PHY1112 Lab 5

Data and Recursion – Down the Rabbit Hole

February 6th, 2024

|  |  |  |  |
| --- | --- | --- | --- |
| Part | 1 | 2 | Total |
| Points | 15 | 15 | 30 |
| Score |  |  |  |

Objectives

1. Create a recursive function to calculate factorials.
2. Unit test corner cases and compare with math.factorial().
3. Read in and use a large data set using NumPy.

Part 1: Recursion and testing – Going down the rabbit hole and figuring out how to get back up

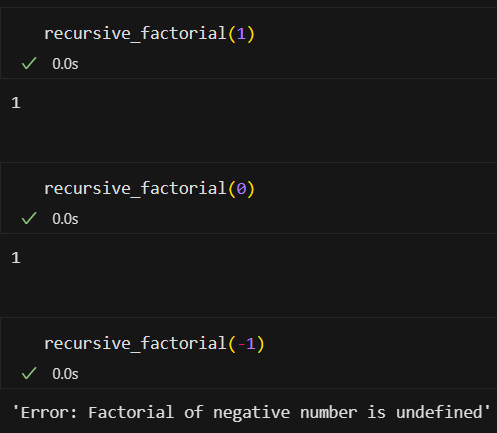
1. (5 points) Create a new file named “lab5.py” and create a recursive function named recursive\_factorial() that calculates the factorial

for some integer a. Note that this can be written in a closed form like above as

.

1. A screenshot of a computer

   Description automatically generated(5 points) Run your recursive\_factorial() function for a suitable set of inputs, including edge cases, and work out the math by hand below to show that your function evaluates to the correct answer.



1! = 1

2! = 2\*1 = 2

3! = 3\*2\*1 = 6

4! = 4\*3\*2\*1 = 24

Edge cases n = 0 results in 1 (product of no values is 1), n < 0 is undefined.

1. (5 points) Write a unit test for your recursive\_factorial() function in a similar format as the example below for the function recursive\_multiplication(). Make sure to follow the guidelines laid out in lecture 8.

A screenshot of a computer program

Description automatically generated

Here is an example recursive function that does multiplication for an integer b (a bit different than what was done in lecture:

def recursive\_multiplication(a, b):

if b == 1:

return 0

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

# the following unit test would suffice to test recursive\_multiplication()

inputs = [(0,0), (1,0), (0,1), (0,10), (3,5), (4,6)]

outputs = [0, 0, 0, 0, 15, 24]

for i in range(len(inputs)):

check\_string = “(“ + str(inputs[i][0]) + “\*” + str(inputs[i][1]) + “)”

check\_value = recursive\_multiplication(inputs[i][0],inputs[i][1])

check\_result = (check\_value == outputs[i])

print(check\_string, “ == ”, str(outputs[i]), “? ”, check\_result, sep=’’)

Here is what would be printed to the screen. We want all the tests to be “True”.

(0\*0) == 0? True

(1\*0) == 0? True

(0\*1) == 0? True

(0\*10) == 0? True

(3\*5) == 15? True

(4\*6) == 24? True

Part 2: Handling large data – NumPy to the rescue!

1. (5 points) Read in the daily high temperature and daily low temperature data in Ottawa during the year 2022 from the file “weather\_data\_lab5.csv” (found on Brightspace) using np.genfromtext(). There should be one year’s worth of weather data in that file, however the only columns of interest are the “Max Temp (°C)” and “Min Temp (°C)” columns.
   * The usecols keyword argument will be helpful here for only grabbing the columns of interest.
2. (10 points) Using vectorization, determine the lowest and highest daily temperatures in Ottawa in 2022, as well as on which day they occurred (between day 0 and 364). Further, determine the largest difference between the daily high and low in any one day, and also on which day that occurred.
   * Note that there are some skipped entries in the data. These skipped entries will have values of NaN, and as such we would want to skip them in our statistics. To skip them, use the np.nanmax(), np.nanmin(), np.nanargmax(), and np.nanargmin() functions.

A screen shot of a computer

Description automatically generated

**Remember to submit this document filled out, along with your python file on Brightspace!**

CODE:

'''

Filename:       lab5.py

Author:         Patrick Geraghty

Date Created:   2024/02/06

Date Modified:  2024/02/06

Description:

'''

# Part 1: Factorials

# define a function 'recursive\_factorial'

*def* recursive\_factorial(*n*):

    '''

    (int) -> int

    This function takes an integer n and returns the factorial of n. The function uses recursion to calculate the factorial of n.

    Preconditions: n is an integer

    '''

    # base cases

    # if n is 1, return 1

    if n == 1:

        return 1

    # if n is 0, return 1 (product of no values)

    elif n == 0:

        return 1

    # if n is negative, return an error message

    elif n < 0:

        return "Error: Factorial of negative number is undefined"

    # recursive case

    else:

        return n \* recursive\_factorial(n-1)

# define a function to test the recursive\_factorial function

*def* factorial\_test(*test\_cases*):

    '''

    (list) -> None

    This function takes a list of integers and prints the factorial of each integer in the list and compares it to a long-hand computation of the factorial.

    Preconditions: test\_cases is a list of integers

    '''

    # iterate through the test cases

    for i in range(len(test\_cases)):

        # set the factorial to 1

        factorial = 1

        # iterate through the range of the test case and multiply the factorial by the value

        for j in range(1, test\_cases[i] + 1):

            factorial \*= j

        # print the factorial and the comparison as a boolean

        print(*f*'{test\_cases[i]}! = {factorial}, comparison: {factorial == recursive\_factorial(test\_cases[i])}')

# Part 2: Handling Large Data Sets

import numpy as np

# define a function to load the data

*def* load\_data():

    '''

    () -> np.array

    This function returns a numpy array of the data in the 'weather\_data\_lab5.csv'.

    Preconditions: filename is a string

    '''

    # 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('weather\_data\_lab5.csv', *usecols*=(9,11), *skip\_header*=1, *delimiter*=',', *dtype*=*float*)

# define a function to calculate the weather statistics

*def* weather\_stats():

    '''

    () -> None

    Prints the lowest and highest temperatures as well as the day (0-364) when they occurred. Furthermore, prints the largest difference between the daily high and low in any one day, as well as the day which it occured.

    Preconditions: None

    '''

    # load the data

    data = load\_data()

    # define the high and low temperatures

    high = data[:,0]

    low = data[:,1]

    # calculate the difference between the high and low temperatures

    diff = high - low

    # print the statistics

    # use np.nanmin and np.nanmax to ignore NaN values and find the max and min within the high and low lists. Use np.nanargmin and np.nanargmax to find the index of the max and min values.

    print(*f*'Lowest temperature: {np.nanmin(low)} on day {np.nanargmin(low)}')

    print(*f*'Highest temperature: {np.nanmax(high)} on day {np.nanargmax(high)}')

    # use np.nanmax to ignore NaN values and find the max difference between high and low. Use np.nanargmax to find the index of the max difference.

    print(*f*'Largest difference between high and low: {np.nanmax(diff)} on day {np.nanargmax(diff)}')