CSC 201: Lab 7 – Arithmetic Quiz with Progress Bar

Concepts: Object-Oriented Programming, writing a custom **class** to represent data and using another custom class for a graphical widget

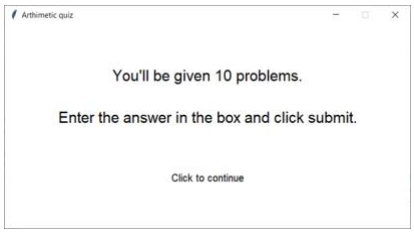
Part 1: Getting Started

- Download the Lab7 zip file. Unzip/extract the zip file into a **Lab7** folder. You should have 5 files: graphics2.py, button.py, progress_bar.py, problem.py, and quiz.py.

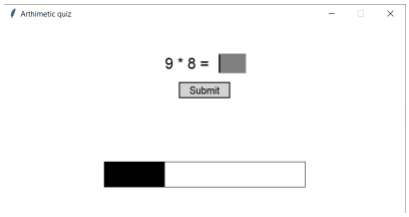
Writing a class in Python gives us a new data type which bundles together data and behaviors for objects created of that type. In this lab, you will define one class, and then add code to a "client" for another class whose definition is written for you.

Preview: *after* you complete lab parts 2-4, this program will quiz the user on randomly generated arithmetic problems. The user will enter an answer to each problem and the quiz will tally up the number of problems answered correctly. The program also displays a progress bar so that the user can see how much of the quiz they have completed. Here are three screen shots showing the first screen of the quiz, a screen during the quiz, and the ending screen.

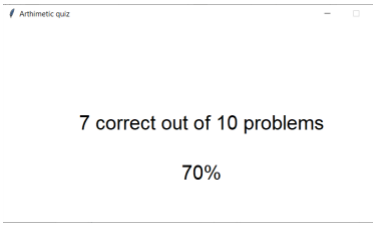
Start



During



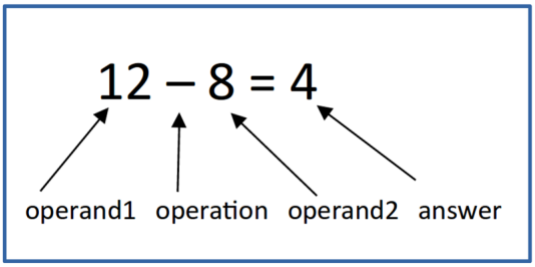
End



Part 2: QuizProblem Class in problem.py

The QuizProblem class represents an arithmetic problem for our quiz. Since our target audience is young children, the numbers for each type of problem will be randomly generated so that the problem will always have a non-negative integer answer (0, 1, 2, …)

- Open problem.py. Add your name(s) to the top comment.
- Define the QuizProblem class using these specifications:



The QuizProblem class should have:

- 1 constructor (__init__)
- 4 private helper methods
- 2 public accessor methods,
- and a special **__str__** method. *(Recall that a "**method**" is simply a **function** that's enclosed within a **class**.)*

The QuizProblem class has four instance variables.

One is defined in __init__ method:

- **operation** (str): the symbol for the operation ('+','-','*', or '/')

The following three are defined and used in other methods:

- **operand1** (int): the number left of the operation symbol
- **operand2** (int): the number right of the operation symbol
- **answer** (int) : the answer to the problem

2.1 Write constructor (__init__)

The constructor (**__init__**) should take in ONE parameter (besides **self**, of course!) named **maxRandom**, which should specify the largest allowable integer for generating random numbers for the problem.
*Example: when maxRandom is 10, then the two random numbers being added together in the addition problem should each be between 0 and **10 (inclusive**), although the answer can obviously be larger.*

The __init__ method should choose a random operation from the list ['+', '-', '*', '/'] (ie. QuizProblem.OPERATION_LIST), and assign that random operation to the **operation** instance variable. Remember you must refer to it as **self.operation**. If you just say **operation = …**, you'd be creating a new local variable named operation, NOT changing the operation instance variable.

Next, in the body of the __init__ method, you should call the appropriate private method (from the 4 below) to generate valid random numbers for the operands and answer based on that operation. (The **maxRandom** parameter must be passed into each of the private helper functions, since they will use that value when doing the actual random number generation). Remember that you must put **self.** in front of your method (function) call.

2.2 Write 4 private mutator methods

Write four "private" helper methods (because they "help" another method, __init__, do its job)
_randomizeAddition(...), _randomizeSubtraction(...), _randomizeMultiplication(...), _randomizeDivision(...)
These should each take **maxRandom** as a parameter and use it when generating numbers.
These "private" helper methods don't return any value – their purpose is to modify the instance variables, and they are inaccessible outside the class.

- **operand1**, **operand2**, and **answer** are instance variables you will define and assign values to within EACH of these 4 mutator methods
- For addition and multiplication, **operand1** and **operand2** should be assigned random whole number values 0 through **maxRandom** (inclusive), then **answer** can be assigned the correct result based on those two numbers.
- For subtraction, to avoid getting a negative number for an answer, assign random values to **operand2** and **answer**, then make **operand1** be the sum of **answer** and **operand2**.
- For division, similarly, work backward from the **answer** to guarantee a whole number answer, but also make sure **operand2** can't be zero!

2.3 Write 2 public accessor methods

Write two accessor methods (no parameters other than *self*):

- **getAnswer(...)** simply returns the **answer** instance variable (self.answer). Easy!

This function will be needed by our quiz.py client program to check if the user's answer is correct.

- **getQuestionString(...)** returns a string of text like "<operand1> <operation> <operand2> =" (e.g. "12 - 8 ="). This function will be useful in our quiz.py client program, because we want to display the arithmetic question to the user without showing them the answer. *Tip: use an f-string, inserting* **{self.<instancevar>}** *as needed.*

2.4 Write __str__(…)

The special **__str__(…)** function should return a full string representation of the problem's equation, with format "2 + 5 = 7" (but fill in the appropriate instance variable values for 2, 5, 7, and '+').

2.5 Test your QuizProblem Class

- Save and execute problem.py. The main() function creates and prints information about one QuizProblem with random values 0 or 1, then 25 more random problems using 0 through 10. Look at the 25 problems to make sure that the numbers/equations meet the criteria described. (No negative answers. No decimal answers. No division by 0. At least two of the numbers in each problem will always be in the range 0 through 10.)

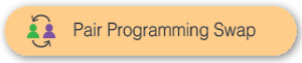
```
Question:  1 - 1 =
Answer:  0
Equation: 1 - 1 = 0

0 * 4 = 0
8 / 4 = 2
6 - 3 = 3
12 - 5 = 7
6 / 6 = 1
10 + 2 = 12
2 * 4 = 8
4 + 10 = 14
15 - 10 = 5
14 / 2 = 7
…
```

You should have 25 random, but *mathematically correct* problems.

Concept/Vocab Note: Like the **str** (string) or **int** (integer) classes, the **QuizProblem** class you just created is **"immutable"**, since there are no (public) methods that change its value after it gets constructed.

Part 3: ProgressBar Class in progress_bar.py



ProgressBar class is complete. You only need to learn how to use it then write code in main create and work with progress bar objects!

This class represents a graphical "progress bar" which displays the percentage of a task that has been completed.

ProgressBar has four methods **(which have been finished for you!):**

ProgressBar(topLeft, width, height, color) constructs a ProgressBar object so that topLeft (Point) is the top left corner of the ProgressBar object, width (float) is the horizontal dimension, height (float) is the vertical dimension, color (str) is the color for the growing fill which by default will be black.

Example: myProgress = ProgressBar(Point(50, 100), 400, 30, 'red')

draw(<GraphWin>) which draws the progress bar in the window

undraw() which undraws the progress bar in the window

update(<GraphWin>, <newPercent>) which updates the ProgressBar object to display the newPercent received as a parameter.

Example: myProgress.update(window, 40)

You will add some code in the main method of the ProgressBar Class to demonstrate that you understand how to use the ProgressBar class.

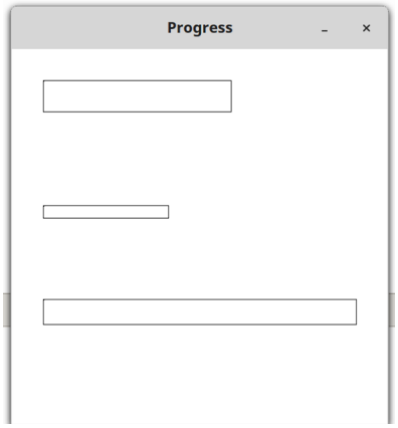
- Create three ProgressBar objects as described and draw them in the window.
  A progress bar with top left corner (50, 100), width of 300, height of 50, and color of 'red'
  A progress bar with top left corner (50, 250), width of 200, height of 20, and color of 'yellow'
  A progress bar with top left corner (50, 400), width of 500, height of 40, and color of 'blue'

- sleep for 2 seconds

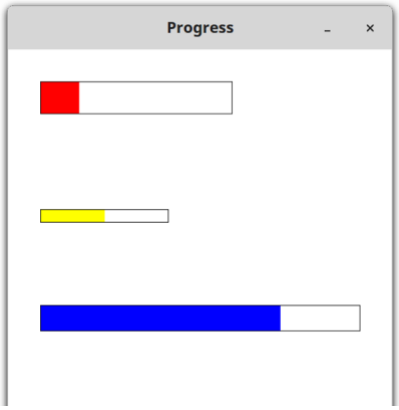
- update the first progress bar to 20%, the second progress bar to 50%, and the third progress bar to 75%

- sleep for 2 more seconds.

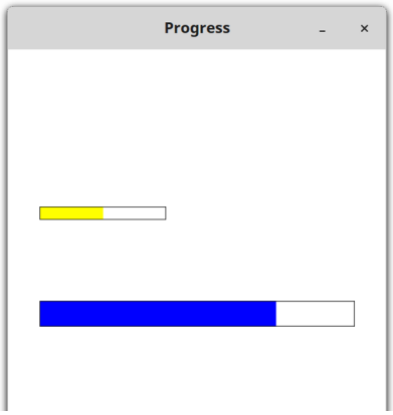
- undraw the first progress bar
- execute progress_bar.py and compare your output to the windows below.



initial window



after update

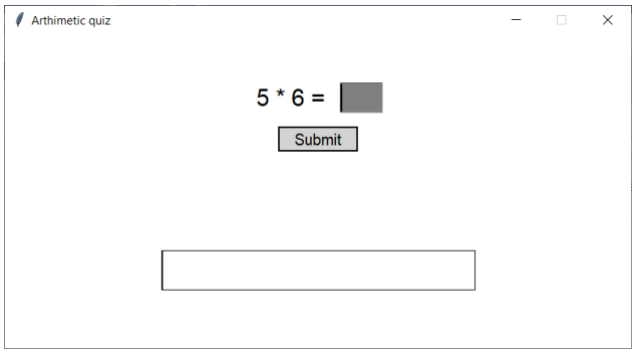


after undraw

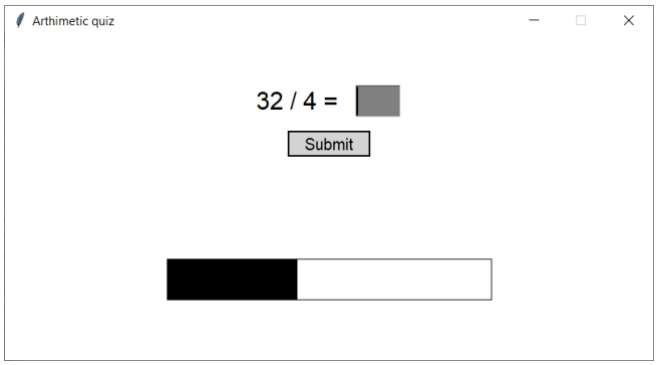
Part 4: Client Code—quiz.py

Pair Programming Swap

- Open quiz.py. Add your name(s) to the top.
- You only need to work on the main() function. ALL other functions are complete.
- Execute quiz.py which uses your QuizProblem class and a Button class (defined in button.py and is complete) to produce a basic arithmetic quiz. The user is given only one chance to answer each problem, unless the user enters a non-digit answer or non-positive answer (like 'a' or 5.4 or -8) in which case the user is told to try again. The user must click on the submit button to progress to the next problem.
- You only need to work on the main method by following the instructions in the code comment.
- You will add a ProgressBar object to the quiz so that the user can gauge how much more of the quiz they still need to do.
- Add code to function main to create and draw a ProgressBar object once the quiz starts.



- As each problem is completed, the progress bar should be updated to show the percentage of the quiz completed. The percentage needs to be computed so that it can be passed to the update method of the ProgressBar object. The image below is after 4 of 10 problems had been completed which means 40 would have been passed to the update method.



- Finally, when the submit button is removed from the window, the ProgressBar object should also be removed.
- Change NUM_PROBLEMS to other values to check that your ProgressBar is updating correctly especially when the number of problems doesn't divide "evenly" into 100. (Try 3 problems, 7 problems, 13 problems,….)

Part 5: Upload your work!

- Update the partner "completion comment" and assistance section at the top of each file.
- Upload **progress_bar.py**, **problem.py**, and **quiz.py** to Moodle.