The quiz for 6.0002 is on MITx. Do not write your answers on this paper. Only fill out this front page with your name, username, and recitation.

You are allowed 1 page (2 sides total) on which you can write anything. On your computer, you are only allowed to have the MITx website open and IDLE (or whatever Python IDE you prefer). Close everything except for these. If the test proctors see that you have anything else open on the computer or if we believe you are trying to cheat, you will be asked to leave immediately and given a zero on the exam.

You have 90 minutes to complete the exam.

You must check out with an exam proctor after you have finished your exam but before you leave. Hand in this paper exam when you checkout. Any student who does not check do this will receive a 0 on the exam.
1) True/False [20 points]

A greedy solution to the 0/1 knapsack problem will never produce the optimal solution.
False

In Python, we can use `random.seed(100)` at the beginning of a program to generate the same sequence of random numbers each time we run a program.
True

Let $h$ be a hash function. If $x \neq y$, then $h(x)$ will always be unequal to $h(y)$.
False

Inferential statistics assumes that a random sample of a population has similar characteristics to the full population.
True

Knowing only the mean, we can draw a unique Gaussian (normal) distribution.
False

You roll an unfair (weighted) die. The distribution of the numbers rolled is a uniform distribution.
False

Consider deriving the probability of a coin flip coming up heads by running $m$ trials of 100 flips each. If the coin is fair, the mean probability of the $m$ trials will go to 0.5 as $m$ goes to infinity.
True

In a Gaussian (normal) distribution, approximately 95% of the values fall within one standard deviation of the mean.
False

Every optimization problem can be fully defined by providing an objective function to be maximized or minimized.
False

A danger of overfitting is that the model captures experimental error in the data.
True
2) [10 points]

Consider the following set of subjects:

Computer Science: 9am - 10am  
Biology: 10am - 1pm  
Physics: 11am - noon  
Chemistry: noon - 2pm  
French: 2pm - 3pm  
English: 1pm - 3pm  
Gym: 3pm - 4pm

Assume that each subject is an object of type Subject, and that the Subject class has `getStartTime` and `getEndTime` methods with the obvious semantics. Assume further that `subjectList` is a Python list that contains the subjects in the table, but not necessarily in the same order in which they occur in the table.

Consider defining (not solving) the optimization problem of finding a list of courses, call it \( s \), such that \( s \) includes as many courses as possible and no course overlaps in time.

Write the objective function for this problem. Feel free to use non-Python notation e.g., a summation.

**Maximize the number of courses to take**

Write the constraints, if any, for this problem. If there are none, write "none."

**Course times do not overlap**
3)  

a) [5 points]  
You have a bucket with 4 red balls and 4 green balls. You draw 3 balls out of the bucket. Assume that once you draw a ball out of the bucket, you don't replace it. What is the probability of drawing 3 balls of the same color?  
Write your answer in terms of a fraction.  
\[
\frac{3}{7} \times \frac{2}{6} = \frac{6}{42} = \frac{1}{7}
\]

b) [20 points]  
Write a Monte Carlo simulation to solve the above problem. Feel free to write a helper function if you wish.

```python
def drawing_without_replacement_sim(numTrials):
    '''
    Runs numTrials trials of a Monte Carlo simulation of drawing 3 balls out of a bucket containing 4 red and 4 green balls. Balls are not replaced once drawn. Returns a float - the fraction of times 3 balls of the same color were drawn in the first 3 draws.
    '''
    # Your code here

def oneTrial():
    balls = ['r', 'r', 'r', 'r', 'g', 'g', 'g', 'g']
    chosenBalls = set([])
    for t in range(3):
        # For three trials, pick a ball
        ball = random.choice(balls)
        chosenBalls.add(ball)
        # Remove the chosen ball from the set of balls drawn. Add it to a set of balls we picked
        balls.remove(ball)
    # If there's only one ball in the set then we picked the same ball each time.
    return len(chosenBalls) == 1

def drawing_without_replacement_sim(numTrials):
    numTrue = 0
    for trial in range(numTrials):
        if oneTrial():
            numTrue += 1
    return float(numTrue)/float(numTrials)
```

Restrictions:
- An analytic solution to this problem will receive 0 points.
- Do not import or use functions or methods from `pylab`, `numpy`, or `matplotlib`.  

```python
def oneTrial():
    balls = ['r', 'r', 'r', 'r', 'g', 'g', 'g', 'g']
    chosenBalls = set([])
    for t in range(3):
        # For three trials, pick a ball
        ball = random.choice(balls)
        chosenBalls.add(ball)
        # Remove the chosen ball from the set of balls drawn. Add it to a set of balls we picked
        balls.remove(ball)
        # If there's only one ball in the set then we picked the same ball each time.
        return len(chosenBalls) == 1

def drawing_without_replacement_sim(numTrials):
    numTrue = 0
    for trial in range(numTrials):
        if oneTrial():
            numTrue += 1
    return float(numTrue)/float(numTrials)
```
import random, pylab

# You are given this function
def getMeanAndStd(X):
    mean = sum(X)/float(len(X))
    tot = 0.0
    for x in X:
        tot += (x - mean)**2
    std = (tot/len(X))**0.5
    return mean, std

# You are given this class
class Die(object):
    def __init__(self, valList):
        """ valList is not empty """
        self.possibleVals = valList[:]
    def roll(self):
        return random.choice(self.possibleVals)

# Implement this -- Coding Part 1 of 2
def makeHistogram(values, numBins, xLabel, yLabel, title=None):
    """
    - values, a sequence of numbers
    - numBins, a positive int
    - xLabel, yLabel, title, are strings
    - Produces a histogram of values with numBins bins and the indicated labels
      for the x and y axis
    - If title is provided by caller, puts that title on the figure and otherwise
      does not title the figure
    """
    # TODO

# Implement this -- Coding Part 2 of 2
def getProbability(die, numRolls, numTrials):
    """
    - die, a Die
    - numRolls, numTrials, are positive ints
    - Calculates the expected mean value of the longest run of a number
      over numTrials runs of numRolls rolls, and the 95% confidence interval.
      Rounds the mean and confidence interval to 3 decimal places.
    - Calls makeHistogram to produce a histogram of the longest runs for all
      the trials. There should be 10 bins in the histogram
    - Choose appropriate labels for the x and y axes.
    - Returns a tuple of the mean and the 95% confidence interval calculated
    """
    # TODO

# One test case
print getProbability(Die([1,2,3,4,5,6,6,7]), 500, 10000)
a) [15 points]

Write a function called `makeHistogram(values, numBins, xLabel, yLabel, title=None)`, with the following specification:

```python
def makeHistogram(values, numBins, xLabel, yLabel, title=None):
    
    - values, a list of numbers
    - numBins, a positive int
    - xLabel, yLabel, title, are strings
    - Produces a histogram of values with numBins bins and the indicated labels
      for the x and y axes
    - If title is provided by caller, puts that title on the figure and otherwise
      does not title the figure
```

Restrictions:
- Do not write import `pylab`
- You should only be using the `pylab.hist`, `pylab.title`, `pylab.xlabel`, `pylab.ylabel`, `pylab.show` functions from the `pylab` module.

```python
def makeHistogram(values, numBins, xLabel, yLabel, title=None):
    pylab.hist(values, numBins)
    pylab.xlabel(xLabel)
    pylab.ylabel(yLabel)
    if title != None:
        pylab.title(title)
    pylab.show()
```

b) [30 points]

Write a function called `getProbability(die, numRolls, numTrials)`, with the following specification:

```python
def getProbability(die, numRolls, numTrials):
    
    - die, a Die
    - numRolls, numTrials, are positive ints
    - Calculates the expected mean value of the longest run of a number
      over numTrials runs of numRolls rolls, and the 95% confidence interval.
      Rounds the mean and confidence interval to 3 decimal places.
    - Calls makeHistogram to produce a histogram of the longest runs for all
      the trials. There should be 10 bins in the histogram
    - Choose appropriate labels for the x and y axes.
    - Returns a tuple of the mean and the 95% confidence interval calculated
```
A run of numbers counts the number of times the same dice value shows up in consecutive rolls. For example:

- a dice roll of 1 4 3 has a longest run of 1
- a dice roll of 1 3 3 2 has a longest run of 2
- a dice roll of 5 4 4 5 5 2 5 has a longest run of 3

When this function is called with the test case given in the file, it will return the tuple (5.312, 2.337). The first number in the tuple is the mean and the second number in the tuple defines the 95% confidence interval. Your simulation may give slightly different values.

Restrictions:

- Do not import or use functions or methods from `pylab`, `numpy`, or `matplotlib`.
- If you do not see the return value being printed when testing your function, close the histogram window.

def getProbability(die, numRolls, numTrials):
    longestRuns = []
    for t in range(numTrials):
        longestRun = 1
        curRun = 1
        lastRoll = die.roll()
        for g in range(numRolls - 1):
            nextRoll = die.roll()
            if nextRoll == lastRoll:
                curRun += 1
            else:
                if curRun > longestRun:
                    longestRun = curRun
                curRun = 1
            lastRoll = nextRoll
        if curRun > longestRun:
            longestRun = curRun
        longestRuns.append(longestRun)
    (mean, std) = getMeanAndStd(longestRuns)
    makeHistogram(longestRuns, 10, 'Length of Longest Run', 'Number of Occurrences')
    return (round(mean, 3), round(2*std, 3))