

Chapter 1: The Beginner's Tour Of Python

WHAT YOU WILL LEARN:

- **HOW TO USE PYTHON'S EDITOR**
- **THE DIFFERENCE BETWEEN IDLE AND THE SHELL**
- **ABOUT THE COLORS USED IN IDLE**
- **WHAT ARE BLOCKS OF CODE?**

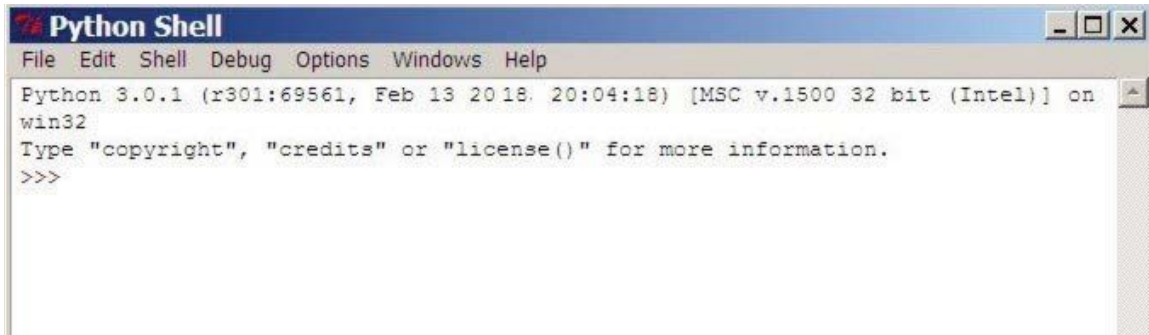
IDLE AND THE SHELL

First, you can start Python in Windows by clicking on **Start>Programs>Python 3.x>IDLE**. Go ahead, try it...I'll wait here. *IDLE is software that helps you to communicate with the computer.* It is on the outside of the main program, just as a snail's shell is on the outside of the snail. Whack on a snail's shell and he will get the message. IDLE works in the same way. Enter commands into IDLE and it will send the message to your computer. IDLE acts as your *interpreter* and translates what you say into a language that the computer can understand. If you really want to know, IDLE stands for **I**nteractive **D**eveLopment **E**nvironment. Why did they choose the L in the middle of the word, "development?" I have no idea and it's not important for the purposes of this book, so let's move on and get over it.

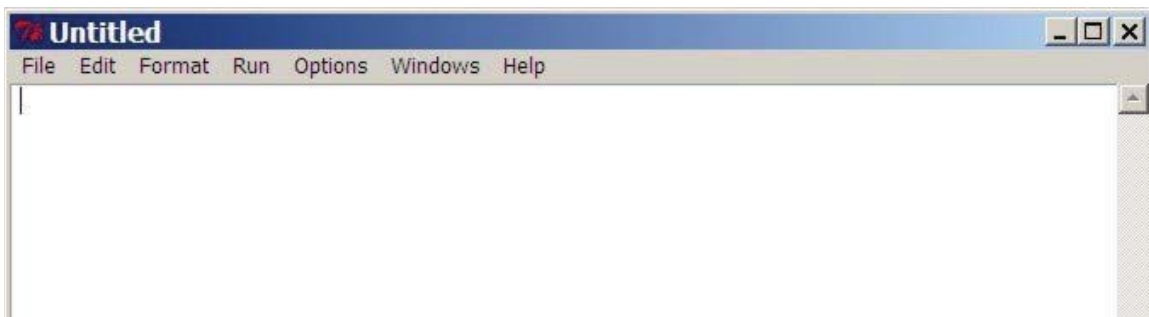
There are two windows to work from in IDLE. There is the **Edit Window** and the **Shell Window**. The Python Shell window will say, "Python Shell" at the top of the window, while the Edit window will say, "Untitled" and have a "run" command listed on the top menu bar. If Python starts in the Shell Window and you want to use the Edit Window, just choose **File<New Window** and it will open an Edit Window. If you want to use the Shell Window if the Edit Window starts go to the "Run" menu at the top of the window and choose "Python Shell." The *Python Shell is an interactive interpreter*. This means that when you press the enter key, it checks your source code and may give you some feedback. If you see (`>>>`), this is the first command prompt. It is letting you know the interpreter is patiently waiting for you to type something. If you see (...), this is the

secondary prompt waiting for you to type something more. If you found these two windows they should look something like this:

The Python Shell looks like this:



The editor looks like this:



There are other editors that you can use for programming but to keep things simple in this book we will use the IDLE software that is packaged with Python.

Why are there two windows in IDLE and how do I use them? Python allows you to work in script mode or in interactive mode. What's the difference?

Script Mode is great for writing programs you can save and run later. It is generally used for the final product.

Interactive mode is for testing and trying small ideas quickly.

Most people use both of these together. Script mode for working on their main program and interactive mode for trying new ideas in the same way you would use scratch paper.

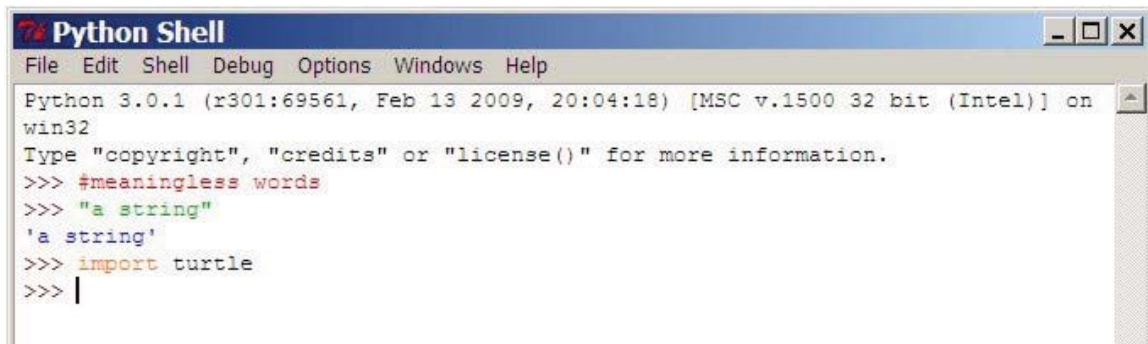
A third function of IDLE is that it is also a debugger. You can find the debugger button at the top of the Shell window. What's a debugger? It's a program to help you kill bugs. No not the one crawling up your leg; but the bugs, or problems, that are in the code we are working on.

IDLE Colors

Did you notice that IDLE changes the colors of your text? What's the meaning? Let's look at a list of some of the most common syntax colors and their meanings in IDLE.

COMMON PYTHON SYNTAX COLORS:

Keywords	orange
Strings	green
Comments	red
Definitions	blue
Misc. Words	black



```
Python Shell
File Edit Shell Debug Options Windows Help
Python 3.0.1 (r301:69561, Feb 13 2009, 20:04:18) [MSC v.1500 32 bit (Intel)] on
win32
Type "copyright", "credits" or "license()" for more information.
>>> #meaningless words
>>> "a string"
'a string'
>>> import turtle
>>> |
```

You needn't memorize these at this point, but knowing the meaning of the colors can help you see clearly where you typed something wrong. If you were trying to type a string and its not green, you probably forgot the quotation marks or did something else the computer didn't like. The colors can also be changed according to your personal preferences and may vary in different editors.

BLOCKS OF CODE

While we are looking at the windows, how is your text arranged? The text is arranged in lines, groups, and blocks. A **block** is just a group of code that goes together. Like a city block, it can be divided into smaller groups like houses on the block, cars on the block, dogs, lions, etc. on that block. This concept is important to tell the computer how to read and follow your code. To a computer, arranging your code with the proper grouping of lines and blocks is like a map that says; "first do this, second do that, or repeat this. "

Blocks are one way we dictate the running order in programming. You can have blocks in blocks just as you can have groups in groups. You may have a group of students under

100 years old. But, within that group you may have another group called “girls.” Inside of that group you may have another group or block called “girls with green hair.” Blocks make it easy to keep things or instructions in our code together in their correct group. This helps us refer to them and to direct the computer to use that group of instructions, in the order we want the computer to use them. Blocks are defined by the number of spaces used to indent each line of code. In the following examples I will use dots to show you clearly how many spaces would be in the code. This; “...” refers to three spaces. Use spaces, not dots, when you type your code.

Note: The color code in the following examples is not a part of IDLE. These colors are used to emphasize what parts of a block belong together.

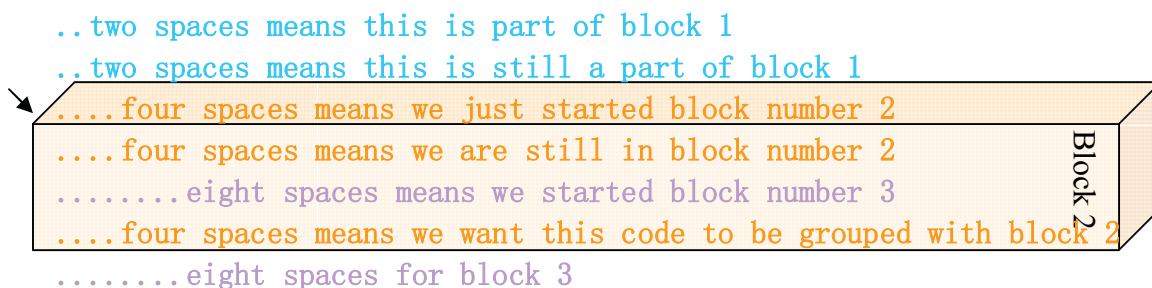
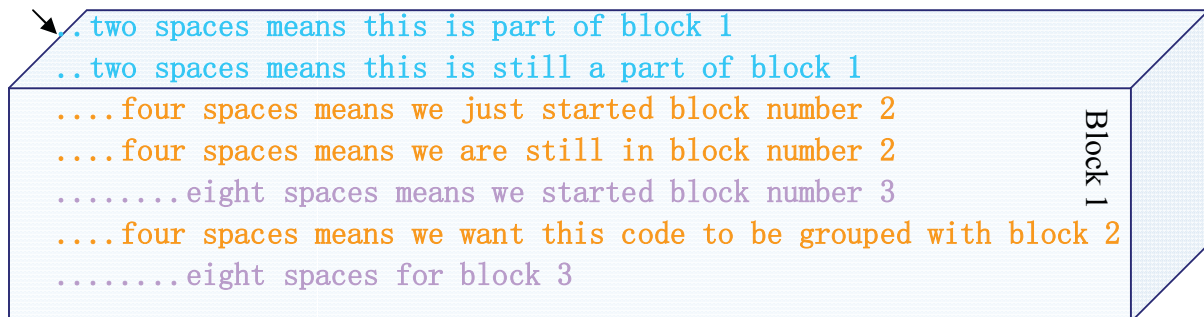
```

2 spaces  ..students under a hundred years old (Block 1)
4 spaces  ....girls (Block 2)
8 spaces  .....girls with green hair (Block 3)

```

The number of times we indent helps the computer know how we are grouping the information. (Hang in there, a few more ideas and you will make your first game).

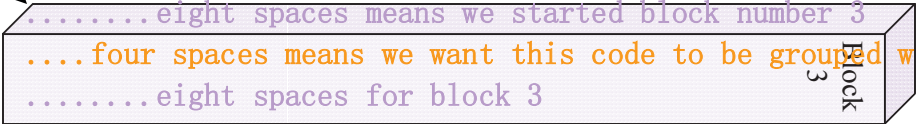
Here is another example;



```

..two spaces means this is part of block 1
..two spaces means this is still a part of block 1
....four spaces means we just started block number 2
....four spaces means we are still in block number 2
.....eight spaces means we started block number 3
....four spaces means we want this code to be grouped with block 2
.....eight spaces for block 3

```



Whatever the number of spaces you choose, keep it consistent so you don't get confused. *Why do we care about blocks and grouping programming statements?* Ah, I'm glad you asked, but I won't tell you until later. (Don't worry you will get to play with these shortly).

VOCABULARY REVIEW: *(Yes, you should read these again)*

Algorithms are sets of instructions that tell the computer what to do.
Block is just a group of code that goes together.
high level language is a computer language that is closer to human language and easier for us to use than low level or machine languages.
IDLE is software, (an editor), that helps you to communicate with the computer.
Interactive mode is for testing and trying small ideas quickly.
Low level languages are used when the programmer wants more direct control over the machine he is using.
Programming is the art and science of making the computer do what you want it to do by creating programs.
Programs are algorithms and source code packaged together to achieve your objective(s).
Python Shell is an interactive interpreter.
Script Mode is great for writing programs you can save and run later. It is generally used for the final product.
Source code is all the algorithms, statements, and instructions that we used in a program.

Chapter 2: You are now a programmer!

WHAT YOU WILL LEARN:

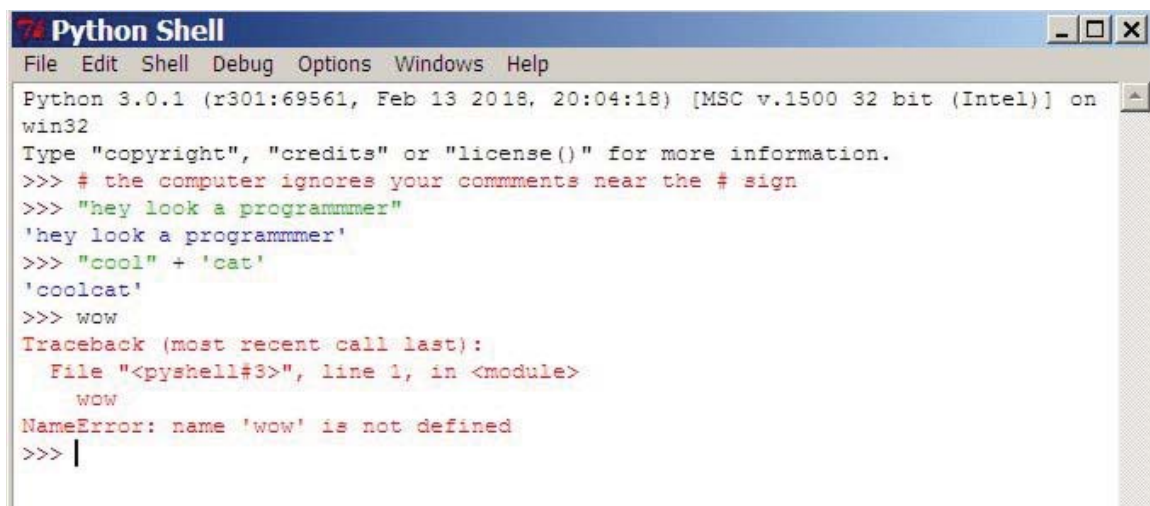
- **YOUR FIRST PROGRAM**
- **STRINGS**
- **VARIABLES**
- **OPERATORS**
- **KEY WORDS (TO USE OR NOT TO USE)**
- **BOOLEAN DATA TYPES**

LET'S START PROGRAMMING

Ok, let's start playing...err... programming. Open your Python Shell window to type some things. Type the following code exactly as you see it. *Then hit the enter key after each line.*

```
# the computer ignores comments near the # sign
"hey look a programmer"
"cool" + 'cat'
wow
```

If you entered these correctly, it should look like this:



```
Python Shell
File Edit Shell Debug Options Windows Help
Python 3.0.1 (r301:69561, Feb 13 2018, 20:04:18) [MSC v.1500 32 bit (Intel)] on
win32
Type "copyright", "credits" or "license()" for more information.
>>> # the computer ignores your comments near the # sign
>>> "hey look a programmer"
'hey look a programmer'
>>> "cool" + 'cat'
'coolcat'
>>> wow
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    wow
NameError: name 'wow' is not defined
>>> |
```

If you type the # key in front of some text, the computer ignores you. This is useful for adding comments in your programs that will not be misunderstood as instructions by the computer; or if you just feel like being ignored.

STRINGS

If you noticed I typed the words “hey look a programmer” in quotation marks. Later I used; ‘ ‘ and got the same exciting result. When we hit enter, the computer just repeated what we typed back to us. It printed the characters as output on the screen. A sequence of characters, words, or sentences like this one is called a *string*. When we use single or double quote symbols to tell the computer what is in our string, we call these quote symbols “*delimiters*.” We tell the computer that we are entering or ending a string by using single or double quotation marks. The computer don’t care which kind you use at this point. We can now say we have declared or “delimited” the string. In IDLE strings are green and the output here is blue. The error message is red.

You can also add strings together using a math operator. We will talk more about that in a moment. Putting the string “cool” with ‘cat’ produced ‘coolcat’ . If I want a space between them when they are added together, I should add one in my string; “cool “ + “cat” or “cool” +” cat” would generate ‘cool cat’ .

What happened when I typed, “wow?” The computer said; “Blah blah blah...is not defined.” This is the computer’s way of saying; “huh? I don’t understand.” Python attempts to give you an idea of what went wrong. When I typed the word, I did not include it inside of quotation marks to tell the computer that I was entering a string. So, the computer went, “Huh?”

Have you ever asked; “When am I going to use this kind of math?” In programming you should remember some simple math concepts to make your life easier. Don’t worry, I’ll be brief. To begin with, the world of math has animals called “operators” and “variables.”

VARIABLES

Variables are like little like boxes or containers to put different things in. In math your teacher may have told you that $1 + x =$ some other number. The 1 is an *integer*, (a complete number as opposed to part of a number like $\frac{1}{2}$), and the x is a variable. In programming, you get to name your variable anything you want. You can create an imaginary box with anything you want to put in it, and define/label that imaginary box, (or variable), by any name you choose.

Let’s try it. Let’s imagine a box of chocolate. We want to tell the computer that the box labeled “chocolate” has happiness and joy inside of it. To do this we use the = sign to define what a variable means for the computer.

In the Python shell window type:

```
Chocolate = "happiness and joy"
```

Hit the enter key.

Now, let's type the word without quotes;

```
Chocolate
```

Hit enter.

Your computer should reply, 'happiness and joy'

You just *defined* a variable. This is very useful in programming. You will constantly be teaching the computer how to think as you write programs.

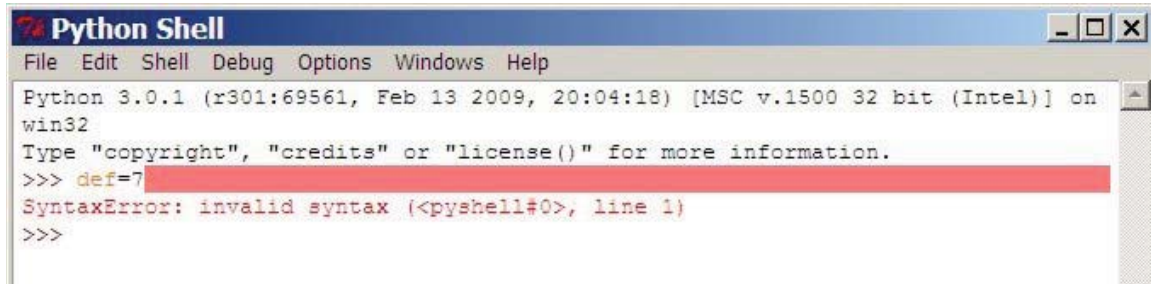
Variables are chunks of data stored in the computers memory. There are generally three types of data stored in variables. Variables can be in the form of **integers** or in a **string** as mentioned previously. The second type of data, called a **float**, refers to the *non-whole numbers like decimals*. Remember to use the = sign to assign a variable. If you want the computer to think the variable *x* means 5 is in the box named *x*, then you type:

```
x=5
```

Now the computer holds a 5 in memory and when you type an x, it tells you '5' if you hit the enter key. You can name a variable almost anything and use symbols like the _ underscore. But there are some rules. You can't use special key words that Python understands as having special meanings. Don't use these words:

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

If you should accidentally, (or just out of rebellion), use these words something like this will happen:

A screenshot of a Python Shell window. The title bar reads "Python Shell". The menu bar includes "File", "Edit", "Shell", "Debug", "Options", "Windows", and "Help". The main text area shows the following content:

```
Python 3.0.1 (r301:69561, Feb 13 2009, 20:04:18) [MSC v.1500 32 bit (Intel)] on
win32
Type "copyright", "credits" or "license()" for more information.
>>> def=7
SyntaxError: invalid syntax (<pyshell#0>, line 1)
>>>
```

“`SyntaxError: invalid syntax...`” is Python’s ways of saying, “Hey, you don’t know what you are talking about and neither do I! Speak Python!” Don’t take it personal, as a new programmer you will get used to these insults. You will find your computer has an attitude.

Other than these minor rules, you can name a variable anything you want. To sound more like proper geeks, we don’t always call them “names,” we sometimes call them “identifiers.” Remember *identifiers are just names*.

OPERATORS

Operators do something, like add, multiply, divide, subtract, or compare.

Notice that we already used the = sign to assign identifiers. So, we can’t use the equal sign as an operator. Instead we must use == to mean; “equals.” *What other useful things can be done with operators?* In the Python shell type:

“words words words words words”

Hit enter

Now type:

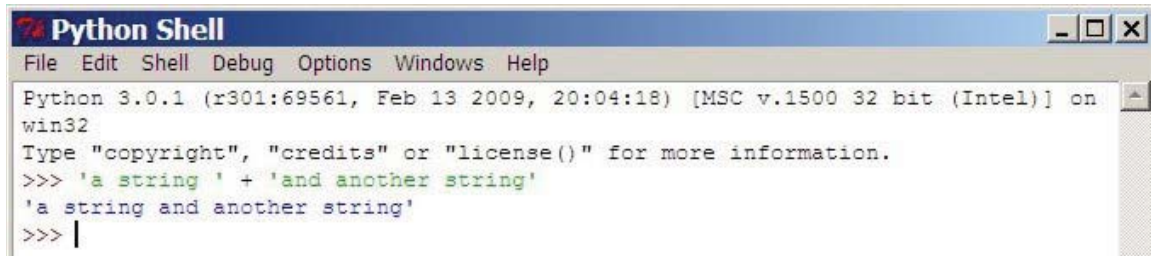
“words ” *10

Hit enter.

Which is easier? (I could have really used this all those times in primary school when I had to write a few hundred sentences). Operators are life saving tools for the programmer. The * symbol is used to express multiplication in Python. You also had to add a space at the end of “words “ before the closing quotation marks or your 10 words look like one long one. The most common operators are:

- + for addition
- for subtraction
- / for division
- * for multiplication

You can also use operators with strings of data. Try it. Make two strings with any words you want between quotation marks and add them together. Like this:

A screenshot of a Python Shell window. The title bar reads "Python Shell". The menu bar includes "File", "Edit", "Shell", "Debug", "Options", "Windows", and "Help". The main text area shows the following: "Python 3.0.1 (r301:69561, Feb 13 2009, 20:04:18) [MSC v.1500 32 bit (Intel)] on win32", "Type 'copyright', 'credits' or 'license()' for more information.", ">>> 'a string ' + 'and another string'", "'a string and another string'", and ">>> |".

```
Python 3.0.1 (r301:69561, Feb 13 2009, 20:04:18) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 'a string ' + 'and another string'
'a string and another string'
>>> |
```

We also have *comparison operators* for...you guessed it; *comparing things!*

Here are some of those:

- > for greater than
- < for less than
- <= less than or equal to
- >= greater than or equal to
- == equal to
- != not equal to

Remember that we already used the = symbol to define or tell the computer the meaning of our variables. So we don't confuse our little computer, we must use == to express the traditional meaning of "equal to."

Now try some of these until you get bored. Any math expression will do. But wait! Use only integers; no decimals at this point.

Due to the way a computer deals with *floating point numbers*, (*numbers with a decimal point*); you won't always get the same results as a calculator. It is too complex to get into for a beginners book, so we will have to skip the explanations for now.

For now, remember that Python is not your math teacher; although they may look similar. In math, 7.0 is still an integer because numbers like 0, 5, 177, are integers. But computers are not so clever, so in Python the number 7 is an integer but since 7.0 includes a decimal point; it's called a floating point number.

LOGICAL OPERATORS

The words “*and*, *or*,” and “*not*” are *logical operators*. A *simple condition* is a comparison that only uses two values:

$9 < 19$

A *compound condition* is a comparison using more than two values:

$x < 10 \text{ and } x > 5$

These logical operators generally mean the same thing in programming as they do in English. At this level, I will only introduce you to these as new terms and concepts.

BOOLEAN DATA TYPES

What’s a Boolean, something you eat? You and the computer sometimes need to know if something is true or not. A *Boolean data type* is referring to two possible values; **True** or **False**.

Boolean expressions are sometimes called *conditional expressions* because they are based on the condition that something is either true or false.

Let’s play with some of this new knowledge. Open IDLE and type the following, but feel free to play with your own ideas after you try these:

$8 > 3$

Hit enter.

Ah, the computer agrees, it says, “**True.**”

$3 > 8$

Hit enter.

Hey! The computer just said, “**False.**”

He better watch it, I just found the off switch! (I told you the computer has an attitude)! Boolean values or “True and False” remarks are very useful in programming.

VOCABULARY REVIEW:

<i>Boolean data type is referring to two possible values; True or False.</i>
<i>Comparison operators are for comparing things, (<, >, ==, !=, etc.).</i>
<i>Compound condition is a comparison using more than two values, (x < 10 and x > 5)</i>
<i>Conditional expressions (also called Boolean expressions), are based on the condition that something is either true or false.</i>
<i>Delimiters quotation marks to tell the computer we are entering a string.</i>
<i>Float non-whole numbers like decimals..</i>
<i>Floating point numbers numbers with a decimal point</i>
<i>Identifiers are names</i>
<i>Integer a complete number as opposed to part of a number like ½</i>
<i>Logical operators the words “and, or,” and “not”</i>
<i>Operators do something, like add, multiply, divide, subtract, or compare.</i>
<i>Simple condition is a comparison that only uses two values, (9 < 10)</i>
<i>String a sequence of characters, words, or sentences in quotation marks.</i>
<i>Variables are like little like boxes or containers to put different things in.</i>

Chapter 3: Your First Game!

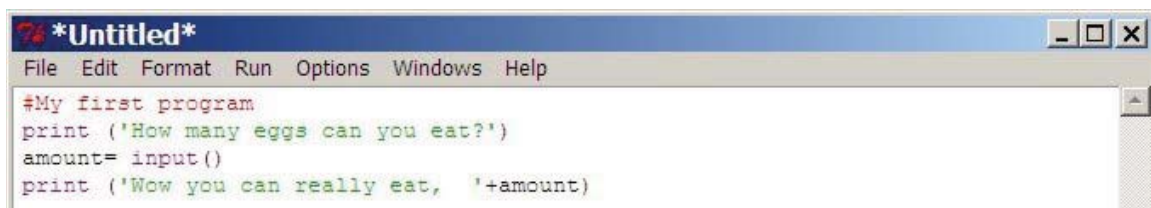
WHAT YOU WILL LEARN:

- **SAVING YOUR PROGRAM**
- **THE MEANING OF SOME CODE**
- **CONDITIONAL STATEMENTS**
- **WHILE STATEMENTS**
- **LOOPS**

SAVING YOUR FIRST PROGRAM

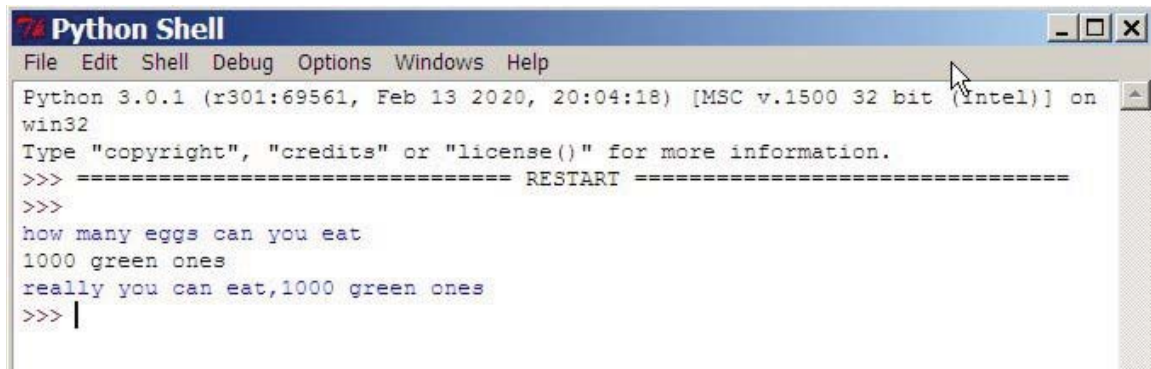
To help you understand your first game you will write a smaller program first. Now you will work in the editor window and learn how to save your file. Open the *editor window*, **not** the *shell window*, and type:

```
#My first program
print( 'How many eggs can you eat?' )
amount=input()
print( 'Wow you can really eat, ' +amount)
```



Save your program and give it a name, but add `.py` after the name you choose. Then close Python. Reopen IDLE and go to *file*, then *recent files*, and choose your program name. Go to the run button at the top of the edit window and choose “Run module.” When it asks you, “How many eggs can you eat?” type a number or a word.

It should respond and look something like this:



```
Python Shell
File Edit Shell Debug Options Windows Help
Python 3.0.1 (r301:69561, Feb 13 2020, 20:04:18) [MSC v.1500 32 bit (Intel)] on
win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
how many eggs can you eat
1000 green ones
really you can eat,1000 green ones
>>> |
```

If your program didn't work, go back and make sure everything is typed exactly as I did typed it, and make sure you did typed in the right windows. Before we go on to your first mini game I will explain more about the program you just made.

EXPLAINING THE CODE

```
#My first program
```

This first line of your program was just for you. Remember that the computer ignores comments that begin with the # sign.

```
print( 'How many eggs can you eat?' )
```

This line was to tell the computer the words you wanted it to show or print on the computer screen.

```
amount=input()
```

The above line was to do two things. We made up the name of a variable and called it "amount." We also used the `input()` command to tell the computer that the computer user would put some data into the computer.

```
print( 'Wow you can really eat, ' +amount)
```

This line was to tell the computer to print these words on the computer screen but to add our variable. Our variable was called amount and now contains in memory whatever you typed for the input.

Change the lines above and make up your own mini programs to practice and see what happens. When you are ready, go on to the next part and make your first mini game.

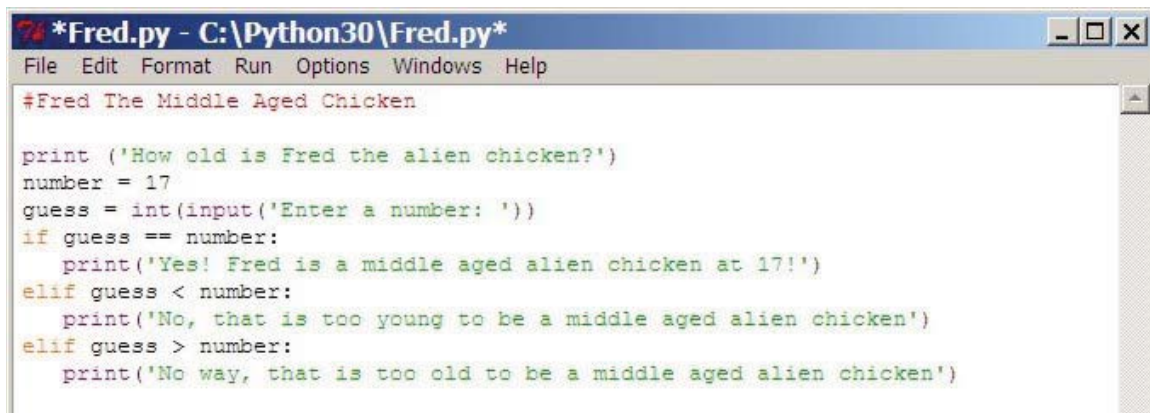
CONDITIONAL STATEMENTS

As a programmer you will often find yourself needing to use *a statement based on a condition*. If you eat 2000 candy bars: you will get sick. This kind of conditional statement is called an “**if**” statement.

We will use the **if** and the **elif** statements to make your first simple game. Let’s name your game; “Fred; The Middle Aged Alien Chicken.”

To start programming I will make a note of the name of my game at the top of the window as a reminder using the # key.

Enter the following code and then we can play and explain it:



```
*Fred.py - C:\Python30\Fred.py*
File Edit Format Run Options Windows Help
#Fred The Middle Aged Chicken

print ('How old is Fred the alien chicken?')
number = 17
guess = int(input('Enter a number: '))
if guess == number:
    print('Yes! Fred is a middle aged alien chicken at 17!')
elif guess < number:
    print('No, that is too young to be a middle aged alien chicken')
elif guess > number:
    print('No way, that is too old to be a middle aged alien chicken')
```

Make sure you are in the right window and that you type everything exactly as you see above. Then click on *run* and *run module* to try it. Enter a number and it will give you a reply. Enter another number and ...ahh!! it just repeats what you entered! You close the game and run it again to try again. So, at this point, you can only make one guess before the game replies and stops working. Don’t worry; we will fix that in a moment when you learn about loops. For now, I will explain the code we have so far:

```
#Fred The Middle Aged Chicken
```

Again the # sign makes this information to ignore for the computer, but a reminder or information for us humans and aliens.

```
print ( 'How old is Fred the alien chicken?' )
```

This tells the computer to print the string of characters inside the (“ ”) to the screen.


```
number=17
```

This tells the computer that the variable we called “number” will mean “17.”

```
guess=int(input( 'Enter a number:' ))
```

This tells the computer that the variable we call “guess” will mean; we want an integer input from the computer user. A string of characters, ‘Enter a number,’ to make the request is inside the inner set of parenthesis. *What’s with the parenthesis in parenthesis?* This tells the computer what to do first. Just as in math, the parenthesis are done from the inner to the outer ones; the inner pair first, followed by the outer pair. Here, the string (‘Enter the number’) will be seen by the computer before the (input()) is expected. The inner parenthesis in (input()) has the meaning of (input (something here)). So, the computer displayed, “Enter a number” and waited for us to input an integer before going on.

```
if guess==number:
```

This was to tell the computer that if the variable we called “guess” really is equal to the integer typed in the input : (then) do... The : sign means “then” in Python.

```
print( 'Yes!, Fred is a middle aged alien chicken at 17!' )
```

This tells the computer to print or put the string inside the (‘) on the computer screen. Notice that this and all the print commands are indented the same amount. This will become important soon when we explain how to create blocks of code that tell the computer what goes together. We will do this to teach about loops and how to fix the little problem of our game only allowing one guess before it goes stupid and just repeats what we type again.

```
elif guess<number:
```

Here’s a new one. *What’s an elif?* No, not that. It means “or else if...” Here, we are telling the computer that in the past a guess was made and we told it what to do; but if what we said may happen didn’t happen; here is something else to do. So, (elif), the “guess” variable, is less than the “number” variable we told the computer meant “17.” Read that again if you didn’t get it. Think of Python as an old man with the ability to think like Einstein but the slurred vocabulary of a drunken two year old. Not to criticize it; this is one of the ingenious aspects of Python. Saying little with a powerful meaning can help us make powerful programs by writing code within a very short time.

```
print( 'No, that is not too young to be a middle aged alien chicken' )
```

Here we indent and tell the computer to print a string to the screen again.

```
elif guess > number:
```

This is a repeat of the elif above, but changing the meaning to; “if the guess is greater than the number then...”

```
print( 'No way, that is too old to be a middle aged alien chicken' )
```

Again, we indented and told the computer to print a string of characters.

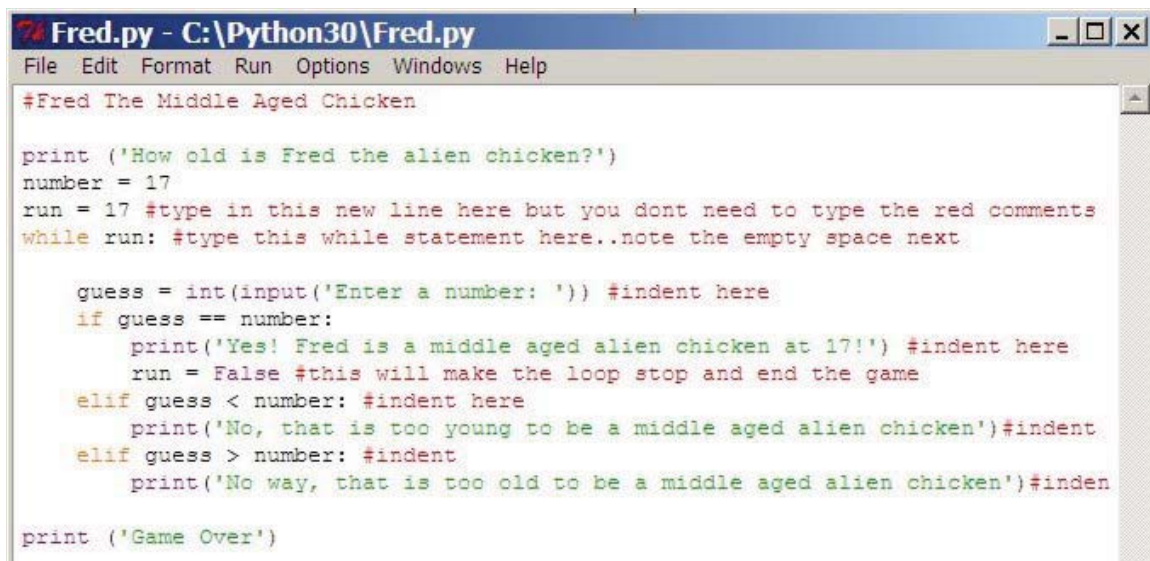
Remember pages ago when we talked about blocks? You were younger then, but maybe you remember. Well, we need to review that stuff and expand on the meaning to be able to teach you about loops and how to fix our games problem of stopping after one guess.

LOOP FOR “AWHILE” STATEMENTS

If you haven't guessed it, a **loop** is when the computer goes back and repeats something in a cycle until something breaks it out of that cycle. The **while statement** tells the computer to continue looping while some condition is continuing to happen; usually while it is still true.

As long as some condition is true in a while statement, a block of statements or code will be repeated. It will break out of that loop when the condition is no longer true for that block of code.

Let's fix our game. Change your code to look like this:



```
Fred.py - C:\Python30\Fred.py
File Edit Format Run Options Windows Help

#Fred The Middle Aged Chicken

print ('How old is Fred the alien chicken?')
number = 17
run = 17 #type in this new line here but you dont need to type the red comments
while run: #type this while statement here..note the empty space next

    guess = int(input('Enter a number: ')) #indent here
    if guess == number:
        print('Yes! Fred is a middle aged alien chicken at 17!') #indent here
        run = False #this will make the loop stop and end the game
    elif guess < number: #indent here
        print('No, that is too young to be a middle aged alien chicken')#indent
    elif guess > number: #indent
        print('No way, that is too old to be a middle aged alien chicken')#inden

print ('Game Over')
```

Most of this code was already explained, so we will just look at the changes and the new things. You didn't need to type any of the red comments for your code to work. These were just there to help you. The spaces are also ignored by the computer and are placed in code to make it easier for humans to read. The indentations are important changes. We will discuss those in a moment. Now, if you try the game, you can continue guessing until you guess correctly. The game will only end when you guess the correct answer. Try it.

The code:

```
run=17
```

Here we named another variable called "run," and told the computer that it also means "17."

```
while run:
```

The *while* statement is known as a *looping* statement. It tells the computer to repeat the next block of code as long as it continues to be true. Notice the empty line after the line of code with the while statement. In this example, I used a blank lines before and after the block of code that the while statement applies to. This entire block will continue to repeat forever until the run variable becomes false. So, the meaning of this would be to repeat the next block of code while the run variable is true; "then, (:), do this..." Here we can add `if` or `elif` statements to tell the computer when our run variable is true or when our run variable becomes false. If the run variable becomes false, our while run loop is no longer true and will stop looping/repeating.

```
guess=int(input( 'Enter a number: ' ))
```

Here we created another variable called "guess." We told the computer that when we use this word we mean the integer input that a person will type when he sees the string; ('Enter a number: '), on the computer screen.

```
if guess==number:
```

This means that if the guess, (input), really equals, (==), the number, (17), then, (:)...

```
print( 'Yes! Fred is a middle aged...' )  
run=False
```

These two statements are indented equally to tell the computer to run them if the `guess == number` happens. This tells the computer that if this happens and the person inputs the number 17, then do the following indented things/lines of code. If this happens first print the string in parenthesis to the screen, then make the meaning of our run variable; “false.” This causes the while statement to now be false and the computer knows to stop repeating this loop and jump out of this set of indented lines, or this block of code, and continue on the next block, (indicated by less indentation). That would make it jump to the print statement with the string (‘Game over’).

If the number 17 is not guessed/our input, this would make the while statement continue to be true. So, the computer should continue and repeat the while block again by going to the next line in the block:

```
elif guess < number:
```

This means, “or else if the guess is less than 17 then, (:)...

```
print( 'No, that is too young...' )
```

Print the sting in parenthesis to the computer screen.

```
elif guess > number:
```

Or else if the guess is greater than 17 then...

```
print( 'No, that is too old...' )
```

Print the sting in parenthesis to the computer screen.

```
print( 'Game over' )
```

This statement will not run as long as the computer is looping the while block of code, because we let the computer know this is outside the while block by not indenting it. It will only run when our while block become false because 17 was the input.

Whew! Now you can understand how Python can save you time. Look at how many English words it just took to explain what we were doing. Then compare that to how short the Python blocks of code are:

VOCABULARY REVIEW:

If Statement a conditional statement/ a statement based on a condition.

Loop when the computer goes back and repeats something in a cycle until something breaks it out of that cycle.

While statement tells the computer to continue looping while some condition is continuing to happen; usually while it is still true.