A reminder about the Honor Code

For previous mini-projects, we have had instances of students submitting solutions that have been copied from the web. Remember, if you can find code on the web for one of the mini-projects, we can also find that code. Submitting copied code violates the Honor Code for this class as well as Coursera's Terms of Service. Please write your own code and refrain from copying. If, during peer evaluation, you suspect a submitted mini-project includes copied code, please evaluate as usual and email the assignment details to *iipphonorcode@online.rice.edu*. We will investigate and handle as appropriate.

Mini-project description - "Stopwatch: The Game"

Our mini-project for this week will focus on combining text drawing in the canvas with timers to build a simple digital stopwatch that keeps track of the time in tenths of a second. The stopwatch should contain "Start", "Stop" and "Reset" buttons. To help guide you through this project, we suggest that you download the provided **program template** for this mini-project and build your stopwatch program as follows:

Mini-project development process

1. Construct a timer with an associated interval of 0.1 seconds whose event handler increments a global integer. (Remember that `create_timer` takes the interval specified in milliseconds.) This integer will keep track of the time in tenths of seconds. Test your timer by printing this global integer to the console. Use the CodeSkulptor reset button in the blue menu bar to terminate your program and stop the timer and its print statements. **Important:** Do not use floating point numbers to keep track of tenths of a second! While it's certainly possible to get it working, the imprecision of floating point can make your life miserable. Use an integer instead, i.e., 12 represents 1.2 seconds.
2. Write the event handler function for the canvas that draws the current time (simply as an integer, you should not worry about formatting it yet) in the middle of the canvas. Remember that you will need to convert the current time into a string using `str` before drawing it.
3. Add "Start" and "Stop" buttons whose event handlers start and stop the timer. Next, add a "Reset" button that stops the timer and reset the current time to zero. The stopwatch should be stopped when the frame opens.
4. Next, write a helper function `format(t)` that returns a string of the form `A:BC.D` where `A`, `C` and `D` are digits in the range 0-9 and `B` is in the range 0-5. Test this function independent of your project using this testing template http://www.codeskulptor.org/#examples-format_template.py. (Just cut and paste your

definition of `format` into the template.) Note that the string returned by your helper function `format` should always correctly include leading zeros. For example: `format(0)` = `0:00.0`, `format(11)` = `0:01.1`, `format(321)` = `0:32.1`, and `format(613)` = `1:01.3`.

5. Insert a call to the `format` function into your draw handler to complete the stopwatch. (Note that the stopwatch need only work correctly up to 10 minutes, beyond that its behavior is your choice.)
6. Finally, to turn your stopwatch into a test of reflexes, add to two numerical counters that keep track of the number of times that you have stopped the watch and how many times you manage to stop the watch on a whole second (1.0, 2.0, 3.0, etc.). These counters should be drawn in the upper right-hand part of the stopwatch canvas in the form `"x/y"` where `x` is the number of successful stops and `y` is number of total stops. My best effort at this simple game is around a 25% success rate.
7. Add code to ensure that hitting the "Stop" button when the timer is already stopped does not change your score. We suggest that you add a global Boolean variable that is `True` when the stopwatch is running and `False` when the stopwatch is stopped. You can then use this value to determine whether to update the score when the "Stop" button is pressed.
8. Modify "Reset" so as to set these counters back to zero when clicked.

Steps 1-3 and 5-7 above are relatively straightforward. However, step 4 requires some adept use of integer division and modular arithmetic. So, we again emphasize that you build and debug the helper function `format(t)` separately following the tips in the Code Clinic Tips page. Following this process will save you time. For an example of a full implementation, we suggest that you watch the video lecture on this mini-project.

Grading Rubric - 13 pts total (scaled to 100 pts)

- 1 pt - The program successfully opens a frame with the stopwatch stopped.
- 1 pt - The program has a working "Start" button that starts the timer.
- 1 pt - The program has a working "Stop" button that stops the timer.
- 1 pt - The program has a working "Reset" button that stops the timer (if running) and resets the timer to 0.
- 4 pts - The time is formatted according to the description in step 4 above. Award partial credit corresponding to 1 pt per correct digit. For example, a version that just draw tenths of seconds as a whole number should receive 1 pt. A version that draws the time with a correctly placed decimal point (but no leading zeros) only should receive 2 pts. A version that draws minutes, seconds and tenths of seconds but fails to always allocate two digits to seconds should receive 3 pts.
- 2 pts - The program correctly draws the number of successful stops at a whole second versus the total number of stops. Give one point for each number displayed. If the score is correctly reported as a percentage instead, give only one point.
- 2 pts - The "Stop" button correctly updates these success/attempts numbers. Give only one point if hitting the "Stop" button changes these numbers when the timer is already stopped.

- 1 pt - The "Reset" button clears the success/attempts numbers.

✓ Terminer