = open("doctors_aid_outputs.txt", "w")

outputs_file = ""
```

At the beginning of our code, we define our list (patient_list) where all patients' information is kept. Then, to read the "input.txt" file in our folder, we open it with the "open" command and assign it to our "all_inputs" variable. To print the information to be generated, we open our file with the open command and assign it to the "all_outputs" variable. The "outputs_file" variable is assigned to the text to be printed.

3.2)

```
for i in range(len(a)):

    read_file()

    if operation == "create":
        create()

    elif operation == "remove":
        remove()

    elif operation == "list":
        list()

    elif operation == "probability":
        probability()

    elif operation == "recommendation":
        recommendation()
```

This is the bottom part of our code. The reason it is at the bottom is to see the functions. Here it loops as much as the line we wrote in the "inputs.txt" file and reads the lines in order. It calls the required function according to the command at the beginning of the line it reads.

3.3)

```
def read_file():
    global inputs, operation
    inputs = all_inputs.readline().rstrip("\n").split(", ")

    if inputs == ["list"]:
        operation = "list"

    else:
        SpaceCharacter = inputs[0].index(" ")
        operation = inputs[0][:SpaceCharacter]
        inputs[0] = inputs[0][SpaceCharacter + 1:]
```

Here we define our "read_file()" function. We use ".rstrip" and ".split" to read the entered texts properly and assign them to the "inputs" variable. The "inputs" variable contains the information in each line and this variable will help us while writing the codes. If our "inputs" variable is anything but "list", we need to separate the first word from the second word. That's why we separate these two words in the "else" part.

3.4)

```
def create():
    if inputs are not in patient_list:

        patient_list.append(inputs)
        outputs_file = "Patient {} is recorded.\n".format(inputs[0])
        saving_output_file(outputs_file)
        return patient_list

    else:
        outputs_file = "Patient {} cannot be recorded due to duplication.\n".format(inputs[0])
        saving_output_file(outputs_file)
        return patient_list
```

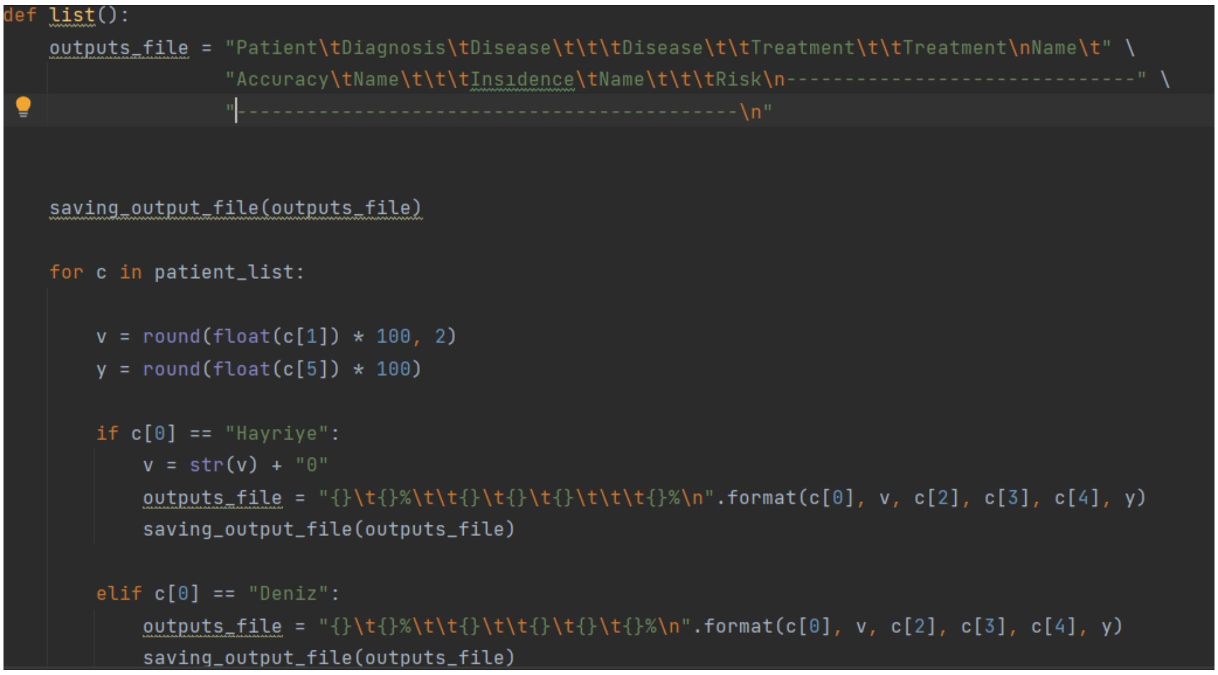
Here we define our "create()" function. If the "inputs" variable is not in the "patient_list" list, we add it to our list. Otherwise, we do not add it and write the necessary text and send it to the necessary function to write it in the "outputs.txt" file.

3.5)

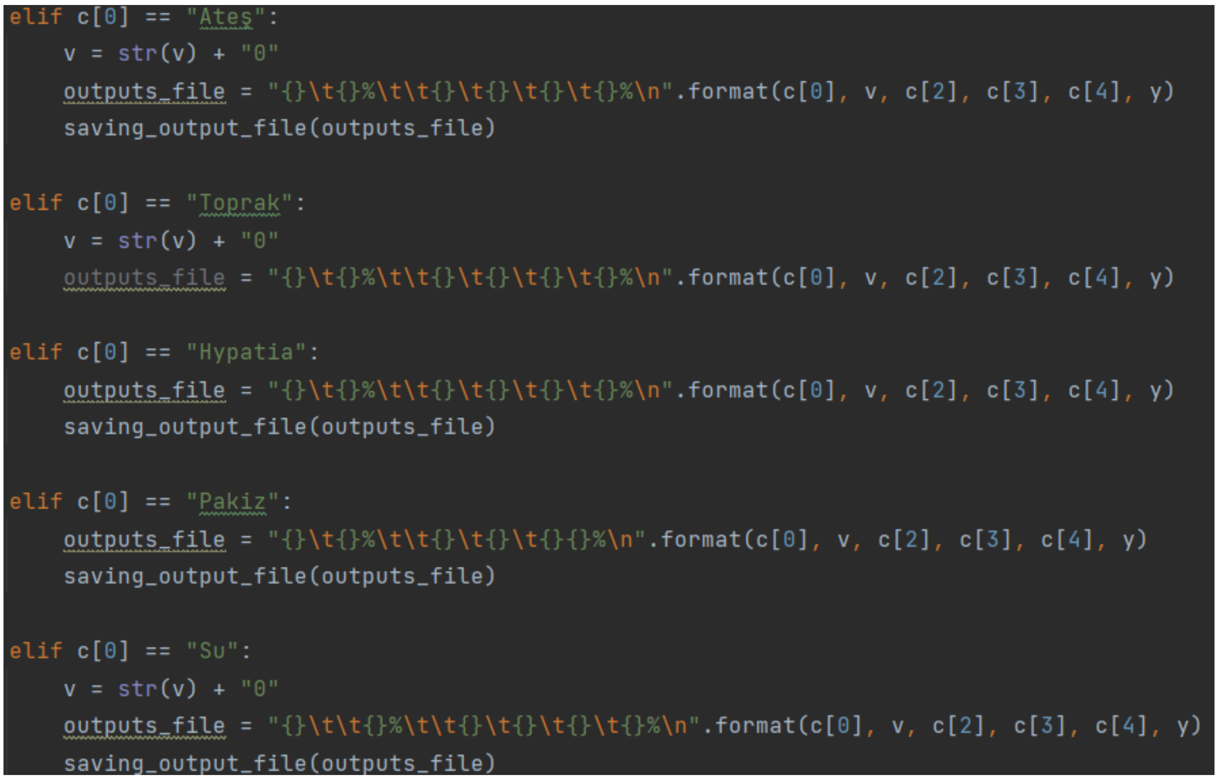
```
def remove():
    just_one_item = []

    for m in patient_list:
        just_one_item.append(m[0])

    if inputs[0] in just_one_item:

        a = just_one_item.index(inputs[0])
        patient_list.pop(a)
        outputs_file = "Patient {} is removed.\n".format(inputs[0])
        saving_output_file(outputs_file)
        return patient_list

    elif inputs in patient_list:

        patient_list.remove(inputs)
        outputs_file = "Patient {} is removed.\n".format(inputs[0])
        saving_output_file(outputs_file)
        return patient_list


    else:
        outputs_file = "Patient {} cannot be removed due to absence.\n".format(inputs[0])
        saving_output_file(outputs_file)
        return patient_list
```

Here we define our "remove()" function. First, we create our "just_one_item" list. This list has been created to use only the lines that write commands and names. First, we put our information in the "patient_list" into the "for" loop and process it one by one, and assign it to the "m" variable. Then we look for the name written in the "patient_list" for lines that contain only names and then lines that are otherwise. If there is, we remove it from the list, otherwise, we say "not on the list". And again, we direct it to the output function to write the text.

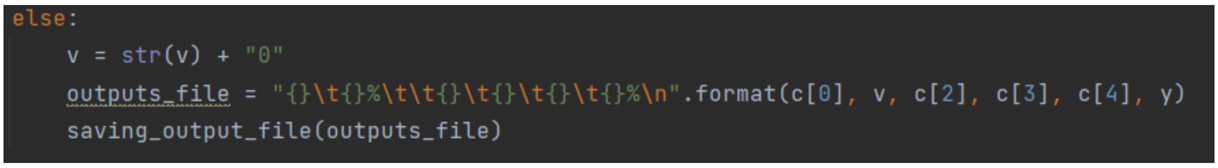
3.6)



```python
def list():
    outputs_file = "Patient\tDiagnosis\tDisease\t\t\tDisease\t\tTreatment\t\tTreatment\nName\t" \
                   "Accuracy\tName\t\t\tInsıdence\tName\t\t\tRisk\n-----------------------------" \
                   "-----------------------------------------\n"


    saving_output_file(outputs_file)

    for c in patient_list:

        v = round(float(c[1]) * 100, 2)
        y = round(float(c[5]) * 100)

        if c[0] == "Hayriye":
            v = str(v) + "0"
            outputs_file = "{}\t{}%\t\t{}\t{}\t{}\t\t\t{}%\n".format(c[0], v, c[2], c[3], c[4], y)
            saving_output_file(outputs_file)

        elif c[0] == "Deniz":
            outputs_file = "{}\t{}%\t\t{}\t\t{}\t{}\t{}%\n".format(c[0], v, c[2], c[3], c[4], y)
            saving_output_file(outputs_file)
```



```python
elif c[0] == "Ates":
    v = str(v) + "0"
    outputs_file = "{}\t{}%\t\t{}\t{}\t{}\t{}%\n".format(c[0], v, c[2], c[3], c[4], y)
    saving_output_file(outputs_file)

elif c[0] == "Toprak":
    v = str(v) + "0"
    outputs_file = "{}\t{}%\t\t{}\t{}\t{}\t{}%\n".format(c[0], v, c[2], c[3], c[4], y)

elif c[0] == "Hypatia":
    outputs_file = "{}\t{}%\t\t{}\t{}\t{}\t{}%\n".format(c[0], v, c[2], c[3], c[4], y)
    saving_output_file(outputs_file)

elif c[0] == "Pakiz":
    outputs_file = "{}\t{}%\t\t{}\t{}\t{}{}%\n".format(c[0], v, c[2], c[3], c[4], y)
    saving_output_file(outputs_file)

elif c[0] == "Su":
    v = str(v) + "0"
    outputs_file = "{}\t\t\t{}%\t\t{}\t{}\t{}\t{}%\n".format(c[0], v, c[2], c[3], c[4], y)
    saving_output_file(outputs_file)
```



```python
else:
    v = str(v) + "0"
    outputs_file = "{}\t{}%\t\t{}\t{}\t{}\t{}%\n".format(c[0], v, c[2], c[3], c[4], y)
    saving_output_file(outputs_file)
```

Here we define our "list()" function. First, we write our titles and add our lines. Then we write our information in order, using spaces specific to each name.

3.7)

```
def probability():
    k = []

    for m in patient_list:

        if m[0] == inputs[0]:

            a = int(m[3][0:2]) * float(m[1])
            b = int(m[3][3:9])
            c = round(100 - float(m[1]) * 100, 2)
            d = float(m[1]) * 100

            real_rate = round((a / (round(((b - a) * c / 100)) + a)) * 100, 2)

            small = str(m[2]).lower()

            outputs_file = "Patient {} has a probability of {}% of having {}.\n".format(m[0], real_rate, small)
            saving_output_file(outputs_file)

        else:
            continue

        k.extend(m)

    if inputs[0] not in k:
        outputs_file = "Probability for {} cannot be calculated due to absence.\n".format(inputs[0])
        saving_output_file(outputs_file)
```

Here we define our "probability()" function. For each patient, we find the

probability result by obtaining the number values in the correct data type and making the necessary calculations.

3.8)

```
def saving_output_file(x):
    all_outputs.write(x)
```

Here we create our "saving_output_file" function. We called this function when we were going to write output in all functions so far. And in this function, we print the command from the function to the file.

Time spent analyzing, designing, implementing, testing, and reporting

I tried to understand what we should do by reading the file sent on the evening of the day of the assignment. And I thought about what I could benefit from by examining the lecture notes. The next day I started writing my code. I finished my code by working 2-3 hours a day for about 1 week. After taking a break for a few days, I started writing my report. It looks like it will take me about 4 hours in total.

Reusability

I think other programmers can easily use the code I wrote. Because it is not a complex structure. It looks pretty clear and orderly.

4.USER CATALOGUE

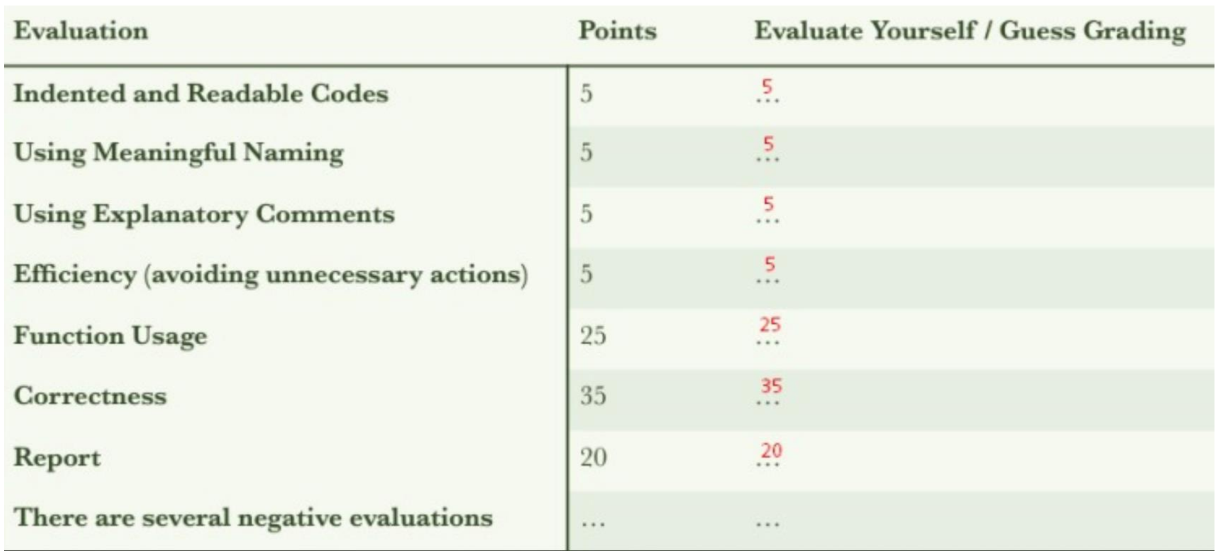
4.1)Program's user manual/tutorial

To use the code, you must first create a text file named "doctors_aid_inputs" in the file containing the code. Then, after writing the name of the operation you have done, you should write the patient's information by leaving 1 space between each word. You need to do this when creating the patient. For other procedures, you only need to write the patient's name. When you're done typing, you'll see a file called "doctors_aid_outputs" in the same folder. When you enter this file, you will see your output.

4.2)Restrictions on the program

Let's look at the limitations of the program. If there are 2 patients with the same name but different information, our program will not know which patient to remove in commands other than "create" and "list". Therefore, it will be necessary to enter the other information of the patient one by one.

GRADING TABLE



Evaluation	Points	Evaluate Yourself / Guess Grading
Indented and Readable Codes	5	5 …
Using Meaningful Naming	5	5 …
Using Explanatory Comments	5	5 …
Efficiency (avoiding unnecessary actions)	5	5 …
Function Usage	25	25 …
Correctness	35	35 …
Report	20	20 …
There are several negative evaluations	…	…