PPFI PROJECT ASSIGNMENT

02

Monte Carlo Birthday Simulations: Demystifying the Birthday Paradox

Project Title:

Monte Carlo Birthday Simulations: Demystifying the Birthday Paradox

Objective:

The objective of this project is to implement a Python program that conducts Monte Carlo simulations to explore and analyze the probabilities associated with the Birthday Paradox. By allowing users to input group sizes, the program generates random birthdays and performs multiple simulations to determine the likelihood of two or more people sharing the same birthday within a given group. The project aims to enhance the understanding of probability concepts through practical experimentation and provide insightful visualizations of the simulation results.

The Program in Action:

When you run ***birthdayparadox.py***, the output will look like this:

```
Birthday Paradox, by Student_Name (Student_Registration_No.)
--snip--
How many birthdays shall I generate? (Max 100)
> 23
Here are 23 birthdays:
Oct 9, Sep 1, May 28, Jul 29, Feb 17, Jan 8, Aug 18, Feb 19, Dec 1, Jan 22,
May 16, Sep 25, Oct 6, May 6, May 26, Oct 11, Dec 19, Jun 28, Jul 29, Dec 6,
Nov 26, Aug 18, Mar 18
In this simulation, multiple people have a birthday on Jul 29
Generating 23 random birthdays 100,000 times...
Press Enter to begin...
Let's run another 100,000 simulations.
0 simulations run...
10000 simulations run...
--snip--
90000 simulations run...
100000 simulations run.
Out of 100,000 simulations of 23 people, there was a
matching birthday in that group 50955 times. This means
that 23 people have a 50.95 % chance of
having a matching birthday in their group.
That's probably more than you would think!
```

Hints to Prepare the Project:

- **Generate Random Birthdays:**
  - ✓ Create a function named **getBirthdays** to generate a list of random date objects for birthdays.
  - ✓ Utilize the **datetime** module to work with dates, ensuring that all birthdays share the same year for the simulation.
- **Check for Matching Birthdays:**
  - ✓ Develop a function named **getMatch** to identify a birthday that occurs more than once in the list.
  - ✓ Utilize comparisons between birthdays to detect matching occurrences.
- **User Input and Group Size:**
  - ✓ Modify the code to allow user input for the number of birthdays to generate.
  - ✓ Ensure that the user can input values up to 100, as indicated in the comments.
  - ✓ Validate user input to guarantee a valid group size for simulations.
- **Simulation Loop and Progress Updates:**
  - ✓ Optimize the simulation loop to provide more informative progress updates.
  - ✓ Consider adjusting the frequency of progress reporting and potentially the number of simulations.
- **Simulation Results and Statistics:**
  - ✓ Analyze the simulation results and display additional statistics.
  - ✓ Calculate and display the average number of matches across all simulations.

Requirements:

- **Implement the Birthday Paradox Simulation:**
  - ✓ Develop a Python script named ***birthdayparadox.py***.
  - ✓ Implement the Birthday Paradox simulation logic following the provided instructions.
- **Provide Accurate Simulation Results:**
  - ✓ Ensure the simulation accurately calculates and displays the probability of matching birthdays within a given group size.
  - ✓ Validate that the simulation produces meaningful outcomes for different group sizes.
- **User Input and Validation:**
  - ✓ Allow user input to determine the number of birthdays to generate in each simulation.
  - ✓ Validate user input to ensure it is a decimal value within the specified range (1 to 100).
- **Enhance Display of Generated Birthdays:**
  - ✓ Improve the display of generated birthdays for better readability.
  - ✓ Include month names and day numbers in the output.
- **Optimize Simulation Loop:**
  - ✓ Optimize the simulation loop for better progress updates.

- ✓ Consider adjusting the frequency of progress reporting and potentially the number of simulations.

- **Simulation Results and Statistics:**
  - ✓ Analyze the simulation results and display additional statistics.
  - ✓ Calculate and display the average number of matches across all simulations.

Instructions:

- **Create the Python Script:**
  - ✓ Develop a Python script named birthdayparadox.py.
- **Implement Simulation Logic:**
  - ✓ Follow the instructions provided in the initial code snippet to implement the simulation logic.
  - ✓ Ensure that the simulation accurately reflects the Birthday Paradox concept.
- **User Interaction and Replay Option:**
  - ✓ Allow users to input the number of birthdays to generate for each simulation.
  - ✓ Display the generated birthdays and check for matching occurrences.
  - ✓ Provide feedback on the simulation results.
  - ✓ Implement a loop that allows users to replay the simulation.
  - ✓ Ask the user if they want to run another simulation after each game.
- **Testing:**
  - ✓ Ensure the modified code runs without errors and produces meaningful results.
  - ✓ Test with different group sizes to validate the accuracy of the simulation.

Source Code (paste the code below)

Project Report Preparation: (Total = 15 points)

Understanding Birthday Paradox Simulation (1 points):

Explain the concept of the Birthday Paradox as implemented in the provided Python code. How does the simulation approach the problem, and what is the significance of conducting Monte Carlo experiments in this context?

Game Implementation (5 points):

- How is the user prompted to input the number of birthdays to generate in the simulation? Explain the modifications made to allow the user to input different group sizes, ensuring the input is valid.
- Describe how the program generates and displays random birthdays. Highlight any enhancements made to improve the display format, making it more readable.
- Analyze the current simulation loop. Discuss any optimizations made to provide more informative progress updates and potential adjustments to the number of simulations.

Guessing Mechanism (3 points):

- Explain the mechanism used to determine if there are two birthdays that match within a generated group. Discuss the role of the **getMatch** function in identifying matching birthdays.
- Explore the code related to the display of simulation results. How is the presence or absence of matching birthdays communicated to the user?

Replay Option (1 points):

Investigate the code related to replaying the simulation. How is the program designed to allow users to run the simulation multiple times without restarting the program?

Coding Style and Structure (1 points):

Evaluate the coding style and structure of the provided Python code. Comment on the clarity of variable names, code organization, and adherence to Python coding conventions.

Edge Cases and Testing (3 points):

Discuss any identified edge cases and the corresponding testing approaches applied to ensure the robustness of the simulation. How is the program tested with different group sizes?

Conclusion (1 points):

Provide a brief conclusion summarizing the key findings and improvements made during the implementation of the Birthday Paradox Simulation. Reflect on the significance of the project in enhancing understanding of probability concepts.

(Signature of the Faculty)	**(Signature of the Student)**
	Name:
Date:	**Registration No.:**
	Branch:
	Section