**6.1.1**

* Put a header line on the output CSV file to make it clear to those that open it where the values in each column came from.
* 6.1.1 got its own folder due to it having multiple files pertaining to it
  + ‘6.1.1 Main.py’’
    - The main code file
  + ‘6.1.1 Main Values Excel Workbook.xlsx’
    - The file with the uniform, normal, and lognormal Excel histograms
    - Also contains sample output from ‘6.1.1 Main.py’ on it’s first sheet: “6.1.1 Main Values”
      * Sheet name was purposely made the same as the output file name of ‘6.1.1 Main.py’, since that’s where the file values are from.

**6.1.2**

* Zero-padded the key numbers to the left in order to have each ‘-‘ bar of the output “Histogram” start at the same point.
* The rounding of each value generated by the random module functions is done within freqmap using int(round(randomly\_generated\_value))

**Monty\_Hall\_Problem\_Simulation Package**

* game module
  + Game holds the MontyHallGame class, which has
    - Variable prize to hold the number of the door with the prize behind it in the game
    - Function open\_door which returns the door that is not the prize door nor the door the input player has chosen, and
    - Function play\_game which returns True if the player’s final chosen door is equal to the prize door or False if it is not equal.
* player module
  + Player base class
    - Variables
      * player\_name – An identifier for the player object
      * door – The initial door that the player chooses
      * switch – Defines the player’s strategy as to whether they will switch doors when given the opportunity or not
        + True if the player will switch
        + False if the player will not switch
    - Functions
      * switch\_door\_decision – Simulates the player choosing whether to switch doors or not when given the opportunity based on the player object’s switch variable
  + AlwaysSwitchPlayer derived class
    - A Player object with the variable switch set to True
  + NeverSwitchPlayer derived class
    - A Player object with the variable switch set to False

**6.1.3**

* Did 10,000,000 simulations with an “Always Switch” strategy and 10,000,000 simulations with a “Never Switch” strategy. This was done to ensure a large number of simulations for each was performed.

**6.2.1**

* Had trouble getting the time taken in multiprocessing to be less than that of the single process. I believe this was due to me initially having do\_work loop through the result of monty\_hall\_sim (Within the input queue), put each item within it individually into the output queue, and then append each item in the output queue to the result list in main. To fix this, I let do\_work simply put the whole list generated by monty\_hall\_sim into the output queue and then used extend to add the list item in the output queue to the result list in main. This resulted in my program taking only about 10 – 13 seconds, whereas before it would take upwards of 5 to 6 minutes.
* Did 10,000,000 simulations with an “Always Switch” strategy and 10,000,000 simulations with a “Never Switch” strategy. This was done to ensure a large number of simulations for each was performed.
* Items are put into the input queue and the five processes are initialized before starting the timer. This was to follow the written instructions for the exercise that the processes be created and initialized in part a, while only parts b and c are timed. The input queue was set before timing so that the timing would focus solely on the child processes themselves.
* Part E tested number of processes, as well as the time they took for the never switch simulations and the always switch simulations. The total simulations overall was kept at 10,000,000 and the number of simulations for each process was divided evenly among the processes. (ie. 2 processes do 5,000,000 simulations each)

|  |  |  |
| --- | --- | --- |
| **Number of Processes** | **Never Switch Doors Time**  (In Seconds) | **Always Switch Doors Time**  (In Seconds) |
| 1 | 91.467 | 87.453 |
| 2 | 43.028 | 50.278 |
| 4 | 25.501 | 29.629 |
| 5 | 11.122 | 12.914 |
| 10 | 10.577 | 11.375 |
| 20 | 10.696 | 11.151 |
| 25 | 10.139 | 11.374 |
| 40 | 10.218 | 11.858 |
| 50 | 10.183 | 11.912 |
| 100 | 10.765 | 12.546 |

* + From the table, we can see that once 10 processes are used, the number of seconds taken for runtime become fairly consistent, hovering around 10 seconds for Never Switch Doors strategy simulations and 11 seconds for Always Switch Doors strategy simulatins, with only fractions of a second difference. Because of this, the code for part E stuck with using 10 processes.