The Monty Hall Problem

Introduction

This game is based on a fairly well-known game show involving the contestant being given a choice to choose 3 doors. Behind one door is a new car and behind two others a goat. If the contestant chooses the door with the car behind it, they get to home with a brand new car! At first glance, this means there is a 1/3 chance of winning the car. The twist in this game is that after the contestant selects a door, the host will then reveal 1 of the other 2 doors that has a goat behind it and then gives the contestant the option to stay with their original door selection or to change to the remaining door left. At this point, our common sense would tell us that we now have a 50% chance of winning the car by staying or by switching. However, this is not the case. By going through this repeatedly, there is a clear advantage of one choice over the other to better your odds of winning the car.

Design and Implementation

I originally heard about the Monty Hall problem several years ago and enjoyed trying to understand the probabilities of the game. The design period was relatively short as I knew exactly how I planned on implementing this game. During the implementation I have found in a difficult spot of figuring out how to handle and proceed forwarding using the player's first door selection. Originally it would have been 6 very long different branching if statements that would have looked something like:

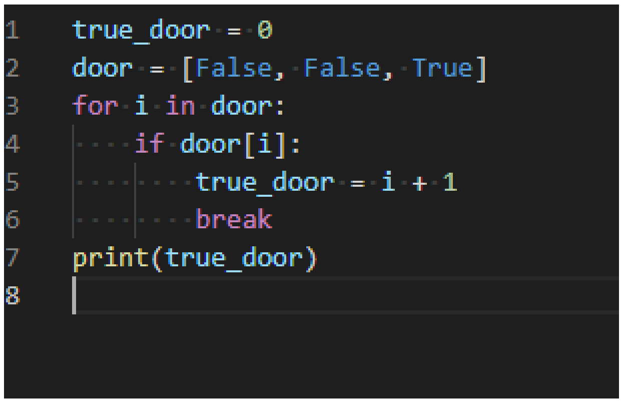
```
If picked_door == 1 and door1 == 'Win':

If picked_door == 1 and door1 != 'Win':

If picked_door == 2 and door2 == 'Win':

If picked_door == 2 and door2 != 'Win':

If picked_door == 3 and door3 == 'Win':

If picked_door == 3 and door3 != 'Win':
```

Ultimately that would've ended up making this program a lot longer than it needed to. I spent an hour trying to figure out if theres a better way and managed to reduce it down to only a single if and else statement. Simply,

```
if picked_door == 'Win'

        etc.

else:

        etc.
```

And the rest of the game branches from these 2 conditionals.

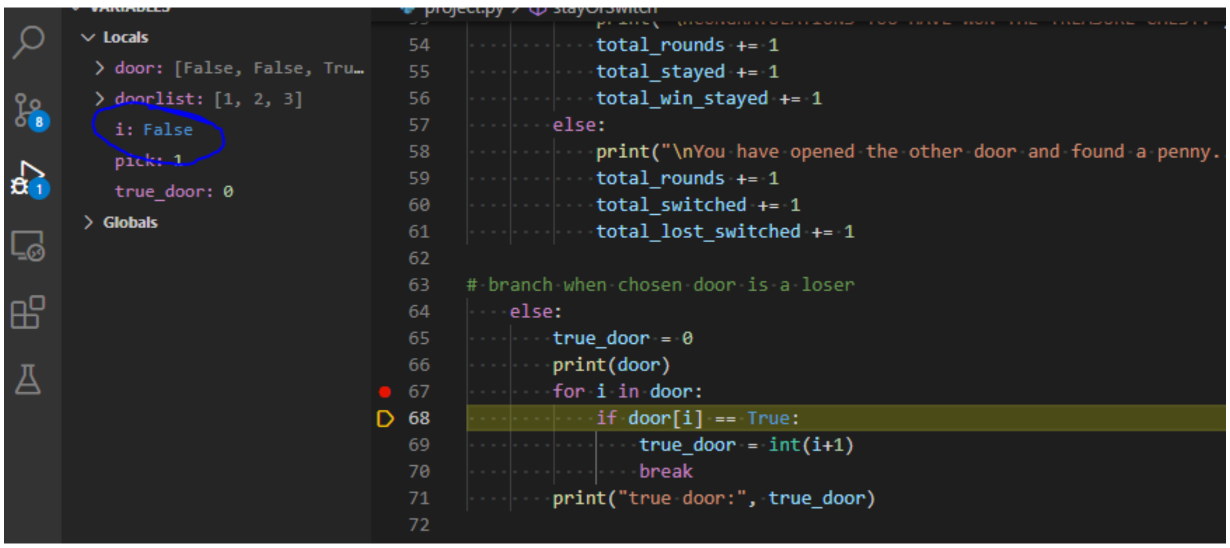
The biggest difficulty I faced was unexpectant behavior when trying to do a for loop in order to go through which of the 3 doors was the winning door.



```python
true_door = 0
door = [False, False, True]
for i in door:
    if door[i]:
        true_door = i + 1
        break
print(true_door)

```

From what we have learned from this course, you would think that the final value of true_door should be 3. However, when running this, the value of true_door continually comes out to be 0.

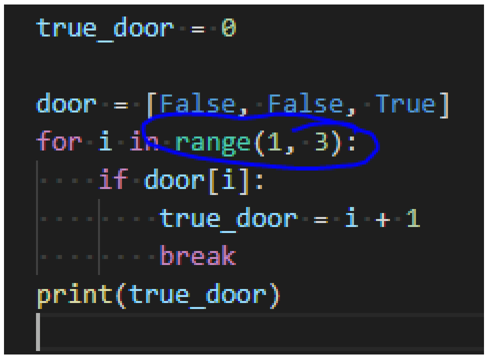
After a painstaking hour or two and with the help of vscode's built-in debug, I discovered that the value of i is NOT being treated as an integer in this instance. The i variable comes out to be a Boolean value. Since the elements in the list door (which is what this for loop is iterating over) are Booleans, Python was automatically (and stupidly) typecasting i as a Boolean instead of integer.



```
Locals
  > door: [False, False, Tru…
  > doorlist: [1, 2, 3]
    i: False
    pick: 1
    true_door: 0
> Globals
```

```python
            total_rounds += 1
            total_stayed += 1
            total_win_stayed += 1
        else:
            print("\nYou have opened the other door and found a penny.
            total_rounds += 1
            total_switched += 1
            total_lost_switched += 1

# branch when chosen door is a loser
    else:
        true_door = 0
        print(door)
        for i in door:
            if door[i] == True:
                true_door = int(i+1)
                break
        print("true door:", true_door)

```

The solution was to adjust the for loop and have it iterate through a simple range.



```
true_door = 0

door = [False, False, True]
for i in range(1, 3):
    if door[i]:
        true_door = i + 1
        break
print(true_door)
```

Conclusion

The challenge expressed above was definitely a learning experience. It reveals to me that there could be limitations on implementing a for loop depending on what type of variable data types you are iterating over. Definitely something to keep in mind for the next time. The shortcoming of my implementation of this game is that there is no twist or my own creative deviation to this already known game. If I was to choose to implement this game again, I probably would alter it so that its not just a carbon copy of the original Monty Hall game such as providing the player 100 doors to choose and then removing 98 of them which would clearly behoove the player to make the better choice and explain the probabilities illusion that would make players choose to stay on their original door.