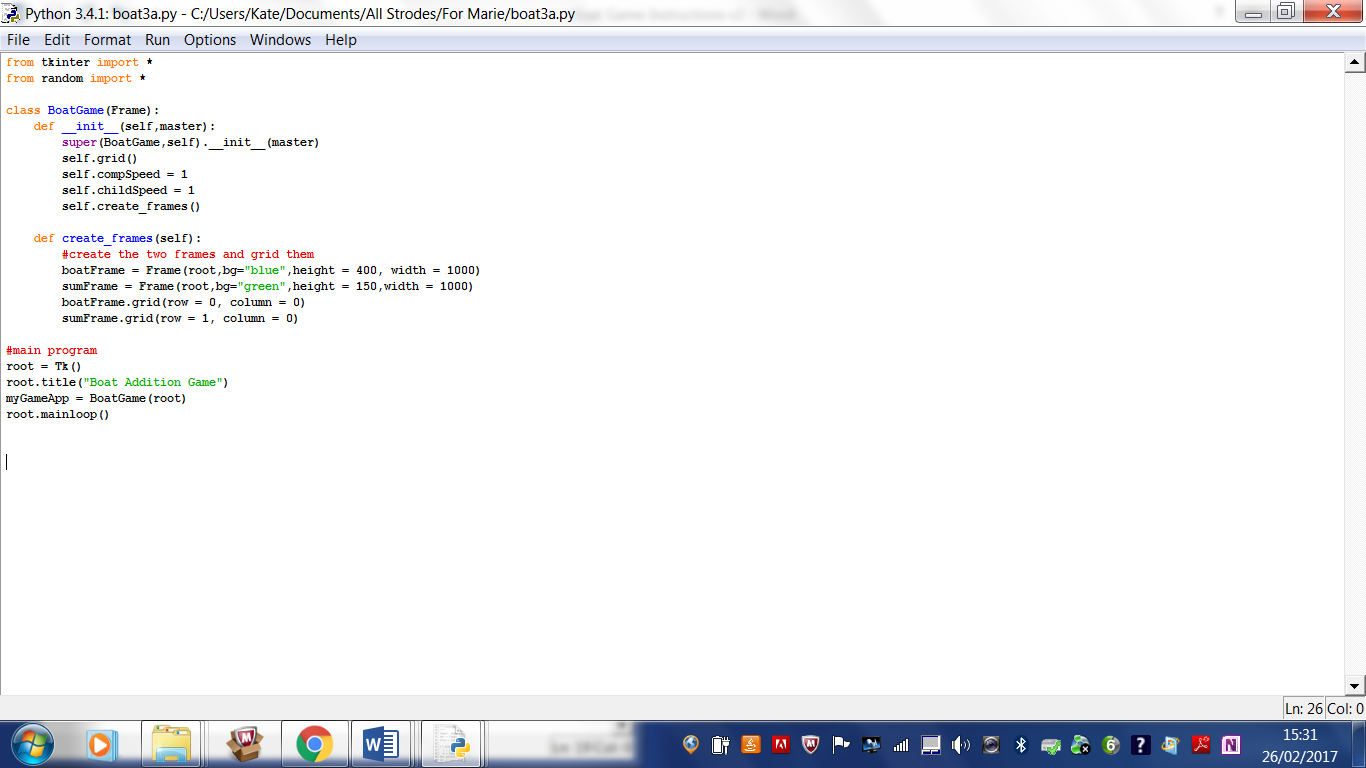
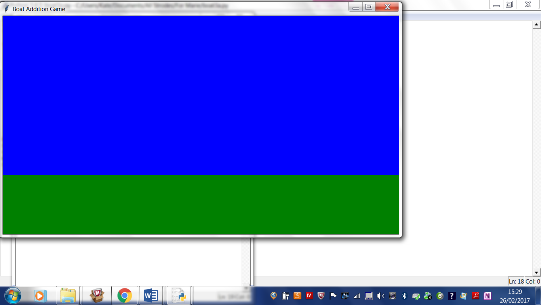
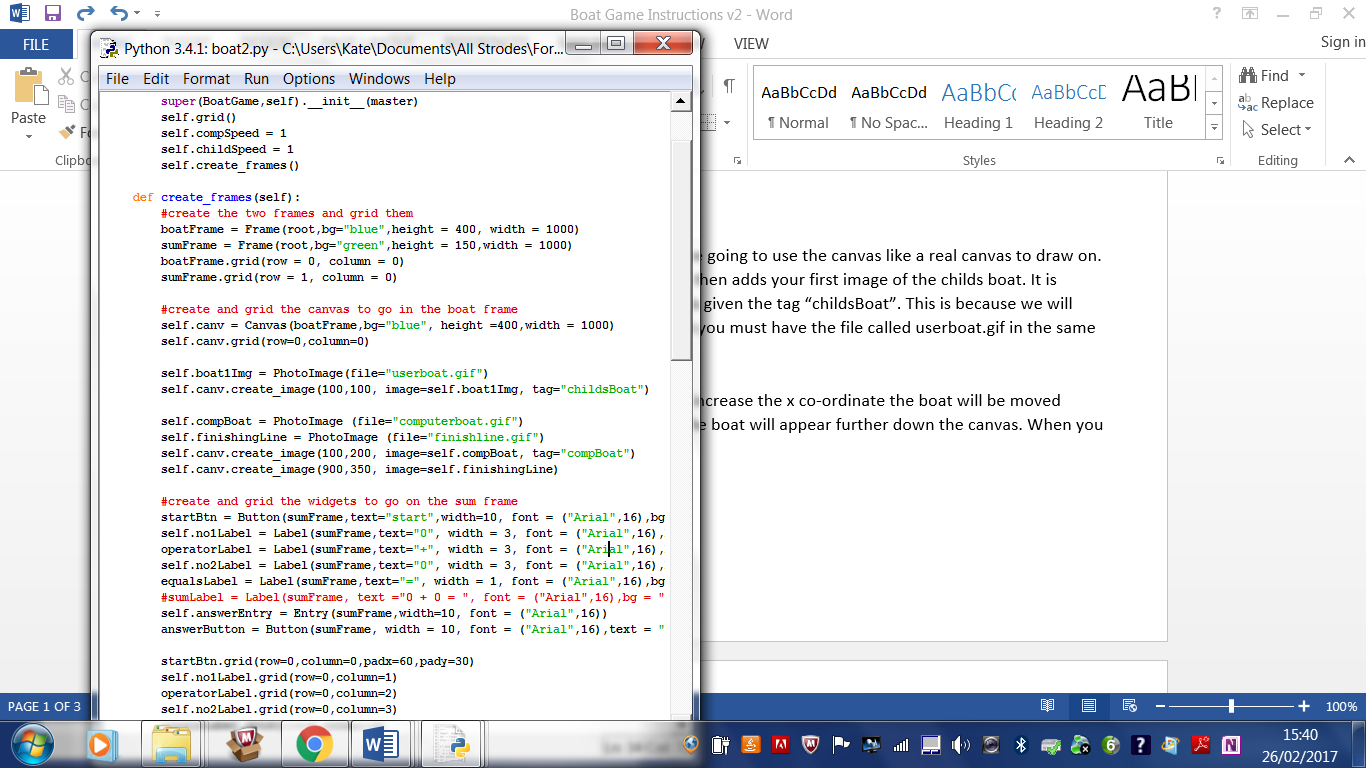
**BOAT ADDITION GAME**

You are going to create a program which is designed for a young child to learn basic arithmetic. It is based on the idea of two boats racing each other. One boat belongs to the computer and the other boat belongs to the child. The computer boat will move at a constant speed. The childs boat will move at the same speed as the computers boat to start off with. The program will repeatedly ask arithmetic questions. If the child gets the question right, their boat moves faster. If they get their question wrong, their boat moves slower. The game finishes when either the computer’s boat or the child’s boat reaches the finish line.

1. **Creating the boat frame and the frame.**

Type in the code above. It sets the start speed of the boats to 1, creates a Tk() window called root and two frames. The blue frame is the boat frame and the boats will sail from left to right on this frame (moving in the x-direction). The green frame which will contain the Maths for the child to do is called sumFrame. Both frames need to be gridded to be seen. Later on you may have to play around with the width of the two frames to make them match when the widgets have been added to the frames. The speeds of the boats will need to be accessed in other functions within the class, so have “self.” In front of them.

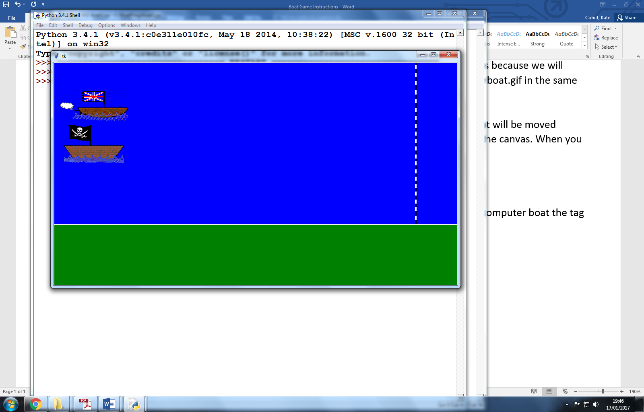
1. **Creating a canvas on the boat frame, and the image of the childs boat**



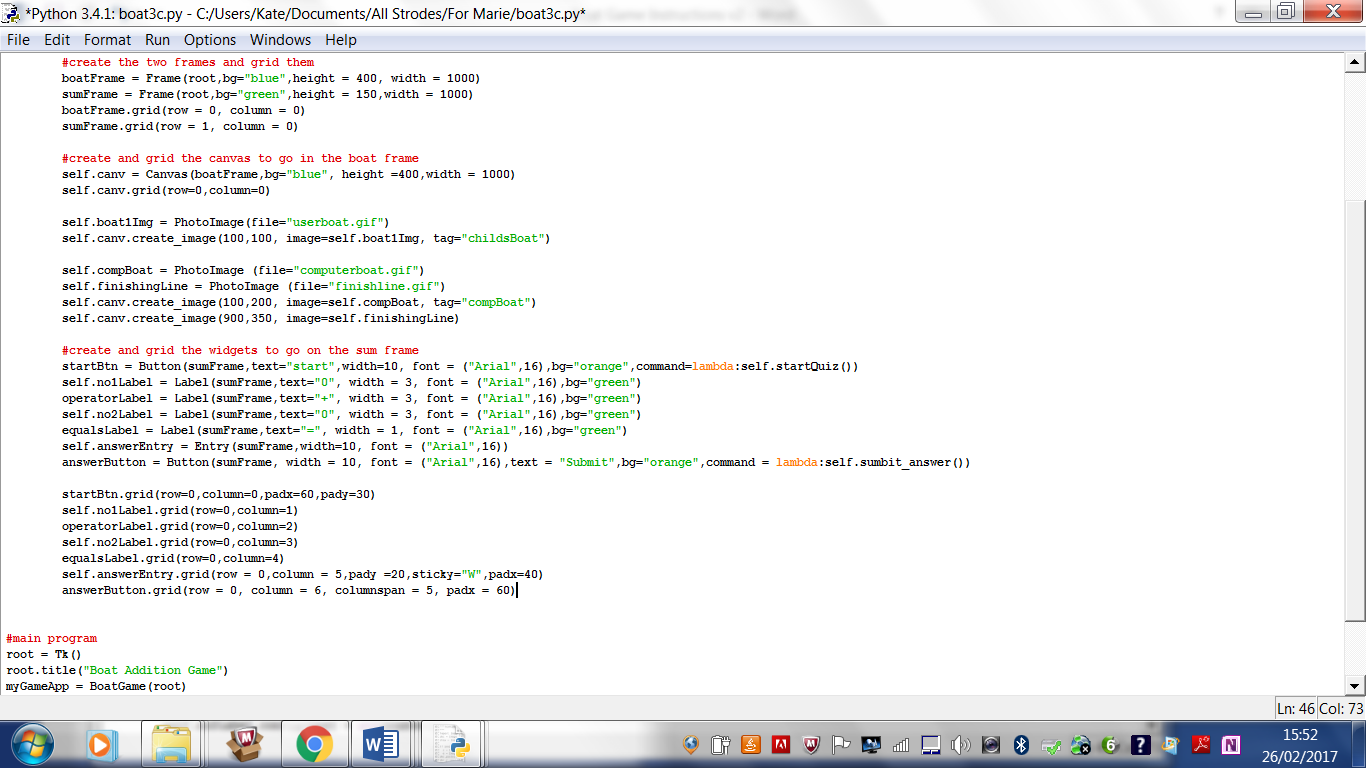
Type in the code above underneath your existing code in the create\_frames function. We are going to use the canvas like a real canvas to draw on. The code creates this canvas and grids it in the boat frame. It then adds your first image of the childs boat. It is positioned at location with co-ordinates (100,100). The boat is given the tag “childsBoat”. This is because we will want to refer to it in the code later on. For this code to work you must have the file called userboat.gif in the same folder as your program. I have used “self.” in front of variables we will want to use in other functions.

Experiment with the co-ordinates. You should see that if you increase the x co-ordinate the boat will be moved across the screen. However as you increase the y-cordinate the boat will appear further down the canvas. When you are happy with this, poistion theboat back at (100,100).

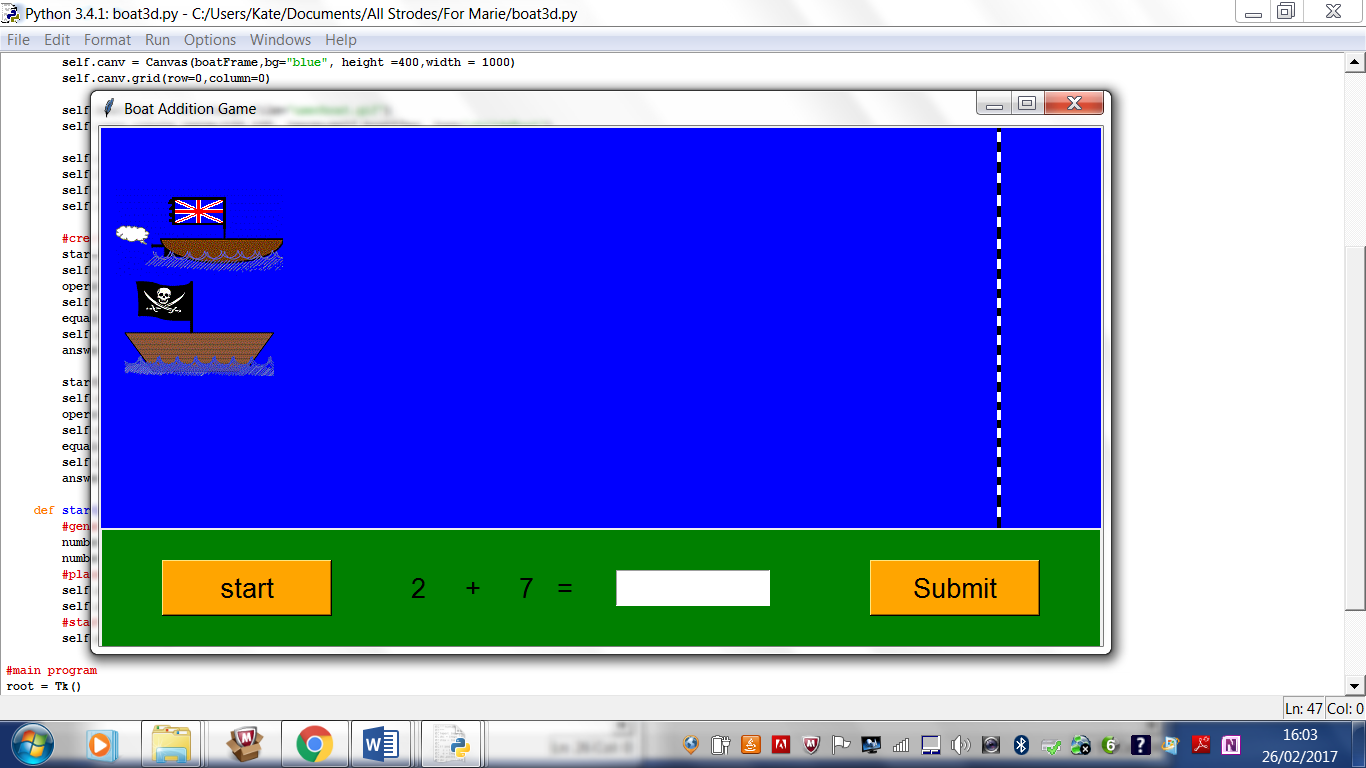
1. **Other images**

Get the computer boat and finishing line images up and running as shown to the left. Give the computer boat the tag “compBoat” and omit any tag for the finishing line. Again both the image files need to be in the same program as your folder. You will need to put “self.” in front of your variable names.

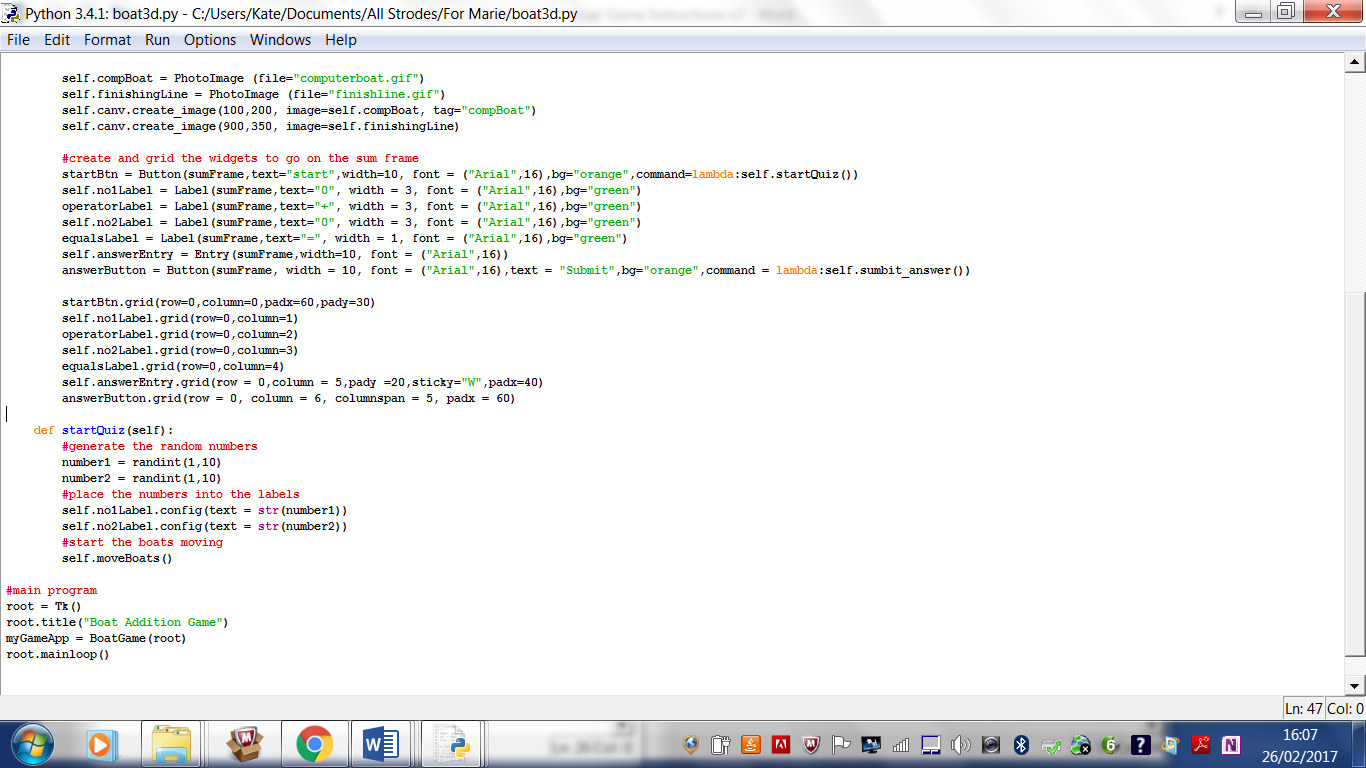
1. **Widgets for the Calculations Frame**

In the same function, create\_frames, underneath your existing code, create the start button, label, entry and submit buttons on the sumFrame and grid them. You might need to change the x padding so the green frame that the green frame takes up the same width as the blue frame above it. Sticky = “E” aligns the widget to the right, whilst sticky=”W” aligns it to the left. Play around with font sizes and colours, and see the effect of changing padx and pady. Don’t code the startQuiz function until step 5 below.

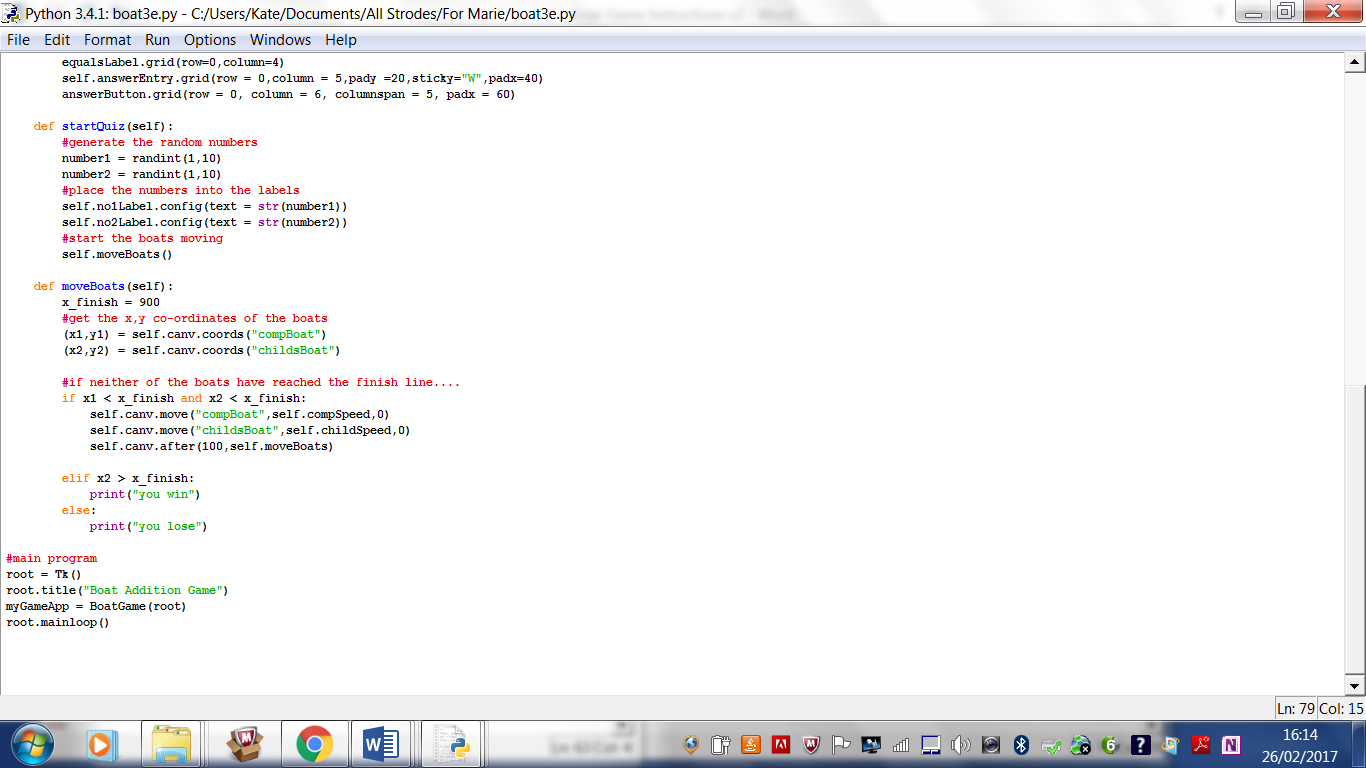
Your program will generate error messages in the shell when you click on either of the buttons as functions start\_quiz and submit\_answer have not been coded yet.

1. **Code the start button**

This is going to cause both the boats to move across the screen slowly and generate the first random question to be answered. Firstly make a new function called startQuiz as shown below. This function is a new method for the class and so should be coded so that it is indented inside the class:



This function generates two random integers, and converts them to strings using str, and then places them in the labels. There will be an error in the shell as function moveBoats has not been written yet, but everytime you click on the start button, a new pair of random numbers to be added is generated.

1. **Code the moveBoat function.**

I had placed my **finish line** at x = 900.

I believe that the co-ordinates returned using the two tags are the **top left** of the two images.

The **move** method moves the tagged image compSpeed or childSpeed pixels in the x direction and 0 pixels in the y direction.

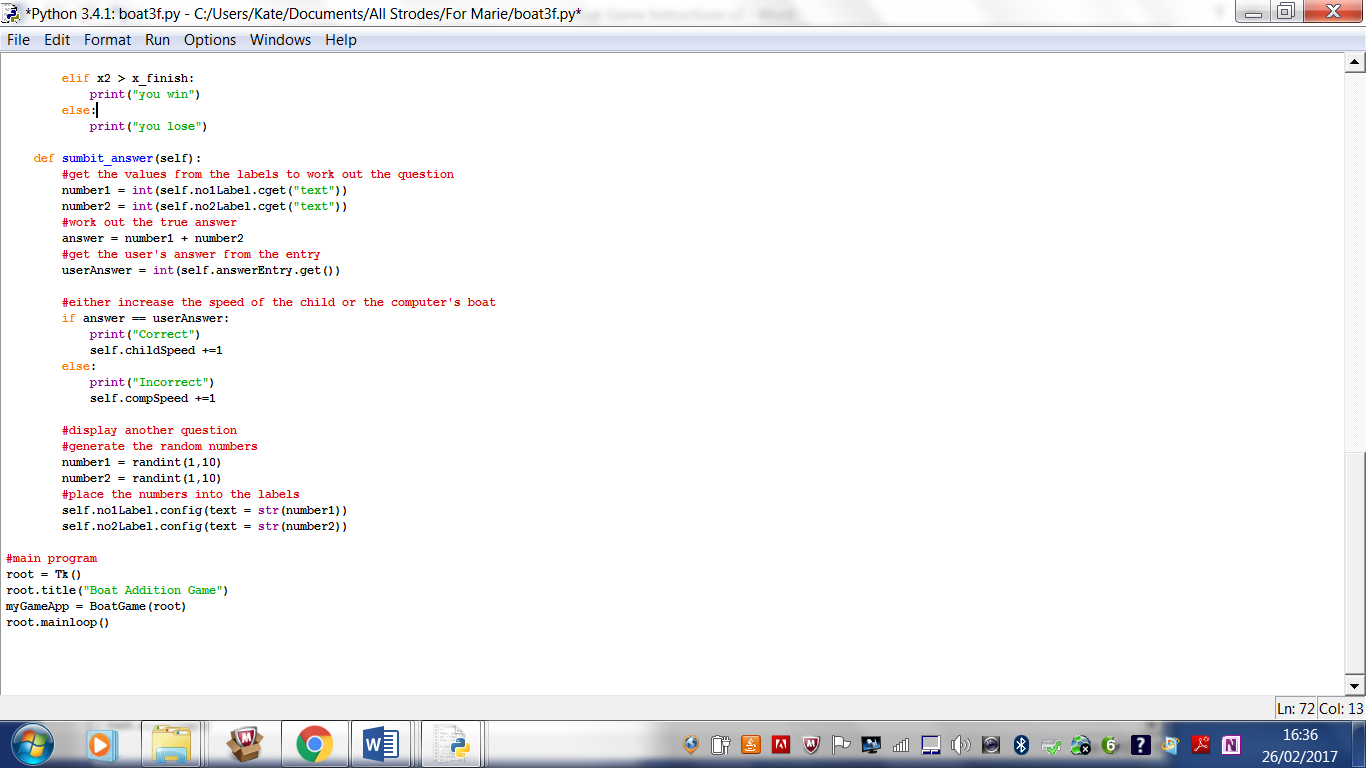
The **after** method waits for 100 millisecs (0.1 sec) and then calls the moveBoats function again. This is an example of a function calling itself. This is known as recursion.

Again, this function is a class method, so ensure it is indented inside the class.

If you run the program, you should find the boats move very slowly across the screen. At the top of the class you set the compSpeed and childSpreed to one pixel (so one pixwl every 1/10the of a second). If you wait long enough there will be a print message in the shell stating that you have won. You can change the speed, but you will probably want to change it back later on.

1. **Coding the quiz.**

This is the final class method to be coded. When the user clicks the submit button, the program retrieves their answer from the entry and the numbers to be added up from the labels on the screen. It works out what the correct answer should be an compares it with the user’s answer. If the user has got the answer correct a message is printed in the shell (not ideal) and the user’s boat speed increases by 1. If they got it wrong the computer boat speed increases and a different message is put in the shell. Finally a new sum is displayed on screen ready to be answered.

cget can be used to get any property such as width, font etc. Here is is used to get the text property of the label, and then converted to an interger so calculation can be done.

get is used to return the value typed in an entry. get() does not work on Labels.

**Extensions**

1. Create a Label which displays whether the user has got their answer right or wrong, rather than having to look at the shell.
2. Scoring – count the number of correct and incorrect questions.
3. Levels of difficulty which the user can set in advance– summing numbers 1-9, 11-20,1-100
4. The user can choose whether to multiply, add or subtract
5. A two player game? You would need a second keyboard and be able to detect input from it.