Due Date: Sunday, June 20th, 11:59pm.

The purpose of this assignment is to gain experience using selection statements effectively.

Background:

Selection statements (if/elif/else combinations) allow us to write code that can execute different statements based on the current values seen in a particular run of the program. We will use this to write a program that performs calculations and selectively reports on different properties of the calculated values.

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
- Project Two tester file: https://mymasonportal.gmu.edu/bbcswebdav/xid-210097764_1
- Project Two template: https://mymasonportal.gmu.edu/bbcswebdav/xid-210097763_1

Notes

- Think carefully about how you will identify ties and placements. Spend the time writing out pseudocode ahead of time, or perhaps a flowchart will work even better for you. Think first, code third. (Think second, too! ☺)
- You need to exactly match the output. We will have a testing file that automatically tries to test your code and give you feedback on your progress.
- Be careful what kinds of selection statements you use, and be sure to test your code with many examples. For instance, lots of if's will not behave *quite* the same as a chain of if-elif-elif's.When it's grading time, your code will get a pretty good testing, so it's best to find the bugs on your own and fix them rather than us finding all the corner cases you didn't consider. Try to turn in perfect code – you can do it!

What are some disallowed things?

- you are **not** allowed to call any kind of min(), max(), or sorting functionality from built-in functions.
- you are not allowed to import anything.

What can I use?

- Numbers, Booleans, strings, variables, and expressions are things we should already understand/can use.
- Conversion functions are okay (`int()`, `float()`, `bool()`, `str()`, etc.).
- Of course we'll need relational operators (`<`, `<=`, `>`, `>=`).
- any control flow (`if`/`elif`/`else`, `while`/`for`, etc.), variables/assignment, others if you've gone way ahead.

Grading Rubric:

```
Code passes shared tests:     90
Well-commented/submitted:     10
-------------------------------
TOTAL:                       100 +5 extra credit!
```

Procedure

We will be sorting out three runners' placements after running a race. Their names will always be Alice, Bob, and Carol. We will compare their running times (given as non-negative integers), and discover/return the times, names, or rankings.

- `def get_times(alice_time, bob_time, carol_time):` Order the three times from fastest to slowest. Return a tuple of all three values, e.g. `return (fastest, middle, slow)` Examples:

```
get_times(4,10,7)   →  (4,7,10)
get_times(9,9,3)    →  (3,9,9)
get_times(6,8,6)    →  (6,6,8)
get_times(10,10,10) →  (10,10,10)
```

- `def get_names(alice_time, bob_time, carol_time):` Order the three names ("Alice", "Bob", and "Carol") fastest to slowest. Whenever there are ties, list the tied runners in alphabetical order. Return a tuple of all three values, e.g. `return (fastest_name, middle_name, slow_name)` Examples:

```
get_names(4,10,7)   →  ("Alice","Carol","Bob")
get_names(9,9,3)    →  ("Carol","Alice","Bob")
get_names(6,8,6)    →  ("Alice","Carol","Bob")
get_names(10,10,10) →  ("Alice","Bob","Carol")
```

- `def get_ranks(alice_time, bob_time, carol_time):` Figure out the placements that will be reported (first, second, third), accounting for ties. When two people tie, they have the same rank (and the next rank isn't seen). Return a tuple of all three integer values, e.g. `return (top_rank, mid_rank, bottom_rank)`. Examples:

```
get_ranks(4,10,7)   →  (1,2,3)
get_ranks(9,9,3)    →  (1,2,2)
get_ranks(6,8,6)    →  (1,1,3)
get_ranks(10,10,10) →  (1,1,1)
```

Extra Credit (+5)

- `def order(alice_time, bob_time, carol_time):` If Alice was the fastest (though perhaps tied for 1st place), the first line will be "Alice is the fastest!" (don't do this for the others).
- If the Bob was *alone* (not tied) in third place, include the line "Way to finish, Bob!" (don't do this for the others).
- describe the top three finishers by name, in order of the fastest time first.
    - Ties must be listed as the same placement – for example, you might have two first places and a third place (notice that we don't use "second place" because the third person does have two people ahead of them).
    - Order of listing: When two people are tied in time, we still describe one per line; who to list first? The alphabetically earlier name. So Alice before Bob before Carol, always for this project.

Here are four example sessions, including both your program's printing as well as the user's responses. Note that the first time we just call order, so we get back a string. To make it clear what the string answers would look like, we immediately call print for the other examples.

```
>>> order(5,10,6)
'Alice is the fastest!\nWay to finish, Bob!\n1. Alice (5 seconds)\n2. Carol (6 seconds)\n3.
Bob (10 seconds)\n'
>>>

>>> print(order(5,10,6))
Alice is the fastest!
Way to finish, Bob!
1. Alice (5 seconds)
2. Carol (6 seconds)
3. Bob (10 seconds)

>>>
>>> print(order(10,8,10))
1. Bob (8 seconds)
2. Alice (10 seconds)
2. Carol (10 seconds)

>>>

>>> print(order(5,5,5))
Alice is the fastest!
1. Alice (5 seconds)
1. Bob (5 seconds)
1. Carol (5 seconds)

>>>
>>> print(order(10,10,8))
1. Carol (8 seconds)
2. Alice (10 seconds)
2. Bob (10 seconds)

>>>
```