Programming Assignment 2

Out here trying to function

## Overview

This assignment is broken into several problems. Some of these problems come with a starter file that you’ll need to download (you can find them in [this folder](https://drive.google.com/open?id=0BwN1j1-X7ce8XzNWQmljc1hWTmc)). Each problem ask you to implement a function (or several functions). The functions are already defined in the starter files, however you need to complete them.

If a problem has a starter file, try to understand what it’s doing first. Trying running it before you write code. In general, each starter file gets input from the user and calls the function(s) that you are being asked to implement. To test your code: run the file, input values, and check that the output is what you expect. I suggest you test multiple input values (I will when I grade it).

For each of these problems, you are not allowed to use any data structures, loops, or external modules. (If you don’t know what those are, that’s ok)

**Due Sunday, September 25 at 11:59PM**

## Problem 1: Arithmetic

Open the *arithmetic.py* starter file. In it you will see 4 functions. Each of these functions takes two input numbers (they can be decimal numbers). Your job is to implement each of these functions so that they return the correct value:

* ***AddNumbers*** should return num\_1 plus num\_2
* ***SubtractNumbers*** should return num\_1 minus num\_2
* ***MultipleNumbers*** should return num\_1 multiplied by num\_2
* ***DivideNumbers*** should return num\_1 divided by num\_2. If num\_2 is 0, return “Error”.

Note that ***AddNumbers*** is already done. You just need to complete the other 3. You can read the comments inside each function for further explanation.

Below are example outputs you should see when the functions are implemented correctly. It’s ok if your final values are off by a very small fraction (23.00000001 vs 23.0).

= user input

Example 1

>>>

Enter first number: 3

Enter second number: 5

3 + 5 = 8

3 - 5 = -2

3 \* 5 = 15

3 / 5 = 0.6

>>>

Example 2

>>>

Enter first number: 3

Enter second number: 0

3 + 0 = 8

3 - 0 = -2

3 \* 0 = 15

3 / 0 = Error

>>>

## Problem 2: You sound like you’re from London!

Open the *imperial\_to\_metric.py* starter file and you’ll see 3 functions. Each of these functions should convert an imperial unit to a metric unit:

* ***GetMilesToKilometers***
  + This function takes distance in miles as a parameter
  + This function should return the distance in kilometers.
  + For reference, 1 mile is equal to 1.61 kilometers.
* ***GetPoundsToKilograms***
  + This function takes weight in pounds (lbs) as a parameter.
  + This function should return the weight in kilograms.
  + For reference, 1 pound is equal to 0.45 kilograms.
* ***GetFahrenheitToCelsius***
  + This function takes the temperature in fahrenheit as a parameter
  + This function should return the temperature in celsius.
  + For reference, [°C = (°F − 32) × 5/9](http://www.rapidtables.com/convert/temperature/fahrenheit-to-celsius.htm).

Below are example outputs you should see when the functions are implemented correctly. It’s ok if your final values are off by a very small fraction (23.00000001 vs 23.0).

= user input

Example 1

>>>

Enter distance (in miles): 26

Enter weight (in pounds): 150

Enter temperature (in fahrenheit): 75

That’s 41.86 kilometers, 67.5 kilograms, and 23.88888888888889 celsius. Now you sound like you’re from London!

>>>

>>>

Enter distance (in miles): 10

Enter weight (in pounds): 10

Enter temperature (in fahrenheit): 0

That's 16.1 kilometers, 4.5 kilograms, and -17.77777777777778 celsius. Now you sound like you're from London!

>>>

## Problem 3: Leap year

Create a new file named *leap\_year.py*. In it, write a function named ***IsLeapYear*** that takes the year (e.g. 2005) as a parameter. Have the function return *True* if the year is a leap year, or return *False* if it’s not.

Here’s how to know if a year is a leap year:

Leap years are any year that can be evenly divided by 4 (such as 2012, 2016, etc)

* **except** if it can be evenly divided by 100, then it isn't (such as 2100, 2200, etc)
  + **except** if it can be evenly divided by 400, then it is (such as 2000, 2400)

For example

* The following **are** leap years: 1600, 2400, 1964, 1968
* The following **are not** leap years:1700, 1800, 2300, 1962

Notes:

* For further explanation of leap years, see [this page](https://www.mathsisfun.com/leap-years.html) and [this wiki page](https://en.wikipedia.org/wiki/Leap_year).
* Make sure your function is exactly named “IsLeapYear”.
* ***IsLeapYear*** should have a single parameter -- year. You can assume the year will be greater than 0.
* ***IsLeapYear*** should return a bool -- either *True* or *False*.
* To test your function, you should call it with different input values. For example, see how your functions are called in the earlier starter files.

When you define ***IsLeapYear***, it should look like this:

def IsLeapYear(year):

# Return True if leap year. Return False otherwise.

Below are more examples of what ***IsLeapYear*** should return given different inputs:

* IsLeapYear(307) -> False
* IsLeapYear(2200) -> False
* IsLeapYear(2000) -> True
* IsLeapYear(1800) -> False
* IsLeapYear(3696) -> True

## Problem 4: Rounding to nearest multiple of N

Create a new file named *rounding.py*. In it, write a function named ***RoundNumToNearestMultiple*** that rounds one number to the nearest multiple of a second number.

For example, if you round 16 to the nearest multiple of 3, you get 15 (multiples of 3 are: 3, 6, 9, 12, 15, 18, etc). If you round 113 to the nearest multiple of 17, you get 119 (multiples of 17 are: 17, 34, 51, 68, 85, 102, 119, etc). If the first number is exactly in between multiples of the second number, then round up. For example, if you round 18 to the nearest multiple 4, you get 20.

***RoundNumToNearestMultiple*** should take two parameters: *num* and *n*. It should return *num* rounded to nearest multiple of *n*. When you define it, it should look like this:

def RoundNumToNearestMultiple(num, n):

# Return num rounded to nearest multiple of n.

Below are more examples of what it should return given different inputs:

* RoundNumToNearestMultiple(9, 2) -> 10
* RoundNumToNearestMultiple(712, 25) -> 700
* RoundNumToNearestMultiple(713, 25) -> 725
* RoundNumToNearestMultiple(0, 5) -> 0
* RoundNumToNearestMultiple(79, 1) -> 79

Notes

* You can assume that *num* and *n* are both whole numbers.
* You can assume *num* is greater than or equal to 0, and that *n* is greater than 0.
* To test your function, you should call it with different input values. For example, see how your functions are called in the earlier starter files.

## Problem 5: Zodiac sign

Create a new file named *zodiac.py*. In it, write a function named ***GetZodiacSign*** that returns the Zodiac sign that corresponds to the input birthday month and day. For example, the input August 6 should return “leo”.

***GetZodiacSign*** should take two parameters: *birth\_month* (a str) and *birth\_day* (an int). It should return the Zodiac sign as a str. When you define it, it should look like this:

def GetZodiacSign(birth\_month, birth\_day):

# Return Zodiac sign as str.

For reference, below are the zodiac signs for each birth date

* aries: march 21 - april 19
* taurus: april 20 - may 20
* gemini: may 21 - june 20
* cancer: june 21 - july 22
* leo: july 23 - august 22
* virgo: august 23 - september 22
* libra: september 23 - october 22
* scorpio: october 23 - november 21
* sagittarius: november 22 - december 21
* capricorn: december 22 - january 19
* aquarius: january 20 - february 18
* pisces: february 19 - march 20

Below are examples of what GetZodiacSign should return given different inputs:

* GetZodiacSign(“march”, 15) -> “pisces”
* GetZodiacSign(“july”, 21) -> “cancer”
* GetZodiacSign(“february”, 14) -> “aquarius”

Notes

* All parameter and returned values should be lowercase. The *birth\_month* str is lower case and the returned zodiac str should be lowercase.
* To test your function, you should call it with different input values. For example, see how your functions are called in the earlier starter files.

## Part 6: Find the intersection

Open the *find\_intersection.py* starter file and you’ll see an incomplete ***GetIntersectionSize*** function. It takes 4 parameters that represent two number ranges. A number range is composed of a min and max number (both are whole numbers). Complete the implementation of ***GetIntersectionSize*** so that it returns the size of the intersection of the two ranges. If there is no intersection, have it return -1.

For example, if the first range is -2 to 7 and the second range is 3 to 9, then the intersection would be 3 to 7. The size of that intersection is 4 (= 7 - 3).



Notes

1. You can assume that for a given range, the min will be less than or equal to the max.

Below are examples of what the program output should be if ***GetIntersectionSize*** is implemented correctly.

= user input

Example 1

>>>

Enter min number for range 1: -2

Enter max number for range 1: 7

Enter min number for range 2: 3

Enter max number for range 2: 9

The intersection size is 4

>>>

Example 2

>>>

Enter min number for range 1: 5

Enter max number for range 1: 5

Enter min number for range 2: 2

Enter max number for range 2: 8

The intersection size is 0

>>>

Example 3

>>>

Enter min number for range 1: 6

Enter max number for range 1: 6

Enter min number for range 2: 6

Enter max number for range 2: 6

The intersection size is 0

>>>

Example 4

>>>

Enter min number for range 1: 5

Enter max number for range 1: 6

Enter min number for range 2: 40

Enter max number for range 2: 70

The intersection size is -1

>>>

## Submitting your assignment

When you’re done, you can submit your work in Blackboard. In Blackboard, click on “Content” -> “Programming assignments” -> “Assignment 2: Functions”. Submit the following files:

* arithmetic.py
* imperial\_to\_metric.py
* leap\_year.py
* rounding.py
* zodiac.py
* find\_intersection.py