PHY1112: Assignment 3

> Perfectly Functional

Assigned: January 23rd, 2024

Due: January 30th, 2024

Learning Objectives

1. Practice looping
2. Practice conditional statements
3. Defining functions and their required and optional arguments

Grade Breakdown

|  |  |  |  |
| --- | --- | --- | --- |
| Part | 1 | 2 | Total |
| Points | 9 | 10 | 19 |
| Score |  |  |  |

**Question 0: Stringing Me Along**

When writing your own functions, in this assignment as well as those that follow, it is very important to include *docstrings*. These are comments similar to a file header, however they go immediately beneath the function declaration.

It is good to include a description of the function, the input arguments, and the output. An example docstring would look like this:



These are immensely useful, and many IDEs will automatically parse them and display them like a help message in real time as you are writing your code. For example, look what happens as I open the bracket to a function call with `pow`:

A screenshot of a computer

Description automatically generated

Thus, when grading your code, things like docstrings will be looked for. **Include docstrings on all of your functions for this course from this assignment onward.**

**Question 1: Weather revisited**

In lab 3, you performed a counted loop (“for loop”) over a list of daily high temperatures in order to count the number of “hot”, “warm” and “temperate” days. This question will expand on this problem, now adding the data for daily low temperature in the same time period.

Using the template provided (“PHY1112\_A3\_Q1\_Template.py”) that already contains high and low temperature data:

1. Write a function `find\_max\_temperature` and use it to determine the *maximum* daytime temperature and the *minimum* nighttime temperature. Report your results.

Note: you may not use the built-in `max()` function.

The maximum temperature throughout the month was 32.6°C, and the minimum temperature was 4.5°C

A screen shot of a computer program

Description automatically generated

Hint: how could one manipulate the data to use a maximum function to find a minimum?  
**(4 marks)**

The lowest temperature can be found by inverting the sign of the given values (i.e. take the negative of each value) such that the maximum value would be the least negative value, which corresponds to the smallest value in the list.

The lowest temperature can also be found using the method demonstrated in the function, where instead of looking for a value greater than the ‘current\_value’, it looks for one smaller ( > instead of < ).

1. Now loop over the elements in the high\_temperatures and low\_temperatures lists *at the same time*. Take the difference between the high and low temperatures for each day. Have your script determine which day of the month had the greatest overall range of temperature, and what that range was. Report your results.  
   **(4 marks)**

Don’t forget to add docstrings!

The greatest temperature was 17.5°C on the 28th.



**(9 marks total, 1 for doc strings/file header/variable naming/comments)**

**Question 2: Hopefully not too “complex”**

You will revisit the quadratic equation function you created for lab 3. Up to now, complex roots were not really considered. In many physics problems, a complex root may be ignored, but in others you might still want it.

Since many times you don’t want complex roots, but sometimes you do, let us make a *keyword argument* (optional argument) to the quadratic formula function to control whether or not it will ignore complex roots. In the case where complex roots are being ignored, and there happen to be any, they should be returned as None.

Write a script that:

1. Modifies your quadratic equation function to now have a keyword argument `complex\_roots` that defaults to ignoring complex roots. Test each root using conditional statements (if statements) to determine whether or not it is complex, and replace any complex root with `None`.  
   **(2 marks)**
2. Further, add a check to ensure that the function input *a* is non-zero. In the case that it is zero, still find the roots accordingly and return them.

Hint: put a=0 into the quadratic equation, and find an expression for the root, then implement this expression in your code.   
**(2.5 marks)**

1. Print the roots for the following combinations of *a*, *b*, and *c.* Toggle your optional complex roots parameter on and off for each combination (using the keyword argment you created in part a).

Show the output for each case below (0.5 marks each)

|  |  |  |
| --- | --- | --- |
| **a** | **b** | **c** |
| 0 | 2 | 4 |
| 1 | -8 | 15 |
| 1 | 2 | 1 |
| 1 | 2 | 3 |
| 1 | -(1+1j) | 1j |

**A screenshot of a computer program

Description automatically generatedA screenshot of a computer program

Description automatically generated**

**(2.5 marks)**

1. Analytically find the roots for the parameters in the last row of the table in part c). How do these roots compare to the results your code gives? Comment on their differences and why you think they might be different.

**(2 marks)**

After much research, confusion, and pain, I modified the function to account for complex coefficients such that the output roots match the roots returned when solved using symbolab.com.

Don’t forget your doc strings!

**(10 marks total, 1 for doc strings/file header/variable naming/comments)**

**Code**

'''

Filename:       PHY1112\_A3\_Q1\_Template.py

Author:         Patrick Geraghty

Date Created:   2024-01-08

Date Modified:  2024-20-28

Description:    Analyze given temperature data and report the minimum and maximum temperatures.

'''

# daily high temperatures in Ottawa for the month of Septermber 2023

# data retrieved from https://climate.weather.gc.ca/climate\_data/daily\_data\_e.html?StationID=49568&timeframe=2&StartYear=1840&EndYear=2023&Day=9&Year=2023&Month=9#

high\_temperatures = [24.1, 25.0, 29.5, 30.3, 32.5, 32.6, 30.1, 20.2, 22.7, 21.7,

                     24.5, 19.8, 22.0, 16.4, 21.9, 22.9, 24.4, 21.6, 20.2, 18.6,

                     21.3, 22.8, 21.9, 24.8, 21.2, 18.8, 19.2, 22.0, 21.9, 24.0]  # in Celcius

low\_temperatures = [08.2, 12.3, 15.6, 18.0, 18.2, 18.8, 20.5, 13.4, 10.6, 12.1,

                    10.8, 11.9, 12.0, 06.6, 04.8, 06.9, 12.5, 13.6, 10.4, 06.9,

                    05.4, 05.6, 07.1, 08.3, 11.4, 07.5, 04.5, 04.5, 09.4, 10.0]  # in Celcius

# 1a. Find the maximum and minimum temperatures for the month of September 2023.

*def* find\_max\_temperature(*high\_temperatures*):

    '''

    (list) -> float

    Return the maximum value in the list.

    Precondition: the list is not empty.

    '''

    # define a variable to hold the current maximum value to reference against other values in the list

    current\_max = high\_temperatures[0]

    # loop through the list and compare each value to the current maximum value

    for i in high\_temperatures:

        # if the value is greater than the current maximum value, set the current maximum value to the value

        if i > current\_max:

            current\_max = i

    # return the current maximum value

    return current\_max

*def* find\_min\_temperature(*low\_temperatures*):

    '''

    (list) -> float

    Return the minimum value in the list.

    Precondition: the list is not empty.

    '''

    # define a variable to hold the current minimum value to reference against other values in the list

    current\_min = low\_temperatures[0]

    # loop through the list and compare each value to the current minimum value

    for i in low\_temperatures:

        # if the value is less than the current minimum value, set the current minimum value to the value

        if i < current\_min:

            current\_min = i

    # return the current minimum value

    return current\_min

# 1b. Find the average temperature for each day in the month of September 2023.

*def* find\_max\_temperature\_range(*high\_temperatures*, *low\_temperatures*):

    '''

    (list, list) -> list

    Return the day with the greatest temperature range.

    Precondition: the lists are not empty and have the same length.

    '''

    # define a list to hold the temperature ranges

    temperature\_ranges = []

    # loop through the lists and calculate the temperature ranges for each day, appending the value to the list

    for i in range(30):

        temperature\_ranges.append((high\_temperatures[i] - low\_temperatures[i]))

    # set the current maximum value to the first value in the list

    max\_range = temperature\_ranges[0]

    # loop through the list and compare each value to the current maximum value

    for i in temperature\_ranges:

        if i > max\_range:

            max\_range = i

    # return the current maximum value, and the index of the value in the list, adding 1 to the index value to account for the first day being day 1 and not 0.

    return "The maximum temperature range was " + *str*(max\_range) + "\u00B0C on day " + *str*(temperature\_ranges.index(max\_range) + 1) + "."

# 2. Quadratic root solver with complex root keyword argument

# function to calculate the discriminant of a quadratic equation

*def* discriminant(*a*,*b*,*c*):

    '''

    (num,num,num) -> num

    Return the discriminant of the quadratic equation.

    Preconditions: 'a', 'b', and 'c' are all numbers.

    '''

    return (b\*\*2) - (4\*a\*c)

# function to find the roots of a quadratic equation

*def* quad\_root\_finder(*a*,*b*,*c*, *complex\_root* = False):

        '''

        (num,num,num) - > NoneType

        Takes three numbers as parameters 'a', 'b', and 'c', as well as keyword argument 'complex\_roots' and prints the roots of the quadratic equation. If 'complex\_roots' is set to True, the function will print the roots as complex numbers (when applicable). By default, 'complex\_roots' is set to False. Furthermore, if a is zero, the function will print the root of the linear equation.

        Preconditions: 'a', 'b', and 'c' are all numbers.

        '''

        # check the value of a to avoid division by zero. If a is zero, return the root of the linear equation

        if a == 0:

            print(-c/b)

        # check if the discriminant is positive, negative, complex, or zero and return the roots characteristic of the discriminant

        # if the discriminant is complex, check the complex\_root keyword argument to see if the user wants complex roots as this would yield complex roots.

        elif *type*(discriminant(a,b,c)) == *complex*:

            if complex\_root == True:

                # calculate the roots of the quadratic equation with complex numbers for coefficients using formulas obtained from https://math.stackexchange.com/questions/44406/how-do-i-get-the-square-root-of-a-complex-number .

                z = discriminant(a,b,c)

                r = abs(z)

                print((-b + r\*\*0.5\*((z+r)/abs(z+r))) / (2\*a))

                print((-b - r\*\*0.5\*((z+r)/abs(z+r))) / (2\*a))

            else:

                # if the user does not want complex roots, print None

                print(None)

        # if the discriminant is negative, check the complex\_root keyword argument to see if the user wants complex roots as this would yield complex roots.

        elif discriminant(a,b,c) < 0:

            if complex\_root == True:

                # calculate the roots of the quadratic equation with a negative discriminant by separating real values from imaginary values.

                print(*complex*((-b)/(2\*a), + (-(-discriminant(a,b,c))\*\*0.5)/(2\*a)))

                print(*complex*((-b)/(2\*a), - (-(-discriminant(a,b,c))\*\*0.5)/(2\*a)))

            # if the user does not want complex roots, print None

            else:

                print(None)

        # if the discriminant is zero, print the single root of the quadratic equation. No need to check complex\_root keyword argument as this would yield real roots.

        elif discriminant(a,b,c) == 0:

                print((-b)/(2\*a))

        # if the discriminant is positive, calculate the roots of the quadratic equation and print them. No need to check complex\_root keyword argument as this would yield real roots.

        else:

                print((-b + (discriminant(a,b,c)\*\*0.5))/(2\*a))

                print((-b - (discriminant(a,b,c)\*\*0.5))/(2\*a))