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 |  |  |  |  |

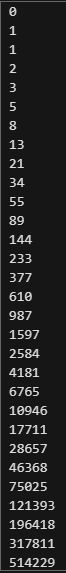
**Question 1: Fibonacci Frenzy.**

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:

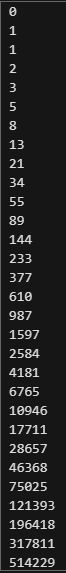
Mathematically, this is defined:

, 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)**

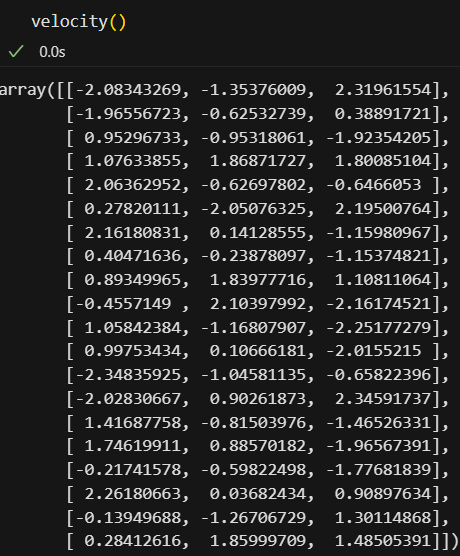
The function works . . . no need for edits (?)

**(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)**  
   **A screenshot of a computer

   Description automatically generated**
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)**

A black background with white numbers

Description automatically generated

A screenshot of a computer

Description automatically generated

1. Which rigid body is closest to the origin? How close is it?

A black background with white text

Description automatically generated

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

A black background with white text

Description automatically generated

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)**

**CODE:**

'''

Filename:       a5\_300349204.py

Author:         Patrick Geraghty

Date Created:   2024-02-11

Date Modified:  2024-02-11

Description:    Contains functions for Assignment 5 and additional functions. Functions include: 'fibonacci\_term', 'fibonacci\_sequence\_loop', 'fibber\_rec', 'fibber\_loop', 'recursive\_multiplication', 'rec\_mult\_loop\_output', 'velocity', 'position', 'distance\_from\_origin', 'velocity\_magnitude', 'closest\_body', and 'fastest\_body'. See docstrings for details.

'''

import time as t

import numpy as np

# Fibonacci Frenzy

# define function 'fibonacci\_term'

*def* fibonacci\_term(*n*):

    '''

    (int) -> int

    Returns the nth term of the Fibonacci sequence using recursion.

    Preconditions: n >= 0, n is an integer.

    '''

    # define cases

    if n == 0:                      # if n is 0, return first term

        return 0

    elif n == 1:                    # if n is 1, return second term

        return 1

    else:                           # if n is greater than 1, calculate nth term

                                    # calculate nth term using recursion

        return fibonacci\_term(n-1) + fibonacci\_term(n-2)

# define function 'fibonacci\_sequence\_loop'

*def* fibonacci\_sequence\_loop(*n*):

    '''

    (int) -> int

    Returns the nth term of the Fibonacci sequence.

    Precondition: n >= 0, n is an integer.

    '''

    n0 = 0                          # set first term of sequence

    n1 = 1                          # set second term of sequence

    # define cases

    if n == 0:                      # if n is 0, return first term

        return n0

    elif n == 1:                    # if n is 1, return second term

        return n1

    else:                           # if n is greater than 1, calculate nth term

                                    # loop through sequence to calculate nth term

        for i in range(2, n+1):     # set range from 2 to n+1 as cases for 0 and 1 are already defined

                                    # calculate nth term

            n2 = n0 + n1

            n0 = n1

            n1 = n2

        # return nth term

        return n2

# simple function to loop through n cases of fibonacci\_term

*def* fibber\_rec(*x*):

    # use time module to measure time

    t0 = t.perf\_counter()

    for i in range(x):

        print(fibonacci\_term(i))

    t1 = t.perf\_counter()

    print(*f*"Recursion time: {t1-t0}")

# simple function to loop through n cases of fibonacci\_sequence\_loop

*def* fibber\_loop(*x*):

    # use time module to measure time

    t0 = t.perf\_counter()

    for i in range(x):

        print(fibonacci\_sequence\_loop(i))

    t1 = t.perf\_counter()

    print(*f*"Loop time: {t1-t0}")

# 2. A Quest for Quality

# define function 'recursive\_multiplication'

*def* recursive\_multiplication(*a*, *b*):

    '''

    (int, int) -> int

    Returns the product of two integers using recursion.

    Preconditions: a and b are integers.

    '''

    # define base case

    if b == 0:                      # if b is 0, return 0

        return 0

    elif a == 0:                      # if a is 0, return 0

        return 0

    elif a < 0 and b < 0:             # if a and b are both negative, return positive product

        return recursive\_multiplication(-a, -b)

    elif a < 0:                       # if a is negative, return negative product

        return -recursive\_multiplication(-a, b)

    elif b < 0:                       # if b is negative, return negative product

        return -recursive\_multiplication(a, -b)

    else:                           # if b is not 0, calculate product using recursion

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

# define simple loop for test cases

*def* rec\_mult\_loop\_output(*a*, *b*):

    '''

    (int, int) -> None

    Returns the products of a range of two integers using recursion.

    Preconditions: a and b are integers.

    '''

    # note that the range of a is equal to the range of b

    for i in range(a, b):

        for j in range(a, b):

            print(*f*"{i} \* {j} = {recursive\_multiplication(i, j)}")

# 3. Rigid Read-In

# define function 'velocity' to contain the data from 'PHY1112\_A5\_Q3\_Velocities.csv' in an array

*def* velocity():

    '''

    () -> np.array

    Returns an array of velocities containing the relevant data in 'PHY1112\_A5\_Q3\_Velocities.csv'.

    Preconditions: 'PHY1112\_A5\_Q3\_Velocities.csv' is a valid file.

    '''

    # load the data from the file using np.genfromtxt. Define necessary columns, skip the header, identify the separator, and define the data type as float

    return np.genfromtxt('PHY1112\_A5\_Q3\_Velocities.csv', *skip\_header*=1, *usecols*=(1,2,3), *delimiter*=',', *dtype*=*float*)

# define function 'position' to contain the data from 'PHY1112\_A5\_Q3\_Positions.csv' in an array

*def* position():

    '''

    () -> np.array

    Returns an array of positions containing the relevant data in 'PHY1112\_A5\_Q3\_Positions.csv'.

    Preconditions: 'PHY1112\_A5\_Q3\_Positions.csv' is a valid file.

    '''

    # load the data from the file using np.genfromtxt. Define necessary columns, skip the header, identify the separator

    return np.genfromtxt('PHY1112\_A5\_Q3\_Positions.csv', *skip\_header*=1, *usecols*=(1,2,3), *delimiter*=',', *dtype*=*float*)

# define function 'distance\_from\_origin' to calculate the distance of each body from the origin

*def* distance\_from\_origin():

    '''

    () -> np.array

    Returns an array of distances from the origin for each body using data from the 'position()' function.

    Preconditions: None

    '''

    pos\_x = position()[:,0]     # set x position to the first column of the position array

    pos\_y = position()[:,1]     # set y position to the second column of the position array

    pos\_z = position()[:,2]     # set z position to the third column of the position array

    # calculate the distance from the origin using the Pythagorean theorem and return the array

    return np.array([np.sqrt(pos\_x[i]\*\*2 + pos\_y[i]\*\*2 + pos\_z[i]\*\*2) for i in range(len(pos\_x))])

# define function 'velocity\_magnitude' to calculate the magnitude of the velocity of each body

*def* velocity\_magnitude():

    '''

    () -> np.array

    Returns an array of the magnitude of the velocity for each body using data from the 'velocity()' function.

    Preconditions: None

    '''

    vel\_x = velocity()[:,0]     # set x velocity to the first column of the velocity array

    vel\_y = velocity()[:,1]     # set y velocity to the second column of the velocity array

    vel\_z = velocity()[:,2]     # set z velocity to the third column of the velocity array

    # calculate the magnitude of the velocity using the Pythagorean theorem and return the array

    return np.array([np.sqrt(vel\_x[i]\*\*2 + vel\_y[i]\*\*2 + vel\_z[i]\*\*2) for i in range(len(vel\_x))])

# define function 'closest\_body' to find the body closest to the origin

*def* closest\_body():

    '''

    () -> None

    Prints the body closest to the origin and the distance from the origin.

    Preconditions: None

    '''

    distances = distance\_from\_origin() # set distances to the array of distances from the origin

    # print the body closest to the origin and the distance from the origin using an f-string

    print(*f*'The closest body to the origin is body {np.argmin(distances) + 1} at a distance of {np.min(distances)}')

# define function 'fastest\_body' to find the body with the fastest velocity

*def* fastest\_body():

    '''

    () -> None

    Prints the body with the fastest velocity and the velocity.

    Preconditions: None

    '''

    velocities = velocity\_magnitude() # set velocities to the array of magnitudes of the velocity

    # print the body with the fastest velocity and the velocity using an f-string

    print(*f*'The fastest body is body {np.argmax(velocities) + 1} with a velocity of {np.max(velocities)}')