Homework 3

Introductory Programming **Allen B. Downey**
Fall 2004

The reading for this assignment is Chapters 2 and 3 of *How to think...*.

3.1 Turtles

1. If you have not already downloaded `World.py`, move into the directory where you keep Python programs and type

   ```
   wget http://wb/ip/code/World.py
   ```

2. Run the program by typing `python World.py`

3. Press the Make Turtle button. A turtle should appear on the canvas and a Turtle Control panel should appear in the control panel. Push the buttons in the Turtle Control panel to move the turtle around, raise and lower the pen, and change the color of the turtle. Here are the names of the buttons and what they do:

   ```
   fd      Move forward.
   lt      Left turn.         rt     Right turn.
   pu      Lift the pen.      pd     Lower the pen.
   ```

   There is a text entry next to `fd` that controls how far the turtle moves when the `fd` button is pushed. This value is an **argument** for the `fd` function. Change the argument and press `fd` again.

4. Press the Run this code button. It executes the code in the text field, which should include the statements `world.clear()` and `bob = Turtle(world)`. The first statement erases all the turtles; the second line creates a new turtle named `bob`.

   Notice that there is no Turtle Control panel for `bob`. In order to control `bob`, you have to write a program.

5. Add a line of code to the program in the text window. To move the turtle, try something like `fd(bob, 90)`. The function `fd` takes two arguments, the name of a turtle and the distance you want the turtle to move. The other turtle control functions are:

   ```
   bk(turtle, distance)          # move a turtle backward
   lt(turtle)                    # turn left
   rt(turtle)                    # turn right
   pu(turtle)                    # pen up
   pd(turtle)                    # pen down
   ```

6. Make a few errors. If the code you type in the text field contains an error, you should get an error message in the window you used to run the program. The message contains information about what was happening when the error occurred, most of which won't make sense to you. For example, if you spell `world` wrong, you'll get something like this:

   ```
   Exception in Tkinter callback
   Traceback (most recent call last):
     File "/usr/lib/python2.2/lib-tk/Tkinter.py", line 1300, in __call__
       return apply(self.func, args)
     File "World.py", line 67, in run_text
       self.inter.run_code(source, '<user-provided code>')
     File "World.py", line 470, in run_code
       exec code in self.globals, self.locals
     File "<user-provided code>", line 2, in ?
   NameError: name 'worl' is not defined
   ```

   The last two lines are probably the most useful. The error occurred in line 2 of the "user-provided code" (that means you). It was a NameError; specifically, the name `worl` is not defined.

   In general, error messages are a mixed blessing. They often contain information that helps you identify the problem, but they also contain information that is extraneous at best and misleading at worst. When you are starting out, it is a good idea to make errors on purpose so that you can see what the messages look like. Try some of the following:

   (a) Leave out one of the parentheses in the function call.

   (b) Put a semi-colon at the end of a line `:)`

   (c) Change `Turtle` to `turtle` or `bob` to `Bob`. Yup, Python is case-sensitive.

7. You can define functions in the text field, too. At this point you might want to quit the program and run it again, so we start fresh.

   Now add the following lines to the text field (after the existing lines):

   ```
   def fdlt(turtle, n):
       fd(turtle, n)
       lt(turtle)
   ```

   Read the program carefully and make sure you understand what it does. Now run it. Did you get what you expected? Reminder: the `def` statement only creates a new function; it doesn't execute it. To run the new function, you have to call (or invoke) it.

8. Add a line of code that invokes `fdlt`, passing `bob` as the first argument and a pleasant distance like 90 as the second. Run the program. What happens if the function call comes before the `def` statement?

9. The text field is convenient for typing and running a few lines of code, but it has the annoying property of vaporizing your code when the program quits. For longer (and longer-lived) programs, it would be better to put the code in a file.

Use emacs to create a file named `turtle_code.py`. Copy and paste the code from the text field into the file. Save the file and then press the `Run file` button. It should execute your code.

10. In `turtle_code.py`, add a function called `ell` that takes a turtle and a distance as parameters and draws an ell-shape by invoking `fdlt` twice. Add a line of code that invokes `ell` (and remove the old line that invoked `fdlt`). Save the modified version of `turtle_code.py` and run it (you don't have to restart `World.py`).

11. Add a function called `square` that draws a square by invoking `ell` twice. Test your function.

12. At this point you should be comfortable editing your program and running it using `World.py`, and you should be comfortable defining and calling functions. If either of those things is not true, now is the time to get help before you attempt the rest of the homework.

13. Modify `turtle_code.py` so that it writes the word "Hello" on the canvas. You can write the letters in any style you like; more important than the style of the letters is the style of the code! A good solution to this problem should define and use functions that are well-named, demonstrably correct, appropriately general, and reusable. Ideally, your solution should be flexible, so that the size of the letters can be controlled by a parameter.

    For each function you write, add a comment that explains concisely what the function does. An important piece of information to document is where the turtle ends up at the end of the function.

    Hints:

    - You might find it useful to use more than one turtle, but you don't have to.
    - By default, turtles wait 0.5 seconds between moves. You can speed up a turtle, or slow it down, by setting its `delay` attribute:

      ```
      bob = Turtle(world)
      bob.delay = 0.01          # a very fast turtle
      ```

    - You might want to start by writing long, repetitive code, and then look for recurring idioms that would make good functions.
    - There is a tradeoff between writing lots of special-purpose functions and writing lots of repetitive code. Your goal should be to find a balance that yields reasonably concise, readable code. To evaluate your code, think about how you would handle the other letters of the alphabet. Do your functions lend themselves to reuse?