CS 112 – Spring 2022 – Programming Assignment 2

Decision Making

Due Date: Monday Feb. 21st, Noon (12 pm)

This assignment is an Individual Effort assignment.
The Honor Code applies.

The purpose of this assignment is to gain experience using selection statements effectively. See the **Assignment Basics** file for more detailed information about getting assistance, running the test file, grading, commenting, and many other extremely important things. Each assignment is governed by the rules in that document. The **Assignment Basics** file can be found under the "Read Programming Assignment Instructions here" link in Blackboard.

Needed Files:
Download these attached files to use the tests we will use when grading your work.

- **PA2tester.py** (use this file to test your code for PA2)
- **PA2template.py** (use this file to type up your code for PA2)

Notice there is no separate "tests.txt" file for this assignment.

Background:
Selection statements (`if/elif/else` combinations) allow us to write programs that can execute different blocks of code based on the value of given **boolean expressions**. This lets us respond to different inputs from the user, change the behavior of our program for different conditions, and in general, allows our programs to make **decisions**.

For this assignment (and in future assignments) we are going to change how we handle input and output. We will no longer be using the function **input()** or the function **print()** to get input from the user or provide output back. Instead, we will write our own functions, specifying inputs in the form of **parameters** and outputs in the form of **return values**. This allows us to specify the expected inputs and outputs more exactly and avoid many (but not all) rounding and text formatting issues. For this assignment the basic structure of these functions will be provided for you in a template that you will complete. See the section on "**Functions:**" below for more details.

Guidelines:

- Think carefully about the **order** you want to check things in. You might want to write some pseudocode or draw a flowchart to help you organize your approach. Think first, then implement.
- Think carefully about what kinds of selection statements you use and be sure to test your code with many examples. For instance, multiple if statements will not necessarily behave the same as a chain of `if-elif-else`.

- When a specific test case is not working, test the specific function with the specific inputs by hand. You can use Python's interactive mode, or the visualizer.
- You are not allowed to **`import`** anything.
- You are not allowed to use loops, lists, sets, dictionaries, or any feature that hasn't been covered in class yet.
- **You are not allowed to use the function `input()` or the function `print()`.** For this assignment (and future ones) you will be writing **functions** and handling input/output differently. See the note below for more details.

Functions:
In this assignment things are a bit different from the previous one. You won't write a monolithic program as you did in Programming Assignment 1, but you're going to implement several functions. As a reminder, the generic structure of a function is the following:

```
def function_name(arg1, arg2, etc.): #This line is the signature
    # commands go here
    # more commands
    # result = ...
    return result #this line returns the result (the output)
```

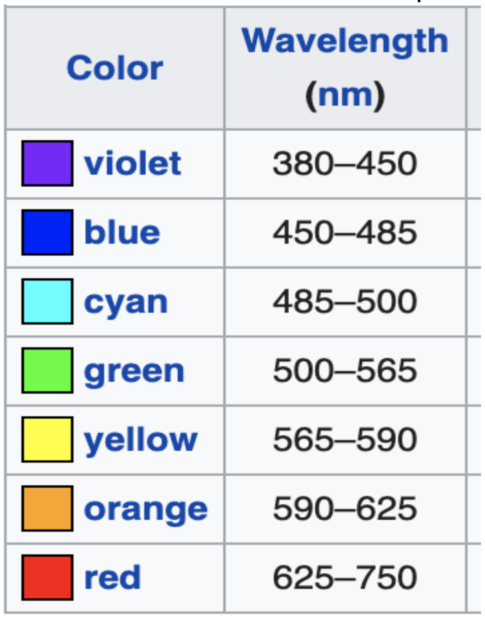
The signature of each function is provided on the next page (and in the template file), and you should ***not*** make any changes to the signature or write a "main body" for your program, otherwise the tester will not work properly. The tester will call and test each function separately.

Task

Complete the following functions so that they perform the described calculations and return the correct output for the inputs that are given. Note that for all the strings described below, capitalization **is** important.

`def spectral_color(wavelength):`

Description: Your friend the physicist is putting together a presentation on the spectrum of light, and they'd like an automated way to color their visualizations by wavelength. You've decided to write a simple function that will convert from wavelength to the common color name as a first step. Consulting Wikipedia, you see the following chart:



Color	Wavelength (nm)
🟪 violet	380–450
🟦 blue	450–485
🟦 cyan	485–500
🟩 green	500–565
🟨 yellow	565–590
🟧 orange	590–625
🟥 red	625–750

Write a function that returns a string with the name of the color associated with a given wavelength of light. Be sure to include `"ultraviolet"` (shorter than violet) and `"infrared"` (longer than red) for wavelengths outside of the ranges given in the chart above. If two ranges overlap, choose the color associated with the range that is lower on the chart (eg: for a wavelength of `500`, the result should be `"green"` not `"cyan"`).

Parameters:

- `wavelength`: a positive number representing the wavelength of light in nanometers

Return value: one of the seven color names given in the above chart as a string, or `"ultraviolet"` or `"infrared"`.

Examples:

```
spectral_color(550)      →     "green"
spectral_color(590)      →     "orange"
spectral_color(999)      →     "infrared"
```

`def bad_broker(price, prev_price):`

Description: Your stockbroker is notoriously bad at giving advice. Rather than pay their commission and fees, you've decided to replace them with a function that should work just as well. By carefully observing what your broker has done in the past, you've determined that they follow very simple rules:

- If the stock is up more than 50%, it's a hot ticket! You should buy more.
- If the stock is down more than 50%, you can't sell now, you'll lose too much money. You should hold the stock and hope things improve.
- If the stock isn't changing very much (up or down less than 50%) then it's boring, and you should sell it.

Write a function which returns a string describing what your broker would tell you to do given the current stock price and the previous stock price.

Parameters:

- `price`: a positive number
- `prev_price`: a positive number

Return value: one of three strings, `"BUY"`, `"SELL"`, or `"HOLD"`

Examples:

```
bad_broker(30,35)         →    "SELL"
bad_broker(200,100)       →    "BUY"
bad_broker(0.99,100.0)    →    "HOLD"
```

`def robot_actions(side_sensor, front_sensor, dirt_sensor):`

Description: You are tasked with writing the logic for a simple robot vacuum. The robot is equipped with three sensors: a sensor that can detect if there is a wall in front of the robot, a sensor that can detect if there is a wall to the left side of the robot, and a sensor that can detect if there is dirt on the ground in front of the robot. Using these sensors, the robot should:

- Turn on the vacuum if the dirt sensor detects dirt
- When there is no dirt detected, move forward as long as there **is a wall to the left side** and **no wall in front.** If there is a wall in front (blocking the robot) the robot should turn
- Stop if there are no walls or dirt detected

Write a function which returns a string describing what the robot should do given what the three sensors are detecting.

Parameters:

- `side_sensor`: A string. can be either `"wall"` or `"clear"`
- `front_sensor`: A string. can be either `"wall"` or `"clear"`
- `dirt_sensor`: A string. can be either `"dirt"` or `"clear"`

Return value: one of four strings, `"stop"` or `"turn"` or `"vacuum"` or `"forward"`

Examples:

```
robot_actions("wall","wall","dirt")       →    "vacuum"
robot_actions("clear","wall","clear")     →    "turn"
robot_actions("clear","wall","dirt")      →    "vacuum"
robot_actions("clear","clear","clear")    →    "stop"
```

Testing:
A template file (**`PA2template.py`**) has been provided which includes the basic outline of the functions you are required to complete. The tester will be calling your functions with different arguments and checking the return values to decide the correctness of your code. Your functions should not ask for user input and should not print anything. When you run your code with the tester or submit your code to **Gradescope**, you can see the results of the tests.

Note: if you want to perform your own tests, you do not need to modify the tester. Simply call the functions you've written providing arguments you want to check.

Submitting your code:
Submit your .py file (correctly renamed) to Gradescope under **Programming Assignment 2**

- Link to Gradescope is on our Blackboard page under "Programming Assignments submit to Gradescope here"
- Do not forget to rename your file **`netID_2xx_PA2.py`**, where `netID` is NOT your G-number but your GMU email ID and `2xx` is your lab section number.
- Do not forget to copy and paste the **honor code statement** from the **Assignment Basics** document at the beginning of your submission .py file.
- Do not forget to comment your code
- **No hard coding!**

Grading Rubric

- **`5pts`** – Correct Submission (file is named correctly)
- **`5pts`** – Well-Documented (code is commented)
- **`40pts`** – Calculations Correct

Total: **`50pts`**

Note: if your code does not run (immediately crashes due to errors), it will receive at most 12 points. No exceptions. As stated on our syllabus, turning in running code is essential.