



Copyright 2010 © by Jody Scott Ginther
CEO of Alien Cat ® Studios, registered Trade Mark
Dedicated to Family Entertainment and Education

“Start Here: Python Programming”
is licensed under a Creative Commons License.
Details can be found at: <http://www.creativecommons.org>

You are free:



To Share - to copy, distribute, display, and perform the work



To Remix - to make derivative works

Under the following conditions:



Attribution - You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work). (Visibly include the title and author's name in any excerpts of this work.)



Noncommercial — You may not use this work for commercial purposes.

Author's websites:

www.AlienCatStudios.com
www.toonzcat.com

This page intentionally left blank just to bother people;

You will always wonder...

Table Of Contents

Introduction.....	1
<i>Who Should Read This Book?.....</i>	<i>1</i>
<i>Who's The Author?.....</i>	<i>1</i>
<i>What's A Programming Language?.....</i>	<i>2</i>
<i>What Is Python?.....</i>	<i>2</i>
<i>How To Use This Book.....</i>	<i>3</i>
<i>Getting And Installing Python.....</i>	<i>3</i>
 Chapter 1: The Beginner's Tour of Python.....	 4
<i>IDLE and The Shell.....</i>	<i>4</i>
<i>IDLE Colors.....</i>	<i>6</i>
<i>Blocks of Code.....</i>	<i>6</i>
 Chapter 2: You Are Now A Programmer!.....	 9
<i>Let's Start Programming.....</i>	<i>9</i>
<i>Strings.....</i>	<i>10</i>
<i>Variables.....</i>	<i>10</i>
<i>Operators.....</i>	<i>12</i>
<i>Logical Operators.....</i>	<i>14</i>
<i>Boolean Data Types.....</i>	<i>14</i>
 Chapter 3: Your First Game!.....	 16
<i>Saving Your First Program.....</i>	<i>16</i>
<i>Explaining The Code.....</i>	<i>17</i>
<i>Conditional Statements.....</i>	<i>18</i>
<i>Loop For "Awhile" Statements.....</i>	<i>20</i>
 Chapter 4: Words Are Boring, Let's Get Graphic!.....	 24
<i>Introduction To Modules.....</i>	<i>25</i>
<i>Lists.....</i>	<i>28</i>
<i>Slices.....</i>	<i>32</i>
<i>Tuples.....</i>	<i>34</i>
 Chapter 5: Functions.....	 36
<i>What's A Functions?.....</i>	<i>36</i>
<i>How Do I Create/Define My Own Functions?.....</i>	<i>37</i>
<i>Scope: Global or Local.....</i>	<i>40</i>
 Chapter 6: Python Parts.....	 43
<i>Modules and Object Oriented Language.....</i>	<i>43</i>
<i>Standard Library Modules.....</i>	<i>44</i>
<i>Dictionaries.....</i>	<i>44</i>

Chapter 7: Other Stuff.....	46
<i>More Keywords and Expressions.....</i>	<i>46</i>
<i>Debugging and Exception Handling.....</i>	<i>47</i>
<i>Now What?.....</i>	<i>47</i>
 Index.....	 49
<i>Review Dictionary of Terms.....</i>	<i>49</i>
<i>Python's Many Flavors.....</i>	<i>51</i>
<i>Python General Resources.....</i>	<i>51</i>
<i>Other Libraries and Modules May Be Found At:</i>	<i>52</i>

Copyright 2010 © by Jody Scott Ginther
CEO of Alien Cat ® Studios, a registered Trade Mark
Dedicated to Family Entertainment and Education

www.AlienCatStudios.com
www.toonzcat.com

*This book is dedicated to my father;
who inspired me to teach myself.*

Introduction

WHO SHOULD READ THIS BOOK?

- **ANY BEGINNER WHO WANTS TO LEARN PROGRAMMING IN PYTHON**
- **TEACHERS OR ANYONE WHO WANTS TO TEACH PYTHON TO BEGINNERS.**
- **PARENTS WHO WANT THEIR CHILD TO STUDY PROGRAMMING**
- **PEOPLE WHO ARE CURIOUS**
- **PEOPLE WHO ARE BORED AND HAVE NOTHING BETTER TO DO**
- **You**

This book is meant to help you begin learning the basics of Python programming version 3 or later. It is a brief introduction to Python. At the time of this writing, there are many resources for earlier versions of Python. However, since changes were made in the later versions of Python, using older books and resources can cause some confusion. The author recommends to all new students of programming to begin with Python version 3 or later. If you find source code that you would like to study or use, search the internet for conversion tools that can help you convert the older versions of code to be functional in 3.0 or later.

The author uses the theory that visual learning, humor, and action, (experiential learning), are the best ways for most people to quickly learn something from a book. The author attempts to be as brief as possible to get the new programmer into programming as fast as possible. When you are ready to go deeper into Python, there are many excellent free resources and books on the internet.

WHO'S THE AUTHOR?

Who cares? “I just want to get into programming quickly!” The author chose not to include many useless details about himself that you probably don’t care about anyway. Suffice it to say that he has taught internationally for many years, and is qualified to teach many subjects. The focus here is on learning to program, not on useless information. If you really want to know, look at the cover of the book or check his websites.

WHAT'S A PROGRAMMING LANGUAGE?

Every field of study has new words that you must learn to communicate. When you studied biology, chemistry, and other subjects you were faced with learning new words to talk about that subject. Now you are learning programming so you must also learn more new words. Before you know it, you will be able to speak to world renowned geniuses and geeks in words they understand. You will have the added benefit of understanding what you are talking about. A programming language is a language you can use to communicate with a computer. **Programming** is the art and science of making the computer do what you want it to do by creating programs.

What are programs? **Programs** are algorithms and source code packaged together to achieve your objective(s). Ahhh! More strange words! What's an algorithm or a source code? **Algorithms** are sets of instructions that tell the computer what to do. Algorithms tell the computer how to reach a goal or objective. In daily life someone may ask you for directions to the nearest chocolate factory. You may say; "Here's the algorithm for you; first, go straight ahead until you come to a street light. Turn left at the street light and go to the second parking lot on your right. Park your car in the customer parking area. Enter the back entrance to the factory and eat chocolate until you are too fat to fit through the door." This set of instructions is an algorithm. What's an algorithm? If you said, "It's a set of instructions!" you were paying attention. Good boy...or girl; you know what you are.

Ok, an algorithm is a set of instructions; then what is source code? **Source code** is all the algorithms and instructions that we used in a program. All the words, commands, secret symbols, and other stuff we typed into our program is the source code.

Now that you understand what algorithms and source code are, let's repeat the statement; Programs are algorithms and source code packaged together to achieve an objective(s)

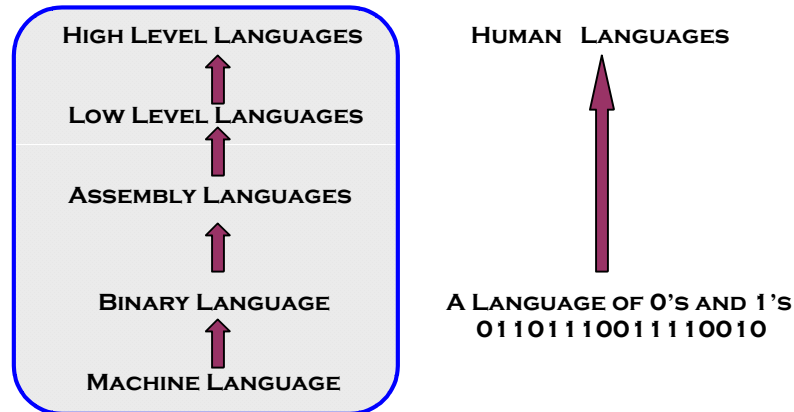
WHAT IS PYTHON?

Python is a computer language. Computers are stupid and don't understand English. So, we have to use computer languages to translate what we say into Computerish or Computerese. Actually, the language computers speak is referred to as a binary language. Binary language is a language based on two words; "on" and "off" represented by the numbers 0 and 1. Humans have trouble communicating in binary. If I say, "011 001 101," you would say, "Huh?" So, languages that are easier for human brains to grasp were invented.

Why so many computer languages? Different languages were designed for different purposes. Some are better at math, some are better at controlling computer hardware, and some are better for the internet. Python is a general purpose language. It can be used for many different purposes.

Python is known as a scripting language, (uses scripts), and is a high level language. A **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. High level languages also take care of many tasks like manipulating the memory of the computer for you. **Low level languages** are used when the programmer wants more direct control over the machine he is using.

This chart gives you a brief over-view of the levels of programming languages.



For the sake of getting you into programming as soon as possible, this book will not expand on Python's history, details on other programming languages, or other details that would delay us from getting started. You can find excellent details on history and other stuff you may want to know at Python's web site, (www.python.org).

HOW TO USE THIS BOOK

This book is for the beginner. It was written for the person who knows a little about using a computer; but knows nothing about programming. If you are new to Python and programming, then read and do everything. If you are an experienced programmer and read this book anyway, then jump around at will to the parts you don't know...or just read everything anyway for its entertainment value. If this is the only book you have with you and you run out of toilet paper...use your imagination. Of course, if you using the free PDF version of the book you have a problem.

GETTING AND INSTALLING PYTHON

You can get your free copy of Python; choose the version you want; choose the Python your operating system; and see any special installation instructions you may need at

<http://www.python.org/download/>.

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?**

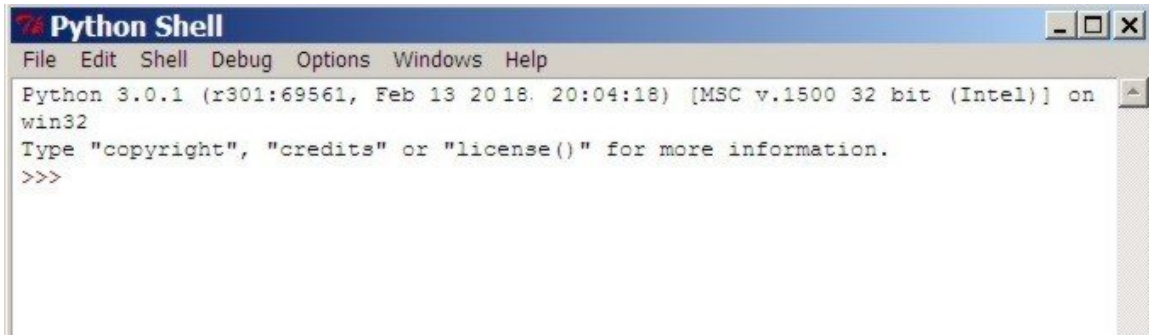
IDLE AND THE SHELL

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**eve**L**opment **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.

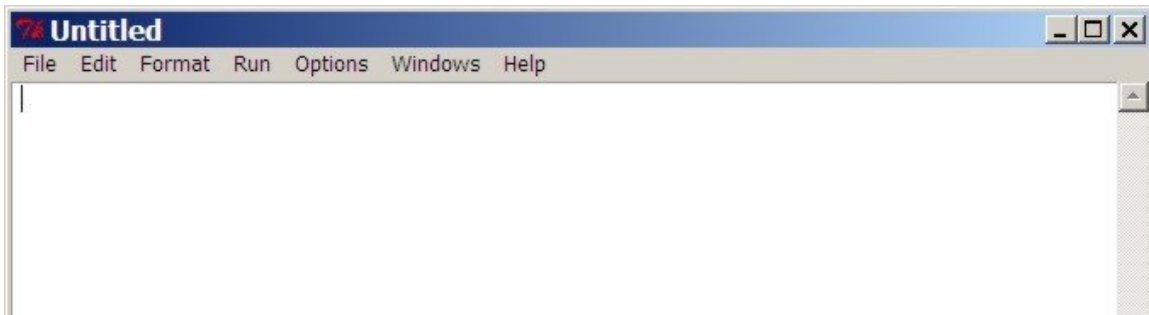
There are two windows to work from in IDLE. There is the **Edit Window** and the **Shell Window**. The Python Shell window will say, "Python Shell" at the top of the window, while the Edit window will say, "Untitled" and have a "run" command listed on the top menu bar. If Python starts in the Shell Window 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.

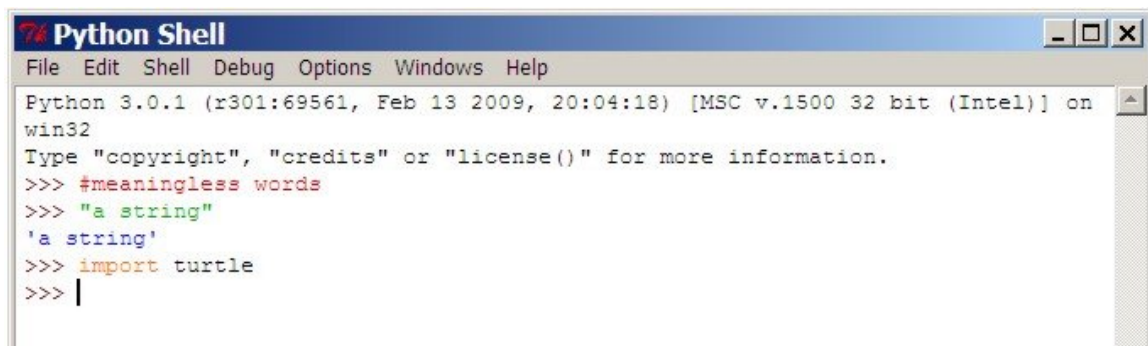
A third function of IDLE is that it is also a debugger. You can find the debugger button at the top of the Shell window. What's a debugger? It's a program to help you kill bugs. No not the one crawling up your leg; but the bugs, or problems, that are in the code we are working on.

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



The screenshot shows a 'Python Shell' window with a menu bar (File, Edit, Shell, Debug, Options, Windows, Help). The text area displays the following code with syntax highlighting: 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 (orange) >>> "a string" (green) 'a string' (green) >>> import turtle (orange) >>> |

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.

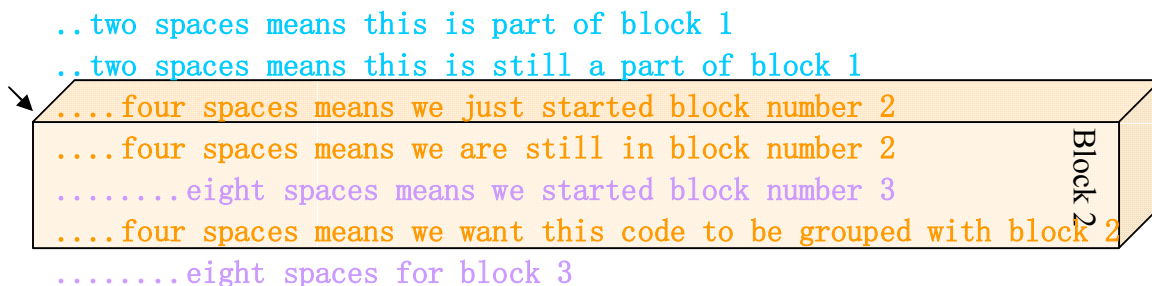
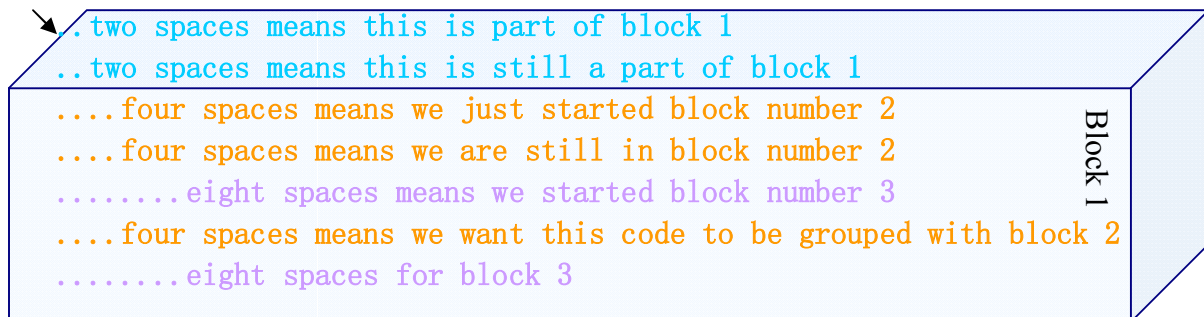
```

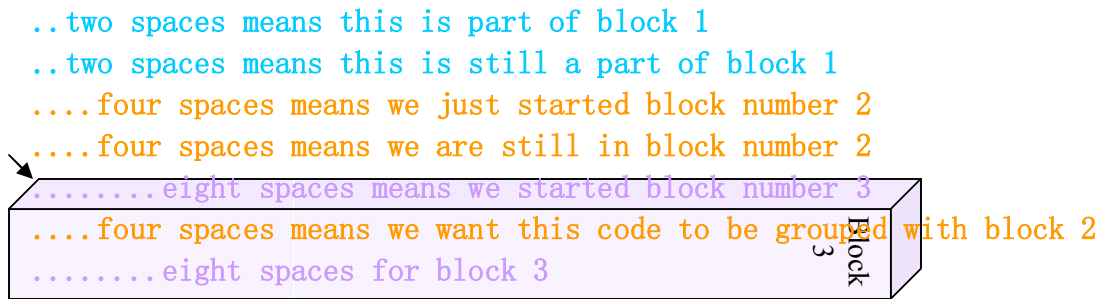
2 spaces  ..students under a hundred years old  (Block 1)
4 spaces  ....girls  (Block 2)
8 spaces  .....girls with green hair  (Block 3)

```

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;





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:

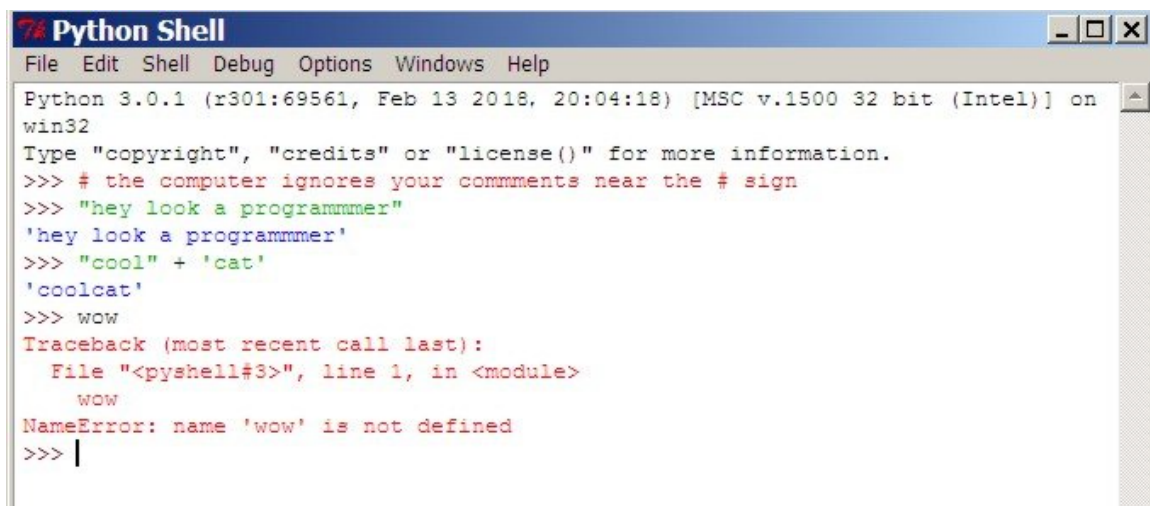
- **YOUR FIRST PROGRAM**
- **STRINGS**
- **VARIABLES**
- **OPERATORS**
- **KEY WORDS (TO USE OR NOT TO USE)**
- **BOOLEAN DATA TYPES**

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.

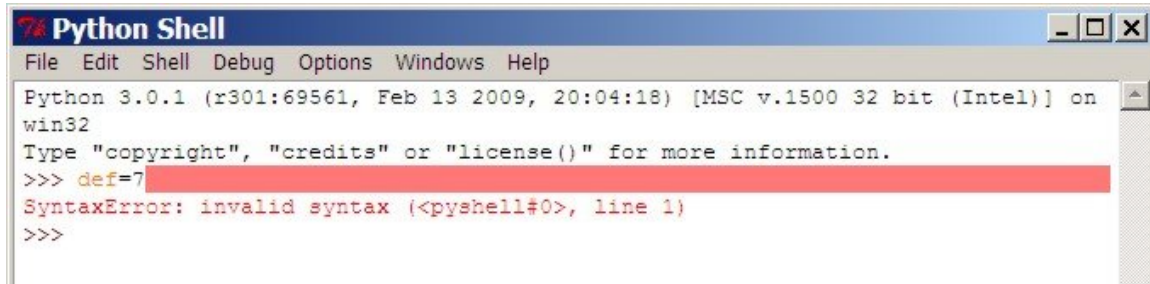
Variables are chunks of data stored in the computers memory. There are generally three types of data stored in variables. Variables can be in the form of **integers** or in a **string** as mentioned previously. The second type of data, called a **float**, refers to the *non-whole numbers like decimals*. 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 says "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.", ">>> def=7", and a red error message: "SyntaxError: invalid syntax (<pyshell#0>, line 1)". The prompt ">>>" is shown again at the bottom.

```
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:

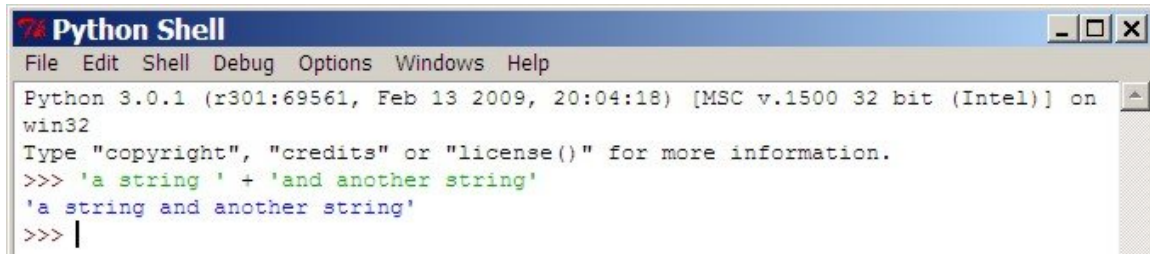
`“words ” *10`

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 says "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.", and a prompt ">>>". The user has entered "'a string ' + 'and another string'" and the shell has responded with "'a string and another string'". The prompt ">>>" is followed by a cursor.

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:

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

Chapter 3: Your First Game!

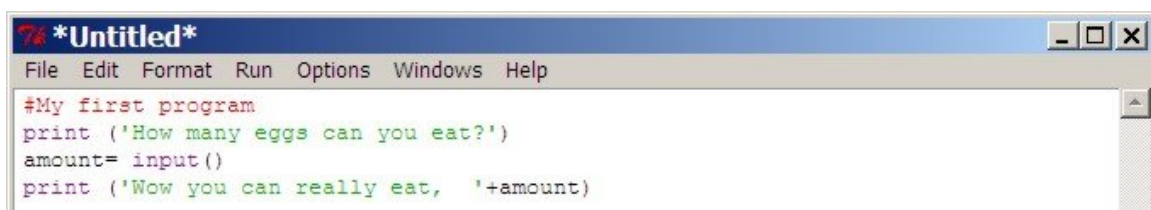
WHAT YOU WILL LEARN:

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

SAVING YOUR FIRST PROGRAM

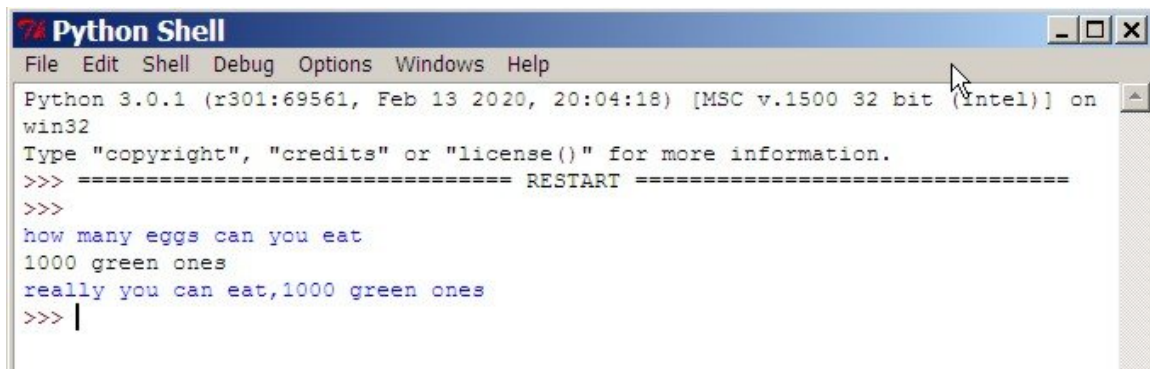
To help you understand your first game you will write a smaller program first. Now you will work in the editor window and learn how to save your file. Open the *editor window*, not the *shell window*, and type:

```
#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.

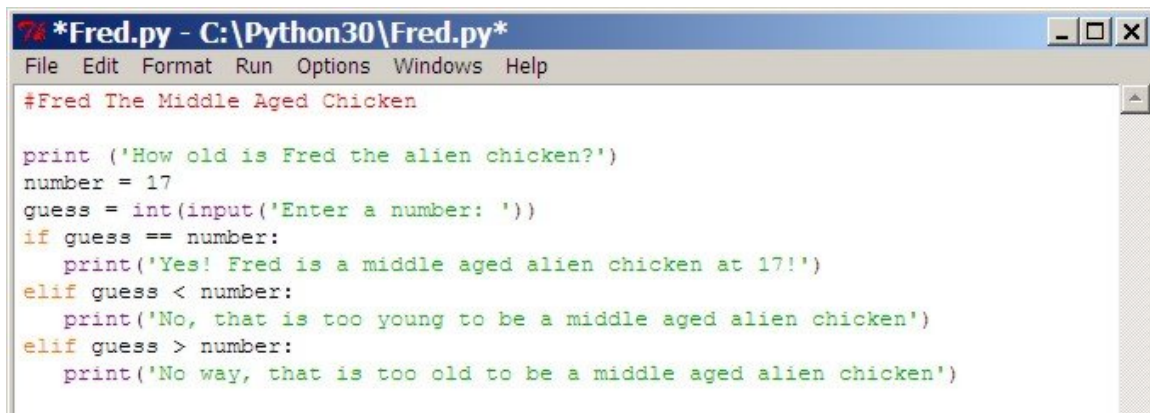
CONDITIONAL STATEMENTS

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.

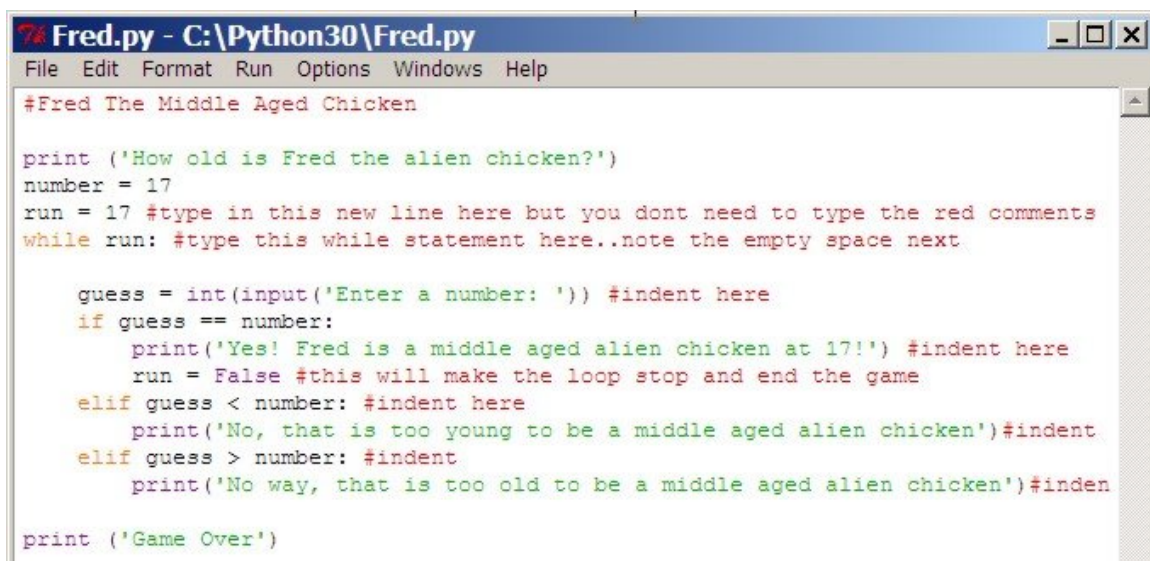
Remember pages ago when we talked about blocks? You were younger then, but maybe you remember. Well, we need to review that stuff and expand on the meaning to be able to teach you about loops and how to fix our games problem of stopping after one guess.

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...

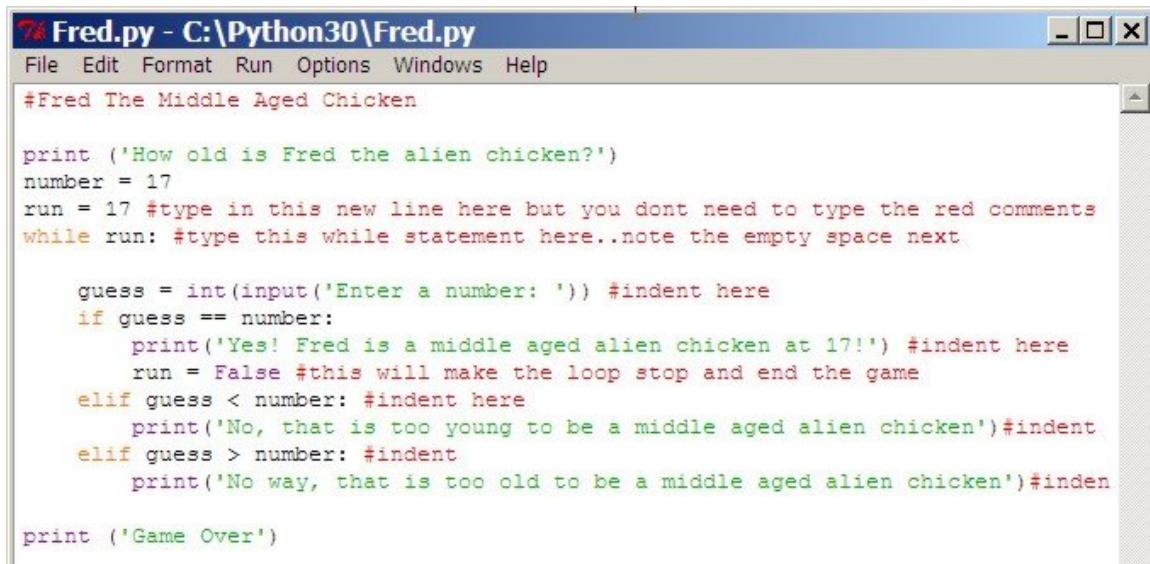
```
print( 'No, that is too old...' )
```

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:



```
#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')
```

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.

Chapter 4: Words Are Boring, Let's Get Graphic!

WHAT YOU WILL LEARN:

- **INTRODUCTION TO GRAPHICS**
- **MODULES**
- **LISTS**
- **SLICES**
- **TUPLES**

GRAPHICS

Maybe you are bored with all this reading game stuff and want to know about graphics. Unfortunately, graphics are a more advanced topic that requires some basic experience in Python before you can get deeply involved with them. However, we can provide a very brief introduction to using graphics in Python without getting too complex.

The first thing you need to understand is that you do not have to program every code command yourself in Python to get things done. This is one of the beautiful things about Python. There are things called “modules” and “libraries” that offer code and mini program modules to turn on or plug into Python that have already done much of the work for you. Python comes with some modules already in it when you download the basic version of Python. Many more modules are available on the internet that you can add, depending upon what kind of programming you plan to do. *See the Index.*

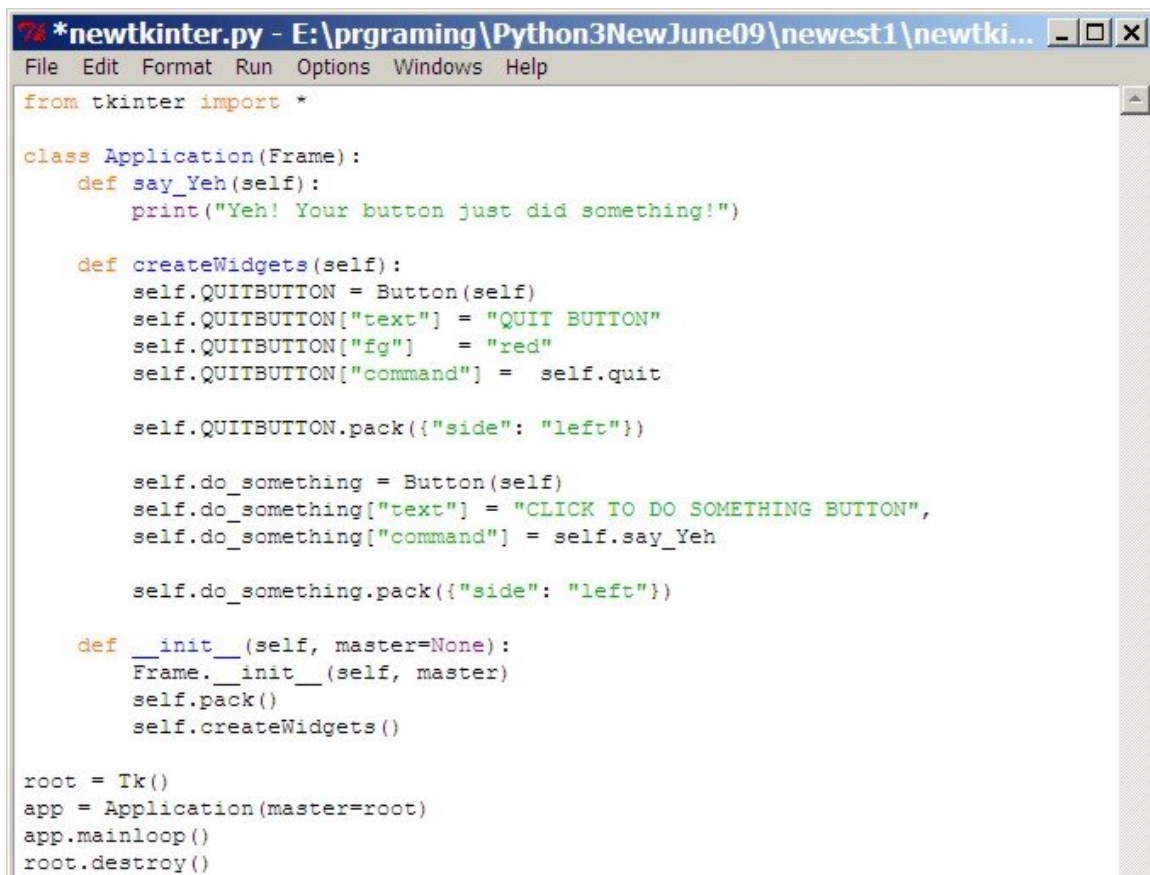
Since this book is designed to get you started as fast and easily as possible, I will only introduce you to one built-in module that comes within the Python you installed.

INTRODUCTION TO MODULES

Modules are programs that can be activated in Python and provide you with more tools and functions to get things done more easily.

We must tell Python that we want to use a module by using the “import” command. The module I will use to introduce you to the world of simple graphics in Python is called “Tkinter.” There is a whole Tkinter world to explore but that’s beyond the scope of this book. Many additional functions for Tkinter can be downloaded and added to Python. I will just introduce you to some simple functions available that are normally within the standard download of Python. In this way, you can use what you have and not worry about downloading or installing modules at this point.

Enough talk, let’s get started. Enter this in your Edit window:

A screenshot of a Python IDE window. The title bar reads '*newtkinter.py - E:\prgraming\Python3NewJune09\newest1\newtki...'. The menu bar includes File, Edit, Format, Run, Options, Windows, and Help. The code editor contains the following Python code:

```
from tkinter import *

class Application(Frame):
    def say_Yeh(self):
        print("Yeh! Your button just did something!")

    def createWidgets(self):
        self.QUITBUTTON = Button(self)
        self.QUITBUTTON["text"] = "QUIT BUTTON"
        self.QUITBUTTON["fg"] = "red"
        self.QUITBUTTON["command"] = self.quit

        self.QUITBUTTON.pack({"side": "left"})

        self.do_something = Button(self)
        self.do_something["text"] = "CLICK TO DO SOMETHING BUTTON",
        self.do_something["command"] = self.say_Yeh

        self.do_something.pack({"side": "left"})

    def __init__(self, master=None):
        Frame.__init__(self, master)
        self.pack()
        self.createWidgets()

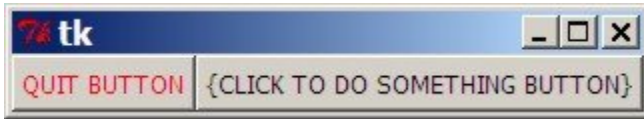
root = Tk()
app = Application(master=root)
app.mainloop()
root.destroy()
```

Now click on *run* then click *run module*, save it, and WOW!

A button appeared! Or at least it should have appeared. If it didn’t appear you either have to check your code or check your eyes. Whenever code that you think is correct disappoints you; go back and make sure every detail of the code was typed correctly. The red error complaints will help you find mistakes. Once every dot, parenthesis, underscore, indentation, and letter is correct; your program will run and the awesome buttons will

appear. Check everything and keep troubleshooting until it works. Don't get frustrated. As a programmer you will spend much of your time solving problems with code.

Your work should look like this:



Click on the; “CLICK TO DO SOMETHING BUTTON.”
The result should look like this:



When you are happy click the; “QUIT BUTTON.” The “QUIT BUTTON” is boring. It just quits the program we just made. The explanations for the code on this will be brief. My objective here is to introduce you to some visual aspects of Python. To learn tkinter in depth is an intermediate topic. After you understand the basics of Python you will be ready to get deeper into modules and graphics.

Code Explanations:

```
from tkinter import *
```

This imports the tkinter module.

```
class Application(Frame):
```

Calls a constructor for the parent class called, "Frame." Huh? Sorry that was geek talk, let's translate that to English. This line of code is bringing in, (calling), the code that makes or constructs a frame. The frame is just a rectangle like a picture frame. We establish it as a parent. A parent and child relationship is used in programming and in 3D animation to describe how you want things to work. For example; your arm bones are parents to your hand bones. If you move your arm your hand will also move. However, you can move your hand without moving your arm. Move a parent object and the child moves with it. The child object can move by itself without moving the parent.

```
def say_Yeh(self):  
    print("Yeh! Your button just did something!")
```

These lines of code define the name, (`say_Yeh`), and a string that we want to print to the computer screen. We will explain more about the `def` statement later.

```
def createWidgets(self):  
    self.QUITBUTTON = Button(self)  
    self.QUITBUTTON["text"] = "QUIT BUTTON"  
    self.QUITBUTTON["fg"] = "red"  
    self.QUITBUTTON["command"] = self.quit  
    self.QUITBUTTON.pack({"side": "left"})
```

These lines define the name and the details of our “Quit” button. The `self` statement will be covered later.

```
self.do_something = Button(self)  
self.do_something["text"] = "CLICK TO DO SOMETHING BUTTON",  
self.do_something["command"] = self.say_Yeh  
self.do_something.pack({"side": "left"})
```

These lines define the name and details of our “CLICK TO DO SOMETHING” button.

```
def __init__(self, master=None):  
    Frame.__init__(self, master)  
    self.pack()  
    self.createWidgets()
```

These are instructions to determine how things within our frame will behave. A deeper study of widgets in tkinter is required to fully understand how to write and use these statements.

```
root = Tk()  
app = Application(master=root)
```

The main program will start here.


```
app.mainloop()
root.destroy()
```

These lines are to instruct the computer to create a loop until certain mouse or keyboard events happen.

Yes, I know we rushed over these explanations. For now, stop whining and just copy the code. Some of these statements will be discussed later in this course while other details will be left for intermediate and advanced studies. To be a master of widgets and graphics requires much more information and practice than can be provided in a basics book. In some cases, you will not truly understand code until you have used it many times. So, relax and do it.

The modules and aspects of Python that you choose to master first will depend upon your objectives. If your objective is to write computer games; you will have to master graphics at a higher level than you will need for simple applications. If you are going to work with stock market forecasting; math modules would be more important to you. Think about your goals and objective and chose the appropriate modules for your continued study.

LISTS

A *list* is a sequence of elements. You could make a list of the family members in your home:

```
Family = ['Dad', 'Mom', 'Brother', 'Sister', 'Me', 'Cat', 'Cockroaches']
```

Each element in the list has a position of reference. In other words, they are numbered for indexing or referring to them later. But, instead of calling the first element in the list; “1,” programmers begin with “0.” So, now you won’t be confused if you see a programmer counting how many cups of coffee he had tonight by beginning with; “0, 1, 2, 3...” Lists can store numbers, strings, both, and they can store other lists.

They may look like this:

```
Numbers:    theAlist=[ 2, 4, 6, 8 ]
Strings:    theBlist=['words', 'abc', 'fuglet']
Both:       theClist=['abc', 4, 'words', 7]
Other Lists: AllLists=[theAlist, theBlist, theClist]
```

We can edit lists. But, the word “edit” is too common for a programmer to use, so let’s use the word “mutable.”

Later you will learn about tuples. Tuples and strings are immutable. Huh? **Mutable** means we can edit it. **Immutable** means we can't edit it. So the secret meaning is that lists can be edited; but strings and tuples can't be edited.

How can I edit a list and why would I want to?

Maybe you would like to edit your list of family members. Remember our list:

Index position:

	0	1	2	3	4	5	6
Family =	["Dad"]	["Mom"]	["Brother"]	["Sister"]	["Me"]	["Cat"]	["Cockroaches"]

Let's say you bought a mouse and he ate the family cockroaches. It was a sad event but life goes on and now you must edit your list of family members.

You need to replace "Cockroaches" with "psycho mouse."

To replace an item in a list you need to re-define what is in a particular index position. Since "Cockraoches" is in the 6th position of our "Family" list, (remember you count from 0), we do this:

```
Family[6]= "Psycho Mouse"
```

Give it a try. First, define what is in your list by typing:

```
Family = ["Dad", "Mom", "Brother", "Sister", "Me", "Cat", "Cockroaches"]
```

Then, type:

```
Family[6]= "Psycho Mouse"
```

Test it by typing:

```
print(Family)
```

```
*family.py - C:/Python30/family.py*
File Edit Format Run Options Windows Help
Family = ["Dad", " Mom", "Brother", "Sister", "Me", "Cat", "Cockroaches"]
Family[6]= "Psycho Mouse"
print(Family)
```

Now run your program.

```
Python Shell
File Edit Shell Debug Options Windows Help
Python 3.0.1 (r301:69561, Feb 13 4009, 20:04:18) [MSC v.1500 32 bit (Intel)] on
win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
['Dad', ' Mom', 'Brother', 'Sister', 'Me', 'Cat', 'Psycho Mouse']
```

What happens if I want to add something to my list? *Adding something to a list is referred to as **appending** the list.*

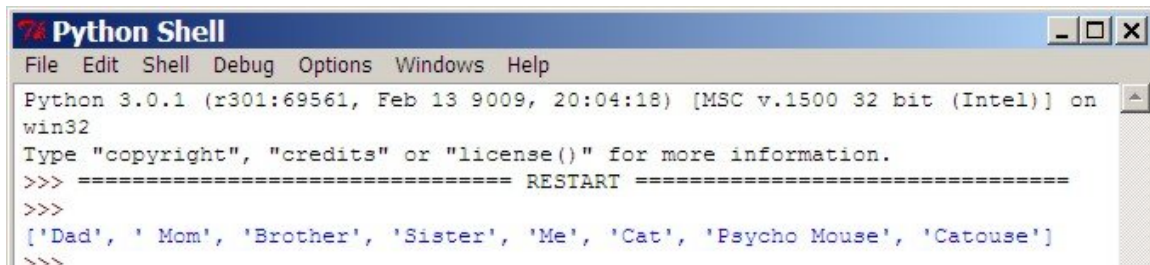
Let's say that our Cat and Psycho Mouse got married and had a baby. We will call it, "Cathouse." Now we have a new member of our family and we must append the list. We add this line before our print statement:

```
Family.append("Catouse")
```



```
family.py - C:\Python30\family.py
File Edit Format Run Options Windows Help
Family = ["Dad", " Mom", "Brother", "Sister", "Me", "Cat", "Cockroaches"]
Family[6]= "Psycho Mouse"
Family.append("Catouse")
print(Family)
```

Run it:



```
Python Shell
File Edit Shell Debug Options Windows Help
Python 3.0.1 (r301:69561, Feb 13 9009, 20:04:18) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
['Dad', ' Mom', 'Brother', 'Sister', 'Me', 'Cat', 'Psycho Mouse', 'Catouse']
>>>
```

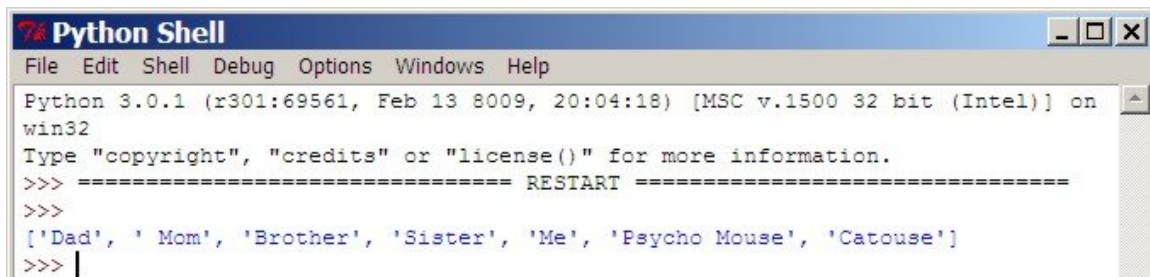
How do I remove an item? I'm glad you asked. Let's say that Psycho Mouse gets angry and eats the Cat. To remove "Cat" from the Family list we use **del** for delete and give the index position:

```
del Family[5]
```



```
family.py - C:\Python30\family.py
File Edit Format Run Options Windows Help
Family = ["Dad", " Mom", "Brother", "Sister", "Me", "Cat", "Cockroaches"]
Family[6]= "Psycho Mouse"
Family.append("Catouse")
del Family[5]
print(Family)
```

Run it to remove the cat:



```
Python Shell
File Edit Shell Debug Options Windows Help
Python 3.0.1 (r301:69561, Feb 13 8009, 20:04:18) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
['Dad', ' Mom', 'Brother', 'Sister', 'Me', 'Psycho Mouse', 'Catouse']
>>> |
```

Just to complicate our lives, my sister marries a guy and now another family has become part of our family. Fortunately, adding two lists together is easy. First we define what is in the second list. Let's add this list under our Family list:

```
Family2=["a guy", "dog", "father", "mother"]
```

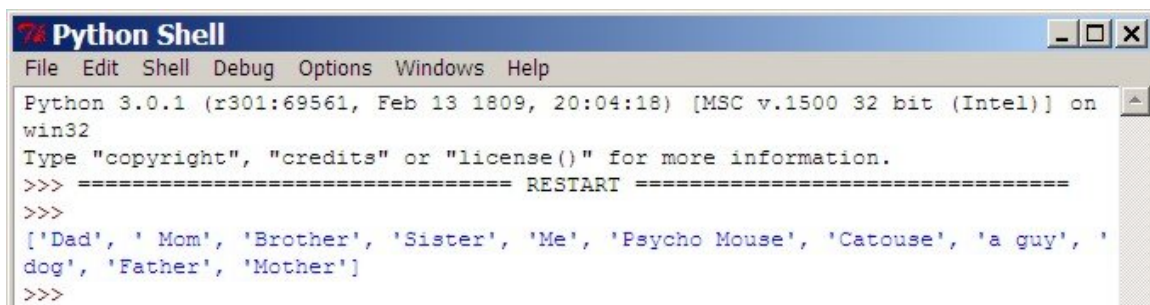
Then to add the two lists change the print statement to:

```
print(Family+Family2)
```



```
family.py - C:\Python30\family.py
File Edit Format Run Options Windows Help
Family = ["Dad", " Mom", "Brother", "Sister", "Me", "Cat", "Cockroaches"]
Family2=["a guy", "dog", "Father", "Mother"]
Family[6]= "Psycho Mouse"
Family.append("Catouse")
del Family[5]
print (Family+Family2)
```

Run it to combine the two lists:



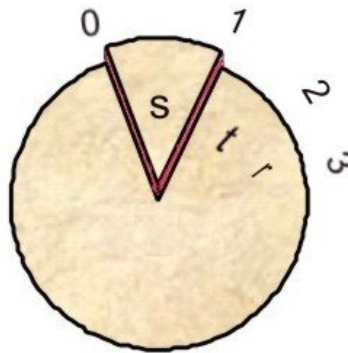
```
Python Shell
File Edit Shell Debug Options Windows Help
Python 3.0.1 (r301:69561, Feb 13 1809, 20:04:18) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
['Dad', ' Mom', 'Brother', 'Sister', 'Me', 'Psycho Mouse', 'Catouse', 'a guy', 'dog', 'Father', 'Mother']
>>>
```

SLICES

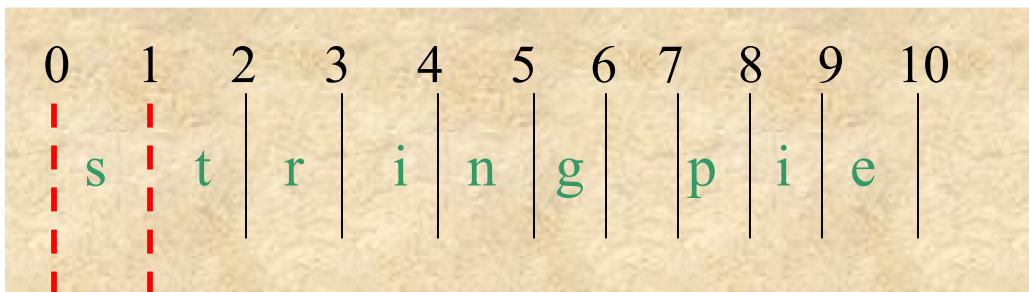
Remember how we used an index to cut one part of a list out to exchange it or delete it? Wouldn't it be nice if you could choose any part of the list; one character; half of an index; all of the list; or any part you want to copy or work with? You can! A *slice* is cutting a list into smaller parts. A **slice** is a subset of a list. Just as you can choose what piece of pie you want to cut and how big you want to cut it; you can cut a slice from a list starting and ending where you want it.

Let's cut up a string pie. Don't worry; you don't need to eat it. The strings always get caught between your teeth. Remember that we use a 0 for our first index position. But, when talking about slices of pie or strings you have to use two reference points. You need an index or reference point for the first cut and one for the final cut of the slice. So, our index points are on each side of the characters in our string.

If we want to slice out the "s" we have to tell the computer the **beginning and ending index for the slice, [0:1]**.



It's probably easier to think of it this way:



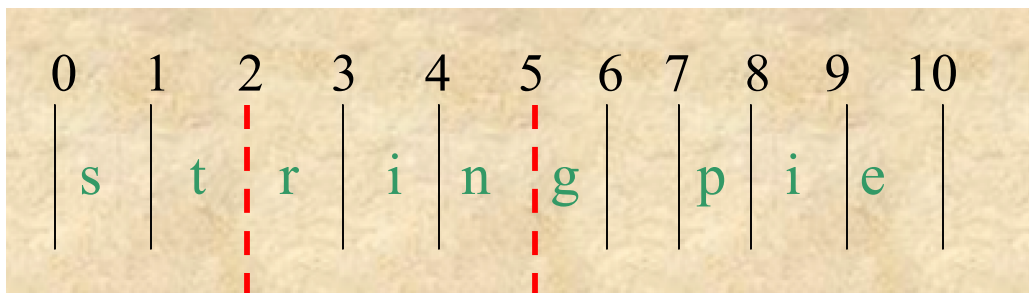
Try this:

```
test.py - C:\Python30\test.py
File Edit Format Run Options Windows Help
piece = "string pie"
print (piece [2:5])
```

Run it:

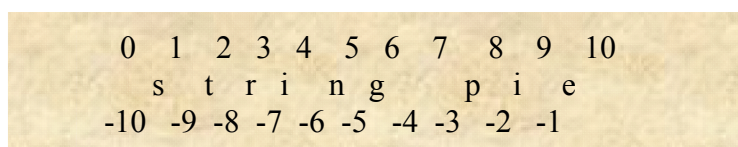
```
Python Shell
File Edit Shell Debug Options Windows Help
Python 3.0.1 (r301:69561, Feb 13 3009, 20:04:18) [MSC v.1500 32 bit (Intel)] on
win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
rin
>>> |
```

I'm not sure what a "rin" tastes like but we successfully sliced a piece of our string from index position 2 to 5!

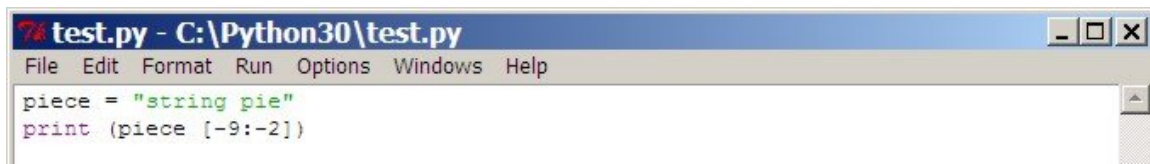


You try a few to practice. First choose the letters you want to cut out of the string. Then choose the beginning and ending cut for your slice of the string. Practice until you are confident in choosing the index positions and getting the slice you wanted.

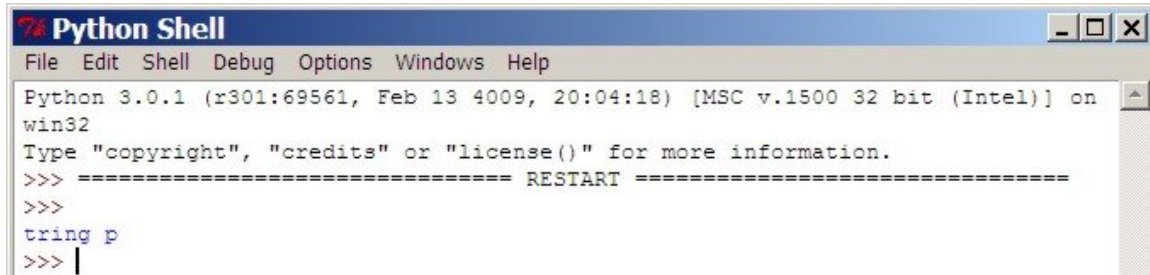
There is another way to refer to index positions in a slice if you want to use it. You can think backwards! There are practical uses for this in more advanced programming. Here I will only introduce you to it and you can play with it if you want to. In this index system, index positions use negative numbers.



Try it:



Run it:



A “tring p” probably tastes better than a “rin,” but I’m only guessing. When you program you will find that making a copy of part or all of what you have typed before will save you a lot of time. Slicing strings is a useful talent.

TUPLES

Let’s talk tuples! A *tuple* is like a list but can’t be edited or changed in its original form. We generally use parenthesis, (), rather than brackets, [], to tell Python we are defining a tuple.

This is a *tuple*:

```
Family3=('a guy', 'dog', 'cat')
```

This is a *list*:

```
Family3=['a guy', 'dog', 'cat']
```

Tuples are not easily changed so why would we use them?

A program can run a tuple with more efficiency than a list. Unless the program is very small, tuples are faster. So, if you don’t need to change your list, it’s generally better to use a tuple. Tuples can contain many *things you may not want to change*. These **constants** in your program may be; music, a sequence of sound files, a sequence of images, etc. You can assign an individual item to a variable or you could assign a group of items in a sequence to a tuple. Tuples can also be **nested**, (*placed*), *inside* of other tuples; called up as *functions*; and can hold any data type. For some applications, tuples are used much more often than lists.

VOCABULARY REVIEW:

<i>Immutable means we can't edit it.</i>
<i>List a sequence of elements</i>
<i>Modules are programs that can be activated in Python and provide you with more tools and functions to get things done more easily.</i>
<i>Mutable means we can edit it.</i>
<i>Nested, (placed), inside</i>
<i>Slice a subset of a list</i>
<i>Tuple like a list but can't be edited or changed in its original form.</i>

Chapter 5: Functions

WHAT YOU WILL LEARN:

- **CALLING FUNCTIONS**
- **WRITING/DEFINING FUNCTIONS**
- **ARGUMENTS**
- **GLOBAL AND LOCAL SCOPE**

WHAT'S A FUNCTION?

*A **function** is a re-usable mini program inside of your program.*

You have already been using functions throughout this book. The “**print()**” *function call* is the one you have used most. A ***function call*** is the code you type to turn on a function.

Calling a function:



Jody S. Ginther ©2010
www.toonzcat.com

HOW DO I CREATE/DEFINE MY OWN FUNCTIONS?

It is common in writing code to use the same code many times. If we assign this code to a function, we only need to call that function again rather than retype the code. To define the meaning of a function for the computer we use the `def` statement.

The `def` statement is made up of:

the keyword, `def`

our function name

followed by parenthesis

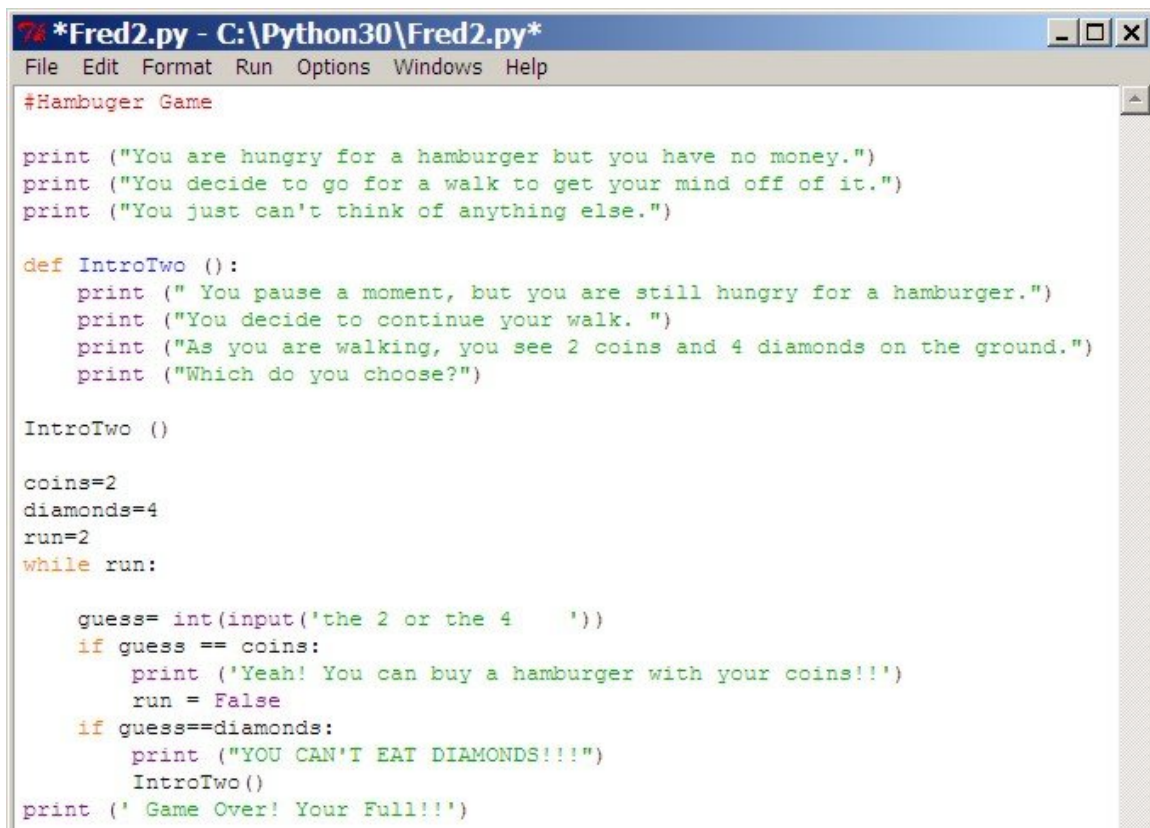
a colon

and finally the def-block.

```
def thename ():  
    here we put the details of our definition block  
    we can put other functions or code here
```

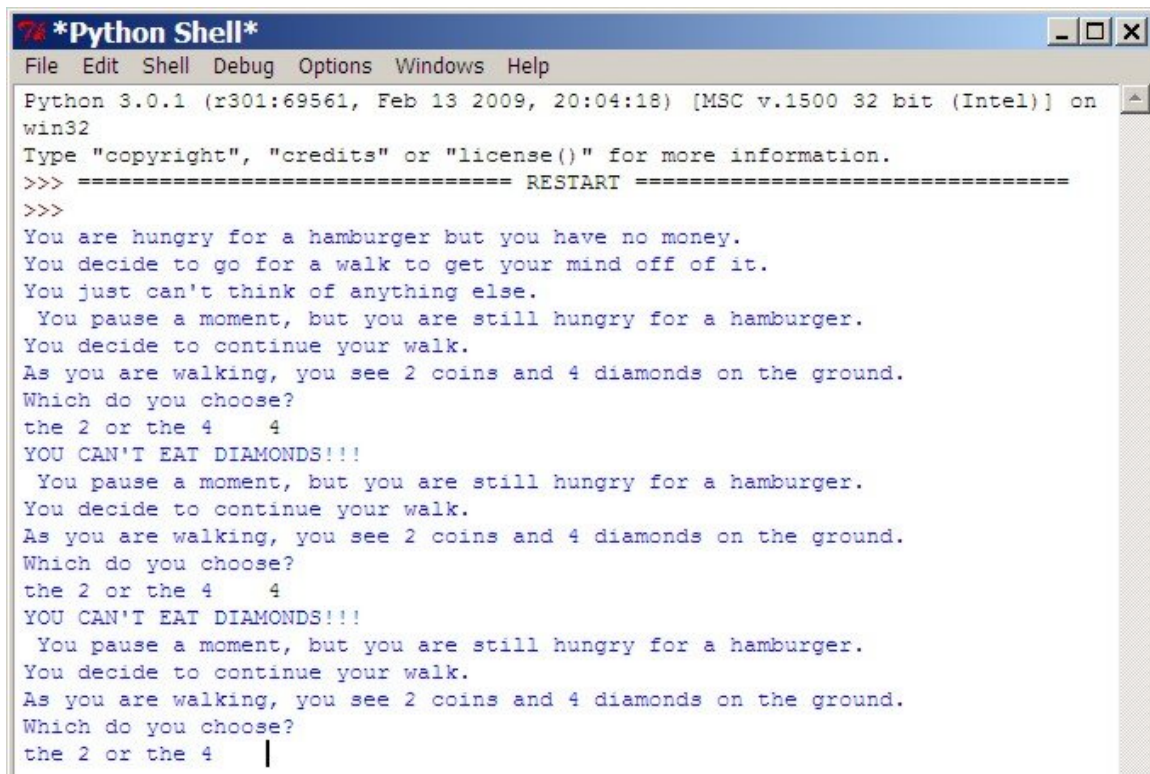
Let's try it. Below is the code for a new game based on our Alien Chicken game. The purpose of this game is to show you that by making and using your own functions, you can save a lot of time retyping things. We have a two part introduction to this game. The second part will be repeated until the game is over. So, this part will be defined as a function and called up when we need it.

Type this:



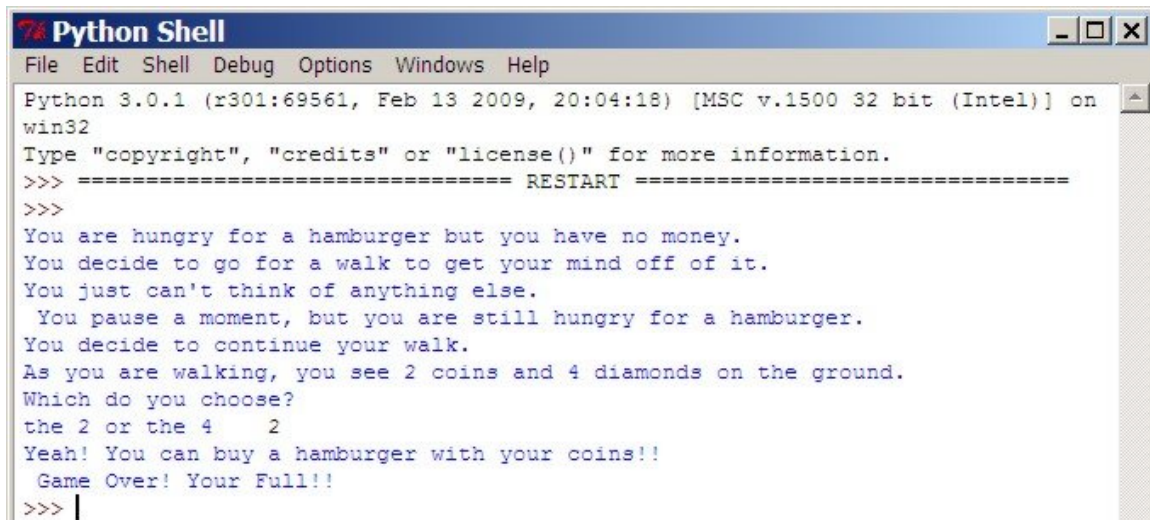
```
*Fred2.py - C:\Python30\Fred2.py*  
File Edit Format Run Options Windows Help  
#Hamburger Game  
  
print ("You are hungry for a hamburger but you have no money.")  
print ("You decide to go for a walk to get your mind off of it.")  
print ("You just can't think of anything else.")  
  
def IntroTwo ():  
    print (" You pause a moment, but you are still hungry for a hamburger.")  
    print ("You decide to continue your walk. ")  
    print ("As you are walking, you see 2 coins and 4 diamonds on the ground.")  
    print ("Which do you choose?")  
  
IntroTwo ()  
  
coins=2  
diamonds=4  
run=2  
while run:  
    guess= int(input('the 2 or the 4 '))  
    if guess == coins:  
        print ('Yeah! You can buy a hamburger with your coins!!!')  
        run = False  
    if guess==diamonds:  
        print ("YOU CAN'T EAT DIAMONDS!!!")  
        IntroTwo()  
print (' Game Over! Your Full!!!')
```

Here is the result if you make the wrong guess a couple of times:



```
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.
>>> ===== RESTART =====
>>>
You are hungry for a hamburger but you have no money.
You decide to go for a walk to get your mind off of it.
You just can't think of anything else.
You pause a moment, but you are still hungry for a hamburger.
You decide to continue your walk.
As you are walking, you see 2 coins and 4 diamonds on the ground.
Which do you choose?
the 2 or the 4    4
YOU CAN'T EAT DIAMONDS!!!
You pause a moment, but you are still hungry for a hamburger.
You decide to continue your walk.
As you are walking, you see 2 coins and 4 diamonds on the ground.
Which do you choose?
the 2 or the 4    4
YOU CAN'T EAT DIAMONDS!!!
You pause a moment, but you are still hungry for a hamburger.
You decide to continue your walk.
As you are walking, you see 2 coins and 4 diamonds on the ground.
Which do you choose?
the 2 or the 4    |
```

This will repeat our function, IntroTwo, until you answer correctly.
Here is the result if you guess correctly:



```
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.
>>> ===== RESTART =====
>>>
You are hungry for a hamburger but you have no money.
You decide to go for a walk to get your mind off of it.
You just can't think of anything else.
You pause a moment, but you are still hungry for a hamburger.
You decide to continue your walk.
As you are walking, you see 2 coins and 4 diamonds on the ground.
Which do you choose?
the 2 or the 4    2
Yeah! You can buy a hamburger with your coins!!
Game Over! Your Full!!
>>> |
```

Let's have a brief look at the code:

```
#Hamburger Game
```

```
print ("You are hungry for a hamburger but you have no money.")  
print ("You decide to go for a walk to get your mind off of it.")  
print ("You just can't think of anything else.")
```

We are repeating the print statement to output each statement as a separate line on the computer screen.

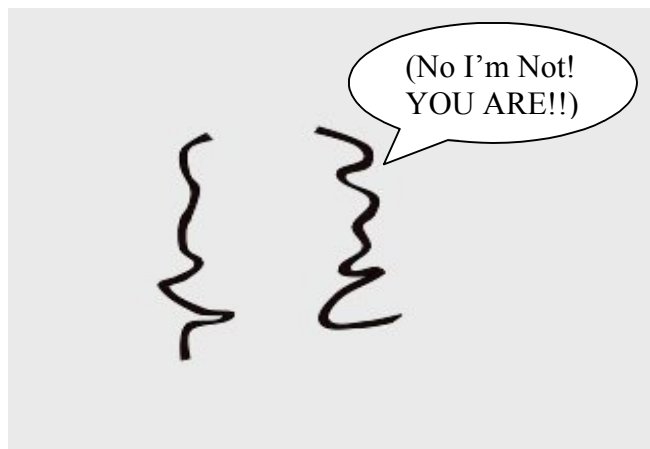
```
def IntroTwo ():  
    print (" You pause a moment, but you are still hungry for a hamburger.")  
    print ("You decide to continue your walk. ")  
    print ("As you are walking, you see 2 coins and 4 diamonds on the ground.")  
    print ("Which do you choose?")
```

```
IntroTwo ()
```

*The brackets, (), in a def statement give us an optional place to add **parameters**. For functions calls, like the *str* function, we refer to the data we put in the parenthesis as **arguments**. Arguments placed in the brackets are special instructions on how we want the function to work and must be separated by commas. In our Hamburger Game we had no additional arguments so the parentheses were left empty.*

Note: Programmers often refer to additional data, whether it be strings, tuples, or whatever, as arguments.

String arguments:



*Jody S. Ginther ©2010
www.toonzcat.com*

```

coins=2
diamonds=4
run=2
while run:

    guess= int(input(' the 2 or the 4 '))
    if guess == coins:
        print ('Yeah! You can buy a hamburger with your coins!!')
        run = False
    if guess==diamonds:
        print ("YOU CAN'T EAT DIAMONDS!!!")
        IntroTwo()
print (' Game Over! Your Full!!')

```

The code here is a repeat of the code we used in our Middle Aged Chicken game. If you don't remember the purpose of each line of code refer to the explanations for that game. The main difference here is that we added our function, `IntroTwo()`.

SCOPE: OR LOCAL

Jody S. Ginther ©2010

When you create a function, sometimes you want to control what part of your code the function will work in. You may want your function to work within a certain block of code or you may want it to be applied to all of your code. Previously, we controlled what block of code a statement influenced by proper indentation. With a function, we can add a statement to clarify its *region of influence or the scope of the function*. A function can't be seen by the computer outside its scope or region of influence. *Scopes are sometimes called **namespaces***. If you want it to work at the top level of your program you must tell Python that it's a global function. *A **global statement** tells Python to be able to use the function everywhere in the program; (to always see it as being in the outer block of code).*

They syntax of this kind of global statement is:

```

x = 7
def FunctionName ():
    global x

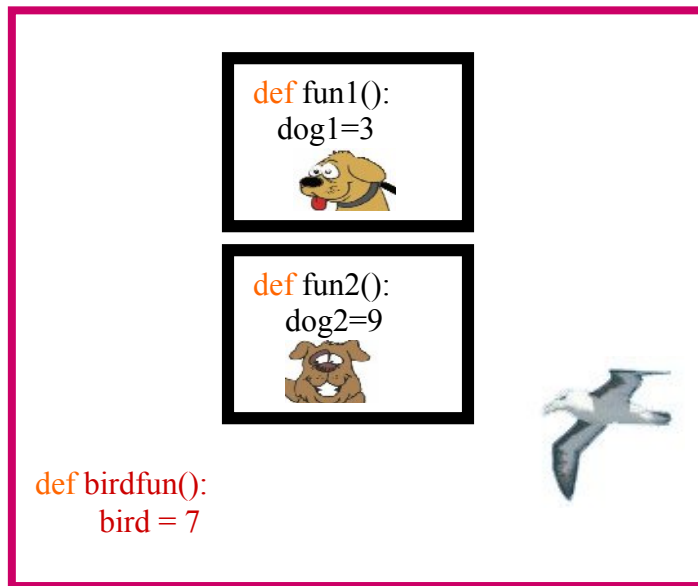
```

What happens if I add another statement as follows defining “local_1= 9?”

```
x = 7
def FunctionName ():
    global x
    local_1= 9
```

The Global variable; “x” is accessed outside the block. The “local_1=9” variable is a local variable accessed only within the “FunctionName” block.

Let’s look at limited scope another way:



The above illustration has three variables and three functions. The variables created under the functions `def fun1()` and `def fun2()` are like dogs within a fence. Any variables you create are limited to roam within the fence. The variable `dog1` has no effect on anything outside the scope of his fence. Neither does the variable `dog2`. However, the bird variable, like a bird, can move or work anywhere within the program. Functions will behave in the same way unless you use a global statement to change their scope.

Remember that if you don’t tell Python that a function call is global, Python will assume that it’s local. Don’t worry if you don’t understand everything at this point. This is just a general overview of these statements. You will learn to use these statements through practice and further study. In more advanced studies, you will also learn to use global statements to control interaction within, or between modules.

This was just a short introduction to functions. You should continue to dive deeper into the study of functions to become a better programmer. But, before writing complex functions you should remember that Python has many functions built in, has many within its standard library, and many are available in the modules you may want to use. You may find one that already does the job for you. It's a good idea to become familiar with what is already out there before using your valuable time to create your own functions. This is a judgment call. If it takes less time to write it than to search for it; then write it.

Python functions can be stored as objects that can be used later as arguments within other functions. This is just a fancy way to say you can keep and copy functions to be like building blocks within more complex functions later.

VOCABULARY REVIEW:

<i>Function</i>	<i>a re-usable mini program inside of your program</i>
<i>Function call</i>	<i>the code you type to turn on a function.</i>
<i>Global statement</i>	<i>tells Python to be able to use the function everywhere in the program</i>
<i>Scope or Namespace</i>	<i>region of influence</i>

Chapter 6: Parts Of Python

WHAT YOU WILL LEARN:

- **WHAT'S OBJECT ORIENTED?**
- **THE STANDARD LIBRARY**
- **DICTIONARIES (NOT THE ENGLISH KIND)**

MODULES AND OBJECT ORIENTED LANGUAGE

Python is referred to as an “object oriented” language. What does that mean? In an effort to shorten programming time, programmers decided that it would be really great if they could divide large programs into smaller pieces. Then, make these pieces to be self-contained mini-programs that could be combined according to the needs of the programmer. These pieces are often referred to as objects. Without getting into the long history and complications that they ran into, we will just jump to the moral of the story. In Python, code is divided, classified, repackaged and combined to create modules to be chosen and used for specific programming purposes. It's a little like using copy and paste rather than re-typing and entire book. Python is a very useful program that can save a lot of time. Python allows us to reuse code as much as possible.

As we mentioned in previous chapters, modules are mini programs that have their own data type class, definitions, etc. Remember how you used the tkinter module? First you had to import it using the import statement. Then you were able to use its special functions.

You will have to import and study every module you choose to use. You must learn the statements and syntax necessary to use that particular module's features. The syntax for using functions in modules often includes a dot operator. So, an example to use a module function would be:

```
nameOfModule.functionName(arguments)
```

There are standard library modules, third party modules, and after you are experienced in Python; modules of your own creation. The internet is loaded with modules for nearly every purpose a programmer may need. Set your goals, and choose accordingly. You may want to start by getting some experience playing with Python's standard library modules.

STANDARD LIBRARY MODULES

The standard library is the library of modules that Python includes with the standard version of Python. If you have Python installed, you have the standard library. Python's standard library includes many things you might need. The standard library includes modules for; tkinter and other graphics, HTML, web browsers, data bases, email, GUI, wave files, multi-threading, templating, math modules, data compression, cryptographic services, generic operating system services, internet tools, networking, debugging, internationalization (language services), documentation generator, etc.

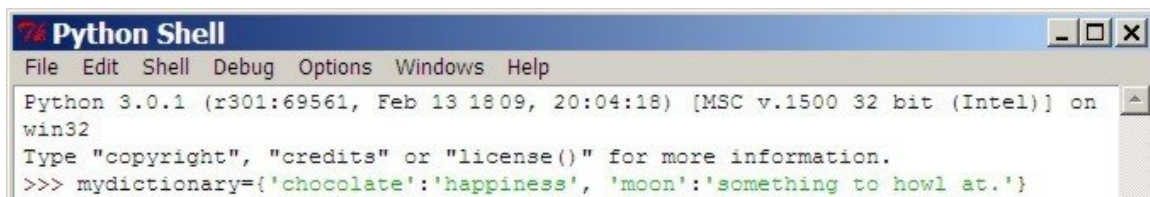
See the index for libraries and module websites and resources.

DICTIONARIES

You can see that programmers need to arrange data many different ways to produce programs quickly. Python and object oriented languages allow many ways to organize your code to access and use it quickly. A dictionary is another way to organize your code. You can create custom dictionaries of code for a specific project or program. Your imagination is the only limitation to what kind of dictionaries you can create. Unlike a normal dictionary however, Python dictionaries don't have to organize the keywords and definitions in any particular order.

*We call the name we choose for our dictionary entry; the **key**. We call the details or definition; the **value**. Programmers often refer to the key and its associated value as a **key-value pair**.*

Just as in a real dictionary, we find the definition, or value, we are looking for by looking up the keyword, or key. We can't look up the definition to find the key. The format for creating an entry into a dictionary is:

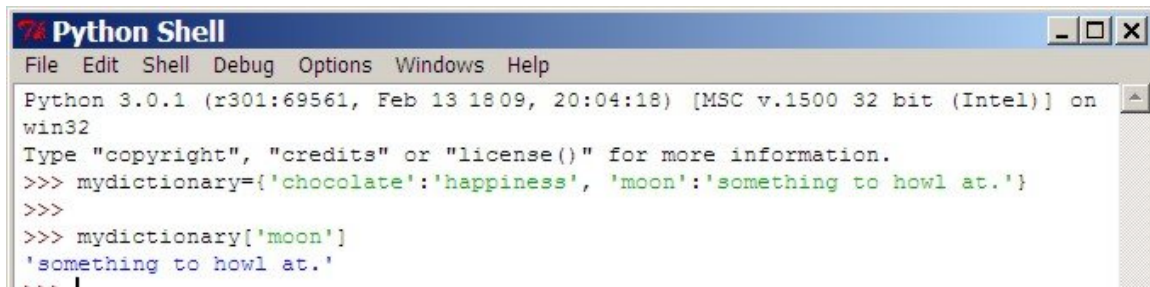
A screenshot of a Python Shell window. The title bar says "Python Shell". The menu bar includes "File", "Edit", "Shell", "Debug", "Options", "Windows", and "Help". The text area shows the following: "Python 3.0.1 (r301:69561, Feb 13 1809, 20:04:18) [MSC v.1500 32 bit (Intel)] on win32", "Type 'copyright', 'credits' or 'license()' for more information.", and a prompt ">>>" followed by the code "mydictionary={'chocolate':'happiness', 'moon':'something to howl at.'}" which has been executed.

```
Python 3.0.1 (r301:69561, Feb 13 1809, 20:04:18) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> mydictionary={'chocolate':'happiness', 'moon':'something to howl at.'}
```

We must make a name for our dictionary first. In this case I just called it, “mydictionary.” We then create a key-value pair for each entry. This dictionary only has two entries, but you can enter as many as you want. Be careful. Note that the entire dictionary is bracketed with {}, the keys and the values are separated by colons, :, and they are enclosed in quotation marks, “. If your dictionary does not work, check your syntax.

How to we find or access our dictionary entries?

To access an entry from our dictionary:

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

```
Python 3.0.1 (r301:69561, Feb 13 1809, 20:04:18) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> mydictionary={'chocolate':'happiness', 'moon':'something to howl at.'}
>>>
>>> mydictionary['moon']
'something to howl at.'
>>>
```

The syntax is:

dictionaryname ['keyword']

You can only use immutable objects for keys, but you can use immutable and mutable objects for values. Create your own dictionary and practice accessing your code.

VOCABULARY REVIEW:

Key	<i>the name we chose for our dictionary entry</i>
Key-value pair	<i>a key and its associated value</i>
Value	<i>the details or definition for a dictionary entry/key</i>

Chapter 7: Other Stuff

WHAT YOU WILL LEARN:

- **KEYWORDS AND EXPRESSIONS**
- **DEBUGGING AND EXCEPTIONS**
- **WHAT'S NEXT**

MORE KEYWORDS AND EXPRESSIONS

This is a very brief summary of the uses of the other primary keywords in Python. You will not get a chance to use some of them until you are at an intermediate to advanced level in Python. I am introducing them to you in case you have any special needs in your programming and see one that may fill those needs. If so, you can get information from the Python.org official site or join a forum to learn how to use it. Let's have another look at the chart we saw before:

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	

You will now recognize some of these statements from the previous chapters. The following is a review of some of the key words we did use and a brief description of those we didn't use:

<i>assert</i>	<i>allows you to insert debugging code into your program</i>
<i>break</i>	<i>breaks out of a block of code even if the while statement remains true</i>
<i>continue</i>	<i>tells Python to skip the remaining statements in the current loop block and jump to the next loop block.</i>
<i>else</i>	<i>statement offers a choice in a while loop...do this or else.....do...</i>
<i>for...in</i>	<i>looping statement to find if something is <u>in</u> a list</i>
<i>if</i>	<i>if a statement is true...</i>
<i>is</i>	<i>similar to the == operator</i>

lambda	<i>is a variation on the def statement with other functionality (most programmers prefer to just use def)</i>
not	<i>works like the English not and makes something the opposite meaning</i>
pass	<i>acts as a neutral place holder in some statements</i>
return	<i>returns a value from a function that can be assigned to another function</i>
try, except, and finally	<i>can be used for testing error messages, (advanced)</i>
while	<i>while true repeatedly run this block of code</i>
with	<i>statement has to do with writing new classes, (advanced Python)</i>
yield	<i>has to do with execution flow control and “freezing.” (advanced)</i>

DEBUGGING AND EXCEPTION HANDLING

What? No, you don’t need a flea collar or special shampoo, **debugging** is *finding and solving problems in a program*. **Exceptions** are *important or exceptional events to notice in the program*. Understanding exception errors is a more advanced topic. For now, check for the most common mistakes in your code first. Typing, syntax, and indentation errors seem to be the most common for beginners. Check that every letter and punctuation mark in your code is correct. If you have a run-time error, (your program crashes), try running one block of your code at a time to see where the problem is. Logic errors, (using the wrong instructions), are more difficult to track down. Study your code, compare it to similar code that works and try to find out what went wrong.

Yeah, I know. These are not very efficient ways to debug programs. When you are at a more advanced level you will understand and be able to use the tools Python provides to help you. For now, trial and error can be a good teacher. Anything learned too easily, is often forgotten easily.

NOW WHAT?

PRACTICE!! Go back to the beginning of this book and do the exercises in each chapter again. But this time, change them. Play and see what you can change and what you must do to get it to work if it no longer works correctly. Practice and playing with what you have already covered will reinforce your understanding of it and your ability to use it. Even though this book just teaches you the basics, it is enough to write programs. Use your imagination, experiment, and find out what you can do.

LEARN MORE!! When you have explored these ideas enough to code some of your own simple programs, continue to collect and practice more Python learning materials and source code. After you have the basics of Python, the best way to become a programmer is to dissect existing open source programs in your areas of interest. Books and materials

are important, but no book can teach you as fast or as well as if you are doing it. Start a programming library of books, learning materials, modules, programs, and code.

Make sure that you select materials, modules, libraries, etc. that are compatible with Python 3.0 or later. A new revised standard was set for Python from version 3.0 onward. Most of the learning materials and code available at the time of this writing are for earlier versions of Python. The commands given to you in tutorials and the code given to you for examples will cause a lot of frustration when they don't work correctly. If you want to take the time to learn some troubleshooting, you can collect older teaching materials and make the code changes to make them work in later versions of Python. You can have a look at Python.org to see what changes have been made. Additional study combined with trial and error will make these resources available to you. You can also search the internet for code conversion programs that can convert older code to the version you are using.

If you are just getting started and don't want to waste your time on the older resources; then make sure all Python modules, libraries, code, books, etc. that you acquire are updated for Python 3 or later. I have found that although some materials I found said they had been updated, the code didn't always work until I modified it for Python 3. Everyone makes mistakes. If you get a book that has some code errors, go to a Python community, a forum, Python. org or do searches on your specific problem. Edit the code and move on.

I hope this short book has helped you and will launch you into the world of Python programming. Good luck!

VOCABULARY REVIEW:

<i>Debugging</i> finding and solving problems in a program
<i>Exceptions</i> are important or exceptional events

Index

REVIEW DICTIONARY OF TERMS:

Algorithms	are sets of instructions that tell the computer what to do.
Assert	allows you to insert debugging code into your program
Block	is just a group of code that goes together.
Boolean	data type is referring to two possible values; True or False .
Break	breaks out of a block of code even if the while statement remains true
Comparison operators	are for comparing things, (<, >, ==, !=, etc.).
Compound condition	is a comparison using more than two values, (x < 10 and x > 5)
Conditional expressions	(also called Boolean expressions), are based on the condition that something is either true or false.
Continue	tells Python to skip the remaining statements in the current loop block and jump to the next loop block.
Debugging	finding and solving problems in a program
Delimiters	quotation marks to tell the computer we are entering a string.
Else	statement offers a choice in a while loop...do this or else.....do...
Exceptions	are important or exceptional events
Float	non-whole numbers like decimals..
Floating point numbers	numbers with a decimal point
For...in	looping statement to find if something is <u>in</u> a list
Function	a re-usable mini program inside of your program
Function call	the code you type to turn on a function.
Global statement	tells Python to be able to use the function everywhere in the program
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.
Identifiers	are names
IDLE	is software, (an editor), that helps you to communicate with the computer.
If Statement	a conditional statement/ a statement based on a condition.
If	if a statement is true...
Immutable	means we can't edit it.
Integer	a complete number as opposed to part of a number like 1/2
Interactive mode	is for testing and trying small ideas quickly.
Is	similar to the == operator
Key	the name we chose for our dictionary entry
Key-value pair	a key and its associated value
Lambda	is a variation on the def statement with other functionality (most programmers prefer to just use def)
List	a sequence of elements
Logical operators	the words "and, or," and "not"
Loop	when the computer goes back and repeats something in a cycle until something

<i>breaks it out of that cycle.</i>	
Low level languages are used when the programmer wants more direct control over the machine he is using.	
Modules are programs that can be activated in Python and provide you with more tools and functions to get things done more easily.	
Mutable means we can edit it.	
Nested , (placed), inside	
Not works like the English not and makes something the opposite meaning	
Operators do something, like add, multiply, divide, subtract, or compare.	
Pass acts as a neutral place holder in some statements	
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.	
Return returns a value from a function that can be assigned to another function	
Scope or Namespace region of influence	
Script Mode is great for writing programs you can save and run later. It is generally used for the final product.	
Simple condition is a comparison that only uses two values, (9<10)	
Slice a subset of a list	
Source code is all the algorithms, statements, and instructions that we used in a program.	
String a sequence of characters, words, or sentences in quotation marks.	
Try, except, and finally can be used for testing error messages, (advanced)	
Tuple like a list but can't be edited or changed in its original form.	
Value the details or definition for a dictionary entry/key	
Variables are like little like boxes or containers to put different things in.	
While statement tells the computer to continue looping while some condition is continuing to happen; usually while it is still true.	
While while true repeatedly run this block of code	
With statement has to do with writing new classes, (advanced Python)	
Yield has to do with execution flow control and "freezing." (advanced	

PYTHON'S MANY FLAVORS:

wxPython, cross-platform GUI toolkit, widget creation, open source code/free, (<http://www.wxpython.org>)

SciPy, scientific tools for Python, (<http://www.skypy.org>)

NumPy, science, linear algebra, (<http://numpy.scipy.org/>)

BioPython, molecular biology, (<http://www.biopython.org/>)

PyDaylight, Chemistry, (<http://www.dalkescientific.com/PyDaylight/>)

Pychart, (home.gna.org/pychart/)

PyGame, game creation, (<http://www.pygame.org>)

Gloss, added to Pygame creates fast running graphics for games, (<http://tuxradar.com/gloss>)

PyKyra, game creation, (<http://www.alobbs.com/pykyra>)

VPython, 3-D and animations, (<http://www.vpython.org>)

PYTHON GENERAL RESOURCES:

tkinter, http://www.ferg.org/thinking_in_tkinter/index.html

<http://www.python.org>

Educational materials and resources for programming and zope, web site dev.,

(<http://openbookproject.net/pyBiblio>)

<http://www.parellelpython.com>

<http://videocapture.sourceforge.net/>

Web site development, database, and search, (<http://www.zope.org/>)

High level web framework, (<http://www.djangoproject.com/>)

OTHER LIBRARIES AND MODULES MAY BE FOUND AT:

Networking Engine, (<http://twistedmatrix.com>)
Python Imaging Library and XML, (<http://www.pythonwar.com>)
494 Projects tagged “Python Modules” (<http://www.freshmeat.net/tags/Python-modules>)
3-D Modules (<http://www.vrplumber.com/py3d.py>)
Computer Graphics, (<http://cgkit.sourceforge.net/>)
Imaging Library, (<http://www.pythonware.com/products/pil/index.htm>)
<http://www.pythonsource.com>
Data tables, by Google,
(http://code.google.com/apis/visualization/documentation/dev/gviz_api_lib.html)
PHP, probably the fastest data base abstraction library, (<http://adodb.sourceforge.net>)
Finance, (<http://quantlib.org/index.shtml>)
Statistics, Many Science application links, StatPy
(<http://www.astro.cornell.edu/staff/loredo/statpy/>)
High Performance Graphics and Gaming, (<http://www.pythonorgre.com>)
Graphs, (<http://zovril.com/2009/05/26/graphy-a-chart-library-for-python/>)
Neural Networks, (<http://leenissen.dk/fann/>)
2-D Plotting, (<http://databits.Iternet.edu/node/75>)
Many, (<http://www.dmoz.org/computers/programming/languages/python/modules>)
Fast XML, PDF creation, Vector Graphics, text,
(<http://www.reportlab.com/software/opensource/>)
Widget construction kit, (<http://www.effbot.org/zone/wck.htm>)
Molecular Graphics, (<http://mgltools.scripps.edu/>)
Fluid, physics, (<http://www.elsa.onera.fr/index.html>)

Copyright 2010 © by Jody Scott Ginther
CEO of Alien Cat ® Studios a registered Trade Mark
Dedicated to Family Entertainment and Education

<http://www.AlienCatStudios.com>
<http://www.toonzcat.com>