PHY1112: Home Assignment 5

> Going in circles

Assigned: February 6th, 2024

Due: February 13th, 2024

Learning Objectives

1. Practice recursion
2. Practice debugging
3. Practice read-in of datasets using NumPy

Grade Breakdown

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Part | 1 | 2 | 3 | Total |
| Points | 7 | 7 | 7 | 21 |
| Score |  |  |  |  |

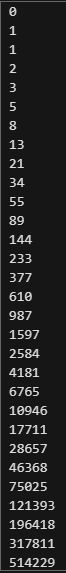
**Question 1: Fibonacci Frenzy.**

In the lab, you practiced recursion by making your own factorial function. Next, let’s approach pattern generation using the Fibonacci Sequence as an example. In case you’re unfamiliar with it, the Fibonacci sequence is a pattern of integers such that the next integer is the sum of the previous two. Like this:

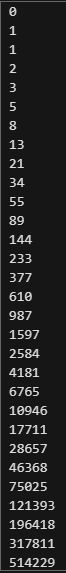
Mathematically, this is defined:

, where

1. Using recursion, write a function `fibonacci\_term` that takes an integer as the input, and returns the number located at that location in the Fibonacci sequence. Using this function, generate the first 30 numbers of the Fibonacci sequence, and take a screenshot of your results.  
    **(2 marks)**



1. Now, write a `fibonacci\_sequence\_loop` function that does the same thing as your function in part a). Use it to generate the first 30 terms of the Fibonacci sequence, and take a screenshot of your results.  
   **(2 marks)**



1. Using the Python built-in time module, determine which of your functions in parts a) and b) is faster at generating the first 30 terms. Why is this the case? Include a screenshot of your results.

**(2 marks)**





**(7 marks total, 1 for docstrings/file header/variable naming/comments)**

**Question 2: Unit Testing -- A Quest for Quality**

For this question, please refer to your lab’s example of multiplication by recursion:

def recursive\_multiplication(a, b):

if b == 0:

return 0

return a + recursive\_multiplication(a, b - 1)

The lab included some example code on testing this recursive multiplication for a variety of inputs, meant to act as a guide for when you wrote your own tests. However, this example did not cover all cases, more specifically, the case of negative ‘a’ and/or ‘b’.

Using the recursive multiplication function from the lab:

1. Write tests for the three missing situations:  
     
   a < 0, b > 0  
   a > 0, b < 0  
   a < 0, b < 0  
     
   Assess whether the function is working as expected for these scenarios. Include a screenshot of your results.  
   **(3 marks)**

A black background with white numbers and symbols

Description automatically generated

1. Some of these situations should not function properly. Fix the function to perform multiplication properly for these inputs, test them again, and return a screenshot of your results.   
   **(3 marks)**

The function works . . . no need for edits (?)

**(7 marks total, 1 for docstrings/file header/variable naming/comments)**

**Question 3: Rigid Read-in**

Using what you know about NumPy arrays, this question will practice reading in data from files. In particular, we will look at rigid bodies that possess an ID number that identifies them, a vector for their position and a vector for their velocity.

The positions and velocities for each rigid body are given in “PHY1112\_A5\_Q3\_Positions.csv” and “PHY1112\_A5\_Q3\_Velocities.csv”, respectively. Both files also contain the ID number.

1. Using one of `np.genfromtxt`, or `np.loadtxt` from NumPy, read in both the position and velocity vectors, and store them into arrays. As these are 3D vectors, and there are 20 rigid bodies, these arrays should be 20 rows and 3 columns (confirm this).   
   **(2 marks)**
2. Taking advantage of the element-wise nature of the NumPy array, calculate the distance of each of the bodies from the origin, and their speed. Your answers should be two arrays that have a length of 20. Print these to the terminal.  
   **(2 marks)**
3. Which rigid body is closest to the origin? How close is it?

Which rigid body is moving the fastest? How fast is it moving?

Print these results to the terminal and include screenshots of all your results.   
  
*Useful NumPy functions for this might include `*argmin*`, and `*argmax*`*   
**(2 marks)**

**(7 marks total, 1 for docstrings/file header/variable naming/comments)**