CS 1: Introduction To Computer Programming, Fall 2013

Assignment 1: Getting started

Due: *Thursday, October 17, 02:00:00*

Coverage

This assignment covers the material in lectures 1 to 5.

Time hints

Numbers in bold at the start of problems (*e.g.* **[10]**) are a crude estimate of the time (in minutes) the problem should take. If it seems like a problem is taking you disproportionately long to complete (*e.g.* if you are spending hours on a problem rated **[10]**), that is a hint that you might want to talk to a TA. These numbers assume that you have attended the lectures and done the portions of the assignment that precede it.

Parts A and B have no time hints, because there is nothing to hand in for those sections.

What to hand in and how to hand it in

What to hand in

You will hand in a single file called `lab1.py`. This will consist of Python code and comments. For this assignment, only part C has anything you need to hand in.

For the exercises where you are asked to evaluate some Python expression or answer a question, write the answer in a comment, along with the exercise number and subsection. For instance, if you are asked to evaluate `2 + 2` for part C, exercise 3, section 6, you would write:

```
# Ex C.3.6:  2 + 2 -->; 4
```

You can use multiple lines if you like, but make sure each line is a comment.

For the exercises where you are asked to write some Python code, write the code with a preceding comment indicating the problem number (and subsection). For instance, if you were asked to write a function of one input which returns the input unchanged as the answer to part C, exercise 5, section 2, you could write:

```
# Ex C.5.2:
def identity(x):
    return x
```

How to submit your assignment

The CS 1 lab submission system is called "csman". The csman web site is at http://csman.cs.caltech.edu. The FAQ (Frequently Asked Questions) list has details of how to submit assignments. Those of you who have CMS cluster accounts will soon receive emails telling you how to log in. If you haven't received such an email by the due date for this assignment, you should email Donnie at donnie@cms.caltech.edu.

Part A: Getting started in the CS computer lab

Location

The CS computer lab is in room 104 of the Annenberg building. If you want access to the building (and the room) after hours (*i.e.* after 5 PM), and assuming that you are enrolled in the course, you can use your ID card to get in the building by holding it up to one of the card readers by one of the doors. There is also a card reader outside the computer lab. Email me (Mike) if the card reader isn't working for you (this is likely for students who have registered late). If you don't have card reader access, email your UID to the course secretary (Lisa Knox, lisa987@cms.caltech.edu) to get access.

The computer lab has a number of computers (hence the name). Collectively, they are known as the "CMS cluster" or the "CS cluster" (we use the two terms interchangeably). Most of them are running the Linux operating system; specifically, a distribution of Linux called "OpenSUSE Linux". You will need to know a little bit about Linux in order to use the system. Hopefully most of you went to the Linux tutorial lecture, but either way you should review the slides from that lecture (posted on the CS 1 web site) before doing any serious work on the computers.

Getting an account

Before you do anything else, you must have a CMS cluster account. The CS 1 home page (where you found this assignment) has instructions on how to do this under "Administrative information". It takes a couple of days for accounts to be

processed, so do this as soon as possible (don't wait for the day before the first assignment is due!).

Logging in locally

Once you have your account, you can log in to the CMS cluster. Go to the CS lab, find an unoccupied machine, and sit down. You will be presented with a login screen. In the box labelled "Username:" enter your login name. The box's label will change to "Password:". Enter your password. If you have done this correctly, you will see your "desktop" which will have a number of icons which you can ignore. There will be a menu bar that says "Applications Places System ..." etc. Click on "Applications". You will see a number of sub-categories including "Accessories", "Internet", etc. Clicking on "Accessories" will bring up another sub-menu; select "Terminal". This will bring up the terminal program. You will use this a lot! We'll get back to this, but first, let's see how to log in remotely (from outside the computer lab).

It is an unfortunate fact of life that many of the computers in the CS lab don't work properly. If you try to log in but fail, this may be the reason. Ask a TA to help. If you can, send email describing the problem to the system administrators at help@cms.caltech.edu. Please indicate the name of the machine that isn't working (which is written on a label on the front panel of each computer). Please specify what isn't working *e.g.* say "I couldn't log in to this machine" instead of just "It's broken".

Logging in remotely

You can log in to the computer lab remotely, but don't expect to be able to do your homework this way — our system neither encourages this nor even supports it, and we don't want you to try to do your work this way. Nevertheless, logging in remotely is sometimes useful, so here's how.

To log in remotely, you have to have a terminal program running on your own computer. For Windows computers, we recommend the PuTTy program (which you will have to install); for Mac OS X, either Terminal.app or iTerm.app is fine, and if you use Linux, you should already know how to find the terminal program ;-)

You should log in to `login.cms.caltech.edu`. *DO NOT* type this into a web browser's URL bar! It isn't a web site, and that won't work. On a computer running Linux or Mac OS X, you should do this:

```
% ssh login.cms.caltech.edu
Password: <enter your password>
```

and you can proceed from there. (The `%` is not something you type; it's just the terminal prompt. Your terminal prompt may be different, but we'll use the `%` character when we want to indicate the prompt. When we tell you to type

something at the terminal, never type the prompt.)

With PuTTy you will need to enter the hostname `login.cms.caltech.edu` in the configuration window that comes up when you start the program, and then click "Open" at the bottom. A terminal window will come up, and you'll be prompted for your login name and your password. Once this is entered, you will be logged in.

In all cases, type:

```
% exit
```

at the prompt to log out.

Using the terminal

From here on, we'll assume that you are working in the CS lab. Some of the things we describe may not work if you are logged in remotely, which will hopefully deter you from trying to work that way. Most importantly, WingIDE (the Python development environment) will not work remotely.

Now would be a great time to review the slides from the Linux tutorial lecture if you haven't already. You will be using the terminal mainly to launch programs (*e.g.* WingIDE) and to manage files (create directories, move files between directories, etc.). You don't need to be an expert in Linux for this class; a minimal number of commands will take you a long way.

Changing your password

The very first thing you should do once you start up a terminal (assuming you haven't done this already) is to change your password. The password that was assigned to you when you got your account is only valid for a few days, and you're expected to change it after you log in. This is easy to do from a terminal using the `passwd` command:

```
% passwd
Changing password for user <your login name>.
Enter login(LDAP) password:
```

and you just enter the new password. You will need to enter it twice (to minimize the chances that you typed it wrong the first time). *Make sure you remember your new password!* If you forget it, you will have to go see the sysadmins and get them to straighten things out.

Configuring the system

All the CS 1 software is installed in the directory `/cs/courses/cs1/install`. In order to give you access to this directory, you need to set up your system paths to tell them

to look for programs in this directory. We have provided a quick way for you to do this, which you only need to do once. From the terminal, type this:

```
% cp /cs/courses/cs1/setup/cs1user.bashrc ~/.bashrc
% source ~/.bashrc
```

After you've done this, you will be able to run all the CS 1 software from the terminal. You will *not* have to do this again, so the next time you log in to the computer, you don't have to repeat this step.

Starting WingIDE

To start up WingIDE, type this in the terminal:

```
% wingIDE &
```

and WingIDE will start up. The `&` is optional but recommended; what it does is allow you to subsequently enter more commands into the terminal. Without the `&`, you would have to wait until WingIDE exits before entering any more commands.

When you start up WingIDE for the first time, a window will come up with a license agreement. Click "Accept" and the main WingIDE window will appear. Make sure that the Python Shell in the lower right-hand corner says "Python 2.7.5". If it doesn't, something has gone wrong and you need to talk to a TA.

Working with WingIDE

The Python Shell

Now that WingIDE is running, we can start experimenting with Python. Let's start by working with the Python Shell in the lower right-hand corner. The first thing you should do is to type the following at the Python prompt:

```
>>> print "hello, world!"
```

If all goes well, Python should reply:

```
hello, world!
>>>
```

Woo hoo!

Note that Python not only printed the string you wanted it to print, but it also printed another prompt, indicating that it is ready for you to enter more commands. Try a few commands. (The commands are what follows the `>>>` prompt, and Python's expected response is printed on subsequent lines. Don't confuse the Python prompt with the terminal prompt; they aren't related.)

```
>>> 2 + 2
```

```
4

>>> 1.0 + 1.0 + 1.0/2.0 + 1.0/6.0 + 1.0/24.0 + 1.0/120.0 + 1.0/720.0 + 1.0/5040.0 + 1.0/40320.0
2.71827876984127

>>> 'foo' + 'bar'
'foobar'

>>> 'foo'.upper()
'FOO'

>>> '     this string has leading and trailing spaces    '.strip()
'this string has leading and trailing spaces'

>>> import math
>>> math.sqrt(2.0)
1.4142135623730951

>>> def func(x, y):
...     return x + y
...

>>> func(2, 3)
5
```

At this point, click on the "Options" button on the lower right side, and select the option "Restart Shell". Then try this:

```
>>> math.sqrt(2.0)
```

You should see something like this:

```
Traceback (most recent call last):
  File "<string>", line 1, in <fragment>
NameError: name 'math' is not defined
```

What this tells us is that when we restart the Python shell, all memory of the previous work we did is lost. Often this is what you want, but sometimes it isn't. This leads us naturally to the next topic.

The Editor

The majority of the WingIDE window is occupied by the top-central window. When you start up WingIDE for the first time, there is a tutorial there called "Introduction for New Users". Do yourself a favor: take a little time and read through this tutorial. It will familiarize you with the way that WingIDE works.

Once you're done with that, click on the "New" button near the top left-hand corner of the window. That will bring up a new tab on the editing window called "untitled-1.py". This is the Python source code editor. It is where you will be doing the majority of your work.

Type the following code into the editing window. (Don't try to cut-and-paste from the browser window; that probably won't work.)

```
def testme():
    print "this is a test"

def foo(x, y, z):
    return x + y + z
```

Notice how the code that you type changes color after you're finished typing it. This is what is called "syntax coloring", and most programming editors do it. It makes code much easier to read. Note also that WingIDE knows how to intelligently indent your code, so you don't have to do that yourself (at least, most of the time you don't).

You can immediately execute this code in the Python shell by hitting the "Run" button. Then, in the Python shell, you can do this:

```
>>> testme()
this is a test
>>> foo(2, 3, 4)
9
>>>
```

When you hit the "Run" button, it's as if you instantly typed the entire contents of the editor into the Python shell. This is often useful when debugging code (which is the process of removing errors from your code).

Saving your work

Much of the time, you will want to save the code you have been typing into a file so that you can continue working on it later. You will also need to save your work to a file in order to submit it for grading. Saving your work is easy: just click the "Save" button near the top. WingIDE will ask you which file you want to save; here there is only one: "untitled-1.py". Click on "Save Selected Files" and a file browser will come up. If you click on "OK", a file named "untitled-1.py" will be written to your home directory. Most of the time, though, you will want to rename the file to something else (which you can do in the dialog box at the bottom of the window), and you will probably want to save the file into another directory (which you can do by navigating there using the file browser). Let's rename the file to `foobar.py` and save it. You will note that the name of the file on the tab changes to `foobar.py`. Then, in the Python shell, do this:

```
>>> import foobar
>>> foobar.testme()
this is a test
>>> foobar.foo(2, 3, 4)
9
>>>
```

Congratulations! You have just written a Python module called `foobar`. The contents of this module are in a file called `foobar.py` which exists in one of your directories, and it can be reloaded later, re-edited, and submitted to the CS 1 grading system.

We *STRONGLY* recommend that you save your work often. Nothing is more frustrating than to work on a program for two hours without saving anything, and have all of that work wiped out because the computer crashed. You should get into the habit of saving your work automatically every ten minutes or so, or else after you've typed (say) ten new lines of code. The "Save" button will save your work to the file with the same name as the name on the tab; that's the one you need to use often. You can do this even more efficiently by typing <control-s> *i.e.* the "control" key held down while you type the "s" key.

To exit WingIDE, go to the File menu and select Quit. When you restart WingIDE, the editor will contain the last thing you were working on, which is very convenient.

That's all for this section.

Part B: [*OPTIONAL*] Getting set up on your own computer

Most of you own your own computer, and many of you will want to work on the assignments on your own computer in the comfort of your dorm. Logging in remotely to the CMS cluster computer to do your homework is *strongly* discouraged; the system is not set up to do this and the system administrators may get annoyed at you if you try to do it (also, it's not at all easy to do if you have a computer running Windows). The alternative is to set up the CS 1 software on your own computer and do your work there. This is *not* required (and in fact you should try to do as much work as possible in the CS lab, where the lab TAs will be there to assist you), but if you want to do this, read on.

In this assignment we will only show you how to set up Python and the WingIDE development environment. Subsequent assignments will also require extra Python packages; we will discuss how to set those up when the time comes.

We will be using Python version 2.7.5 for compatibility with the textbook. Earlier versions will work for most of the course, but there may be occasional problems, so we recommend you install the correct version. Python versions 3.0 and later are *not* acceptable, because there are significant changes to the Python syntax in those versions which are incompatible with the textbook and the lectures.

The version of the WingIDE Python development environment we will use is WingIDE 101 version 4.1.14-1. The "101" refers to a free version of WingIDE which can be downloaded from the WingWare web site.

What to do from here depends on which operating system (OS) your computer is running. Most computers run one of three operating systems: Windows, Mac OS X, or Linux. Which specific version of the operating system (*e.g.* Windows XP vs. Windows Vista) shouldn't make a difference as long as your computer is not too old. The actual software you need to set up is the same regardless of the operating system, but the details of setting it up are different.

Windows

Most of you have computers that run some version of the Microsoft Windows operating system. Fortunately, getting the CS 1 software to work on Windows is not too difficult.

First, you will need to install Python 2.7.5. Download the installer for Python version 2.7.5 here. Click on the link to download it; it will most likely be saved in the directory "`My Documents/Downloads`". Use the Windows Explorer (file manager) to inspect that directory; there should be a file called "`Python-2.7.5`" which is a Windows Installer Package. Click on that file and the installation will begin. A dialog box will ask you if you want to run that file; click on the "Run" button. You will be presented with a series of question/answer dialog boxes. Selecting "Next" on each box is fine, except that you should install all the optional packages under "Customize Python 2.7.5" (except for the Test Suite, which you won't need). Click on each extension you want to install and select "Entire feature will be installed on local hard drive". Once you're done, click the "Next" button at the bottom and Python 2.7.5 will be installed.

Now you need to install WingIDE. You can get the Windows Installer Package here. Again, click on the link to download the installer, and again, use the file manager to inspect the "My Documents/Downloads" directory. You should see a file called "wingide-101-4.1.14-1". Click on it and it will begin the installation process. Select the default values for everything (except that you should agree to the license agreement!). Once you finish, WingIDE will start up and you will have to select "Accept" on another license agreement. Do so, and you will see the main WingIDE window. The Python Shell in the lower right-hand corner should say "Python 2.7.5". If not, something has gone wrong and you should ask your TA for help. If everything is working, exit WingIDE. You can start it up again by using the Start button to find the program, but we recommend you make a shortcut to WingIDE by dragging the menu item to the desktop (ask a TA if you don't know how to do this).

Now you're ready to go.

Mac OS X

Mac OS X is an excellent operating system; I (Mike) use it as my main OS. Installing the CS 1 software on a Mac is almost as easy as installing it on Windows.

Before you start, you need to make sure that you've installed the "Developer Tools CD" that came with your computer. This is important because WingIDE depends on the X Window System, which is included as part of the Developer Tools CD. Without this, nothing else will work. To test if you have this installed, open a terminal window (Terminal.app) and at the prompt, type:

```
% x
```

(Omitting the `%`, as usual) If you see something like this:

```
Xquartz: X11.app = /Applications/Utilities/X11.app/Contents/MacOS/X11
Xquartz: Starting X server: /Applications/Utilities/X11.app/Contents/MacOS/X11 --listenonly
```

then X Windows is installed and you can continue to the next step. If not, you need to install the Developer Tools.

Next, you need to install Python 2.7.5. Most Macs already have some version of Python installed, but it will likely not be the correct one. You can find out by opening a terminal, and at the prompt, type

```
% python
```

If you get this result:

```
Python 2.7.5 (... some other stuff ...)
(... some other stuff ...)
```

then you can skip this step. Otherwise, you need to download the Python 2.7.5 installation package here. Clicking on the link will download the installer to the "`Downloads`" directory. Navigate to that directory using the Finder and search for the file called "`python-2.7.5-macosx.dmg`". Click on it, and a new Finder window will come up with some icons, including one that says "`Python.mpkg`". Click on that icon and the installation will begin. You will be asked a few questions; select the default values and agree to everything and installation will *really* begin. Once it's done, Python 2.7.5 will be installed on your computer. One nice thing about this is that it's perfectly OK to have multiple Python versions all running on the same computer. Python 2.7.5 will actually be located in the directory `/Library/Frameworks/Python.framework/Versions/2.7/bin`.

Finally, you need to install WingIDE. The installation package for WingIDE can be found here. Click on the link to download it to the "`Downloads`" directory. Navigate to that directory using the Finder and search for the file called "`wingide-101-4.1.14-1-i386.dmg`". Click on it, and a new Finder window will come up with some icons, including one that says "`WingIDE.app`". Drag that icon to the Applications folder in order to install WingIDE. Once this is done, you can launch WingIDE by using the Finder to get into the Applications directory and clicking on WingIDE. You will need to agree to the WingIDE license, but after that you'll see the main WingIDE window. The Python Shell will be on the lower right-hand side; if it says "Python

2.7.5" you are ready to go. If not, you have to configure Python. In the menu, select "Edit/Configure Python", and a new window will pop up. Select "Custom" for the Python executable. In the dialog box below this, type in "`/Library/Frameworks/Python.framework/Versions/2.7/bin/python2.7`". Click "OK". Go to the Python Shell and click "Options/Restart Shell". Now it should say "Python 2.7.5". If this doesn't work, ask a TA for help.

Once all this is done, you're ready to go.

Linux

If you use Linux as your primary OS, congratulations, you rock! We like Linux too ;-) Unfortunately, it's difficult to give general instructions for installing the course software on Linux, as each Linux distribution has its own package manager and does things somewhat differently. Compounding the problem is the fact that nearly all Linux distributions already provide Python, but the version of Python provided will probably not be the one we will be using in this course. To determine this, start up a terminal and type

```
% python
```

at the terminal prompt (`%` here is the terminal prompt character; as usual, don't type that). If it prints:

```
Python 2.7.5 (... some other stuff ...)
(... some other stuff ...)
```

then you're fine (the `2.7.5` is the version of Python running). If not, you will need a different version of Python. If your Linux distribution's package manager has a "`python2.7`" package, you can install that. However, the most reliable way to install Python is by compiling it from the source code, which is available here. If you already know how to install packages from source, do so. We recommend you install Python in a separate directory which is just for CS 1 software *e.g* `/usr/local/cs1` (you will probably need root privileges to do this). This will facilitate later installations of other Python packages you will need for subsequent assignments. If you don't know how to do this, ask a TA or one of the instructors to help you. Like riding a bicycle, it's very easy once you know how.

Once you have Python 2.7.5 installed, it's time to install WingIDE 101. WingIDE 101 is available for download here. Download it and run the following commands (we're assuming you want to install it in `/usr/local/cs1`):

```
% tar zxf wingide-101-4.1.14-1-i386-linux.tar.gz
% cd wingide-101-4.1.14-1-i386-linux
% python ./wing-install.py
```

After this, just answer the questions the installer asks. Instead of the default locations (`/usr/local/<whatever>`) change them to `/usr/local/cs1/<whatever>`. Make sure

your `PATH` environment variable (set *e.g.* in your `.bashrc` file) includes `/usr/local/cs1/bin`; again, if you don't know how to do this, ask a TA.

If you've done all this correctly, you can then start WingIDE by typing

```
% wing-101 &
```

and you're ready to go. (The `&` is not required, but if you type it you will be able to enter terminal commands while WingIDE is running.)

The first time you run WingIDE you will have to accept the license of the software; click "Accept" and you won't have to do it again.

Once WingIDE is running, look at the Python shell in the lower right-hand corner. If the Python version is 2.7.5, you are done. Otherwise (and assuming you've installed Python version 2.7.5 somewhere on your system), you'll have to tell WingIDE where the correct version of Python is located. Do this by clicking on Edit/Configure Python, which will bring up a new window. Set "Python Executable" to "Custom". Then enter the location of the correct Python version in the dialog box immediately below. After this is done, click "OK", restart the program, and the Python version in the Python shell should be the correct one. If you have a problem with this, ask your TA for help.

Any other OS

If your computer runs an operating system other than Windows, Mac OS X, or Linux, you are on your own. You can ask a TA for help, but we recommend that you try to obtain a computer that runs one of the three major OSs unless you're willing to invest the time to figure everything out for yourself.

Part C: Exercises

In this section we will do some basic programming exercises to test your understanding of the lecture material and to let you experiment with WingIDE and the Python shell. We will also introduce you to a few features of Python that we didn't talk about in the lectures.

1. [**10**] For each of the following expressions, what value will the expression give? Verify your answers by typing the expressions into the Python shell. Write your answers as a Python comment.

    1. `9 - 3`
    2. `8 * 2.5`
    3. `9 / 2`
    4. `9 / -2`

5. `9 % 2`
6. `9 % -2`
7. `-9 % 2`
8. `9 / -2.0`
9. `4 + 3 * 5`
10. `(4 + 3) * 5`

2. [**10**] Python includes a lot of operators of the form `op=`, where `op` is one of the standard operators. For instance, the following are all operators: `+= -= *= /= %=`. The meaning of these operators is as follows:

   ```
   x op= y
   ```

   is the same as

   ```
   x = x op y
   ```

   For instance, `x += y` is the same as `x = x + y` and `y -= 2` is the same as `y = y - 2`. For each of the following statements, what will be the value of the variable `x` after the statement has been executed? Assume that the statements are entered into the Python shell one after another, so that subsequent statements can depend on the values of previous ones. Write your answers as a Python comment.

   1. `x = 100`
   2. `x = x + 10`
   3. `x += 20`
   4. `x = x - 40`
   5. `x -= 50`
   6. `x *= 3`
   7. `x /= 5`
   8. `x %= 3`

3. [**10**] Write a step-by-step description of what happens when Python evaluates the statement `x += x - x` when `x` has the initial value of `3`. What is `x` after the statement has been evaluated? Write your answer as a Python comment.

4. [**10**] Complex numbers are the sum of a real number and an imaginary number. An imaginary number is just a real number multiplied by the square root of `-1`. The square root of `-1` is often called `i` or `j`. Python allows you to enter complex numbers directly, in the form `Yj` (for imaginary numbers) and `X+Yj` (for complex numbers).

   What are the values of the following expressions?

   1. `1j + 2.4j`
   2. `4j * 4j`
   3. `(1+2j) / (3+4j)`

What are the results of the following two expressions?

1. `(1+2j) * (1+2j)`
2. `1+2j * 1+2j`

Why do you think they're different? What does this tell you about the way Python handles complex numbers?

Write all your answers as a Python comment.

5. [**10**] However, not all complex operations act the way you would expect them to. Enter the following into the Python shell:

```
>>> import math
>>> math.sqrt(-1.0)
```

This will give the following error message:

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: math domain error
```

That means that the `sqrt` function can't handle square roots of negative numbers. Fortunately, the `cmath` (complex math) module exists which can handle problems like these. Now try this:

```
>>> import cmath
>>> cmath.sqrt(-1.0)
```

Notice that this does give the expected answer of `1j`.

What are the values of the following expressions?

1. `cmath.sin(-1.0+2.0j)`
2. `cmath.log(-1.0+3.4j)`
3. `cmath.exp(-cmath.pi * 1.0j)`

Finally, why do you think it's a better idea to write

```
import math
import cmath
```

than

```
from math import *
from cmath import *
```

in a program?

Write all your answers as a Python comment.

6. [**5**] What are the values of the following expressions? (If there are multiple lines in a question, give the value of the expression on the final line.) If an expression gives an error, write the error message. Write all answers as a Python comment.

    1. `"foo" + 'bar'`
    2. `"foo" 'bar'`
    3. ```
       a = 'foo'
       b = "bar"
       a + b
       ```
    4. ```
       a = 'foo'
       b = "bar"
       a b
       ```

7. [**5**] Rewrite the following string using single quote characters on either end of the string instead of triple quotes. Don't use any operators or function calls (you don't need them). Write the answer inside a Python comment.

    ```
    '''A
    B
    C'''
    ```

8. [**10**] Write a single Python expression which will generate a string of 80 `'-'` characters. The expression itself shouldn't be longer than 10 characters. Write the answer inside a Python comment.

9. [**5**] Write a single-line Python string which, when printed using Python's `print` statement, will print:

    ```
    first line
    second line
    third line
    ```

    (without any leading spaces on each line). Write the string inside a Python comment.

10. [**15**] Given variables `x` and `y`, which refer to the values `3` and `12.5` respectively, write a line of code using `print` to display each of the following messages. When numbers appear in the messages, the variables `x` and `y` should be used in the `print` statement. Use the `%` formatting operator and a format string for each example. In this case, write your answer as Python code (not inside a comment).

    1. `The rabbit is 3.`
    2. `The rabbit is 3 years old.`
    3. `12.5 is average.`
    4. `12.5 * 3`

5. `12.5 * 3 is 37.5.`

11. [**10**] Use `raw_input` to prompt the user for a number. Store the number entered as a `float` in a variable named `num`, and then print the contents of `num`. Use the prompt string `"Enter a number: "` as the argument to `raw_input`. Write the answer as Python code (not in a comment).

12. [**10**] Write a function called `quadratic` that takes four input arguments `a`, `b`, `c`, and `x` and computes the value of the quadratic expression

    $ax^2 + bx + c$

    for those values of `a`, `b`, `c`, and `x`. Note that `**` is the power operator in Python, though you don't actually have to use that. (What else could you use if you don't use the `**` operator?)

    Write your answer as Python code (not in a comment).

13. [**30**] DNA is composed of four distinct base pairs: adenine (A), cytosine (C), guanosine (G), and thymine (T). In a DNA molecule, A bases pair with T bases and C bases pair with G bases. A value of interest is the proportion of C/G bases in a DNA molecule, since C/G bases bind more strongly than A/T bases (and thus, a DNA molecule with a large proportion of C/G bases is more stable than one with a large proportion of A/T bases). Given a DNA sequence, this can easily be computed by counting up the number of each base, and taking the ratio of the (G or C) bases as a proportion of the total.

    Use Python's `help()` function to learn about the `count()` method of Python's strings (typing `help(''.count)` at the Python interactive prompt is one way to do this). Then use this to write a function called `GC_content` which takes in a string representing a DNA sequence and returns a single float, which represents the proportion of the bases which are either G or C. You may assume that the input string has only A, C, G, or T bases. For instance:

    ```
    GC_content('ACCAGTGTAG')         --> 0.5
    GC_content('ATATATATA')          --> 0.0
    GC_content('GCGGCCATGCATGGGG')   --> 0.75
    ```

    One pitfall here is that dividing two integers in Python throws away the remainder, so you will want to convert the numerator and denominator of the ratio to floats using the `float()` function (built-in to Python) before doing the division.

    Add a docstring to your function so that typing:

    ```
    help(GC_content)
    ```

    in the Python shell will print out a description of what the function does, what

its input argument should be, and what its output represents.

Make sure you test your function on the examples given, no matter how sure you are that it works correctly!

Part D: Miniproject

Most assignments will have a miniproject which will involve solving larger-scale programming problems. There is no miniproject in this assignment.
