

MONTY HALL PROJECT

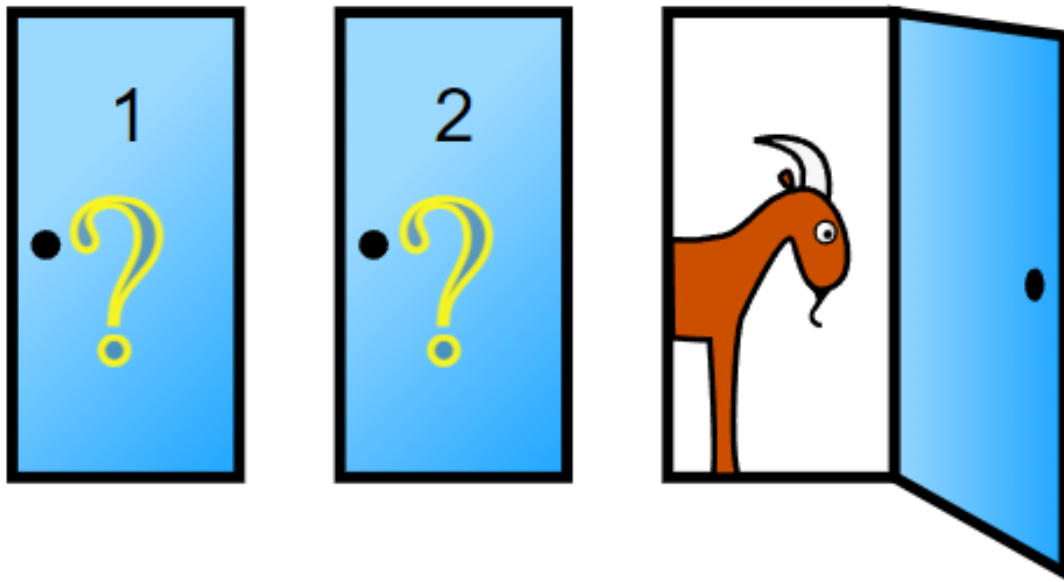
SUBMITTED BY

AVINASH GIRI
LNCTE

INTRODUCTION

The **Monty Hall problem** is a brain teaser, in the form of a probability puzzle, based nominally on the American television game show Let's Make a Deal and named after its original host, Monty Hall.

When the player first makes their choice, there is a $\frac{2}{3}$ chance that the car is behind one of the doors not chosen. This probability does not change after the host reveals a goat behind one of the unchosen doors. When the host provides information about the two unchosen doors (revealing that one of them does not have the car behind it), the $\frac{2}{3}$ chance of the car being behind one of the unchosen doors rests on the unchosen and unrevealed door, as opposed to the $\frac{1}{3}$ chance of the car being behind the door the contestant chose initially.



LOGIC BEHIND THE GAME

A player who stays with the initial choice wins in only one out of three of these equally likely possibilities, while a player who switches wins in two out of three.

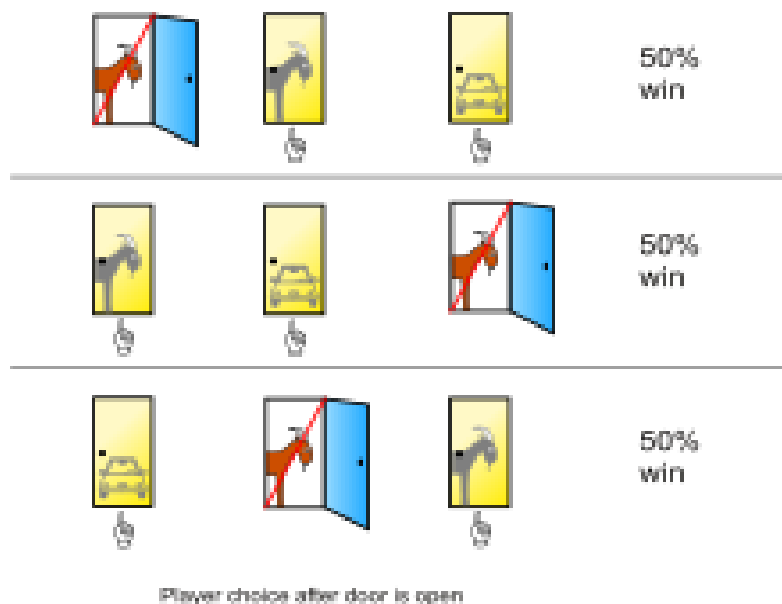
An intuitive explanation is that, if the contestant initially picks a goat (2 of 3 doors), the contestant *will* win the car by switching because the other goat can no longer be picked – the host had to reveal its location – whereas if the contestant initially picks the car (1 of 3 doors), the contestant *will not* win the car by switching. Using the switching strategy, winning or losing thus only depends on whether the contestant has initially chosen a goat (2/3 probability) or the car (1/3 probability). The fact that the host subsequently reveals a goat in one of the unchosen doors changes nothing about the initial probability.

A different selection process, where the player chooses at random *after* any door has been opened, yields a different probability.

Most people conclude that switching does not matter, because there would be a 50% chance of finding the car behind either of the two unopened doors. This would be true if the host selected a door to open at random, but this is not the case. The host-opened door depends on the player's initial choice, so the assumption of independence does not hold. Before the host opens a door, there is a 1/3 probability that the car is behind each door. If the car is behind door 1, the host can open either door 2 or door 3, so the probability that the car is behind door 1 and the host opens door 3 is

$\frac{1}{3} \times \frac{1}{2} = \frac{1}{6}$. If the car is behind door 2 – with the player having picked door 1 – the host *must* open door 3, such the probability that the car is behind door 2 and the host opens door 3 is $\frac{1}{3} \times 1 = \frac{1}{3}$. These are the only cases where the host opens door 3, so if the player has picked door 1 and the host opens door 3, the car is twice as likely to be behind door 2 as door 1. The key is that if the car is behind door 2 the host must open door 3, but if the car is behind door 1 the host can open either door.

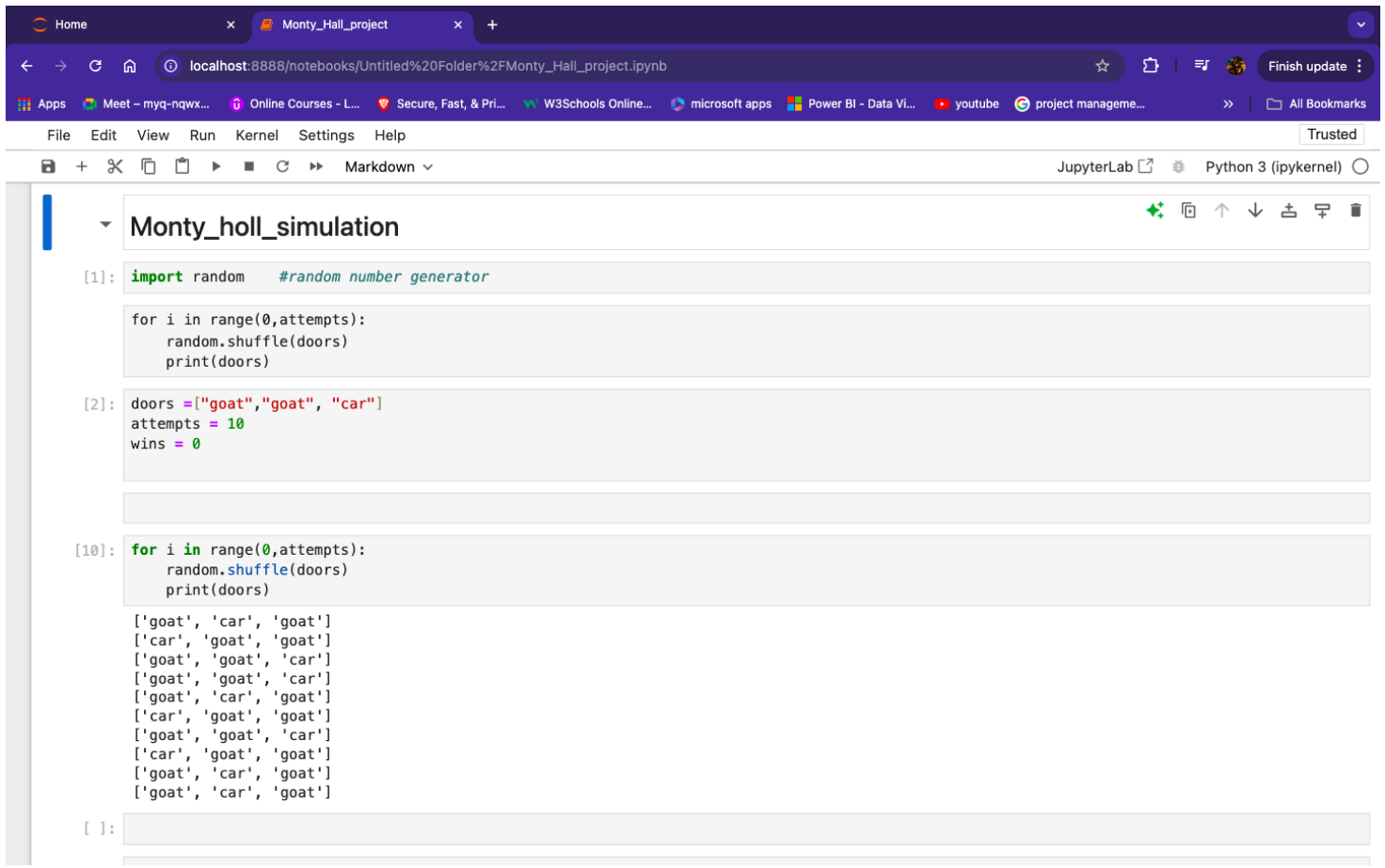
Another way to understand the solution is to consider together the two doors initially unchosen by the player. As Cecil Adams puts it, "Monty is saying in effect: you can keep your one door or you can have the other two doors". The 2/3 chance of finding the car has not been changed by the opening of one of these doors because Monty, knowing the location of the car, is certain to reveal a goat. The player's choice after the host opens a door is no different than if the host offered the player the option to switch from the original chosen door to the set of both remaining doors. The switch in this case clearly gives the player a 2/3 probability of choosing the car.



The code shown below shows how the percentage Of win increases by switching

Step 1 → import random for random guess, as we have 3 have inputs (“goat”, ”car”, “goat”).

As in output we can clearly see the guessing generated by system.



```
[1]: import random #random number generator

for i in range(0, attempts):
    random.shuffle(doors)
    print(doors)

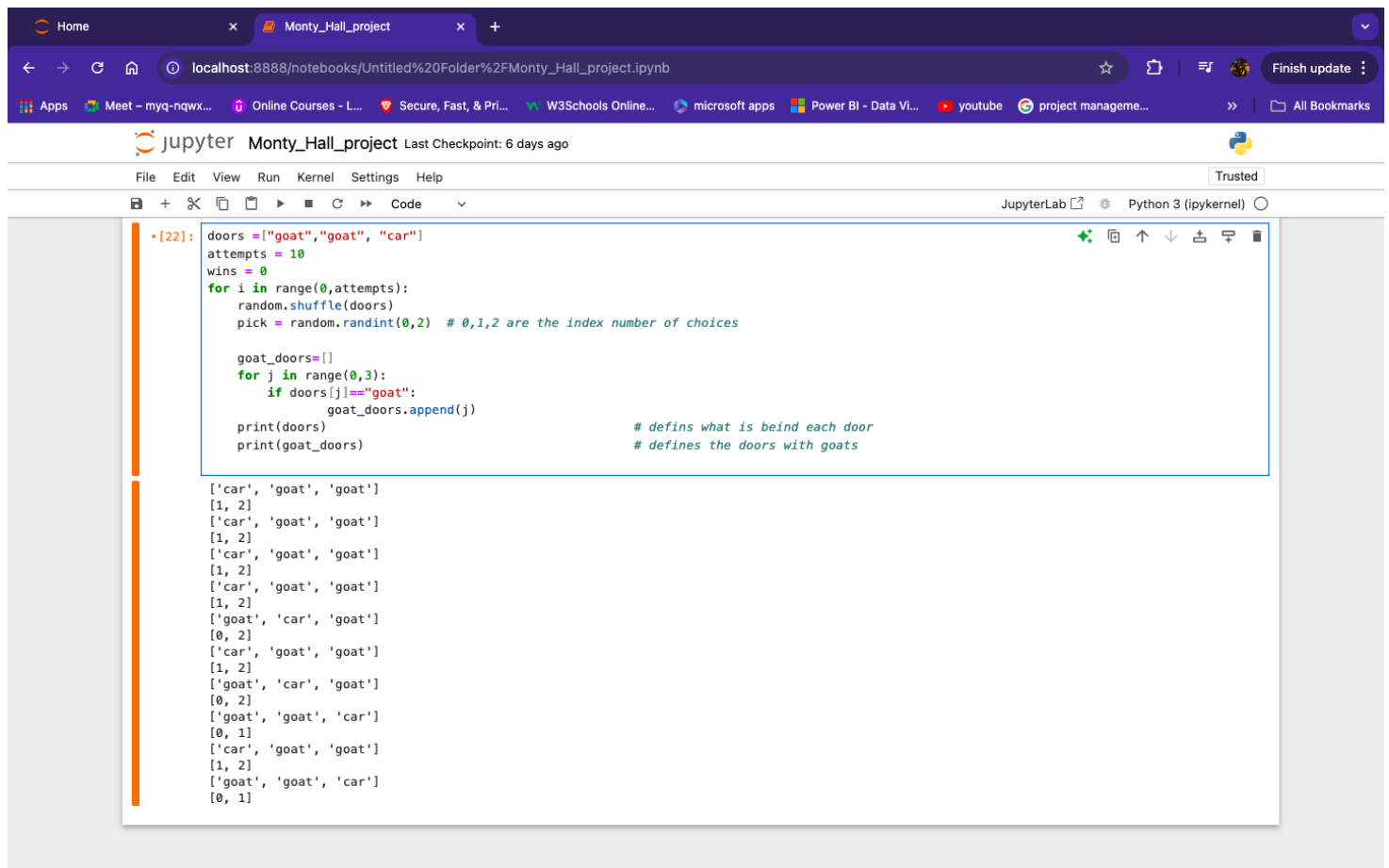
[2]: doors = ["goat", "goat", "car"]
attempts = 10
wins = 0

[10]: for i in range(0, attempts):
        random.shuffle(doors)
        print(doors)

['goat', 'car', 'goat']
['car', 'goat', 'goat']
['goat', 'goat', 'car']
['goat', 'goat', 'car']
['goat', 'car', 'goat']
['car', 'goat', 'goat']
['goat', 'goat', 'car']
['car', 'goat', 'goat']
['goat', 'car', 'goat']
['goat', 'car', 'goat']
```

Step 2 → Obtaining the position of “goat” as shown below.

So, that the host must remove the one goat from the choices left after picking one door by user.

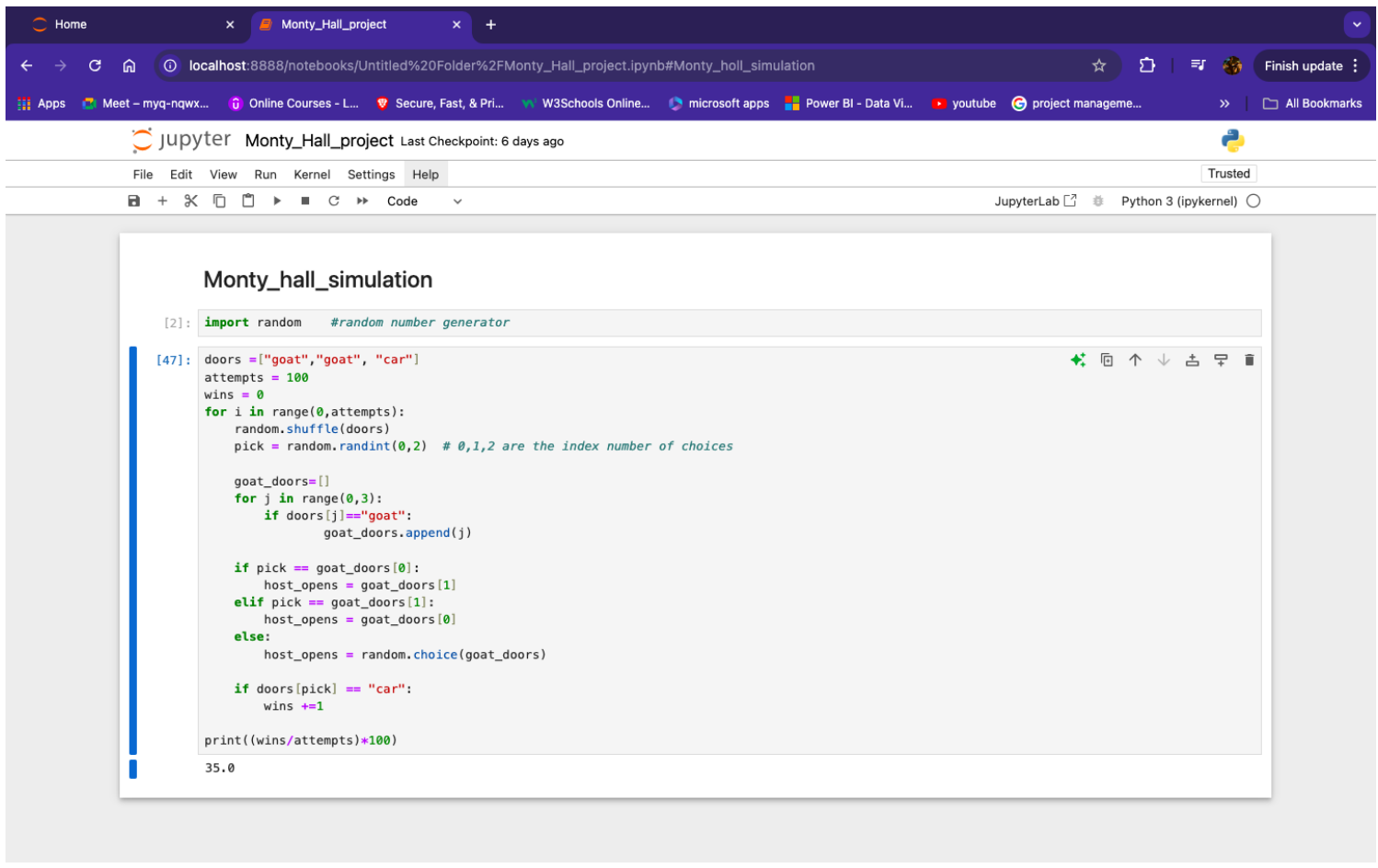


```
• [22]: doors = ["goat", "goat", "car"]
        attempts = 10
        wins = 0
        for i in range(0, attempts):
            random.shuffle(doors)
            pick = random.randint(0, 2) # 0, 1, 2 are the index number of choices

            goat_doors = []
            for j in range(0, 3):
                if doors[j] != "goat":
                    goat_doors.append(j)
            print(doors) # defines what is behind each door
            print(goat_doors) # defines the doors with goats

['car', 'goat', 'goat']
[1, 2]
['car', 'goat', 'goat']
[1, 2]
['car', 'goat', 'goat']
[1, 2]
['car', 'goat', 'goat']
[1, 2]
['goat', 'car', 'goat']
[0, 2]
['car', 'goat', 'goat']
[1, 2]
['goat', 'car', 'goat']
[0, 2]
['goat', 'goat', 'car']
[0, 1]
['car', 'goat', 'goat']
[1, 2]
['goat', 'goat', 'car']
[0, 1]
```

Step 3 → Now the output as shown below, of the code represents that if the user don't switch the probability of the card "car" is about to be 33% or near 33%.
It's because if one card is chosen by the user now the next step is done by the host that he/she remove on more card which has "goat" behind. And now user asked if he/she wants to switch or not



The screenshot shows a JupyterLab environment with a browser window at the top displaying the URL `localhost:8888/notebooks/Untitled%20Folder%2FMonty_Hall_project.ipynb#Monty_hall_simulation`. The JupyterLab interface includes a menu bar (File, Edit, View, Run, Kernel, Settings, Help) and a toolbar. The main area contains a code cell titled "Monty_hall_simulation" with the following Python code:

```
[2]: import random #random number generator

[47]: doors = ["goat", "goat", "car"]
attempts = 100
wins = 0
for i in range(0, attempts):
    random.shuffle(doors)
    pick = random.randint(0, 2) # 0, 1, 2 are the index number of choices

    goat_doors = []
    for j in range(0, 3):
        if doors[j] == "goat":
            goat_doors.append(j)

    if pick == goat_doors[0]:
        host_opens = goat_doors[1]
    elif pick == goat_doors[1]:
        host_opens = goat_doors[0]
    else:
        host_opens = random.choice(goat_doors)

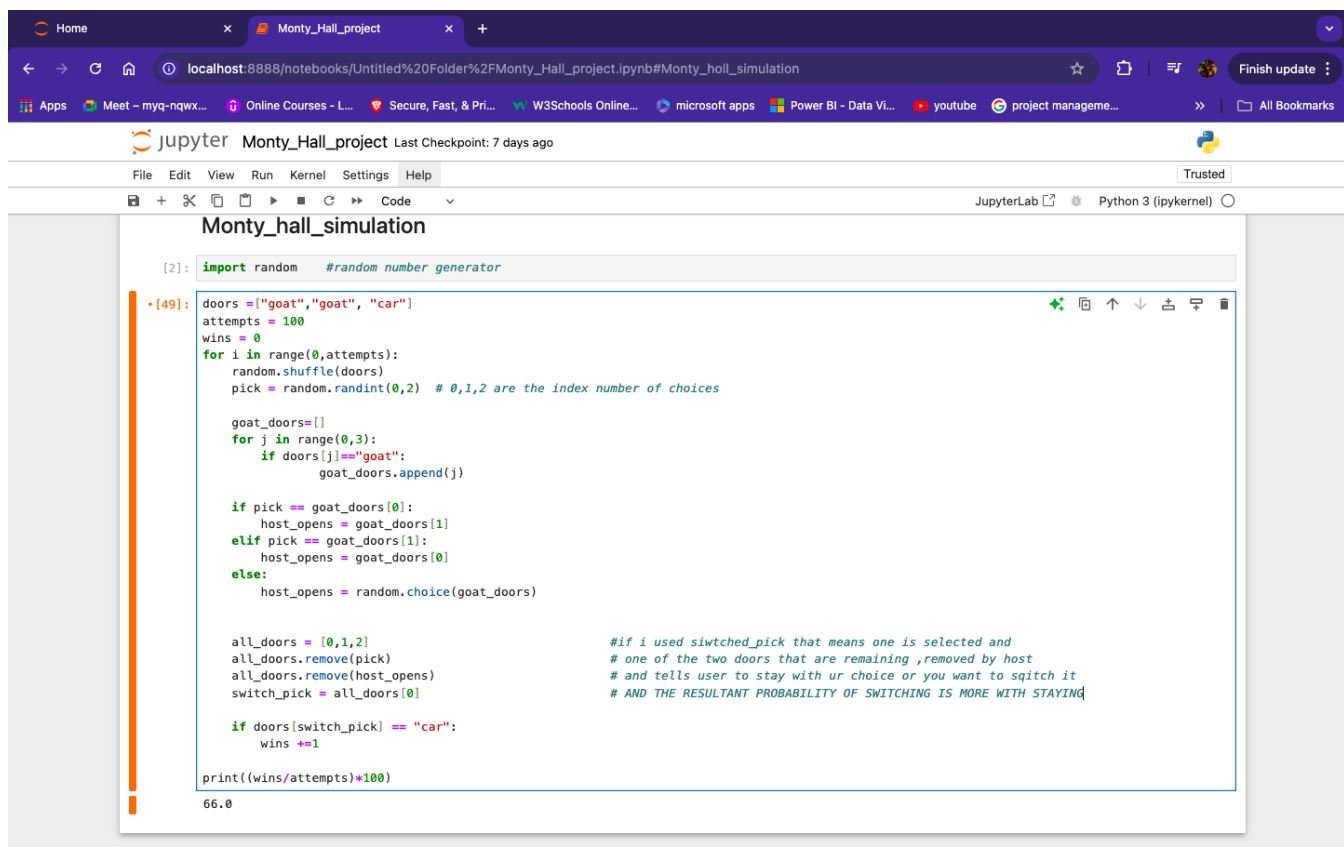
    if doors[pick] == "car":
        wins += 1

print((wins/attempts)*100)
```

The output of the code is `35.0`.

Step 4 → Now if user choose the switch option given by the host and switch his/her option and jump on another card , the possibility of card "car" behind the card the chosen by the user after switching if now getting bigger as the attempts increases.

As you can see the output of below code clearly represents that switching increases the possibility of coming "car" as output increases.



```
[2]: import random #random number generator

+ [49]: doors = ["goat","goat", "car"]
      attempts = 100
      wins = 0
      for i in range(0,attempts):
          random.shuffle(doors)
          pick = random.randint(0,2) # 0,1,2 are the index number of choices

          goat_doors=[]
          for j in range(0,3):
              if doors[j]!="goat":
                  goat_doors.append(j)

          if pick == goat_doors[0]:
              host_opens = goat_doors[1]
          elif pick == goat_doors[1]:
              host_opens = goat_doors[0]
          else:
              host_opens = random.choice(goat_doors)

          all_doors = [0,1,2]
          all_doors.remove(pick)
          all_doors.remove(host_opens)
          switch_pick = all_doors[0]

          if doors[switch_pick] == "car":
              wins +=1

      print((wins/attempts)*100)

      66.0
```

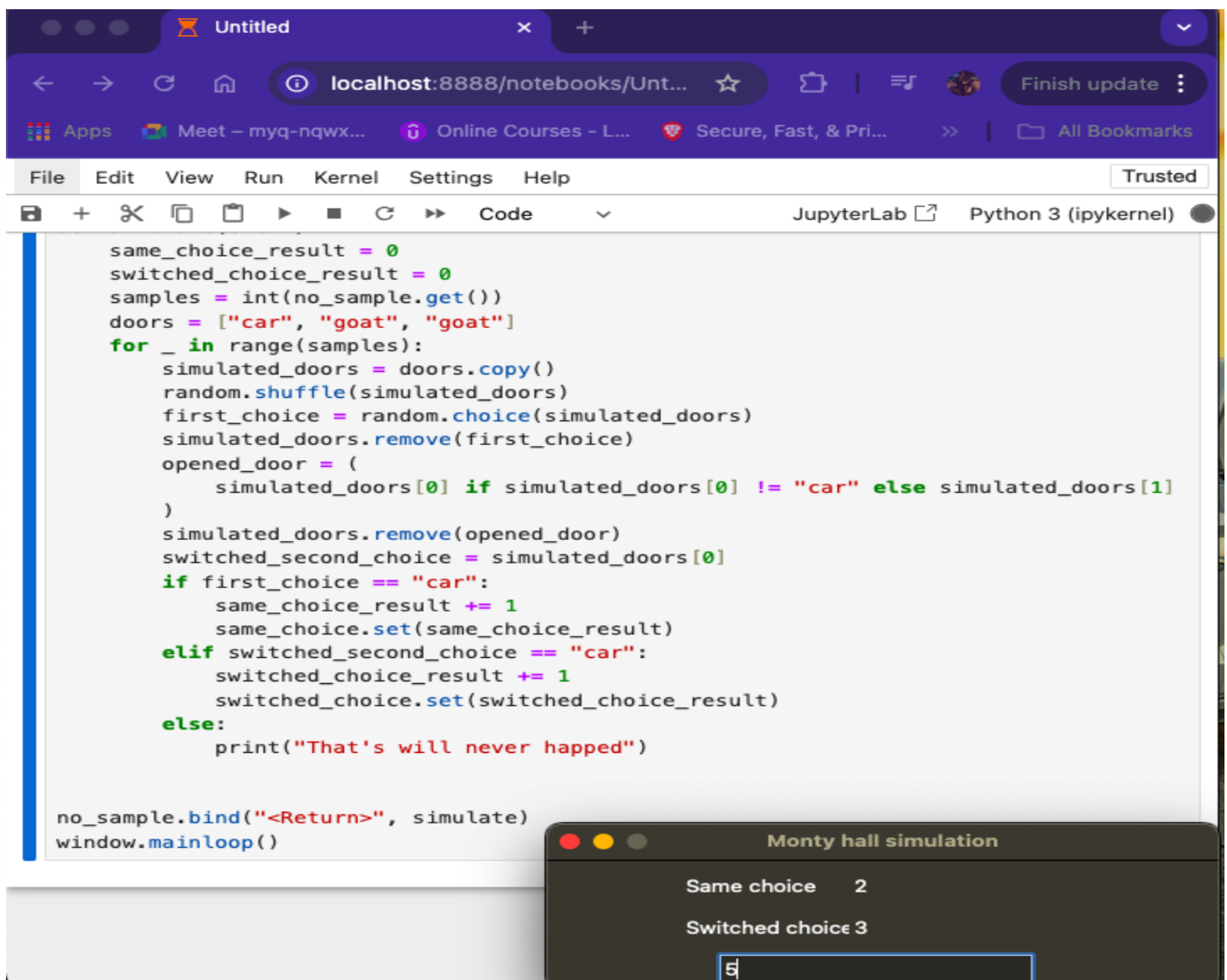
Code of the realgame that denotes the possibility of outcome after switching or stayed with same card

Now we can see that after the completion of game the output shown below in “Black window” denotes the two options:

1. Same choice
2. Switched choice

Same choice denotes that to be stayed with the choice or not whereas switched choice denotes that if he/she wants to change or not.

And the input window shows attempts it means the number of input user entered and the result decide the if you stayed with same choice “ how much time you win” or if u change or switch the card “how much time you wins”.



The screenshot displays a JupyterLab environment with a code editor and a terminal window. The code editor contains a Python script for a Monty Hall simulation. The script initializes variables for the number of samples, doors, and results. It then enters a loop where it simulates the game for each sample. In each iteration, it randomly selects a door, removes the car from the other two doors, and then simulates the player's choice. If the player stays with their first choice and it was the car, the 'same_choice_result' is incremented by 1. If the player switches to the second door and it was the car, the 'switched_choice_result' is incremented by 1. Otherwise, it prints a message indicating that the chosen door was not the car. The script binds the input to the 'simulate' function and starts the main loop.

```
same_choice_result = 0
switched_choice_result = 0
samples = int(no_sample.get())
doors = ["car", "goat", "goat"]
for _ in range(samples):
    simulated_doors = doors.copy()
    random.shuffle(simulated_doors)
    first_choice = random.choice(simulated_doors)
    simulated_doors.remove(first_choice)
    opened_door = (
        simulated_doors[0] if simulated_doors[0] != "car" else simulated_doors[1]
    )
    simulated_doors.remove(opened_door)
    switched_second_choice = simulated_doors[0]
    if first_choice == "car":
        same_choice_result += 1
        same_choice.set(same_choice_result)
    elif switched_second_choice == "car":
        switched_choice_result += 1
        switched_choice.set(switched_choice_result)
    else:
        print("That's will never happed")

no_sample.bind("<Return>", simulate)
window.mainloop()
```

The terminal window, titled "Monty hall simulation", shows the output of the simulation. It displays "Same choice 2" and "Switched choice 3", indicating the number of wins for each strategy. Below the output, there is an input field with the number "5" entered, representing the number of samples.

This is an example of the result/output with changing of attempts.

The image shows a JupyterLab interface with a code editor and a terminal window. The code editor contains a Python script for a Monty Hall simulation. The script defines variables for the number of samples, the doors, and the results of the simulation. It uses a loop to simulate the game multiple times, updating the results based on whether the user's choice was correct or incorrect. The terminal window displays the output of the simulation, showing the number of successful outcomes for both 'Same choice' and 'Switched choice' strategies.

```
same_choice_result = 0
switched_choice_result = 0
samples = int(no_sample.get())
doors = ["car", "goat", "goat"]
for _ in range(samples):
    simulated_doors = doors.copy()
    random.shuffle(simulated_doors)
    first_choice = random.choice(simulated_doors)
    simulated_doors.remove(first_choice)
    opened_door = (
        simulated_doors[0] if simulated_doors[0] != "car" else simulated_doors[1]
    )
    simulated_doors.remove(opened_door)
    switched_second_choice = simulated_doors[0]
    if first_choice == "car":
        same_choice_result += 1
        same_choice.set(same_choice_result)
    elif switched_second_choice == "car":
        switched_choice_result += 1
        switched_choice.set(switched_choice_result)
    else:
        print("That's will never happed")

no_sample.bind("<Return>", simulate)
window.mainloop()
```

Monty hall simulation

Same choice	2
Switched choice	8

10