In [1]:

```
# import dependencies
import numpy as nmp
import random
import matplotlib.pyplot as plt
import numpy as np
```

In [2]:

```
# the game round class
class Round:
    # initialize
    def __init__(self, n_doors=3, switch=True):
        self.n_doors = n_doors

        # set a zero array representing the doors
        self.doors = nmp.zeros(n_doors)
        # pick a random door to hold the prize
        self.doors[random.randint(0,n_doors-1)] = 1
        # an array to keep track of which doors are closed
        self.closed_doors = nmp.arange(n_doors)
        # whether the player should switch
        self.switch = switch
        # pick a target
        self.target = self.closed_doors[random.randint(0, len(self.closed_doors)-1)]

    def play(self):
        # if the player switches
        if self.switch:
            # while there are doors remaining to be opened
            while len(self.closed_doors) > 1:
                # update the target
                options = nmp.copy(self.closed_doors)
                options = nmp.delete(options, nmp.where(self.closed_doors == self.target)[0])
                self.target = options[random.randint(0, len(options)-1)]
                # reveal a door
                self.__reveal_door()
            # return whether the player won
            return int(self.doors[self.closed_doors[0]])
        else:
            # if the player doesn't switch, just reveal whether they won
            return self.doors[self.target]

    # picks a door to reveal
    def __reveal_door(self):
        # if there are more than two doors remaining, pick a random door to open that doesn't
contain the prize
        if len(self.closed_doors) > 2:
            to_open = self.__reveal_losing_door()
        # if there are only two doors remaining, pick whatever the player didn't pick
        else:
            to_open = self.closed_doors[nmp.where(self.closed_doors != self.target)[0]]

        # update which doors are closed
        self.closed_doors = nmp.delete(self.closed_doors, nmp.where(self.closed_doors == to_open)[0
])

    # reveals a door that the player didn't pick and that doesn't contain the prize
    def __reveal_losing_door(self):
        # initialize the options to be the currently closed door
        options = nmp.copy(self.closed_doors)
        # delete the option that contains the prize
        options = options[options != nmp.where(self.doors == 1)[0]]
        # delete the option that the player picked
        options = options[options != self.target]

        # pick a random option
        to_open = options[random.randint(0, len(options)-1)]
        # return the door number
        return to_open
```

In [3]:

```
# plot the data
def plot_result(data, title):
    fig, ax = plt.subplots()
    results = ("won", "lost")
    x_pos = nmp.arange(len(results))

    ax.bar(x_pos, [sum(data), len(data) - sum(data)])
    ax.set_xticks(x_pos)
    ax.set_xticklabels(results)
    ax.set_title(title)
    plt.show()

    print("Wins: %.2f%%" % (100/len(data) * sum(data)))
```

In [4]:

```
def play_rounds(n, doors=3, switch=True):
    results = []
    for ii in range(n):
        r = Round(n_doors=doors, switch=switch)
        results.append(r.play())
    title = "%d doors, player " % doors
    if switch:
        title += "switches"
    else:
        title += "doesn't switch"
    plot_result(results, title)
```

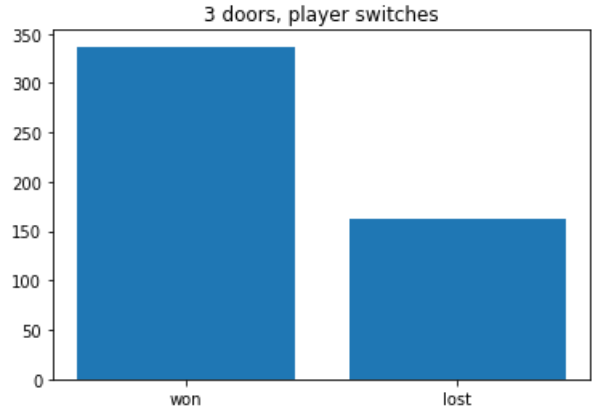
In [5]:

```
# number of rounds to play
n_games = 500
```

Results

In [6]:

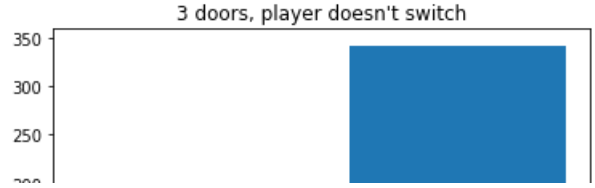
```
play_rounds(n_games, doors=3, switch=True)
```

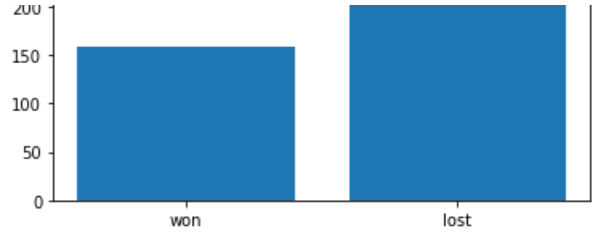



```
Wins: 67.40%
```

In [7]:

```
play_rounds(n_games, doors=3, switch=False)
```

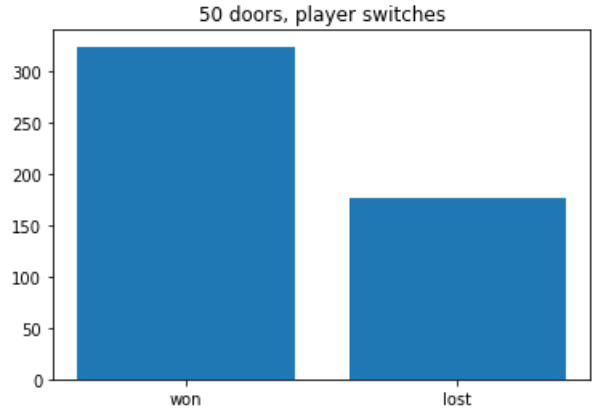





```
Wins: 31.60%
```

In [8]:

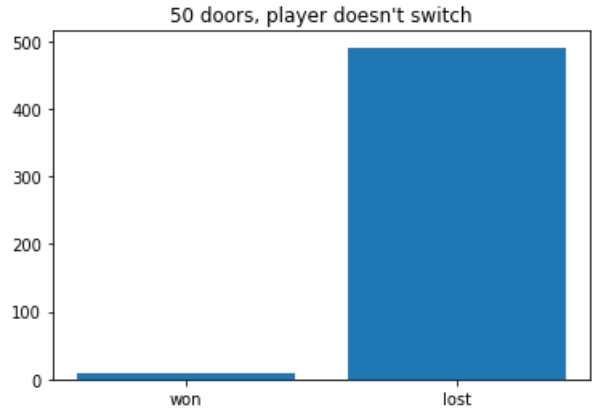
```
play_rounds(n_games, doors=50, switch=True)
```




```
Wins: 64.80%
```

In [9]:

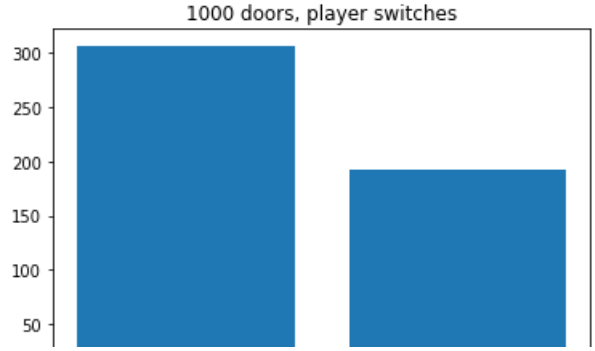
```
play_rounds(n_games, doors=50, switch=False)
```




```
Wins: 1.80%
```

In [10]:

```
play_rounds(n_games, doors=1000, switch=True)
```




```
Wins: 61.40%
```