Intro to Python

Module 1

You turn in each homework assignment on the class site. I will grade them on the site. I often grade quickly, so don't be surprised to see me grade your homework soon after you turn it in.

I will not accept late homework, because the first thing I will do in our classes is to go over the previous homework. Please be sure to read my comments.

I work very hard at giving comments on assignments. Please go back after your assignment has been graded and read my comments. I really want to help, and those individual comments are my way of trying to help you individually.

There is no required book. But if you want a more detailed version of the material, I recommend "Learn to Program with Python 3" by Irv Kalb. It is a superset of the material in this course.

All of my notes and sample file are available to you on the class site. The file are time locked. Each set of notes is available on the day of the class. Each set of sample files is available at the start of the class. Each homework assignment is available at the beginning of the class.

There will be a 10 minute break each hour.

++++

Vocabulary is very important to me. Take out a piece of paper or open a file, and I will give you definitions of words as the course progresses.

A little more about my background:
Started working in software at 16.
First computer IBM 1130 computer with 8K of “core” memory.

Rutgers University, NJ BS in Computer Science
Moved to CA to take first job.

Worked in compiler design – (ask if anyone knows what that is)
Got Masters degree at night while working during the day.

I have used many different languages
Fortran, APL, Assembly, PL/1, Cobol, WatFor, RPG, PL/M.
Worked for Apple Computer for a few years in UI software. Learned object oriented programming using Object Pascal
Worked for Acuson as manager of UI group –
Most of my work was in Adobe Director with its programming language Lingo. (including an entire eLearning System)
Flash and ActionScript.
Over the past few years, I’ve gotten into Python.

The same concepts I learned in my first programming language are still usable today. Once you learn the basic concepts of one programming language, you will find that you can pick up a new computer language very quickly. No matter what the language – and there are many – the underlying concepts are very similar. The key things that we will learn about: variables, assignment statements, if statements, while loops, function calls – are all concepts that are easily transferable to any other language.

In this course, we will be using the Python language. Python was designed to be very clear and easy to read. The syntax – the way that you write Python code was specifically designed to allow programmers to build programs quickly without the need for a lot of extraneous words and symbols. I will talk about the ‘readability’ of program very often, and it is a key concept in writing code in what has become known as a ‘Pythonic’ way.

Python and versions:

Python was created in the 90's by Guido von Rossum. (Worked for Google, now at DropBox in SF.) It has two current versions, 2.7 and 3.7 or 3.8. We will be using the "Python 3".

(We used to use the earlier version 2.7, but have recently upgraded. There are very minor differences as it applies to this course.)

Sample programs in Python:

So, what can you do with Python? Python is a general-purpose language. That is, it can be used for a very wide variety of jobs. That's why it is becoming more and more popular. Let me give you 3 quick examples of programs written in Python:

1) +++++ Magic 8 Ball (explain toy)

Here is a program that simulates the Magic 8 Ball toy. It is a good example of the types of programs we will be working with in this class. [Run] Notice that it asks the user for input, does some computation, and generates some output. These are the three main steps in all computer programs.

++++ Now, let's take a look at the underlying Python code of that program. I am not expecting you to understand any of this now, but if you have never seen computer code before, this should give you a good basic idea of what it looks like. [Describe major features]

2) Connect 4 – This is an example of another style of programming that can be done in Python. While the base Python language doesn't know anything about screens and pointing devices (mouse), clever programmers can write code that extends Python. In this case, there is a free package called PyGame that allows for these types of interactions. This is a program that plays the game Connect 4. I found the basic program in a book written by someone I know (Al Sweigart), but I rewrote

about 90% as a coding exercise, to practice my style of coding in Python. [Run]
This is the type of thing that you can do with much more experience in programming. I do NOT expect you to be able to build something like this with only this class.

3) Robot – this is a robot that was built and programmed by high school students for an international competition called VEX robotics. I am a beta tester for a new Python development environment, and I have reprogrammed the robot completely in Python.

Python is maintained as an "open source" project supported by the Python Software Foundation. This means that there is no single company behind the software.

Python has been installed on all the computers in here. But you should install it on your own computer. Go to the center of the Python universe:

++++ Python.org. [Show]
http://www.python.org

Click on Downloads

Click on Windows, Mac OSX, or other

Choose Python 3.7 (or whatever the latest version is)

Install resulting downloaded file:

Python-3.7-xxxx.yyy

_____
Version 3.7 has already been installed on all the computers in the classroom.

IDLE:

When you download Python, you also get a development environment called IDLE. This is the program that we will use to write and run Python code. There are many different environments you can use to write Python code, but as part of the Python download, IDLE is free. IDLE is cross platform, Mac, Windows and Unix/Linux.

Here's how to get to it:

Windows: Click on the Start button, type in IDLE

Mac: Open Applications, open Python 3.7, double click on IDLE

Open IDLE – See welcome screen. Opens the Python "Shell".

Can bring up preferences to change fonts/sizes if you wish.

Hello World:

When you are learning a new computer language there is a tradition that programmers write what is called the "Hello World" program. That is, you write a program that writes out "Hello World", just to make sure that you can get something to work. Let's do that now with Python. The Python Shell gives you a prompt that looks like three greater than signs. This means that the shell is ready for you to type something. At the >>> prompt, enter the following:

++++
print('Hello World!')

(press Return or Enter)

It prints Hello World!. There is your first program. You told the computer to do something, and it did what you told it to do.

Congratulations! You have just written your first computer program. My work is done here. You can add “Python programming” to your resume and get a job as a professional computer programmer!

Here is the same program in other languages:

+++++
In C:

```
#include <stdio.h>

int main(void)
{
  printf("Hello World\n");
  return 0;
}
```

+++++
In C++:

```
#include "std_lib_facilities.h"

int main()
{
        cout << "Hello World!\n";
        return 0;
}
```

+++++
In Java:

```
public class HelloWorld {
   public static void main(String[] args) {
      System.out.println("Hello, World");
   }
}
```

Notice how simple and readable the Python version in comparison to these other languages. The readability and simplicity are big reasons why Python is becoming more popular.

_____

Creating Python in a file:

So far, you have only seen a single line of Python code:

```
>>> print('Hello World!')
```

You typed it into the Shell and pressed Enter or Return to make it run. Typing one line at a time into the Shell is a great way to learn Python, and it is very handy for trying things out quickly. But soon we'll have you writing programs with tens, hundreds, maybe thousands of lines of code. The Shell is not an appropriate place to trying to write large programs. Python, like every other computer language, allows you to put the code that you write into a file and save it. Programs saved this way can be opened at any time and run without having to retype them. Here's how we do this in Python.

Just like any standard word processor or spreadsheet program, to create a new file, you go to the File menu, and select New File (denoted from here on as: File -> New File). You can also use the keyboard shortcuts of Control + N (Windows) or Command + N (Mac).

This will open up a new blank editing window, waiting for you to enter Python code. It behaves just like any text editing program that you have ever used. You enter your Python code line by line here similar to the way you were doing it in the Shell. However, when you press Return or Enter at the end of a line, the line does not run – it does not produce immediate results as it did in the Shell. Instead, the cursor just moves down to allow you to enter another line. You can use all the standard text editing features that you are used to: Cut, Copy, Paste, Find, Replace, etc. You can move around the lines of code using the arrow keys or by clicking the mouse. When a program gets long enough,

scrolling becomes enabled. You can select multiple lines using the standard click and drag, or click to create a starting point and shift click to mark an ending point.

Let's build a simple program containing three print statements. Open a new file. Notice that when you open the file, it is given name of "Untitled" in the window title. Enter the following:

```
print('I am now entering Python code into a Python file.')
print('When I press Return or Enter here, nothing happens.')
print('This is the last line.')
```

When you type the word print IDLE colorizes it (both here in the editing window and when you type it in the shell). This is IDLE letting you know that this is a word that it recognizes. IDLE also turns all the words enclosed in quotes to green. This also is an acknowledgement from IDLE that it has an understanding of what you are trying to say.

Notice that when you started typing, the window title changed to the name '*Untitled*'. The asterisks around the name are there to show that the contents of the file have been changed, but the file has not been saved. That is, IDLE knows about the new contents, but the contents have not yet been written out to the hard disk. To save the file, type the standard Control + S (Windows) or Command + S (Mac). Alternatively you can do a File -> Save. Since this is the first time the file is being saved, you will see the standard Save dialog box. Feel free to navigate to a folder where you will be able to find your Python files(s) or click on the New Folder button to create a new folder. In the top of the box where it says "Save as:" enter a name for this file. Since we are just testing things out, you can name the file "Test". However, Python files should always end with an extension of ".py". Therefore, you should enter the name "Test.py" in the Save As box.

NOTE: If you save your Python file without a ".py" extension, IDLE will not recognize it as a Python file. If Python does not know that your file is a Python file, it will not colorize your code. This may not seem important now, but will turn out to be very helpful when you start writing larger programs. So make it a habit right from the start to always end your Python file names with an extension of ".py"

Now that we have a saved Python file, we want to run or "execute" the statements in the file. To do that, select Run -> Run Module, or press the F5 shortcut key. If everything went well, the program should print the following in the Shell:

I am now entering Python code into a Python file.
When I press Return or Enter here, nothing happens.
This is the last line.

Now let's quit out of IDLE. Quitting is done using Control + Q (Windows) or Command + Q (Mac) keys. Alternatively, you could select IDLE -> Exit (Windows) or IDLE -> Quit IDLE (Mac).
When you are ready to open IDLE again, you have two choices. First, you could open IDLE by typing IDLE into the Start menu (Windows) or by double clicking the IDLE icon (Mac). If you then want to open a previously saved Python file, you can select File -> Open ... and navigate to the file you want to open.

However, if you want to open IDLE and open a previously saved Python file, you could navigate to the saved Python file (e.g., find the Test.py file that you just saved) and open IDLE by opening the file. On Windows, if you just double click on the icon, a window typically opens and closes very fast. This runs the Python program, but you cannot keep the window open. Instead, to open the file and IDLE, you need to right click on the file icon. From the context menu that appears, select the second item, "Edit with IDLE".

On a Mac, you can simply double-click on the file icon. If double clicking on the Python file opens some program other than IDLE, you can fix that with this one-time change. Quit whatever program opened. Select the Python file. Do a Command I (or File -> Get Info ...), which will open a long dialog, box. In the section labeled "Open with:", select the IDLE application (IDLE.app). Finally click on the Change All ... button below. Once you do that, IDLE should be able to double click on any file whose name ends in ".py" and it should open with IDLE.
Programming typically involves iterations of edits to one or more Python files. Each time you make changes and you want to test the new code, you must save the file, and then run it. If you don't save the file before you try to run, IDLE will prompt you, asking you to save the file. You will

quickly become familiar with the typical development cycle of: edit your code, save the file (Command or Control + S), and run the program (F5).

One other very nice feature of Python and IDLE is that the environment is almost completely platform-independent. That is, the IDLE environment looks almost identical on a Windows computer, on a Mac, and on a Linux system. The only differences you will see are those associated with the particular operating system (e.g., look of the window's title bar, location of menus, look of dialog boxes, etc.). These are very minor details. Overall, the platform that you run on does not matter.

But perhaps even more importantly, the code that you write is platform-independent. If you create a Python file on one platform, you can move that file to another platform and it will open and run just fine. I typically develop most of my Python code on a Mac, but I often bring these same files into classrooms and open them and run them on Windows systems.

Programming 101:

Now it's time to get into programming 101. This may be very basic, but I want to start right at the beginning. Please ask questions of you are not following, tangents can be good.

The two basic building blocks of programming are code and data. Code is a set of instructions that tell the program what to do in different circumstances. But we'll start our discussion with data.

Data is the quantities, characters, or symbols on which operations are performed with a computer. Examples of data include number of students in class, grade point average, name, whether a light switch is on or off.

Four types of Data:

There are many types of data, but in this class, we will deal with four basic types:

Numbers:
(Integers)
- Number of students in the room
- Person or team score in a game
- Course number
- Date in a month
- Temperature

(Floating point)
- Grade point average
- Price of something
- Percentages
- Irrational numbers, like Pi

String data (also called Text data):
- Name
- Address
- Course Name
- Title of a book, song, or movie
- Sentence
- Name of a file on the computer

Boolean (True or False)
- Light Switch on
- Inside
- Alive
- Listening

Notice that I split numbers into two different types: Integers (whole numbers) and numbers with a decimal point. In Python and most computer languages, numbers with a decimal point are called "floating point" numbers. These two types of numbers are handled very differently in the computer, and Python makes a clear distinction between them.

Let's talk in a little more detail about the different types of data, will be using four different types of data for most of what we will do in this course: Integer, Float, String, Boolean

++++ Integer Number – represents any numeric data (age, length, width, height, score, credits, ...). Can be positive, zero, or negative. 12, 50, 0, -3, -25

Floating Point number – represents any numeric data with a decimal point in it. 1.5, .5, -3.21, 1.0, 0.00000

String – represents textual data – a sequence of characters: first name, last name, street, city, state, part name, etc.

String data is always represented with quote characters before and after – in Python, you can use double or single quotes: "Irv", "Kalb", 'This is some string data', "OK". Both are OK. Also, you can embed a quote character inside the opposite type of quotes like this: "Here's some data". Look at the "Hello World" example again, this is a string value.

Boolean data can only have one of two values: True or False. These two words must be written with the first letter in uppercase. The easiest way to think of this is to use a name that describes the positive state: isOn, isInside, isListening. That way it is clear what True means and what False means.

++++
Here is a fake but typical form that you might see if you go to buy something online. Let's look at the fields in the form to see what type of data each represents:

As the end user, you would type characters into each of these fields. But as the programmer who is writing this program, you have to think about what types of data you would use to represent the information that the user entered into these fields.

Name and address are strings. Just strings of characters
Number of Widgets represents an integer value, e.g., 10
Total to Pay represents a floating point number, e.g. 1.25
Receipt is what we commonly recognize as a checkbox. But if you were writing the program behind this form, you would represent the answer to the receipt question with a boolean, True or False

Variables:

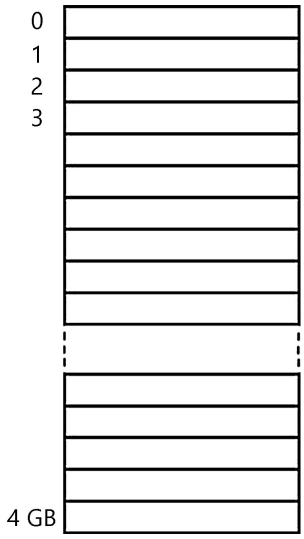
In programming, we need to remember and manipulate data all the time. This is a fundamental part of computer programming. The way to remember and manipulate data is to use a variable.

So, here is my first definition:

++++DEF: A Variable is a **named** memory location that holds a value.

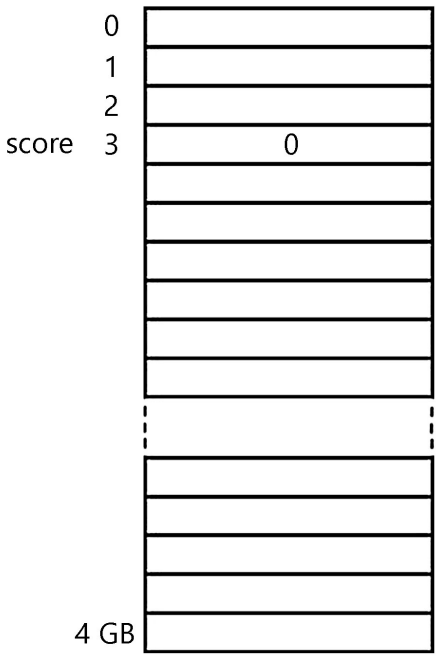
The contents of a variable can change or vary – where the word comes from.

You probably have heard the term RAM that stands for Random Access Memory. It is the active part of storage inside your computer. You can think of this memory as a simple list or array of numbered slots, starting at zero and going up to as much memory as you have in your computer. The amount doesn't matter, but here is a memory diagram showing memory starting a slot 0 and going up to slot 4 gigabytes.



In reality, behind-these-scenes, the way that Python stores data and variable names is more complex. For example, different types of data (integer, float, string, and Boolean) take up different amounts of memory rather than a single "slot". But thinking of each piece of data as being stored in a single slot in memory provides a good "mental model" – a good way to think about what a variable is.

++++Example: Imagine you are playing a computer game. The game typically has to keep track of your score. To do this, some programmer creates a variable, and the program changes the value of the variable over time. The programmer asks the language to create a variable, gives it a name, and puts some starting value into the variable. To keep a score, you typically start the value at zero, and every time something good happens in the game, the programmer's code adds to the value of the variable. If the game calls for it, the programmer's code can subtract from the value of the variable.



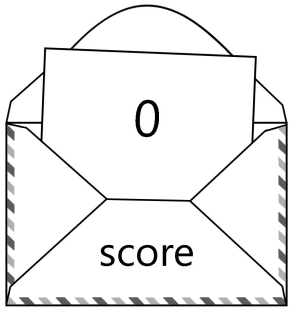
Another way to think of a variable is as an envelope or a box into which you can put in a value. A variable is a container - a storage space - with a name. The contents are the value. The name never changes, but the contents can change over time.

Using the example of a score, imagine that we have an envelope or box with the name “score” on it. Inside, we put the contents – a value.

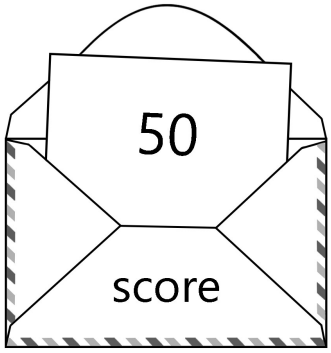
We have talked about variables and how they are used to store data, but I haven’t shown you yet how to use a variable in Python. Let’s do that right now.

So much for the theory. In practice, in Python, you create and use variables in an assignment statement.

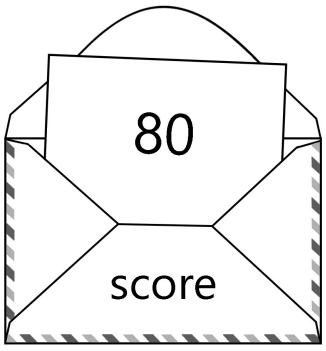
Using the example of a score, imagine that we have a box/envelope with the name “score” on it. Inside, we put the contents – a value. Lets start it off with zero.



If the user does something good: kills a bad guy, makes a good shot, finds a hidden item, etc., lets say the user gains 50 points. So, we take the current value (contents) of score, and add 50 points – the current value becomes 50.



Now, the user does something else good, and is awarded another 30 points. We take the current value of 50 and add 30 points to it, giving us a total of 80. So we have a variable called “score”, and its value changes over time. The computer remembers the current value – because it is stored in a variable.



Assignment statements:

So, we've talked about variables, but I haven't shown you yet how to use a variable in Python. Let's do that right now.

++++ DEF Assignment statement – an assignment statement is a line of code in which a variable is given a value.

++++ Assignment statement has the form:

<variable> = <expression>

(When I put things in angle brackets <>, it means that you should replace the brackets and the word with something that you choose – this is a placeholder.)

It works like this, the expression – everything on the right side of the equals sign is evaluated, a value is computed, and that resulting value is assigned to the variable on the left.

++++ Examples :

```
age = 29
name = 'Fred'   #notice that strings turn green
alive = True  #notice that True turns color, Keyword
gpa = 3.9
```

When Python executes an assignment statement like this, Python looks for a variable that was already used. If was never seen before, Python allocates a piece of memory, sticks a name onto it, and puts in the given value.

++++
In pure computer programming terms, the equals sign is not called equals, it is called the "assignment operator". In an assignment statement,

everything on the right of the equals sign is calculated, and the result is ASSIGNED to the variable on the left.

Whenever you see an assignment statement you can read or think of the equal sign by saying the words “is assigned to”, or “is given the value of”, or “is set to”. For example, it might be helpful and more clear to you to read this line:

+++++
age = 29

as:
    age is assigned 29
or:
    age is given the value of 29
or:
    age is set to 29
or:
    age gets 29
or:
    age becomes 29

+++++
After executing that line, enter this line and press Return or Enter:

age = 31

Python tries to do the same sequence of steps, but now it finds that there already is a variable named age. Rather than creating a new variable, Python overwrites the current value of the variable age with the new value of 31. If you remember the conceptual way of representing a variable as a container (for example, as an envelope), think of this line as replacing the old value inside the envelope with a new value. The variable name stays the same, but the contents change.

Variable Names:

Every variable must have a name. It is best to make names very descriptive. For example, let's say I was building a virtual aquarium, and I wanted to keep track of the number of fish in my aquarium. The computer doesn't care what you use for a variable name. You could use x as a name, or some odd sequence of characters, such as: xddqf. Or you could create a name like numberOfFishInAquarium. This is much more clear. It is a good idea to make your variable names as descriptive as possible. It makes code much more readable and understandable in the long run.

++++In Python (and all computer languages), there are rules for naming a variable. Here are Python's rules:

Must start with a letter (or an underscore)
Cannot start with a number
Can have any number of characters.
Can include letters, underscores, dollar signs ...
Cannot have spaces in it.
Cannot have math symbols (+, -, /, *, parentheses)

++++
DEF: Convention – an agreed upon way of doing things.

In programming, you can create any variable name you wish. as long as it follows the rules. However, when creating variable names, I strongly encourage you to use a 'naming convention' – that is, a consistent approach for creating names for variables. As a convention, if you create a name, like 'score', where the name is just one word, the convention is to use all lower case letters. However, we often want to create names made by putting together two or more words. Take for example, the variable name: numberOfFishInAquarium which is created by putting together five words. In the Python world there seem to be two common naming conventions.

++++The first convention, and the one that I prefer, is called "camel case". It deals with how to format a name that is made up of multiple words. The rules of the camel case convention are very simple:

The first word is all lower case. Then for every additional word in the name, make the first letter upper case, and all other letters lower case. Here are some examples

humanScore
computerScore
anotherVariableName
countOfBadGuys
bonusPointsForCollectingMushrooms

++++There is another convention that some Python programmers use, and that is to separate words with underscores:

this_is_a_variable_name
number_of_fish_in_aquarium

But this seems more difficult to me, so I will use camel case. I bring it up here because if you look at some code written in Python, you may see variables written this way. You may use whatever convention you want, but you should be consistent in naming variables. If you do programming in a company, the company may insist on a particular naming convention so all programmers will understand each other's code.

Python keywords:

I'm sure that you have heard that computers only understand ones and zeros. This is true. When you write code in Python, Python cannot run or "execute" your code directly. Instead, it takes the code that you write and "compiles" it, or converts your code into a language of ones and zeros that the computer can understand. Every language has such a compiler.
When the Python compiler reads your code, it looks for special words called "key words" to understand what your code is trying to say.

++++Here is a list of the Python keywords. Don't worry about the details right now. I'm just showing you these now because you cannot use these

as a variable name. If you try, the Python compiler will generate an error message:

and	if
as	import
assert	in
break	is
class	lambda
continue	not
def	or
del	pass
elif	print
else	raise
except	return
exec	try
finally	while
for	with
from	yield
global	True
False	None

Case Sensitivity:

The computer language called C was developed in the early 70's. It was one of the first "high level" computer languages, and has had a great deal of influence on many current computer languages. Languages such as: C++, JavaScript, ActionScript (language of Flash) and Java can all trace their roots to C. There are many similarities amongst these languages, but each one has a different purpose.

When C was created about 40 years ago, computers were very slow, and they wanted the language to be as fast as possible. One way to make the it fast was to have a rule that variable names and keywords would be case sensitive – case matters. Python has inherited this trait from C. So a variable named "Abcd" is NOT the same is "abcd". And print is not the same as Print. This will bite you many times over. You will spend a great deal of time scratching your head about why your program doesn't work.

This is why a naming convention makes sense. If you follow a naming convention, you will make fewer naming errors.

More Complicated Assignment Statements:

++++ Remember that the general form of an assignment statement is:

<variable> = <expression>

Look at this:

```
myAge = 31
yourAge = myAge
```

An assignment statement takes the value of the stuff on the right hand side, in this case, it is the variable myAge – which is 31, then assigns it to a new variable called yourAge. So, if a variable appears on the right side of an assignment statement, the evaluation is done using its value.

++++ Let's take this a little further:

```
numberOfMales = 5
numberOfFemales = 6
numberOfPeople = numberOfMales + numberOfFemales
```

First we use two assignment statements to set up two variables. In the third line, we evaluate everything on the right side of the equal sign. We see two variables, so to generate a result, we use the value of those variables, 5 and 6, do the math, get an 11, and assign that result into the variable on the left side of the equals sign.

This is NOT an equation like in math. We are NOT saying that these things are equal.

++++ To make this concept clearer, look at this:

```
myAge = 25
myAge = myAge + 1
```

The second line says, (on the right side), take the current value of myAge, add one to it, and put the newly calculated value back into the variable myAge. This statement changes the value of the variable myAge by adding one to it.

Print statements:

Now, how do we know if things are working? We would like to be able to output something to tell us what's going on in our program. Remember the print statement from Hello World?

++++ Print statement – used for debugging:

```
print(<whatever you want to see>)

yourAge = 21
print(yourAge)
myAge = 31
print(myAge)
```

And the computer prints whatever you tell it to print

+++

The print statement can also print multiple things on a single line if you separate them with a comma. Each comma is replaced by a space:

```
print(yourAge, myAge)
```

This can be used to nicely format output. This allows you to get a description of what value you are printing:

```
print('yourAge is', yourAge)
print('myAge is', myAge)
```

More assignment statements, with all types of data.

```
learningPython = True
print(learningPython)
priceOfCandy = 1.11
print(priceOfCandy)
myName = “Irv”
print(myName)
```

Now let’s try:

```
print(learningpython)          (<- notice lowercase p)
```

This generates an error message, a traceback. This is an error because learningpython is different from learningPython. Case is important. When you get a traceback error like this, look at the last line first to find out what is wrong. The wording will often tell you what went wrong.

This will be the cause of many early errors in your programming. This is why I strongly recommend using the camel case convention. If you follow the convention consistently, you won’t have as many of these types of errors.

print()

by itself will print a blank line. This is used when you have a lot of print statements, and you want to break them up a little.

Simple Math:

++++ Now let’s do some simple math. Python, and all computer languages allow a standard set of math operators:

+ Add
- Subtract
/ Divide
// Integer divide
* Multiply
() Grouping (we'll come back to this)
** Raise to the power of
% Modulo (or remainder)

Let's try some very simple math. In the shell, try out the following:

++++ Simple Math:

```
x = 9
y = 6
print(x + y)
print(x – y)
print(x * y)
print(x / y)
print(x // y)
print(x % y)
print(x ** y)
```

Computers can represent integers perfectly. They use base 2 instead of base 10 (like us humans), but that's not a problem. When they deal with floating point numbers, there must be some rounding, they cannot represent them perfectly (e.g., print(1.0/3.0)).

Raise to the power – the ** operator:

```
>>> x = 2
>>> y = 3
>>> print (x ** y)
```

8

Modulo:

Remainder of dividing:

<integer1> / <integer2>

Example:

```
ageInMonths = 29
years = ageInMonths / 12
months = ageInMonths % 12
print(years, 'years and', months, 'months')
```

2 years and 5 months

Order of Operations:

Back in middle school (I think), my math teacher went through a long description of a topic called the “order of operations”. We were told that some math operators had precedence over other ones. For example, look at this:

++++

```
x = 5 + 4 * 9 / 6
```

What operations are done in what order?

++++
I don’t know, I don’t care.

```
x = 5 + ((4 * 9) / 6)
```

I never learned that lesson. And in the programming world, you don’t have to. Instead use parentheses to group things and force the order of operations. When you use parentheses, the part in the innermost parentheses is always computed first. I strongly encourage you to group parts of expressions using parentheses.

++++++++

First Python Programs:

As we saw earlier, just like any word processor or spreadsheet program, to create a new file, you go to the File menu, and select New File. You can also use the keyboard shortcuts of Control + N (Windows) or Command + N (Mac).

+++++

Let's write a program that calculates how much money in a wallet.

But before we type any code, let's save the file. You save just like in an standard program using Command/Control S. Or do a file -> Save. In the File dialog box, give the file a name. The name must end in ".py". Enter a name like:

MoneyInWallet.py

and save it somewhere where you will be able to find it, for example on the desktop.

++++
Now, enter the following:

```
numberOfOneDollarBills = 3
numberOfFiveDollarBills = 2
total = numberOfOneDollarBills + (5 * numberOfFiveDollarBills)
print ('Total amount is', total)
```

When you type in this code, unlike using the Shell, the statements do not run, they just modify the file.

After entering the code, do a save (Command/Control S).

Now we are ready to run. To run the program, you can select Run -> Run Module, or just use the shortcut key: F5.

Notice that the output of your program comes out in the Shell window.

+++ Second example

Simple pay formula with overtime:

```
rate = 10.00
totalHours = 45
regularHours = 40
overTimeHours = totalHours - regularHours
pay = (rate * regularHours) + ((rate * 1.5) * overTimeHours)
print('For working', totalHours, 'hours, I should be paid', pay)
```

++++Pythagorean theorem:

$hypot^2 = side1^2 + side2^2$

OR

hypot = square root of ($side1^2 + side2^2$)

To write in Python (square root is raise to the one half power):

```
side1 = 3
side2 = 4
hypot = ((side1 ** 2) + (side2 ** 2)) ** 0.5
print(hypot)
```

Shorthand naming convention:

++++++
I have additional naming convention that I use. We use variables to keep track of the number of things all the time. I have developed a naming convention that I have been using for many years which is essentially a shortcut. Rather than writing out the words "number of" at the beginning of a variable name, I just use the letter 'n'. For example instead of:

numberOfStudents, numberOfGoodGuys

I would write:

nStudents, nGoodGuys

+++++
In class assignment:

Modify the MoneyInWallet.py program. Two changes:

1) Change all 'numberOf' to 'n'.
(Use IDLE's Edit -> Replace)

2) Add in ten dollar bills and twenty dollar bills and fifty dollar bills.

MoneyInWallet.py:

```
nOnes = 2
nFives = 8
nTens = 4
nTwentys = 5
nFifties = 8
total = nOnes + \
                    (nFives * 5) + \
                    (nTens * 10) + \
                    (nTwentys * 20) + \
                    (nFifties

print('Total amount is', total)
```

Notice in the assignment statement that does the calculation the line got a little long. If you think a line is getting too long to read, you can add a backslash character ("\") at a logical breaking point to indicate that the line should continue onto the next line.

Adding Comments:

Very often while writing code, you want to explain to the reader (who could be yourself or someone else), what the code is doing. To do this you can add a comment. Comments are completely ignored by Python – they are there only for humans. There are three ways to add comments in Python.

++++ 1) Start a line with the "#" character, followed by your comment:

```
#  This is a comment
#  All comment lines are ignored by the compiler
#  ----------------------------------
```

++++ 2) You can put a comment at the end of a line to explain details

```
score = 0     #  Initializing the score
```

++++ 3) Rare: Multiline comment

Start and a long comment block with three quotes """

```
"""
This is a long comment block
It is really long
It can be any length
"""
```