MCT-242 : COMPUTER PROGRAMMING-I

using Python 3.9

Prepared By:

Mr. Muhammad Ahsan Naeem

YouTube Playlist

https://www.youtube.com/playlist?list=PLWF9TXck7O_wMDB-VriREZ6EvwkWLNB7q

©Department of Mechatronics and Control Engineering, University of Engineering and Technology, Lahore, Pakistan

LAB 15: RANDOM NUMBERS: CLO-4

Random number generation is required in many applications e.g. games. In rolling a regular six-sided die, the program should return a random number 1 to 6 and in playing card game a random card out of 52 should be generated. They are also needed in modern cryptographic protocols that provide computer security, communication privacy, and authentication.

Generating a true random number is not easy. In computational world, these numbers are generated by some deterministic computation, but they look random and are good enough for most of the applications that require random numbers. The programs which generates random numbers are known as pseudo-random number generators.

Random Module:

In Python we have a module named as **`random`** to generate random numbers. Inside **`random`** module we have a function named as **`random()`** that returns a random float between 0.0 and 1.0 (including 0.0 but not 1.0). Every time we run this function; we will (hopefully) get a different random number. Below is a code where this function is executed five time in a loop, and you can see five different numbers generated in output. Every time you run this program you will see different numbers.

```
import random
for i in range(5):
    x = random.random()
    print(x)
```

One sample output is:

```
0.04662754909050548
0.21505891170555735
0.0030661034653519660
0.6706308320354954
0.5576178113940513
```

To generate a random integer, we have two functions in **`random`** module namely **`randint(a,b)`** and **`randrange(a,b)`**. Both will generate a random integer and the only difference is that in **`randint(a,b)`** the random integer generated can be from **`a`** to **`b`** both inclusive while in **`randrange(a,b)`** the random integer generated is from **`a`** (inclusive) to **`b`** (not inclusive).

If we want to program 5 times roll of a six-sided dice, we can do it like:

```
import random
for i in range(5):
    x = random.randint(1,6) #or random.randrange(1,7)
    print(x)
```

Tasks:

[1] Write a program that will generate a random number between **5** and **10** (both inclusive) and will ask user to guess that number. If user guesses wrong, it will ask him to guess again until the user guesses the correct number. Also display number of trials user has taken to guess the number. **Important:** Once a random number is generated, it should not be changed during guesses from user.

Sample output is (Suppose computer generated 6):

```
I have selected a number between 5 and 10

Guess what is it: 5
No, it is wrong!
Guess what is it: 7
No, it is wrong!
Guess what is it: 8
No, it is wrong!
Guess what is it: 6
Yes, you guessed it right

You tried 4 times to guess the number
```

[2] Modify above program such that the random number generated is from 0 to 100 (both inclusive) and after each guess from user, if it is not correct, tell him the number is above the guessed value or below that.

Sample output is (Suppose computer generated 38):

```
I have selected a number between 0 and 100

Guess what is it: 5
No, it is ABOVE 5
Guess what is it: 22
No, it is ABOVE 22
Guess what is it: 44
No, it is BELOW 44
Guess what is it: 40
No, it is BELOW 40
Guess what is it: 35
No, it is ABOVE 35
Guess what is it: 38
Yes, you guessed it right

You tried 6 times to guess the number
```

[3] There are two regular dice, both are rolled, and sum of the outputs is noted. The player will repeat this process n number of times and the sum will be noted for each trial. If the sum is greater than or equal to 7, for more than 50% of the trials, the player wins. Otherwise, he loses. Write a program that will ask user for number of trials. The program should roll the two dice for that number of time and should display the output of each dice along with the sum. Finally, it will display whether the user won or lost according to the rule described above.
Note carefully in sample output that user is asked to enter some key for next trial. Key entered by user will be used nowhere but it's important to have it like that to keep user's interest.

Sample Output is:

```
Enter the number of Trials: 5

(Press any key for next trial)
Trial 1: Dice1=3 Dice2=4 Sum=7

(Press any key for next trial)
Trial 2: Dice1=2 Dice2=6 Sum=8

(Press any key for next trial)
Trial 3: Dice1=3 Dice2=4 Sum=7

(Press any key for next trial)
Trial 4: Dice1=5 Dice2=1 Sum=6

(Press any key for next trial)
Trial 5: Dice1=1 Dice2=5 Sum=6

The Sum>=7 occurred 3 times in 5 trials.
CONGRATULATIONS!!! You are WINNER
```

Another Sample output is:

```
Enter the number of Trials: 4

(Press any key for next trial)
Trial 1: Dice1=1 Dice2=3 Sum=4

(Press any key for next trial)
Trial 2: Dice1=3 Dice2=6 Sum=9

(Press any key for next trial)
Trial 3: Dice1=2 Dice2=1 Sum=3

(Press any key for next trial)
Trial 4: Dice1=4 Dice2=1 Sum=5

The Sum>=7 occurred 1 times in 4 trials.
```

```
SORRY!!! You LOST
```

[4] Make a simple Quiz application for kids to test their simple addition skill. The program should ask sum of two number (both operands in range 1-15 both inclusive) generated randomly and should display whether the entered answer is correct or not.

Sample Output is: (9+11= was displayed by the program and 20 is entered by user)

```
9+11=20
Correct!
```

Another Sample Output is: (12+15= was displayed by the program and 26 is entered by user)

```
12+15=26
Incorrect!
```

Now modify the code such that the question appears 10 times and at the end number of correct answers is also displayed.

[5] Modify above code in a way that now subtraction and multiplication questions are also asked. So, now there can be three types of questions; addition, subtraction or multiplication. The two operands are generated randomly (1-15) as previously and moreover the operation (+ , - or *) is also chosen randomly.

To choose the operation randomly, we can generate a random number 1-3 and use three `if` statements incorporating one operation each.

Again ask 10 questions and display the number of correct answers.

[6] **Probability of Getting same number on rolling two dice.**

What is the probability of getting same numbers on two dice rolls? Here is the sample space of the experiment:

1,1	**1,2**	**1,3**	**1,4**	**1,5**	**1,6**
2,1	**2,2**	**2,3**	**2,4**	**2,5**	**2,6**
3,1	**3,2**	**3,3**	**3,4**	**3,5**	**3,6**
4,1	**4,2**	**4,3**	**4,4**	**4,5**	**4,6**
5,1	**5,2**	**5,3**	**5,4**	**5,5**	**5,6**
6,1	**6,2**	**6,3**	**6,4**	**6,5**	**6,6**

The highlighted part shows the occurrence of similar numbers on both rolls. There are total 36 possibilities and out of those 6 have the same numbers. From your elementary knowledge of probability, you must know that the probability of above event is:

$$P(Same\ number\ on\ both\ dice) = 6/36 = 1/6 = 0.1667$$

The above method of finding the probability is **Theoretical** method. The other method to find probability is the **Experimental** method where we actually repeat the experiment and count the

number of occurrences of a particular event. Hence, in above case two dice must be rolled and occurrences of same number on both should be counted. But how many times the experiment must be repeated? Should we obtain results with two or three experiments?
Ideally, we should repeat the experiment infinite times to get the same result as of the theoretical calculations. But of course, practically it is not possible. In reality even repeating above experiment 200 times will take reasonable time. But here a computer program can help us repeat the experiment even a few hundreds of thousands times at the cost of few seconds!

So, write a program that will simulate the rolling of two dice and will check whether the two outputs are same or not. The program must ask user to enter the number of times he wants to repeat the experiment and then should count the number of times the event in question occurred. The program then must display the probability of the event.

Execute the program by entering different number of experiments starting from 10 to 100 and then 1000, 10,000, 100,000 etc. Observe carefully that the final answer of **Experimental** probability will get closer and closer to the **Theoretical** value i.e. 0.1667 as we increase the number of experiments.