monty-hall

January 18, 2025

1 MONTY HALL

[1]:
```python
import random

def initialize() -> tuple:
    """
    This function creates and returns a list of three integers. Two zeroes "0"␣
 ↪and a "1".
    The position of the "1" in the list is random.
    e.g: return [0,0,1]
    """
    doors = [0,0,1]
    random.shuffle(doors)
    return doors, doors.index(1)

def player_choice() -> int:
    """
    This function returns a random number between 0 and 2 (inclusive).
    It will represent the choice of the player among the three doors.
    It is between 0 and 2 because it's an index that will used to access the␣
 ↪choice in an array.
    """
    return random.randint(0,2)

def host_choice(doors: list, prize_index:int, player_index:int) -> int:
    """
    This is the door that the host reveals after the player chose a door.
    If a player chose a door the host should reveal another door with no prize.
    The index is between 0 and 2.

    Parameters
    ----------
    :param doors: a list of three elements. e.g. [1,0,0]
    :param prize_index: the index of the "1" in the list. e.g. 0
    :param player_index: the index of the door chosen by the player. e.g. If␣
 ↪the player chose the third door it's 2
    :return: the index of the door chose by the host. e.g. 1
```

```
    """
    return random.choice([i for i in range(3) if i != player_index and i !=␣
 ↪prize_index])

def get_alt_index(doors: list, player_index:int, host_index:int) -> int:
    """
    Given the player's choice and the host's choice, this helper function␣
 ↪returns the index of the remaining door.
    e.g. if the list is [0,1,0], the player's choice is 0 and the host's choice␣
 ↪is 2, this function returns 1.
    """
    return [i for i in range(3) if i != player_index and i != host_index][0]

def play_game(switching_prob=0):
    """
    You don't have to modify this function.
    switching_prob is the probability that the player switches the original␣
 ↪door that he chose.
    This function simulates the game once and returns the number behind the␣
 ↪final choice of the player.
    1 means that the player won a prize, 0 means that he didn't.
    """
    doors, prize_index = initialize()
    player_index = player_choice()
    host_index = host_choice(doors, prize_index, player_index)
    alt_index = get_alt_index(doors, player_index, host_index)

    final_index = random.choices([player_index, alt_index],
    weights=[1-switching_prob,switching_prob], k=1)[0]

    return doors[final_index]

def simulate(n=1000):
    """
    You don't have to modify this function either.
    It simulates the game n times and displays the probabilities of winning the␣
 ↪prize, for three scenarios.
    X=1 means that the player won the prize and X=0 means that he didn't.
    """
    result_1, result_2, result_3 = [], [], []
    for i in range(n):
        result_1.append(play_game(switching_prob=0))
        result_2.append(play_game(switching_prob=1))
        result_3.append(play_game(switching_prob=0.5))

    print(f"Keeping the original door")
```

```
    print(f"P(X=1) = {result_1.count(1)/len(result_1)}   | P(X=0) = {result_1.
 ↪count(0)/len(result_1)}")
    print("*"*50, "\n")
    print(f"Switch the door")
    print(f"P(X=1) = {result_2.count(1)/len(result_2)}   | P(X=0) = {result_2.
 ↪count(0)/len(result_2)}")
    print("*"*50, "\n")
    print(f"Randomly switch the door with a probability of 0.5")
    print(f"P(X=1) = {result_3.count(1)/len(result_3)}   | P(X=0) = {result_3.
 ↪count(0)/len(result_3)}")

if __name__ == "__main__":
    simulate(1_000)
```

```
Keeping the original door
P(X=1) = 0.317   | P(X=0) = 0.683
**************************************************

Switch the door
P(X=1) = 0.683   | P(X=0) = 0.317
**************************************************

Randomly switch the door with a probability of 0.5
P(X=1) = 0.524   | P(X=0) = 0.476
```