HACETTEPE UNIVERSITY

DEPARTMENT OF COMPUTER ENGINEERING

BBM 103: INTRODUCTION TO PROGRAMMING LABORATORY I

ASSIGNMENT 2: DOCTOR'S AID

STUDENT NUMBER: 2220356114

NAME: UYGAR

SURNAME: ERSOY

DELIVERY DATE : 24.11.22

TABLE OF CONTENTS

ANALYSIS

In this problem, a system called Doctor's Aid -that makes probability-based decisions about people with cancer related illnesses- will be designed for clinicians. Information about the patient, such as patient's name, patient's disease name, diagnosis accuracy, disease incidence, the treatment name for the disease and the risk of the treatment will be provided through an input file called "doctors_aid_inputs.txt". Input file will be read line by line and every command in the described line will be executed with the rest of the information in that line. Each command will have a unique purpose and their output will be written in an output file called "doctors_aid_outputs.txt".

Create command will record the patient to the patients list. It will record patient's name, diagnosis accuracy, disease name, disease incidence, treatment name and treatment risk. Depending on the condition it will provide two outcomes.

- If person "x" was already recorded, then it will output "Person "x" cannot be recorded due to duplication"
- If person "x" was not recorded, then it will output "Person "x" is recorded"

Remove command will delete a patient from patients list. It will have two possible outcome depending on the given patient.

- If patient "x" in the records, then it will output "Patient "x" is removed"
- If patient "x" is not in the records, it will output "Patient "x" cannot be removed due to absence"

List command will list the information of every currently recorded patient in the patients list.

Probability command will calculate the actual fatality probability of a given patient with the information of the patient in the patients list. It may result in two possible outcome.

- If patient is recorded, then it will output "Patient "x" has a probability of "y"% of having "z" ".
- If patient is not recorded, then it will output "Probability for "x" cannot be calculated due to absence."

Recommendation command will compare the result of probability command's calculation with the treatment risk and provide a system suggestion. It could output three possible scenarios.

- If probability command results with higher fatality probability than the treatment risk and the patient is recorded, then it will outcome "System suggests "x" to have the treatment."
- If probability command results with lower fatality probability than the treatment risk and patient is recorded, then it will outcome "System suggests "x" NOT to have the treatment."
- If patient is not recorded, then it will output "Recommendation for "x" cannot be calculated due to absence."

Each of the command will result in one of the output of their own and those outputs will be written to the "doctors_aid_outputs.txt" file. To read and write to files, a new functions will be implemented for each.

DESIGN

Reading From Input File

To read from a file read_file() function will be implemented. This function will be provided with the name of the input file. Inside of the read_file() function, input file will be opened and each of its line will be iterated over. Each line will be checked with if-elif-else conditionals individually and depending on the command stated in each line, required functions will be called to process the information with required parameters from the line. For each line this process will be carried out.

Creating a New Patient

To create a new patient record, firstly a empty python list, named patient_list will be created at the local scope. Afterwards, create() function will be implemented with one parameter, that is, the information of the patient (name, diagnosis accuracy, disease name, disease incidence, treatment name, treatment risk). This function will receive a boolean value from a function called is_recorded(). This function takes one parameter, name of the patient, and looks if patient is already recorded in the patient_list or not. Then, it returns a boolean value. Depending on the return value, create() function will record the patient to patient_list and provide an output accordingly.

Deleting an Existing Patient

To delete an existing patient from records, a function named remove() will be used. It will have one parameter, patient name, and search over the patient_list to see if it is recorded or not. If it is in the list, it will delete the record and provide an output accordingly.

Listing All Patients with Their Information

To list the patients with all their information, a function named get_list() will be used. It will not take any parameter. It will iterate over the recordings in the patient_list and output them in a table format with headers describing what the information actually is. It will provide the table as an output.

Patient’s Probability of Having the Disease

To calculate the probability of fatality of the disease, a function named probability() with two parameters will be used. It will take patient’s name and a integer value, either 1 or 0 that specify where the function is called from. It will iterate over patient_list and calculate the probability for patient by some formula if patient is recorded. Depending on the value of integer parameter and the patient’s state of being recorded, function will output a message or return the probability value.

A Patient’s Recommendation for a Particular Treatment

To give a recommendation, a function named recommendation() with 1 parameter will be used. It will take the patient’s name as a parameter and compare the value from probability function with treatment risk of the patient. It will call probability function with integer value of 0 to make it return probability value and not output a message. Depending on the result of the comparison, it will provide an output.

Writing the Results of Each Function to a File

To write a message to the output file, a function named write_file() will be used. It will take two parameters, namely, the name of the output of the file and the message to write to the output file. Each of the mentioned functions will be created and will be ready to use in the Assignment2.py file. At the bottom of the file 3 variables named, inputs_file, outputs_file and patient_list will be created. inputs_file and outputs_file will store the name of input and output files. patient_list will be

a python list and store the patient recordings. To kick-start the process, read_file() function will be called at the end of the program with the name of the input file as a parameter. read_file() function will open the file and loop over each line of the file. Necessary cleaning and arrangements will be performed on each line inside of read_file() function and depending on the command on each line, mentioned functions will be called with required parameters. Inside of each of the function, except write_file, read_file and is_recorded(), write_file() function will be called with the name of the output file and a message to write the output of each of the function to the file. When there were no lines for read_file() to read, the program will reach to the end.

PROGRAMMER'S CATALOG

def read_file(input_file):

The whole program is functioning due to this function. It reads an input file, cleans and rearranges the lines in the file and calls other functions to process the information in the file. It takes a string parameter, input_file, which is the name of the input file. To start the process, input file is opened in read mode ,"r", to be able to read its context. In python, with statement gives users a flexibility to not manually close the file. It does that by itself.

```
#define a function to read lines from .txt file
def read_file(input_file):

    #open the file in "r" mode to be able to read it
    with open(input_file, "r") as input_file:
```

After opening the file with an alias input_file, by using a for loop as in below, each line in the file is looped over and arrangements are done on them to further use. Inside of the for loop, the new line character at the end of the each line is replaced with an empty string and to be able to harness the patient information, the values in the line is split by ", " string. This concludes with a list with string values.

```
#iterate over each line in the file
for line in input_file:
    #get rid of the new line character and split the line by "," character to process it in the future
    command = line.replace("\n", "").split(", ")
```

```
#check whether or not the desired keyword is in the first item of variable command
#replace the keyword with an empty string for further use
#call the desired function with the required or additional parameters
if "create" in command[0]:
    command[0] = command[0].replace("create ", "")
    create(command)

#second argument in probability function states where the function is called
elif "probability" in command[0]:
    command[0] = command[0].replace("probability ", "")
    probability(command[0], 1)

elif "recommendation" in command[0]:
    command[0] = command[0].replace("recommendation ", "")
    recommendation(command[0])

elif "remove" in command[0]:
    command[0] = command[0].replace("remove ", "")
    remove(command)

elif "list" in command[0]:
    get_list()
```

To call the function according to the command in each line, conditional if-elif-else blocks are used as in the above. In the list variable “command”, first values contain the commands. Conditionals checks this situation and if they meet a match, the first element in the “command” variable is modified by replacing the command name with empty string, which lefts the “command” variable with the information about patient (not the case for get_list() function). After that each function is called with required parameters to process the information from the line in input file.

def write_file(output_file, message):

This function take the name of the output file as a string and a message as a string to write to the output file. Using with statement, file is opened in append mode, “a”, which enables to keep adding new outputs to the file without overwriting it. .write() method enables to write to a file. This function will append to the output file every time it is called and add a new line at the end each time. Then, with statement will close the file.

```
#define a function to write the outputs to a .txt file
def write_file(output_file, message):

    #open the file in append mode to keep adding new lines to file
    with open(output_file, "a") as out:
        #write the message to the output file
        out.write(f"{message}\n")
```

def is_recorded(patient_name):

This function will take a name of a patient as a string value and will search it in the patient_list.

```
#define a function to check if the person is in the recordings already or not
def is_recorded(patient_name):
    #check if the records are empty or not
    if len(patient_list) != 0:
        #iterate over the records
        for record in patient_list:
            #if person is in records, return True to state he/she is already recorded
            if record[0] == patient_name:
                return True
        #return False to state that person is not in the records
        return False
    #return False to state that list is empty, therefore person is not in the records
    else:
        return False
```

Firstly, it will look at if the patient_list is nonempty or not. If it is nonempty, it will iterate over the list to look for the given patient’s name. If it finds a match, it will return True. When there is no match, it means it could not find a match and it return False. Also if the list is empty, that means there is no patient is recorded to the list. This function will be useful when creating records.

def create(patient):

This function takes a list as a parameter. This list contains the name of the patient, diagnosis accuracy, disease name, disease incidence, treatment name and treatment risk in this order. All of the values are strings. According to the value returned by is_recorded() function, if the returned

value is False, this function record the patient to the patient_list and calls write_file() function to output a message to output file, otherwise, it will call the write_file() function to output a message to output file. For further easiness in probability() function, it converts the values of diagnosis accuracy and treatment risk to float values.

```
#define a function to add patient info to patient list
def create(patient):

    #check whether or not the person is alreay recorded
    if is_recorded(patient[0]):
        #if person already exist, call write_file function with a suitable message
        message = f"Patient {patient[0]} cannot be recorded due to duplication."
        write_file(outputs_file, message)
    else:
        #if person is not recorded yet, do some altercations to person's data for further use
        #record the patient to patient_list and call write_file function to indicate recording is succesfull
        message = f"Patient {patient[0]} is recorded."
        patient[1] = float(patient[1])
        patient[-1] = float(patient[-1])
        patient_list.append(patient)
        write_file(outputs_file, message)
```

def probability(patient_name, check):

This function takes two parameters. patient_name parameter stores the name of the patient as a string value. Second parameter, check, stores an integer value, that is either 1 or 0. Inside of the function, a variable called is_found is set to False. Then, the patient_list list is searched to see if the patient is recorded in the list or not. If patient is recorded, then the value of is_found is set to True.

```
#define a function to calculate the person's probability of having the disease
def probability(patient_name, check):

    #set is_found variable to check if person is recorded or not
    is_found = False

    #iterate over records of patients
    for patient in patient_list:
        #check if the person's name is in the record
        if patient_name == patient[0]:
            #if person's name is in records set is_found variable to True
            is_found = True
```

is_recorded() function is not suitable to check the patient is recorded or not in probability() function because, is_recorded() function will return a boolean value of either True or False. It will not provide the necessary information about the patient that will be essential to calculate the probability function. If is_recorded is used here, to get the information about the patient, iteration over patient_list will be must in either way.

The calculation process of the actual fatality probability of the patient is as follows: First of all, diagnosis accuracy and disease incidence of the patient must be extracted from the patient_list. When the above for loop is executed, if statement will find a match if the patient is recorded in the system. The information about the patient will be captured in a list variable called patient in this case. patient[1] stores the diagnosis accuracy as a float, it was transformed to a float in create() function, and patient[3] stores the diagnosis accuracy as a string.

A variable called accuracy will store the diagnosis accuracy and disease_incidence will store the disease incidence. Because of the type of the disease_incidence is string, this will make a problem during the calculation process. To get the integer values out of this variable, it is split by “/” character which results a list with 2 string data, that can be transformed to an integer. This

process is shown in the code below. After this process, all the data is ready to calculate the probability of the patient.

```
#extract the values of accuracy of results and disease_incidence from person's record
accuracy, disease_incidence = patient[1], patient[3]
#split the disease_incidence by "/" character to enable calculations on it
disease_incidence = disease_incidence.split("/")
```

When a diagnosis is applied on a patient on some level of accuracy, there is two cases to consider. True positive cases and False positive cases. For example, a patient is diagnosed with a cancer with 99.9 % accuracy and the disease has a disease incidence of 50/100000. What this means is actually there are 50 * 99.9 cases, called True positive, that are actually have this disease. And 0.1 * 99950 cases that does not have this disease, called False positive, but seems to have it because of the diagnosis accuracy. In that case, patient actually has a probability of True positive / (True positive + False positive) * 100 percent chance to have the disease.

```
#calculate the true_positive value accordingly
true_positive = int(disease_incidence[0]) * accuracy * 100
#calculate the false_positive value accordingly
false_positive = (int(disease_incidence[1]) - int(disease_incidence[0])) * (1 - accuracy) * 100
#calculate the value of probability according to the formula
probability = f"{((true_positive / (true_positive + false_positive)) * 100):.2f}"
```

The code above calculates this probability. It converts the values of disease_incidence to integers and performs the mentioned calculations accordingly. disease_incidence[0] will store the numerator of the disease incidence and disease_incidence[1] will store the denominator of the disease incidence. Then for convenience, using f-strings, the probability variable will store the fatality probability of the patient with two decimal point precision as a string.

```
#check if the value of probability is an integer
if int(float(probability)) - float(probability) == 0:
    #if it is an integer, get rid of the precision of the decimal points and convert it to string for further use
    probability = str(int(float(probability)))
```

The code above will check if the probability value has all zero values after the decimal point. It is first transformed to float value, because it is a string at first. Then it is converted to integer value. When the integer form subtracted from float form of the probability value, it will result with 0 if and only if the probability value has trailing with all zeros after decimal point. If this is the case, the code above will get rid of all that zeros.

```
#check if the probability function is called from read_file function
if check == 1:
    #if it is called from read_file function, call write_file function with the appropriate message
    message = f"Patient {patient_name} has a probability of {probability}% of having {patient[2].lower()}."
    write_file(outputs_file, message)
#check if the probability function is called from recommendation function
if check == 0:
    #if it is called from recommendation function, return probability value to be used in the recommendation function
    return probability
```

Other parameter of probability() function decides what to do with this calculation. If check parameter is given the value of 1, this means that this function has been called within the read_file() function. In this situation this function will call the write_file() function with a message and result with an output message to the output file. If the value of check is given as 0, this means it is called within the recommendation() function. In this case, probability() function will return the value of probability as a string, depending on the mentioned situation, with or without a two point decimal precision.

```
#if personn is not found, call write_file function to state person was not found
if not is_found:
    write_file(outputs_file, f"Probability for {patient_name} cannot be calculated due to absence.")
```

If the value of the is_found variable did not change within the for loop at the beginning of the function, this means the patient is not recorded. In this situation, probability function will call the write_file() function with a message to output a message to the output file.

def recommendation(patient_name):

This function takes the name of the patient as the only parameter as a string. Inside of the function, variable is_found is set to False to see if the patient is recorded in the patient_list or not. The patient_list is iterated over and in each iteration, the name of the patient in each record is checked to see if the given patient's name has a match or not. If a match occurs, is_found variable is updated to True. is_recorded() function is not suitable in this case as well because, treatment risk of the patient will be required later in the function, which is_recorded() function does not supply. After the match found, probability() function is called with parameters, namely, name of the patient as a string and a check value of 0 as an integer. Returned value of this action is converted to a float and stored in a variable called prob in the code below.

```
#define a function to recommend to patient whether to take the treatment or not
def recommendation(patient_name):
    #set is_found to False to check person is recorded or not
    is_found = False

    #iterate over patients in patients_list
    for patient in patient_list:
        #check if person's name is in records
        if patient_name == patient[0]:
            #if person is in records, set is_found to True to indicate person is recorded
            is_found = True
            #get the value of probability of having the disease from probability function
            prob = float(probability(patient_name, 0))
```

After that, prob value is compared with the treatment risk value of the patient. Treatment risk value is extracted from the list patient and multiplied by 100. It is stored as the last item in that list. If the treatment risk is higher than the fatality probability of the patient, function will cal write_file() function to write a message to not recommend the treatment. Otherwise, recommendation() function will call the write_file() function to output a message to output file to recommend the treatment. The process can be observed in the code below.

```
# check if treatment risk is higher than having the disease probability
if prob < patient[-1] * 100:
    #if it is higher, call write_file function to indicate treatment is risky
    write_file(outputs_file, f"System suggests {patient[0]} NOT to have the treatment.")
else:
    #if it is not higher, call write_file function to indicate treatment can be appliable
    write_file(outputs_file, f"System suggests {patient[0]} to have the treatment.")
```

If the is_found variable is False after the loop at the beginning of the function, this means that patient is not recorded. In this case, recommendation() function will call the write_file() function to output a message to the output file.

```
#if person is not in records, call write_file function to state person was not found
if not is_found:
    write_file(outputs_file, f"Recommendation for {patient_name} cannot be calculated due to absence.")
```

def get_list():

This functions takes no parameter and just prints a table with the information about the recorded patient. It has headers that indicates the context of each of the information and a separator that separates the headers from patient information.

```
#define a function to tabulate the records in patients_list
def get_list():

    #import built-in math library
    import math

    #set the table headers and rows accordingly
    header_top = "Patient\tDiagnosis\tDisease\t\t\tDisease\t\tTreatment\t\tTreatment"
    header_bottom = "Name\tAccuracy\tName\t\t\tIncidence\tName\t\t\tRisk"
    dashes = "-" * 73
```

Inside of the function, built-in math module is imported in order to align each of the entry to table. header_top and header_bottom variables contains the header information as a string and as a separator, 73 dash character is used.

```
#put them all in a list to ease the further usage
headers = [header_top, header_bottom, dashes]

#call write_file function to output the headers of the table
for _ in range(len(headers)):
    write_file(outputs_file, headers[_])
```

Headers and separator variables is stored inside of a list and with the use of a for loop, they are outputted to the output file by being given as a parameter to write_file() function inside of a loop. Every time the get_list() function called, it will output headers and separator even with no patient recorded. This is performed in the code above.

```
#set the spaces between each column according to longest value for each column to align all records
max_word_spaces = [8, 12, 16, 12, 16, 0]

#initiate a variable to hold patients' records
patient_data = ""
```

A list named max_word_spaces that contains integers is created to specify the character number for each of the column in the table that get_list() function is going to produce. This integer values will align each entry neatly. An empty string named patient_data is created for future operations to hold the information of each individual patient. Each variable is created in the code above.

```
#iterate over records in patient_list
for patient in patient_list:
    #iterate over each patient's own records
    for index in range(len(patient)):
```

Using nested for loops, every information of each patient in the patient_list is iterated over in the code above. Patient expression holds the record for every patient and using range(len(patient)) expression enables the user to loop over each index of the patient list in the code above. For example, in a list with length 5, index variable will have a value of from 0 up to 5 (exclusively).

```
#multiply the value of diagnosis accuracy by 100 to get the percent value with two point decimal precision
#concatenate patient data with diagnosis_accuracy and add percent symbol
#convert it to string and add necessary "\t" values by calculating left-over spacing
if index == 1:
    diagnosis_accuracy = f"{(patient[index] * 100):.2f}%"
    patient_data += diagnosis_accuracy + int(math.ceil((max_word_spaces[index] - len(diagnosis_accuracy)) / 4)) * "\t"
```

If index is equal to 1, this means the index of the diagnosis accuracy of the patient in the patient list above. patient[index] expression will give a float value of diagnosis accuracy. Using f-string, this value is multiplied by hundred to get the percent value of it and represented by two decimal point precision in the above code. “%” character is added to this string to emphasize it is a percent value. In order to align this value in the table with other values, max_word_spaces list will be used. For example for diagnosis accuracy column in the table, maximum character length is defined as max_word_spaces[1] equal to 12. Accuracy of the diagnosis will have 6 characters (such as 99.99%). The left-over 6 character spaces will be occupied by “\t” characters. This will be performed in according to code above. Generally 1 “\t” value will occupy a 4 space character. When the left-over space is divided by 4 and using math.ceil() function, it will be one upped to the nearest integer. This will produce the desired number of “\t” characters, which concludes with aligned columns with entries. For instance, when 6 left-over space is divided to 4, it will give 1.5. Using math.ceil() function will round it to 2. Multiplying 2 with “\t” will end up with filling the left-over 6 spaces. This will produce aligned column entries. After that, the aligned patient information will be concatenated with patient_data variable to be outputted to output file later.

```
#check the information of patient is treatment risk or not
elif index == len(patient) - 1:
    #multiply the float value by 100 to get percent value by two percent decimal precision
    #if precision part is all zeros, drop the zeros and concatenate with patient_data
    if int(patient[index] * 100) - (patient[index] * 100) == 0:
        patient_data += f"{int(patient[index] * 100)}%"
    else:
        #if precision points not all zero, drop the ones at the right-side if possible
        #concatenate with the patient_data
        risk = f"{(patient[index] * 100):.2f}".rstrip("0")
        patient_data += f"{risk}%"
```

If the index is equal to the index of the last item in the patient list, following process will be executed. Last item in the patient list is the treatment risk value in float representation. It will be multiplied with 100 to get the percent value. Then converted to integer with int() function. If the subtraction of these two value equal to 0, that means it has trailing zeroes after decimal point. In this case the value that is converted to integer will be used to drop those zeroes at the end. Using f-string, the value is transformed to string and added a “%” character. After that, patient_data is concatenated with this value.

If the subtraction does not yield 0 in the above code, this means treatment risk value multiplied by 100 has nonzero values after decimal point, however, at the very last number still could be 0. Using f-string and rstrip() method, the value is rounded up to 2 decimal point precision and if there is zero at the end, it is strip from the string value. Afterwards, the string value is concatenated to patient_data with “%” character by using f-string in the code above.

```
    else:
        #concatenate the patient_data with the informations of the patient
        #add required "\t" characters according to the left-over spacing
        patient_data += patient[index] + int(math.ceil((max_word_spaces[index] - len(patient[index])) / 4)) * "\t"
#call write_file function to write the patient info to the file
write_file(outputs_file, patient_data)
#reset the patient_data value to empty string for next patient
patient_data = ""
```

If index is not equal to 1 or the index of the last item in the list patient, mentioned process for index equal to 1 will be executed using math.ceil() and max_word_spaces list according to the max character length of current column in the list. After that, string value will be concatenated to the patient_data variable. This will conclude the one iteration of the inner loop, which means, every information of a patient is added to the patient_data variable with appropriate "\t" values. Outside of the inner for loop, write_file() function will be called to output a row of information of a patient to the output file. Afterwards, patient_data will be reset to empty string for the next patient's information.

def remove(patient_name):

This function takes the name of the patient as a parameter. It is the only parameter and it is stored in patient_name variable as a list with single string inside of it. This function will get the patient's name and look for it in the patient_list list. If it finds a match, it will remove it from the list and output a message. Otherwise, it will output a message about patient's absence.

```
#define a function to remove a patient from records
def remove(patient_name):

    #set is_deleted to False to to search the records to find patient in it
    is_deleted = False
```

First of all, is_deleted variable is set to False to search the patient_name in patient_list. is_recorded() function won't be useful here because it only will return a boolean value about patient's state of being recorded. It won't supply must needed information for removal, such as index of the person in the patient_list.

```
#iterate over records to look for patient
for index, patient in enumerate(patient_list):
    #check if patient's name is in the records
    if patient[0] == patient_name[0]:
        #delete the person's record from the patient_list
        del(patient_list[index])
        #set the is_deleted to True to indicate person is deleted
        is_deleted = True
```

To find the index of the patient in the patient_list, enumerate() function will be useful. What enumerate does is that gives a counter to each value in each iteration and return them as a enumerate object. When start parameter not given to enumerate function, it will start it from 0 as default. For example, in each iteration index variable will get values from 0 to length of the patient_list (exclusively) and patient variable will get the list values from patient_list. When the first values inside of patient list an patient_name list match, it means that the patient is recorded in the patient_list and ready to be deleted. In this case, using del() expression with the index variable that has the index of the patient in the patient_list, patient's record is removed from the records. Afterwards, is_deleted variable is set to True to indicate the deletion is successful in the code above.

```
#if person is deleted, call the wrtite_file function to state the situation
if is_deleted:
    message = f"Patient {patient_name[0]} is removed."
    write_file(outputs_file, message)
#if is_deleted is False, then person was not in the list
#call write_message function to indicate the situation
else:
    message = f"Patient {patient_name[0]} cannot be removed due to absence."
    write_file(outputs_file, message)
```

If the patient is deleted, is_deleted variable has a value of True. In the code above, this situation will cause a call to the write_file() function with a message saying that patient is removed. It will output the message in the output file. However, if is_deleted variable has a value of False, this means that person was not in the patient_list. In this situation, write_file() function will be called with a message that saying person cannot be removed due to absence. It will output the message in the output file.

GLOBAL EXPRESSIONS AND DATA STRUCTURES

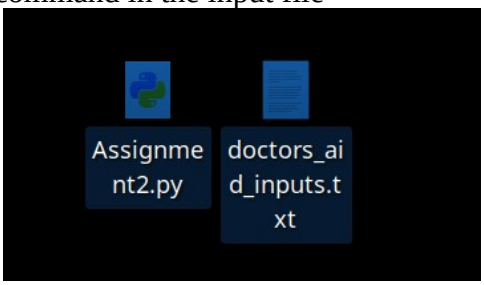
```
#create a patient_list to hold all the patient patients' records
patient_list = []

#set the input and output files name
inputs_file = "doctors_aid_inputs.txt"
outputs_file = "doctors_aid_outputs.txt"

#call the read_file function to kick-start the whole process
read_file(inputs_file)
```

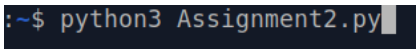
- patient_list is a multi-dimensional python list that holds all the patient records as a list inside of it.
- inputs_file and outputs_file are string variables that holds the name of input and output files, respectively.
- read_file(inputs_file) is a function call, that will kick-start all the process in this system.

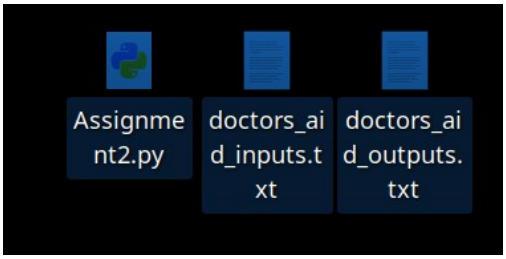
- The only functions in this program that has a return value are is_recorded() and probability() functions. probability() function will return a value if it is called with 0 as a parameter value in check. In other cases, functions will do some operations and output their results with write_file() function to an output file.

USER CATALOG

- Open the Terminal on the computer
- Using the cd command, go to the file location of Assignment2.py
- Be sure that Assignment2.py and input file is at the same location
- If not, modify the input file name in the .py file to indicate the location of the file
- Using command python3 Assignment2.py in the Terminal, execute the program
- Check the file location. A new file named of your choice in the .py file is created that outputs of each command in the input file







RESTRICTIONS ON THE PROGRAM

- Each command in the input file must be in the correct form. There is no check operation performed for it. For example, in create command, order must be create, patient name, diagnosis accuracy, disease name, disease incidence, treatment name, treatment risk. There must be no more than one " " character between words.
- Lower-case and Upper-case was not checked in the program. Commands in the input file must be all lower-case and names of the patients will be capitalized only. Otherwise, Hayriye and hayriye will be treated as two different patient.
- Diagnosis accuracy and treatment risk will be convertible to integers. Otherwise, program will raise an error while trying to convert them.
- Longer than maximum length words should not be entered as an input in the input file. This will crash the alignment of the list function. (longest ones in each section are Hayriye, Prostate Cancer, Targeted Therapy)

APPENDICES

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	19
There are several negative evaluations	...	...