CS 112, Lab 2 – Exercise – Conditionals

UPDATED: Due: Sunday, September 20th, 11:59pm
You must be present in your scheduled lab time to get credit.

Files:

- **download** this file, and rename it (with our convention like *gmason76_230_L2.py*):
  http://cs.gmu.edu/~marks/112/labs/templateL2E.py
- You should also **download** this file for testing:
  http://cs.gmu.edu/~marks/112/labs/testerL2E.py
- Grading is based on the fraction of passing tests. You can run it like this:
  `python3 testerL2E.py gmason76_230_L2.py`

As this is an **Exercise**, you can read any and all materials, ask us questions, talk with other students, and learn however you best learn in order to solve the task. Just create your own solution from those experiences, and turn in your work.

Conditionals (if, elif, else blocks) allow us to perform different actions when we receive different inputs to the program. If the user types in different responses, we can cause different lines of code to run or be skipped accordingly. We always make the decision of where to go next by evaluating a boolean expression, and jumping to the correct next place based on it being True or False.

Functions allow us to name a block of code, but they also allow us to write that code so that it defines some parameters, and each call must supply actual values for each parameter in order to complete the function call. Furthermore, a function call must always return a value. You can replace any and all of a function's body in our lab exercise if you'd like, just keep the function's name and parameters the same.

Notice that when we write functions, we almost never perform user interactions directly (no input(), no print()). The goal is to push as much user interaction out of our many function definitions, so that they are reusable in many places. User interaction will tend to be quarantined into one small tiny part of our program, or else we might just be writing pieces of code function by function, so that we *could* build a larger program out of those pieces, even if we don't actually do so for time's sake.

Notes

- We are going to be filling in function definitions for the first time. Because functions have a formal way to accept incoming values and send back a returned value, we won't be calling `input()` or `print()` anywhere in this assignment.
- We can finally use proper "unit testing", so there is just a single testing file that contains many specific calls of our code. It will give useful feedback when you're not yet passing test cases; you

should not be modifying the testing file, but it can be very useful to look at the specific test cases that fail, to see exactly how it calls your code, what value it expected, and how yours went wrong.

Task 0 - Example

This is our first time writing function definitions (which you learned about by completing your Zyante Chapter 4 reading). We provide one entire function definition as an example; it even has a comment indicating where the student would have written their code (and it includes a solution). Just take a look at it to get a feel for how writing functions works.

- `def is_even(n):` Determine if `n` is even or not. `return True` when it is even, and `return False` when it is not.
  - assume that `n` is an integer. It might be negative, zero, or positive, and your code needs to correctly answer if it's even (divisible by two) with a boolean value.
- Examples:
  - `is_even(5)` → `False`
  - `is_even(102)` → `True`
  - `is_even(-3)` → `False`
  - `is_even(0)` → `True`

Task 1 – Letter Grade

Given an integer `score`, we determine their letter grade and return the string. Possible return values are `"A+"`, `"A"`, `"A-"`, `"B+"`, `"B"`, etc. It will exactly match our own semester's grading scale, so look at the grading scale in our syllabus.

- `def letter_grade(score):` Given a non-negative integer `score`, determine what the actual grade should be based on our semester grading scale; return a string directly representing that grade.
  - score will be a non-negative integer.
- Examples:
  - `letter_grade(95)` → `"A"`
  - `letter_grade(89)` → `"B+"`
  - `letter_grade(61)` → `"D"`

Task 2 – Selectively Using Arguments

This time, our function to implement accepts three arguments. But we want to only find the sum of the two largest values of the three.

- `def sum2biggest(a,b,c):` Given three integer arguments, determine what the two biggest ones are, add them together, and return that result.
  - all three parameters `a`, `b`, and `c` will be integers; note that they can be positive, zero, or negative!
- Examples:
  - `sum2biggest(5,3,4)` → `9`
  - `sum2biggest(-5, -3, -1)` → `-4`
  - `sum2biggest(6,4,6)` → `12`

- `sum2biggest(0,0,0)` → `0`

Turning It In

Add a comment at the top of the file that indicates your name, userID, G#, lab section, a description of your collaboration partners, as well as any other details you feel like sharing. Please also mention what was most helpful for you. Once you are done, run the testing script once more to make sure you didn't break things while adding these comments. If all is well, go ahead and turn in just your one .py file you've been working on over on BlackBoard to the correct lab assignment. We have our own copy of the testing file that we'll use, so please don't turn that in (or any other extra files), as it will just slow us down.