CS 237 Lab 1

- What this lab covers:
  - Basic introduction to Python and Jupyter Notebook.
  - Basic introduction to Monte Carlo (probability) simulation.

In [1]:
# Here are some imports which will be used in the code in the rest of the lab

# Jupyter notebook specific
from IPython.display import Image
from IPython.core.display import HTML
from IPython.display import display_html
from IPython.display import display
from IPython.display import Math
from IPython.display import Latex
from IPython.display import HTML

# Imports used for the code in CS 237
# @Author: Kai Bernardini (kaidb@bu.edu)
# Due Date: TBD
import numpy as np # arrays and functions which operate on array
import matplotlib.pyplot as plt # normal plotting
from numpy.random import seed, randint, uniform
from collections import Counter

%matplotlib inline

Problem 1

Read and understand the function `dieRoll(n_trials)` below, which simulates the experiment of rolling a fair, six-sided die `n_trials` times.

- The sample space is `{1, 2, 3, 4, 5, 6}`.
- The experiment is equi-probable, i.e., the probability of any particular outcome is $\frac{1}{6}$.
- If we record the outcome for a large number of experiments, we would expect the number of outcomes to be evenly distributed. In other words, for a large number of trials, we would expect
  \[
  \frac{\text{number of times we observed } k}{\text{n Trials}} \approx \frac{1}{6}
  \]
  for $k \in \{1, 2, 3, 4, 5, 6\}$

TO DO: For this first problem, simply provide the Python code which would display a histogram of the results of the experiment for 10000 trials with appropriate labels.

In [2]:
```python
def roll_die(n_trials = 10000, seed=-1):
    """
    - Simulates rolling a fair die n_trials times, i.e., a number is selected from
    - (1,2,3,4,5,6) with equal probability n_trials times. By default, n_trials is set to 10000
    - randint(a,b) uniformly and randomly selects an integer x such that a<= x < b
    - The second argument for randint determines the shape and size of the result array. In this case, we only need
    - a one dimensional array (a list)
    """
    trials_and_results = randint(1,7,(n_trials)) # this creates a 1D array of length n_trials of
    random integers 1..6
    return trials_and_results

example_trials = roll_die()
```
Problem 2

Now we will display the same results showing the distribution of probabilities, instead of an explicit histogram.

TO DO: Complete the function stub below which takes the list returned by roll_die(...), or any other experiment returning numerical results, and produces a frequency distribution; this should have the same shape as the histogram, but the Y axis will be probabilities instead of the frequency. Again, create appropriate labels. Demonstrate your function, again, on the list example_trials produced in Problem 1.

```python
In [4]: # Solution
def show_distribution(outcomes, title='Probability Distribution'):
    num_trials = len(outcomes)
    X = range( int(min(outcomes)), int(max(outcomes))+1 )
    freqs = Counter(outcomes)
    Y = [freqs[i]/num_trials for i in X]
    plt.bar(X,Y,width=1.0,edgecolor='black')
    plt.xlabel("Outcomes")
    plt.ylabel("Probability")
    plt.title(title)
    plt.show()

show_distribution(example_trials,title='Probability Distribution for Single Die Toss')
```
Motivation for Monte Carlo simulation

For the case of a fair die, the distribution is very easily computed. In general, it is very difficult to write down a closed form solution for the distribution of real world events. This is where simulation comes into play-- instead of mathematically computing the distribution explicitly, you can use this method of repeating experiments, and recording outcomes to understand the probabilistic rules governing some real world event. When you can come up with an analytical result, this is a nice way of confirming its correctness!

Problem 3

You will now do the same thing you did in the previous problems, but with a new experiment: instead of rolling one die and recording the value, you will simulate rolling $n$ dice and recording their sum. For example, if $n = 2$ and the first die shows up as a 3, and the second die shows up as a 1, the sum (and the value we record) would be 4. TO DO: Complete the two functions stubs below and then demonstrate by providing the single line of code which would print out the probability distribution for rolling 2 dice 10,000 times.

In [5]:
# Solution
def roll_and_add_dice(n_dice, n_trials = 10000):
    ""
    - Perform the trials
    - randint(a,b) uniformly and randomly selects an integer x such that a<= x < b
    - The second argument determines the shape of the result. In this case, we only need
    - a one dimensional array (a list)
    ""
    trials_and_results = randint(1,7,(n_dice, n_trials))
    acum = np.zeros(n_trials)
    for i in range(len(trials_and_results)):
        acum += trials_and_results[i]
    return acum

show_distribution(roll_and_add_dice(2),title='Probability Distribution for Sum of Two Dice')

# Here’s another way to do it, showing the power of list comprehensions with numpy! A one-line solution!
def roll_and_add_dice(n_dice, n_trials = 10000):
    return [ sum(randint(1,7,[n_dice])) for k in range(n_trials) ]

Problem 4

TO DO: For the final problem of this lab, provide code which will display the probability distribution for the experiment of running the “flip a coin until you get a head” experiment 1000 times.
In [6]: # Solution

    # one trial of the experiment, if it reaches the upper bound, return upper bound
    def roll_until_heads(upper_bound = 100):
        for k in range(1,upper_bound):
            if randint(0,2) == 1:
                return k
        return k

def run_experiment(n_trials = 10000):
    return [roll_until_heads() for k in range(n_trials)]

show_distribution(run_experiment(),title='Probability Distribution for Toss Until Heads')