# COMS BC 3997 - F22: Problem Set 0

Grace Hopper

**Introduction:** Hello and welcome to COMS BC 3997 - F22! I know what you are thinking, "ugh, why do I have to do PS 0? We haven't even learned anything yet." That might be true, but we wanted to give you all an opportunity to **practice** completing and turning in assignments for this class, as after **PS 0**, we will not be so lenient and forgiving in our handling of incorrectly submitted problem sets.

This first assignment **should not** feel too arduous or time consuming – in fact, it may even seem trivial. If it is, good! It is meant to get you back in the groove of things and ready to tackle this semester, not provide too much mental stress. If, for whatever reason, you are struggling with this assignment, **PLEASE** reach out to one of us, your teaching staff – or even fellow classmates or friends. We really want to make sure everyone comes into the class confident that they can succeed.

The following exercises will mostly test some basic math and probability concepts as well as getting you more comfortable with LaTeX. Without further ado, let's jump right in!

**Problem 1 – 2 Points:** Imagine you are playing Settlers of Catan with your friends – because what else would you be doing on a Friday night – and you want to decide what tiles to place your newest settlement next to (for those of you who don't know what Catan is, don't worry about it). Each tile has a number on it $i \in [2, 12]$ (a number from 2 to 12, inclusive), and you want to figure out the probabilities of rolling each of these numbers using two fair, 6-sided dice. Fill in the second column of Table 1.

Now suppose that one of these dice has been tampered with and can magically only land on "1" or "2" – each with probability $\frac{1}{2}$. Fill in the third column of Table 1 with the new probabilities.

**Solution 1:**

| x | P(x is rolled) | P(x is rolled with unfair die) |
|---|---|---|
| 2 | $\frac{1}{36}$ | $\frac{1}{12}$ |
| 3 | | |
| 4 | | |
| 5 | | |
| 6 | | |
| 7 | | |
| 8 | | |
| 9 | | 0 |
| 10 | | 0 |
| 11 | | 0 |
| 12 | | 0 |

Table 1: Dice Roll Probabilities

**Problem 2 – 1 Point:** A conditional probability is just a normal probability, conditioned on some other event. For example, if $P(A)$ is the probability of event $A$ occurring and $P(B)$ is the probability of event $B$ occurring, then $P(A|B)$ is the probability of event $A$ occurring given that event $B$ occurs. One of the most famous and useful rules of conditional probability is called Bayes' Rule:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Let's suppose that we wish to predict the probability that it is raining outside without just looking it up on our phone or looking out the window. We know that on any given day, the probability that it rains is 0.2, the probability our roommate takes their umbrella with them is 0.3 (regardless of rain or shine), and the probability our roommate takes his umbrella given it's raining is 0.95. What is the probability it is raining given that our roommate just took his umbrella? What if he hadn't taken his umbrella?

**Solution 2:**

**Problem 3 – 1 Point:** Although taking derivatives won't be strictly necessary for this class, we will talk briefly about some basic optimization strategies. For the sake of this example, let's suppose that we wish to minimize some loss function $\mathcal{L}$ with respect to a variable $x$.

$$\mathcal{L}(x) = 3x^2 - 2x + 1$$

Given this loss function, what value of $x$ minimizes the loss? In other words, for what value of $x$ is this function minimal?

**Solution 3:**

**Problem 4 – 1 Point:** For this last problem, we will deal with the basics of asymptotic notation. For those of you who have never seen this before, do not fret! We will only really deal loosely with the notion of Big-O notation in this class. For a really simple beginner's guide, see: https://rob-bell.net/2009/06/a-beginners-guide-to-big-o-notation/. Briefly, Big-O notation is a way for us to talk about assymptotic run-times of algorithms. For those of you who have explored sorting algorithms in past CS classes may recall that insertion sort takes $O(n^2)$ time – where $n$ is the length of the list to sort – while merge sort takes $O(n \ln n)$ time.

What is the runtime of the following `foo` algorithm in Big-O notation? What does the following algorithm do?

```
def sum_list(lst):
    """
    Function that returns the sum of elements in lst
    :param lst: input list
    :type lst: [int]
    :return: sum of the list
    :rtype : int
    """
    rtn = 0
    for elt in lst:
        rtn += elt
    return rtn


def foo(lst):
    """
    Foo Algorithm
    :param lst: input list of length n
    :type lst: [int]
    :return: ???
    :rtype : int
    """
    for elt in lst:
        if elt < 0:
            return -1
    return sum_list(lst)
```

**Solution 4:**