Computer Science 1 — CSci 1100
Lab 2 — Writing and Executing Functions
Fall Semester 2012-2013

Lab Overview

The primary goals of this lab are make sure everyone is comfortable using the Wing IDE, and to practice writing, testing and running functions using strings. If you complete the four goals below — checkpoints 1,2,3 and 4 below — you will have completed this lab. Show all completed checkpoints to your lab TA or one of the undergraduate mentors to obtain credit. You will get 25% of the credit for each completed checkpoint. If you need to

Throughout the lab period, if you have a question, start by chatting with your neighbor to see if s/he might have the answer or have a similar question. You are encouraged to help each other as much as possible. If you and your neighbor do not have the answer, raise your hand and ask the grad TA or the undergraduate mentor for help. If many students are requesting help at once, you may need to be patient. That's why it is better to start by asking your fellow students in the lab.

If you are finished early, help a class mate. Often one learns better when trying to explain. We will also give you some additional problems to try and get fancy.

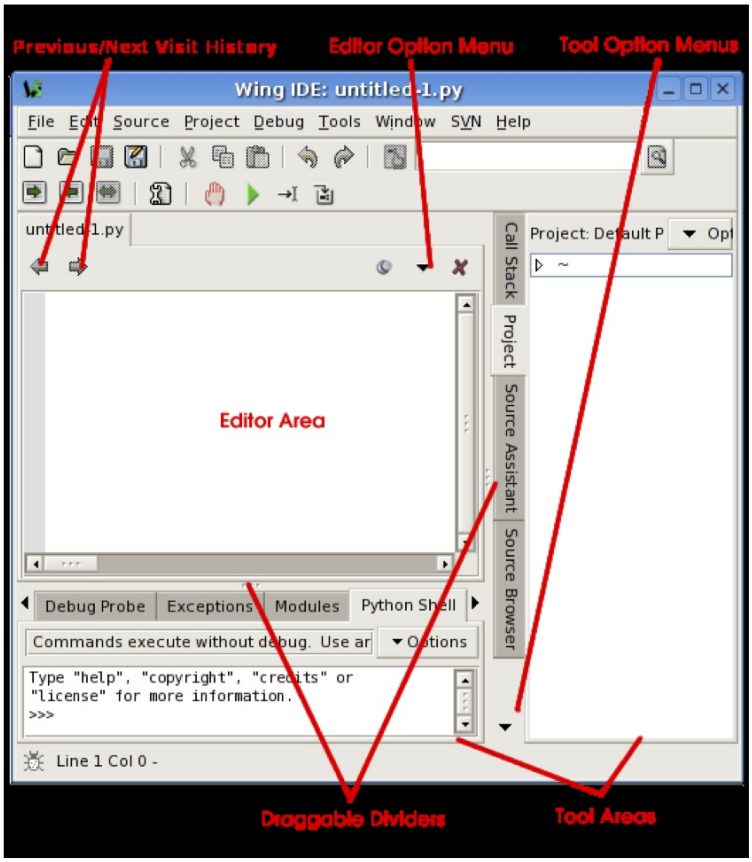

Figure 1: The different areas of the Wing IDE tool

Checkpoint 1: Starting Wing IDE

We are assuming you are all set with the virtual environment or have the equivalent packages installed.

First, make sure that Dropbox is actually running. You should see a small box symbol on the top right corner (see also in Figure 2). When you click it, it should show you options for Dropbox. If Dropbox is not running, then files you create will not be saved to the cloud. If it is not running, you can click on "Click on me to start Dropbox" to accomplish what the link says.

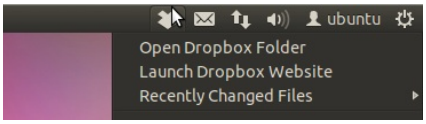

Figure 2: Check that dropbox is running

Create a new folder in Dropbox called **Labs** and a subfolder called **Lab2**. You will store all the files you create in this lab in this directory.

Next, start Wing IDE by clicking on the feather symbol on the left toolbar. You can learn about Wing IDE by clicking on **Help** > **Tutorial**. We recommend you get through this tutorial on your own. However, scroll down on the first page of the tutorial and click on **Getting Around Wing IDE** link. You will see the screen in Figure 1.

You can move the dividers around to minimize them. We will work mainly with two areas: the **Editor Area** and the **Python Shell** just below the Editor area. You can minimize the tool area to the right for now.

Ok, we are now ready to write our first program. Click on **File** > **New** or the small document item (the left most icon on top) to create a new file. Write

```
print 'eggs and spam!'
```

and execute by pressing the green arrow on the top bar. It will ask you to save it first. We will save it to the Dropbox. Navigate to **Dropbox** > **Labs** > **Lab2** and save it as **framed_greeting_1.py**. This will be the script for Checkpoint 1. You will see that the name of the file has changed from untitled-1.py to the name you have given.

Make sure you save each checkpoint into a seperate file. This will allow you to show them to the TA and also go back to a previous correct version of your program if you have made a mistake.

Now, get rid of the print statement above and change the code in this file to print 5 lines:

- First line: 10 stars.
- Second line: one star, 8 spaces, and another star
- Third line: one star, 2 spaces, the string 'spam', 2 spaces, and another star
- Fourth line: identical to the second line.
- Fifth line: identical to the first line

Remember the following important points:

1. To make the sure the numbers of stars and spaces are correct, you must use the replication function we learnt in class. Remember, you can just take a single character and multiply it by the number of times you must repeat it. Example: `'a'*5` repeats 'a' 5 times.

2. Given that the first and second lines are repeated as the fifth and fourth lines, you should use two variables to store these lines in. Then, you can just print these lines out when needed.

Now, when completed, save your script and run it. The output should look like this:

```
**********
*        *
*  spam  *
*        *
**********
```

Congratulations! You have now completed Checkpoint 1.

Checkpoint 2: Writing our first function

First save your current script (Python program) as **framed_greeting_2.py** for Checkpoint 2. Go to **File** > **Save As** and give it the new name. Now, you have two copies of the current file, but you are going to be making changes to this new copy. Make sure you save it in the same directory as the first file.

It is time to get fancy. We will introduce a function that prints any word, not just the word spam. For now, we will stick to four letter words.

Create a function that takes as input a word and returns the string that must appear on the third line of output. Call this function:

```
def frame_word(input_word)
```

If it is given `"eggs"`, it should return the string `"*  eggs  *"`.

Then, change your code to call this function with the word `"spam"`. Make sure the output looks the same as before. Make sure you save your script before running it.

Now, change your program so that you ask the user for a four letter word, read the input word, and print it out as framed. Remember to use `raw_input()` to read user input. The output should look like this:

```
Please enter a 4-letter word: eggs
**********
*        *
*  eggs  *
*        *
**********
```

Note that on the first line the `Please enter a 4-letter word:` came from your program, while the `eggs` is typed by the user. The rest of the output, of course, is generated by your program.

Congratulations, you have completed Checkpoint 2.

Checkpoint 3: Must frame words of any length

First save your current script as **framed_greeting_3.py** so that you can modify it for Checkpoint 3.

As an aside, please note that the checkpoints of this lab are illustrating a good way to write programs: build small pieces of code, test them, save them, and then extend them to build more complicated programs.

Now, we will make things more interesting. What if the string we enter is not 4 letters? Can we still have a nice frame? Try it with a longer word and see what happens?

```
Please enter a word: eggs and spam
**********
*        *
*  eggs and spam  *
*        *
**********
```

The lines 1,2,4,5 do not look right. We need to make sure that they are sufficiently long to cover the string we have constructed from the input word. String length (Lecture 4 notes) to the rescue!

You can look at the length of the string returned by your function and then adjust the length of each of the output strings accordingly. To think about how, please return to the lengths of the five output strings from Checkpoint 1: their lengths depend on the length of the word `spam` that appears in the middle of the output. Now that you have a different length string you will need to adjust the lengths of the strings forming the 1st and 5th line and the 2nd and 4th line.

The output from the string should look like the following, of course:

```
Please enter a word: eggs and spam
******************
*                *
*  eggs and spam  *
*                *
******************
```

Now, try it with the following strings:

`egg, spam, bacon`

Does it look right? If so, congratulations. You have completed Checkpoint 3.

Checkpoint 4: Framing a Greeting

First save your current script as **framed_greeting_4.py** so that you can modify it for Checkpoint 4.

Ok, now we are going to change the code so that we can frame two words. The first change you need to make is to ask the user to input a first name and then to input a last name. It should look like this:

```
Please enter your first name: John
Please enter your last name: Cleese
```

where the user has typed `John` and then `Cleese`.

The output we want looks something like

```
Please enter your first name: John
Please enter your last name: Cleese
************
*          *
*  Hello   *
*  John    *
*  Cleese! *
*          *
************
```

In other words, we want the program to output three framed lines containing the words 'Hello', the first name, and the last name followed by a '!'. (Hint: you need to combine the last name with the string `'!'`.)

You can add some code to your existing code and print these additional lines. But, with multiple lines, the spacing becomes an issue. You need to make sure that the end of the lines are aligned. If you used your current function and the new strings, you would get the following middle lines:

```
*  Hello  *
*  John  *
*  Cleese!  *
```

This is not good, we want to add enough spaces to make sure that they are aligned. To fix this problem, start by changing your function to:

```
def frame_word(input_word, total_length)
```

which works as before, but adds an additional spaces to the right of the `input_word` before the last star to make sure that the resulting string is of length `total_length`.

You last problem is to figure out the total length and pass it as an argument to `frame_word`. Fortunately, we can use the built-in function `max` (Lecture 3) to compute the maximum of the lengths of our three output strings (in this case, `Hello`, `John` and `Cleese!`). Change the remainder of the code to do so this. You musth also change the code for the first and second lines to make sure that they also make use of the total length you have just computed. Now, save and run your code. You should see what we showed above:

```
Please enter your first name: John
Please enter your last name: Cleese
************
*          *
*  Hello   *
*  John    *
*  Cleese! *
*          *
************
```

Try it with a couple more strings. If it all looks good, congratulations! You have completed Lab 2. Save everything and make sure you have shown all the checkpoints to your TA before you leave the lab.

Finished Early? Try this for extra credit.

Suppose you are done and want a new challenge. Here is an extra credit option you can try before you leave the lab. Instead of adding spaces to the right of the input strings, suppose we wanted to center the strings. This means, you will need to add spaces both to the left and to the right. The total length of the string remains the same, but you need to change your function. Plus, make sure that your code works for strings of even and odd lengths both.

Try it and see if it works:

```
Please enter your first name: John
Please enter your last name: Cleese
*************
*           *
*   Hello   *
*   John    *
*  Cleese!  *
*           *
*************
```

and

```
Please enter your first name: Washington
Please enter your last name: Irving
****************
*              *
*     Hello    *
*  Washington  *
*    Irving!   *
*              *
****************
```

If it does, congratulations! Show it to your TA before you leave the lab for credit.