BBM103

INTRODUCTION TO THE PROGRAMMING

Assignment 2 : Doctor's Aid

2210356079 Cansu ASLAN

23.11.2022

CONTENTS SECTİON

- **ANALYSIS________________________________3-3**

- **DESIGN__________________________________3-9**

  CREATE() FUNCTION

  CALCULATE_PROB(I) FUNCTION

  PROBABILITY() FUNCTION

  RECOMMENDATION() FUNCTION

  REMOVE() FUNCTION

  INFO_LIST() FUNCTION

  WRITE_OUTPUT(I) FUNCTION

  REMAINING CODES

- **PROGRAMMER CATALOUGE____________________10-13**

  SPENT TIME

  ATTACHMENT OF THE CODE

- **USER CATALOUGE__________________________14_14**

  INSTRUCTIONS

ANALYSIS

In the health sector, the main problem can be called organizing data, keeping it, and calculating mistakes. Organizing data is a difficult job to do because every person has their data and there are plenty of them. It is close to impossible to do this job manually so we need the help of the computers in this area. Keeping this data organized way is another struggle. Using concrete folders is a great expense item and also it occupies too much place. This way cannot be sustainable and efficient. A simple program can handle both problems. Finally making a diagnosis is a serious business because it is about life and death. Calculating all those little numbers by hand can occur a lot of mistakes it can be irrevocable at some point. It may be useful to consult another observation that is non-subjective. I have tried to remedy problems by taking them into consideration.

DESIGN

The assignment is requested us to carry out the given commands with the help of the information from a file full of patient information. The file should include the asked command, patients' names, diagnosis accuracy of the patient, diagnosed disease name, disease incidence, requested treatment for the patient, and finally the risk of the treatment. The program will create another file that includes the commands from the doctor's input file to be more organized in the hospitals.

The program can add a new patient to the system, the creation steps include passing the given information relevant to the patient in the input file.

It is able to remove the patients if the doctor is done with the patient. After this operation, the patient will be removed her/his all information. It means you cannot process any of the patient's information if you try it system will give you a message related with due to the absence of the patient.

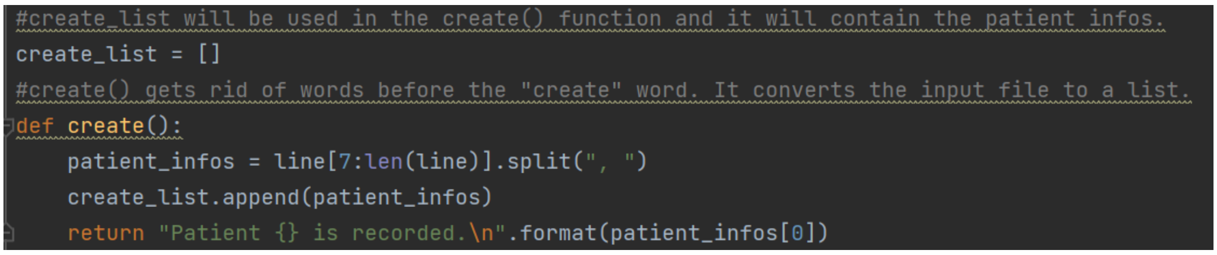
The program can calculate the probability of having the diagnosed disease. No matter how advanced the health system is and how low the margin of error is, it is still risky enough to lead to very big mistakes. Because the larger the examined crowd, the more noticeable the reflection of the margin of error becomes. It is unacceptable to misdiagnose a person with cancer and there will be some consequences. To prevent mistakes like this the confusion matrix will be used during the calculation steps.

According to the results of the probabilty function, the program will recommend whether to have the treatment or not to have to treatment.

The program will make a table including all information about the recorded patients. This function will be helpful to see the all patients together and checking the input file.

CREATE FUNCTION():

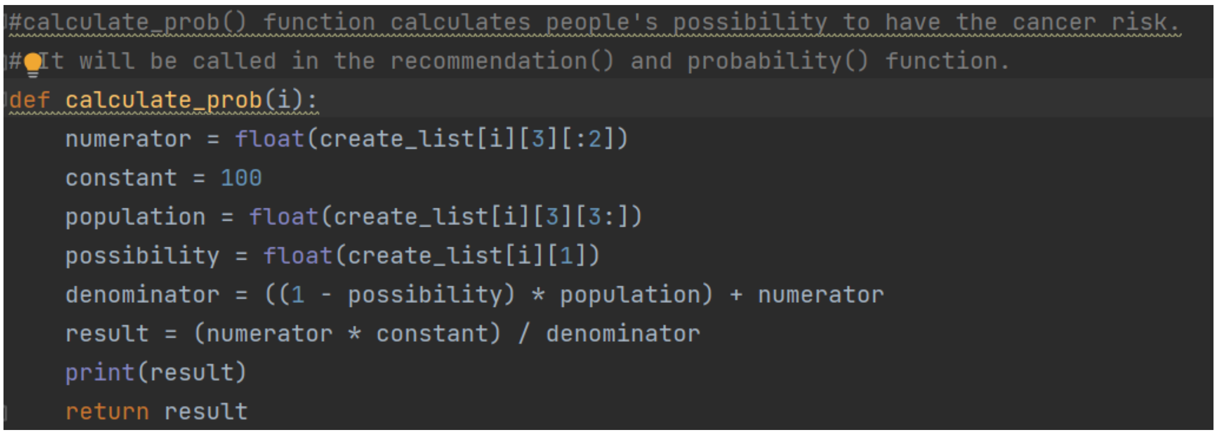
Firstly, I built an empty list for adding new elements after the implementation. After this, I defined a function which is called create(). To start with, create function takes from the seventh character to the end of every line which is read and the split method is useful to create a list. The reason that why I specified the range after the seventh character is because I wanted to eliminate the create command which is written in the first six characters. I used "," in the split method because I needed to split every piece of information of the patients in a list. Afterward append method is adding the chosen parts to the list. Finally, we need a return for the function. I declared it "Patient {} is recorded." The curly bracket is for the name variable. I used the format method to write different names for the output file. Format method takes the name variables from the patient_infos. It takes the first character of this list. "\n" is to make the new line move to the bottom line. This function will be called when there is "create" in the read lines.



```python
#create_list will be used in the create() function and it will contain the patient infos.
create_list = []
#create() gets rid of words before the "create" word. It converts the input file to a list.
def create():
    patient_infos = line[7:len(line)].split(", ")
    create_list.append(patient_infos)
    return "Patient {} is recorded.\n".format(patient_infos[0])
```

CALCULATE_PROB(i) FUNCTION:

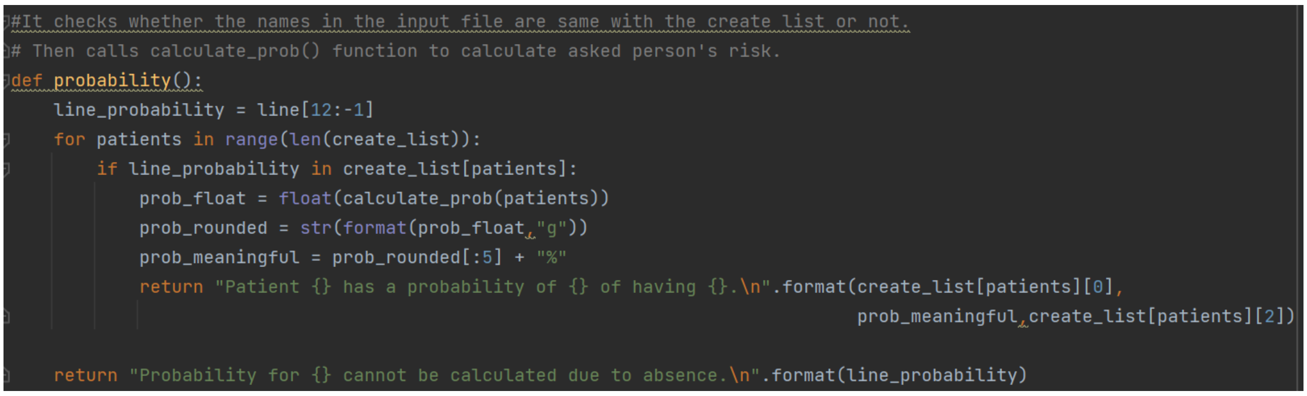
I defined a calculate_prob(i) function to get rid of repeating the calculations shown below. The meaning of the "i" in the paranthesis is for showing that the function needs a parameter to work. In this function, we use confusion matrix logic to understand the probability of whether to have the disease or not. Even though we have advanced technology nowadays, there is always a margin of error. We need to make some calculations so as not to ignore it. Float function converts the strings from the create_list, which is we generated previously to the floating numbers. The purpose of the indexing the elements is for getting the numbers which is given in the input text. Finally function's return is the calculation of the transaction. This function will be called while the implementation of the recommendation() and probability() functions.



```python
#calculate_prob() function calculates people's possibility to have the cancer risk.
#It will be called in the recommendation() and probability() function.
def calculate_prob(i):
    numerator = float(create_list[i][3][:2])
    constant = 100
    population = float(create_list[i][3][3:])
    possibility = float(create_list[i][1])
    denominator = ((1 - possibility) * population) + numerator
    result = (numerator * constant) / denominator
    print(result)
    return result
```

PROBABILITY() FUNCTION:

The purpose of this function is to print the probability of a patient having the disease. Firstly function takes the elements of the lines starting from the twelfth character to the penultimate. After that, I made a loop for the process. The loop examines the part of the create_list in the range of its length. The patient variable, you realize there is a variable. It is changing according to the range of the list. In the loop, there is an if statement which is controlling whether taken line interval and create_list's indexes are equal or not. If it is equal then it goes inside the if block. On that occasion, the function calls the calculate_prob() function to determine the process. The float function converts the result to a floating number. After that function uses the format function and "g" operator to comply with the desired. The purpose of it is to take only meaningful numbers to take into account. Then we convert the result to the string to add the "%" symbol. After all the procedures it returns the result with a message. The message is "Patient {} has a probability of {} having {}.\n" I used curly brackets and the format method to have a dynamic return message. The first curly bracket is for the name, the second is for the probability of having the disease, and the last one is the cancer name. All of them are personal so we couldn't just assume that they are the same. "\n" is for printing the other line below this line. If the conditions at the first "if" block couldn't be met then the loop jumps to the return of the for loop which is this message "Probability for {} cannot be calculated due to absence.\n". Again I used the format function to have a dynamic output. Curly brackets stand for the names in the line_probabililty. If the outcomes of the line_probability and create_list are not equal then the function sends you that warning.

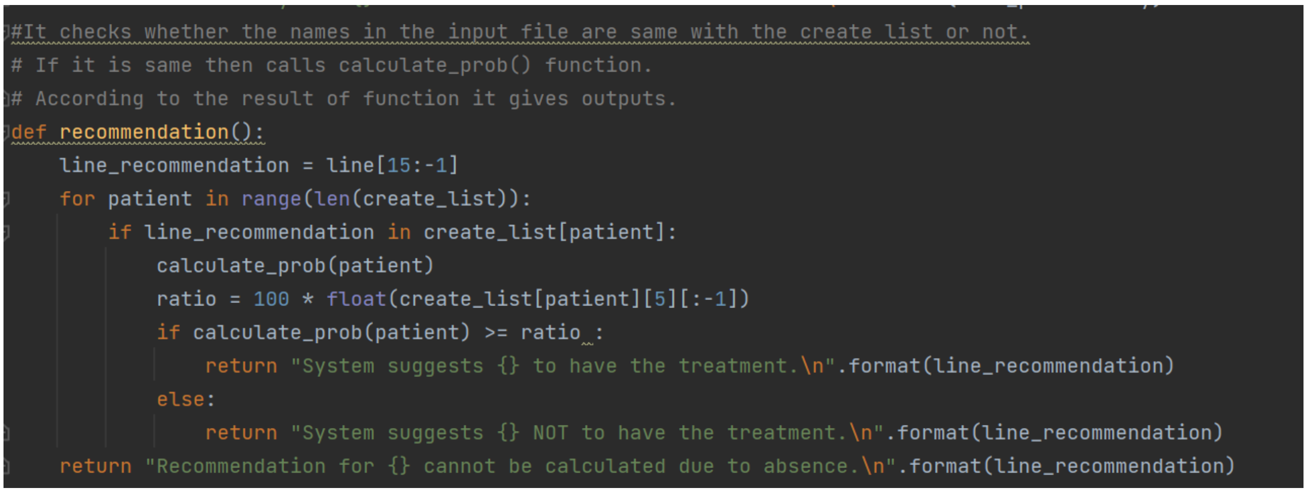


```python
#It checks whether the names in the input file are same with the create list or not.
# Then calls calculate_prob() function to calculate asked person's risk.
def probability():
    line_probability = line[12:-1]
    for patients in range(len(create_list)):
        if line_probability in create_list[patients]:
            prob_float = float(calculate_prob(patients))
            prob_rounded = str(format(prob_float,"g"))
            prob_meaningful = prob_rounded[:5] + "%"
            return "Patient {} has a probability of {} of having {}.\n".format(create_list[patients][0],
                                                                            prob_meaningful,create_list[patients][2])

    return "Probability for {} cannot be calculated due to absence.\n".format(line_probability)
```

RECOMMENDATION() FUNCTION:

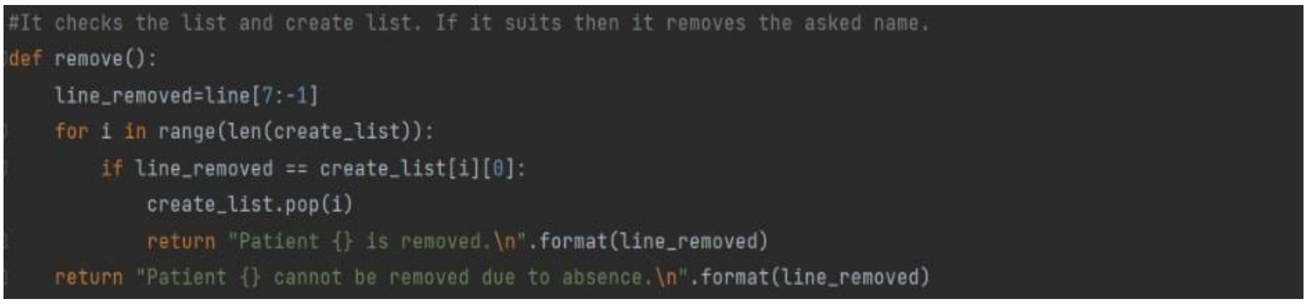
This function has used the recommendations to the patient on whether to have the treatment or not. Firstly function takes the characters of the read lines starting from fifteen to the penultimate. Afterward, the loop that I created starts running. It takes the length of the list and determines the interval of the process. Inside the loop "if" statement checks for accuracy. If they are equal then the function lets you get into the "if" block. Once you entered the "if " block function calls the calculation_prob() function to make a comparison. As you can see here, there is a variable named ratio. It takes the specific risks of every patient. It compares the calculated value and the specific risk. If the calculated value is bigger or equal to the specific person's risk then it returns the "System suggests {} to have the treatment.\n" otherwise it returns "NOT" to have the treatment. I used the format method to have a dynamic return message in both returns. "\n" stands for printing the following messages to the bottom line. In case line_recommendation and create_list[patient] are not equal then it gives a direct return message by skipping the whole part which is "Recommendation for {} cannot be calculated due to absence.\n". It means that no such patient is found on the patient list. Format method prints the name cannot be found.



```
#It checks whether the names in the input file are same with the create list or not.
# If it is same then calls calculate_prob() function.
# According to the result of function it gives outputs.
def recommendation():
    line_recommendation = line[15:-1]
    for patient in range(len(create_list)):
        if line_recommendation in create_list[patient]:
            calculate_prob(patient)
            ratio = 100 * float(create_list[patient][5][:-1])
            if calculate_prob(patient) >= ratio :
                return "System suggests {} to have the treatment.\n".format(line_recommendation)
            else:
                return "System suggests {} NOT to have the treatment.\n".format(line_recommendation)
    return "Recommendation for {} cannot be calculated due to absence.\n".format(line_recommendation)
```

REMOVE() FUNCTION:

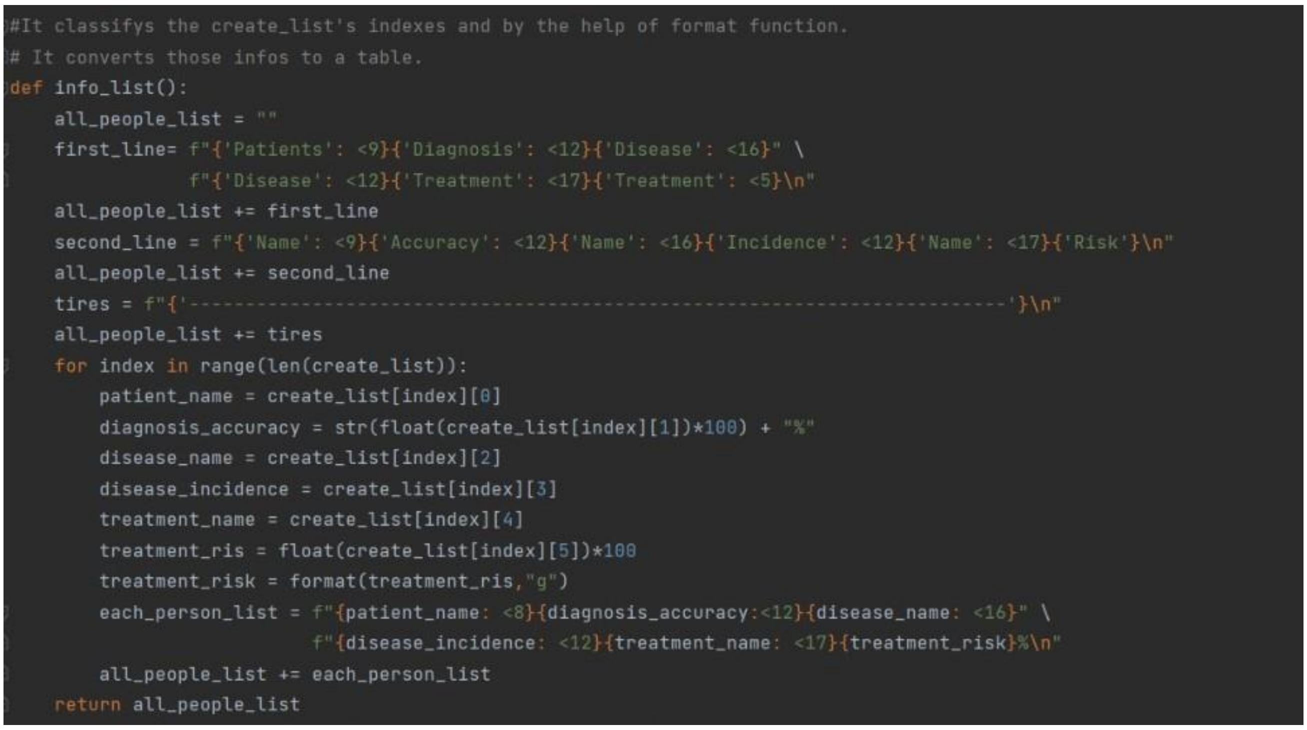
The remove function is helpful to delete an existing patient from the records of the hospital or the system. Firstly it takes the characters of the input lines starting from the seventh to the penultimate index and then it starts a loop with the "for " condition. It determines the length of the loop by looking at the length of the create_list. Then "if" condition checks the lengths of elements. If it is true function lets you get inside the "if" block and after that point, the "pop" command deletes the equal element from the list. Finally, it returns the "Patient {} is removed.\n" message. I got help from the format function, too. Whereas conditions are not met then the code goes to the last line of the function and this line returns a message "Patient {} cannot be removed due to absence.\n". It happens only if the patient doesn't exist in the list so the function cannot process the patient's information. Curly brackets for the format method. It prints the name which is user asked to process.



```python
#It checks the list and create list. If it suits then it removes the asked name.
def remove():
    line_removed=line[7:-1]
    for i in range(len(create_list)):
        if line_removed == create_list[i][0]:
            create_list.pop(i)
            return "Patient {} is removed.\n".format(line_removed)
    return "Patient {} cannot be removed due to absence.\n".format(line_removed)
```

INFO_LIST() FUNCTION:

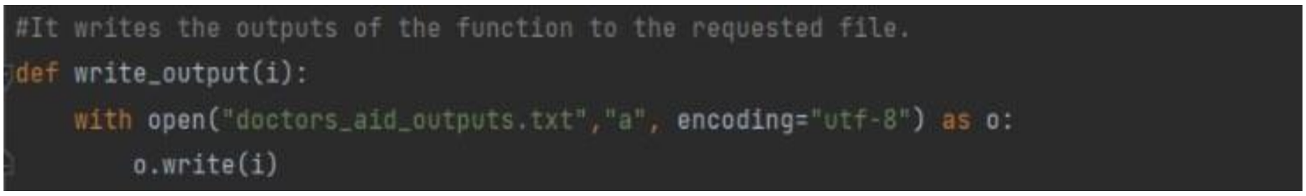
Info_list function is useful to give an output of the information of the patients in a table format. First of all, I created an empty string called “all_people_list”. The table will be written in this string. To create a table I preferred to use the f-string command. It helped me to determine the edges of the columns. For the first and the second lines, I designed the column names to have a better-looking table, and also third line’s purpose is the same. I added the lines to the empty string and to have the following bars “\n” was my solution. After adding the little details to the empty string I created a “for” loop to convert the information to a table. It takes the specific indexes and assigns them to the variables. For a few of them, I had to make some adjustments to have the desired version, like adding the “%” symbol, getting rid of meaningless digits, or rounding numbers. Every index will be written in the empty string in the desired format. Again, I used an f-string to have a proper output I decided on the ranges. Lastly, function returns the empty list but with a difference now it includes our table.



```python
#It classifys the create_list's indexes and by the help of format function.
# It converts those infos to a table.
def info_list():
    all_people_list = ""
    first_line= f"{'Patients': <9}{'Diagnosis': <12}{'Disease': <16}" \
                f"{'Disease': <12}{'Treatment': <17}{'Treatment': <5}\n"
    all_people_list += first_line
    second_line = f"{'Name': <9}{'Accuracy': <12}{'Name': <16}{'Incidence': <12}{'Name': <17}{'Risk'}\n"
    all_people_list += second_line
    tires = f"{'-------------------------------------------------------------------------'}\n"
    all_people_list += tires
    for index in range(len(create_list)):
        patient_name = create_list[index][0]
        diagnosis_accuracy = str(float(create_list[index][1])*100) + "%"
        disease_name = create_list[index][2]
        disease_incidence = create_list[index][3]
        treatment_name = create_list[index][4]
        treatment_ris = float(create_list[index][5])*100
        treatment_risk = format(treatment_ris,"g")
        each_person_list = f"{patient_name: <8}{diagnosis_accuracy:<12}{disease_name: <16}" \
                           f"{disease_incidence: <12}{treatment_name: <17}{treatment_risk}%\n"
        all_people_list += each_person_list
    return all_people_list
```

WRITE_OUTPUT(i) FUNCTION:

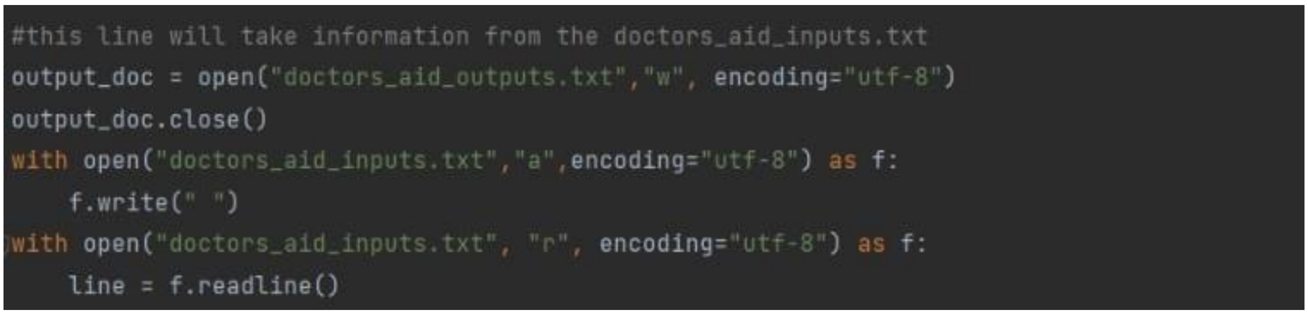
It opens the desired file. It will be understood by looking at the first quotes and the “a” stands for append the asked elements to the list. “i” in the parenthesis is showing that the function has to take a parameter to start running. “encoding= “utf-8” is helpful not to have struggled with the Turkish letters, it prevents printing the meaningless symbols to the output file. “o” is representing the file name and o.write(i) means when it is called function will add the output to the specific file.



```
#It writes the outputs of the function to the requested file.
def write_output(i):
    with open("doctors_aid_outputs.txt","a", encoding="utf-8") as o:
        o.write(i)
```

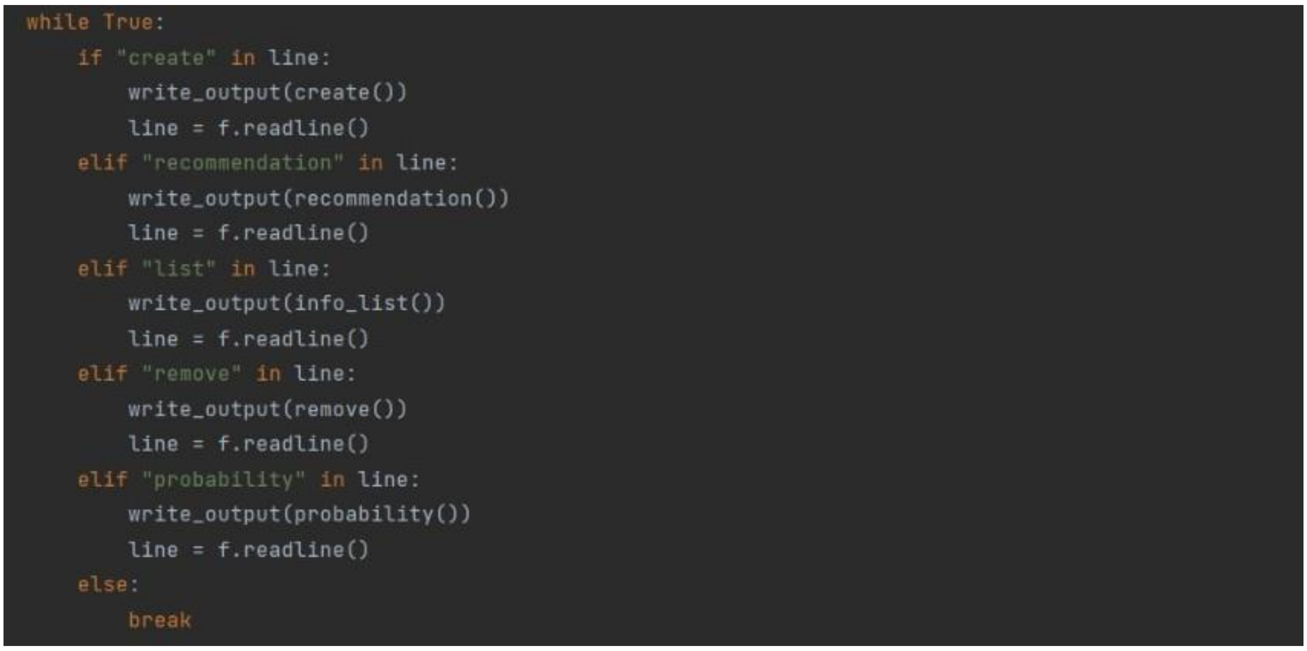
REMAINING CODES:

I created the output_doc variable to prevent taking the same output in the file over and over. In the first quotes I wrote the file that I want to process. “w” means “write” and encoding= “utf-8” prevents the function from printing meaningless symbols. After that I closed the file, it helps to see the output only one time whenever the user runs the program. I added a “ ” to the end of the list for taking to the process the last line because all the lines before the last one had a space after and it occurred some problems. I used “a” which appends the asked character. I tried the solve the problem that way. Finally, I made the program read the first line with the open() command. In the first quotes, I wrote the name of the input file. In the second quote, there is an “r” letter which means read and in the last one, encoding provides a layout for Turkish characters. I defined the read lines as “lines” to use
later.



```
#this line will take information from the doctors_aid_inputs.txt
output_doc = open("doctors_aid_outputs.txt","w", encoding="utf-8")
output_doc.close()
with open("doctors_aid_inputs.txt","a",encoding="utf-8") as f:
    f.write(" ")
with open("doctors_aid_inputs.txt", "r", encoding="utf-8") as f:
    line = f.readline()
```

The “while” condition provides to call the functions that I defined before. While the conditions are true then, the functions start running. The “if” condition confirms the accuracy. In the first one, it is searching for the create function. If that word exists on the line being examined, it calls the write_output(i) function and I wrote the function inside of the write_output(i) function instead of the “i” parameter according to the matching word. The write_output(i) function returns the result of the function inside of it to the file that I customized to see the results. Finally, it reads another line from the input text which is taken from the user. The “elif ” conditions that you can see are checking the lines for the other functions. After each of them, a new line will be read. This provides the program not to stop. The “Else” condition is for the mistaken inputs. If the commands don’t match with the functions then it runs the “else” block and doesn’t process the word. It doesn’t give a result just ignores the command. It prevents some problems that can occur.



```python
while True:
    if "create" in line:
        write_output(create())
        line = f.readline()
    elif "recommendation" in line:
        write_output(recommendation())
        line = f.readline()
    elif "list" in line:
        write_output(info_list())
        line = f.readline()
    elif "remove" in line:
        write_output(remove())
        line = f.readline()
    elif "probability" in line:
        write_output(probability())
        line = f.readline()
    else:
        break
```

PROGRAMMING CATALOUGE:

TIME SPENT ON THIS ASSIGNMENT

I spent two hours to analyzing the problem and understanding what it is asking for. Designing took three hours. Implementing the code took five hours but debugging and testing took me three days because of the little mistakes like missing lines in the input file etc. I spent four hours to writing the report. I didn't use "try, except" function to prevent the possible problems so I can say that if user follows the insturactions in the "User Catalouge" then this program can be useful for a simple data system.

```
#create_list will be used in the create() function and it will contain the patient infos.
create_list = []
#create() gets rid of words before the "create" word. It converts the input file to a list.
def create():
    patient_infos = line[7:len(line)].split(", ")
    create_list.append(patient_infos)
    return "Patient {} is recorded.\n".format(patient_infos[0])
#calculate_prob() function calculates people's possibility to have the cancer risk.
# It will be called in the recommendation() and probability() function.
def calculate_prob(i):
    numerator = float(create_list[i][3][:2])
    constant = 100
    population = float(create_list[i][3][3:])
    possibility = float(create_list[i][1])
    denominator = ((1 - possibility) * population) + numerator
    result = (numerator * constant) / denominator
    print(result)
    return result
#It checks whether the names in the input file are same with the create list or not.
# Then calls calculate_prob() function to calculate asked person's risk.
```

```
def probability():
    line_probability = line[12:-1]
    for patients in range(len(create_list)):
        if line_probability in create_list[patients]:
            prob_float = float(calculate_prob(patients))
            prob_rounded = str(format(prob_float,"g"))
            prob_meaningful = prob_rounded[:5] + "%"
            return "Patient {} has a probability of {} of having
{}.\n".format(create_list[patients][0],

prob_meaningful,create_list[patients][2])
    return "Probability for {} cannot be calculated due to
absence.\n".format(line_probability)
#It checks whether the names in the input file are same with the create
list or not.
# If it is same then calls calculate_prob() function.
# According to the result of function it gives outputs.
def recommendation():
    line_recommendation = line[15:-1]
    for patient in range(len(create_list)):
        if line_recommendation in create_list[patient]:
            calculate_prob(patient)
            ratio = 100 * float(create_list[patient][5][:-1])
            if calculate_prob(patient) >= ratio :
                return "System suggests {} to have the
treatment.\n".format(line_recommendation)
            else:
                return "System suggests {} NOT to have the
treatment.\n".format(line_recommendation)
    return "Recommendation for {} cannot be calculated due to
absence.\n".format(line_recommendation)
#It checks the list and create list. If it suits then it removes the asked
name.
def remove():
    line_removed=line[7:-1]
    for i in range(len(create_list)):
        if line_removed == create_list[i][0]:
            create_list.pop(i)
            return "Patient {} is removed.\n".format(line_removed)
    return "Patient {} cannot be removed due to
absence.\n".format(line_removed)
#It classifys the create_list's indexes and by the help of format function.
# It converts those infos to a table.
```

```
def info_list():
    all_people_list = ""
    first_line= f"{'Patients': <9}{'Diagnosis': <12}{'Disease': <16}" \
                f"{'Disease': <12}{'Treatment': <17}{'Treatment': <5}\n"
    all_people_list += first_line
    second_line = f"{'Name': <9}{'Accuracy': <12}{'Name':
<16}{'Incidence': <12}{'Name': <17}{'Risk'}\n"
    all_people_list += second_line
    tires = f"{'-------------------------------------------------------------------
--'}\n"
    all_people_list += tires
    for index in range(len(create_list)):
        patient_name = create_list[index][0]
        diagnosis_accuracy = str(float(create_list[index][1])*100) + "%"
        disease_name = create_list[index][2]
        disease_incidence = create_list[index][3]
        treatment_name = create_list[index][4]
        treatment_ris = float(create_list[index][5])*100
        treatment_risk = format(treatment_ris,"g")
        each_person_list = f"{patient_name:
<8}{diagnosis_accuracy:<12}{disease_name: <16}" \
                           f"{disease_incidence: <12}{treatment_name:
<17}{treatment_risk}%\n"
        all_people_list += each_person_list
    return all_people_list
#It writes the outputs of the function to the requested file.
def write_output(i):
    with open("doctors_aid_outputs.txt","a", encoding="utf-8") as o:
        o.write(i)
#this line will take information from the doctors_aid_inputs.txt
output_doc = open("doctors_aid_outputs.txt","w", encoding="utf-8")
output_doc.close()
with open("doctors_aid_inputs.txt","a",encoding="utf-8") as f:
    f.write(" ")
with open("doctors_aid_inputs.txt", "r", encoding="utf-8") as f:
    line = f.readline()
    while True:
        if "create" in line:
            write_output(create())
            line = f.readline()
        elif "recommendation" in line:
            write_output(recommendation())
            line = f.readline()
```

```
elif "list" in line:
    write_output(info_list())
    line = f.readline()
elif "remove" in line:
    write_output(remove())
    line = f.readline()
elif "probability" in line:
    write_output(probability())
    line = f.readline()
else:
    break
```

USER CATALOUGE:

This is a simple function to keep and process data. It can be useful for the hospitals or clinics. There are some simple requirements you have to keep track. First of all you have to check the functions from the design part and decide which ones you want to use. After that you have to create a text file which includes the patient's information and command. It should be like in the shown order in the example input file.

```
create Hayriye, 0.999, Breast Cancer, 50/100000, Surgery, 0.40
create Deniz, 0.9999, Lung Cancer, 40/100000, Radiotherapy, 0.50
create Ateş, 0.99, Thyroid Cancer, 16/100000, Chemotherapy, 0.02
probability Hayriye
recommendation Ateş
create Toprak, 0.98, Prostate Cancer, 21/100000, Hormonotherapy, 0.20
create Hypatia, 0.9975, Stomach Cancer, 15/100000, Immunotherapy, 0.04
recommendation Hypatia
create Pakiz, 0.9997, Colon Cancer, 14/100000, Targeted Therapy, 0.30
list
remove Ateş
probability Ateş
recommendation Su
create Su, 0.98, Breast Cancer, 50/100000, Chemotherapy, 0.20
recommendation Su
list
probability Deniz
probability Pakiz
```

Finally you have to be sure about patient information that you gave to the system because the system will be processing what is given. It cannot process the information make sense or true or in a different order. This is all you have to do for running this program, after that you can see the outputs of what you asked for.