Monty Hall

You're on a game show. The host shows you 3 doors. Behind one door, he tells you, is a fabulous prize. Behind the other two doors there is nothing. You have to pick the right door to win the prize.

You pick door 1. The host throws open door 2 and reveals.. nothing! He then makes you this offer: do you want to stay with door 1 or do you want to switch to door 3?

What should you do?

This is called the Monty Hall problem (http://en.wikipedia.org/wiki/Monty_Hall_problem).

There are 3 tactics you can use:

1. Always stay with your original choice
2. Toss a coin and use the result to pick from the two closed doors
3. Always switch to the other, unopened door

There are two ways to solve this problem. One way is to analyse it and use the rules of probability. The other way is to *model* it: to try it over and over again with random choices and see which of the three options turns out best over a large number of tries. You can model it using a pack of playing cards, but we're going to do it with a small Python program.

First, we'll use `def` to create a routine that looks like this:

```
play(tactic)
```

Where tactic is 1, 2 or 3 as listed above. This routine will return `True` if we pick the door with the prize, `False` otherwise. We'll put this routine inside a loop, run it 1000 times and count the number of times that we win. We'll do it again for each of the three tactics to see which is best.

There's just one more thing to say about the definition of the problem: the host *knows where the prize is*. After you make your first choice, the host open one of the two doors that you *haven't* chosen, but will *never open the door that conceals the prize*.

Let's get started. As usual, start by opening the python shell (IDLE) and use `file->new window` and then, in that new window, `file->save as`. Navigate to the code club drive and set the name of your program. Mine is called neal_monty. Use your own name, so that everyone's program has a different name.

Type in this piece of code. You don't need to type in the comments (the lines that start with "#"). Be careful to get the indentation (spaces at the start of lines) correct. I added a second parameter `messages` that we can use to report what's going on when we run it.

```
# Modelling solution to the Monty Hall problem in Python

from random import choice

# tactic: what to do when offered
#         1 -> keep "pick"
#         2 -> pick at random from closed doors
#         3 -> swap to other closed door
# messages: True if you want messages printed to show what's going on
def play(tactic, messages):
    doors = [1,2,3]

    # make our first choice of door
    pick = choice(doors)

    # pick which door the prize will be behind
    prize = choice(doors)

    if messages:
        print("Run for pick = " + str(pick) + ", tactic = " + str(tactic));

    # The host will open a door. The host KNOWS WHERE THE PRIZE IS!!
    can_open = [1,2,3]

    # The host will NOT open the door with the prize behind it.
    can_open.remove(prize)

    # The host will NOT open the door you have chosen
    can_open.remove(pick)

    if messages:
        print("Host is going to open one of these:" + str(can_open))

    opens = choice(can_open)

    if messages:
        print("Host has opened door " + str(opens))

    # These are the doors that are still closed
    closed = [1,2,3]
    closed.remove(opens)

    new_pick = 0

    if tactic == 1:
        new_pick = pick
    elif tactic == 2:
        new_pick = choice(closed)
    else:
        # tactic must be 3
        closed.remove(pick)
        # can only be one door left - the one we want
        new_pick = closed[0]

    # how did we do?
    if new_pick == prize:
        return True
    return False
```

As you type the program in, see if you can follow how it works. There are only two things in the program that we haven't used before.

The program starts of f by creating a *list* named `doors`. We're going to pick one of these doors at random as our choice, and we're going to choose another, also at random, to conceal the prize. We use `choice()` for this – just as we did for the times table trainer.

Next, we have to choose which door the host is going to open. We want to create a list `can_open` to holds all the doors that the host is allowed to open. That list starts out as [1,2,3] (all of the doors). The host is not allowed to open the door that we picked. Here's something new:

```
can_open.remove(3)
```

this removes the value 3 from the list, wherever in the list it is. Of course, we can't simply use "3", we need to use the door number that we picked:

```
can_open.remove(pick)
```

we also want to remove the door that has the prize behind it:

```
can_open.remove(prize)
```

Now click `run->run module` (or press F5). When the program runs it should not print anything or report any messages. All it has done is defined `play()`. If you get an error when you try to run it, compare the code above carefully with what you typed. Correct any mistakes and try again.

After you have run the program, go into the Python shell and try running play() – don't type the ">>>" that is Python prompting you to type something in:

```
>>> play(1, True)
>>> play(1, True)
>>> play(1, True)
```

It might work, it might not. By the time you have run it 2 or 3 times you are sure to get an error like this:

```
ValueError: list.remove(x): x not in list
```

The problem is that `remove` only works if the thing you're trying to remove is in the list. If `pick` and `prize` are the same, the first remove will work but the second remove will fail: will generate an error.

Change the line `can_open.remove(prize)` to look like this:

```
    if (prize != pick):
        can_open.remove(pick)
```

Save the modified program and run it again. As before, go to the Python shell and try running `play()` a few times:

```
>>> play(1, True)
>>> play(1, True)
>>> play(1, True)
```

This time, you should not see any errors. Take another look at the program and see if you can spot the other new thing; it's that strange word `elif`.

Previously we've made a 2-way choice, using `if` and `else`, like this:

```
if guess == answer:
     print("Correct")
else:
     print("Wrong")
```

`elif` is short for `else if`, and it allows us to make 3-way choices, like this (remember Goldilocks and the three bears)

```
if temperature > perfect:
     print("Too hot")
elif temperature < perfect:
     print("Too cold")
else
     print("Just right")
```

You can have as many `elif` as you like, one after another. Notice that the `if` and the `elif` need a condition (something that will work out to be either `True` or `False`) and that the condition is followed by a colon (`:`). However, the `else` doesn't need either of those things – it tells the program what to do if *none of the other conditions is true*.

Now you should understand how the whole of play() works. If you are unclear about any of this, ask for help.

Finally, all that we need to do is to add a bit more code at the end to run `play()` repeatedly for each different tactic. Type in this code *below* the code for `play()`:

```
for tactic in range(1,4):
    success = 0
    for i in range(0, 1000):
        if play(tactic, False):
            success = success + 1
    print("For tactic " + str(tactic) + " won the prize " + str(success) + "
times")
```

As before, click `run->run module` (or press F5). What is the best tactic?

If you are unclear about any of this, ask for help. Otherwise, read on for some extras...

Extras

1. It seemed sensible that we used a random number to set values for `pick` and `prize`. However, consider this: if we are going to pick one of them at random, we can leave the other one fixed. That might sound crazy, but think about it. Now, save your current program with a new name (I called mine neal_monty2) and modify it so that prize is always behind door 1. Does the program give the same result as before? Alternatively, you could have kept pick the same and used a random number for the door concealing the prize...

2. There are some other problems that can be modelled in a similar way. You could try "scissor/paper/stone" or use Wikipedia to look up "prisoners dilemma".