Jessica Hamilton
Computational
Exam 01

1. General python questions:
    a. Explain the similarities and differences between the tuple, list, and numpy array data types.
        i. A tuple looks like a list, but it is immutable. If you need an object to hold values but not do anything, a tuple is good. There is not a way to modify the values within the tuple. A list is mutable just as an array is mutable. Arrays are great as running iterations through for loops and you can apply mathematical functions to the array as a whole rather easy. Lists though are easier to work with when you are trying to delete or replace particular values within it. But you can not add multiple lists together as you can with numpy arrays.
    b. Explain the difference between a for loop and a while loop. Which is better to avoid getting stuck in an infinite loop.
        i. With a for loop, the number of iterations are defined when setting up the loop, the counter of the iterations is also known and is set up from the beginning. A while loop does not specify the number of iterations, it will continue until a specified condition is met. The likelihood of getting stuck in an infinite loop is higher with a while loop. Especially since the number of iterations are unknown and a condition has to be met in order to exit the loop. If that condition is very specific, there is a greater chance the loop will continue indefinitely, or at least an extremely long time.
    c. It is often said that everything in python is an object. Explain what is meant by this statement? How might this make your life easier and/or harder as you're coding?
        i. It is said that everything is python is an object because it can be assigned to a variable and it can be passed to a function as an argument. This does not necessarily mean each object will have attributes and methods, but it can. Or it could have one or the other. If everything is considered an object, you can consider many ways you can work with/ handle the object. This can allow you to use the word 'book' as an array, a list, an integer, a float, a class, a function, etc. It is more than just an actual book, or even just a string.
    d. Find a simple "object" in your room. What object did you find? Describe it and/or attach a picture. Imagine that you want to turn it into a Python object. Which attributes and methods would you assign your object? {You need at least 3 of each]
        i. My object is my UNGLeads water bottle. My water bottle has the colors of blue, white and gold. It can contain over 32 ounces of fluids, marked with ridges noting how many ounces exists at that level of the bottle. It has a

screw top and a rubber ring to assist in twisting the top to open or close. If considered a python object. Attributes could be the following: containers (class), water-bottle (class), university promotional item (instance), jug (instance), etc. Methods could consist of append (add stuff into the bottle), remove (stuff out of bottle), sum (total what is inside), etc.

2. When integrating the function f(x) = x 2 on an interval from a = 0 to b = 104 using the limit definition of the integral (i.e., Exercise 09), is it more accurate to perform the summation from a → b, from b → a, or does it not matter? Assume a fixed bin width h<<1. Explain your reasoning and/or your results (e.g., if you actually code it). If there's a difference between the two summation directions, explain the reasons for the difference.
    a. When integrating over such a large interval with a small width, it is best to integrate from $a \rightarrow b$. This allows the computer to hold onto more precise values while iterating over the interval. This is due to the fact that the computer has a built-in limit to the amount of digits it can retain and once you have several large numbers already, it is harder and harder to add smaller values(10*(-9) and have them actually affect the resultant value. Therefore, starting with the longest values (lowest) and going to the smaller values(largest) it best. I believe on a smaller scale, more on the lines of exercise 09, it would not matter as much since the precision is not to that many decimal places.
3. Consider the following problem. Explain whether the Monte Carlo method would be appropriate for calculating a numerical solution: Rockets accelerate by expelling some of their mass at high velocity to create thrust. Newton's second law and the conservation of momentum imply that the rocket will move. The problem is to calculate a rocket's velocity as a function of time, including effects due to air resistance, and then calculate the total amount (and thus cost) of fuel to put the rocket into a geostationary orbit. - do not solve
    a. The Monte Carlo method would not be a way to determine the velocity of a rocket as a function of time including air resistance. Perhaps we could determine it starting with the acceleration function and using the definition of an integral to approximate it over a known interval, such as the time it takes to reach the orbit or the final position. We can then relate that to the amount of fuel that is needed. A better method to determine the velocity as a function of time will be to employ a (finite) difference method based on the definition of a derivative. This can allow us to find the function based on random sampling and estimate the total amount of fuel needed to put the rocket into geostationary orbit.
4. Choose 1 of the following problems. Write pseudo-code to solve the problem and then actually code up the problem to find a solution. Explain your results.

    B. Hi-Ho Cherry-0: pseudo code as follows: See exam01.py for actual code

```
Import numpy as np
Import matplotlib.pyplot as plt
Import numpy.random as rand

#two different functions for each version?
```

```
Def old_cherry
Def new_cherry

#initialize values for the tree and the basket and set range for turns
Cherry_tree = 10
Basket = 0
For turn in range(50):
    #randomly generate integers between 0 and 7 to simulate the different
    options the spinner can land on
    Spin = rand.uniform(0,6)

    #**Set up  for second/new version with no weights:**
    #Setup for loop for spins and if statement to clarify what happens in each
    spin. May not need for loop for spin.
    For each_value in range(0,spin):
        If each_value ==0:
            Cherry_tree = cherry_tree -1
            Basket = Basket + 1
            Elif len(Basket) == 10:
                Print("Number of turns",: turns)
                break
        Elif each_value == 1:
            Cherry_tree = cherry_tree - 2
            Basket = Basket + 2
            Elif len(Basket) == 10:
            Print("Number of turns",: turns)
                break
        Elif each_value ==2:
            Cherry_tree = cherry_tree - 3
            Basket = Basket + 3
            Elif len(Basket) == 10:
                Print("Number of turns",: turns)
                break
        Elif each_value ==3:
            Cherry_tree = cherry_tree -4
            Basket = Basket + 4
            Elif len(Basket) == 10:
                Print("Number of turns",: turns)
                break
        Elif each_value == 4 or each_value ==5:
            If len(basket) < 2 and >0:
                Cherry_tree = Cherry_tree +1
                Basket = Basket -1
                elf len(basket>=2:
                        Cherry_tree = Cherry_tree +2
                        Basket = Basket - 2
                Else:
                        continue
        Elif each_value == 6:
                Length = len(Basket)
                Cherry_tree = cherry_tree + length
                Basket = 0
```

#Set up for initial/original version with weights:
#setup a second if statement to randomly generate a value between 0 and 1 for the weight of the spin
#Setup for loop for spins and if statement to clarify what happens in each Spin, may not actually need for loop!

```
Weight = rand.random()
        If weight >= 0.75:
                Spin = rand.uniform(4,6,1)
        Else:
                Spin = rand.uniform(0,3,1)
For each_value in range(0,spin):
        If each_value ==0:
                Cherry_tree = cherry_tree -1
                Basket = Basket + 1
                Elif len(Basket) == 10:
                        Print("Number of turns",: turns)
                        break
        Elif each_value == 1:
                Cherry_tree = cherry_tree - 2
                Basket = Basket + 2
                Elif len(Basket) == 10:
                Print("Number of turns",: turns)
                        break
        Elif each_value ==2:
                Cherry_tree = cherry_tree - 3
                Basket = Basket + 3
                Elif len(Basket) == 10:
                        Print("Number of turns",: turns)
                        break
        Elif each_value ==3:
                Cherry_tree = cherry_tree -4
                Basket = Basket + 4
                Elif len(Basket) == 10:
                        Print(“Number of turns”,: turns)
                        break
                Elif each_value == 4 or each_value ==5:
                    If len(basket) < 2 and >0:
                        Cherry_tree = Cherry_tree +1
                        Basket = Basket -1
                    elf len(basket>=2:
                        Cherry_tree = Cherry_tree +2
                        Basket = Basket - 2
                    Else:
                        continue
                Elif each_value == 6:
                    Length = len(Basket)
                    Cherry_tree = cherry_tree + length
                    Basket = 0
        #if in a function:
        Turns_played = turn
        Return Turns_played
    Print(“The amount of turns it takes to win:”, turn)
```

Initial thoughts and assumptions:
Here we are assuming that the function is focused on the probability for one player and there is one cherry tree and basket per person. With the old version, the area of the triangles for each spin is not equal. The first four triangles are equal in area and have equal probability of being picked by the spinner "finger" so to speak. The next three triangles also have the same area as each other but are half as big as the first four and therefore have a lower probability of being picked. So In my function, I am setting the weight approximately to be a 4:3 ratio where the first four will have a 4/7 probability and the last three will have a 3/7 probability. Once the weight is determined by one randomly generated value, the function generates random values to match the triangles with a random value generated with a uniform distribution. This allows us to weigh the different probabilities for 1-4 and 5-7 but weight the individual triangles evenly. With the new version, all triangles have the same area and they have the same probability for the spinner ‘finger’ to pick it. So, here in my function, I chose to randomly generate integers in a uniform distribution. The key to this function is to make sure to update the cherry tree, basket, and amount of turns it takes to complete correctly.

Results from coding when estimating number of turns (N=50) from a total of 100 games:
Average number of Turns for old version: 26.1
Average number of Turns for new version: 39.0

When running the code for the game, the newer version takes more turns in general to win versus the older version. This is reasonable when you consider the probability of getting one of the spins increasing the number of cherries in your basket is greater than the spins decreasing the number of cherries in your basket. This will cause the number of turns necessary to win to be smaller than if the probability of getting a spin to increase or decrease the number of cherries is equal for all. With that being said, if you are designing this game for young children, the old version will be better for them since their attention span is far shorter and winning a game quicker is better for them. Otherwise, you risk losing their attention and they will wander off. As for my daughter’s sake, if she had a greater chance in getting a turn that takes away the cherries, she would get frustrated more quickly and leave the game faster. The quicker the game can play out, the better for everyone :)