BBM103

Introduction to Programming Laboratory I

Assignment 2

Doctor’s Aid Utility

B2210356060

Ömer Kayra Çetin

24.11.2022 23:00

# Analysis

We are expected to make a probability-based decision system called Doctor's Aid for clinicians. This aid utility will take inputs from a text file named "doctors_aid_inputs.txt". In the input file there are five commands for the user to use. These commands are "create", "remove", "list", "probability", and "recommendation". We are asked to define five functions which will carry out these commands and define two other functions which will read input from a file and write an output to a file. In the end we will produce an output file named "doctors_aid_outputs.txt" with respect to the given input.

Create command will activate a create function which will add patient information in a list format to a multi-dimensional python list. This multi-dimensional python list will be our patient data. We must check if the patient to be created is already on the list. If it is the case, return a message accordingly to that. If it is successfully added, return another message. This message will be added to the output file.

Remove command will activate a remove function which will remove patient information from our patient data. It will return a message output whether it is successfully removed or the patient is absent. This message will be added to the output file.

List command will activate a list function which will return the current patient data formatted to be a table. This table will be added to the output file.

Probability command will activate a probability function which will return the patient's real risk of having the disease calculated by using the patient's information in patient data. If there is no data for the patient, it will return another message. Returned output will be added to the output file.

Recommendation command will activate a recommendation function which will return a message depending on the patient's probability of having the disease and risk of treatment which is found in patient data. If there is not such a patient, it will return another message. The output will be added to the output file.

Read input function will read the input file and write output function will write output to the output file. Write output function will be used with command functions to write their outputs to the output file.

# Design

## Reading Input

Input file will be in a txt file format so we can open the file and read from it using readlines() function.

## Writing Output

Output file will be in a txt file format so we can append to it using write() function. All of the return messages from commands will be added to the output file by this way.

## Creating Multi-dimensional Built-in Python List

In the input file, create command will add a patient to the multi-dimensional list which is the main patient data. In the create command lines, after the create command, there is the data for the patient. We can separate the data from create command, then we can separate the data from its commas to add to the patient data as a list.

## Remove Patient

We can just look for the patient's name to be deleted in patient data using a for loop. If it is there, remove it and return a message. If it is not there return a not found message.

## Listing Patient Data

For listing patient data, we can first write heading lines. Then write a for loop to write all the data in the multi-dimensional patient data list. In the process we can define a liner function to line every information with its correct heading. Liner function will take the information string and the number of characters it should have. The function will add spaces to it until it fits the designed character number.

## Probability Calculation

With using confusion matrix and patient information in patient data, I use an equation to calculate it up to two decimal places. Then return a message including probability percentage of having the disease. If there is no patient in that name, it will return an error message.

## Recommending Treatment

By comparing treatment risk, which is found in the patient data, and the patient's probability of having the disease, the recommendation function returns a message. If treatment risk percentage is higher than the disease probability, it will suggest not to have treatment. If it is lesser, it will suggest to have treatment. If it can't find the patient, it will return an error message.

# Programmer's Catalogue

## The Overall Time Spent for Implementing

I spent approximately one hour for analyzing the problem. Designing, implementing, testing, and reporting happened simultaneously and repeatedly, and it required seven to eight hours to complete.

## Environmental and Global Variables

```
# Environmental Variables
CURRENT_DIR_PATH = os.getcwd()
INPUT_FILE_NAME = "doctors_aid_inputs.txt"
INPUT_FILE_PATH = os.path.join(CURRENT_DIR_PATH, INPUT_FILE_NAME)
OUTPUT_FILE_NAME = "doctors_aid_outputs.txt"
OUTPUT_FILE_PATH = os.path.join(CURRENT_DIR_PATH, OUTPUT_FILE_NAME)
# Global Variables
patient_data = []  # This is the multidimensional built-in python list.
```

I used five environmental variables to make implementing the code a bit easy. This way you can just change the input file name or output file name and the code will still work accordingly to that.

I implemented the multi-dimensional python list as a global variable named patient data to make it easier to add patients to it.

## Functions and Their Re-useability

### Read Input Function

```
def read_input():
    with open(INPUT_FILE_PATH, "r", encoding="utf-8") as file:  # Used encoding utf-8 to represent turkish letters.
        data = file.readlines()
    for i in range(len(data)):
        data[i] = data[i].replace("\n", "")  # Got rid of new line characters in lines.
    return data
```

This function reads the input from its path specified as an environmental variable in a reading mode using utf-8 encoding. The reason behind the utf-8 encoding is to represent Turkish letters. It reads lines to a data variable and the for loop will get rid of new line characters. In the end the function will return the data list. The function is re-usable as long as INPUT_FILE_PATH is declared.

## Write Output Function

```
def write_output(output):
    with open(OUTPUT_FILE_PATH, "a") as file:
        file.write(output)
```

The function is simply writing the given output argument to the output file's path declared as an environmental variable. The function is re-usable as long as OUTPUT_FILE_PATH is declared.

## Create Patient Function

```
def create(line_of_input_data):
    global patient_data
    # Split the line into two parts, command and patient info.
    # Take the patient info and split it from commas to get the info one by one in a list.
    current_patient_info = line_of_input_data.split(" ", 1)[1].split(", ")
    patient_name = current_patient_info[0]
    for patient in patient_data:
        if patient[0] == patient_name:
            return "Patient {} cannot be recorded due to duplication.\n".format(patient_name)
    patient_data.append(current_patient_info)
    return "Patient {} is recorded.\n".format(patient_name)
```

This function takes a line from the input data. The variable current_patient_info is declared by splitting the line into a list of two elements one being the create command and the other being the patient information using split(" ", 1) function. Then using the split(", ") function on patient information, we get a list of patient information separately. And I declare another variable named patient_name as the first element of current_patient_info because it is where the name information is. Using a for loop to check if the patient has already been created, if that's the case return an error message in a string format. If that's not the case append the patient information to the global patient_data which is made global in the first line. It returns a confirmation message. This function is re-usable on where we detect a create command in the input file.

## Remove Function

```
def remove(name):
    for patient in patient_data:
        if patient[0] == name:
            patient_data.remove(patient)
            return "Patient {} is removed.\n".format(name)
    return "Patient {} cannot be removed due to absence.\n".format(name)
```

This function checks the patient data with a for loop to detect the patient wanted to be removed. The name is taken as an argument. If it finds the patient, it removes it from patient data and returns an approval message. If it doesn't find, it returns an error message. This function is re-usable on this project whenever we want to remove a patient.

## List Patients Function

```
def list_patients():
    line1 = "Patient Diagnosis\tDisease\t\t\tDisease\t\tTreatment\t\tTreatment\n"
    line2 = "Name\tAccuracy\tName\t\t\tIncidence\tName\t\t\tRisk\n"
    line3 = "-------------------------------------------------------------------------------\n"
    write_output(line1)
    write_output(line2)
    write_output(line3)

    # Defining a liner function to line the words to correct spots.
    # Adds tab to string until it reaches its designed character number.
    def liner(string, length):
        space_needed = length - len(string)
        while space_needed > 0:
            string += "\t"
            space_needed -= 4
        return string
```

This function takes no arguments. In this part of the function, I declare the heading lines and add them to the output file using write_output function. Then I declare a liner function which is used for lining the words in the patient_data list before adding them to output file. The liner function takes a string and a specified length as arguments and returns the string added tabs until the string takes up the same place as length.

```
for patient in patient_data:
    patient_lines = ""
    patient_copy = patient.copy()  # Creating a copy in order not to mess with my actual data.
    # Get diagnosis accuracy to the wanted format.
    patient_copy[1] = format(float(patient[1])*100, ".2f")+"%"
    # Get treatment risk to the wanted format.
    patient_copy[5] = str(round(float(patient[5])*100))+"%"
    for i in range(len(patient)):
        # Check the index to correctly line the new line.
        if i == 0:
            patient_copy[0] = liner(patient_copy[0], 8)
        elif i == 1:
            patient_copy[1] = liner(patient_copy[1], 12)
        elif i == 2:
            patient_copy[2] = liner(patient_copy[2], 16)
        elif i == 3:
            patient_copy[3] = liner(patient_copy[3], 12)
        elif i == 4:
            patient_copy[4] = liner(patient_copy[4], 16)
        elif i == 5:
            patient_copy[5] = patient_copy[5] + "\n"
        # With every iteration add the newly created line to patient lines.
        patient_lines += patient_copy[i]
    write_output(patient_lines)
```

In this part of the function, I loop for patients in the patient data. Then I create an empty string named patient_lines to add lines to be outputted. Then I copy the patient data in order not to change it for the later use in code. Then I get the diagnosis accuracy and treatment risk to the wanted format and declare them back to the copied patient. After that, I create a for loop to add each element to the patient lines and by checking the index number I correctly line them to the wanted format by using the liner function I declared earlier in the function. After the loop has ended, I write the output to the output file using write_output function.

## Probability Function

```
def probability(name):
    for patient in patient_data:
        if patient[0] == name:
            cancer_people = int(patient[3].split("/")[0])  # Get the numerator in disease incidence.
            whole_population = int(patient[3].split("/")[1])  # Get the denominator in disease incidence.
            misdiagnose_percentage = 100 - round(float(patient[1])*100, 2)  # Calculate wrong diagnose percentage.
            # Make the necessary calculations to find the disease probability percentage.
            probability_percentage = (cancer_people/((whole_population*misdiagnose_percentage)/100+cancer_people))*100
            probability_percentage = round(probability_percentage, 2)  # Round it to two decimal places
            if probability_percentage == int(probability_percentage):  # Drop the zeros in the decimal part
                probability_percentage = int(probability_percentage)
            # Make the return statement a list of two elements because recommendation function
            # Will need cancer probability percentage as a float, as in the second element.
            return ["Patient {} has a probability of {}% of having {}.\n".format(
                name, probability_percentage, patient[2].lower()), probability_percentage]
    return ["Probability for {} cannot be calculated due to absence.\n".format(name)]
```

This function takes a patient name as an argument. Then in a for loop, it checks if it matches any patient name in patient data. If it finds a match, it gets numbers needed for the calculation such as cancer_people, whole_population and misdiagnose_percentage. In the comments they are explained. For actually calculating probability percentage we do a couple of mathematical operations. First let’s divide cancer_people by whole population times misdiagnose percentage, divided by 100 and then multiplied by cancer people. Then we can multiply that with 100 and find the probability percentage. At last, we round it to two decimal places. Additionally, I use an if statement to check if the decimal part of the probability percentage is zero. If so, I drop them declaring it to int type of itself. The function will return a list of two elements, first one being an approval message and second one being the probability percentage as a float rounded to 2 decimal points. This is because the recommend function will need the probability percentage. If it doesn’t find a matching name in the patient data, it returns a list with one element which is an error message. The reason for its being a list is for consistency for later use.

## Recommendation Function

```
def recommendation(name):
    for patient in patient_data:
        if patient[0] == name:
            treatment_risk = float(patient[5])*100  # Get treatment risk in a percentage form.
            cancer_percentage = probability(name)[1]  # Get percentage of cancer probability from probability function.
            if treatment_risk > cancer_percentage:
                return "System suggests {} NOT to have the treatment.\n".format(name)
            else:
                return "System suggests {} to have the treatment.\n".format(name)
    return "Recommendation for {} cannot be calculated due to absence.\n".format(name)
```

This function takes patient name as an argument. In a for loop it checks if the name matches any name in the patient data. If it finds one, then it gets treatment risk and cancer percentage explained in comments. Then checking which one is larger, it produces an output message. If it doesn't find any match, it will return an error message.

## Declaring Input Data

```
input_data = read_input()
```

Using the read input function, I declare an input_data variable for it to be used in main loop.

## Main Loop

```
for line in input_data:
    command = line.split(" ", 1)[0]  # Seperate commands as they are the first word a line has.
    # Check commands to use the right functions with their right inputs.
    if command == "create":
        write_output(create(line))
    elif command == "remove":
        write_output(remove(line.split()[1]))
    elif command == "probability":
        # Probability function returns lists as outputs due to its usage in recommendation function.
        # So we just need the zero index elements in both of its outputs.
        write_output(probability(line.split()[1])[0])
    elif command == "recommendation":
        write_output(recommendation(line.split()[1]))
    elif command == "list":
        list_patients()
```

This is essentially where the program works. It iterates the lines in input data using a for loop. It gets commands first in a command variable, how is explained in the comment. Then it checks which command it is by an if/elif statement block. After that, it uses the right function with their right inputs to carry out the command. In every command except list is used in write output function to write the output returned by them. List is an exception

because of the way I implemented it. List function uses write output function in its body. Any inconvenience situation is explained in the comments. When the loop ends, it indicates that we are done writing outputs and our output file is ready to be looked at.

## Used Data Structures

```
patient_data = []
```

This is my multi-dimensional python list. It gets patient information in a list where each element is a different information. This makes it some kind of a two dimensional array. This is the main list when we want to apply some command to a patient.

```
input_data = read_input()
```

This is my input data list. It gets values of all the lines in the input file as elements. Its purpose is to be used in main loop to correctly carry out commands.

# User Catalogue

## User Manual

```
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
```

1. First create an input list named “doctors_aid_inputs.txt”.

2. You have five options when writing a line.

- Create command:

This creates a patient. You first write create, then give the information of the patient in the order of: patient name, diagnosis accuracy, disease name, disease incidence, treatment name, treatment risk.

- Remove command:

Write remove and the patient you want to remove.

- List command:

Write list and it lists all the patient’s information.

- Probability command:

Write probability and the patient’s name that you want to learn the probability of having the disease.

- Recommendation command:

Write recommendation and the patient’s name that you want to learn the system’s opinion of whether to have the treatment or not.

3. Fill your file using these five commands.

4. Put the file in the same directory the program is in.

5. Run the program using a code editor or IDE.

6. The program will give an output accordingly the inputs you wrote.

# Restrictions on The Program

- If the inputs are not written properly, there is a high chance that you will encounter with an error.

# 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 | ... | - |