Lab 2: Conditionals

Labs are graded for participation rather than correctness. Keep all your lab code in your course GitHub repo to receive credit for your work. We'll be looking to see that you have at least partially completed all problems in each lab.

If you finish the lab assignment early, you may get started on the homework.

Getting started

In your local repo*, create a folder for Lab 2. Create a new .py file for this week's problem and save it in the Lab 2 folder.

*Not sure what this means? Follow the GitHub instructions on Canvas. If you're still not sure, check with an instructor or TA.

Documentation and format guidelines

Every file you create for this course should have a file comment.
Programs should all start with `main()`

NEW: Use constants in place of magic numbers and magic strings.
A constant is a variable whose value will never change. For example, if we wanted to represent the number of months in the year, we would use a constant because that number will always be 12.

In Python, constant names should be in ALL_CAPS with underscores between words. Like variables, constant names should be meaningful e.g. `MONTHS_IN_YEAR = 12`.

A magic number is a literal numeric value that is used in calculations. For example, in the following code, the number 12 is a magic number:

```
annual_salary = float(input("Enter your annual salary: "))
monthly_pay = annual_salary / 12
```

Magic numbers are considered bad practice because they can make your code less readable. From now on, do not use magic numbers in your code. Instead, use constants to represent those values. For example:

```
MONTHS_IN_YEAR = 12
annual_salary = float(input("Enter your annual salary: "))
monthly_pay = annual_salary / MONTHS_IN_YEAR
```

There are exceptions to the magic number rule: the numbers 0, 1, and 2. In general, the meaning of these numbers in a calculation is pretty clear.

String literals should also be represented as constants if the same string will be used more than once. This helps reduce the risk of bugs caused by typos in your strings.

General tips

Test every program you write by running it several times, providing different inputs each time and verifying the output.

The problem: Writing a Buzzfeed-style quiz

Have you ever taken one of those online quizzes that tells you, say, what TV character you are? Or what food you would be if you were a food? E.g. https://www.buzzfeed.com/jesseszewczyk/egg-personality-quiz?origin=nofil. Today you're going to write your own quiz.

We've provided an example quiz below, but yours doesn't have to be exactly like ours.

Your program needs to meet the following requirements:

1. Ask at least three questions.
2. Each question must have at least two possible answers.
3. After the user has answered all the questions, tell them the result of the quiz.

You also must check the user's input. We expect them to enter a string (for example, in my quiz I prompt them to enter *A, B*, or *C*). If they enter something invalid or unexpected, then just assume they entered "A".

- For example: You ask "Where do you live? A -- Cupboard under the stairs. B -- Burrow. C -- Hogwarts castle."
- User types "Hogwarts"
- Your program notes that this is not one of our three valid answers. Change the answer to *A.*

Finally, allow the user to enter upper- or lower-case letters, and accept either as valid answers. You will find Python's string methods `upper()` and `lower()` helpful.

```
hello = "Hello"
uppercase_hello = hello.upper()
print(uppercase_hello) #  prints HELLO
lowercase_hello = uppercase_hello.lower()
print(lowercase_hello) # prints hello
```

```
print(hello) # prints "Hello"
```

If you look carefully at the example above, you'll notice that when **`hello`** is printed, its value has not changed. That's because these methods *do not modify* the string; you'll have to save the results to a variable.

Warning! These question/answer combinations multiply fast. If you have 3 questions with 3 possible answers, that's 27 possible answer combinations. You can combine them to make things easier (i.e., you don't need to have 27 total possible outputs -- check out our example below).

Once you're confident your quiz is working, check to see if it's as efficient as it can be and refactor if necessary. If multiple branches in your program have the same answer, you can refactor to remove some branches.

Example -- Which Harry Potter character are you?
Note that we stole these questions from a Buzzfeed quiz. Feel free to do some googling for yours, too.

Our Harry Potter quiz asks three questions, each with three possible answers:

When planning a trip, you...
A: Find the hot parties.
B: Sorts out all the logistics
C: Lets everyone else take charge

What are you most afraid of?
A: Not being accepted
B: Losing someone close to me
C: Looking stupid in front of others

What was your favorite toy as a kid?
A: She-Ra
B: He-Man
C: Video games

We calculate the user's character as follows:

- A A A: Ginny
- A A B: Draco
- A A C: Sirius
- A B *x* (ABA, ABB, ABC): Dobby
- A C *x*: Voldemort
- B *x x* : Hermione

- C A A: Luna
- C A B: Hagrid
- C A C: Ron
- C B *x*: Tonks
- C C *x*: Slughorn