1. **Shell vs. Editor**
2. Go to the start menu, search for ‘IDLE,’ and open ‘IDLE (Python GUI).’
3. You should see the following (with a blinking cursor after the >>>):

*Python 2.7.10 (default, May 23 2015, 09:44:00) [MSC v.1500 64 bit (AMD64)] on win32*

*Type "copyright", "credits" or "license()" for more information.*

*>>>*

1. This is the **Shell**.
2. The **Shell** will be used for testing code. You can only input things line by line, and they’ll happen immediately once you hit Enter.
3. For notation’s sake, throughout this tutorial, when I want you to type something into the **Shell** (and then hit enter), it’ll look like the following:

>>> print ‘hello, world!’

hello, world!

1. Having done that, you should see some blue text saying ‘hello, world’ on the next line
2. Now go to File, then click New File. Alternatively, Ctrl+N
3. This should just be a completely empty white box, no >>>, although you’ll still have a blinking cursor. This is the **Editor**
4. The **Editor** is where we’ll compile our finished code, and save it. To get this code to run, you need to press F5. It’ll ask you to save first – do so in a consolidated place for all of your code (sorted by project, probably). Note that when you ‘save as,’ sometimes you manually have to type the extension .py, or it won’t save with the Python association.
5. For notation’s sake, throughout this tutorial, when I want you type something into the **Editor**, it’ll look like this (note that anything after #’s in Python are treated as comments. So you could safely copy/paste the entire thing, or just the code between my comments. Same result either way):

###Editor code: save and run

print "I don't even like Fetty Wap"

###Editor code: save and run

1. Running this code, you’ll see that you get the same result as before out of the **Shell**: some blue text with what you printed. Quick sidenote: because you have a single apostrophe in the string of text, you have to encase the string with double quotes (otherwise, as before, you can just use single quotes).
2. There will be some times that we don’t want to run the code in the **Editor** straight away. In those scenarios, it’ll look like this:

###Editor code: save only

print "I don't even like Fetty Wap"

print “But I do”

###Editor code: save only

1. Okay, that’s it.
2. **Variables**

The building blocks of coding.

1. Open the **Shell**
2. First we’ll do Variables. Super basic concept.

>>> myFirstVariable = 50

1. Output: absolutely nothing! Because we’re not asking the **Shell** to print anything, you can’t see it happening. But it happened.
2. All we’re doing here is storing the value 50, for later use, in the variable “myFirstVariable.” We can access it later, like this:

>>> print myFirstVariable

50

1. Just so we don’t have to type myFirstVariable all the time, let’s change that over to myf for ease.

>>> myf = 50

1. Variables can be pretty much anything. Here, we just used an integer. But it could be a string (e.g., myf = “Fetty Wap has a cool face”), or a decimal (e.g., myf = 50.24124), or a list or a dictionary or some other things which we’ll get to later. But basically, you’re storing information for future use.
2. \*Gibberish variables + CamelCase
3. Take a look at the various operators you can use with numbers in Python, and play around a bit in the **Shell**: <http://www.tutorialspoint.com/python/python_basic_operators.htm>

>>> myf + myFirstVariable

100

>>> newValue = myf + myFirstVariable

>>> newValue

100

>>> alsoWithStrings = "The Chicago Cubs are fabulous"

>>> print alsoWithStrings

'The Chicago Cubs are fabulous'

>>> moreAccurate = alsoWithStrings + 'ly overrated'

>>> print moreAccurate

The Chicago Cubs are fabulously overrated

1. **For loops**

For when you need to do something, over and over and over again.

1. Next, ‘For loops.’ Up until now, we’ve had to print things line by line:

>>> print 1

1

>>> print 2

2

>>> print 3

3

>>> print 4

4

1. ‘For loops’ give us a way to quickly automate this. The syntax is **For** \_\_\_\_ **in** \_\_\_\_\_:

>>> for i in range(1,5):

print i

1

2

3

4

1. Four things to note about this.
   1. Python treats this as going *up to* 5, but not *including* 5. That’s why you printed 1-4. If you want to print 1-5, use range(1,6).
   2. We don’t have to use the i. We can choose to do anything, that number of times. We can even affect things that are outside the loop!

>>> myf = 60

for i in range(1,5):

print "just checking that we're still in the loop!"

myf += 1

just checking that we're still in the loop!

just checking that we're still in the loop!

just checking that we're still in the loop!

just checking that we're still in the loop!

>>> print myf

64

* 1. Also, we can iterate through anything, not just ranges.

>>> for i in "hello world":

print i

h

e

l

l

o

w

o

r

l

d

* 1. Finally, i simply functions as a variable here. You could have equally written something silly:

>>> for chicagoCubs in range(0,5):

print chicagoCubs

1

2

3

4

1. **If/then statements**

Simple but powerful logic gate.

1. Let’s reset our variable:

>>> myf = “Fetty Wap is mediocre”

1. The format of if/then statements is, “if \_\_\_\_\_\_\_ is TRUE: do stuff.” So how do we check if something is true?

>>> myf == “Fetty Wap is mediocre”

True

>>> myf == 500

False

1. Now let’s use that variable as a logic gate.

>>> if myf=="Fetty Wap is mediocre":

print 'Yeah he is'

Yeah he is

1. Three things of note:
   1. You don’t actually have to say “Then” in Python, it’s implied
   2. Notice the double equal sign we had to use, == . We do this during if/then statements, because a single = is used to set variables, and here we’re checking if two things *are already* equal to each other.
   3. If you set the right side of the check to something else, then your ‘if statement’ will not do anything. E.g.,
      1. >>> if myf =="Why are we talking about Fetty Wap so much":

print 'Yeah he is'

###Output: nothing####

because myf is still “Fetty Wap is mediocre,” not “Why are we talking about Fetty Wap so much”

1. We can do a whole bunch of interesting things with if/then statements. We can ask if things are **not equal (!=)**, if two statements are **both true (and)**, if **either of two statements are true (or)**. First, real quick, do this:

>>> mySecondVariable = 877

1. Okay, now to demonstrate !=, and, or:
   1. >>> if myf != "Taylor Swift is where it's at":

print "Damn straight"

Damn straight

>>> if myf == "Fetty Wap is mediocre" and mySecondVariable==877:

print "This is basically magic"

This is basically magic

>>> if myf == "Sufjan Stevens, tho" or mySecondVariable==877:

print "Where’d this hipster come from"

Where’d this hipster come from

1. There are two other things we can do with if/then statements. **Else if** (in Python this is shortened to **Elif**) and **Else**. First, let’s change myf over to an integer:

>>> myf = 50

1. Okay, now check out these two sets of code, with a minor difference on the elif’s:
   1. >>> if myf==40:

print "this won't happen"

elif myf==30:

print "this won't happen either"

elif myf==20:

print "nor this"

else:

print "this one will print"

“this one will print”

because myf is 50, and none of our conditions are met, we default down to the “else” statement, and do the last print

* 1. >>> if myf==40:

print "this won't happen"

elif myf==50:

print "this won't happen either"

elif myf==50:

print "nor this"

else:

print "this one will print"

“this won’t happen either”

because we changed the 1st elif statement to elif myf==50, which is true. Note that the script stops checking after that, so **even though the 2nd elif myf==50 is true**, the print statement “nor this” never gets read (and neither does the else: statement).

1. **Surprise Test!**

Alright! We’ve come a long way. Now, using the **Shell** to test out ideas, and the **Editor** to compile your code, put all of these skills together to solve this classic, basic coding puzzle:

*"Write a program that prints the numbers from 1 to 100. But for multiples of three print “Fizz” instead of the number and for the multiples of five print “Buzz”. For numbers which are multiples of both three and five print “FizzBuzz”."*

For an extra challenge, do this in 142 characters or less!\* (As with all coding, there are multiple solutions; some just more efficient, some more readable, etc.).

\*As measured by Word, not Twitter (Twitter counts every time you use a “newline,” or “\n”).

(if you get stuck, it’s very easy to find solutions online. E.g., here, where I sourced the exact question text: <http://c2.com/cgi/wiki?FizzBuzzTest>.  **I strongly recommend working through the problem yourself, though, and not peeking**)

Notes for lecture: (Timing, tradeoff of characters vs. readability vs. speed)