ECE 122: Introduction to Programming for ECE- Spring 2021

Project 2: Fun with DNA
(a complete procedural programming example)

Due Date: **Deadline: see website, class policy and moodle for submission**
This is an individual project (discussions are encouraged but no sharing of code)

Description

The goal of the project is to design and implement a basic medical patient database where administrators (you) will be able to inquire the current list of patients, add or remove patients, compare similarity in DNA strand between patients, and analyze certain (medical) conditions.

DNA molecules are quasi one-dimensional structures with a diameter of 2nm, consisting of two separate strands in a double helix form. The two strands are held together by pairing between the nitrogenous bases in the nucleotides of each strand. The nitrogenous base of a DNA nucleotide can be one of four different molecules: Adenine (A), Guanine (G), Thymine (T), and Cytosine (C). From the National Human Genome Research Institute: *Sequencing DNA means determining the order of the four chemical building blocks - called "bases" - that make up the DNA molecule. The sequence tells scientists the kind of genetic information that is carried in a particular DNA segment. For example, scientists can use sequence information to determine which stretches of DNA contain genes and which stretches carry regulatory instructions, turning genes on or off. In addition, and importantly, sequence data can highlight changes in a gene that may cause disease... The human genome contains about 3 billion base pairs that spell out the instructions for making and maintaining a human being... Researchers now are able to compare large stretches of DNA - 1 million bases or more - from different individuals quickly and cheaply. Such comparisons can yield an enormous amount of information about the role of inheritance in susceptibility to disease and in response to environmental influences. In addition, the ability to sequence the genome more rapidly and cost-effectively creates vast potential for diagnostics and therapies.*

We will work with the following database which contains 16 patients, and we will limit the analysis of their DNA strand to a particular segment of size 20.

ID	Name	Age	DNA-strand (20 length)
1	Andrea	37	GGATCACAGTCTACACTGCT
2	Bob	28	CACTCCAACCCCGGCCCCTG
3	Brooke	34	AGTCCGAGGAGAGGGTGCTT
4	Connor	27	CAGAGTATGTATACCACTGG
5	James	25	GTAGGATACGGCGGAGGGCA
6	Jenna	44	CGTCAATACGGTTCAATGCC
7	John	45	CTACTGCATGCTCTTGTGGT
8	Julie	37	TCATCTGCATGGAGAGGGTG
9	Kate	48	GGCATGGGTGGGGGTGCTGG
10	Keith	28	CCCGTGATCTGGACCTCCCA
11	Kelly	25	TCCACAGCTCATTGTACCGA
12	Luke	33	GTGTAGAGAGGGGCTTGTCC
13	Mark	34	TTCCAGATAGCGTTTCTGTT
14	Pat	26	TCGGTGTAGGTGCTAATCGA
15	Taylor	30	CTATGCTACTGCGGTTAACG
16	Tony	55	GGGATGGCAAGTACATTTTT

The project must include three files:

1. `Patient.py` file/module that contains the user-defined type (object data) Patient.
2. `dna_tool.py` file/module that contains **all** the necessary functions to operate the patient database. It also contains a main function that can be used for debugging.
3. `project2.py` file containing the main program.

Submission/Grading Proposal

Only **one zip file** must be submitted on moodle. Make sure you know how to create and submit a zip file before the submission date (and that your zip file is no empty once submitted). This project will be graded out of 100 points:

1. Your program should implement all basic functionality/Tasks and run correctly (90 points).
2. Overall programming style: program should have proper identification, and comments. (10 points).

The project is designed to be incremental, you can then debug, test and run your code after each new task/option is implemented. However, after Task 1 done all the other Tasks/options can be completed in any order. Use your preferred IDE to read, write and save your files. However, make sure your program is running using the command prompt. Do not forget to comment your code. Make sure you obtain the **same output** for the **same input** in the examples (this includes syntax, blank spaces, and skipping blank lines). Your program will also be tested with different inputs by the graders. Finally, you are free to consider additional functions if you wish to do so. However, you cannot use programming concepts different than the ones we have seen in Chapter 2: if, for/while, functions, print/input, and data object. This means no methods associated with String or List as we will see in Chapter 3 (such as append, insert, remove, etc.).

Overview of the main program functionality

Task-0- [5pts]

At the first execution of the program `project2.py`, the output should include a menu containing 7 options:

```
========================================
DNA "Analyzer" and Patient Management Tool
========================================

1-List; 2-Info; 3-Remove; 4-Insert; 5-Compare; 6-Compare all; 7-Analyze
Command (Enter to exit):
```

If no option is selected (just press Enter), the program stops. And it should print:

```
Thanks for using this tool
Come back soon!
```

How to proceed?

1. In the `project2.py` file: display the welcoming message "DNA Analyzer and Patient...." etc.
2. Use a call to a function `display_menu` that will print the menu.
3. The function to print the menu, `display_menu`, is already provided to you in the file `dna_tool.py`;
4. Consider using a `while` loop (such as `while true`) that will keep printing the menu and keep asking the user to enter a new command; this while loop should exit if you just press enter and display the goodbye message seen in the example above; Hint: look at Project 1.
5. We note that the header of the main file already contains the instruction `import dna_tool as dna` which will allow you to call the functions in the `dna_tool.py` file using the dot operator.

Task-1- [20pts]

Let us now see what should happen when option 1 is selected.

```
Command (Enter to exit): 1
        Name    age     DNA-strand (20 length)
-------------------------------------------------------------
1       Andrea  37      GGATCACAGTCTACACTGCT
2       Bob     28      CACTCCAACCCCGGCCCCTG
3       Brooke  34      AGTCCGAGGAGAGGGTGCTT
4       Connor  27      CAGAGTATGTATACCACTGG
5       James   25      GTAGGATACGGCGGAGGGCA
6       Jenna   44      CGTCAATACGGTTCAATGCC
7       John    45      CTACTGCATGCTCTTGTGGT
8       Julie   37      TCATCTGCATGGAGAGGGTG
9       Kate    48      GGCATGGGTGGGGGTGCTGG
10      Keith   28      CCCGTGATCTGGACCTCCCA
11      Kelly   25      TCCACAGCTCATTGTACCGA
12      Luke    33      GTGTAGAGAGGGGCTTGTCC
13      Mark    34      TTCCAGATAGCGTTTCTGTT
14      Pat     26      TCGGTGTAGGTGCTAATCGA
15      Taylor  30      CTATGCTACTGCGGTTAACG
16      Tony    55      GGGATGGCAAGTACATTTTT

1-List; 2-Info; 3-Remove; 4-Insert; 5-Compare; 6-Compare all; 7-Analyze
Command (Enter to exit):
```

Here the entire list of current patients is displayed, along with some info: name, age, and a sample of their DNA strand. At the end the menu selection is printed again and the program is waiting for you to make another selection.

How to proceed? —read everything up to the end of the section before coding—

1. In the `project2.py` file you first need to include a call to a function `initialize` that will return a list of patients (each item of the list is a data objects of type Patient...explained below). Option 1 must then be implemented inside the `while` loop. The option should include a call to function `display` (you can use as argument the list of patients, there should be no return value). Hint: look at the quiz app lec-2.9.

2. In the `Patient.py` file. You need to implement a class that defines the type data object `Patient`. The latter should include the constructor (i.e. function `__init__`). You will need several attributes, you could use for example: `name`, `age`, and `strand`. All the attributes could be initialized to `None` by default.

3. **Remark:** the header of the file `dna_tool` contains the instruction `from Patient import Patient` which will allow you to use the Patient data type. It also contains the instruction `import random` since the random function is used by the function `random_base` (explained below). The header includes `random.seed(5)` which sets the seed for reproducibility of results. The random module will be discussed in detail later in class, but you do not need to know how does it work for this project. Finally, the header includes a global variable `MAX_STRAND` set to 20, that will be kept unchanged in the entire project.

4. In the `dna_tool.py` file, you need to Implement the `initialize` function that creates and returns a list of Patient data objects. Ideally we would like to read all the patients data from a file (so we could easily consider hundreds of patients if needed) but we will do that later in the semester. Here, you will need to instantiate by hand all the 16 patients presented at the beginning of the project. However, the DNA strand should *not* be hard coded by hand but generated randomly. The file includes the function `random_base` that returns a base (as a string) chosen at random between (A,C,T,G). To generate the strand for each patient, you will need to implement a new function `random_strand` that generate a `MAX_STRAND` (20) long size string by successively appending to the right of the generated string the result of a call to `random_base`. Since the random seed as been hard-coded to be equal to 5 at the beginning of the file, your random results should be reproducible (same results for all code execution) and exactly equal to what is provided in the beginning of the project (a good way to check). Your code will be tested using different random seed by the graders.

5. In the `dna_tool.py` file, you also need to implement the function `display` that displays all the patient information as presented in the output example (you can use `\t` to separate each attribute).

6. **A necessary (but not sufficient) condition for option 1 to work:**
   you can first start by executing directly the file `dna_tool.py` where you can gradually uncomment the code in the main function to test successively: the function `random_strand`, the class `Patient`, the function `display`. Here is the output that you should obtain:

```
****TEST the random_strand function****
GGATCACAGTCTACACTGCT

****TEST the class Patient****
```

```
Tom 20 CACTCCAACCCCGGCCCCTG

****TEST the display function****
        Name    age     DNA-strand (20 length)
-------------------------------------------------------------
1       Tom     20      CACTCCAACCCCGGCCCCTG
2       Lucy    25      TCTTGTAAACTCGGAAACTG
```

Task-2- [15pts]

When option 2 is selected, the program should give you back some statistics: the number of patients, the ratio of patients ranked by ages, the mean values of the employees' ages.

```
Command (Enter to exit): 2
#Patients 16
<20: 0.0%
20's: 37.5%
30's: 37.5%
40's: 18.75%
50's: 6.25%
>=60: 0.0%

Age Mean: 34.75

1-List; 2-Info; 3-Remove; 4-Insert; 5-Compare; 6-Compare all; 7-Analyze
Command (Enter to exit):
```

How to proceed?

1. In the `project2.py` file: implement the option 2 that contains a call to a function `info` (you can use as argument the list of patients, there should be no return value)

2. In the `dna_tool.py` file: the function `info` that displays the employee statistics as presented in the output example. You will need a for loop that scans through all Patient objects in the list.

3. **Condition for option 2 to work:**
   By executing directly the file `dna_tool.py` and keep uncommenting the code in the main function to test the function `info`. Here is the output that you should obtain:

```
****TEST the random_strand function****
GGATCACAGTCTACACTGCT

****TEST the class Patient****
Tom 20 CACTCCAACCCCGGCCCCTG
```

```
****TEST the display function****
        Name    age     DNA-strand (20 length)
-------------------------------------------------------------
1       Tom     20      CACTCCAACCCCGGCCCCTG
2       Lucy    25      TCTTGTAAACTCGGAAACTG

****TEST the info function****
#Patients 2
<20: 0.0%
20's: 100.0%
30's: 0.0%
40's: 0.0%
50's: 0.0%
>=60: 0.0%

Age Mean: 22.5
```

Task-3- [5pts]

Using option 3, you will have the possibility to remove a patient from the list, if you provide his id number. Let us suppose Kelly decides to be removed from the database. Here we see the output results after option 3 is selected, followed by 1 and 2.

```
 Command (Enter to exit): 3
Who do you want to remove (enter number): 11

1-List; 2-Info; 3-Remove; 4-Insert; 5-Compare; 6-Compare all; 7-Analyze
Command (Enter to exit): 1
        Name    age     DNA-strand (20 length)
-------------------------------------------------------------
1       Andrea  37      GGATCACAGTCTACACTGCT
2       Bob     28      CACTCCAACCCCGGCCCCTG
3       Brooke  34      AGTCCGAGGAGAGGGTGCTT
4       Connor  27      CAGAGTATGTATACCACTGG
5       James   25      GTAGGATACGGCGGAGGGCA
6       Jenna   44      CGTCAATACGGTTCAATGCC
7       John    45      CTACTGCATGCTCTTGTGGT
8       Julie   37      TCATCTGCATGGAGAGGGTG
9       Kate    48      GGCATGGGTGGGGGTGCTGG
10      Keith   28      CCCGTGATCTGGACCTCCCA
11      Luke    33      GTGTAGAGAGGGGCTTGTCC
12      Mark    34      TTCCAGATAGCGTTTCTGTT
13      Pat     26      TCGGTGTAGGTGCTAATCGA
14      Taylor  30      CTATGCTACTGCGGTTAACG
15      Tony    55      GGGATGGCAAGTACATTTTT

1-List; 2-Info; 3-Remove; 4-Insert; 5-Compare; 6-Compare all; 7-Analyze
Command (Enter to exit): 2
#Patients 15
<20: 0.0%
20's: 33.33333333333333%
30's: 40.0%
40's: 20.0%
```

```
50's: 6.666666666666667%
>=60: 0.0%

Age Mean: 35.4

1-List; 2-Info; 3-Remove; 4-Insert; 5-Compare; 6-Compare all; 7-Analyze
Command (Enter to exit):
```

As soon as you enter Kelly ID `11`, her entry is removed from the list of employees in the database.
How to proceed?

1. In `project2.py`, you can use the built-in `del` function (seen in class) to remove an item with a given index from a list (it will automatically left shift all the other items with higher indexes). Hint: see Project 1.
2. You will also test that the user number input makes sense (should be between 1 and the number of patients), if not nothing happens.

Task-4- [10pts]

Now with Option 4, we would like to add a new patient to the database. Here is an example (option 4, followed by 1 and 2):

```
Command (Enter to exit): 4
Enter Name: Yoda
Enter Age: 400
Enter DNA strand : TTTTTCCCCCGGGGGAAAAA

1-List; 2-Info; 3-Remove; 4-Insert; 5-Compare; 6-Compare all; 7-Analyze
Command (Enter to exit): 1
        Name    age     DNA-strand (20 length)
-------------------------------------------------------------
1       Andrea  37      GGATCACAGTCTACACTGCT
2       Bob     28      CACTCCAACCCCGGCCCCTG
3       Brooke  34      AGTCCGAGGAGAGGGTGCTT
4       Connor  27      CAGAGTATGTATACCACTGG
5       James   25      GTAGGATACGGCGGAGGGCA
6       Jenna   44      CGTCAATACGGTTCAATGCC
7       John    45      CTACTGCATGCTCTTGTGGT
8       Julie   37      TCATCTGCATGGAGAGGGTG
9       Kate    48      GGCATGGGTGGGGGTGCTGG
10      Keith   28      CCCGTGATCTGGACCTCCCA
11      Kelly   25      TCCACAGCTCATTGTACCGA
12      Luke    33      GTGTAGAGAGGGGCTTGTCC
13      Mark    34      TTCCAGATAGCGTTTCTGTT
14      Pat     26      TCGGTGTAGGTGCTAATCGA
15      Taylor  30      CTATGCTACTGCGGTTAACG
16      Tony    55      GGGATGGCAAGTACATTTTT
17      Yoda    400     TTTTTCCCCCGGGGGAAAAA

1-List; 2-Info; 3-Remove; 4-Insert; 5-Compare; 6-Compare all; 7-Analyze
```

```
Command (Enter to exit): 2
#Patients 17
<20: 0.0%
20's: 35.294117647058826%
30's: 35.294117647058826%
40's: 17.647058823529413%
50's: 5.88235294117647%
>=60: 5.88235294117647%

Age Mean: 56.23529411764706

1-List; 2-Info; 3-Remove; 4-Insert; 5-Compare; 6-Compare all; 7-Analyze
Command (Enter to exit):
```

I am not a geneticist, but clearly Yoda got some problems, he may not even be human.

How to proceed?

1. Basically, you will need to use a new function `add_new_patient` that uses the list of patients as arguments and return a new list of patients. This function will ask the user all the questions related to the patient (as shown in the example), it will create a new Patient and append it to the list of patients.

2. If the input length DNA strand is not equal exactly to `MAX_STRAND` (20), the program should complain and keep asking to enter an acceptable DNA.

3. **Condition for option 2 to work:**
   By executing directly the file `dna_tool.py` and keep uncommenting the code in the main function to test the function `add_new_patient`. Here is an example output of what you should obtain:

```
****TEST the random_strand function****
GGATCACAGTCTACACTGCT

****TEST the class Patient****
Tom 20 CACTCCAACCCCGGCCCCTG

****TEST the display function****
        Name    age     DNA-strand (20 length)
------------------------------------------------------------
1       Tom     20      CACTCCAACCCCGGCCCCTG
2       Lucy    25      TCTTGTAAACTCGGAAACTG

****TEST the info function****
#Patients 2
<20: 0.0%
20's: 100.0%
30's: 0.0%
40's: 0.0%
50's: 0.0%
```

```
>=60: 0.0%

Age Mean: 22.5

****TEST the add_new_patient function****
Enter Name: Spock
Enter Age: 156
Enter DNA strand : CATG
Bad input! -length must be 20
Enter DNA strand : CCCCCCCC
Bad input! -length must be 20
Enter DNA strand : ACTGACTGACTGACTGACTG
        Name    age     DNA-strand (20 length)
-------------------------------------------------------------
1       Tom     20      CACTCCAACCCCGGCCCCTG
2       Lucy    25      TCTTGTAAACTCGGAAACTG
3       Spock   156     ACTGACTGACTGACTGACTG
```

Task-5- [10pts]

Now with Option 5, we would like to compare the DNA strand between two patients. Let us run it a couple of times:

```
Command (Enter to exit): 5
First patient (enter number): 3
Second patient (enter number): 6
Patients 3 and 6 common strand is xGTCxxxxxxGxxxxxxxxx
They are similar at 20.0%

1-List; 2-Info; 3-Remove; 4-Insert; 5-Compare; 6-Compare all; 7-Analyze
Command (Enter to exit): 5
First patient (enter number): 12
Second patient (enter number): 16
Patients 12 and 16 common strand is GxGxxGxxAxGxxCxTxTxx
They are similar at 40.0%

1-List; 2-Info; 3-Remove; 4-Insert; 5-Compare; 6-Compare all; 7-Analyze
Command (Enter to exit):
```

The code returns a new common strand that contains the common base and replace everything that is not similar with "x". The code is also returning the percentage of similarity.

How to proceed?

1. You need to implement the function `compare` that accepts two arguments of type Patient and return the new common DNA strand.
2. You also need to implement the function `check_completeness` that accepts the new common strand as argument and return the correct percentage.

3. You will also test that the user number input makes sense (should be between 1 and the number of patients), if not nothing happens. Also if you use the same number twice, you will get 100% similarity.

4. By executing directly the file `dna_tool.py` to test these two functions (you can also comment back the TEST for Option 4 as I did), you should get:

```
****TEST the random_strand function****
GGATCACAGTCTACACTGCT

****TEST the class Patient****
Tom 20 CACTCCAACCCCGGCCCCTG

****TEST the display function****
        Name    age     DNA-strand (20 length)
-----------------------------------------------------------
1       Tom     20      CACTCCAACCCCGGCCCCTG
2       Lucy    25      TCTTGTAAACTCGGAAACTG

****TEST the info function****
#Patients 2
<20: 0.0%
20's: 100.0%
30's: 0.0%
40's: 0.0%
50's: 0.0%
>=60: 0.0%

Age Mean: 22.5

****TEST the compare function****
xxxTxxAAxCxCGGxxxCTG

****TEST the check_completeness function****
50.0
```

Task-6- [10pts]

Now with Option 6, we would like to compare the DNA strand between all patients and return some information about their DNA similarity.

```
Command (Enter to exit): 6
James vs Andrea 35.0%
Jenna vs Andrea 45.0%
Jenna vs James 45.0%
John vs Andrea 40.0%
John vs Jenna 35.0%
Julie vs Andrea 35.0%
Julie vs James 35.0%
```

```
Kate vs John 35.0%
Keith vs Bob 35.0%
Keith vs Connor 40.0%
Kelly vs Kate 40.0%
Luke vs Brooke 35.0%
Luke vs James 35.0%
Luke vs Kate 45.0%
Luke vs Keith 35.0%
Mark vs John 50.0%
Mark vs Luke 40.0%
Pat vs John 40.0%
Pat vs Keith 35.0%
Taylor vs Bob 45.0%
Taylor vs James 55.00000000000001%
Taylor vs Luke 35.0%
Tony vs Andrea 40.0%
Tony vs Brooke 35.0%
Tony vs Julie 35.0%
Tony vs Kate 40.0%
Tony vs Luke 40.0%

1-List; 2-Info; 3-Remove; 4-Insert; 5-Compare; 6-Compare all; 7-Analyze
Command (Enter to exit):
```

The code is returning all the two by two comparisons if they are greater than 33%.

How to proceed?

1. You need to implement the function `compare_all` that accepts the list of patients as argument and does not have any return statement. Hint: you may want to use a nested `for` loop, and use the functions `compare`, and `check_completeness`.
2. Display only the comparisons if they are greater than 33%.

Task-7- [15pts]

Now with Option 7, we are looking at the possibility to detect particular conditions in the DNA. Let us consider the following example where option 1 is selected followed by option 1 and 2.

```
Command (Enter to exit): 7
Which condition are you looking for: Diabetes
Enter sequence: CTA
Patients with the Diabetes condition:  25.0%

1-List; 2-Info; 3-Remove; 4-Insert; 5-Compare; 6-Compare all; 7-Analyze
Command (Enter to exit): 1
        Name    age     DNA-strand (20 length)  Diabetes
-------------------------------------------------------------------------------
1       Andrea  37      GGATCACAGTCTACACTGCT    True
2       Bob     28      CACTCCAACCCCGGCCCCTG        False
3       Brooke  34      AGTCCGAGGAGAGGGTGCTT        False
4       Connor  27      CAGAGTATGTATACCACTGG        False
```

```
5       James   25      GTAGGATACGGCGGAGGGCA        False
6       Jenna   44      CGTCAATACGGTTCAATGCC        False
7       John    45      CTACTGCATGCTCTTGTGGT        True
8       Julie   37      TCATCTGCATGGAGAGGGTG        False
9       Kate    48      GGCATGGGTGGGGGTGCTGG        False
10      Keith   28      CCCGTGATCTGGACCTCCCA        False
11      Kelly   25      TCCACAGCTCATTGTACCGA        False
12      Luke    33      GTGTAGAGAGGGGCTTGTCC        False
13      Mark    34      TTCCAGATAGCGTTTCTGTT        False
14      Pat     26      TCGGTGTAGGTGCTAATCGA        True
15      Taylor  30      CTATGCTACTGCGGTTAACG        True
16      Tony    55      GGGATGGCAAGTACATTTTT        False

1-List; 2-Info; 3-Remove; 4-Insert; 5-Compare; 6-Compare all; 7-Analyze
Command (Enter to exit): 2
#Patients 16
<20: 0.0%
20's: 37.5%
30's: 37.5%
40's: 18.75%
50's: 6.25%
>=60: 0.0%

Age Mean: 34.75
Diabetes: 25.0%

1-List; 2-Info; 3-Remove; 4-Insert; 5-Compare; 6-Compare all; 7-Analyze
Command (Enter to exit):
```

How to proceed:

1. You would need to add one more attribute in the class `Patient`, such as the Boolean `has_condition`. It can be used to display the information True/False in the list of patients.

2. You need to implement the function `find_pattern` that accepts the list of patients and the DNA sequence of the condition. The function should return how many patients have this condition (int). Inside the function you should also be able to modify the new attribute `has_condition` of each patient accordingly.

3. As you can see, the output of option 1 and option 2 are modified as well. Obviously the functions `display` and `info` will have to be modified to accept an additional argument related to the condition name. Note that Option 2 returns the percentage of patients who have this particular condition. ***Be careful*** to maintain backward compatibility, everything you have done so far should work the same way.

4. Unless you decide to do Option 8 (which is optional), if you select 7 again, here the condition name should change along with new information.

Some tips for finding the condition pattern for a given patient DNA (Remember you are not allowed to use any methods from the String class).

- You need to scan all the letter in the DNA string (from left to right) while extracting a subset of the same size than the condition pattern.

- compare the extracted subset with the pattern, and stop here if you find a match.
- Be careful of the corner case, if the pattern is located at the end of the DNA strand.

Task-8- Bonus [5pts]- no help from TA

If you select Option 7 again, you will keep appending the conditions and information related to them. Hint: you can use a list to record the conditions and use the `has_condition` attributes has a list of Boolean. Below is an example.

```
Command (Enter to exit): 7
Which condition are you looking for: Diabetes
Enter sequence: CTA
Patients with the Diabetes condition:  25.0%

1-List; 2-Info; 3-Remove; 4-Insert; 5-Compare; 6-Compare all; 7-Analyze
Command (Enter to exit): 7
Which condition are you looking for: Blue-eyes
Enter sequence: GGA
Patients with the Blue-eyes condition:  37.5%

1-List; 2-Info; 3-Remove; 4-Insert; 5-Compare; 6-Compare all; 7-Analyze
Command (Enter to exit): 7
Which condition are you looking for: Three-eyes
Enter sequence: AAA
Patients with the Three-eyes condition:  0.0%

1-List; 2-Info; 3-Remove; 4-Insert; 5-Compare; 6-Compare all; 7-Analyze
Command (Enter to exit): 1
        Name    age     DNA-strand (20 length)  Diabetes        Blue-eyes       Three-eyes
-------------------------------------------------------------------------------------------
1       Andrea  37      GGATCACAGTCTACACTGCT    True            True            False
2       Bob     28      CACTCCAACCCCGGCCCCTG    False           False           False
3       Brooke  34      AGTCCGAGGAGAGGGTGCTT    False           True            False
4       Connor  27      CAGAGTATGTATACCACTGG    False           False           False
5       James   25      GTAGGATACGGCGGAGGGCA    False           True            False
6       Jenna   44      CGTCAATACGGTTCAATGCC    False           False           False
7       John    45      CTACTGCATGCTCTTGTGGT    True            False           False
8       Julie   37      TCATCTGCATGGAGAGGGTG    False           True            False
9       Kate    48      GGCATGGGTGGGGGTGCTGG    False           False           False
10      Keith   28      CCCGTGATCTGGACCTCCCA    False           True            False
11      Kelly   25      TCCACAGCTCATTGTACCGA    False           False           False
12      Luke    33      GTGTAGAGAGGGGCTTGTCC    False           False           False
13      Mark    34      TTCCAGATAGCGTTTCTGTT    False           False           False
14      Pat     26      TCGGTGTAGGTGCTAATCGA    True            False           False
15      Taylor  30      CTATGCTACTGCGGTTAACG    True            False           False
16      Tony    55      GGGATGGCAAGTACATTTTT    False           True            False

1-List; 2-Info; 3-Remove; 4-Insert; 5-Compare; 6-Compare all; 7-Analyze
Command (Enter to exit): 2
#Patients 16
<20: 0.0%
20's: 37.5%
30's: 37.5%
40's: 18.75%
50's: 6.25%
```

```
>=60: 0.0%

Age Mean: 34.75
Diabetes: 25.0%
Blue-eyes: 37.5%
Three-eyes: 0.0%

1-List; 2-Info; 3-Remove; 4-Insert; 5-Compare; 6-Compare all; 7-Analyze
Command (Enter to exit):
```