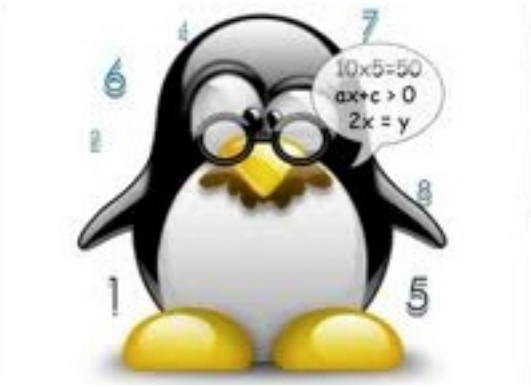
CS150 - Assignment 4
Math Whiz
Due: Wednesday Oct. 10, at the beginning of class



For this assignment, you are going to be implementing a timed game where the user answers as many simple math questions as possible in a given time period.

1 Generating random equations

Before you start working, make sure that you understand at a high-level how the game is supposed to be played. If you want to see a working program in action, see the lap prep for this week to see how to run a working version. We're going to build up the final version of this program one piece at a time. As you build each piece, make sure you understand where it fits into the larger program and make sure it's working correctly before moving on.

The first thing we'll need is the ability to generate a random equation. Our random equations will contain `+`, `-` and `*` for operators. The following pseudocode[1] describes one way to generate a random equation:

[1] Pseudocode is a way to describe an algorithm in detail without getting bogged down in language specifics. It is a compromise between English and code.

* generate a random number between 1 and 10 to start the equation
* for each operator that you want to add on:
  - pick a random operator from `+, -, *` and add it to the equation
  - pick a random number between 1 and 10 and add it to the equation

Using this approach, we can generate a random equation with as many operators as we would like.

- Write a function called `random_equation` that takes as input a single parameter, the number of operators to generate in the random equation, and returns a *string* representing a random math equation involving the numbers 1-10 and operators `+, -, *`. Here are a few example runs:

```
>>> random_equation(4)
'5 - 6 - 7 * 6 * 4'
>>> random_equation(1)
'3 * 7'
>>> random_equation(2)
'8 - 5 * 2'
>>> random_equation(7)
'8 + 7 + 4 - 7 - 2 * 3 * 2 * 4'
```

Hints: You'll likely need to use some sort of loop structure to get the repetition (i.e. `for` or `while`). Think about which one is more appropriate here. Also, notice that our equation is a string, so we'll be building up a string similar to how we did in the last assignment by appending on new pieces.

2 Getting an answer

Now that we have an equation, we need to have some way of repeatedly prompting the user for an answer until that answer is correct. **Before writing any more code, read through this whole section since I give some hints/advice at the end.**

There is a function called `eval` built in to Python that evaluates any expression represented as a string and returns the value represented by that expression. We're going to use this function to figure out what the answer is to our random equation. For example:

```
>>> eq = random_equation(4)
>>> eq
'7 * 5 - 2 - 4 * 8'
>>> eval(eq)
1
>>> x = eval(eq)
>>> x
1
```

Notice that the value that `eval` returns is an `int`, which we would expect. If you're curious, `eval` can be used to evaluate more complicated string expressions that include function calls, etc.

- Write a function called `query_equation` that takes as a parameter a string representing an equation (e.g. what is returned from your `random_equation` function). This function should prompt the user with the equation and then wait for an answer from the user. If the user gets it wrong, then it will output a message to the user indicating this and then prompt the user again with the equation (`eval` will be useful in figuring out if the user's answer is correct). The function should continue to prompt the user until they get the equation right. When the user does finally get it right, the function should print "Correct". The message the user gets if they answer incorrectly should depend on how close there are:
  - If the user is within 10% of the correct answer, then you should print "Close. Try again."
  - If the user is further off than this, then print "Keep trying!" (or something similar, free to get creative with these).

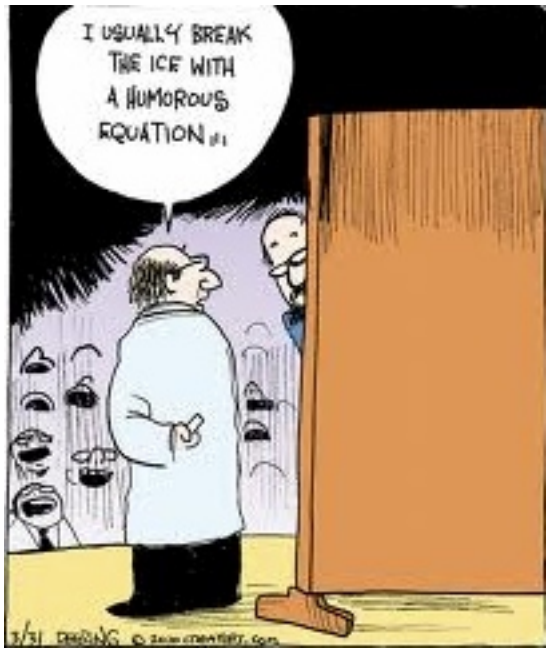
There are many ways you can tackle this function, but I suggest an incremental approach:

1. Write the function without a loop so that it only queries the user once and then either prints out "Correct" or "Incorrect" depending on whether or not the user got it right.
2. Add some sort of loop to repeatedly query the user when incorrect.
3. Add in "Close. Try again.", that is output when the user gets it wrong depending but is within 10% of the correct answer.

Here is a quick example with `query_equation`:

```
>>> eq = random_equation(2)
>>> eq
'5 + 9 + 7'
>>> query_equation(eq)
5 + 9 + 7 = 25
Keep trying!
5 + 9 + 7 = 22
Close. Try again.
5 + 9 + 7 = 20
Close. Try again.
5 + 9 + 7 = 21
Correct
```

3 Playing the game



We now have all of the pieces we need to put together our final program.

- Write a function called `play_game` that takes a single parameter, the game duration in seconds. The function should use the `time` function in the `time` module to time the user. As long as the elapsed time hasn't gone over the input game duration, you should present the user with a random equation and then keep track of how many they get correct (you should be thinking about some kind of loop structure). When time runs out, you should print out how many the user got correct and how long the game was played for.

This function should NOT be a lot of code, but should utilize the two previous functions that you've written. The key responsibility of this function is timing and keeping track of how many it got correct. Recall from the lab prep that we can measure how much time has elapsed by first recording a starting time and then measuring the difference between the current time returned by `time()` and the initial start time.

As with the previous function, I would suggest an incremental approach. There are many ways you could tackle this, but one would be:

1. Start by getting the timing loop working. For example, just get any input from the user (using `raw_input`) and then print it out. Do this over and over again until the time has elapsed.
2. Change your loop now to generate a random equation and then use that equation to query the user.

3. Finally, add in the bookkeeping part where you keep track of how many the user got right and then at the end, print out the game summary.

NOTE: because we are doing the timing outside of the `query_equation` function, time will NOT expire until the user gets the answer right. This is fine and how I expect yours to work.

4 The finishing touches

When your `play_game` function is working, you can finish things up:

- Complete your program by adding some statements at the end of your file to play the game automatically when your program is run. You should ask the user if they want to play a game. If they say "yes", then you should prompt the user to see how long they want to play for (in seconds) and then the game should start. If the user does not say "yes", just give the user a nice goodbye message. All of this code should be outside of your `play_game` function.

5 Extra Credit

You may earn up to 2 points of extra credit on this assignment by adding improvements to your program. If you do, include in your comments at the top of your program what you added. Below are some suggestions, but feel free to add you own:

- (0.5 points) Add a prompt for difficulty level and change the equation difficulty accordingly. Also, include the level in the game summary.
- (1 point) Add parenthesis to the equations (this can be tricky...).
- (0.5 points) Improve the user experience during guessing by adding more feedback.
- (? points) Add your own ideas. Points will be awarded based on difficulty and innovativeness.

6 When you're done

Make sure that your program is properly commented:

- You should have comments at the very beginning of the file stating your name, course (including section number), assignment number and the date.
- Each function should have an appropriate *docstring*
- Other miscellaneous comments to make things clear

In addition, make sure that you've used good ***style***.

What to hand in:

You should have implemented:

- `random_equation`
- `query_equation`
- `play_game`
- Additional code at the end to initiate the game, etc.

Submission procedure

Submit your .py file online using the digital submission link on the course web page. You must have submitted it online before the beginning of class on Wed.

Grading

		points
`random_equation`		
	correctly formed equations	3
	all 3 operators randomly	2
	number varies based on input	1
`query_equation`		
	print correct/incorrect correctly	2
	loops until correct	3
	close answers get different feedback	1
`play_game`		
	generates random equation and queries user	1
	loops until time expired	3
	prints out game summary	1
Final gaming code		
	correctly handles user input for game starting	1
	gets game duration	1
Lab prep		3
Comments, style		3
extra credit		2
total		25 (+2)