

Using IDLE for 15-110

Step 1: Installing Python

Download and install Python using the **Resources** page of the 15-110 website. Be sure to install version ~~3.5.0~~ and the correct version depending on whether you have a PC or a Mac.

Find Python on your computer to make sure it was installed correctly. Notice that IDLE was installed along with it.

Preview	Yesterday, 11:04 PM	36.2 MB	Application
Python 3.3	Yesterday, 10:52 PM	--	Folder
IDLE	Yesterday, 10:52 PM	174 KB	Application
Python Documentation.html	Yesterday, 10:52 PM	98 bytes	Alias
Python Launcher	Yesterday, 10:52 PM	293 KB	Application
Update Shell Profile.command	May 13, 2013, 4:57 PM	3 KB	Termin...l script
QuickTime Player	Yesterday, 11:04 PM	16.1 MB	Application
R	Feb 4, 2014, 10:39 AM	3.9 MB	Application
Reminders	Yesterday, 11:04 PM	5.5 MB	Application
RStudio	Jan 28, 2014, 5:12 PM	91.9 MB	Application
Safari	Yesterday, 11:04 PM	34.1 MB	Application
SketchBookExpress	Apr 22, 2014, 6:56 PM	131.1 MB	Application
Skype	Apr 7, 2015, 7:21 AM	62.4 MB	Application

Step 2: Opening IDLE

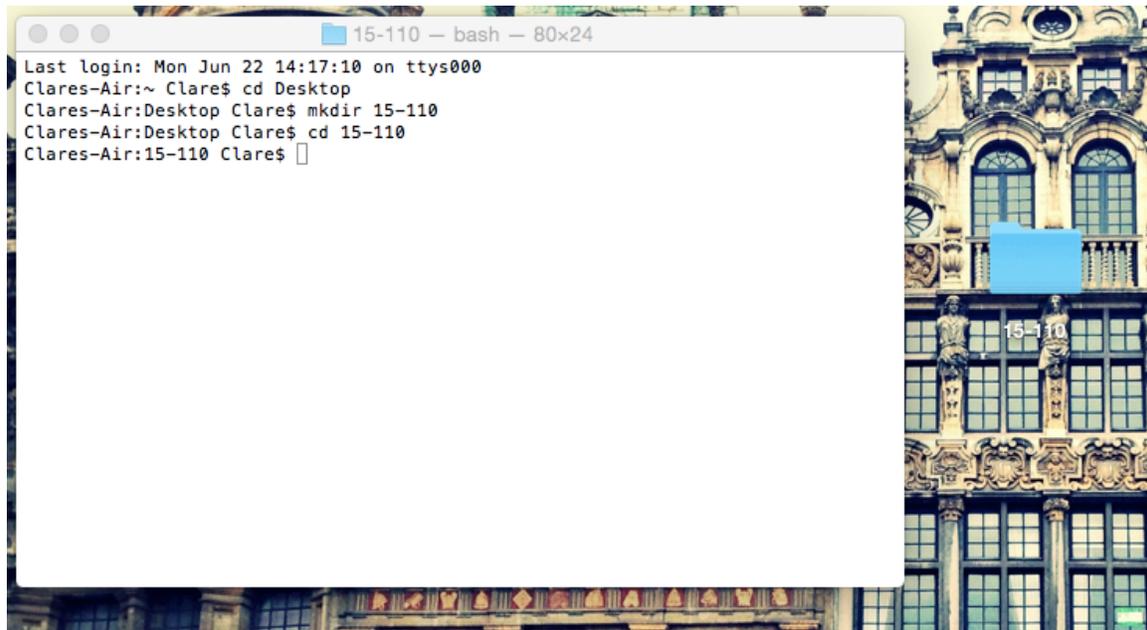
You can open IDLE by clicking on it from within your computer's programs or you can open it from your computer's terminal.

Note: For the purposes of this tutorial we'll be making a folder in the Desktop to save our files in. This will most likely mirror what you will be doing on your own computer for programming assignments. When you're working in lab you'll be making your files in a private folder within the server, so getting to them will be a bit different than it is here. We'll show you how to create these folders and access them in lab.

2a: Opening IDLE from the Terminal

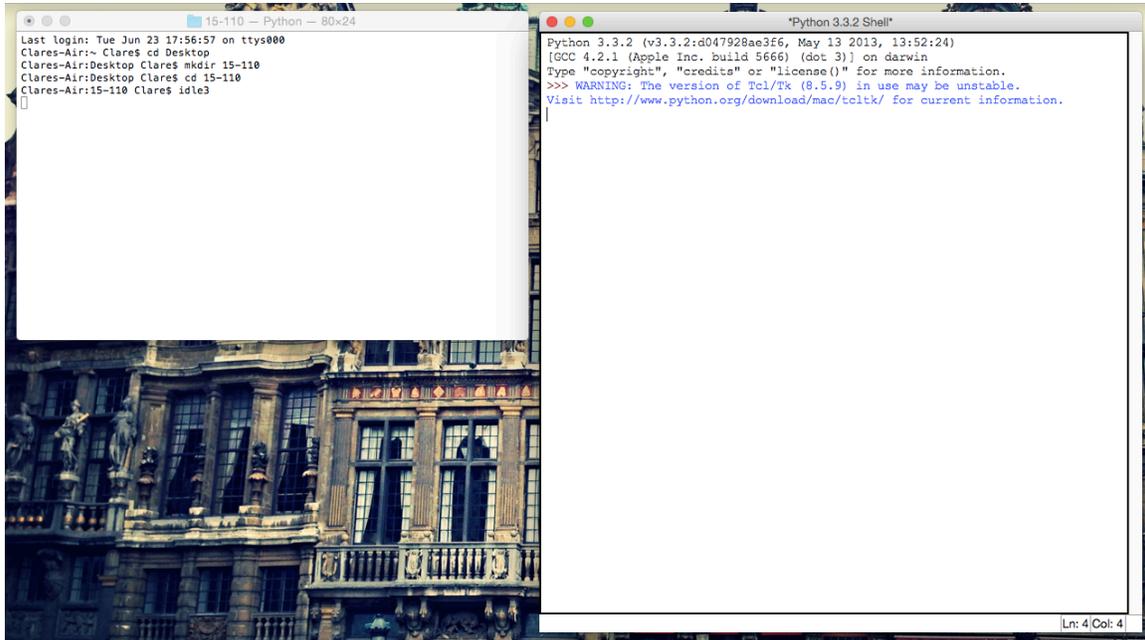
- Open your computer's terminal or command line.

- Say I want to create a folder on my desktop where I will store all my 15-110 files. I would type:
 - `cd Desktop` (cd stands for “change directory”)
 - `mkdir 15-110` (mkdir stands for “make directory”)
 - `cd 15-110`
- So we’ve made the folder 15-110 on our desktop by entering these 3 lines in the terminal, pressing Return after each. Here’s what this looks like in the terminal:

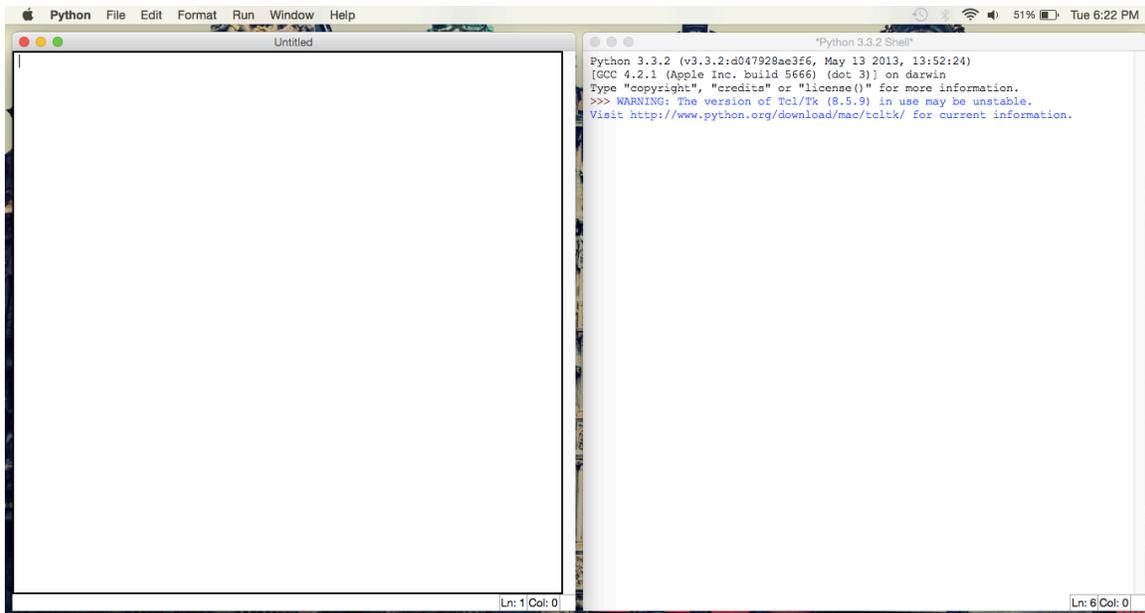


You can see this created a folder called 15-110 on my Desktop

- We can open an IDLE file inside our 15-110 folder by typing `idle3` and pressing return.



- This opened the python shell. We'll use this in a little while, but for now we need to open a text editor. We do this by going to File->New Window in the shell. Here's what it looks like when both the shell and the text editor are open:

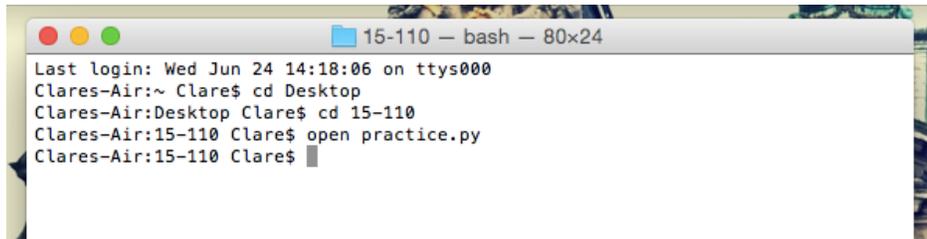


2b. Opening a pre-existing IDLE file

- You can only do this if you have a saved .py file somewhere on your computer. (Note that we have not

yet saved the above file. This step may be something to come back to later)

- In the terminal, make sure you have used `cd` to enter the folder in which the file you want to open has been saved. For this example I have saved `practice.py` inside our `15-110` folder.
- Type
`cd Desktop`
`cd 15-110`
`open practice.py`
You should see the IDLE window appear
- Remember that you can also open pre-existing files by clicking on them from within the folder they are in if you find that easier.



```
15-110 - bash - 80x24
Last login: Wed Jun 24 14:18:06 on ttys000
Clares-Air:~ Clares$ cd Desktop
Clares-Air:Desktop Clares$ cd 15-110
Clares-Air:15-110 Clares$ open practice.py
Clares-Air:15-110 Clares$
```

Step 3: Writing Code in IDLE

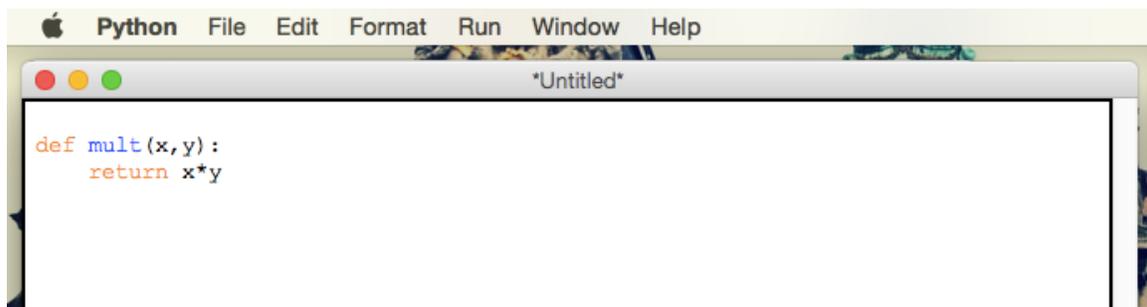
Everything we type now will be entered into the IDLE window, not the terminal.

We can use IDLE to write functions. Let's write a simple function called `mult` that multiplies two numbers together.

- First we define the function by typing
`def mult(x, y):`
`def` stands for define, followed by the name of the function `mult` followed by `x` and `y` in parenthesis separated by a comma. `x` and `y` are the two variables we want to multiply together. They are called **arguments** or values that we will be working with or manipulating within the function. Any arguments in any function are given in parenthesis after the name of the function separated by commas. Then we finish defining the function by putting a semicolon at the end of the line.

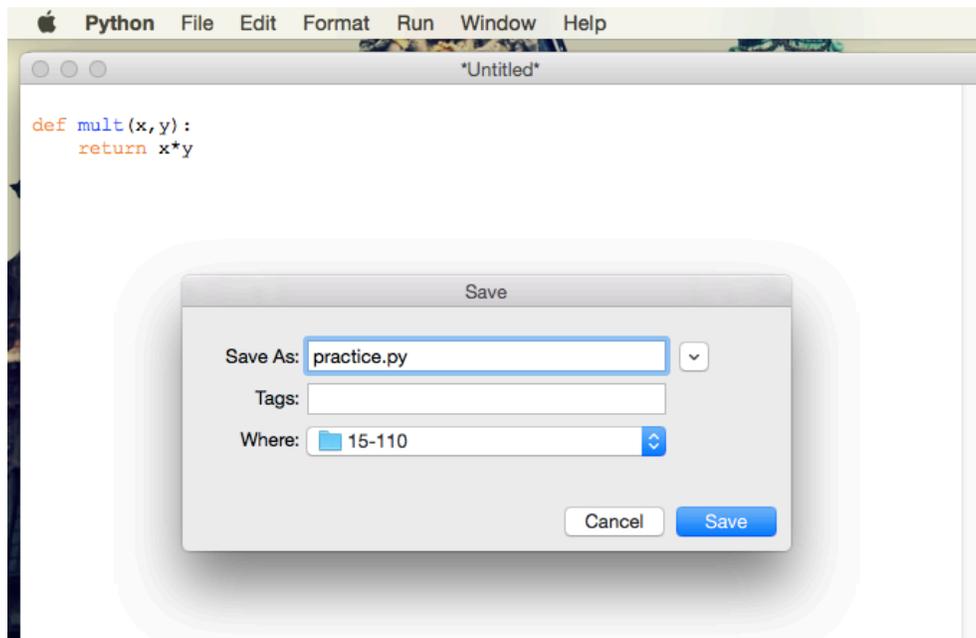
- Next we go to the next line and type `return x*y`
This should be indented from the previous line. We type return before what we want the result or output of the function to be. In this case we want the result to be the product of the two arguments x and y. Note that * denotes the operator for multiplication in Python.

Here's what `mult` looks like written in IDLE:



```
Python File Edit Format Run Window Help
*Untitled*
def mult(x,y):
    return x*y
```

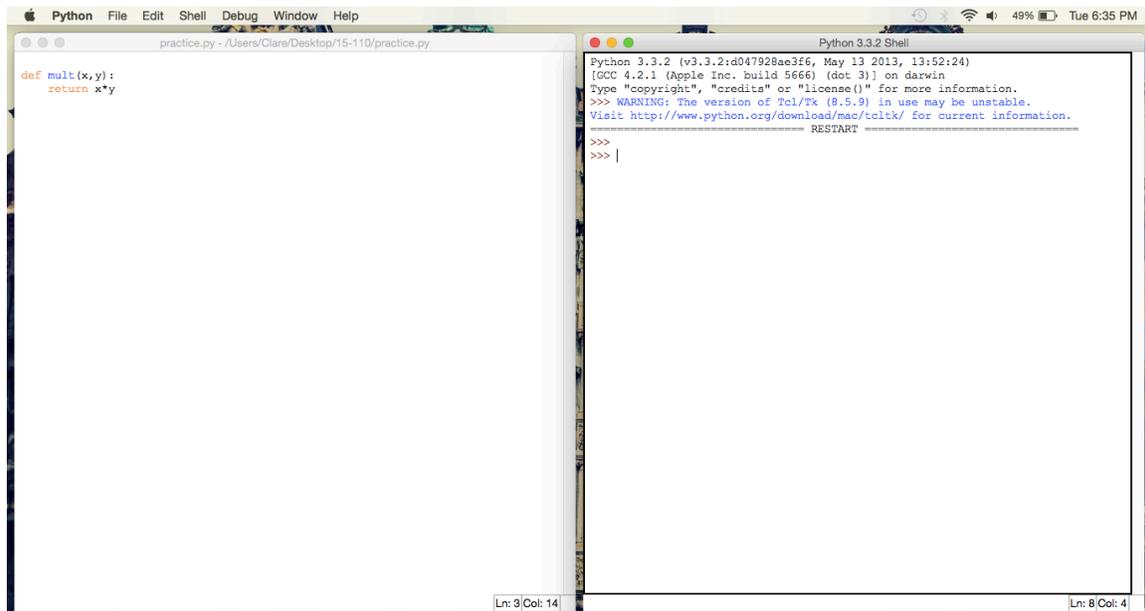
Be sure to go to File->Save or press Ctrl S (for PC) or Command S (for Mac) and save in the 15-110 that we created earlier. Make sure the file name ends in .py. Let's call this one `practice.py`.



Step 4: Running Code in IDLE

Now that we've written a function, let's test it out by running it. Make sure you have saved your .py file as recently as you have made any changes to your code.

Within IDLE go to Run->Run Module or press fn F5 on your keyboard. You will see that we restarted the **shell**.



The screenshot shows the Python IDLE interface. On the left is a code editor window titled 'practice.py' with the following code:

```
def mult(x,y):  
    return x*y
```

On the right is a 'Python 3.3.2 Shell' window. It displays the Python version information and a warning about Tcl/Tk. Below this, the prompt '>>>' is shown twice, indicating the shell is ready for input.

After the >>> is where we will start typing. Note that you can use the shell for doing basic python operations. For example we can make a variable `a` represent the number 5 and the variable `b` represent the number 4 by typing

```
>>> a = 5  
>>> b = 4
```

following each line by pressing return.

If we ask the shell what `a` is, it should return 5 just like if we ask the shell what `b` is, it should return 4. Here's what all this looks like in the shell:

```
Python 3.3.2 Shell
Python 3.3.2 (v3.3.2:d047928ae3f6, May 13 2013, 13:52:24)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> WARNING: The version of Tcl/Tk (8.5.9) in use may be unstable.
Visit http://www.python.org/download/mac/tcltk/ for current information.
===== RESTART =====
>>>
>>> a = 5
>>> b = 4
>>> a
5
>>> b
4
>>> |
```

Now let's get to running our `mult` function. Since we've already got variables `a` and `b` assigned to the numbers 5 and 4, let's multiply `a` and `b` together. So `a` and `b` are our arguments. Remember that we call functions by their name followed by their arguments in parenthesis, separated by commas. So we type

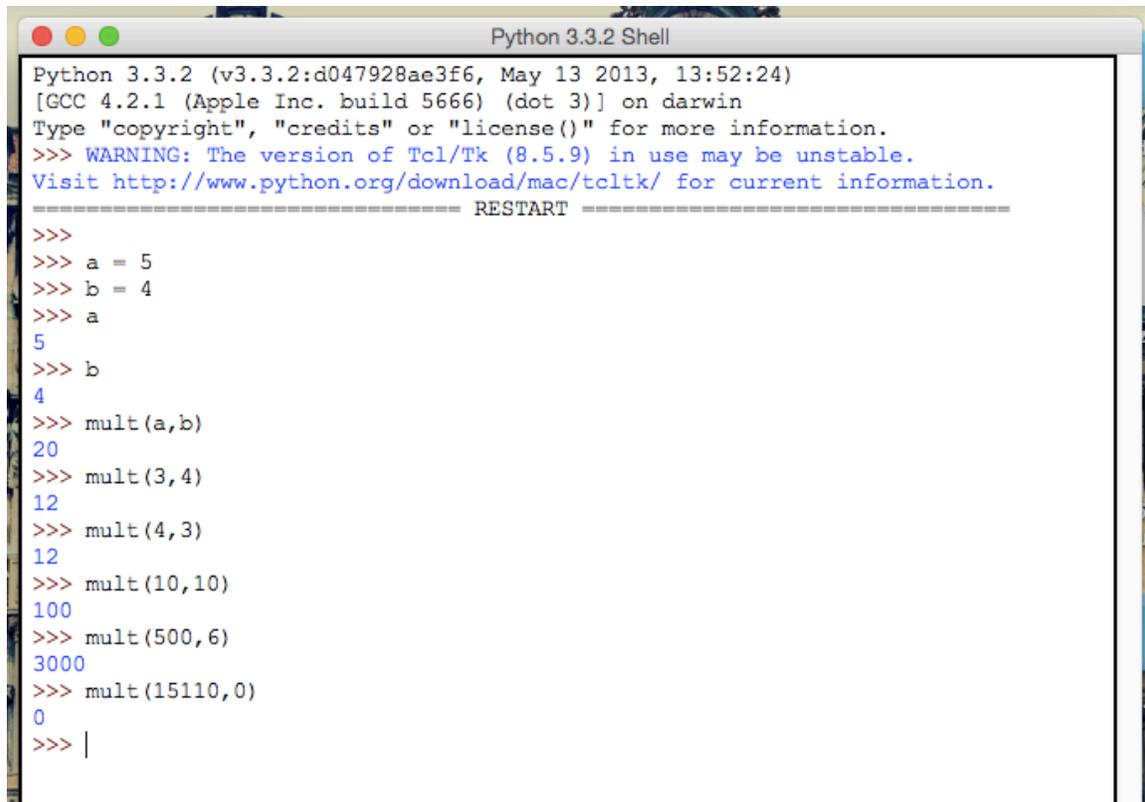
```
>>> mult(a,b)
```

and press enter and the shell should output the number 20 to us like this:

```
Python 3.3.2 Shell
Python 3.3.2 (v3.3.2:d047928ae3f6, May 13 2013, 13:52:24)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> WARNING: The version of Tcl/Tk (8.5.9) in use may be unstable.
Visit http://www.python.org/download/mac/tcltk/ for current information.
===== RESTART =====
>>>
>>> a = 5
>>> b = 4
>>> a
5
>>> b
4
>>> mult(a,b)
20
>>> |
```

This is the output we expected since the product of 5 and 4 is 20. It looks like our `mult` function works! However, we should always test it on a few different sets of arguments to make sure.

We don't have to enter arguments in the shell as variables like we did in the last test; we can just enter numbers themselves as the arguments when running in the shell like so:



```
Python 3.3.2 Shell
Python 3.3.2 (v3.3.2:d047928ae3f6, May 13 2013, 13:52:24)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> WARNING: The version of Tcl/Tk (8.5.9) in use may be unstable.
Visit http://www.python.org/download/mac/tcltk/ for current information.
===== RESTART =====
>>>
>>> a = 5
>>> b = 4
>>> a
5
>>> b
4
>>> mult(a,b)
20
>>> mult(3,4)
12
>>> mult(4,3)
12
>>> mult(10,10)
100
>>> mult(500,6)
3000
>>> mult(15110,0)
0
>>> |
```

Notice that we get the output we expect for every pair of arguments we call `mult` with. Therefore we can conclude that our function works!

If for some reason there is a problem with our function, we go back to the text editor window of IDLE where we wrote our function, make edits, save the changes, and go to Run->Run Module again to restart the shell. Then begin testing the function again.

Remember, if you have any difficulty or questions, post to Piazza or come to office hours.
Happy programming!