

LEARN PYTHON IN 24 HOURS



PYTHON

PROGRAMMING GUIDE FOR BEGINNERS

AFLEXSYSTEM.COM

PYTHON MADE SIMPLE

© Copyright 2015 - All rights reserved.

This document is geared towards providing exact and reliable information in regards to the topic and issue covered. The publication is sold with the idea that the publisher is not required to render accounting, officially permitted, or otherwise, qualified services. If advice is necessary, legal or professional, a practiced individual in the profession should be ordered.

From a Declaration of Principles which was accepted and approved equally by a Committee of the American Bar Association and a Committee of Publishers and Associations.

In no way is it legal to reproduce, duplicate, or transmit any part of this document in either electronic means or in printed format. Recording of this publication is strictly prohibited and any storage of this document is not allowed unless with written permission from the publisher. All rights reserved.

The information provided herein is stated to be truthful and consistent, in that any liability, in terms of inattention or otherwise, by any usage or abuse of any policies, processes, or directions contained within is the solitary and utter responsibility of the recipient reader. Under no circumstances will any legal responsibility or blame be held against the publisher for any reparation, damages, or monetary loss due to the information herein, either directly or indirectly.

Respective authors own all copyrights not held by the publisher.

The information herein is offered for informational purposes solely, and is universal as so. The presentation of the information is without contract or any type of guarantee assurance.

The trademarks that are used are without any consent, and the publication of the trademark is without permission or backing by the trademark owner. All trademarks and brands within this book are for clarifying purposes only and are the owned by the owners themselves, not affiliated with this document.

Bonus Free Python Programming Videos

When you subscribe via email, you will get free access to a toolbox of exclusive subscriber-only resources. All you have to do is enter your email address to the right to get instant access.

To get instant access to these incredible tools and resources, click the link below: => [Click here for the bonus content](#) <=

Table of Contents

[Introduction](#)

[Chapter 1: Setting Up Python on Your computer](#)

[Chapter 2: Interacting with Python](#)

[Chapter 3: Boolean's in Python](#)

[Chapter 4: Boolean's and Conditional expressions](#)

[Chapter 5: Use of *if – elif* statements in making Multiple Tests in Python](#)

[Chapter 6: Use of Nested Control flow statements in Python](#)

[Chapter 7: Compound Boolean Expressions in Python](#)

[Chapter 8: Python Data Variables \(Numbers, Lists, Tuples, Strings and Dictionaries\)](#)

[Chapter 9: Basic Operators in Python for Calculations](#)

[Chapter 10: Opening and Closing Files](#)

[Chapter 11: Reading and Writing Files in Python](#)

[Chapter 12: Managing Databases](#)

[Conclusion](#)

[Bonus Free Python Programming Videos](#)

Introduction

I want to thank you and congratulate you for downloading the book, “PYTHON MADE SIMPLE” .

This book contains has simple lessons on one can use numbers and texts in python as a way of interacting with Python.

Every chapter in this book has well organized and numbered steps which you will need to follow so as to grow you expertise in the use of Python as scripting language. In addition, this book also provides you with codes which you will need to practice with in your Python to learn how to run Python codes.

It is my hope that typing the codes will increase your knowledge and understanding of the various capabilities that can be achieved in Python.

It is my delight that after reading this book, you should expect to have a much better understanding of how you can code and write algorithms that will simply work for you in using Numbers and Texts. This will make your life easier and increase potential as a Python Expert.

I believe that the instant you made a decision to read this book; you were headed in the right direction. It is indeed a fantastic idea to read this book. It is my anticipation that after reading this book, you will have significantly gained much knowledge and skills in using Python not only for using numbers and texts; But also your general problem solving skills in Python!

Isn't learning Python this way a fantastic idea? Well, I'm happy that you chose this book as your guide to learning Python and get to the next level in Python programming. I'm in deed very excited for the great potential.

I thank you in advance for downloading this book, I hope that you will benefit from this rich resource and hope that you practice this on a daily basis.

Chapter 1: Setting Up Python on Your computer

To be able to use python, you need to ensure that your computer has all the requirements needs to run python. This chapter offers you all the instructions you need to follow to get started. There are different settings for python depending with the computer you want to use.

If you are using Mac OS X computers: You will not need to install Python program since you already have a Python 2. Neither will you need a any other version such as Python 3.

Warning

You many need other programs such as PowerShell if you are going to use windows, Terminal when using OS X and bash if you prefer using Linux. But if this is your first time using this program, then you will need to learn how they can be used first.

However, the most basic requirements include having Python program and everything starts here: **How do I install python?**

This is a very simple process

- First, download the latest version of python and for compatibility with other programs that you may need, use the product version 2.7.1 of python. Find it in the following link: [Python Download](#).
- Do you know how to install a program on a computer? Yes, just double click the icon downloaded from your download folder and wait for further instructions.
- But, ensure that you accept all the default settings of the program till you are asked to press the finish button.
- The program will then inform you if the installation is completed.

Getting ready to use Python

- Check to see that all programs are in the right locations: start from **My Computer**, locate the directory C: \ python 27. You should be able to locate all the python program files here.

- Next, copy all the details of the directory beginning from *C:* to 27 and exit from the window.
- Then, go for the start button, and right click on ***My Computer***.
- Select *properties*.
- Select; ***Advanced System Settings***.
- Click on ***Environment Variables***.
- In that folder, search for the folder variable path.
- Go for the file called Path and right click on it, select edit.
- Search for the field called Variable Value using the right arrow.
- Introduce a semi colon at the end of the word and past the directory path of the python folder you had copied previously.
- Then press on the okay button.

How to create a first Python Program

- Go to start, and then my computer
- Select drive C and click on it.
- Click on the Python 27 folder.
- Next, right click on the space and create a new folder.
- Name the new folder *Python Programs*
- Save all the new programs you create in this folder.
- Then go to *Start* and type *Run* in the *Search* box at the bottom and press ENTER.
- Next, type *notepad* in the new window field referred to as *Open*.
- Type the following words in Notepad just as it is written here:

File: Hello.py print "Hello World!"

- Go for the file in the control panel and select Save As.
- In the section ***Save in*** browse for the *C:* drive and then select the folder *PythonPrograms*.
- For the field *File name* remove everything that is there and type in *Hello.py*.
- In the field *Save as type* select *All Files*

- Click on *Save*. You have just created your first Python program.

Run the program you have just created

- Click on the *Start* button and click on *Run*.
- Input *cmd* in typing field followed by OK.
- A new dark small window will pop up.
- In the new window, type *cd C:* and then press the key Enter.
- However, in case you type *dir*. You will get a list of all the directory folders in the drive C:\.
- Search for the folder labeled *PythonPrograms* created by you in the previous step.
- Next, type *cd PythonPrograms* and press the Enter button. This is a direct way of accessing the python programs folder.
- Just by typing *dir* and you are supposed to see the file named *Hello.py*.
- Try running the program by typing *python Hello.py* followed by Enter.
- The output should be a single line reading *Hello World!*
- This is your first program, Bravo!! You have just been able to create and run your first Python program.

Python for Mac Users When using Mac, Python comes already installed with your Mac OS X. The versions however may vary. You can download a new version of python when the version you have is a very old. This can be accessed by following the hyperlink here [Download](#). This is one of the latest binary versions of Python. It is able to runs on both Power PC and Intel systems. After the download is complete, install it on your computer.

You are now ready to create your first program!

How to start IDLE on Mac

1. While in windows we talk of the start button
2. In Mac, it is initiated from the *Terminal* window, and then typing *python*. This will start the python shell automatically.
3. Use the *>>>* to prompt for Python shell.

4. Then, you can type in the python shell prompt ***import idlelib.idle***
5. This will activate the **IDLE IDE**

Python IDLE on Windows and Mac

- Go to my programs
- Locate python on the list of programs
- Select IDLE to start IDLE
- Then go to File, and select *New Window*
- Type your program in “Hello World”
- Next, locate the File menu and click on Save. Type in *Helloworld.py*
- This allows it to be saved as a plain text file. The extension of the file name (.py) helps to enable the python text reader to be able to open it and read it. It also enables other programs which are text editors such as Notepad, Notepad+++ or TextEdit to read the program.
- After saving the program, go to *Run* and select *Run Module* or use the F5 key on your computer to run your program.
- The output should read: Hello world!

If using Mac OS X

The following will need to be done:

1. Using your browser, download the TextWrangler to be used as a text editor and install it on your computer. This can be accessed on <http://www.barebones.com/products/textwrangler/>
2. The editor (TextWrangler) to be placed in the docks where it can be accessed very easily.
3. Go to the terminal program; search for the TextWrangler and you should be able to find it with ease.
4. Next, place your terminal in your docks too.
5. Then, try to run your Terminal program.
6. In the terminal program, run python. This is performed by typing the things you need done and hitting return. So, type Python and hit RETURN.
7. Next, try to exit Python by Typing quit(), Enter. This will exit you

from Python.

8. This will **Return** you to a similar prompt that was there in the program even before you entered Python. If not there, seek to find out why this is not there.
9. You may also need to learn how one can make a directory in the Terminal.
 10. Moreover, find out how the directory can be changed from the Terminal.
 11. A file can also be created using your editor in the directory. Follow the file, then, Save or use Save As and then locate the directory you have created for saving your Programs.
 12. Try also, getting back to the terminal using your keyboard by switching windows.
 13. Get back to the Terminal and try to locate your newly created file.

In windows:

1. You may also need to have notepad++. This can be accessed from <http://notepad-plus-plus.org/> on your browser. This is an alternative text Editor. Install it on your computer. This does not require you to be the administrator.
2. Place it on the desktop or start button where it can be easily accessed or in Quick Launch. Select the most preferable options during installation.
3. Similarly, run PowerShell from the Start menu. Search for it and you can just press Enter to run it.
4. It is easier to access the program by using a short cut on the desktop and using a quick launch for convenience.
5. In addition, use the PowerShell program which is also the Terminal program.
6. In the PowerShell program, initiate python. Just like in the other terminal programs, you run programs by typing the name and then press Enter.
 - 1) In case the python installed is not responsive, download it again and

install it afresh from the following link <http://python.org/download>.

2) The best version is Python 2.7.5. The latest Version may not be compatible with some of the programs you will need to use here, therefore, don't go for Python 3 and any other.

3) In case you do not have Administrative rights, you will be better off using the ActiveState Python.

4) Check to ensure that the Python program just installed is in good working condition. If not, use PowerShell to try if it can be activated by typing the following:

```
[Environment]::SetEnvironmentVariable "Path",  
"$env:Path;C:\Python27", "User")
```

5) Then close PowerShell and restart the Program to confirm whether the Python now runs. In case it is not working, you will need to Restart your computer to integrate all the programs.

7. Next, exit PowerShell by typing quit() followed by Enter to move out of python.

8. Check if the initial prompt has been retained even before you typed python in the program. In case it is not there, seek to solve the bug in the program.

9. In the same way like TextWrangler which is also a text Editor like PowerShell, find out how you can make a directory directly in the PowerShell (Terminal).

10. Similarly, try to change into a directory in the PowerShell (Terminal).

11. Moreover, find out if you can be able to create a file in this directory using file, Save or Save As and locate the directory to which it will be stored.

12. Also, while in the PowerShell (Terminal), try switching windows using your keyboard. Search it up if you find a problem figuring it out.

13. In the PowerShell (Terminal), try to list the directory just to check if the newly created file is in the directory.

The word terminal as used in this content refers to either the "Shell" or "PowerShell" in this book.

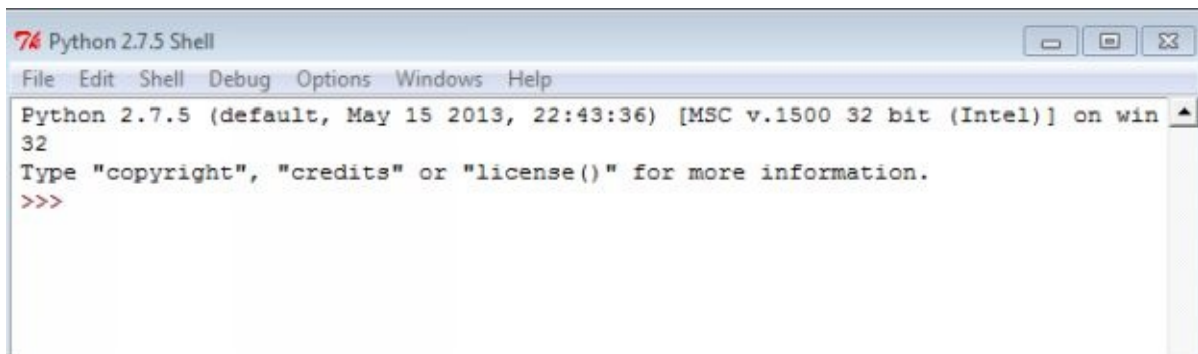
Warning

Most of the time, installing Python in Windows may face a few challenges. It may fail to get configured to the right path. This can be solved by typing the following text in the python PowerShell

```
[Environment]::SetEnvironmentVariable("Path", "$env:Path;C:\Python27",  
"User")
```

Then, either **Restart** the **PowerShell** or the whole computer so as it gets fixed.

Windows: What You Should See Just by opening Python This is what you should see



To find help, Type `help ()` followed by enter. This is what you will see!

```
>>> help ()
```

```
Welcome to Python 2.7! This is the online help utility.
```

```
If this is your first time using Python, you should definitely check out  
the tutorial on the Internet at http://docs.python.org/2.7/tutorial/.
```

```
Enter the name of any module, keyword, or topic to get help on writing  
Python programs and using Python modules. To quit this help utility and  
return to the interpreter, just type "quit".
```

```
To get a list of available modules, keywords, or topics, type "modules",  
"keywords", or "topics". Each module also comes with a one-line summary  
of what it does; to list the modules whose summaries contain a given word  
such as "spam", type "modules spam".
```

For the previous versions of Python; this is what you will see as the output:

```
> python
```

```
ActivePython 2.6.5.12 (ActiveState Software Inc.) based on  
Python 2.6.5 (r265:79063, Mar 20 2010, 14:22:52) [MSC v.1500 32 bit (Intel)] on win32
```

It is still correct if you see different information than mine, but yours should be similar.

Python on Linux Linux is a unique operating system that comes with a very different ways of installing programs on it. But, installing any software on any computer requires your ability to follow instructions as directed by the software during installation. Default settings are always recommended and vary with the OS to be used.

My assumption is that if have Linux, then, you probably understand how to install a package. This is how this can be done:

1. The best text editor in Linux is the Gedit text editor.
2. It is installed using the Linus package manager
3. Place the program gedit on the desktop as a short cut to the program or pin it on the start button for easy access.
 - 1) Run gedit to see whether the program is fully configured or it will need some quick fixes. Ensure the program has been set to the default mode.
 - 2) Next, get to Preferences and select the Editor tab.
 - 3) Then, edit the tab width to 4.
 - 4) Check on the box representing the part Insert spaces instead of tabs.
 - 5) Ensure that Automatic Indentation is turned on.
 - 6) Similarly, go to open in the file control menu, select View tab, then, check "Display line numbers" to turn it on.
4. Locate your terminal program.
5. It may be referred to as GNOME Terminal, Konsole, or xterm.
6. Transfer the Terminal to your docks.
7. Try a run of your Terminal program.
8. Run python in the terminal program by typing required command and

pressing Enter.

- 1) If your python is not responsive or missing in your Terminal. Download it address and install it. It is recommended that you try getting python version 2 and not 3.
9. Then, exit python by typing quit() and press Enter to exit from python.
10. This should return you to the python prompt before you typed Python. Incase this does not take place, try to fix the bug.
11. Try making a directory from the Terminal.
12. In addition, try changing into a directory in Terminal.
13. Create a file using your editor and save it in the directory created. This can be done by using the file and then Save or Save As. to the directory required.
14. Try shifting between windows to get to the directory.
15. Confirm in the directory whether the program created is listed.

How to find solutions in the internet The internet search engine is the best place for you to find solutions on Python programming. Most of the challenges may have already been done and solved by people and posted online. Hence, in some parts in this book; I will from time to time provide direction on how best to find solutions on the search engines.

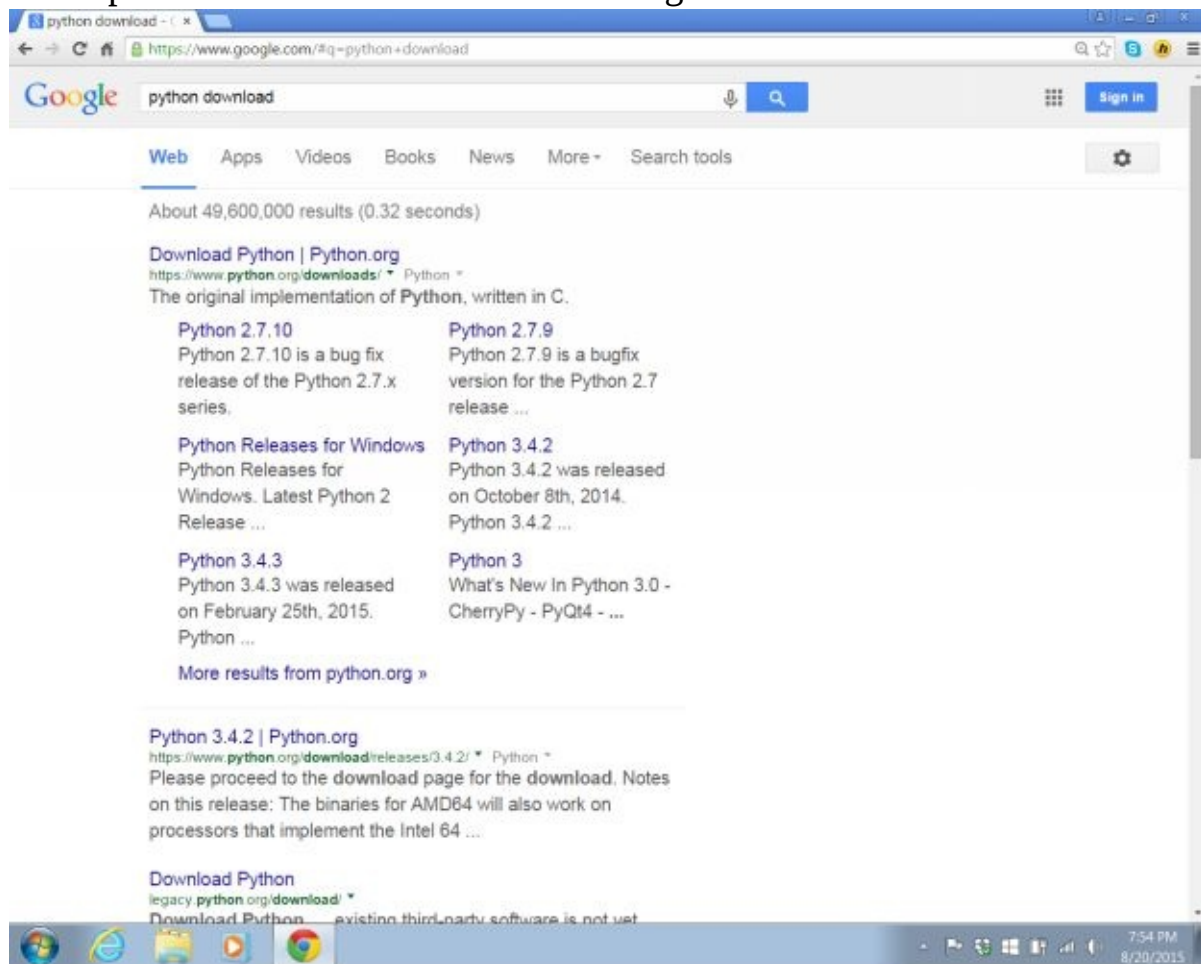
This is the best way by which you can become an independent learner in programming. The ability to trouble shoot and find solutions shows the potential of being a good programmer in the near future. This is the main goal of this book. Allowing you to exploit your potential and become the best in Python programming.

Google is the market place for all this information. Thanks to the World Wide Web developers. You can find a response to everything you need.

For instance, if you want to download Python software:

1. Go to <http://google.com/>
2. Type: python download
3. Read the websites listed to find the best answer.

A sample screenshot of this search on Google is attached below:



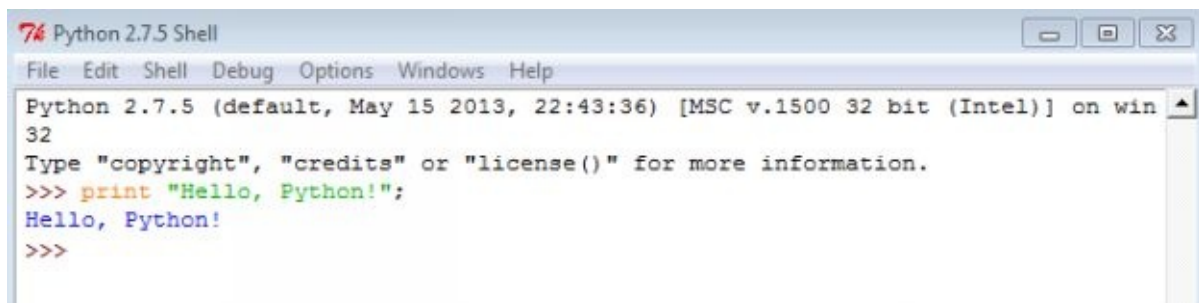
Chapter 2: Interacting with Python

To be able to interact with python: It is important that you know how, when, why and what can be done. In chapter one, I introduced to you how to invoke python.

To prompt the python program, type the text print “Hello, python!”; and press enter.

The output will be shown as:

Hello, Python! (Indicated in blue in the screen shots) This will be seen below:

A screenshot of a Windows-style application window titled "Python 2.7.5 Shell". The window has a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Windows", and "Help". The main text area shows the following text: "Python 2.7.5 (default, May 15 2013, 22:43:36) [MSC v.1500 32 bit (Intel)] on win 32", "Type 'copyright', 'credits' or 'license()' for more information.", ">>> print 'Hello, Python!'", "Hello, Python!" (highlighted in blue), and ">>>". The window has standard Windows window controls (minimize, maximize, close) in the top right corner.

This is the first program suggesting that the program is active and working.

Python Identifiers

An identifier is a name that can be used in the identification of a variable, class, function, class, module or object. In most cases, in **Naming rules in Python**

1. When using a class name, always start with an upper case letter.
2. All other identifiers must always use a lower case letters
3. When an identifier starts with a leading underscore, it is a suggestion that the identifier is private
4. Two leading identifiers suggests that the identifier is strongly private
5. Ending an identifier with two underscores suggests that the identifier is language defined with a special name.

Reserved words in Python

This are words which cannot be used as constants, variable or key words as an identifier in Python. The words are always in lower case letters only. They include:

yield	global
with	from
while	for
try	finally
return	exec
raise	except
print	else
pass	elif
or	def
not	continue
lambda	class
is	break
in	assert
import	and
if	

Lines and indentation in Python

Python as a program does not know how to identify blocks of algorithms as a block of codes for class and sometimes functions. Hence, the only way Python is able to read and understand your program is by the use of blocks of codes created by indentations.

The number of spaces in the indents is not fixed. Moreover, all statements within the codes need to be indented in the same length.

For instance, see the following code created in Python 2.7

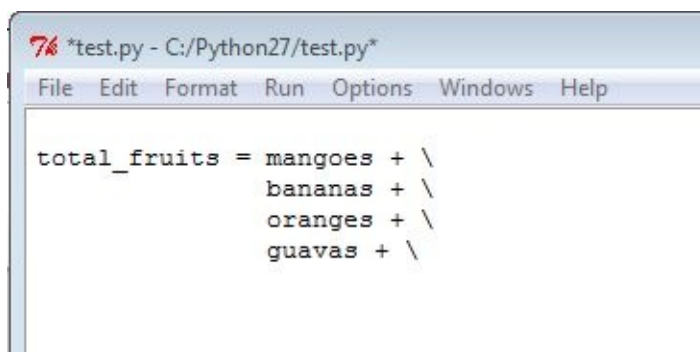
```
if True:
    print "True"
else:
    print "False"
```

Mixing the indentations in most cases bring an error in the python shell when the program is run by pressing on F5 key or by selecting run followed by run module, either way, it is the same.

```
if True:
    print "Answer"
    print "True"
else:
    print "Answer"
    print "False"
```

Multiple lines in python can also be used Most of the statements in python end up with a new line. However, line continuation can be used by the use of characters (\). This shows that the line is a continuation from the previous statement.

For example

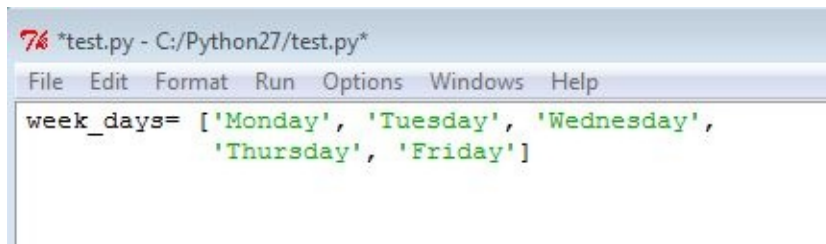


```
*test.py - C:/Python27/test.py*
File Edit Format Run Options Windows Help

total_fruits = mangoes + \
                bananas + \
                oranges + \
                guavas + \
```

However, when the statements are contained within [], (), or {} types of brackets. They do not need the use of the line continuation characters.

For examples



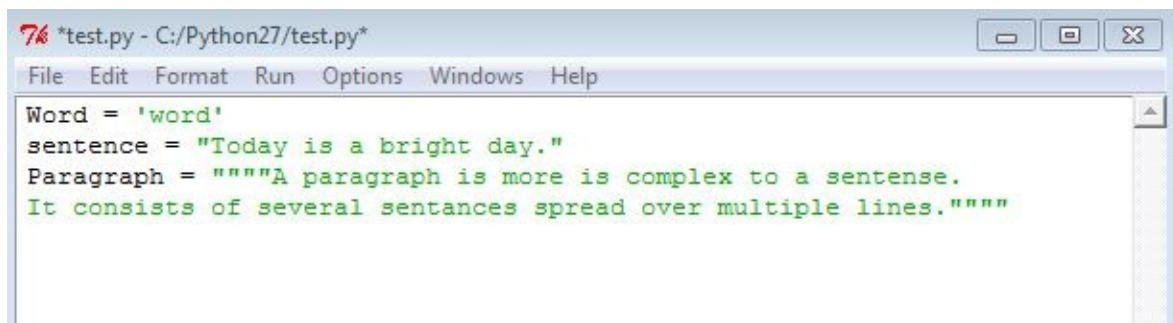
```
7% *test.py - C:/Python27/test.py*
File Edit Format Run Options Windows Help
week_days= ['Monday', 'Tuesday', 'Wednesday',
            'Thursday', 'Friday']
```

Using Quotation in python

One can use single quotes (‘’), or double quotes (“”) and sometimes triple quotes (‘‘‘ or ’’’’) as a way of representing string literals. The rule with the use of quotes is that they should be the same when being used at the start and at the end of the string.

The triple quotes are often used when spanning a string over several lines.

Check out the use of the quotes below:



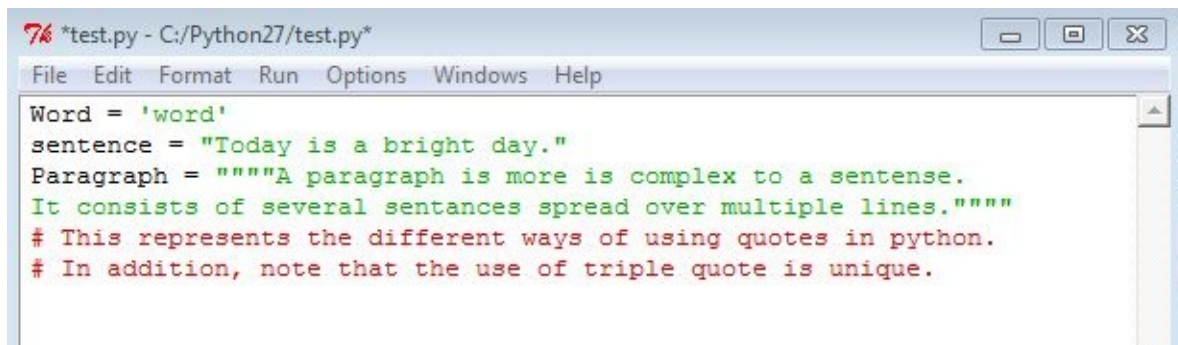
```
7% *test.py - C:/Python27/test.py*
File Edit Format Run Options Windows Help
Word = 'word'
sentence = "Today is a bright day."
Paragraph = """A paragraph is more is complex to a sentence.
It consists of several sentences spread over multiple lines."""
```

Using comments in Python

To be able to understand your program, one may need to make comments in between the codes or even after the codes. The comments can be protected from being recognized as codes in the program in a variety of ways.

A hash sign (#) is used at the start of a comment. The program would automatically assume everything starting with a hash sign. However, when it is used. The rule is that the all the comments must not be used as string literals, *i.e.* they should not be placed inside quotes.

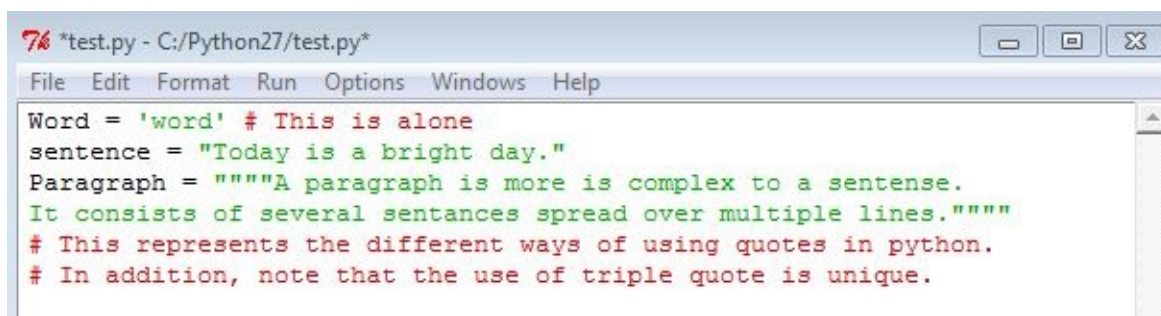
Also, all the characters that are preceded by the hash sign to the physical end of the line are recognized as part of the comment. The interpreter ignores them automatically.



```
*test.py - C:/Python27/test.py*
File Edit Format Run Options Windows Help
Word = 'word'
sentence = "Today is a bright day."
Paragraph = """A paragraph is more is complex to a sentence.
It consists of several sentences spread over multiple lines."""
# This represents the different ways of using quotes in python.
# In addition, note that the use of triple quote is unique.
```

Note also, that a comment can be placed in the following ways:

1. It may come after a line of code
2. You can also comment after as in the shown below.
3. Comments can also be in multiple lines, but every line needs to start with a hash (#) sign.



```
*test.py - C:/Python27/test.py*
File Edit Format Run Options Windows Help
Word = 'word' # This is alone
sentence = "Today is a bright day."
Paragraph = """A paragraph is more is complex to a sentence.
It consists of several sentences spread over multiple lines."""
# This represents the different ways of using quotes in python.
# In addition, note that the use of triple quote is unique.
```

Blank Lines

Blank lines in python are ignored automatically. They could be a blank line with only white space. They can also be blank line with a comment using a hash (#) sign. They are all ignored by python.

However, when developing an interactive interpreter. A blank line can be used to terminate a multiline statement.

Using multiple statements on a single line To create multiple statements on a single line, a semicolon is used. However, this is on condition that the new statement does not represent a new block of code that can initiate a different process.

The semi colons can be used as follows:

A screenshot of a Python IDE window titled "Untitled". The window has a menu bar with "File", "Edit", "Format", "Run", "Options", "Windows", and "Help". The code editor contains a single line of Python code: `import sys; x = 'foo'; sys.stdout.write (x + '\n')`. The code is color-coded: `import` is orange, `sys` is blue, `x =` is black, `'foo'` is green, `sys.stdout.write` is blue, and `(x + '\n')` is black.

Multiple Statement Groups as Suites:

When there are groups of a single code expressed as single statements in Python, they are referred to as suites.

Compound statements involving the use of words such as `if`, `def`, `while`, `class`; require a header line and a suite.

Moreover, also important are the header lines. They start a statement with a key word and they terminate with a colon (:). They require a few other lines so as they can make up a suite.

Here is an example of the use of `if`, `elif` and `else` to make a suite.

Note: the indentation as used is important.

A screenshot of a Python IDE window titled "Untitled". The window has a menu bar with "File", "Edit", "Format", "Run", "Options", "Windows", and "Help". The code editor contains a Python `if-elif-else` suite. The code is color-coded: `if` is orange, `expression` is blue, `:` is black, `suite` is black, `elif` is orange, `expression` is blue, `:` is black, `suite` is black, `else` is orange, `:` is black, and `suite` is black. The suites are indented under their respective header lines.

In summary

Interacting with python requires that you should be able to comprehend the basic rules for using Python. A full functional code will require that the right indentations, blank lines, colons, comments and other interacting approaches are correctly applied.

Chapter 3: Boolean's in Python

Simple Conditions

Boolean is a test or conditional system first developed by a mathematician called George Boole. This has been derived and used in Python. It is set as a condition or a test. In this chapter, simple tests will be done to show comparison of values. This will also be translated to ensure that you comprehend every aspect of the comparison in the handling of the arithmetic in Python. Using the following examples, try them out in python and get to know what happens with each one of them in Python *Shell*.

```
2 < 6
3 > 8
x = 12
x > 11
2 * x < x
type(True)
```

The conditions as used in this case can either be expressed as true or false. They are also not expressed in quotes because that will change their meaning and operations completely. There is a perfect example for the use of Boolean values developed in the 19th century.

The application of Booleans in python is the short form of the name to bool. This is applied in tests results for the use of expressions: True or False conditions or tests.

This is the simple history, how the Boolean's started and what they represent in Python.

Simple if, else, elif Statements use and application in Python

If statement

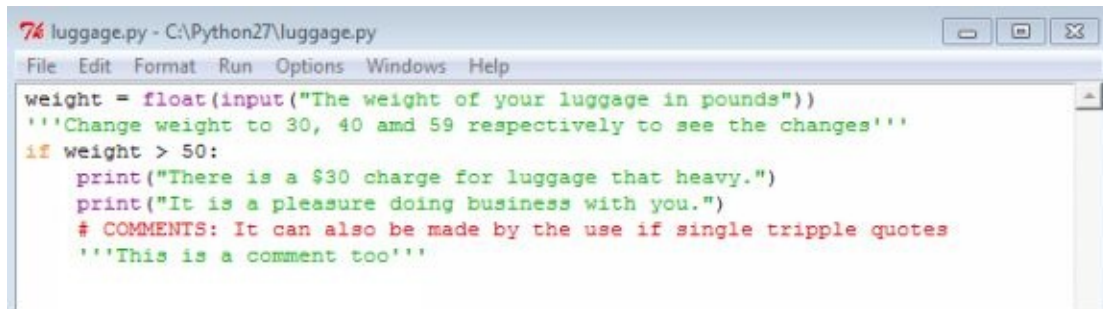
There are various statements that can be use in Python. They are a perfect set of conditional expressions which can be used to express logic and provide a response in the form of an output in Python.

For example, try out this short code and save it as luggage.py. Run it twice or

thrice but changing the values of the input 30, 40 and then 56.

Note that the input determines the nature of the result.

I am providing you with screen shots so that in the process of writing the codes on your program, you also learn and see how the programs is supposed to be indented and many other applications covered previously.

A screenshot of a Python IDE window titled 'luggage.py - C:\Python27\luggage.py'. The window has a menu bar with 'File', 'Edit', 'Format', 'Run', 'Options', 'Windows', and 'Help'. The code editor shows the following Python code:

```
weight = float(input("The weight of your luggage in pounds"))
'''Change weight to 30, 40 and 59 respectively to see the changes'''
if weight > 50:
    print("There is a $30 charge for luggage that heavy.")
    print("It is a pleasure doing business with you.")
    # COMMENTS: It can also be made by the use if single tripple quotes
    '''This is a comment too'''
```

The second line of the code is an if statement. It can be interpreted to be like a story being read in an English book. When it is true that the weight of the luggage is greater than 50, then a statement of the extra charge is made. But, when it is not true and the value is lower than 50, then don't do the indented part of the code: skip the indented parts of printing the charge value for the extra luggage.

In every of the events, whenever the if code has been executed. It makes a decision on whether to proceed with the indented parts or not.

Hence, the next line will be printing the statement, "thank you for doing business"

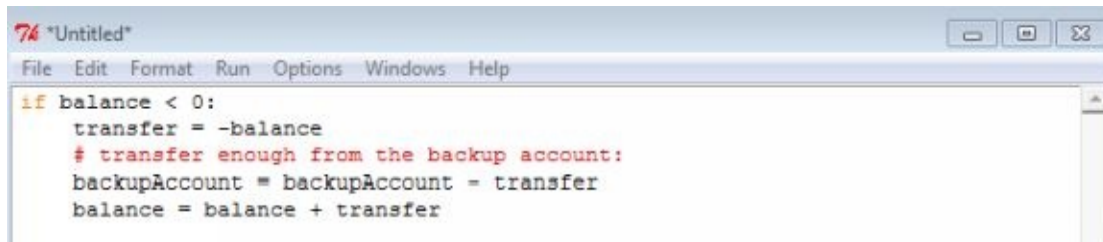
Overall, the if statement general Python syntax is expressed as follows

if **condition** :

 Statement (s) block

What this represents is that: If the condition will be in deed true as stated, then proceed to the indented statements. But, if the condition set is not applicable in this case and not true, then skip the indented statements.

For more practice, let us consider a teller machine in a supermarket which executes the following code in every transaction is conducts:



```
if balance < 0:
    transfer = -balance
    # transfer enough from the backup account:
    backupAccount = backupAccount - transfer
    balance = balance + transfer
```

This is an if expression with a block of indented statements. The block may have a single or several lines of statements. The main meaning of the code is that whenever the balance of the account goes to negative. It is returned to 0 through a series of other steps which will be able to transfer some from the back up account to ensure the account is always at 0 (zero).

This means that when the condition is true, then it has a choice of doing something. But, when the condition is false, then it has nothing to do and only say “Thank you and Good Bye”

if-else Statements

The general syntax for this statement is expressed as follows:

If expression:

Statement (s)

else:

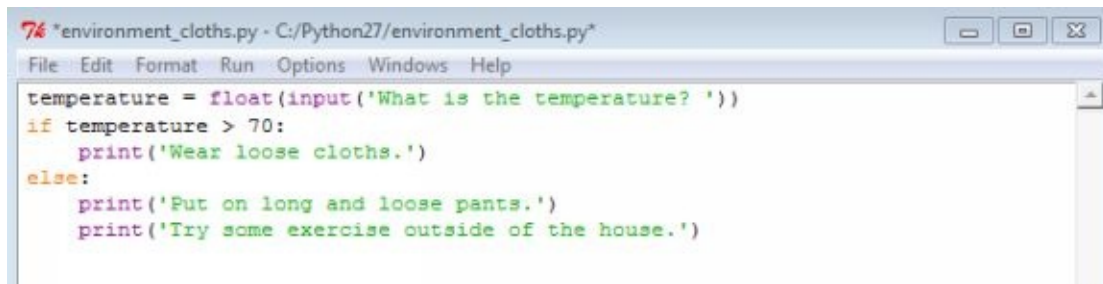
Statement (s)

The statement blocks can have as many numbers of statements as they can be included in the program.

Let’s try to understand the if ---else statements by looking at the following example:

Create a program below in your python and name it `environmental_cloths.py`. Then run the program and see the results. You can also play around with the input values as 60 and 85 respectively to see what the results will look like.

You are expected to find different results with different inputs. Here is the code for you application.



```
*environment_cloths.py - C:/Python27/environment_cloths.py*
File Edit Format Run Options Windows Help
temperature = float(input('What is the temperature? '))
if temperature > 70:
    print('Wear loose cloths.')
else:
    print('Put on long and loose pants.')
    print('Try some exercise outside of the house.')
```

Just like I have expressed in the previous example, this codes can be read like an English story. Other words such as otherwise can be used instead of the word else, however, we use shorter words in Python to reduce the bulk of wording in programming.

Note also, that the code has two indented blocks of code. The after the if and after the else. The first block is the if statement. The indentation is executed only of the if is true otherwise, nothing happens. Next, when it is false, the else is executed as well. Hence, when using the if.. else statement, only one of the headings can be executed at any one, when if is true, else indented block is not executed, but when if is false, then the else indented block is executed.

Chapter 4: Boolean's and Conditional expressions

Most of the mathematical comparisons can be made in python. However, they lack the symbolism due to the absence of the standard keys on almost all the keyboards in existence

Below is a representative coding for arithmetic symbols used in mathematics and python.

Meaning	Math Symbol	Python Symbols
Less than	<	<
Greater than	>	>
Less than or equal	\leq	<=
Greater than or equal	\geq	>=
Equals	=	==
Not equal	\neq	!=

Note: There need not be space between the symbols when used in python.

Also note that a single equals sign is used for assigning values in Python as previously covered. Hence, when used in tests, a second equals sign needs to be used in Python. It is annoying, is it?

Therefore, when using a single equal sign for testing for equality in python and when not intending to use it for assigning values to a variable; this will automatically generate an error.

Testing for equality never assign values and hence never need you to place a variable on the left of the double equal symbols. Most of the expressions may be tested for equality or inequality (!=). Hence, no numbers may be required,.

Using the following values, try to make prediction of the results. Each line can be tested on its own in python shell.

```
x = 6
x
x == 6
x == 8
x
x != 8
x = 9
4 == x
4 != x
'hi' == 'h' + 'i'
'HI' != 'hi'
[1, 2] != [2, 1]
```

Note: Assignment cannot be made using an equality check. Strings are always case sensitive. Moreover, order is very important in all lists.

Using this example, try it out in Python Shell.

```
'b' > 6
```

Often, when the comparison is not what is expected or makes sense, an exception is made for the code.

The inexactness in Python is not acceptable especially when using float values and strings. The following examples can also be used: $0.3 + 0.4$ to be equal to 0.7 . Try creating a code that will be able to express the following expression in the Python Shell.

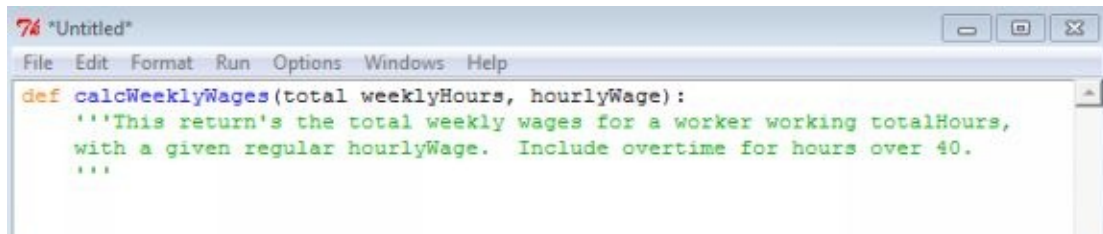
A perfect practical example can be expressed using the following pay code for normal and overtime. Using an individual's work schedule for working hours for the week and regular wage for every hour worked for the week and with consideration of the hourly overtime.

When the hours worked are over 40, then they are considered as overtime. They

are then paid double the normal rate. This can allow us to make a python code that will be able to do the calculation for the money that is payable per employee.

The set up for the function can be expressed as follows:

Try to read the set up for the function as expressed below:



```
def calcWeeklyWages(total weeklyHours, hourlyWage):  
    '''This return's the total weekly wages for a worker working totalHours,  
    with a given regular hourlyWage. Include overtime for hours over 40.  
    '''
```

This code will be able to handle two cases, when less than 40 hours of total time is worked and when more than 40 hours of total hours worked is covered.

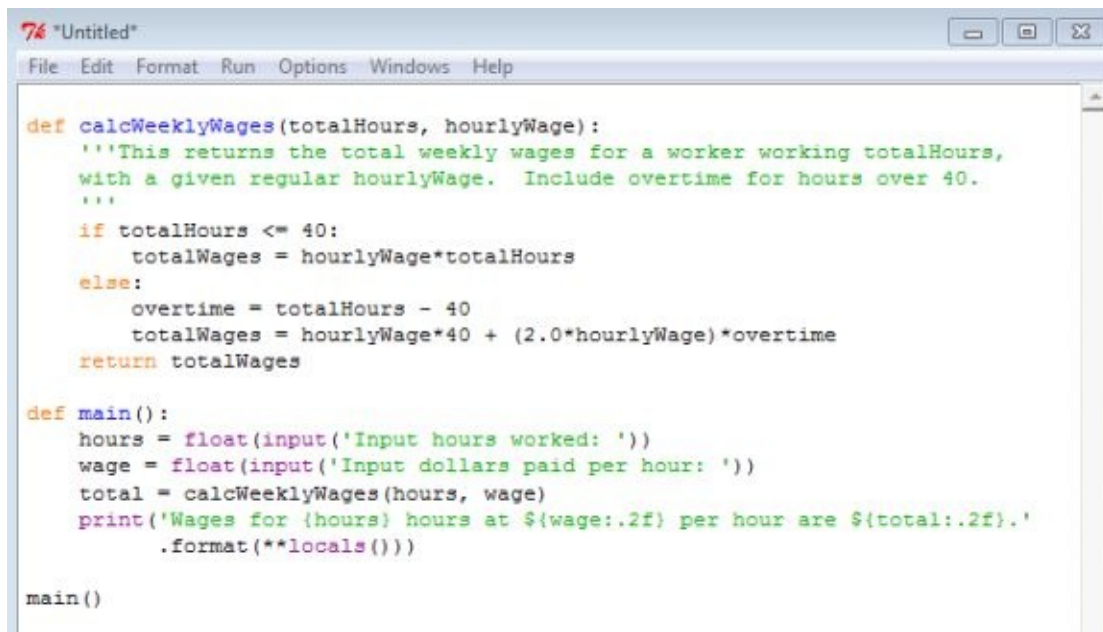
When more than 40 hours of total work time is covered, then, a new variable will have to be introduced called overtimeHours. This will be able to manage the amount of time worked in excess of the 40 hours total work time and included in the total wage for the month.

Since we are training you as a programmer. Try to see how this solution can be read in English.

Then, try to run this program out and see the results:

Use this programs named wages.py as an example of a program for this exercise.

This program has adopted the use of floating stings point format. ([String Formats for Float Precision](#)) so as to allow the use of the cents in the answer to two decimal places:

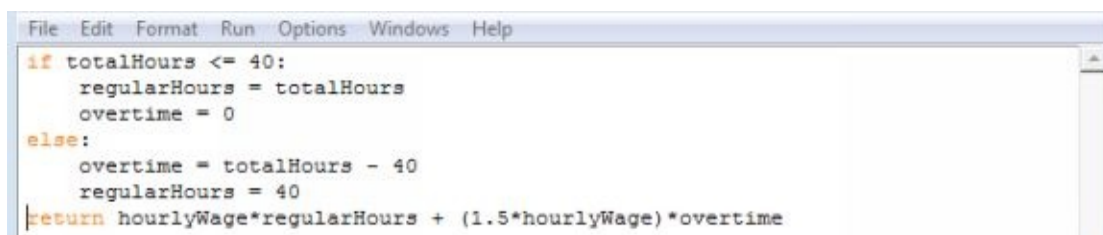


```
def calcWeeklyWages(totalHours, hourlyWage):  
    '''This returns the total weekly wages for a worker working totalHours,  
    with a given regular hourlyWage. Include overtime for hours over 40.  
    '''  
    if totalHours <= 40:  
        totalWages = hourlyWage*totalHours  
    else:  
        overtime = totalHours - 40  
        totalWages = hourlyWage*40 + (2.0*hourlyWage)*overtime  
    return totalWages  
  
def main():  
    hours = float(input('Input hours worked: '))  
    wage = float(input('Input dollars paid per hour: '))  
    total = calcWeeklyWages(hours, wage)  
    print('Wages for {hours} hours at ${wage:.2f} per hour are ${total:.2f}.'  
        .format(**locals()))  
  
main()
```

In most cases, this will be able to manage the numeric values but this can accommodate the decimal conversions from strings via floats and not integers.

Alternatively, a different version of the expression code for the body of calculating weekly wages can be used in the code used above.

This is just a general formula which will be able to manage the expression in the formula and allow the parameters to accommodate the if statement. The same problem can be solved using many other ways in way that will still provide you with the same output.



```
if totalHours <= 40:  
    regularHours = totalHours  
    overtime = 0  
else:  
    overtime = totalHours - 40  
    regularHours = 40  
return hourlyWage*regularHours + (1.5*hourlyWage)*overtime
```

Practical Exercise

Assignment One: Graduationlist.py

Develop a program name graduation.py. This program should be able to classify students on the basis of how many credits they have. The program should be able to print a statement on whether they have enough credits to be able to graduate. At Kenyatta University, it is a requirement that one needs about 120 credits so that they can be able to graduate.

Assignment 2: Head or Tails Exercise

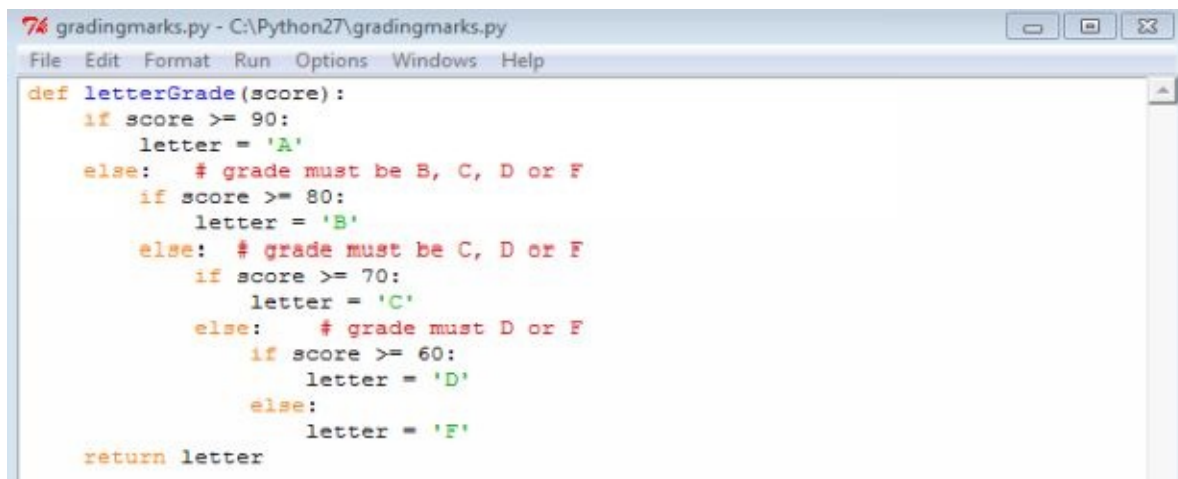
Develop a program and name it ht.py. Ht refers to heads and tails. The code needs to introduce a function flip () , this will represent a single flip of a coin which will be able to represent either the head or a tail. You try come up with a range of numbers by the use 0 or 1 with a random.randrange (2), Also, use the if --- else statement that will allow the printing of heads when the result is zero (0) and tails when the results is (1).

Create a simple repeat loop which will call flip () 12 times so as to try your code if can generate a random sequence of 10 heads and tails.

Chapter 5: Use of *if – elif* statements in making Multiple Tests in Python

When one would like to create more than two distinct cases and the conditions have only two possible conditions which can either be true or false. The only direct choice may be between the other two options. For instance, when one has about 20 questions, you can be able to make more cases by use of more questions. When there are more than two choices, a single test can be used to reduce the possibilities and other tests added will be able to lower the possible further. Statements will be required to be placed in a indented statement block and another added choice will be an extra if statement to the block.

For instance, a teacher would like to make a conversion table for numerical marks to grades (A, B, C, D, E or F). The marks limits are 90, 80, 70, 60 respectively. Hence, this can be resolved very easily by stating all the possible options in the else clause:



```
def letterGrade(score):  
    if score >= 90:  
        letter = 'A'  
    else: # grade must be B, C, D or F  
        if score >= 80:  
            letter = 'B'  
        else: # grade must be C, D or F  
            if score >= 70:  
                letter = 'C'  
            else: # grade must D or F  
                if score >= 60:  
                    letter = 'D'  
                else:  
                    letter = 'F'  
    return letter
```

It is not mandatory that the indentation be used for every case when using the if -- else expression. It is indeed unique and must be strictly followed in this case.

However, the indentation can be cleared using an alternative if ... elif clause. The elif combines both the if and else clause to become an elif block in the python code:

See this code below and note the indentation applied in this case.


```
# Options using an elif clause:
def letterGrade(score):
    if score >= 90:
        letter = 'A'
    elif score >= 80:
        letter = 'B'
    elif score >= 70:
        letter = 'C'
    elif score >= 60:
        letter = 'D'
    else:
        letter = 'F'
    return letter
```

The syntax applied in this use of if, elif –else clause is illustrated below. This helps you

```
File Edit Format Run Options Windows Help
if condition1 :
    StatementBlockForTrueCondition1
elif condition2 :
    StatementBlockForFirstTrueCondition2
elif condition3 :
    StatementBlockForFirstTrueCondition3
elif condition4 :
    StatementBlockForFirstTrueCondition4
else:
    StatementBlockForEachConditionFalse
```

Note that all use of if, elif, and the final else are all aligned on one line. The number of elif to be used is not limited with every of the line followed by an indented block.

What happens is that one of the indented block will be the one to be executed, when it is the one that corresponds to the first true conditions set using the if statement. Otherwise, when all the condition are false, the final else will be executed.

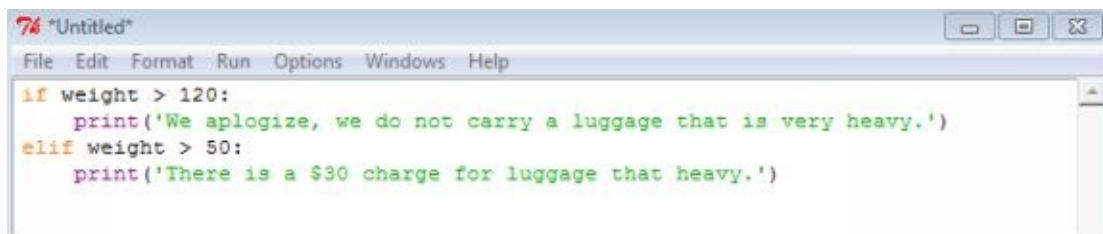
The use of *if* and *elif* is an important forms of python Contraction. Care need be observed when using the contracted form of python codes. It is supposed to written as *elif* and not elseif. The gradingmarks.py is a good example of the use of this code illustrated above.

Alternative syntax

The other way in which the if statement can be applied is by the use of the if-elif-and removing the else statement in the code. This suggests that the syntax if-elif- else is modified so that the final block of code with else is removed or omitted. This is the same as the normal if statement that does not include an else in its code. This allows for the no block of code to be executed and only occurs when none of the conditions in the tests is true. However, when an else code is included, one of the indented code has to be executed. In the absence of the else code; at most one of the code as to be executed.

This code will only be able to print a line whenever there will be a problem with the suitcase.

Here is an example of the code:

A screenshot of a code editor window titled "Untitled*" with a menu bar (File, Edit, Format, Run, Options, Windows, Help) and standard window controls. The code inside is as follows:

```
if weight > 120:
    print('We apologize, we do not carry a luggage that is very heavy.')
elif weight > 50:
    print('There is a $30 charge for luggage that heavy.')
```

PRACTICE EXERCISE

Sign Exercise

Develop a program and save it as `identity.py` to be used to collect numbers from clients. Print out the number and categorize it as positive, negative or zero

Grade Exercise

Write the program `gradingmarks.py` and save it now as `gradingmarks2.py`.

Make modification of the python code so as to have a letter grade function that makes the opposite of the grades starting from F, E,D, ... A.

Try to run your code and make different inputs to ensure that your code is working and providing you with the right output in Idle.

Wages Exercise

Similarly, make modification of the `wages.py` and save it as `wages1.py`. However, ensure that you create a problem that indicates that people are paid double the money for any hours worked above 50 hours per week. This will show that they get to be paid normal rate within normal hours for 40 hours a week, and \$1.5 for hours worked extra of 40 but less 50 hours and double any hours worked above 50 hours.

This can be expressed as below for a total of:

$$10*40 + 1.5*10*20 + 2*10*5 = \$800.$$

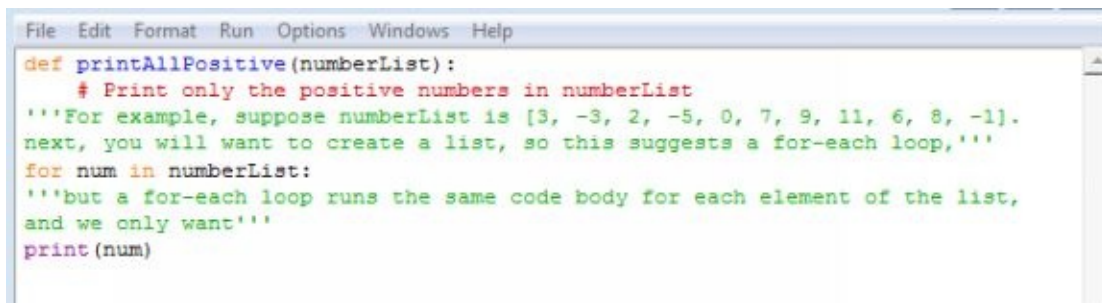
Run your program several time and make modifications to ensure that it provides you with the right calculations. Use comments where appropriate to help people understand your code.

Chapter 6: Use of Nested Control flow statements in Python

By now I believe you are becoming conversant with the abilities and potential of Python language. It can be applied in a variety of ways as simple statements and as well as complex or combined statements. One such example is well demonstrated in the use of for-if and else statements.

In this case now, note that for and if statements can be nested inside each other in an indented block of code. For instance, from a series of values, one can be able to only select positive values from a mix of numerical in an arbitrary list of numbers.

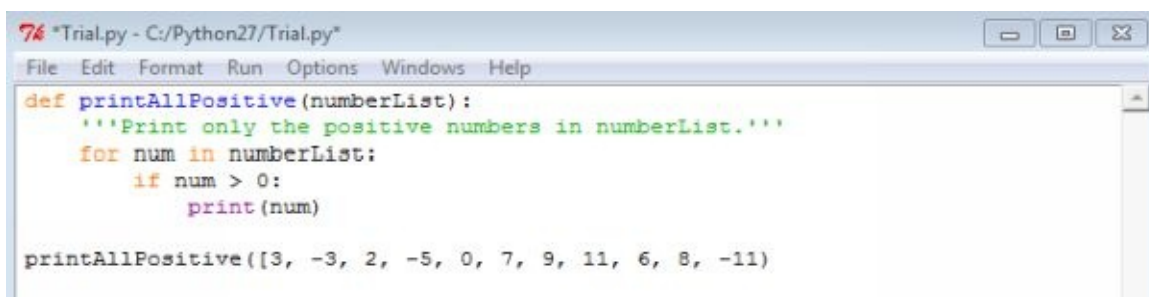
For example. Using the following code, it is easier to select only positive values from a mix of numbers:



```
File Edit Format Run Options Windows Help
def printAllPositive(numberList):
    # Print only the positive numbers in numberList
    '''For example, suppose numberList is [3, -3, 2, -5, 0, 7, 9, 11, 6, 8, -1].
    next, you will want to create a list, so this suggests a for-each loop,'''
    for num in numberList:
        '''but a for-each loop runs the same code body for each element of the list,
        and we only want'''
        print(num)
```

Just for a moment, try to read the program by following the comments provided in the code. Using a human eye, you will be able to select the numbers carefully, but using a computer, this will be made systematically checking every number consistently. Using the if statement, every number can be checked determine whether it can be printed.

This can be observed below:



```
76 *Trial.py - C:/Python27/Trial.py
File Edit Format Run Options Windows Help
def printAllPositive(numberList):
    '''Print only the positive numbers in numberList.'''
    for num in numberList:
        if num > 0:
            print(num)

printAllPositive([3, -3, 2, -5, 0, 7, 9, 11, 6, 8, -11])
```

Nesting of the statements as in the use of If expands the possibilities of

applications of Loops. This will allow you to be able to use different things in loops and allowing consistency and decision making between alternatives.

Besides the if loops, there are also the while loops which also work with the if statements. Nesting if statements and the while loop allows you to be able to make far better nested programs.

Pythons can be used to express graphical interpretation of results.

For instance, when a balloon is released and allowed to bounce within a room, this can be expressed in Python to expresses the randomness of times it will bounce off to the edges of the room.

This can be allowed to operate from random locations which can be applied in Python. The program can be run in python shell.

Find a sample program that can be used to express a bouncing balloon.

```
bounceBall (-3, 1)
```

This parameter provides the amount and shape of movement of the balloon when bouncing in every of the animation. Other values can also be expressed in shell and magnitude of less than 10 values can be used.

Animations before the use and applications were purely scripts, one was required to define the direction and how many moves to be considered. However, the direction of every motion may change with every bounce. Hence, in python, the graphics objects shapes combined with a central animation step.

This can be expressed in the following example code:

```
shape.move(dx, dy)
```

The use of the values, dx and dy allows the values to change as the ball gets to the boundary. Using an example, a bouncing balloon may get to the left side as it is moving to the left and up. The bouncing effect will have impact on the nature of the horizontal movement but the ball will keep with its upward motion. A reverse effect in the motion of the horizontal movement can also be captured in Python. This allows the bouncing balloon to change horizontal shift and hence the negative sign to be used in coding will be the reverse. This can be expressed as below:

$$dx = -dx$$

In this case, the value of dy will not change. However, this may not be the case in every of the animation step since it does not happen every other time. This therefore provides for the use of if statement. The center of the balloon can be estimated using the coordinates which is expressed in the form of x and y (x,y). The lowest point the bouncing balloon can reach can be expressed as $xLow$.

The value of $xLow$ should always be greater than 0 while the value of edge of the window is also at a coordinate 0. Otherwise, at this position, the values of the ball would be half way of the screen. The x coordinate of the center of the balloon also needs to be the length of the radius.

The process of animation takes place in small steps and this is possible by allowing the ball to take one first small and quick step of where it needs to go ($xLow$), the ball can then also be reversed to the original position where it belongs.

There are associated bounding variables that can be used to express the bouncing variables $xHigh$, $yLow$ and $yHigh$, which represents the radius away from the edge in the form of coordinates and same conditions to assess the possible bouncing off each of the possible edge. One coordinate will automatically reverse in whichever way as long as one of the edge is hit. This code can be expressed as follows:

```
if x < xLow:
    dx = -dx
if x > xHigh:
    dx = -dx
if y < yLow:
    dy = -dy
if y > yHigh:
    dy = -dy
```

This can also be expressed using the `elif` sentence structure. This will allow for the extra testing. This will imply that if is indeed true that $x < xLow$, then, it may be impossible for it to be true in the case when $x > xHigh$. Therefore, both tests are not needed together. The use of the `elif` clause will also provide for elimination of the unnecessary tests when both x and y is to be used.

```

if x < xLow:
    dx = -dx
elif x > xHigh:
    dx = -dx
if y < yLow:
    dy = -dy
elif y > yHigh:
    dy = -dy

```

The middle if should not be changed to make it an elif as used in many other cases. This is because there is a possibility that a ball can reach a corner and therefore, it may require that both dx and dy should be changed.

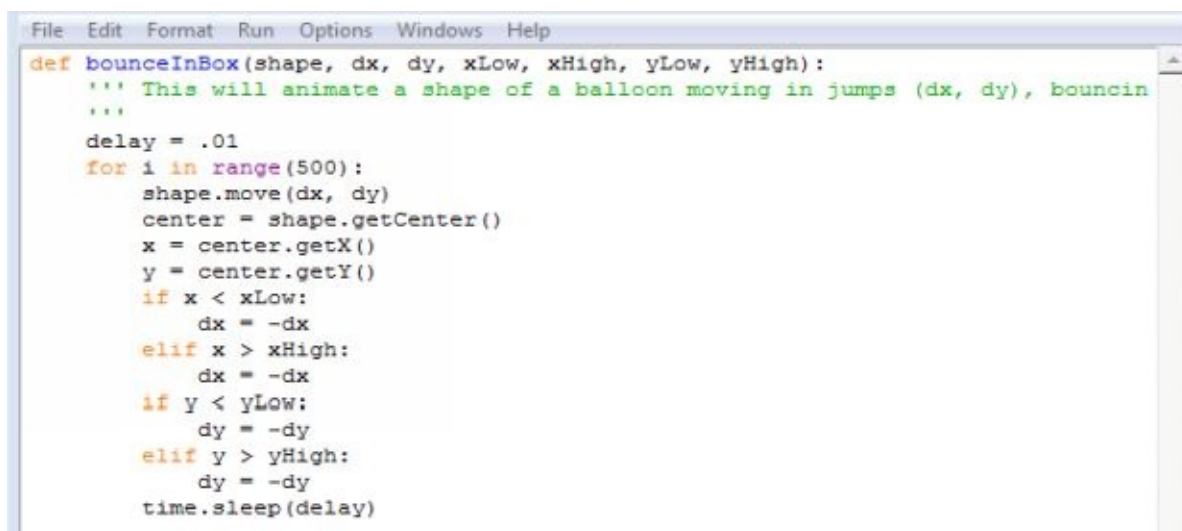
A variety of access methods for graphics are present that can be used using different shapes such as circles which can be defined by describing the centre point.

They can be accessed by using getCenter() method. This will be able to generate a clone of the points which can be returned. Every coordinate can be retrieved by using getX() and getY() criteria.

This may be a new feature that can be applied in the definition of the central function of a bouncing balloon in a box bounceInBox.

This animation will be able to develop a repeat loop which can be accomplished through a series of 600 steps.

This can be captured in this experiment.



```

File Edit Format Run Options Windows Help
def bounceInBox(shape, dx, dy, xLow, xHigh, yLow, yHigh):
    ''' This will animate a shape of a balloon moving in jumps (dx, dy), bouncin
    ...
    delay = .01
    for i in range(500):
        shape.move(dx, dy)
        center = shape.getCenter()
        x = center.getX()
        y = center.getY()
        if x < xLow:
            dx = -dx
        elif x > xHigh:
            dx = -dx
        if y < yLow:
            dy = -dy
        elif y > yHigh:
            dy = -dy
        time.sleep(delay)

```

This will move a balloon while bouncing from any point within a rectangle. This

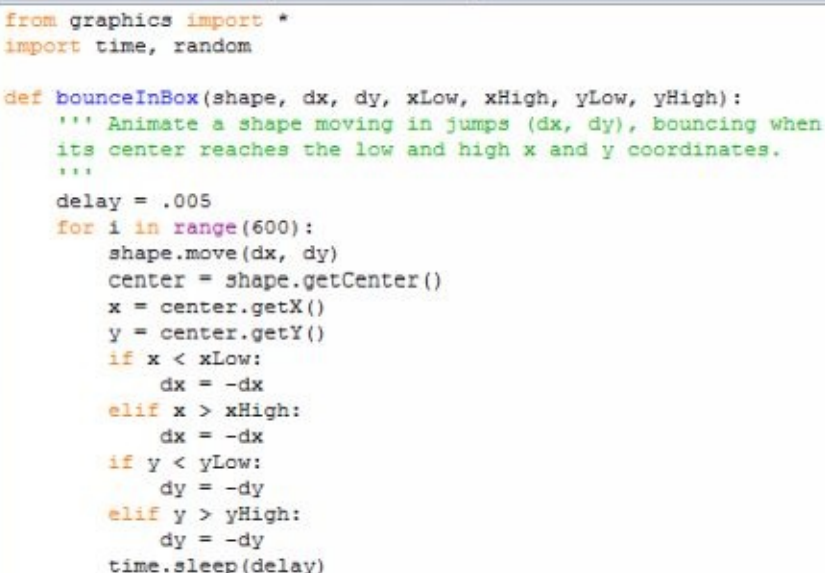
is what is expressed in the function of the code: getRandomPoint. This function allows the use of randrange function derived from the main module random.

Parameters with functions range and randrange expresses the end which has been defined in the past and which is the last value that is actually desired.

```
def getRandomPoint(xLow, xHigh, yLow, yHigh):  
    '''Return a random Point with coordinates in the range specified.'''  
    x = random.randrange(xLow, xHigh+1)  
    y = random.randrange(yLow, yHigh+1)  
    return Point(x, y)
```

The full program that will allow one to animate a complete program that will be able to show a repeating bouncing balloon/ball is as expressed below. Note that there is the use of the function, bounceInBox and getRandomPoint to as the program to be complete.

These are also not the only functions that can be done. Not everything has been described as demonstrated in the code below: many other things will need more research to understand how they function. But, they can be used together so that a program will be able to function appropriately. This program will be able to show a ball bouncing off the sides of the window.



```
File Edit Format Run Options Windows Help  
from graphics import *  
import time, random  
  
def bounceInBox(shape, dx, dy, xLow, xHigh, yLow, yHigh):  
    ''' Animate a shape moving in jumps (dx, dy), bouncing when  
    its center reaches the low and high x and y coordinates.  
    '''  
    delay = .005  
    for i in range(600):  
        shape.move(dx, dy)  
        center = shape.getCenter()  
        x = center.getX()  
        y = center.getY()  
        if x < xLow:  
            dx = -dx  
        elif x > xHigh:  
            dx = -dx  
        if y < yLow:  
            dy = -dy  
        elif y > yHigh:  
            dy = -dy  
        time.sleep(delay)
```



```

def getRandomPoint(xLow, xHigh, yLow, yHigh):
    '''Return a random Point with coordinates in the range specified.'''
    x = random.randrange(xLow, xHigh+1)
    y = random.randrange(yLow, yHigh+1)
    return Point(x, y)

def makeDisk(center, radius, win):
    '''return a red disk that is drawn in win with given center and radius.'''
    disk = Circle(center, radius)
    disk.setOutline("red")
    disk.setFill("red")
    disk.draw(win)
    return disk

def bounceBall(dx, dy):
    '''Make a ball bounce around the screen, initially moving by (dx, dy)
    at each jump.'''
    win = GraphWin('Ball Bounce', 290, 290)
    win.yUp()

```

```

radius = 10
xLow = radius # center is separated from the wall by the radius at a bounce
xHigh = win.getWidth() - radius
yLow = radius
yHigh = win.getHeight() - radius

center = getRandomPoint(xLow, xHigh, yLow, yHigh)
ball = makeDisk(center, radius, win)

bounceInBox(ball, dx, dy, xLow, xHigh, yLow, yHigh)
win.close()

bounceBall(3, 5)

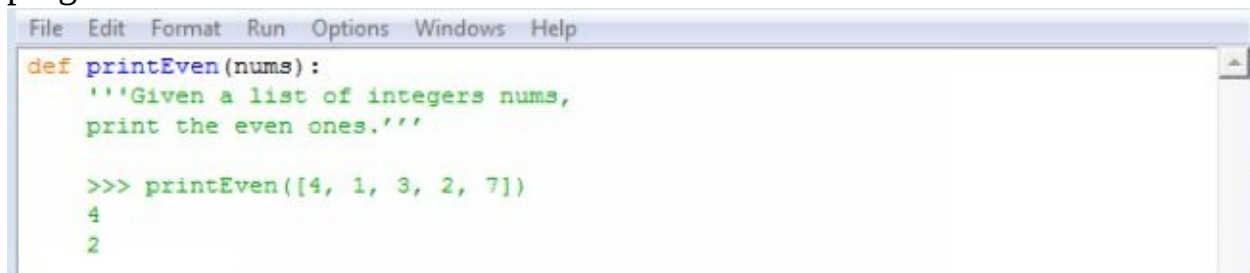
```

Practical Exercise Write a program excellent.py that will have a function printShort in the heading: The program should be able to read a list of strings and then it should be able to print the one with at most three characters.

Test the program developed using a variety of hints to determine the len of functions.

This should be tested in the python shell interaction.

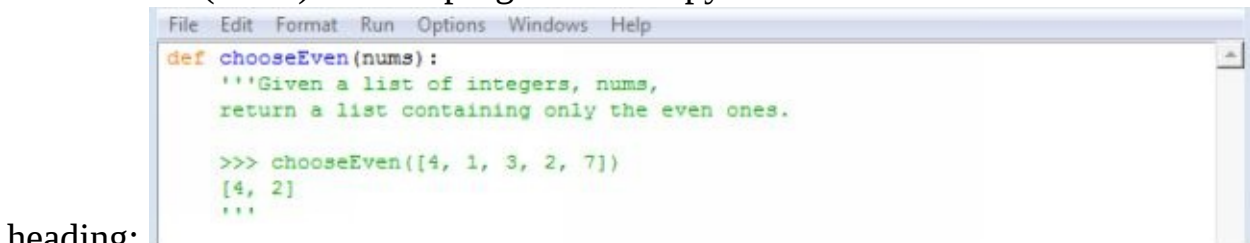
This will; have to start with the line >>> **Even Print Exercise** Develop a program and name it good.py with a function printEven and heading: Use different numbers and ensure that it will be able to print even numbers. The program will be similar to this one:



```
File Edit Format Run Options Windows Help
def printEven(nums):
    '''Given a list of integers nums,
    print the even ones.'''

    >>> printEven([4, 1, 3, 2, 7])
    4
    2
```

Even List Exercise Just like above, create another program and name it good2.py. This should be able to select only even numbers using the function chooseEven (nums) Write a program even2.py with a function chooseEven with



```
File Edit Format Run Options Windows Help
def chooseEven(nums):
    '''Given a list of integers, nums,
    return a list containing only the even ones.'

    >>> chooseEven([4, 1, 3, 2, 7])
    [4, 2]
    '''
```

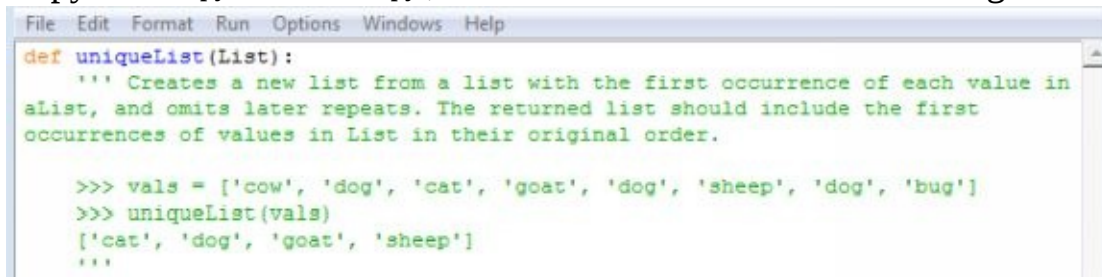
heading:

Test the function and confirm whether only the even numbers are chosen from the list required.

Unique List Exercise Create a new program named crazylib.py which can be used in getting the keys using the getKeys function. The program will be able to create a list of every occurrence of keys from a cue developed in a story format. The keys will be developed in the order in which they appear in the list. However, they will have repetitions. The original version used to get keys and deleting every duplicate in the data allowing the formation of sets from the keys. The only challenge in that the sets may not be ordered. Consequently, when the

program iterate through the codes, the order of the cues will bear no resemblance to the order in which they occur.

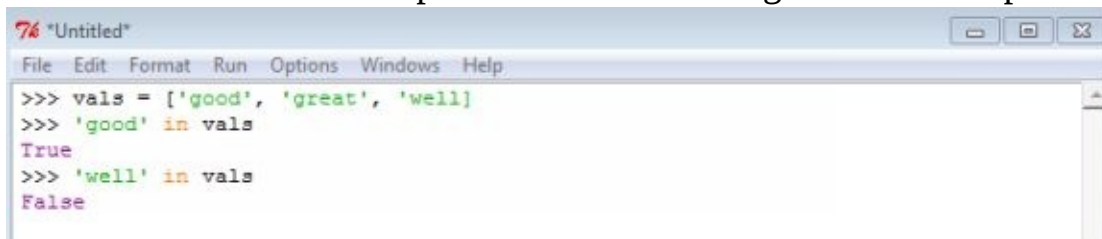
Copy madlib2.py to madlib2a.py , and add a function with this heading:



```
File Edit Format Run Options Windows Help
def uniqueList(List):
    ''' Creates a new list from a list with the first occurrence of each value in
    alist, and omits later repeats. The returned list should include the first
    occurrences of values in List in their original order.

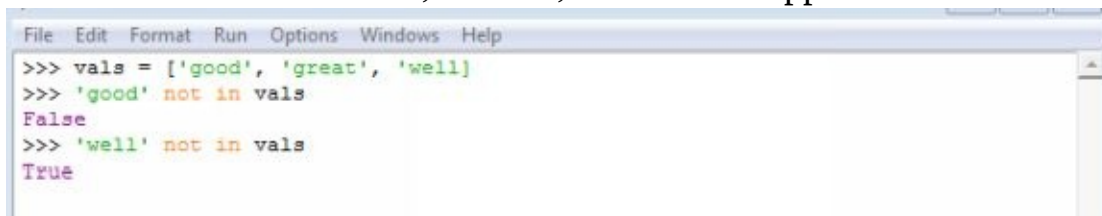
    >>> vals = ['cow', 'dog', 'cat', 'goat', 'dog', 'sheep', 'dog', 'bug']
    >>> uniqueList(vals)
    ['cat', 'dog', 'goat', 'sheep']
    '''
```

One can use the Boolean operators in for checking of membership in a sequence:



```
76 *Untitled*
File Edit Format Run Options Windows Help
>>> vals = ['good', 'great', 'well']
>>> 'good' in vals
True
>>> 'well' in vals
False
```

It can also be used with not , as not in , to mean the opposite:



```
File Edit Format Run Options Windows Help
>>> vals = ['good', 'great', 'well']
>>> 'good' not in vals
False
>>> 'well' not in vals
True
```

Hence, what you can note here is that the work is in two forms:

1. This determines an item that is in the sequence
2. This is an item that is not in the sequence

To be able to make this a success. One needs to make sure that the list is in order. A syntax can also be used to append elements to a new lists which may or may not be in the new list.

After using the uniqueList function, one can also replace the last line of the getKey so that it can also use the uniqueList so as to remove the duplicates in the keyList.

Try to create this and run it to ensure that cues values are in order.

Chapter 7: Compound Boolean Expressions in Python

Perhaps, this can be best expressed using valid examples. Let us consider this case below:

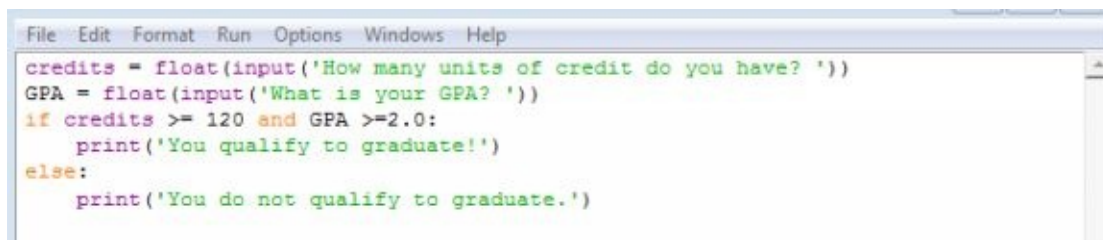
For one to be eligible for graduation from a university in US, it was set that students needed 120 credits and a total GPA of minimum 2.0.

Using Python, create a program that can be used to select students that would be graduand's for the year.

This suggests that this can only be true when credits will be equal to or greater than 120.

While GPA was also greater or equals to 2.0.

A good program will look like this:



```
File Edit Format Run Options Windows Help
credits = float(input('How many units of credit do you have? '))
GPA = float(input('What is your GPA? '))
if credits >= 120 and GPA >= 2.0:
    print('You qualify to graduate!')
else:
    print('You do not qualify to graduate.')
```

Application of compound expressions

This will show that a compound condition will be true if both of the conditions are true. However, it will be seen as false when at least one of the conditions is false.

Exercise

Previously, we had discussed the use of if and elif statements in cases where both tests where they will appear in the same block when the condition to be applied was true.

```
if x < xLow:
    dx = -dx
elif x > xHigh:
    dx = -dx
```

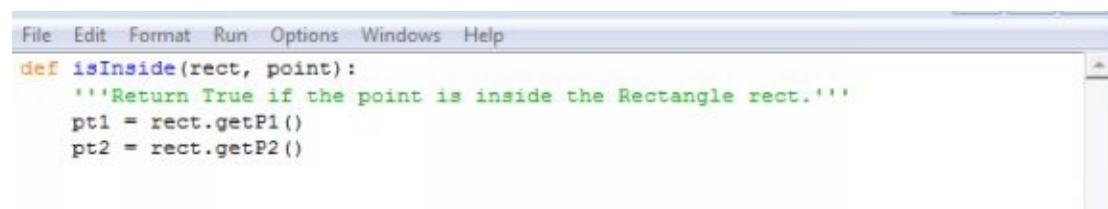
However, this can be stated in a much easier way as shown below: For instance, if $x < x_{\text{Low}}$ or $x > x_{\text{High}}$, switch the sign of dx . This can be translated directly into Python:

```
if x < xLow or x > xHigh:
    dx = -dx
```

The use of the word `or` creates a compound condition:

It is recommended that most of the complicated tests can be enshrined inside a function.

It is often convenient to encapsulate complicated tests inside a function. Think how to complete the function starting:



A rectangle is defined by two diagonally opposite points. This is how a rectangle can be prepared. The two corner points are defined by the function `getP1` and `getP2`. Python calls the points obtained as `pt1` and `pt2`. The values of the points can be expressed in the coordinate of x and y of `pt1`, `pt2`. Similarly, the point can also be expressed in the point form of `getX()` and `getY()`.

Just in case we introduce variables for the x coordinate points of `pt1`, `point`, `pt2`. The same coordinates can also be expressed as `end1`, `va1` and `end 2`. The relationship between the two coordinates can be expressed using mathematical concepts.

For instance, this can be expressed as:

$$\text{end1} \leq \text{val} \leq \text{end2}$$

However, this may not be sufficient to express the position of the corners. It is expected that the corners points are supposed to be diagonally opposite, and the second coordinates also need to be high compared to the first points. For instance, `end1` may be 300; `end 2` may be 150, while `va1` may be 180. Note that

the val is between the two coordinates of end1 and end2. These values can be substituted into the expression.

300 <= 180 <= 150

This expression is false and the values of 150 and 300 will need to be reversed. This can only be a complicated situation. Also this is an issue which must be revisited for both the x and y coordinates. The solution this can be solved using a new function isBetween to handle one coordinate at a time. Introduce an auxiliary function isBetween to deal with one coordinate at a time.

```
File Edit Format Run Options Windows Help
def isBetween(val, end1, end2):
    '''Return True if val is between the ends.
    The ends need not be in increasing order.'''
```

This expression may only be true when the original expression, end1 is less of equal to val and is also less or equals to end2. This will be true. There is also a likelihood that the case may be reversed. This two options can be combined as two unique possibilities using Boolean connectors *and* and *or*. This will make the relationship true in one of the relationship and using or, this will be the right connective.

Therefore, the complete code in python that will be able to function here would be expressed in the following ways:

A correct but redundant function body would be:

```
def isBetween(val, end1, end2):
    '''Return True if val is between the ends.
    The ends need not be in increasing order.'''

    if end1 <= val <= end2 or end2 <= val <= end1:
        return True
    else:
        return False
```

As a technique of understanding your code: try to extract the meaning of the code by interpreting the code to English.

This means that if the statement is True, return True. Similarly, if the condition is False, return False. In both condition, as described, the same values will be returned as set out in the test conditions.

```
def isBetween(val, end1, end2):  
    '''Return True if val is between the ends.  
    The ends need not be in increasing order.'''  
  
    if end1 <= val <= end2 or end2 <= val <= end1:  
        return True  
    else:  
        return False  
    return end1 <= val <= end2 or end2 <= val <= end1
```

Summary

It is not a requirement that you need an if-else statement so that you can be able to choose between True or False values. This can be used directly using the Boolean expressions as described before.

Similarly, other expressions when using the compound expressions as in this case:

`end1 <= val <= end2`

The main characters are end1 and end2. This is simply a standard mathematical syntax which can be used in chaining comparisons. The numbers of comparisons can be chained using these expressions using approximate notations. Note also that this can only be successfully coded in Python unlike other programs like Java, and C++.

They can also be translated in ways that can be read in many other common languages as shown by the expression below:

`end1 <= val`

`val <= end2`

On the other hand, an auxiliary function isBetween can also be used. This is an isInside function. The isBetween function can be used to find the values of the x coordinates; and also to check for the values of the why coordinates as well.

```
isBetween(point.getX(), p1.getX(), p2.getX())  
# used separately  
isBetween(point.getY(), p1.getY(), p2.getY())
```

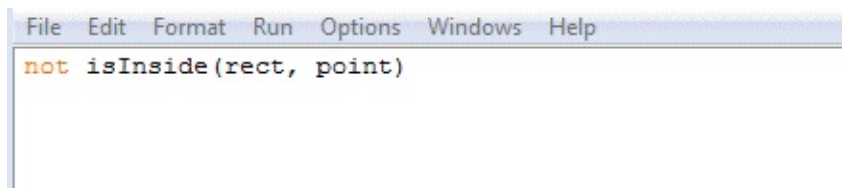

The two questions can be combined and the two point may be required to be between the sides and between the top and bottom, similarly, the right connectors to be applied is *and*.

But, how do you complete the `isInside` method? Probably, you will need to find this out. Find out more information from the link below:

<http://anh.cs.luc.edu/python/handson/3.1/handsonHtml/ifstatements.html#id16>

Also note that when you want to solve for an opposite condition. In many cases borrowing from English, a word *not* is introduced in the expression.

If a point is not inside the required position such as not in a rectangle, you will be prompted to state that the point is not inside a rectangle (`rect`). This condition can also be applied in this case.

A screenshot of a code editor window. The window has a menu bar with the following items: File, Edit, Format, Run, Options, Windows, and Help. Below the menu bar, the text `not isInside(rect, point)` is displayed in a monospaced font. The word `not` is highlighted in orange, and `isInside` is highlighted in blue. The rest of the code is in black.

Therefore, this a not *condition*.

It is only True when *condition* set is False. It is also False when *condition* is True.

A program below is an example of creating values inside a rectangle and is referred to as `selectbutton.py`.

This is a complete program that applies the use of *isInside* expression as a simple application for selecting colors. Do not care so much about the lengths of the rectangles.

A number of efforts can be done to ensure that the code can be shortened to ensure that small rectangles can be used in this case and make it a more powerful code:

This program applies the use of `isBetween` and `isInside` which are discussed below: This programs makes make colored rectangles which can be used as buttons and which can also be used as picture component. The code that has been used in creating the rectangle is similar and is inside a function

makeColoredRect. This is a good example for use of graphical tools in Python.

The code that has been developed is quite long and is enshrined with the graphical codes for drawing pictures and buttons and has sections with queries for asking the use to select the colors for a picture. You will also observe the codes have an if-elif-else test to check on which buttons have been pressed and set the colour of the picture element as commanded by the program.

```
'''Make a choice of colors via mouse clicks in Rectangles --
A demonstration of Boolean operators and Boolean functions.'''

from graphics import *

def isBetween(x, end1, end2):
    '''Return True if x is between the ends or equal to either.
    The ends do not need to be in increasing order.'''

    return end1 <= x <= end2 or end2 <= x <= end1

def isInside(point, rect):
    '''Return True if the point is inside the Rectangle rect.'''

    pt1 = rect.getP1()
    pt2 = rect.getP2()
    return isBetween(point.getX(), pt1.getX(), pt2.getX()) and \
           isBetween(point.getY(), pt1.getY(), pt2.getY())

def makeColoredRect(corner, width, height, color, win):
    ''' Return a Rectangle drawn in win with the upper left corner
    and color specified.'''

    corner2 = corner.clone()
    corner2.move(width, -height)
    rect = Rectangle(corner, corner2)
    rect.setFill(color)
    rect.draw(win)
    return rect
```

```

def main():
    win = GraphWin('pick Colors', 400, 400)
    win.yUp() # right side up coordinates

    redButton = makeColoredRect(Point(300, 340), 70, 20, 'blue', win)
    yellowButton = makeColoredRect(Point(300, 300), 70, 20, 'green', win)
    blueButton = makeColoredRect(Point(300, 260), 70, 20, 'red', win)

    house = makeColoredRect(Point(50, 190), 170, 140, 'orange', win)
    door = makeColoredRect(Point(80, 140), 30, 90, 'white', win)
    roof = Polygon(Point(40, 190), Point(240, 190), Point(140, 290))
    roof.setFill('black')
    roof.draw(win)

    msg = Text(Point(win.getWidth()/2, 375), 'Click to choose any color.')
    msg.draw(win)
    pt = win.getMouse()

    if isInside(pt, redButton):
        color = 'red'
    elif isInside(pt, yellowButton):
        color = 'yellow'
    elif isInside(pt, blueButton):
        color = 'blue'
    else :
        color = 'white'
    house.setFill(color)

    msg.setText('Click to choose a door color.')
    pt = win.getMouse()

    if isInside(pt, redButton):
        color = 'red'
    elif isInside(pt, yellowButton):
        color = 'yellow'
    elif isInside(pt, blueButton):
        color = 'blue'
    else :
        color = 'white'
    door.setFill(color)

    win.promptClose(msg)

main()

```

The only new aspect with this program will be the use of the long statements `isInside` as shown below:

```

return isBetween(point.getX(), pt1.getX(), pt2.getX()) and \
       isBetween(point.getY(), pt1.getY(), pt2.getY())

```

Python is an interactive program that will be able to read the statements as they continue to the next line even when there is an unmatched pair of brackets and parenthesis. It is recommended that when you have a long line that will be able

to run over the screen or paper. This should be continued to the next line.

To make a final character on a line with a backslash (\) will be useful since it suggests that the line continues to the next line. This is the best way of making your work neat so that almost all the statements will be able to fit into the same line.

In addition, this will also be the best way of making your codes simple. Unlike Python, many other programs will require a special way of terminating a line such as ';' which serves as command for the program not to pay attention to the new lines created. In some cases in Python, an extra parenthesis can also be used and it will do no harm to the program as shown in the above example using the *isBetween*.

Practical Exercise You are eligible to vie for a US senator and US Representative position when you are at least 30 and 25 years old respectively. In addition, you must have been a US Citizen for at least 9 and 7 years respectively for these positions.

Write a program and call it congress.py to obtain the age and length of citizenship from the user to contest for the two positions.

Enhance your program so that you can be able to obtain the age, length of citizenship and print one of the statements listed below

1. You are eligible for both senate and representative and Senate.
2. You eligible only senate
3. You are ineligible representative.

Chapter 8: Python Data Variables (Numbers, Lists, Tuples, Strings and Dictionaries)

Variables are reserved memory locations in python where values can be stored. In Python, you are allowed to create a variable which you can reserve in the memory.

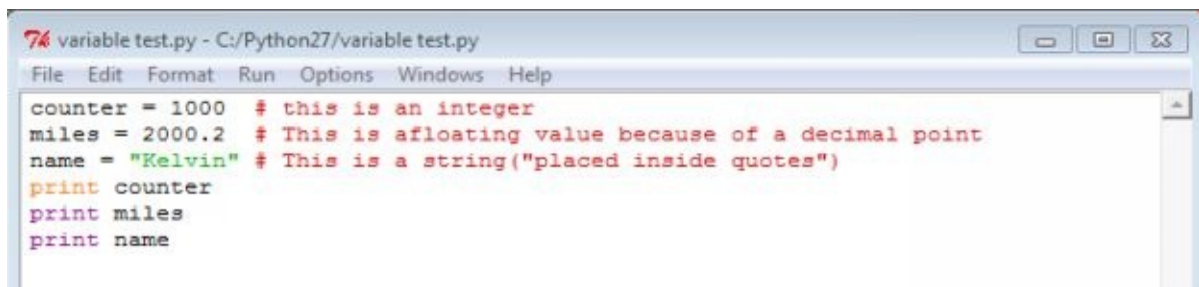
In the basis of the type of data input on python, the interpreter will be able to provide memory and make a decision on whether it can be stored on the memory reserved for the operation. Hence, when you assign a different data type to specific variables as you decide, you will be able to keep the integers, decimals and or characters in this variable.

How do you assign a value to a variable?

An equal sign is used as an operand for assigning variable to a name. The operand to the left of the equal sign is referred to as the name of the variable while the operand to the right hand side is the value of the variable to be stored in Python.

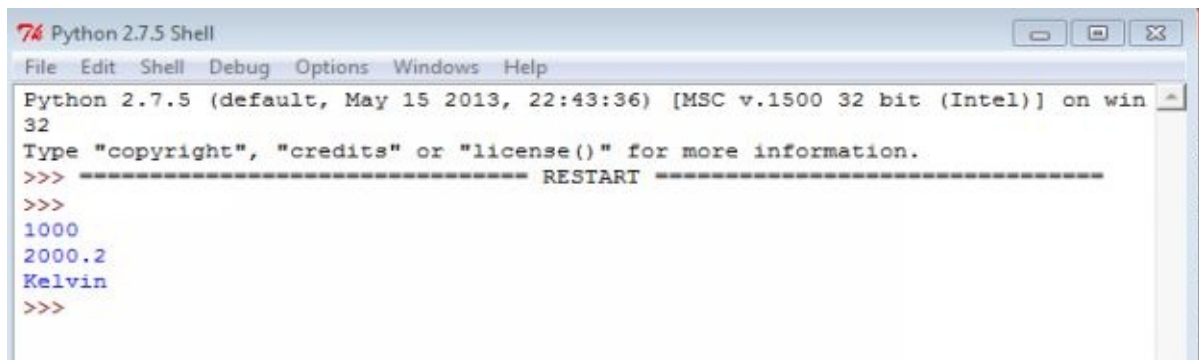
Name of variable = Value of the variable

This can be shown in the figure below:

A screenshot of a Python IDE window titled 'variable test.py - C:/Python27/variable test.py'. The window contains a menu bar with 'File', 'Edit', 'Format', 'Run', 'Options', 'Windows', and 'Help'. The code editor shows the following Python code:

```
counter = 1000 # this is an integer
miles = 2000.2 # This is a floating value because of a decimal point
name = "Kelvin" # This is a string("placed inside quotes")
print counter
print miles
print name
```

When this code is saved and run. The following results are shown in the output.

A screenshot of a Python 2.7.5 Shell window. The window has a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Windows', and 'Help'. The main text area shows the following content: 'Python 2.7.5 (default, May 15 2013, 22:43:36) [MSC v.1500 32 bit (Intel)] on win 32', 'Type "copyright", "credits" or "license()" for more information.', '>>> ----- RESTART -----', '>>>', '1000', '2000.2', 'Kelvin', '>>>'. The text is displayed in a monospaced font with some color coding: '>>>' is red, '1000' is blue, '2000.2' is green, and 'Kelvin' is blue. The window title bar says 'Python 2.7.5 Shell'.

It is a simple process. Just type in your python program the text as provided and it should offer you the same output as shown in the example above.

There are various data types that can be used in python:

They include:

1. Numbers
2. Strings
3. Lists
4. Tuple
5. Dictionaries

Numbers in Python

You can create a number as an object when you assign a value to them. Just like the name and value of the variable operands are stated, they can be used also for assigning numbers. For instance: Var1 = 2

Var2 = 4

Types of numerical supported by Python

Python best works with the following numericals

1. int (recognized as integers)
2. long (long integers [are represented as hexa or octadecimals])
3. float (floating point real values)
4. complex numbers

Examples

Int = 10, 100, -792, -0x250, 0x75

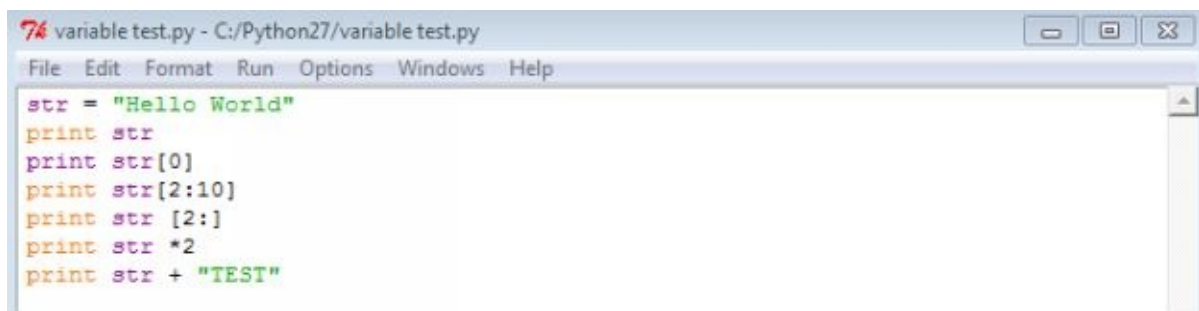
Long numbers = 51924576L, 0122L, 0xDEFABCGDRT, 53267687675L, -05238675443L

Float = 0.0, 14.90, -21.9, 32.3 +e20, -90

Complex numbers = 3.14j, 45.j, 9.322e-36j, 0.876j

Python strings

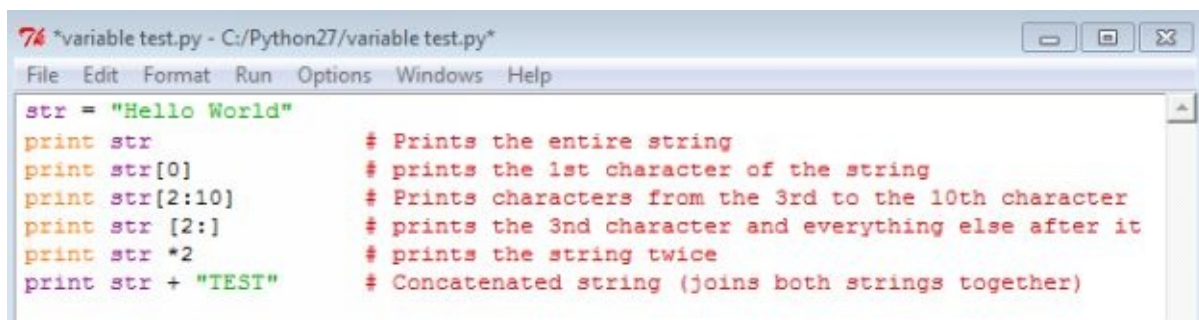
A string is a contiguous set of characters that are placed in between the quotation marks For example:



```
7% variable test.py - C:/Python27/variable test.py
File Edit Format Run Options Windows Help
str = "Hello World"
print str
print str[0]
print str[2:10]
print str [2:]
print str *2
print str + "TEST"
```

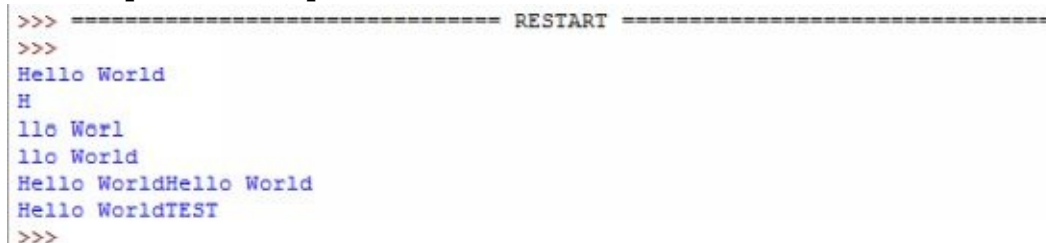
What does this program do?

See the comments in the program:



```
*variable test.py - C:/Python27/variable test.py*
File Edit Format Run Options Windows Help
str = "Hello World"
print str           # Prints the entire string
print str[0]        # prints the 1st character of the string
print str[2:10]     # Prints characters from the 3rd to the 10th character
print str [2:]      # prints the 3rd character and everything else after it
print str *2        # prints the string twice
print str + "TEST"  # Concatenated string (joins both strings together)
```

The output with respect to the comments in the same order is as shown below:



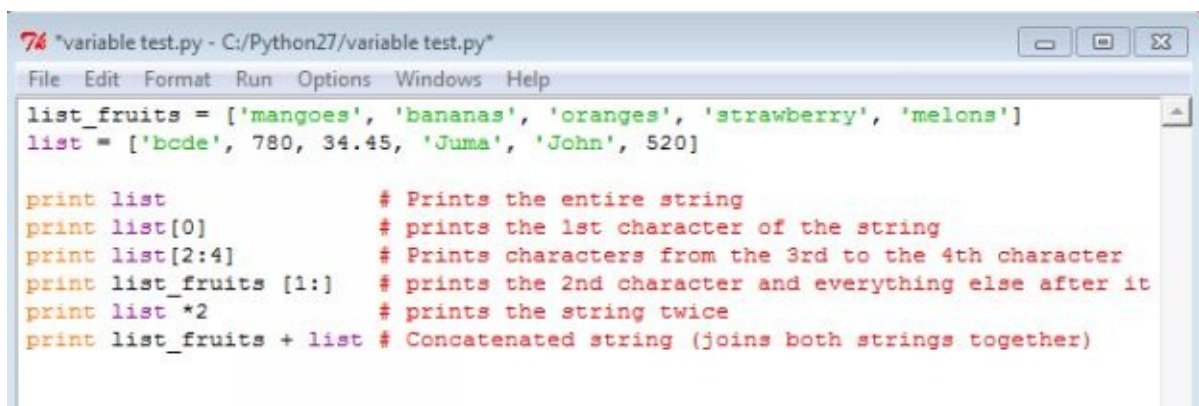
```
>>> ----- RESTART -----
>>>
Hello World
H
llo Worl
llo World
Hello WorldHello World
Hello WorldTEST
>>>
```


For more help on strings on python for using classes, functions and data, just type `[help ("string") or help ('string') and then press enter]` in the python shell.

A full list on how the classes can be implemented will be provided for you to follow and how they can be applied.

Python Lists:

Python lists are very variable and use in compound data types. A list is simply a number of items that are separated by commas and enclosed within square brackets ([]) For example:



```
74 "variable test.py - C:/Python27/variable test.py"
File Edit Format Run Options Windows Help

list_fruits = ['mangoes', 'bananas', 'oranges', 'strawberry', 'melons']
list = ['bcde', 780, 34.45, 'Juma', 'John', 520]

print list                # Prints the entire string
print list[0]             # prints the 1st character of the string
print list[2:4]           # Prints characters from the 3rd to the 4th character
print list_fruits [1:]    # prints the 2nd character and everything else after it
print list *2             # prints the string twice
print list_fruits + list  # Concatenated string (joins both strings together)
```

Running this program will produce the following output in the order of the command print

```
>>> ===== RESTART =====
>>>
['bcde', 780, 34.45, 'Juma', 'John', 520]
bcde
34.45, 'Juma'
['bananas', 'oranges', 'strawberry', 'melons']
['bcde', 780, 34.45, 'Juma', 'John', 520, 'bcde', 780, 34.45, 'Juma', 'John', 520]
['mangoes', 'bananas', 'oranges', 'strawberry', 'melons', 'bcde', 780, 34.45, 'Juma', 'John', 520]
>>>
```

Python has an interactive command for help on lists. Just type `[help ('list') or help ("list") followed by enter]` in python shell. This will provide a list of all forms of classes that can be applied on lists **Python Tuples**

This is another form of a sequence data type. It is very similar to the lists in python. It also consists of values that are separated by commas. They are however enclosed by parentheses ().

They are often considered as read only lists.


```
variable test.py - C:/Python27/variable test.py
File Edit Format Run Options Windows Help

tuple = ('mangoes', 'bananas', 'oranges', 'strawberry', 'melons')
tiny_tuple = ('bcde', 780, 34.45, 'Juma', 'John', 520)

print tuple           # Prints the entire string
print tuple[0]        # prints the 1st character of the string
print tuple [2:4]     # Prints characters from the 3rd to the 4th characte
print tiny_tuple [1:] # prints the 2nd character and everything else after
print tiny_tuple *2    # prints the string twice
print tuple + tiny_tuple # Concatenated string (joins both strings together)
```

Outputs of the above code in the order of the print command are as follows:

```
>>>
('mangoes', 'bananas', 'oranges', 'strawberry', 'melons')
mangoes
('oranges', 'strawberry')
(780, 34.45, 'Juma', 'John', 520)
('bcde', 780, 34.45, 'Juma', 'John', 520, 'bcde', 780, 34.45, 'Juma', 'John', 520)
('mangoes', 'bananas', 'oranges', 'strawberry', 'melons', 'bcde', 780, 34.45, 'Juma', 'John', 520)
>>>
```

For help on tuples, just type `help ('tuple')` or `help ("tuple")` followed by enter] in python shell. This will provide a list of all forms of classes that can be applied on lists **Python Dictionary**

These are hash table types values. They are associative arrays or a hash which are similarly found in Perl and consists of key value pairs.

Examples of pairs include the following values:

```
variable test.py - C:\Python27\variable test.py
File Edit Format Run Options Windows Help

dict = {'fruits': 'bananas', 'name': 'juma', 'code': 6789, 'dept' : 'marketing'}
print dict           # Prints the entire string
print dict.keys ()   # prints all keys
print dict.values () # prints all values
```

The output is expected to look like this when this program is run:

```
>>>
{'dept': 'marketing', 'code': 6789, 'name': 'juma', 'fruits': 'bananas'}
['dept', 'code', 'name', 'fruits']
['marketing', 6789, 'juma', 'bananas']
>>>
```

Data variables apply in all data codes development and understanding how they work and operate is very essential in learning program.

Connection between Lists and Dictionaries

It is possible to work with both lists and dictionaries and changing between lists and dictionaries and vice versa. This is a common application for most of the programmer and sometimes they have to interchange the data types especially when working with databases which will be learned later in this book (Chapter 12). This is a strength in Python that can never be found in any other language since it has this functionality, that is why Python is referred to as an interactive language.

Let us consider the following example of a dictionary: D is the Dictionary and L is the list.

```
D = {"list": "Listed", "dictionaries": "dictionary", "functional": "Functionality"} |
```

we could turn this into a list with two-tuples:

```
L = [("list", "Listed"), ("dictionaries", "dictionary"), ("functional", "Functionality")]
```

Note that both the D and L have the same information. It can also be described that their entropy (organization and arrangement of each word) is also the same. But, it should be noted that sometimes the information may be harder to be retrieved when as a list compared to a dictionary.

When we want to find values in Key's, we will have to first go through the tuples of the lists compare the components of the list with the keys. This can be well performed when used as a dictionary.

Lists from Dictionaries

It is very easy to develop a list from a dictionary. This is achieved by using the function (items (), keys (), and values ()).

Just the method keys () will be able to create a list. This will plainly consist of keys of the dictionary. In addition, values () will be able to make a list of values. Moreover, items () can also be used to create of list of items made of 2 tuples with keys and values (keys, values)-pairs:

```
File Edit Format Run Options Windows Help
>>> w = {"house": "Haus", "cat": "", "red": "rot"}
>>> items_view = w.items()
>>> items = list(items_view)
>>> items
[('house', 'Haus'), ('cat', ''), ('red', 'rot')]
>>>
>>> keys_view = w.keys()
>>> keys = list(keys_view)
>>> keys
['house', 'cat', 'red']
>>>
>>> values_view = w.values()
>>> values = list(values_view)
>>> values
['Haus', '', 'rot']
>>> values_view
dict_values(['Haus', '', 'rot'])
>>> items_view
dict_items([('house', 'Haus'), ('cat', ''), ('red', 'rot')])
>>> keys_view
dict_keys(['house', 'cat', 'red'])
>>>
```

However, when a method `items()` is used in a dictionary, we may not be able to get a list back. But, we can be able to use the item views, the item views will be converted to a list when used with a list function. No information will be lost by converting a dictionary into a list.

It is also possible to return to the original dictionary from the `items()`, even when formed as a list of tuples, they have the same entropy. However, the effectiveness of the same approaches is similar.

Therefore, dictionary also serves as an efficient method to access, replace, change and vary elements of the dictionary. However, in lists, these have to be coded by a programmer.

Turn Lists into Dictionaries

The lists can also be turned into Python; if these lists satisfy certain conditions. Let us consider two lists containing dishes and other countries:

```
File Edit Format Run Options Windows Help
>>> dishes = ["pizza", "doughnuts", "bread", "hamburger"]
>>> countries = ["China", "Sweden", "Italy", "USA"]
```

A dictionary can be created that will be able to assign a dish to a country. This can be done using a function `zip()`. This will be able to join the lists like a zip. This will be able to iterate over a list. This suggests that we have to wrap a `list()` function on the `zip` so as to call to get a list:

```
File Edit Format Run Options Windows Help
>>> dishes = ["pizza", "doughnuts", "bread", "hamburger"]
>>> countries = ["China", "Sweden", "Italy", "USA"]

>>> country_specialities = list(zip(countries, dishes))
>>> print(country_specialities)
[('China', 'pizza'), ('Sweden', 'doughnuts'), ('Italy', 'bread'), ('USA', 'hamburger')]
>>>
```

The country specific lists of dishes are now completely converted to a list. This is a list of two tuples which has keys and values which can also be automatically be convertible to a dictionary by using the function `dict()`.

```
>>> country_specialities_dict = dict(country_specialities)
>>> print(country_specialities_dict)
{'Italy': 'bread', 'China': 'pizza', 'USA': 'hamburger', 'Sweden': 'doughnuts'}
>>>
```

The only mystery will be to understand the use of the function `zip()` when one list is long that the other.

This is not complicated either. The program will be able to ignore any extra lists that may not be paired in any way.

```
>>> dishes = ["pizza", "bread", "doughnuts", "hamburger"]
>>> countries = ["Italy", "China", "Sweden", "USA", "Switzerland"]
>>> country_specialities = list(zip(countries, dishes))
>>> country_specialities_dict = dict(country_specialities)
>>> print(country_specialities_dict)
{'Sweden': 'China', 'Italy': 'pizza', 'USA': 'hamburger', 'Spain': 'bread'}
>>>
```

Here, Switzerland will not be paired in this case and will be ignored in the output as presented.

All in one

Normally, we recommend not implementing too many steps in one programming

expression, though it looks more impressive and the code is more compact. Using "talking" variable names in intermediate steps can enhance legibility. Though it might be alluring to create our previous dictionary just in one go:

```
>>> country_specialities_dict = dict(list(zip(["pizza", "doughnuts", "bread", "hamburger"]
      ["Italy", "Germany", "Spain", "USA", "Switzerland"])))
>>> print(country_specialities_dict)
{'doughnuts': 'Sweden', 'hamburger': 'USA', 'bread': 'Italy', 'pizza': 'China'}
```

This can also be written in this other way:

```
dishes = ["pizza", "doughnuts", "bread", "hamburger"]
countries = ["Italy", "China", "Sweden", "USA"]
country_specialities_zip = zip(dishes, countries)
print(list(country_specialities_zip))
country_specialities_list = list(country_specialities_zip)
country_specialities_dict = dict(country_specialities_list)
print(country_specialities_dict)
```

Summary

The data variables come in the five ways: They can be in form of numbers, lists, tuples, strings and finally dictionaries. Each of them is very unique and working with python at every single process requires a basic understanding of their functions and classes.

In python, to better understand how this can be applied. You can seek help from the python by typing help ('object') or help ("object") followed by enter. This will provide you with a solution on how any of the variables can be applied.

Chapter 9: Basic Operators in Python for Calculations

Operators are the functions which can be used in the manipulation of the values of the operands. For instance, let us consider the expression $5 + 15 = 20$. In this example: 5 and 15 are referred to as operands, while the plus sign (+) is referred to as the operator.

Operators

Python language supports the following types of operators.

- Arithmetic operators
- Comparison operators for establishing relations between values
- Assigning operators for equating values to a name
- Logic operator for determining the logic between values on either side of the operators.
- Membership operators
- Operators for influencing the identity of objects/values

Here is a list of the various forms of operators and examples on how they can be applied in a real python algorithmic development scenario in programming. They have been looked at one by one.

Python Arithmetic Operators

Let us assume that a variable named (a) has a value of 5 and variable (b) holds a value of 15, therefore:

Operator	Description	Example
+	Sums up the values of the values on either side of the operator.	$a + b = 20$
-	Gives the difference of the two values on either side of the operator	$a - b = -10$
*	Gives you a product of values on either sides of the	$a * b = 75$

	operator	
/	Provides a division of the two values	$b / a = 3$
%	Divides left by right hand values of the operand and gives the value of the remainder	$b \% a = 0$
**	Makes the second value the exponential power of the first value	$a^{**}b = 5 \text{ to the power } 15$
//	This is also referred to as the Floor Division In this division, the digits after the decimal point are removed from the result of the quotient.	

Operators for comparison of values Comparison operators make a comparison of the values to the left and right of the operand and helps define relationship between the values as set by the operand. They are also referred to as the relational operators. Use the values for (a) and (b) as illustrated above.

Operator	Description	Example
==	It sets as a condition for the values to the left and right if the operand to be true only when they are equal.	It is not true when $(a == b)$
!=	It sets as a condition for the two values to be true only when the values are not equal	Same as above
<>	It sets the condition to be true when the values of the operands are not equal.	It is true only when $(a <> b)$. The application is similar to the sign

		!=
>	This emphasizes on the value of the left operand. When it has a higher value than the right value, It accepts the condition as true.	It is not true if (a > b)
<	It also emphasizes on the value to the left of the operand. When the value is less than the right value, it accepts the set condition as true.	It is true when (a < b).
>=	This is applied only when the value of the left operand is higher or same as the value to the right operand. This recognizes the condition as true.	It is not true when (a >= b).
<=	If the value to the left is lower than or equals to right. This enables the condition as set to be true.	It is true when (a <= b).

Python for Assigning Operators

Operator	Description	Example
=	It assigns the value of the right hand side of the operand to the left hand side.	c = a + b
+= Add AND	This operand adds values on the right to left operand and equates sum to the left operand	c += a is equal to c = c + a
-=	It gets the difference between	c -= a is also c

Subtract AND	the right from left operand and gives the division to left operand	$= c - a$
$*=$ Multiply AND	Its action is same as above but then as a product of the right and left operand and makes the result equal to the left operand.	$c *= a$ is same as $c = c * a$
$/=$ Divide AND	Same as above but makes a division	$c /= a$ is same as $c = c / a$ and also $c = c / a$
$\%=$ Modulus AND	Same as above but makes floor division and gives a quotients and equates the quotient to the left operand	$c \%= a$ is equivalent to $c = c \% a$
$**=$ Exponent AND	Assigns an exponential (power) on operators and equates to the value of the left operand	$c **= a$ is same as $c = c ** a$
$//=$ Floor Division	It creates a division of operators and equates the value calculated as equal to the left value of the operand	$c //= a$ is expressed as $c = c // a$

Python Membership Operators

These operators evaluate membership of values in a sequence in various python data forms such as strings, lists, or tuples. There are only two forms of python membership operators:

Operator	Description	Example
is	It evaluates to be true when the variables on either side of the operators are similar or equates to the same object. But it is false	x is y .

	otherwise.	
not in	It becomes true when a variable in the sequence is not found and otherwise, becomes false.	x not in y

Python Identity Operators

This only serves to make a comparison of two memory locations for two different objects. Similarly, just like membership operators, there are only two membership operators as explained below:

Operator	Description	Example
is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.	x is y
is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.	x is not y

Python Operators ordered list

The table below ranks the operators in terms of their precedence in descending order.

Operator	Description
**	Exponential (raises one value to the power of the other)
~	Complement,
+	plus

-	unary minus
*	Multiply
/	divide
%	modulo
//	floor division
+	Addition
-	Subtractions
>>	Right shift bitwise
<<	left bitwise
&	Bitwise 'AND'
^	exclusive 'OR' and regular 'OR' bitwise
<= >	Comparison operators
< >	comparison
=	comparison
<> == !=	Equality operators
= %= /= - = += *= **=	Assigning values operators
Is, is not	Identity operators
in not in	Creating membership operators

not or and	Creating logical operators
------------	----------------------------

Chapter 10: Opening and Closing Files

Since the start of this chapter, we have been dealing with basics of trying to read and write inputs in python. This time, I would like us to explore the use of writing and reading real files outside python. Yes, python is a very practical program and one can basically be able to do anything as long as he knows what exactly needs to be done in Python.

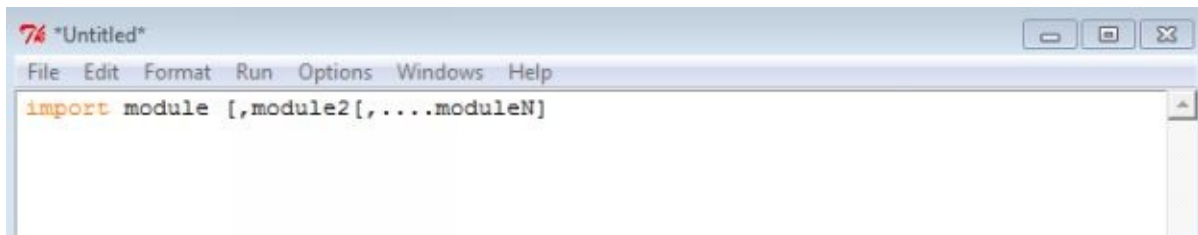
Python has simple functions and methods that enable one to be able to manipulate files. This is achieved by the use of the file objects in python.

But before, we get to it:

It is first important to first describe some few other functions such as import. How do you bring in the file from a different directory to Python? This is how it is achieved in Python by using the import statement: **Import statement.**

Any python source file can be used a module through the execution of the import statement from some other source file in python.

The import function contains the following module.



The function import has to be placed at the top of the script.

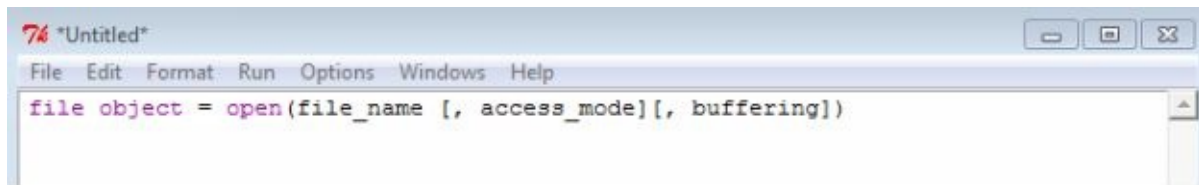
The module would then be loaded only once however, despite the number of times the import function is used in the script. This allows the program to prevent multiple imports of the file to be executed in the program. You will note that when importing a file, for instance, a XML file converted as a CSV file, first, it has to be imported before any execution code can be defined. This will define what the program will be able to do on the file imported into python.

Python has in built basic functions and methods necessary to manipulate files by default. The file object can be used to do the execution and manipulation of the

files you need.

The open Function A file has to first be imported before it can be read and write in Python. Even before you are able to read and write a file in Python, you will need to first open it up to be read. This is achieved using the inbuilt open () function in Python. This function allows the creation of an object as a file which will then enable the execution of other support functions associated with the file.

The open function comes as below:



Here are parameter details:

- **file_name:** is used as string that contains the file name of the file that is to be opened in python.
- **access_mode:** Determines the mode by which the file is required to be opened. It allows for reading, writing and appending among other modes available in Python as will be demonstrated in the table below. However, it always recommended applying the default access mode as read (r).
- **Buffering:** This a value which can either be zero or any other number, often 1 is used. When the buffer value is fixed as zero (0). No buffering is applied for the files being read. However, when the buffer value is used as (1), buffering is applied at the time the file is being accessed. Always, as long as the buffer value is an integer and a value more than one, the buffering process is initiated in accordance with the suggested buffer size. But if the buffer value is negative, the buffer size adopts a default characteristic.

Modes of opening file is as described in the table below:

Modes	Description
r	A file is opened but it can only be read. The reading starts from the start of the file. This is used as a default mode setting.

rb	This allows the opening of the file but for reading only in the binary mode. The reading starts from the start of the file. This is used as a default mode setting.
r+	This mode allows the opening of a file for reading and writing at the same time. The reading starts from the start of the file. This is used as a default mode setting.
rb+	This mode allows the opening of a file for reading and writing at the same time but in the binary mode. The reading starts from the start of the file. This is used as a default mode setting.
w	Allows a file to be opened but can only be written. It allows the existing file to be overwritten. However, a new file will be created when none of the file described exists.
wb	Allows a file to be opened but can only be written in the binary format. It allows the existing file to be overwritten. However, a new file will be created when none of the file described exists
w+	Allows a file to be opened and can be read and written at the same time. It allows the existing file to be overwritten. However, a new file will be created when none of the file described exists
wb+	Allows a file to be opened and can be read and written in the binary format. It allows the existing file to be overwritten. However, a new file will be created when none of the file described exists
a	This allows a file to be opened and can only allow appending of new information. Hence, the file pointer will be at the end of the file if it exists and it is present in the append mode. A new file will also be created if the file mentioned does not exist.
ab	This allows a file to be opened and can only allow appending of new information only in the binary

	format. Hence, the file pointer will be at the end of the file if it exists and it is present in the append mode. A new file will also be created if the file mentioned does not exist.
a+	It executes the opening of a file and allows appending and reading of the file. The file pointer is at the end of the file if the file exists. Hence, the file pointer will be at the end of the file if it exists and it is present in append and read mode. A new file will also be created if the file mentioned does not exist.
ab+	It executes the opening of a file and allows appending and reading of the file in the binary format. The file pointer is at the end of the file if the file exists. Hence, the file pointer will be at the end of the file if it exists and it is present in append and read mode in binary format. A new file will also be created if the file mentioned does not exist.

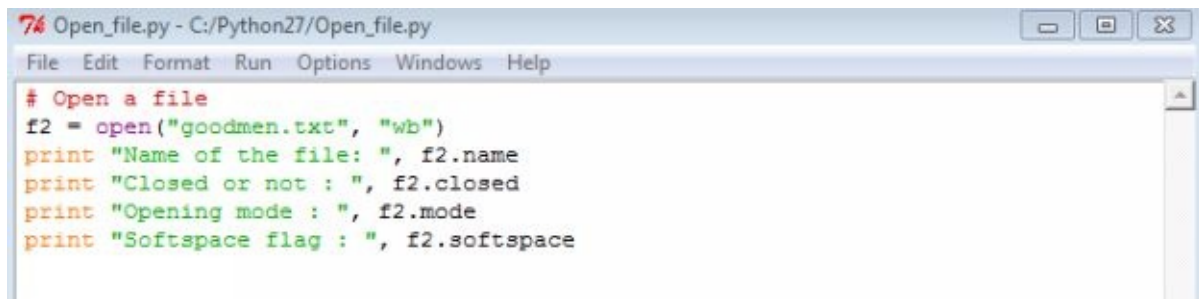
How to use the *file* Object Attributes Immediately a file has been opened, then, it becomes a file object for which much other information can be derived in python relating to the information in the file.

The following attributes can be executed to the file object:

Attribute	Description
file.closed	This allows Python to returns true when the file is closed, otherwise, it is considered as false.
file.mode	This returns the mode of access for which the file was opened.
file.name	It returns the file name opened in Python.
file.softspace	This will returns false when space is required, otherwise, it will be true.

An example of the program to be executed includes

Example



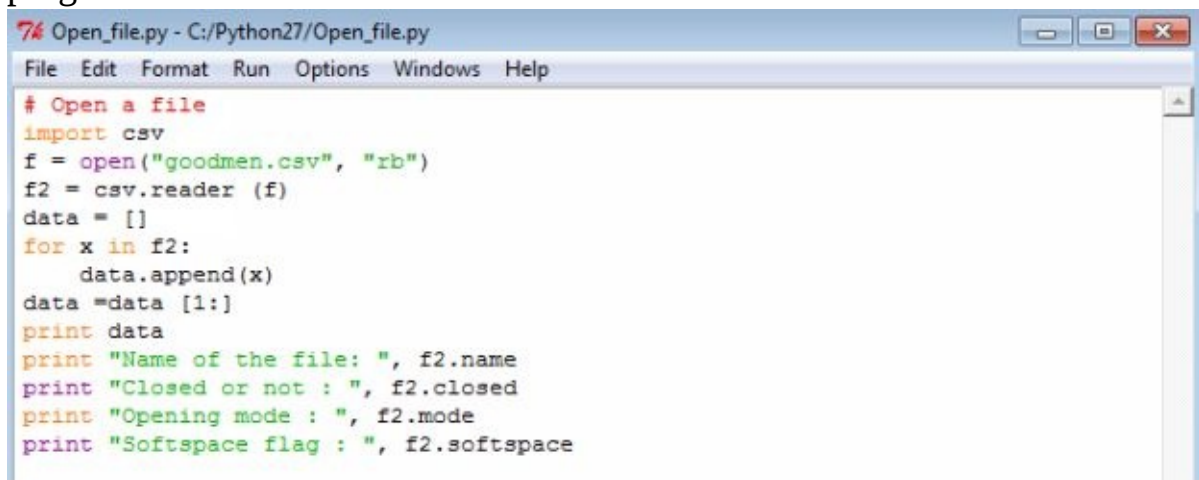
```
# Open a file
f2 = open("goodmen.txt", "wb")
print "Name of the file: ", f2.name
print "Closed or not : ", f2.closed
print "Opening mode : ", f2.mode
print "Softspace flag : ", f2.softspace
```

The output will be as seen in this screenshot:



```
>>>
Name of the file: goodmen.txt
Closed or not : False
Opening mode : wb
Softspace flag : 0
>>>
```

In most cases, when working with a real/actual file such as an excel file. Then, the import function can be used in this case. Assuming you have a file saved as a CSV file name “goodmen.csv”. You can open it and use it in many other ways. Moreover, you can also check out the name, confirmed if the file is closed or not, confirm the mode and softspace flag as illustrated above. The following program can be used to access it:



```
# Open a file
import csv
f = open("goodmen.csv", "rb")
f2 = csv.reader(f)
data = []
for x in f2:
    data.append(x)
data =data [1:]
print data
print "Name of the file: ", f2.name
print "Closed or not : ", f2.closed
print "Opening mode : ", f2.mode
print "Softspace flag : ", f2.softspace
```

Use any other file with a diff name and see what happens with the file. Just change the name goodmen.csv with your new file name in the csv format.

Closing of a file using the `close()` attribute in Python This allows Python to close a *file* object and assumes any information that had not been written and closes the file object. This ensures that the program does not execute any other function after that had been done.

However, python is also able to close a file automatically when a different object is assigned to the file. But, it is a good practice to always close the file using the `close ()` function.

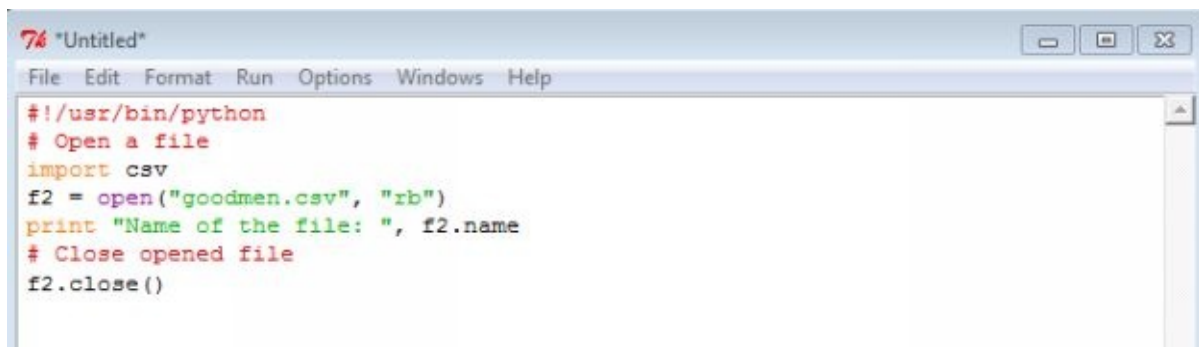
This is implemented using the following syntax:

```
fileObject.close();
```

For example:

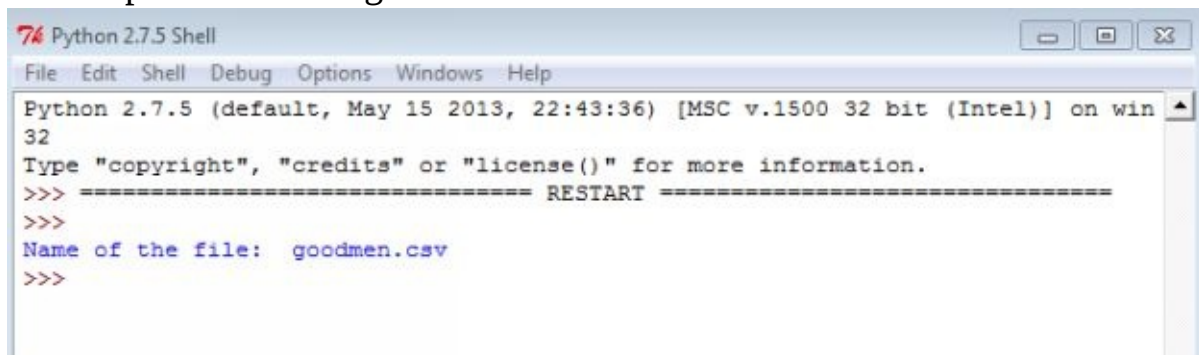
The previous file opened was `f2`.

The file can therefore be closed using the attribute:

A screenshot of a Python IDE window titled "Untitled". The menu bar includes File, Edit, Format, Run, Options, Windows, and Help. The code in the editor is as follows:

```
#!/usr/bin/python
# Open a file
import csv
f2 = open("goodmen.csv", "rb")
print "Name of the file: ", f2.name
# Close opened file
f2.close()
```

The output after running this file will be as shown in the screen shot below:

A screenshot of a Python 2.7.5 Shell window. The title bar says "Python 2.7.5 Shell". The menu bar includes File, Edit, Shell, Debug, Options, Windows, and Help. The output text is as follows:

```
Python 2.7.5 (default, May 15 2013, 22:43:36) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
Name of the file:  goodmen.csv
>>>
```

Summary

It is important to know that before you are able to read a file in Python. It must

be opened first, using the open function as described in this chapter.

Next, you will be able to execute and find out the attributes of the file using the various attributes as mode, filename, space and many more attributes.

Finally, always ensure that the program is closed using the file_name object.close() function in Python. It allows you to run another program and makes the python process quick in its execution of programs even in limited memory capacity.

Chapter 11: Reading and Writing Files in Python

In the last chapter, we learned how to be able to open and close a file.

A file that is open can allow reading and writing depending with the mode selected for execution in Python. Using the file objects allows one to access methods that will enable one to manipulate data in various ways. This makes life for people easier. In this chapter therefore, we would like to know how we can be able to read file using the *read()* and *write()* methods for reading and writing files.

Using *write()* Method This attribute allows one to add anything in the form a string to an open file. Note that Python strings need to have binary data and may not just accept any form of text to be used in this case.

In addition, using the *write()* technique many not introduce a newline character ('\n') at the end of the string.

The syntax for the *write()* can be expressed as below

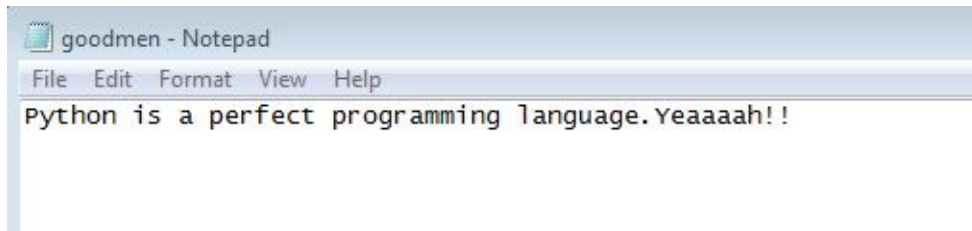
```
fileObject.write(string);
```

Here, is an example of how this can be applied in the real Python code:

```
#!/usr/bin/python
# Open a file
f2 = open("goodmen.txt", "wb")
f2.write( "Python is a perfect programming language.\nYeaaaah!!\n");
# Close opened file
f2.close()
```

Since, this file does not exist in python, it will create a new file called “goodmen.txt” and then it will close after execution of the filename object.close().

When the file is opened, it will contain the following text in it. Open it in Notepad.



How to use the `read()` attribute in Python This method allows strings to be read from an open file. The python strings may contain binary data besides the use of the data in text.

The syntax for reading files is expressed as follows in Python:

```
fileObject.read([count]);
```

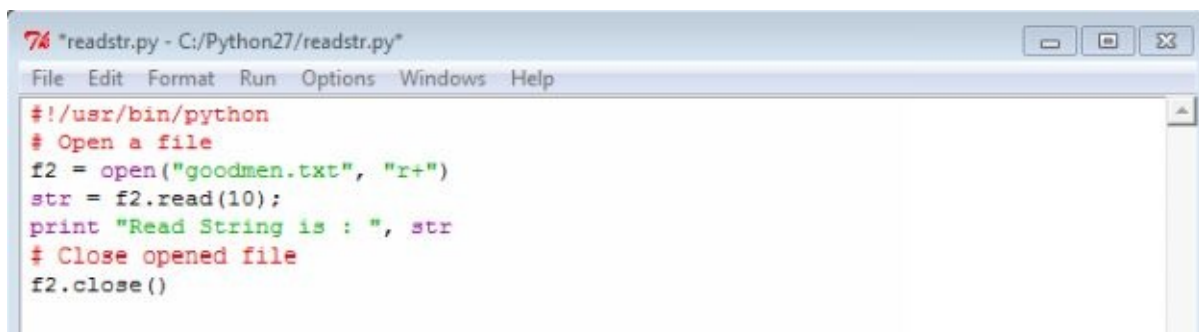
In python, the file object is the assigned name in python at the time the file is being opened. For instance, *goodmen* files above were assigned an object value `f2`.

In this case, this allows the file opened to be read from the start of the file and when the count is absent, then, it will read the entire document to the end of the file. Hence, it is important to specify how much information may be needed when trying to get specific information in Python.

Note: You direct Python to do everything that you would like it to do.

For example: Using the previous file in this chapter “*goodmen.txt*”

We can read the above file in python and see what results it generates in Python.



The output will be expressed as follows:

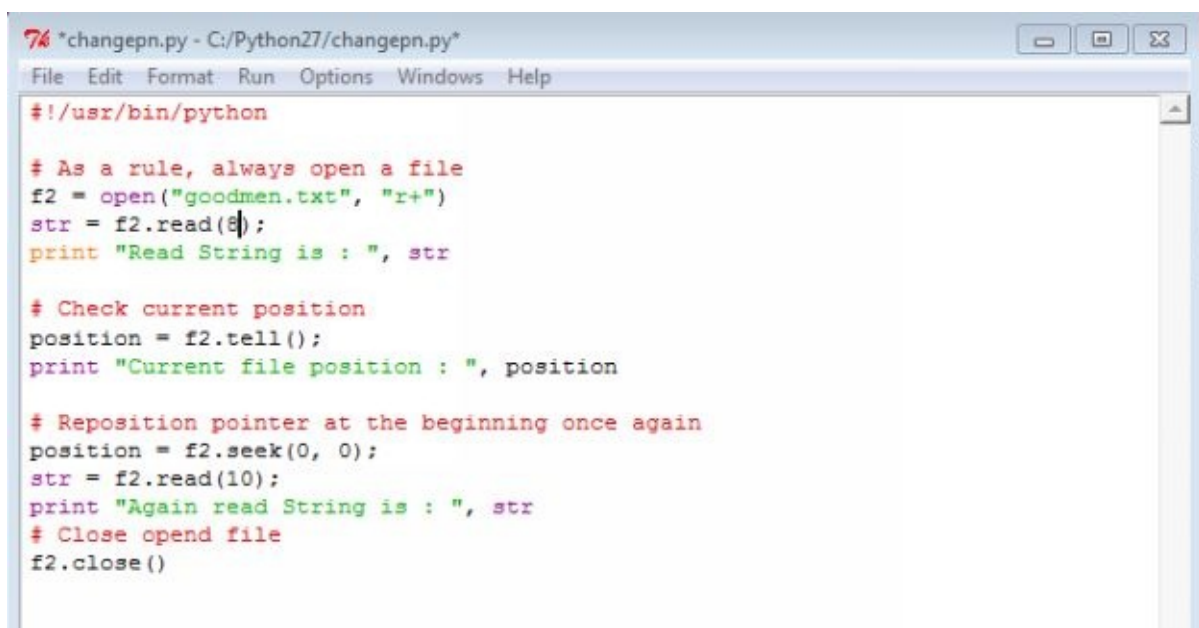
```
>>> ===== RESTART =====
>>>
Read String is : Python is
>>>
```

Getting the positions of files This is the method that can be used in finding the position of a text within a file. This uses the *tell()* attribute in python. This allows Python to do the next read or write from the position indicated from the beginning of the file created.

However, we can also be able to change the current file position of a file using the *seek(offset[, from])* method. The argument *offset*; suggests the number of bytes to be moved while the other argument, *from* shows the specific position where the bytes will be moved from the current position.

When the value from is set as zero (0), This suggests that the file has to be read from the beginning of the file as the main reference point while a value of one (1) is used to refer t the current position. A value of 2 is applied to indicate the end of the file as the main reference point.

Using an example below: Let us take a file *foo.txt*, which we created above.



```
*changepn.py - C:/Python27/changepn.py*
File Edit Format Run Options Windows Help

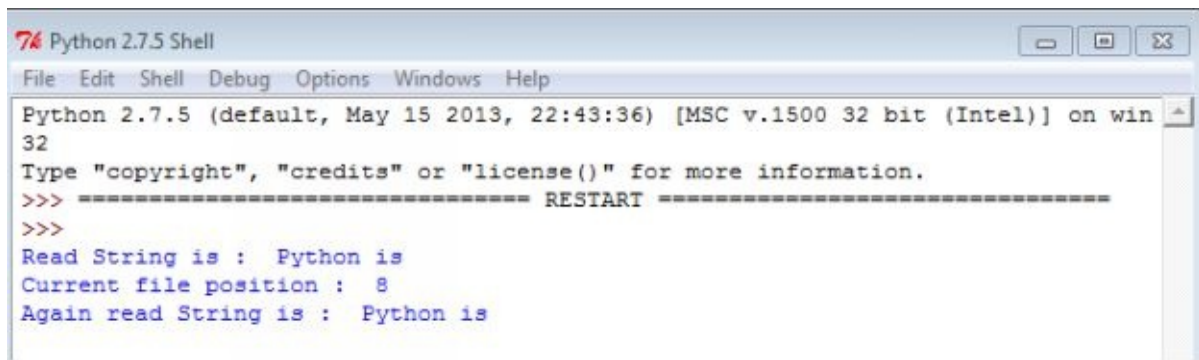
#!/usr/bin/python

# As a rule, always open a file
f2 = open("goodmen.txt", "r+")
str = f2.read(4);
print "Read String is : ", str

# Check current position
position = f2.tell();
print "Current file position : ", position

# Reposition pointer at the beginning once again
position = f2.seek(0, 0);
str = f2.read(10);
print "Again read String is : ", str
# Close opend file
f2.close()
```

The output will be as represented in the screen shot below:



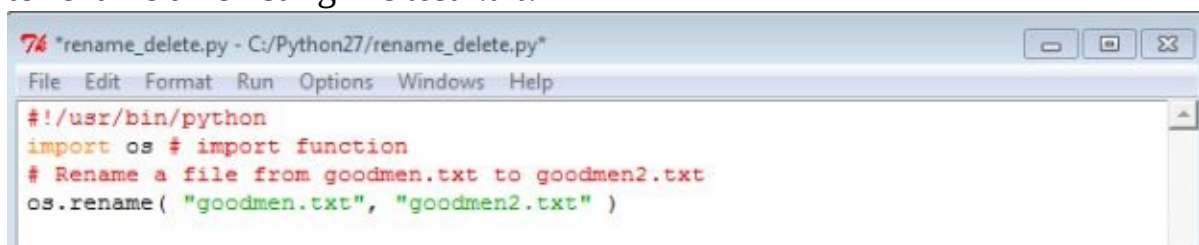
```
Python 2.7.5 Shell
File Edit Shell Debug Options Windows Help
Python 2.7.5 (default, May 15 2013, 22:43:36) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
Read String is : Python is
Current file position : 8
Again read String is : Python is
```

How to Rename and Delete Files Python also allows one to be able to process files in the python os. This includes processes such as being able to rename file or delete files without the need to manually delete the files. This can best be applied first by the use of the import statement and then you will be able to call any functions that can be performed.

Using the `rename()` syntax The `rename()` attribute allows one to use two arguments. First, you will be required to use the current filename and then followed by the new filename.

```
>>>
os.rename(current_file_name, new_file_name)
```

A good example of the application is as shown below: Following is the example to rename an existing file *test1.txt*:



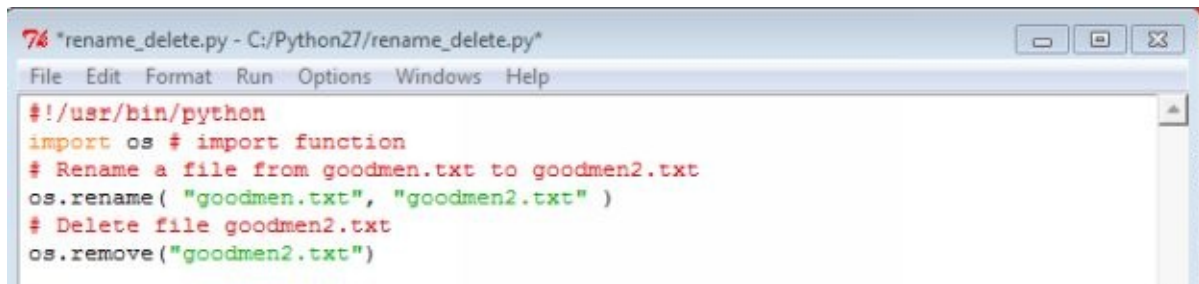
```
*rename_delete.py - C:/Python27/rename_delete.py*
File Edit Format Run Options Windows Help
#!/usr/bin/python
import os # import function
# Rename a file from goodmen.txt to goodmen2.txt
os.rename( "goodmen.txt", "goodmen2.txt" )
```

Deleting files using the `remove()` Method This is applied when one has to delete a file. This requires that you will have to provide a name of the file that has to be deleted in the argument.

```
os.remove(file_name)
```

The main syntax for the process is:

The following is an example on how to delete an existing file created as *goodmen2.txt*.



```
7% *rename_delete.py - C:/Python27/rename_delete.py*
File Edit Format Run Options Windows Help
#!/usr/bin/python
import os # import function
# Rename a file from goodmen.txt to goodmen2.txt
os.rename( "goodmen.txt", "goodmen2.txt" )
# Delete file goodmen2.txt
os.remove( "goodmen2.txt" )
```

Working with Directories in Python All files are contained within various directories, and Python has no problem handling these too. The **OS** module has several methods that help you create, remove, and change directories.

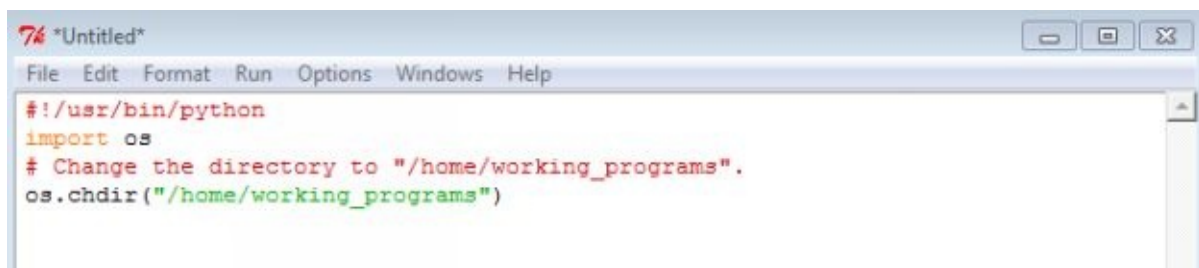
The *mkdir()* Method You can make a directory using the attribute *mkdir()* method inbuilt with the OS in Python. This will be able to create a new directory in the current directory. This is done by introducing an argument that will contain an argument which will have the name of the directory that needs to be created. The syntax to be used is: `os.mkdir("newdir")` The following example can be used for creating a new directory (new_programs) in the current directory.



```
7% *Untitled*
File Edit Format Run Options Windows Help
#!/usr/bin/python
import os
# Create a directory "new_programs"
os.mkdir( "new_programs" )
```

Changing directory Method To change the directory, use the attribute `chdir()` to change the details of the current directory. This is an argument that makes the new current directory.

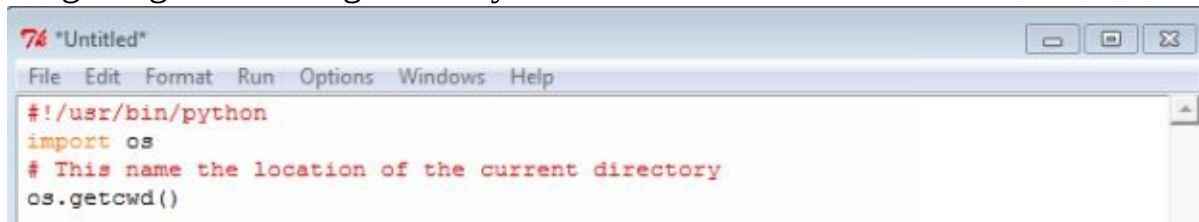
It has the syntax as follows: `os.chdir("newdir")` As a good example for this syntax, let's change to the current new directory "*homeworking_programs*".



```
7% *Untitled*
File Edit Format Run Options Windows Help
#!/usr/bin/python
import os
# Change the directory to "/home/working_programs".
os.chdir( "/home/working_programs" )
```

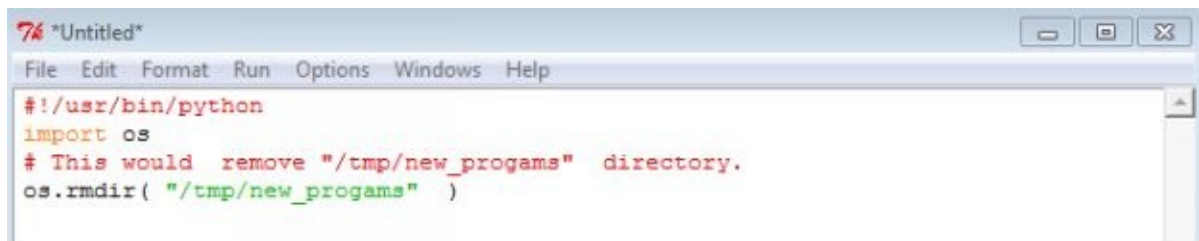
How to get the current working directory This is achieved by the use of the

`getcwd()` attribute in python. It can be able to display the current directory that is working. The syntax used is expressed as: `os.getcwd()` A perfect working code for getting the working directory is as shown in this screen shot

A screenshot of a Python IDE window titled "Untitled". The menu bar includes File, Edit, Format, Run, Options, Windows, and Help. The code editor contains the following text:

```
#!/usr/bin/python
import os
# This name the location of the current directory
os.getcwd()
```

How to delete a directory This is the technique which one can apply when they want to remove a directory. The attribute in function for this process is the `rmdir()` This method enables one to delete completely the directory. This is initiated as an argument in Python. However, note that all the contents that had been previously saved in the directory will need to be removed first before it can be deleted. The syntax for the argument is set as: `os.rmdir('dirname')` To remove “tmp/new_programs” as a directory. The full name as it appears in the directory is provided. Hence, it will search for the name in the current directory.

A screenshot of a Python IDE window titled "Untitled". The menu bar includes File, Edit, Format, Run, Options, Windows, and Help. The code editor contains the following text:

```
#!/usr/bin/python
import os
# This would remove "/tmp/new_programs" directory.
os.rmdir( "/tmp/new_programs" )
```

Manipulation of files and directories There are two special methods which allow one to be able to handle and manipulate different files and directories. These are applicable in both windows, Unix OS.

They include the following methods:

- File Object techniques: The *file* object allows a function to change files.
- OS Object techniques: enables the processing of both files and directories.

Summary In this chapter, we have just learned that we can be able to delete a directory, create a new directory, change a directory, get a directory name and many other functions. This is an important process in Python. It makes life easier. You will be able to create a file in Python, create a new directory, save it

there and even confirm if the file has been saved, and close the file as usual.

Practice Exercise

1. A new file named “1000_keywords.csv” has to be opened, read as a binary file. Write a Python code that will execute this process.
2. The file name also has to be changed to “Smart_words.csv” write a program that will execute the process and stored in the same directory.
3. John is working and is not sure in which directory he has been saving his work. Write him a two line program that will be able to identify the directory he has been saving his work.
4. John has two long lists of students that he needs to merge them together. How can he be able to create a single list of names in Python? How can he also be able to save the list in his current directory named “students”? To avoid confusion, he will be required to delete the other original files in the current directory and save the same files in a different directory named “Old records”. Develop a single python that will be able to accomplish this process.
5. Mike is a tutor for a school. The final exams were done and he has finished marking. However, he has to calculate the overall marks by adding the mean of the cat marks for cat 1 and cat 2 which account for 30% and add the marks to the final exams mark which is out of 70%. He has only a day to go before the deadline. How can he be able to do this automatically without much hustle in Python and accomplish the task in less than 10 minutes for 60 students?

Chapter 12: Managing Databases

The use of databases back to before the 1970's. SQL is one of the few databases that have a wide area of application and more especially with respect to python programming. SQL has a well-structured language with query which is based on creating relationships as a model for sharing in large data banks.

In fact, SQL is often pronounced as sequel when read directly in English. Currently, it is now one of the best standards in the American national standards Institutes (ANSI) after 1987. This is because most of the people prefer using mSQL, postgres SQL, MySQL and other forms of SQL.

What is a database?

A database is defined as a collection of organized data. The organization of the data allows processing of the data either by way of deleting files, rearranging, adding new information, replacing content and many other functions.

A database is also the data itself that has been stored for databank for special reasons. It can also be associated with the database management system. The database management system is the software that can be used in the processing of the data by the user.

The user of the database may not be required to be human but programs and applications too.

Python as introduced at the beginning of this book was expressed as an interactive language. Python has the ability to interact with a database such as SQL. It therefore has the potential of interacting as a user for the database.

It is possible to use SQLite and other SQL forms from Python as a program. It is a standard that the database interface for Python is DB-API. This is used by Python interfaces when interacting with Python. While it is a common interface, it can also be used in relational databases using codes developed in Python for communication with databases. The language used is the same and regardless of the type of the database that will be used and the module that has to be applied.

Already, I have addressed a few examples of databases; SQL and MySQL. Here, we shall look at each one of them individually.

SQLite

This operates a simple relational database system. It has the ability to save the data in the simple database systems in the simple regular data files in simple locations of a computer such as the computer memory in the RAM. It is very compatible with simple applications such as Mozillar-Firefox, Symbian OS, android and many other OS programs.

It is also very fast since it uses simple files and has the capability to operate for large databases.

The operation of the SQLite relies on importing of the module sqlite3. Then, a connection is required with the object. The connection object will represent the database. The argument of the connection is the name of the database such as “companys.db”. It functions both as a name of the file, where the data will be stored and as the name of the database itself. Presence of the file name will have to be established and if the name exists, it will be opened.

Note: the database has to be an SQLite, if the name exists, then it has to be opened.

Let us create a database using the following code. This will be able to open a database company. The file in this case may not need to exist in this case.

```
import sqlite3
connection = sqlite3.connect("company.db")
```

Using this code, you will be able to create a database that will have the name ‘Company’. This is a command that initiate and creates a database company in an SQL server.

When you try to use the command:

```
"sqlite3.connect ('company.db')" again
```

This will open a previous database with a similar name that had been created.

What you developed is an empty database and what has to be done next is to create tables in the format needed so as to be able to insert and retrieve information in the way that is needed.

An SQL code for developing table "staff" in the database "company" looks like this:

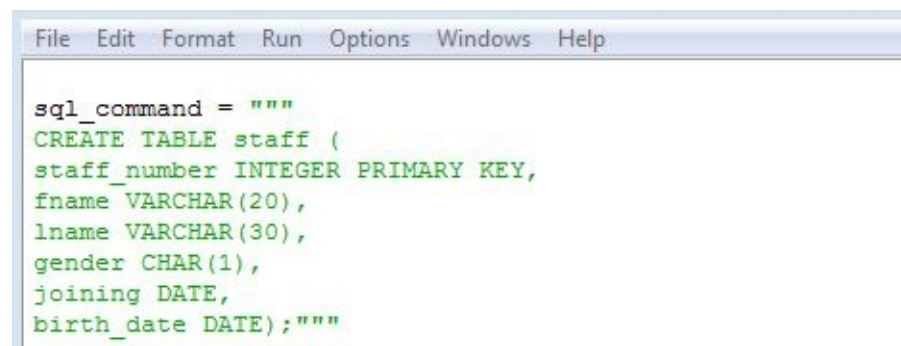
```
CREATE TABLE staff (  
    staff_number INT NOT NULL AUTO_INCREMENT,  
    fname VARCHAR(20),  
    lname VARCHAR(30),  
    gender CHAR(1),  
    joining DATE,  
    birth_date DATE,  
    PRIMARY KEY (staff_number) );
```

This is how it can be done in an SQL in the command shell. This can be done in Python directly. A command can also be sent to an SQL or SQLite. This is only possible with the use of the SQL cursor. The relevance of a cursor is to enable one to access, read, write and use information in a database. It can also be used I fetching for information from a database with a precise output.

In most of the Python programs, it is used for performing most of the commands.

This can be done by calling the cursor method as a way of establishing the connections. Any number of cursors can be created. They can also be used to go over the records and information which are the results from a database.

Therefore, a complete Python code for developing a power company would look like this: Remember the staff table will also be created in this database:

A screenshot of a Python IDE window with a menu bar (File, Edit, Format, Run, Options, Windows, Help) and a code editor. The code editor contains the following SQL command:

```
sql_command = """  
CREATE TABLE staff (  
    staff_number INTEGER PRIMARY KEY,  
    fname VARCHAR(20),  
    lname VARCHAR(30),  
    gender CHAR(1),  
    joining DATE,  
    birth_date DATE);"""
```

In the syntax above, it is clear that the AUTOINCREMENT part has been removed. Note that in SQL, The “INTEGER PRIMARY KEY” also automatically auto increase the values in the field required in the table in SQLite3.

This can also be expressed in the sense that when a column of a table is declared

to be an INTEGER PRIMARY KEY. Every time whenever a NULL is used as an input for the column, The NULL value will be automatically get converted to an integer with a value larger/higher than the previous value by one in the column. When the table is empty, it will automatically give a value one for the existing number. For instance, when a largest existing value for a column is 9234576898344, any new value that will be added will be one value higher than it, hence the value in SQLite will be an unused value selected randomly.

To this far, we have been able to create a database and a table inside it with several columns. But currently, no data has been created on it.

To create the data on the table, we will have to be able to populate the table using a simple Python command “INSERT” command to SQLite. Similarly, we will need to execute the code as a method of populating the table.

Use the below working example to know how making a database can be easy and interesting using SQLite and Python as the best interactive language.

Try the program below:

To run the program you will either have to remove the file company.db or uncomment the "DROP TABLE" line in the SQL command:

```
File Edit Format Run Options Windows Help
import sqlite3
connection = sqlite3.connect("company.db")

cursor = connection.cursor()

# delete
cursor.execute()

sql_command = """
CREATE TABLE staff (
    staff_number INTEGER PRIMARY KEY,
    fname VARCHAR(20),
    lname VARCHAR(30),
    gender CHAR(1),
    joining DATE,
    birth_date DATE);"""

cursor.execute(sql_command)

sql_command = """INSERT INTO employee (staff_number, fname, lname, gender, birth_date)
VALUES (NULL, "William", "Shakespeare", "m", "1961-10-25");"""
cursor.execute(sql_command)

sql_command = """INSERT INTO employee (staff_number, fname, lname, gender, birth_date)
VALUES (NULL, "Frank", "Schiller", "m", "1955-08-17");"""
cursor.execute(sql_command)

# always save the changes made on the database:
connection.commit()

connection.close()
```

To be able to insert the data on a database in python, you will not be able to literary insert the data into a table. But will be required to have a many data in the forms of data types discussed in the beginning of this book. It can be in the form of dictionaries or a list which can be used as the input of the insert statement.

For Example, using a list of persons, it is possible to insert the names in an existing database company.db and a table staff. This can be achieved using the INSERT statement in Python.


```
File Edit Format Run Options Windows Help
import sqlite3
connection = sqlite3.connect("company.db")

cursor = connection.cursor()

staff_data = [ ("Wilberforce", "Washington", "m", "1967-09-22"),
                ("Benard", "Maxon", "M", "1974-08-19"),
                ("Jane", "Fischer", "F", "1979-05-16") ]

for p in staff_data:
    format_str = """INSERT INTO employee (staff_number, fname, lname, gender, birth_date)
    VALUES (NULL, "{first}", "{last}", "{gender}", "{birthdate}");"""

    sql_command = format_str.format(first=p[0], last=p[1], gender=p[2], birthdate = p[3])
    cursor.execute(sql_command)
```

It is possible query the staff table in Python. Follow the following code to know how this can be achieved very easily in Python. This enables you to be able to retrieve any information from the database and as well confirm and verify the information stored in the database.

```
File Edit Format Run Options Windows Help
import sqlite3
connection = sqlite3.connect("company.db")

cursor = connection.cursor()

cursor.execute("SELECT * FROM staff")
print("fetchall:")
result = cursor.fetchall()
for r in result:
    print(r)
cursor.execute("SELECT * FROM staff")
print("\nfetch one:")
res = cursor.fetchone()
print(res)
```

First, save this program as “query_SQL-company.py” and then run this program. This will give you the following products we get the following result, depending on the actual data:

```
*Untitled*
File Edit Format Run Options Windows Help

$ python3 sql_company_query.py
fetchall:
(1, 'Wilberforce', 'Washington' 'm', None '1967-09-22'),
(2, 'Benard', 'Maxon', 'M', None '1974-08-19'),
(3, 'Jane', 'Fischer', 'F', None '1979-05-16'),
(4, 'Esther', 'Wall', 'm', None, '1991-05-11')
(5, 'Jane', 'Thunder', 'f', None, '1989-03-14')

fetch one:
(1, 'Wilberforce', 'Washington' 'm', None '1967-09-22'),
```

MySQL

In order to use MySQLdb, it the module has to be installed on a computer on which you will be working on. However, on other programs such as Ubuntu and Debian, it is very easy.

Just type the following code in Python and wait for it to be installed:

```
*Untitled*
File Edit Format Run Options Windows Help

sudo apt-get install python-MySQLdb
```

Note that besides the import and the connects functions method and everything else as applied in SQLite. The working of MySQLdb operates like this:

- First, you have to import MySQLdb modul in Python
- To create a connection, open a connection to the SQL server
- Use both the sending and receiving commands
- Always close the connection in SQL

The import and connection functions in MySQLdb will look like the codes shown:

File Edit Format Run Options Windows Help

```
import MySQLdb
```

```
connection = MySQLdb.connect (host = "localhost",  
                               user = "testuser",  
                               passwd = "testpass",  
                               db = "company")
```

Practice Exercise

Create a database using SQLite3 and name it “students.db”. In the database, create a table in the database called std8_students and provide the insert and commit functions. Write a code that will be able to retrieve information from the same database and provide a list of the names of students, gender, age and code for the students.

Conclusion



I most sincerely thank you for downloading this book!

It is my belief that this book was able to help you learn how to interact with python and teach you how to use numbers and texts in various ways in Python.

The next step is for you to ensure that you practice how to use this book and information given to you on a daily basis to perfect on how to use numbers and texts in python. Note that python itself is very interactive will require very close attention.

Finally, if you believe that this book has helped you learn a Python. Please take time to share your thoughts on how this can be improved to make the book even better for others

Also, I encourage you to share your reviews through Amazon. This will be greatly appreciated!

Thank you again for reading this book.

I hope that you enjoy interacting with Python.

Bonus Free Python Programming Videos

When you subscribe via email, you will get free access to a toolbox of exclusive subscriber-only resources. All you have to do is enter your email address to the right to get instant access.

To get instant access to these incredible tools and resources, click the link below:

[=> Click here for the bonus content <=](#)