The purpose of this assignment is practice using basic expressions and statements.

Due Date: Sunday, June 13th, 11:59pm

Background

Programs are more interesting when they involve the user – supplying values to use in calculations, asking for names, and so on. We can get a string from the user via the `input()` function, and then convert to other types as needed via other built-in functions of Python, such as `int()`, `float()`, and `str()`.

Variables give us the chance to store values for later, recallable by name. We can even repeatedly update the value associated with that name, as a replacement: forgetting the old value forever, remembering the new value until further notice. As procedural programs are very much a ***sequence*** of executed instructions, the exact order of when we store (or replace) a value for a variable, and when we look up and use the current value, is an important part of understanding how we solve a programming task.

Functions help us write reusable pieces of code that can receive multiple different arguments, saving us from writing the `input()` interaction code with the user (it's also much harder to test/award credit otherwise!). We will be writing different function definitions like in lab, to show our understanding of Python concepts.

This project will get us using variables while calculating different things. We'll explore control structures next, so be sure to master these more basic tasks now.

The current project has two tasks: calculate the distance until a car runs out of gas, and converts a duration in minutes to a more human-friendly time report in the form of (fortnights, days, hours, minutes). There's also some extra credit converting between some bizarre units of measurement! ☺

Refer to our "Project Basics" in its entirety for information on naming your file, performing testing, grading details, and more. Consider it a part of this project's requirements because it is!

- **Project Basics document:** https://mymasonportal.gmu.edu/bbcswebdav/xid-210009785_1
- Template file to help you get started: https://mymasonportal.gmu.edu/bbcswebdav/xid-210011620_1
- Project One tester file: https://mymasonportal.gmu.edu/bbcswebdav/xid-210011621_1

Grading Rubric:

```
Submitted correctly:          5
Code is well commented:      15
Calculations correct:        80
--------------------------------
TOTAL:                      100  + 5 extra credit possible! ☺
```

Tasks

There are two required tasks, and one extra credit (worth 5%). Read the descriptions here, and fill out the template provided.

- `dist_to_empty_km(num_gallons, miles_per_gallon):` This function starts off knowing how many gallons are left, and how many gallons per mile some car gets; you can just use those two *parameters* to calculate. (Assume they're non-negative numbers). Your job is to calculate how far the car can go before it runs out of gas. There's one extra twist: We want to know the distance until empty in kilometers, not miles. For this task, assume that one mile is exactly equal to 1.6 kilometers. here are some sample calls and their expected answers.

```
dist_to_empty_km(0, 5)        →    0
dist_to_empty_km(1,25)        →   40
dist_to_empty_km(2,50)        →  160
dist_to_empty_km(100,0.5)     →   80
```

- `timing(minutes):` This function knows how many minutes there are for a given duration (assume this is a non-negative integer). Your job is to report the time as a total of fortnights, days, hours, and minutes. (There are 14 days in a fortnight.) The answer is returned as a *tuple* of the numbers of fortnights, days, hours, and minutes. This is suggested by the single return statement in the template, `return(num_forts, num_days, num_hours, num_mins)`. As long as you've stored your answers in those four variables it will return the group of them correctly for you. Here are some sample calls and their expected answers.
    - assume `minutes` is a non-negative integer.

```
timing(5)      →    (0,0,0,5)
timing(60)     →    (0,0,1,0)
timing(1439)   →    (0,0,23,59)
timing(1440)   →    (0,1,0,0)
timing(76543)  →    (3,11,3,43)
```

Extra Credit!

- `scruples(depth_feet, length_feet, width_feet):` This function converts from dimensions of a swimming pool (an assumedly large volume) to a very tiny unit of volume (the scruple). Use the following facts to help yourself to calculate how many scruples fit into the described swimming pool! → convert your final answer to an `int`.
    - an acre-foot is one foot deep, one chain wide, and one furlong long.
    - a chain is 66 feet long.
    - a furlong is 10 chains long.
    - there are 325851 gallons in an acre-foot.
    - there are four quarts in a gallon, and 4 cups in a quart.
    - a cup has 8 ounces.
    - 8 fluid drams equals an ounce.
    - there are 60 grains in a fluid dram.
    - there are 20 grains in a scruple.

```
scruples(0,0,0)      →          0
scruples(1,1,1)      →      22980
scruples(1,66,660)   → 1001014272
```