DATA 601: Fall 2019

HW1

Due: Wed. Sep. 18 at 23:55

Learning Objectives

- Gain familiarity with Markdown.
- Explore collection classes in Python.
- Use intermediate level data structures and programming concepts in the context of data related problems.

This is an individual homework assignment.

Please complete this homework assignment within the Jupypter notebook environment.

Submission

Your submission will be graded using a combination of auto-grading and manual grading. In order to ensure that everything goes smoothly, please follow these instructions:

- Please provide your solutions where asked; please do not alter any other parts of this notebook.
- Submit via the HW1 dropbox on D2L. Please ensure that your submitted file is named 'HW1.ipynb'.
- Do not submit any other files.

Question 1: Markdown

(1 point)

Please go through the following Markdown tutorial:

https://www.markdowntutorial.com/ (https://www.markdowntutorial.com/)

In the cell below, use Markdown to typeset your name at heading level 3 and your student ID at heading level 4.

Usman Alim

ID: 000000000

Question 2: List Processing

(6 points)

The questions below ask you to process data stored in lists. To focus on problem solving and to make your code more readable, *you may use built-in functions*. Please try and use comprehensions whenever possible.

1. Given two lists, write a Python function called `myprod(l1, l2)` that produces a list containing all possible pairings — as tuples — of the elements in the two lists, i.e. the Cartesian product of the two lists. The first element of each tuple should come from `l1` and the second element should come from `l2`. You can assume that the lists will not have any duplicates.

   For example, the Cartesian product of the two lists `['A', 'K', 'Q', 'J', '10', '9', '8', '7', '6', '5', '4', '3', '2']` and `['♠', '♥', '♦', '♣']` should yield a standard deck of playing cards.

In [23]:

```python
def myprod(l1, l2):
    ### BEGIN SOLUTION
    return [(e1,e2) for e1 in l1 for e2 in l2]
    ### END SOLUTION
```

In [22]:

```python
'''Check that myprod produces the correct output.'''

l1 = ['A', 'K', 'Q', 'J', '10', '9', '8', '7', '6', '5', '4', '3', '2']
l2 = ['♠', '♥', '♦', '♣']
assert set(myprod(l1,l2)) == {
    ('A', '♠'), ('A', '♥'), ('A', '♦'), ('A', '♣'), \
    ('K', '♠'), ('K', '♥'), ('K', '♦'), ('K', '♣'), \
    ('Q', '♠'), ('Q', '♥'), ('Q', '♦'), ('Q', '♣'), \
    ('J', '♠'), ('J', '♥'), ('J', '♦'), ('J', '♣'), \
    ('10', '♠'), ('10', '♥'), ('10', '♦'), ('10', '♣'), \
    ('9', '♠'), ('9', '♥'), ('9', '♦'), ('9', '♣'), \
    ('8', '♠'), ('8', '♥'), ('8', '♦'), ('8', '♣'), \
    ('7', '♠'), ('7', '♥'), ('7', '♦'), ('7', '♣'), \
    ('6', '♠'), ('6', '♥'), ('6', '♦'), ('6', '♣'), \
    ('5', '♠'), ('5', '♥'), ('5', '♦'), ('5', '♣'), \
    ('4', '♠'), ('4', '♥'), ('4', '♦'), ('4', '♣'), \
    ('3', '♠'), ('3', '♥'), ('3', '♦'), ('3', '♣'), \
    ('2', '♠'), ('2', '♥'), ('2', '♦'), ('2', '♣')}
```

1. Write a Python function called `mytally( li )` that takes a list `li` and returns a dictionary whose keys are the unique entries in `li` and whose values are the counts of each of the unique entries. For example:

   `mytally( [1, 2, 3, 3, 4, 5, 6, 6, 6, 7, 8, 9, 9] )`
   should return
   `{1:1, 2:1, 3:2, 4:1, 5:1, 6:3, 7:1, 8:1, 9:2}`

In [25]:

```
def mytally( li ):
    ### BEGIN SOLUTION
    keys = set(li)
    return {k : li.count(k) for k in keys}
    ### END SOLUTION
```

In [50]:

```
'''Check that mytally returns the correct output'''
assert mytally([1, 2, 3, 3, 4, 5, 6, 6, 6, 7, 8, 9, 9]) == \
    {1: 1, 2: 1, 3: 2, 4: 1, 5: 1, 6: 3, 7: 1, 8: 1, 9: 2}
assert mytally(list('mississippi')) == \
    {'i': 4, 'm': 1, 'p': 2, 's': 4}
import random
random.seed(a=601, version=2)
assert mytally( [random.randint(0,10) for i in range(10000)] ) == \
    {0:855, 1:895, 2:946, 3:961, 4:840, 5:941, 6:875, 7:916, 8:932, 9:91
0, 10:929}
```

1. Write a function called `mysplit( li )` that takes a list `li` and splits it into sublists consisting of runs of identitical elements. The returned list should be sorted in ascending order. You may assume that `li` consists of immutable and comparable objects. For exxample:

   ```
   mysplit( [1, 2, 3, 3, 4, 5, 6, 6, 6, 7, 8, 9, 9] )
   ```
   should return
   ```
   [[1], [2], [3,3], [4], [5], [6,6,6], [7], [8], [9,9]]
   ```

In [35]:

```
def mysplit( li ):
    ### BEGIN SOLUTION
    tally = mytally( li )
    return [[item]*tally[item] for item in sorted(tally)]
    ### END SOLUTION
```

In [49]:

```
'''Check that mysplit produces the correct result'''
tlist = [1, 2, 3, 3, 4, 5, 6, 6, 6, 7, 8, 9, 9]
assert mysplit(tlist) == [[1], [2], [3,3], [4], [5], [6,6,6], [7], [8],
[9,9]]
assert mysplit(tlist[::-1]) == [[1], [2], [3,3], [4], [5], [6,6,6], [7],
[8], [9,9]]
assert mysplit(list('mississippi')) == \
       [list("iiii"), list("m"), list("pp"), list("ssss")]
```

Question 3: Plotting Functions

(6 points)

Please go through the following tutorial, focusing on the first two sections.

https://matplotlib.org/users/pyplot_tutorial.html (https://matplotlib.org/users/pyplot_tutorial.html)

Use `matplotlib.pyplot.plot` (https://matplotlib.org/users/pyplot_tutorial.html) to plot the following sequences for $2 \leq n \leq 100$.

a) $f_n = n^2$

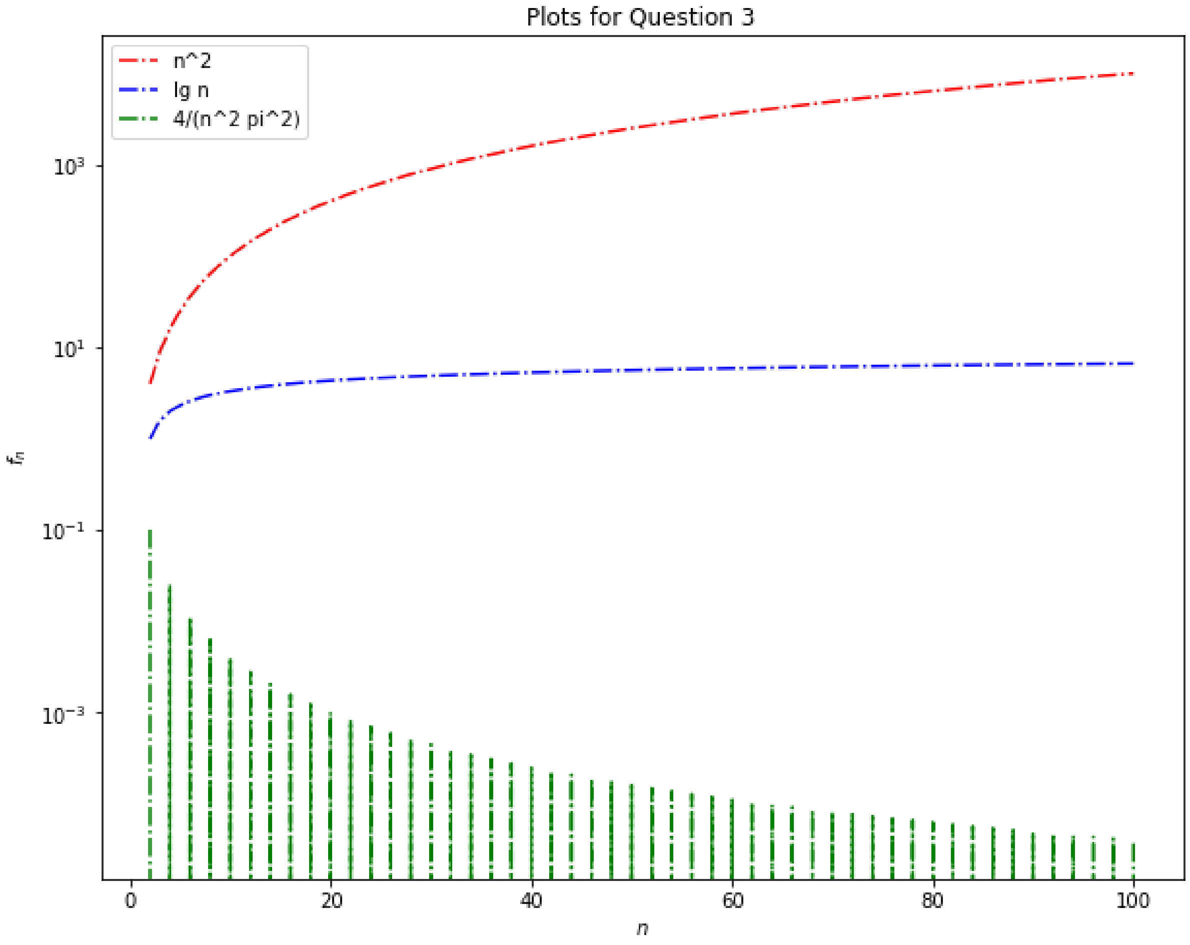
b) $f_n = \log_2 n$
(Use `math.log2(x)` to compute base 2 logarithms. You will need to `import math`)

c) $f_n = \begin{cases} \frac{4}{n^2\pi^2} & \text{if } n \text{ is odd,} \\ 0 & \text{otherwise.} \end{cases}$ (Use `math.pi` for π. You will need to `import math`)

In order to compare the relative growth rates, ***please plot within the same figure***. Use different line styles so that the sequences can be distinguished, and label your axes appropriately. Please also use logarithmic scaling on the vertical axis (`plt.yscale('log')`) so that the relative magnitudes of the sequences is more apparent.

In [76]:

```
''''''
### BEGIN SOLUTION

import math
import matplotlib.pyplot as plt

# First, create lists for the three functions to plot
x = list(range(2,101))
fa = [n**2 for n in x]
fb = [math.log2(n) for n in x]
fc = [4/(n**2 * math.pi**2) if n%2==0 else 0 for n in x]

plt.figure(figsize=(10,8))
plt.plot(x, fa, 'r-.', label="n^2")
plt.plot(x,fb, 'b-.', label="lg n" )
plt.plot(x, fc, 'g-.', label="4/(n^2 pi^2)")
plt.yscale('log')
plt.xlabel('$n$')
plt.ylabel('$f_n$')
plt.title('Plots for Question 3')
plt.legend()
plt.show()


### END SOLUTION
```



Question 4: Estimating a Binomial Distribution

(7 points)

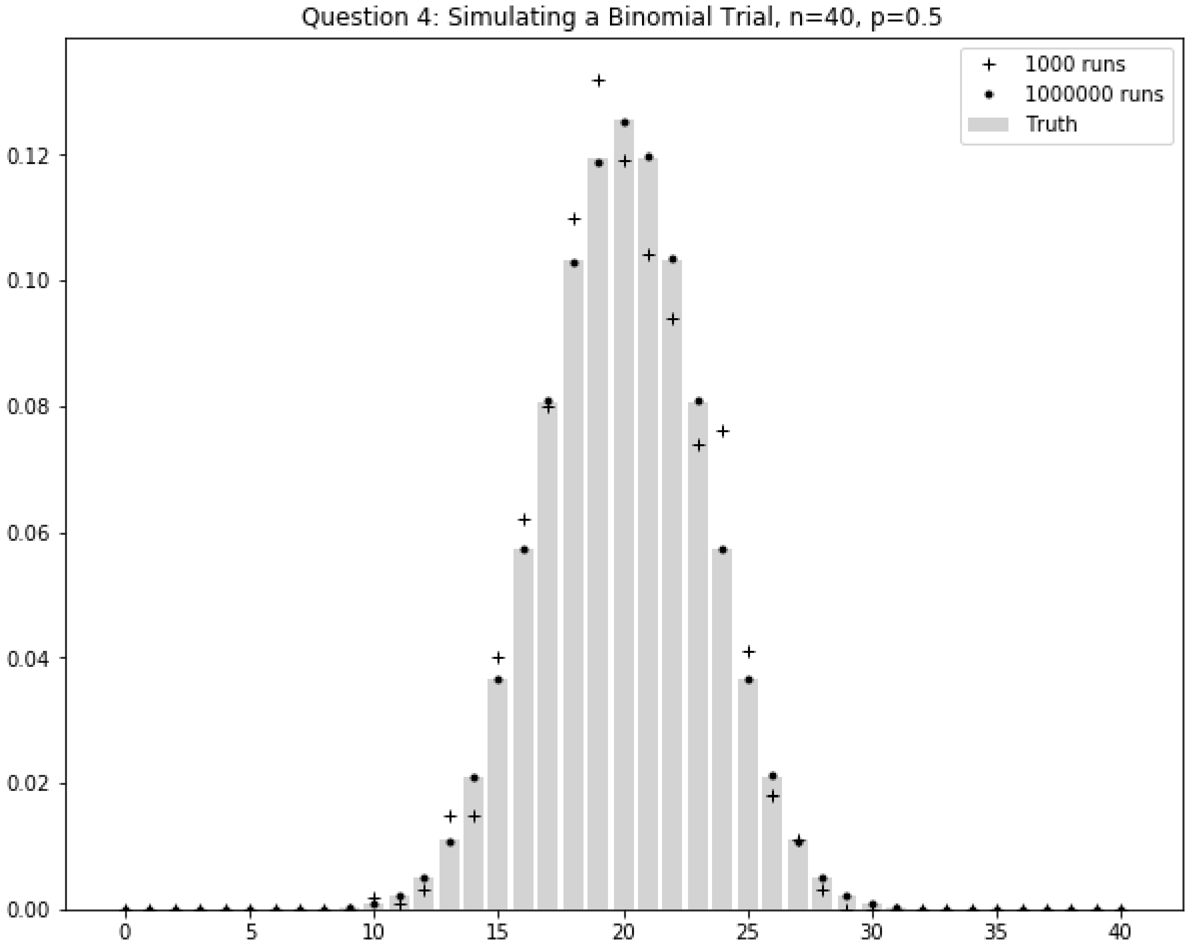
This question asks you to empirically estimate a binomial distribution by simulating binomial trials. A high-level description of the tasks that you need to perform is provided below. You will need to think about suitable data structures and programming constructs that will accomplish the tasks. You may use built-in (https://docs.python.org/3/library/functions.html) functions.

1. Write a function to simulate a binomial experiment. Your function should return the number of successess in n repeated trials where the probability of success for each trial is p. Take $n = 40$ and $p = 0.5$. Use a random number generator to determine if the outcome is a success or a failure. Let ξ be a uniformly distributed random number in the range $[0, 1)$. If $\xi < p$, then the outcome is a success, otherwise it is a failure. You can use `math.random.random()` to generate a uniformly distributed random `float` in the range $[0, 1)$.
2. Repeat the above experiment N times to determine an empirical distribution corresponding to the probability of k success in n trials. Determine two empirical distributions by taking $N = 10^3$ and $N = 10^6$.
3. On the same figure, plot the empirical distributions corresponding to $N = 10^3$ and $N = 10^6$. For comaprison, also plot the true binomial distribution for this scenario. How do the empirial distributions compare to the true distribution?

Please answer this question by inserting one ore more code cells below. Please use a Markdown cell to explain your findings.

In [121]:

```
### BEGIN SOLUTION

# Factorial and combination definition
def factorial(n):
    "Returns n!, the factorial of a non-negative integer n"
    # The variables n and result have local scope.

    result = 1
    while n > 0:
        result *= n
        n -= 1
    return result

def binomial(n,k,p):
    return (factorial(n) / (factorial(n-k)*factorial(k))) * p**k * (1-p)
**(n-k)


# Run the experiments

# Number of times to run
N = [10**3, 10**6]


# parameters for the binomial distribution
n = 40
p = 0.5
truth = [binomial(n,k,p) for k in range(n+1)]

result = [[0]*(n+1) for i in range(len(N))]
random.seed()

for i in range(len(N)):
    for exp in range(N[i]):
        trial = [1 if random.random() < p else 0 for i in range(n)]
        result[i][trial.count(1)] += 1

# normalize result so it is a proper PMF
for i in range(len(N)):
    result[i] = [v/N[i] for v in result[i]]

# Plot results. Could be wrapped in a loop with plot specs
plt.figure(figsize=(10,8))
plt.plot(list(range(n+1)), result[0], 'k+', label="{0} runs".format(N[0
]))
plt.plot(list(range(n+1)), result[1], 'k.', label="{0} runs".format(N[1
]))

# Plot the actual binomial distribution for comparison
plt.bar(list(range(n+1)), truth, color='lightgray', label="Truth")

plt.legend()
plt.title("Question 4: Simulating a Binomial Trial, n=40, p=0.5")
plt.show()

### END SOLUTION
```


Question 4: Simulating a Binomial Trial, n=40, p=0.5
+ 1000 runs
● 1000000 runs
Truth
0.12
0.10
0.08
0.06
0.04
0.02
0.00
0
5
10
15
20
25
30
35
40