PHY 250L – Spring 2018
Python tutorial 1

Welcome back to the PHY 250L Python programming tutorial!

Things you should understand after week 1:

- Spyder
- function definitions
- lists, list manipulations
- including modules: `math`, `matplotlib`
- random number generation
- plotting bivariate data
- histograms

Problems for 2.22.2018 The following problems should be completed and uploaded to Sakai by 09:45 on 2.22.2018. Each problem should correspond to its own python program (*i.e.*, each problem will correspond to a single file). The preferred names for the files are indicated in each problem.

1. `calculate_pi.py`, 30 points
   In this problem, you will calculate/approximate a value for π using a series expression developed by Abraham Sharp in 1717:

$$S_n = \sum_{k=0}^{n} \frac{2\,(-1)^k\,3^{1/2-k}}{2k+1} \tag{1}$$

   This sum gives $S_n \to \pi$ as $n \to \infty$. There are many expressions like this, some of them beautifully weird.[1] Write a program that calculates the approximation of π that this sum gives for different n up to $n = 30$. Your program should output to screen the following quantities, separated by tabs:

$$n \qquad S_n \qquad (S_n - \pi) \qquad (S_n - \pi)/\pi \tag{2}$$

   For a fairly precise value of π, use

```
import math
pi_val = math.pi
```

2. `random_pi.py`, 30 points
   In this problem, you'll approximate π in a completely different (dumber) way: random sampling. Begin by generating two random numbers between 0 and 1, call them x and y. Think about $P = (x, y)$ as a point in the first quadrant of the 2-d plane. The area of the space that P could possibly occupy is

$$A_s = 1 \times 1 = 1 \tag{3}$$

   Now, imagine the quarter circle centered at $(0, 0)$ with radius 1 inscribed in this area (*i.e.*, the part of the unit circle between the positive real and positive imaginary axes, see the figure at right).

[1] Check out those from Ramanujan! They are the infinite-series equivalent of *Alice in Wonderland*.

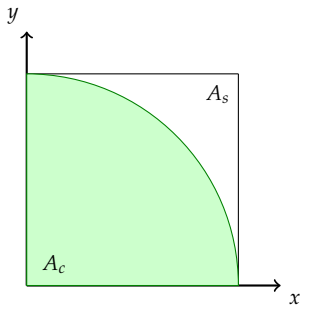

Figure 1: The areas associated with problem 3.

The area of this circle is

$$A_c = \frac{1}{4}\,\pi\,(1)^2 = \frac{\pi}{4} \tag{4}$$

SO! The probability that your randomly generated P lies inside of the circular segment is the ratio of the two areas:

$$p = A_c / A_s = \frac{\pi}{4} \tag{5}$$

Put another way, if your x and y are truly randomly distributed, the fraction of P that lie within the circular arc should be proportional to the ratio of A_c to A_s.

Write a program that generates random points $P = (x, y)$ and approximates π by taking the ratio of the number of points that fall within the circular segment to the total number of points generated. You will need to develop some way of determining whether P is inside the circular segment *without* relying on a value of π; there are several ways to do this. Your program should output the approximate value of π after every 100 points generated, and exit once your approximation of π matches the actual value to five decimal places.

3. `blackjack.py`, 20 points
Random number generation is important for modeling all types of physical systems... but more importantly, it's important for learning how to get good at gambling. In this problem, you'll build code to simulate a basic 52-card shuffle for the game blackjack. In blackjack, suits of cards do not matter, but each car carries a point value:

- A number card's point value is equal to its number (*e.g.*, 3♡ → 3, 3♠ → 3, 9♢ → 9). Number cards range from 2 to 10.
- Each face card (jack, queen, king) carries a point value of 10.
- The ace is a special card in that it can have a value of 1 or 11 – the player gets to decide when the card is dealt.

We will represent each suit of 13 cards as a list of only their point values:

```
suit = [2, 3, 4, 5, 6, 7, 8, 9, 10, 10, 10, 10, 11]
```

Here, we'll assume that the ace is worth 11 points. (You'll see why soon.) With this, you can create a full deck of 52 cards by adding suits:

```
deck = []
for i in range(4):
    deck += suit
```

Code this up and take a look at `deck`. Make sure that it has the appropriate number of elements. "Shuffling" can be modeled by randomizing the order of the elements in `deck`. The random module has a command for this:

```
import random
random.shuffle(deck)
```

Print out `deck` and check to see that the order of its elements have been randomized.

Use these elements to determine the probability that a the first two cards out of a freshly shuffled deck of 52 will add to 21 (and thus win the hand). To do so, you'll begin with an array of 52 "cards", shuffle, check the sum of the first two cards, then repeat for a total of 10^3 decks.

Additionally, your program should make a histogram of the points total of the first two cards in the deck for all 10^3 iterations.