

What is Python

Python is an object-oriented, high level language, interpreted, dynamic and multipurpose programming language.

Python is *easy to learn* yet powerful and versatile scripting language which makes it attractive for Application Development.

Python's syntax and *dynamic typing* with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas.

Python supports *multiple programming pattern*, including object oriented programming, imperative and functional programming or procedural styles.

Python is not intended to work on special area such as web programming. That is why it is known as *multipurpose* because it can be used with web, enterprise, 3D CAD etc.

We don't need to use data types to declare variable because it is *dynamically typed* so we can write `a=10` to declare an integer value in a variable.

Python makes the development and debugging *fast* because there is no compilation step included in python development and edit-test-debug cycle is very fast.

PYTHON FEATURES

There are a lot of features provided by python programming language.

1) EASY TO USE:

Python is easy to very easy to use and high level language. Thus it is programmer-friendly language.

2) EXPRESSIVE LANGUAGE:

Python language is more expressive. The sense of expressive is the code is easily understandable.

3) INTERPRETED LANGUAGE:

Python is an interpreted language i.e. interpreter executes the code line by line at a time. This makes debugging easy and thus suitable for beginners.

4) CROSS-PLATFORM LANGUAGE:

Python can run equally on different platforms such as Windows, Linux, Unix , Macintosh etc. Thus, Python is a portable language.

5) FREE AND OPEN SOURCE:

Python language is freely available(www.python.org).The source-code is also available. Therefore it is open source.

6) OBJECT-ORIENTED LANGUAGE:

Python supports object oriented language. Concept of classes and objects comes into existence.

7) EXTENSIBLE:

It implies that other languages such as C/C++ can be used to compile the code and thus it can be used further in your python code.

8) LARGE STANDARD LIBRARY:

Python has a large and broad library.

9) GUI PROGRAMMING:

Graphical user interfaces can be developed using Python.

10) INTEGRATED:

It can be easily integrated with languages like C, C++, JAVA etc.

PYTHON HISTORY

- Python laid its foundation in the late 1980s.
- The implementation of Python was started in the December 1989 by **Guido Van Rossum** at CWI in Netherland.
- *ABC programming language* is said to be the predecessor of Python language which was capable of Exception Handling and interfacing with Amoeba Operating System.
- Python is influenced by programming languages like:
 - ABC language.
 - Modula-3

PYTHON VERSION

Python programming language is being updated regularly with new features and support. There are a lot of updation in python versions, started from 1994 to current date.

A list of python versions with its released date is given below.

Python Version	Released Date
Python 1.0	January 1994
Python 1.5	December 31, 1997
Python 1.6	September 5, 2000
Python 2.0	October 16, 2000

Python 2.1	April 17, 2001
Python 2.2	December 21, 2001
Python 2.3	July 29, 2003
Python 2.4	November 30, 2004
Python 2.5	September 19, 2006
Python 2.6	October 1, 2008
Python 2.7	July 3, 2010
Python 3.0	December 3, 2008
Python 3.1	June 27, 2009
Python 3.2	February 20, 2011
Python 3.3	September 29, 2012

PYTHON APPLICATIONS

Python as a whole can be used in any sphere of development.

Let us see what are the major regions where Python proves to be handy.

1) CONSOLE BASED APPLICATION

Python can be used to develop console based applications. For example: **IPython**.

2) AUDIO OR VIDEO BASED APPLICATIONS

Python proves handy in multimedia section. Some of real applications are: TimPlayer, cplay etc.

3) 3D CAD APPLICATIONS

Fandango is a real application which provides full features of CAD.

4) WEB APPLICATIONS

Python can also be used to develop web based application. Some important developments are: PythonWikiEngines, Pocoo, PythonBlogSoftware etc.

5) ENTERPRISE APPLICATIONS

Python can be used to create applications which can be used within an Enterprise or an Organization. Some real time applications are: OpenErp, Tryton, Picalo etc.

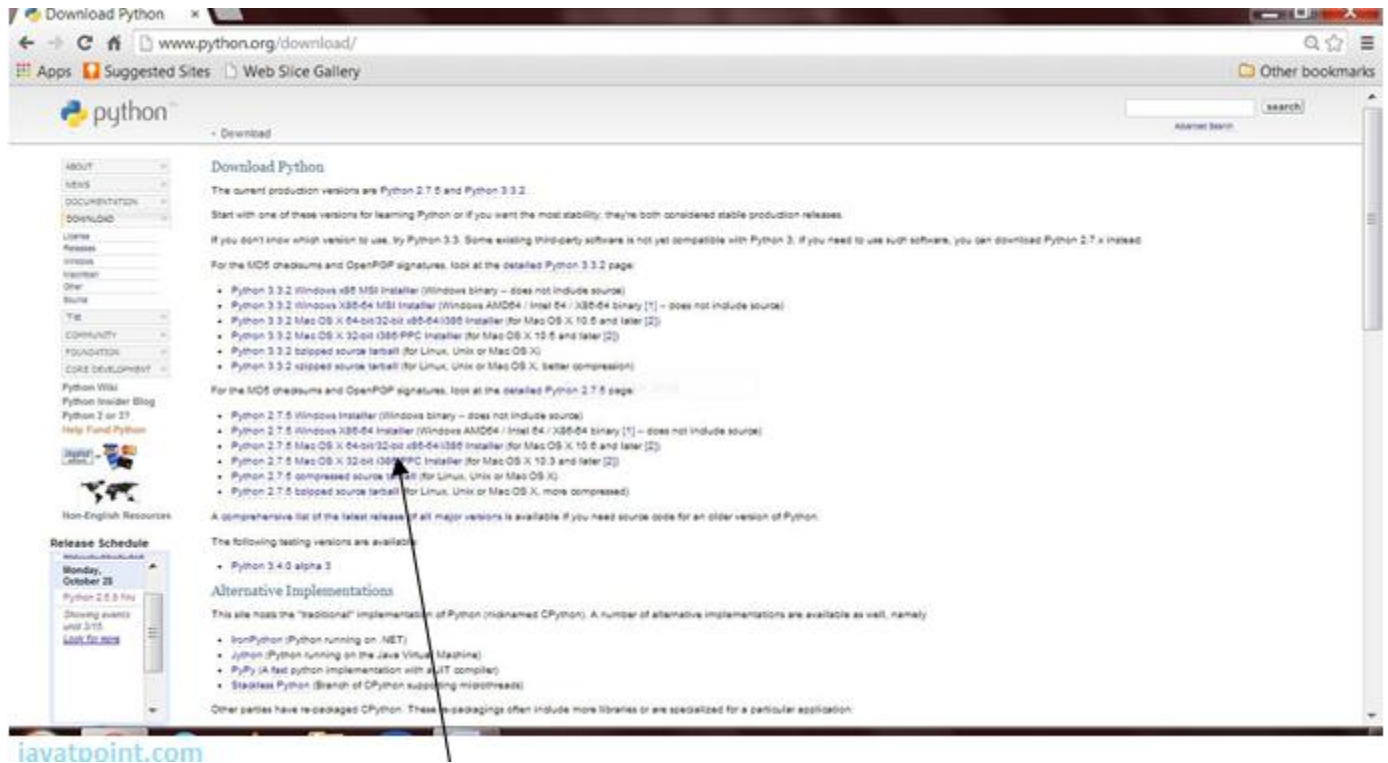
6) APPLICATIONS FOR IMAGES

Using Python several application can be developed for image. Applications developed are: VPython, Gogh, imgSeek etc.

There are several such applications which can be developed using Python

HOW TO INSTALL PYTHON

1. To install Python, firstly download the Python distribution from www.python.org/download.



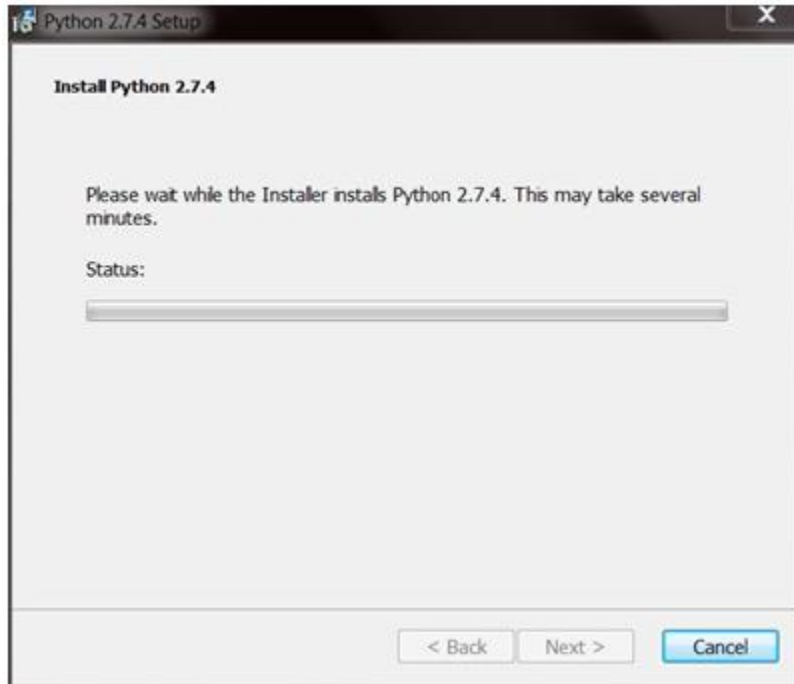
2. Having downloaded the Python distribution now execute it. Double click on the downloaded software. Follow the steps for installation:



iavatpoint.com



iavatpoint.com



iavatpoint.com



iavatpoint.com

Click the Finish button and Python will be installed on your system.

SETTING PATH IN PYTHON

Before starting working with Python, a specific path is to set.

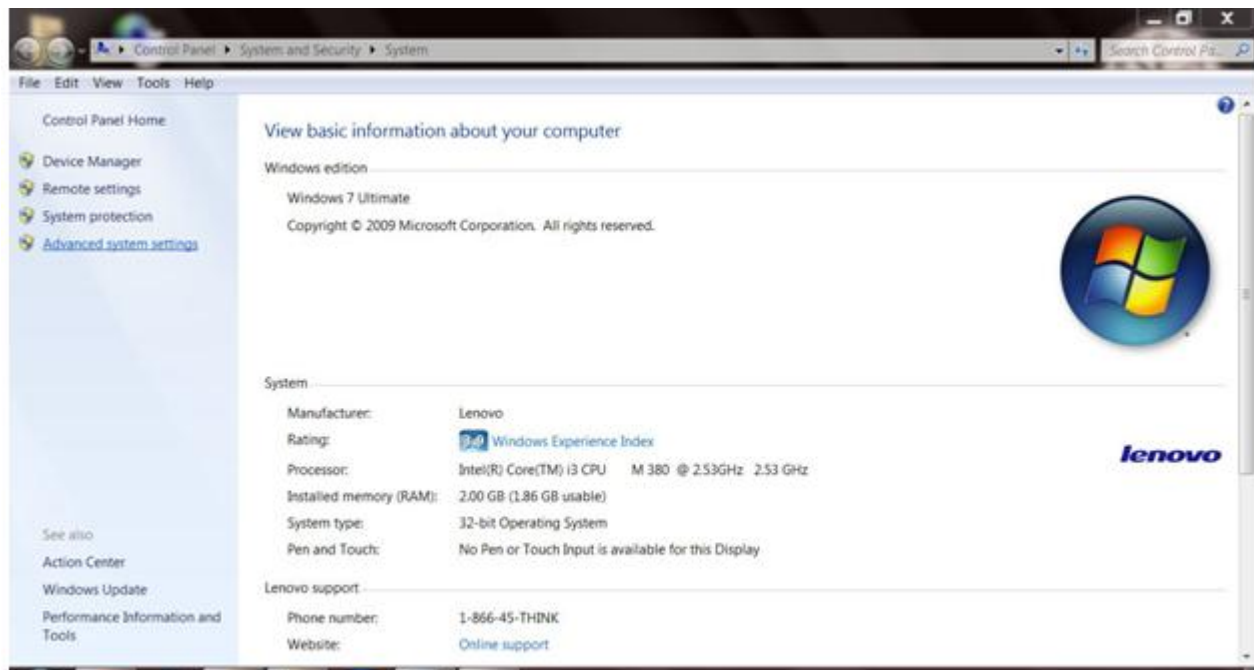
- Your Python program and executable code can reside in any directory of your system, therefore Operating System provides a specific search path that index the directories Operating System should search for executable code.
- The Path is set in the Environment Variable of My Computer properties:
- To set path follow the steps:

Right click on My Computer ->Properties ->Advanced System setting ->Environment Variable ->New

In Variable name write path and in Variable value copy path up to C://Python(i.e., path where Python is installed). Click Ok ->Ok.

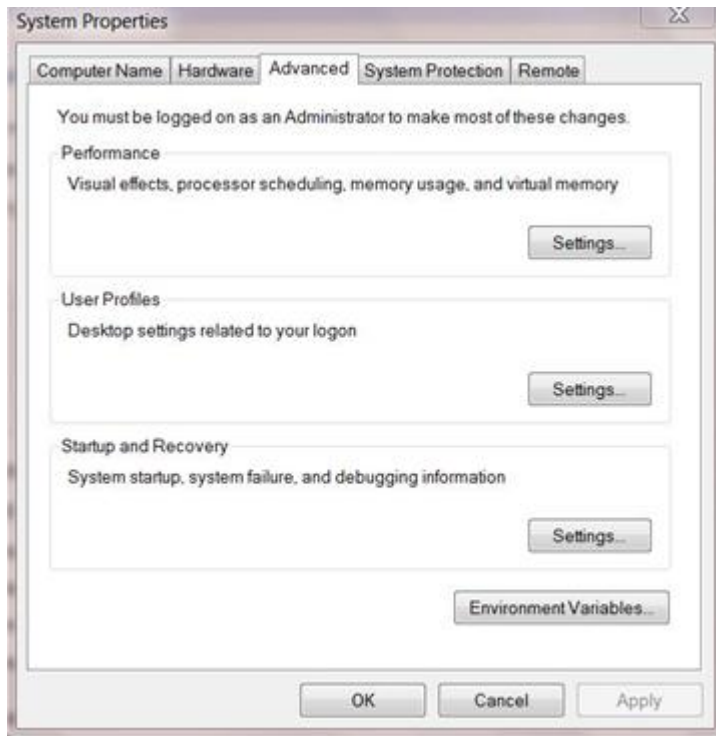
Path will be set for executing Python programs.

1. Right click on My Computer and click on properties.
2. Click on Advanced System settings



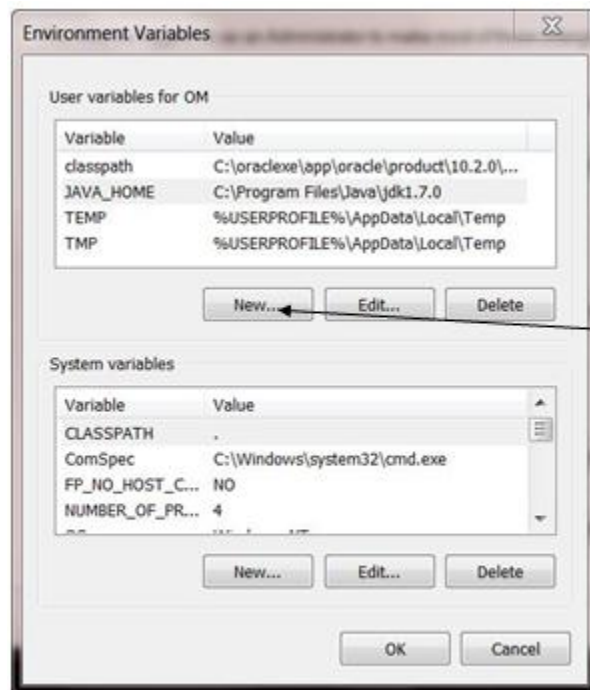
javatpoint.com

3. Click on Environment Variable tab.



iavatpoint.com

4. Click on new tab of user variables.



iavatpoint.com

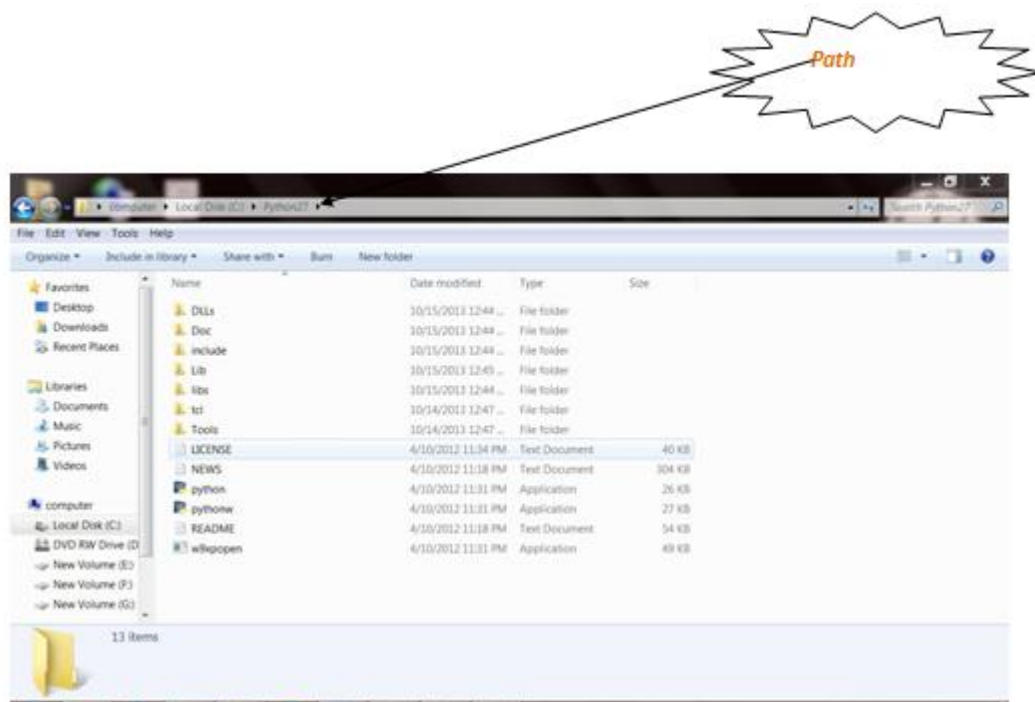


5. Write path in variable name



javatpoint.com

6. Copy the path of Python folder



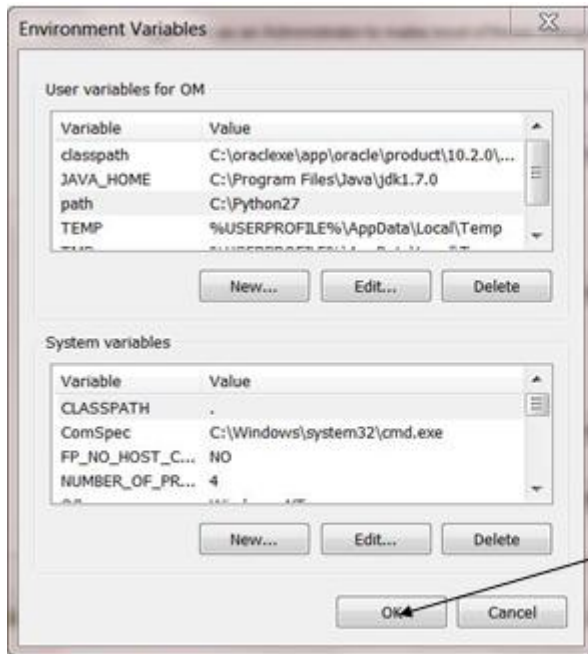
javatpoint.com

7. Paste path of Python in variable value.



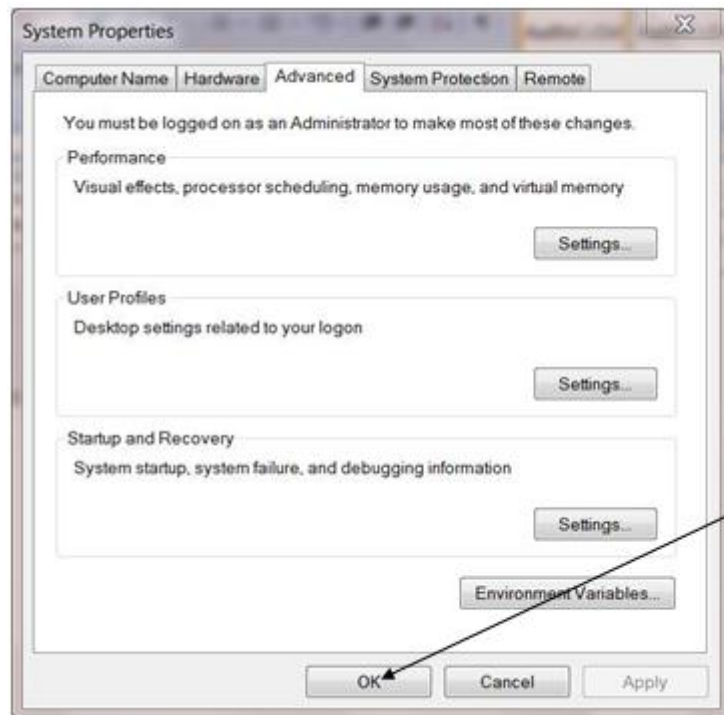
javatpoint.com

8. Click on Ok button:



iavatpoint.com

9. Click on Ok button:



iavatpoint.com

PYTHON EXAMPLE

Python code is simple and easy to run. Here is a simple Python code that will print "Welcome to Python".

A simple python example is given below.

```
1.      >>> a="Welcome To Python"
2.      >>> print a
3.      Welcome To Python
4.      >>>
```

Explanation:

- Here we are using IDLE to write the Python code. Detail explanation to run code is given in Execute Python section.
- A variable is defined named "a" which holds "Welcome To Python".
- "print" statement is used to print the content. Therefore "print a" statement will print the content of the variable. Therefore, the output "Welcome To Python" is produced.

PYTHON 3.4 EXAMPLE

In python 3.4 version, you need to add parenthesis () in a string code to print it.

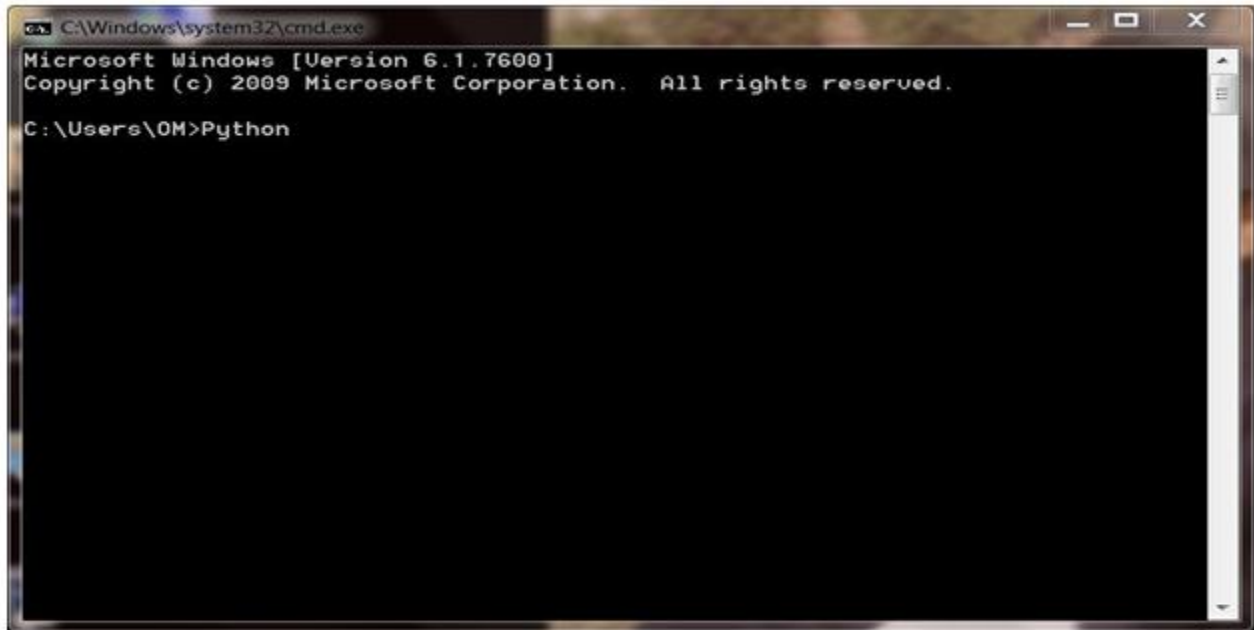
```
1.      >>> a=("Welcome To Python Example")
2.      >>> print a
3.      Welcome To Python Example
4.      >>>
```

HOW TO EXECUTE PYTHON

There are three different ways of working in Python:

1) INTERACTIVE MODE:

You can enter python in the command prompt and start working with Python.

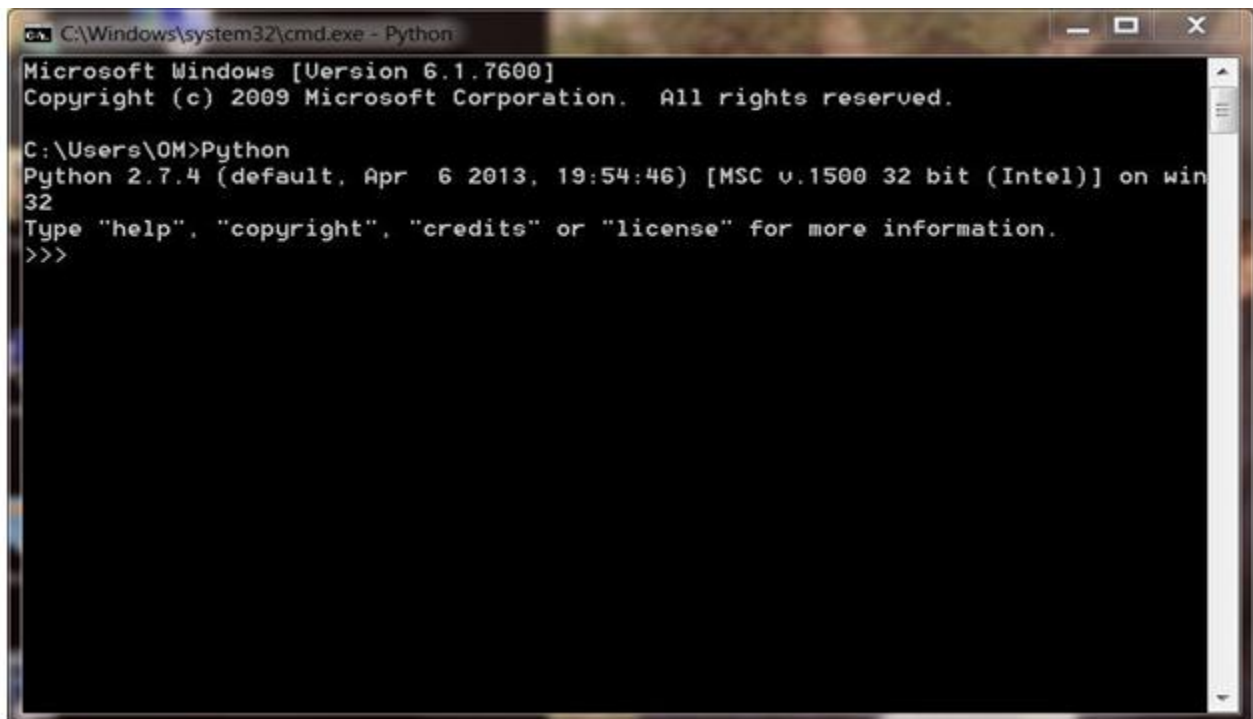


```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\OM>Python
```

javatpoint.cpm

Press Enter key and the Command Prompt will appear like:



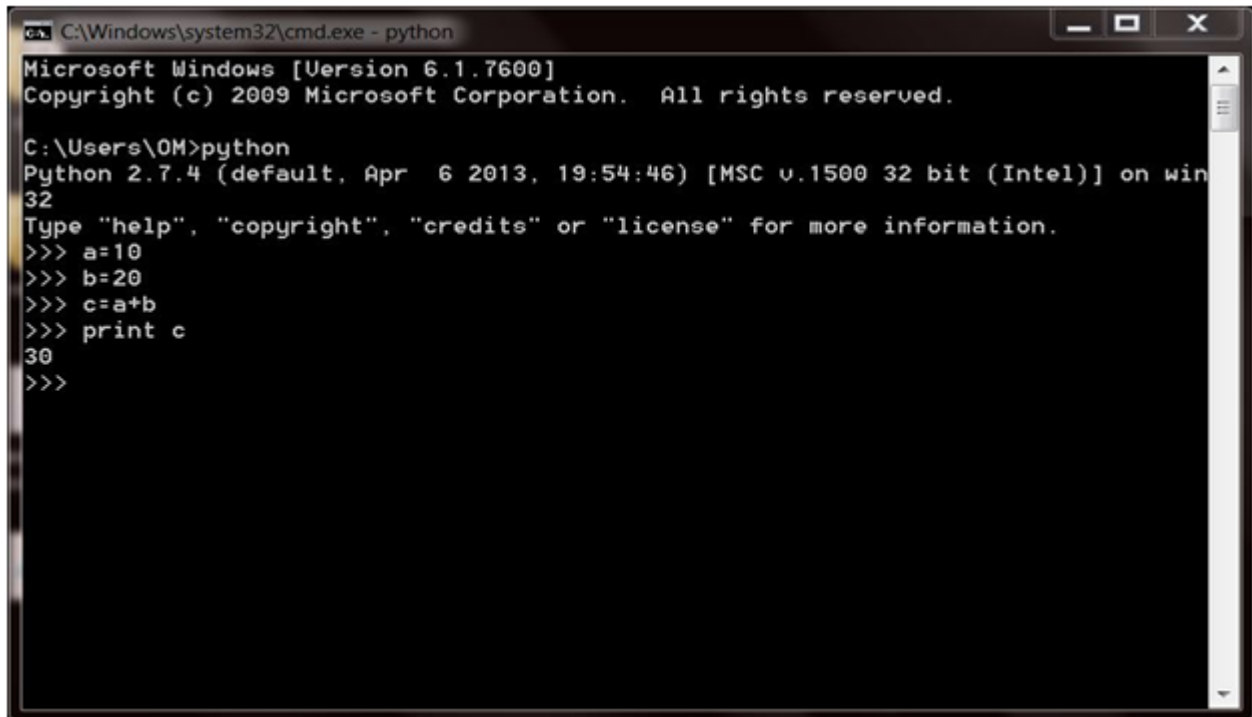
```
C:\Windows\system32\cmd.exe - Python
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\OM>Python
Python 2.7.4 (default, Apr 6 2013, 19:54:46) [MSC v.1500 32 bit (Intel)] on win
32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

javatpoint.com

Now you can execute your Python commands.

Eg:



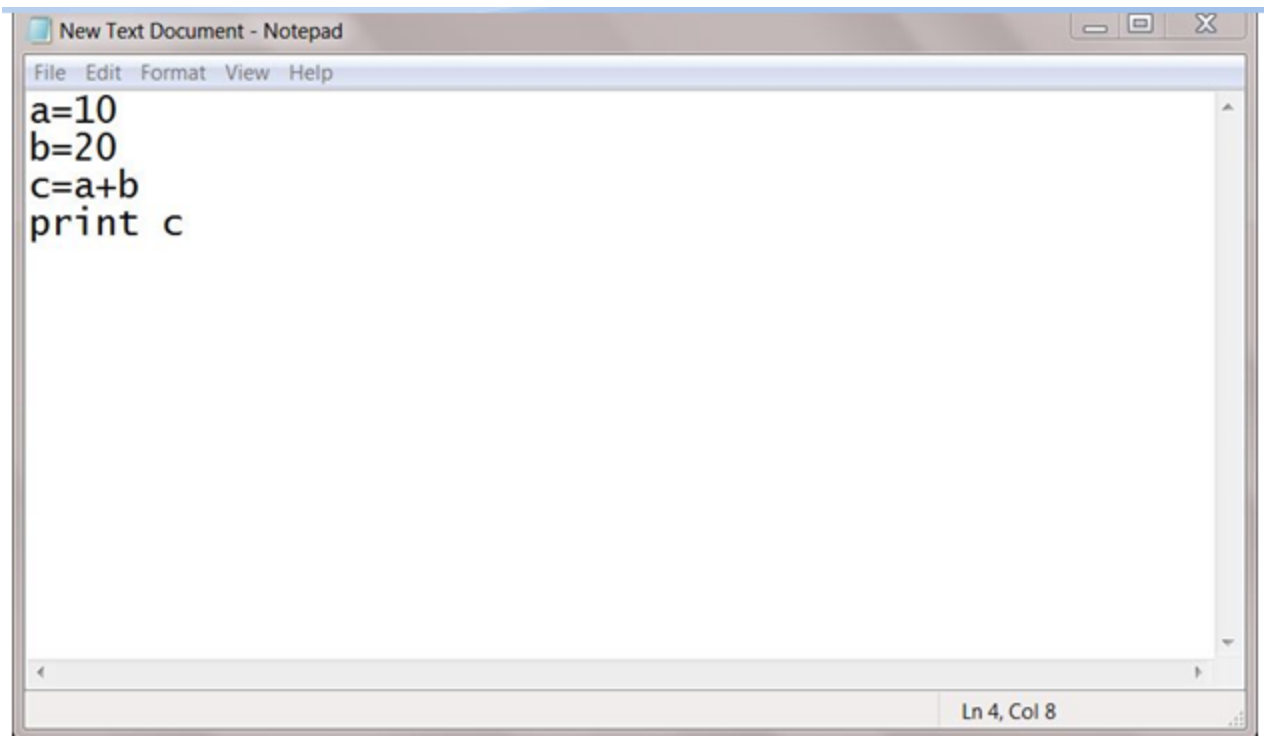
```
C:\Windows\system32\cmd.exe - python
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\OM>python
Python 2.7.4 (default, Apr 6 2013, 19:54:46) [MSC v.1500 32 bit (Intel)] on win
32
Type "help", "copyright", "credits" or "license" for more information.
>>> a=10
>>> b=20
>>> c=a+b
>>> print c
30
>>>
```

iavatpoint.com

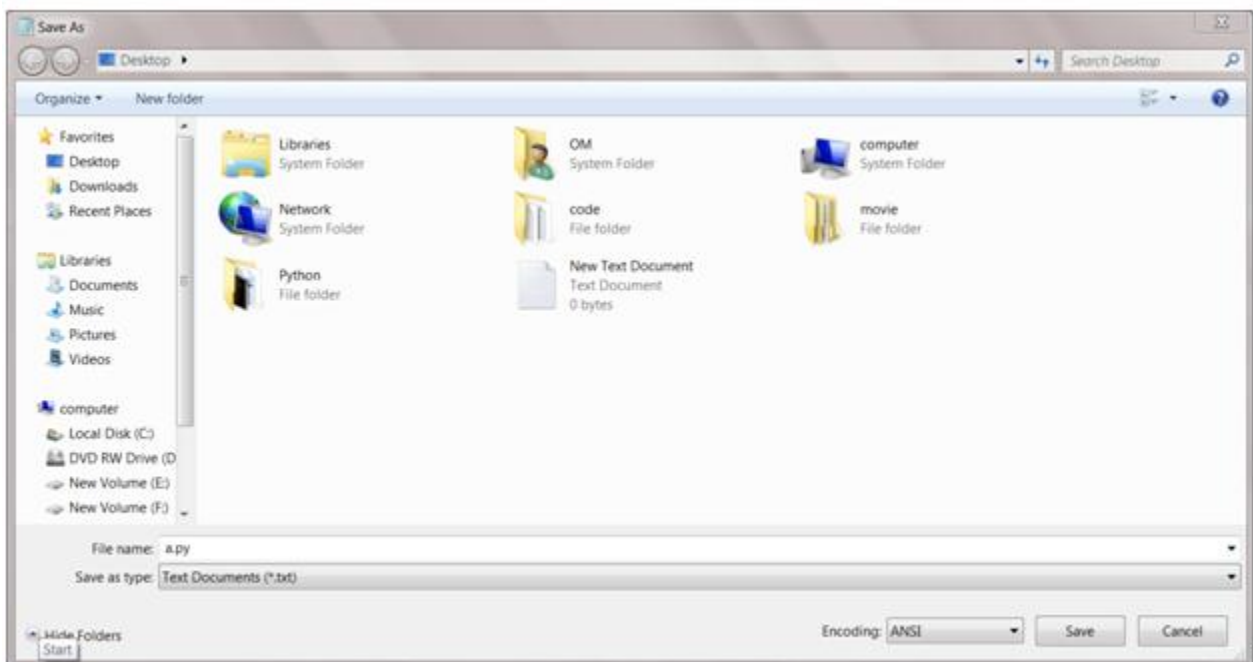
2) SCRIPT MODE:

Using Script Mode , you can write your Python code in a separate file using any editor of your Operating System.



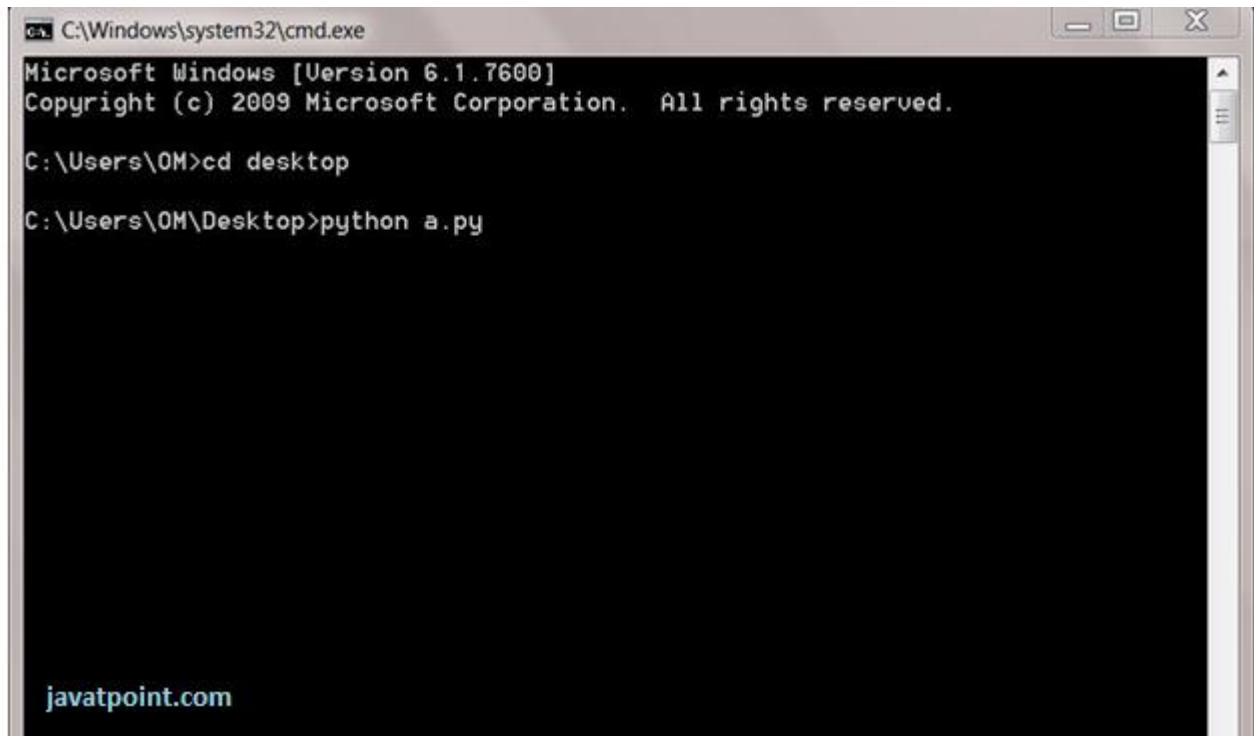
iavatpoint.com

Save it by .py extension.



iavatpoint.com

Now open Command prompt and execute it by :



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\OM>cd desktop
C:\Users\OM\Desktop>python a.py

javatpoint.com
```

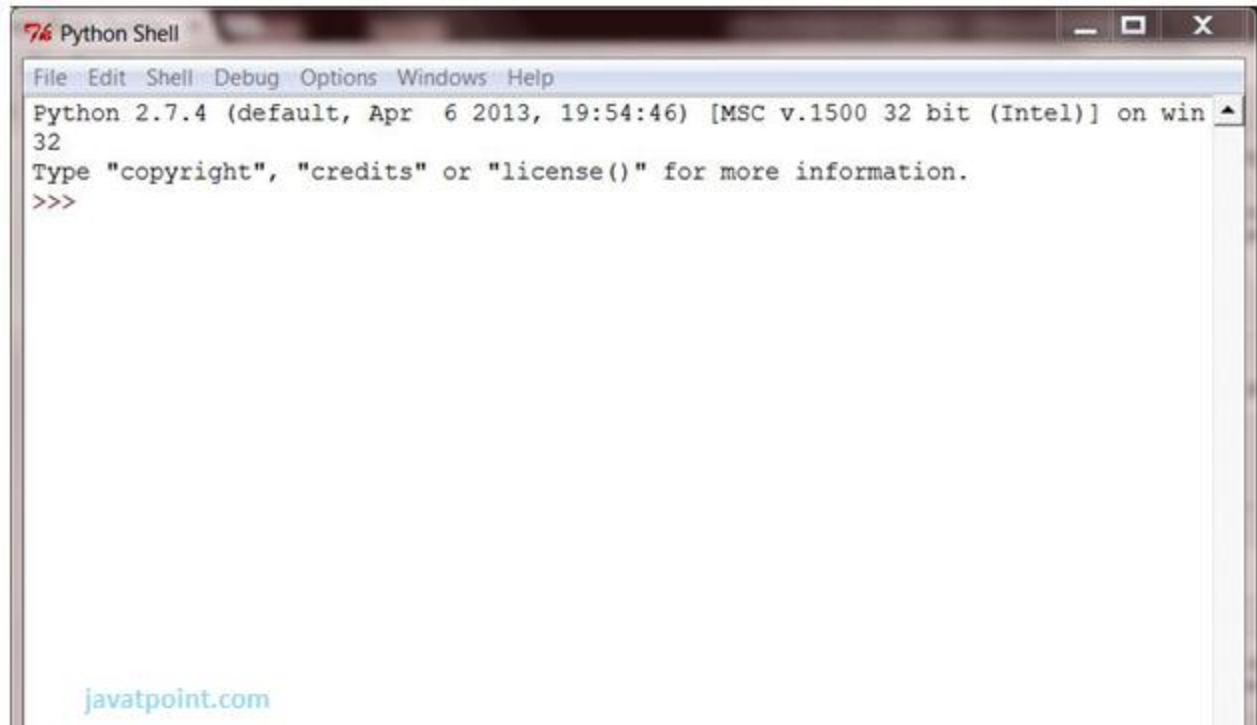
NOTE: Path in the command prompt should be where you have saved your file. In the above case file should be saved at desktop.

3) USING IDE: (INTEGRATED DEVELOPMENT ENVIRONMENT)

You can execute your Python code using a Graphical User Interface (GUI).

All you need to do is:

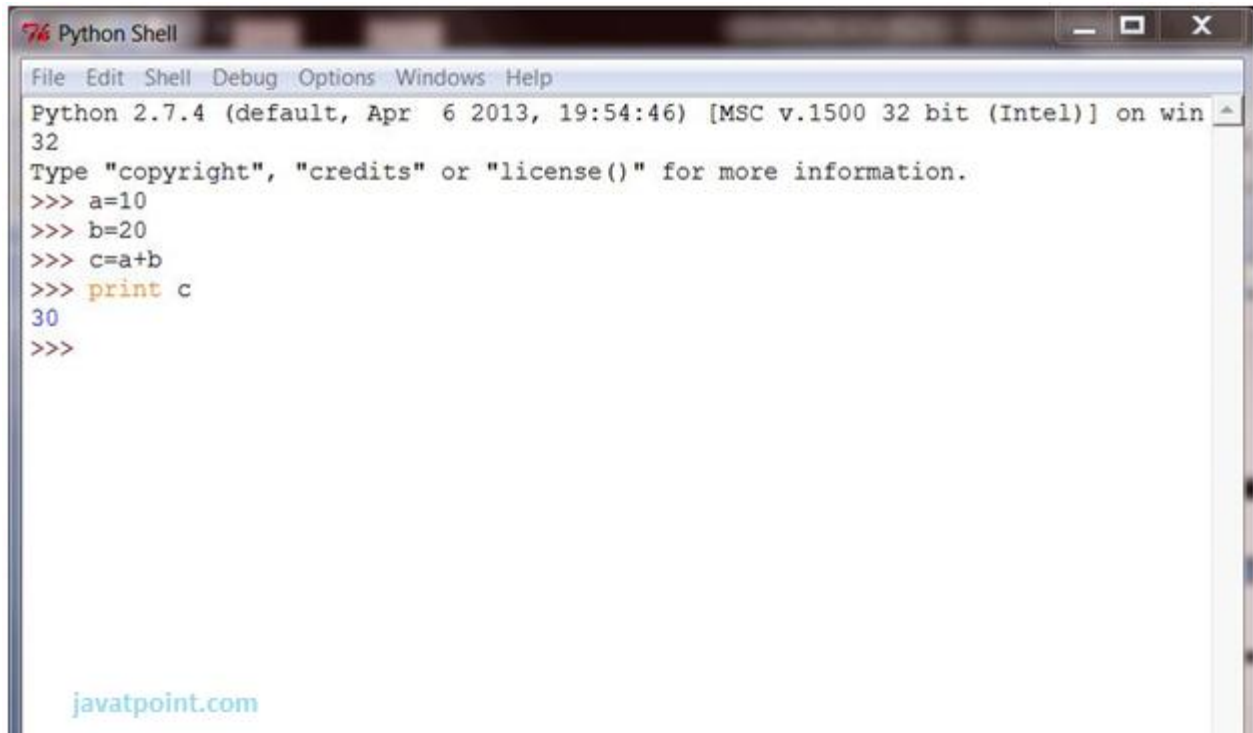
Click on Start button -> All Programs -> Python -> IDLE(Python GUI)



You can use both Interactive as well as Script mode in IDE.

1) Using Interactive mode:

Execute your Python code on the Python prompt and it will display result simultaneously.



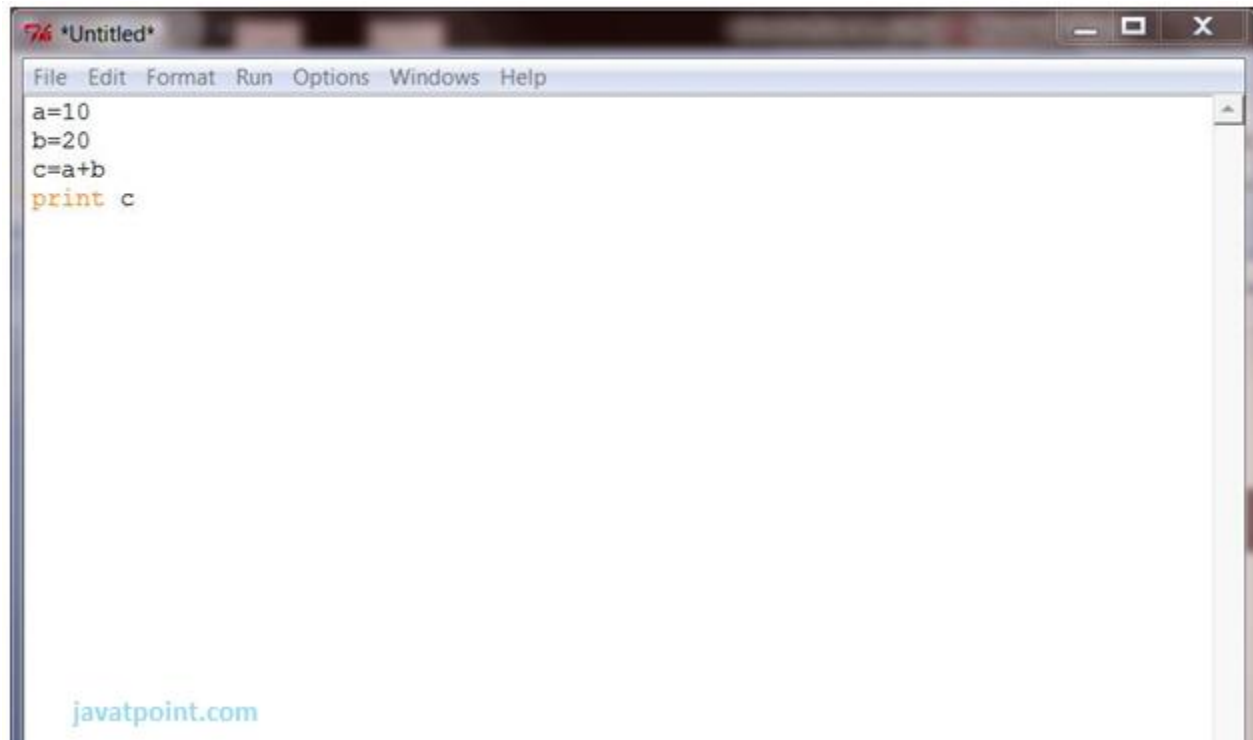
The image shows a screenshot of a Python Shell window. The title bar reads "Python Shell". The menu bar includes "File", "Edit", "Shell", "Debug", "Options", "Windows", and "Help". The main text area displays the following content:

```
Python 2.7.4 (default, Apr 6 2013, 19:54:46) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> a=10
>>> b=20
>>> c=a+b
>>> print c
30
>>>
```

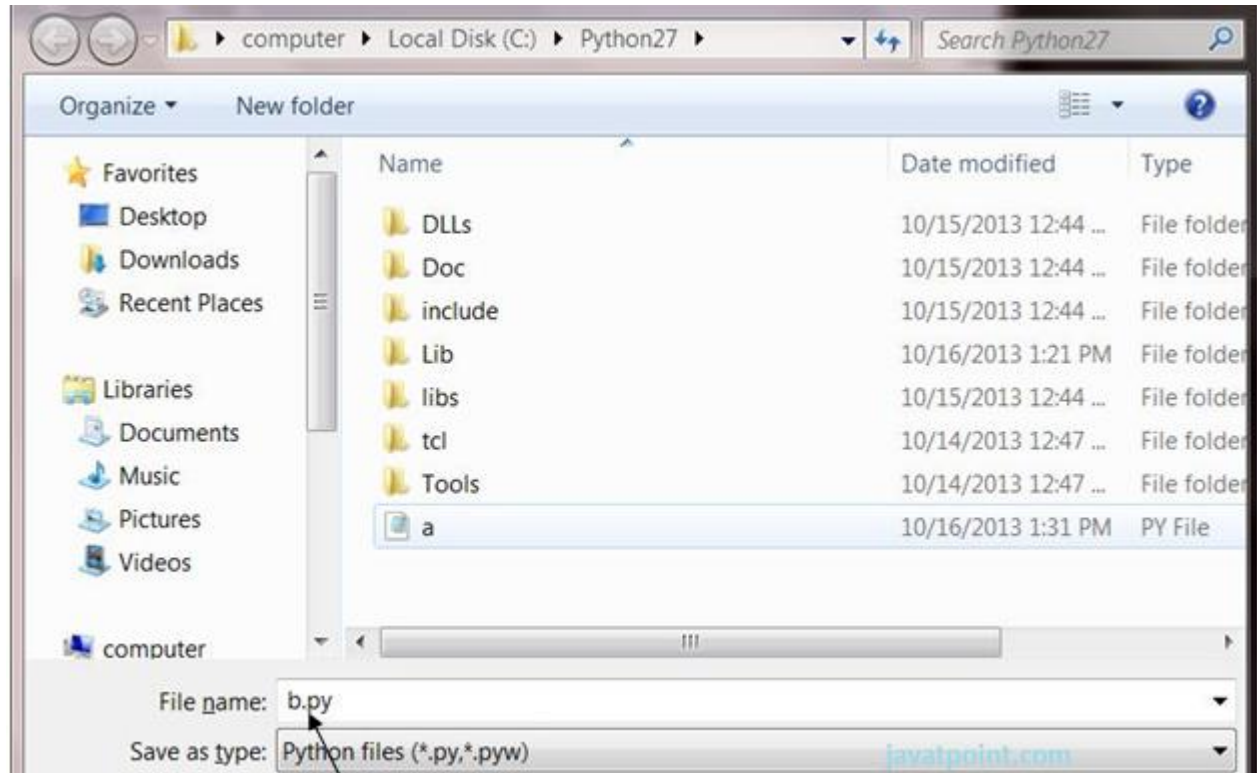
At the bottom left of the window, the text "javatpoint.com" is visible.

2) Using Script Mode:

- i) Click on Start button -> All Programs -> Python -> IDLE(Python GUI)
 - ii) Python Shell will be opened. Now click on File -> New Window.
- A new Editor will be opened . Write your Python code here.



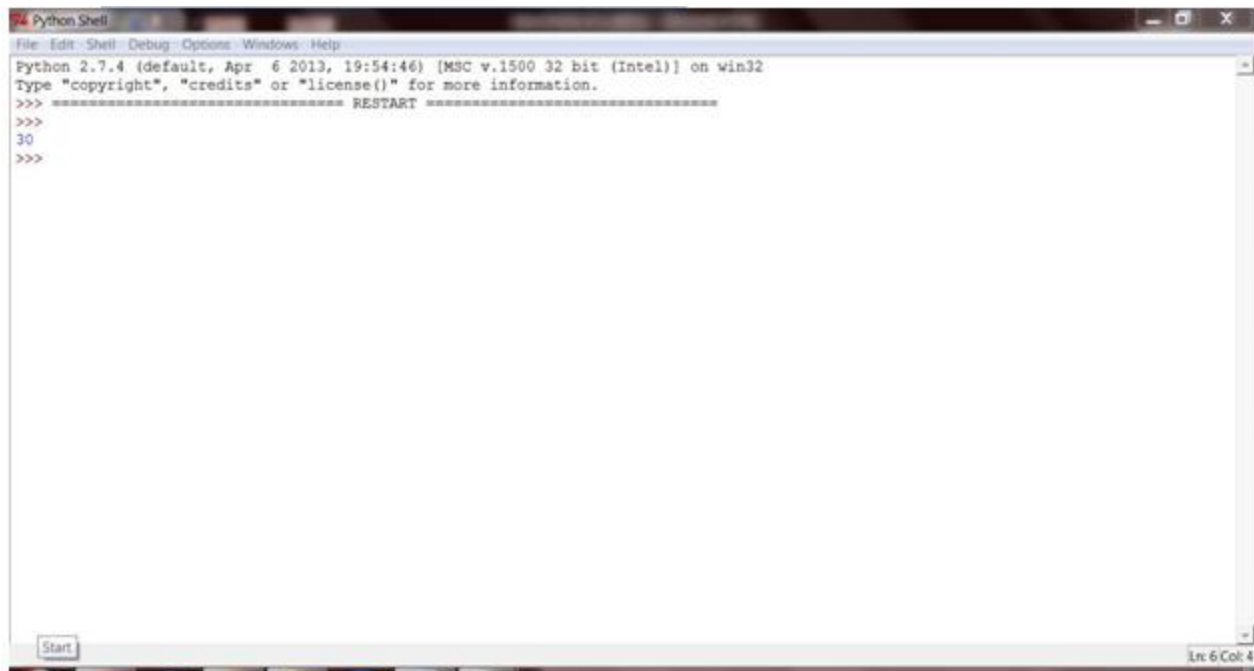
Click on file -> save as



Run then code by clicking on Run in the Menu bar.

Run -> Run Module

Result will be displayed on a new Python shell as:

A screenshot of a Python Shell window. The title bar says "Python Shell". The menu bar includes "File", "Edit", "Shell", "Debug", "Options", "Windows", and "Help". The main text area shows the following: "Python 2.7.4 (default, Apr 6 2013, 19:54:46) [MSC v.1500 32 bit (Intel)] on win32", "Type \"copyright\", \"credits\" or \"license()\" for more information.", a prompt ">>> ", a line of text "===== RESTART =====", another prompt ">>> ", the number "30", and a final prompt ">>> ". At the bottom left is a "Start" button, and at the bottom right is a status bar showing "Line: 6 Col: 4".

javatpoint.com

PYTHON VARIABLES

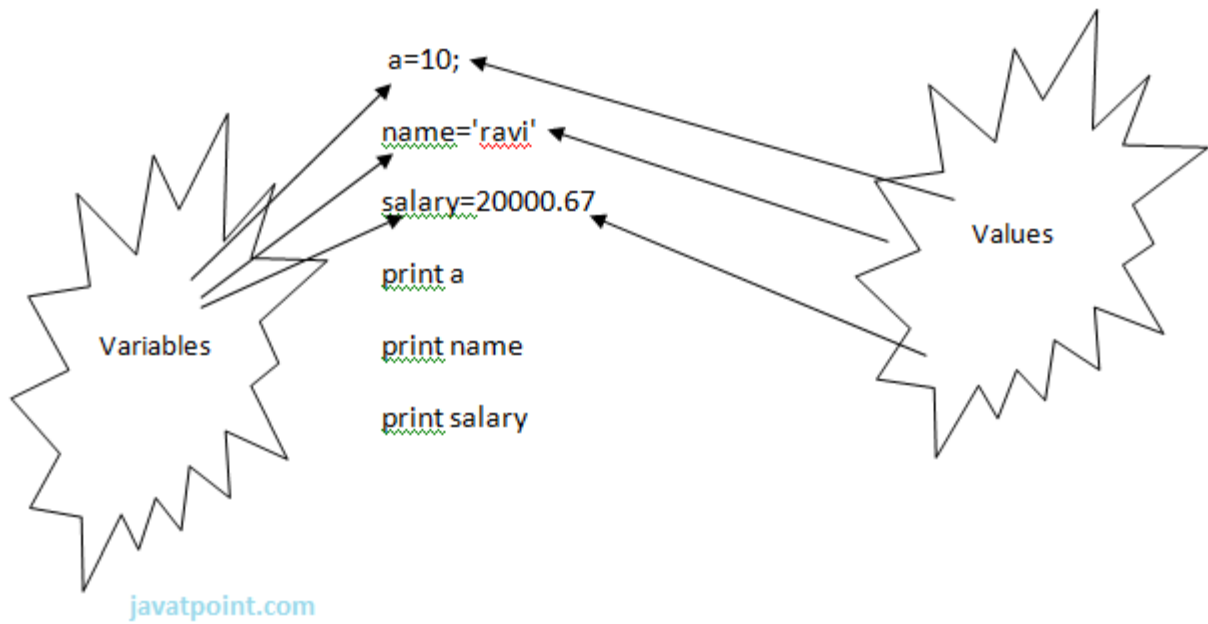
Variable is a name of the memory location where data is stored. Once a variable is stored that means a space is allocated in memory.

ASSIGNING VALUES TO VARIABLE:

We need not to declare explicitly variable in Python. When we assign any value to the variable that variable is declared automatically.

The assignment is done using the equal (=) operator.

Eg:



Output:

1. >>>
2. 10
3. ravi
4. 20000.67
5. >>>

MULTIPLE ASSIGNMENT:

Multiple assignment can be done in Python at a time.

There are two ways to assign values in Python:

1. Assigning single value to multiple variables:

Eg:

1. x=y=z=50
2. **print** x
3. **print** y
4. **print** z

Output:

1. >>>

```
2.      50
3.      50
4.      50
5.      >>>
```

2.Assigning multiple values to multiple variables:

Eg:

```
1.      a,b,c=5,10,15
2.      print a
3.      print b
4.      print c
```

Output:

```
1.      >>>
2.      5
3.      10
4.      15
5.      >>>
```

The values will be assigned in the order in which variables appears.

BASIC FUNDAMENTALS:

This section contains the basic fundamentals of Python like :

i)Tokens and their types.

ii) Comments

a)Tokens:

- Tokens can be defined as a punctuator mark, reserved words and each individual word in a statement.
- Token is the smallest unit inside the given program.

There are following tokens in Python:

- Keywords.
- Identifiers.

- Literals.
- Operators.

TUPLES:

- Tuple is another form of collection where different type of data can be stored.
- It is similar to list where data is separated by commas. Only the difference is that list uses square bracket and tuple uses parenthesis.
- Tuples are enclosed in parenthesis and cannot be changed.

Eg:

```
1.      >>> tuple=('rahul',100,60.4,'deepak')
2.      >>> tuple1=('sanjay',10)
3.      >>> tuple
4.      ('rahul', 100, 60.4, 'deepak')
5.      >>> tuple[2:]
6.      (60.4, 'deepak')
7.      >>> tuple1[0]
8.      'sanjay'
9.      >>> tuple+tuple1
10.     ('rahul', 100, 60.4, 'deepak', 'sanjay', 10)
11.     >>>
```

DICTIONARY:

- Dictionary is a collection which works on a key-value pair.
- It works like an associated array where no two keys can be same.
- Dictionaries are enclosed by curly braces ({}) and values can be retrieved by square bracket([]).

Eg:

```
1.      >>> dictionary={'name':'charlie','id':100,'dept':'it'}
2.      >>> dictionary
3.      {'dept': 'it', 'name': 'charlie', 'id': 100}
4.      >>> dictionary.keys()
5.      ['dept', 'name', 'id']
6.      >>> dictionary.values()
```


7. `['it', 'charlie', 100]`
8. `>>>`

PYTHON KEYWORDS

Keywords are special reserved words which convey a special meaning to the compiler/interpreter. Each keyword has a special meaning and a specific operation. List of Keywords used in Python are:

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

IDENTIFIERS

Identifiers are the names given to the fundamental building blocks in a program.

These can be variables, class, object, functions, lists, dictionaries etc.

There are certain rules defined for naming i.e., Identifiers.

I. An identifier is a long sequence of characters and numbers.

II. No special character except underscore (`_`) can be used as an identifier.

III. Keyword should not be used as an identifier name.

IV. Python is case sensitive. So using case is significant.

V. First character of an identifier can be character, underscore (`_`) but not digit.

PYTHON LITERALS

Literals can be defined as a data that is given in a variable or constant.

Python support the following literals:

I. String literals:

String literals can be formed by enclosing a text in the quotes. We can use both single as well as double quotes for a String.

Eg:

```
"Aman" , '12345'
```

Types of Strings:

There are two types of Strings supported in Python:

a).Single line String- Strings that are terminated within a single line are known as Single line Strings.

Eg:

```
1. >>> text1='hello'
```

b).Multi line String- A piece of text that is spread along multiple lines is known as Multiple line String.

There are two ways to create Multiline Strings:

1). Adding black slash at the end of each line.

Eg:

```
1. >>> text1='hello\  
2. user'  
3. >>> text1  
4. 'hellouser'  
5. >>>
```

2).Using triple quotation marks:-

Eg:

```
1. >>> str2="""welcome  
2. to  
3. SSSIT"""
```

```

4.      >>> print str2
5.      welcome
6.      to
7.      SSSIT
8.      >>>

```

II. Numeric literals:

Numeric Literals are immutable. Numeric literals can belong to following four different numerical types.

Int(signed integers)	Long(long integers)	float(floating point)	Complex(complex)
Numbers(can be both positive and negative) with no fractional part.eg: 100	Integers of unlimited size followed by lowercase or uppercase L eg: 87032845L	Real numbers with both integer and fractional part eg: -26.2	In the form of a+ bj the real part a and imaginary part b eg: 3.14j

III. Boolean literals:

A Boolean literal can have any of the two values: True or False.

IV. Special literals.

Python contains one special literal i.e., None.

None is used to specify to that field that is not created. It is also used for end of lists in Python.

Eg:

```

1.      >>> val1=10
2.      >>> val2=None
3.      >>> val1
4.      10
5.      >>> val2
6.      >>> print val2
7.      None
8.      >>>

```

V.Literal Collections.

Collections such as tuples, lists and Dictionary are used in Python.

List:

- List contain items of different data types. Lists are mutable i.e., modifiable.
- The values stored in List are separated by commas(,) and enclosed within a square brackets([]). We can store different type of data in a List.
- Value stored in a List can be retrieved using the slice operator([] and [:]).
- The plus sign (+) is the list concatenation and asterisk(*) is the repetition operator.

Eg:

```
1.      >>> list=['aman',678,20.4,'saurav']
2.      >>> list1=[456,'rahul']
3.      >>> list
4.      ['aman', 678, 20.4, 'saurav']
5.      >>> list[1:3]
6.      [678, 20.4]
7.      >>> list+list1
8.      ['aman', 678, 20.4, 'saurav', 456, 'rahul']
9.      >>> list1*2
10.     [456, 'rahul', 456, 'rahul']
11.     >>>
```

PYTHON OPERATORS

Operators are particular symbols which operate on some values and produce an output.

The values are known as Operands.

Eg:

```
1.      4 + 5 = 9
```

Here 4 and 5 are Operands and (+) , (=) signs are the operators. They produce the output 9.

Python supports the following operators:

1. Arithmetic Operators.

2. Relational Operators.
3. Assignment Operators.
4. Logical Operators.
5. Membership Operators.
6. Identity Operators.
7. Bitwise Operators.

Arithmetic Operators:

Operators	Description
//	Perform Floor division(gives integer value after division)
+	To perform addition
-	To perform subtraction
*	To perform multiplication
/	To perform division
%	To return remainder after division(Modulus)
**	Perform exponent(raise to power)

eg:

1. >>> 10+20
2. 30
3. >>> 20-10
4. 10
5. >>> 10*2
6. 20
7. >>> 10/2
8. 5

9. >>> 10%3
10. 1
11. >>> 2**3
12. 8
13. >>> 10//3
14. 3
15. >>>

Relational Operators:

Operators	Description
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
!=	Not equal to
<>	Not equal to(similar to !=)

eg:

1. >>> 10<20
2. True
3. >>> 10>20
4. False
5. >>> 10<=10
6. True
7. >>> 20>=15
8. True

9. >>> 5==6
10. False
11. >>> 5!=6
12. True
13. >>> 10<>2
14. True
15. >>>

Assignment Operators:

Operators	Description
=	Assignment
/=	Divide and Assign
+=	Add and assign
-=	Subtract and Assign
*=	Multiply and assign
%=	Modulus and assign
**=	Exponent and assign
//=	Floor division and assign

eg:

1. >>> c=10
2. >>> c
3. 10
4. >>> c+=5
5. >>> c
6. 15

```

7.      >>> c-=5
8.      >>> c
9.      10
10.     >>> c*=2
11.     >>> c
12.     20
13.     >>> c/=2
14.     >>> c
15.     10
16.     >>> c%=3
17.     >>> c
18.     1
19.     >>> c=5
20.     >>> c**=2
21.     >>> c
22.     25
23.     >>> c//=2
24.     >>> c
25.     12
26.     >>>

```

Logical Operators:

Operators	Description
and	Logical AND(When both conditions are true output will be true)
or	Logical OR (If any one condition is true output will be true)
not	Logical NOT(Compliment the condition i.e., reverse)

eg:

```

1.      a=5>4 and 3>2
2.      print a
3.      b=5>4 or 3<2

```



```
4.     print b
5.     c=not(5>4)
6.     print c
```

Output:

```
1.     >>>
2.     True
3.     True
4.     False
5.     >>>
```

Membership Operators:

Operators	Description
in	Returns true if a variable is in sequence of another variable, else false.
not in	Returns true if a variable is not in sequence of another variable, else false.

eg:

```
1.     a=10
2.     b=20
3.     list=[10,20,30,40,50];
4.     if (a in list):
5.         print "a is in given list"
6.     else:
7.         print "a is not in given list"
8.     if(b not in list):
9.         print "b is not given in list"
10.    else:
11.        print "b is given in list"
```

Output:

```
1.     >>>
```

2. a **is in** given list
3. b **is** given **in** list
4. >>>

Identity Operators:

Operators	Description
is	Returns true if identity of two operands are same, else false
is not	Returns true if identity of two operands are not same, else false.

Example:

1. a=20
2. b=20
3. **if**(a **is** b):
4. **print** ?a,b have same identity?
5. **else**:
6. **print** ?a, b are different?
7. b=10
8. **if**(a **is not** b):
9. **print** ?a,b have different identity?
10. **else**:
11. **print** ?a,b have same identity?

Output:

1. >>>
2. a,b have same identity
3. a,b have different identity
4. >>>

PYTHON COMMENTS

Python supports two types of comments:

1) Single lined comment:

In case user wants to specify a single line comment, then comment must start with `#`?

Eg:

1. `# This is single line comment.`

2) Multi lined Comment:

Multi lined comment can be given inside triple quotes.

eg:

1. `""" This`
2. `Is`
3. `Multipline comment"""`

eg:

1. `#single line comment`
2. `print "Hello Python"`
3. `"""This is`
4. `multiline comment"""`

PYTHON IF STATEMENTS

The if statement in python is same as c language which is used test a condition. If condition is true, statement of if block is executed otherwise it is skipped.

Syntax of python if statement:

1. `if(condition):`
2. `statements`

Example of if statement in python

1. `a=10`
2. `if a==10:`
3. `print "Hello User"`

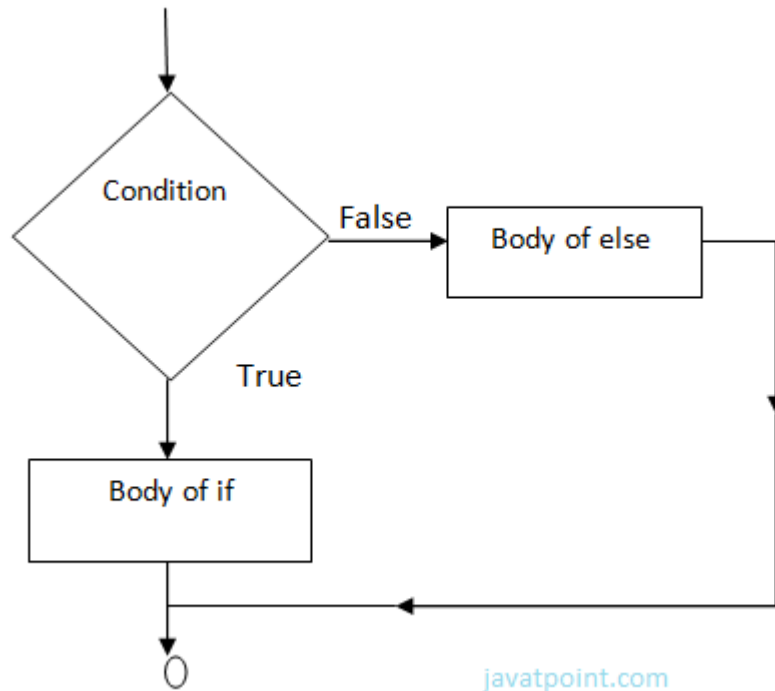
Output:

```
Hello User
```

PYTHON IF ELSE STATEMENTS

Syntax:

1. `if(condition):` False
2. statements
3. `else:` True
4. statements



Example-

1. year=2000
2. `if` year%4==0:
3. `print` "Year is Leap"
4. `else:`
5. `print` "Year is not Leap"

Output:

Year is Leap

NESTED IF ELSE STATEMENT:

When we need to check for multiple conditions to be true then we use elif Statement.

This statement is like executing a if statement inside a else statement.

Syntax:

1. If statement:
2. Body
3. **elif** statement:
4. Body
5. **else:**
6. Body

Example:

1. a=10
2. **if** a>=20:
3. **print** "Condition is True"
4. **else:**
5. **if** a>=15:
6. **print** "Checking second value"
7. **else:**
8. **print** "All Conditions are false"

Output:

All Conditions are false.

FOR LOOP

for Loop is used to iterate a variable over a sequence(i.e., list or string) in the order that they appear.

Syntax:

1. **for** <variable> **in** <sequence>:

Output:

1. 1
- 2.
3. 7

- 4.
5. 9

Explanation:

- Firstly, the first value will be assigned in the variable.
- Secondly all the statements in the body of the loop are executed with the same value.
- Thirdly, once step second is completed then variable is assigned the next value in the sequence and step second is repeated.
- Finally, it continues till all the values in the sequence are assigned in the variable and processed.

Program to display table of Number:

```
1. num=2
2. for a in range (1,6):
3.     print num * a
```

Output:

```
1. 2
2.
3. 4
4.
5. 6
6.
7. 8
8.
9. 10
```

Program to find sum of Natural numbers from 1 to 10.

```
1. sum=0
2. for n in range(1,11):
3.     sum+=n
4. print sum
```

Output:

1. 55

NESTED LOOPS

Loops defined within another Loop is called Nested Loop.

When an outer loop contains an inner loop in its body it is called Nested Looping.

Syntax:

```
1. for <expression>:  
2.     for <expression>:  
3.         Body
```

eg:

```
1. for i in range(1,6):  
2.     for j in range (1,i+1):  
3.         print i,  
4.         print
```

Output:

```
1. >>>  
2. 1  
3. 2 2  
4. 3 3 3  
5. 4 4 4 4  
6. 5 5 5 5 5  
7. >>>
```

Explanation:

For each value of Outer loop the whole inner loop is executed.

For each value of inner loop the Body is executed each time.

Program to print Pyramid:

```
1. for i in range (1,6):  
2.     for j in range (5,i-1,-1):  
3.         print "*",  
4.         print
```

Output:

```
1.      >>>
2.      * * * * *
3.      * * * *
4.      * * *
5.      * *
6.      *
```

WHILE LOOP

while Loop is used to execute number of statements or body till the condition passed in while is true. Once the condition is false, the control will come out of the loop.

Syntax:

```
1.      while <expression>:
2.          Body
```

Here, body will execute multiple times till the expression passed is true. The Body may be a single statement or multiple statement.

Eg:

```
1.      a=10
2.      while a>0:
3.          print "Value of a is",a
4.          a=a-2
```

```
print "Loop is Completed"
```

Output:

```
1.      >>>
2.      Value of a is 10
3.      Value of a is 8
4.      Value of a is 6
5.      Value of a is 4
6.      Value of a is 2
7.      Loop is Completed
8.      >>>
```


Explanation:

- Firstly, the value in the variable is initialized.
- Secondly, the condition/expression in the while is evaluated. Consequently if condition is true, the control enters in the body and executes all the statements . If the condition/expression passed results in false then the control exists the body and straight away control goes to next instruction after body of while.
- Thirdly, in case condition was true having completed all the statements, the variable is incremented or decremented. Having changed the value of variable step second is followed. This process continues till the expression/condition becomes false.
- Finally Rest of code after body is executed.

Program to add digits of a number:

```
1.      n=153
2.      sum=0
3.      while n>0:
4.          r=n%10
5.          sum+=r
6.          n=n/10
7.      print sum
```

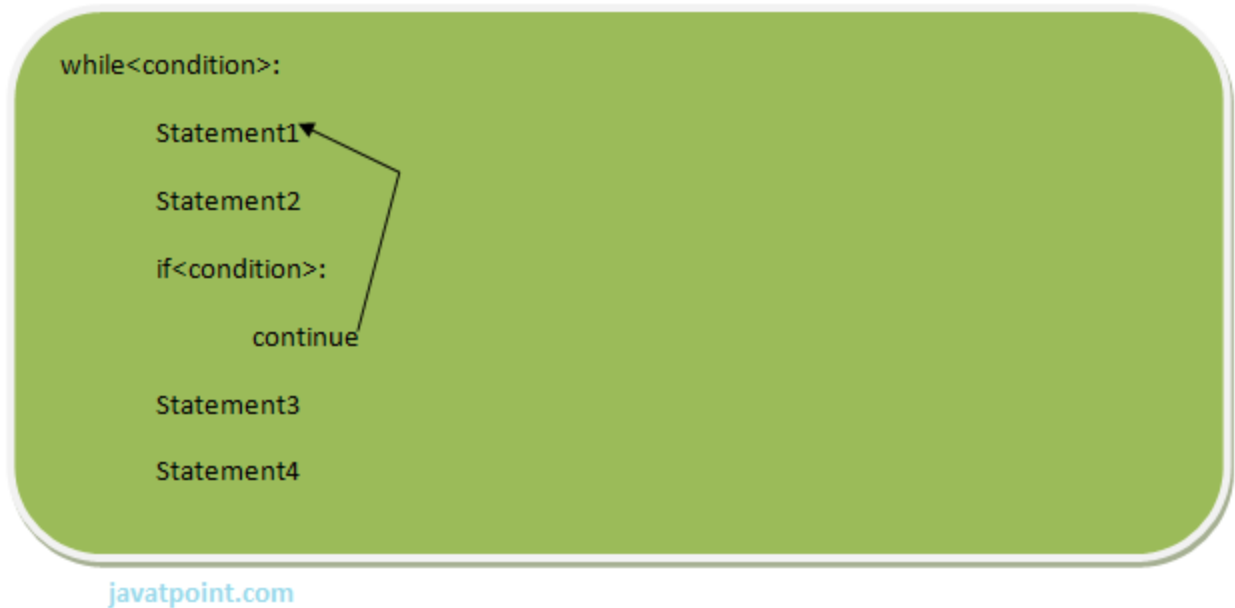
Output:

```
1.      >>>
2.      9
3.      >>>
```

PYTHON BREAK

break statement is a jump statement that is used to pass the control to the end of the loop.

When break statement is applied the control points to the line following the body of the loop , hence applying break statement makes the loop to terminate and controls goes to next line pointing after loop body.



eg:

```
1.     for i in [1,2,3,4,5]:  
2.         if i==4:  
3.             print "Element found"  
4.             break  
5.         print i,
```

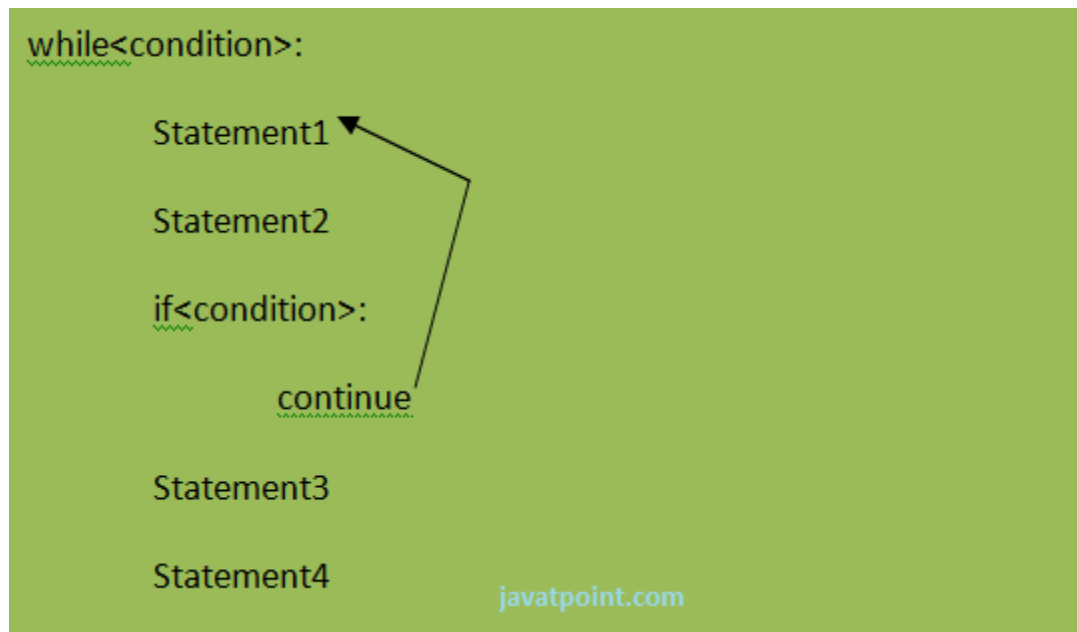
Output:

```
1.     >>>  
2.     1 2 3 Element found  
3.     >>>
```

Flow chart of break:

CONTINUE STATEMENT

continue Statement is a jump statement that is used to skip the present iteration and forces next iteration of loop to take place. It can be used in while as well as for loop statements.



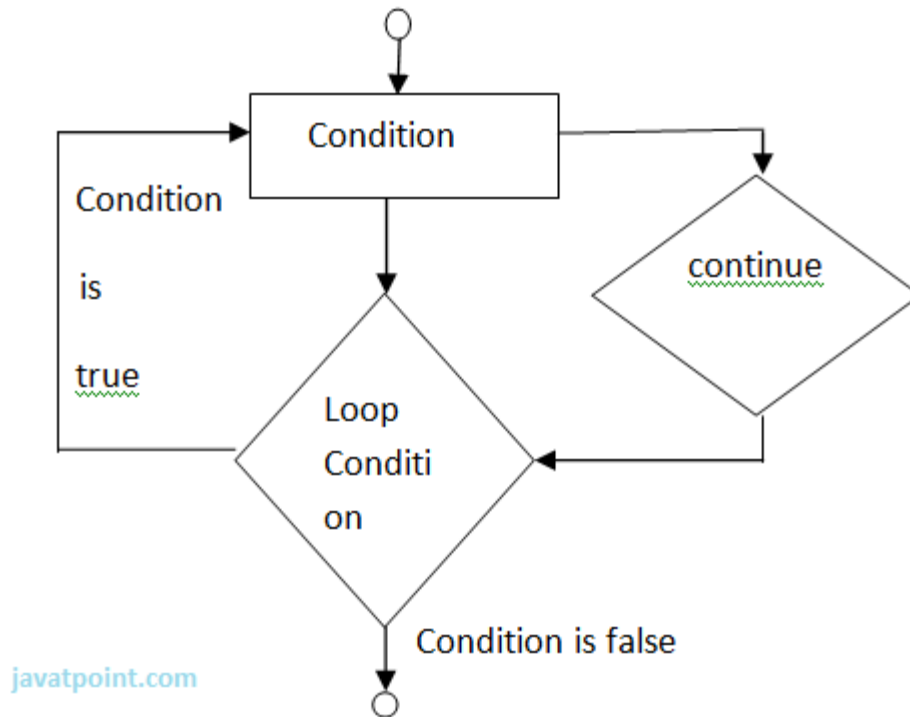
eg:

```
1.      a=0
2.      while a<=5:
3.          a=a+1
4.          if a%2==0:
5.              continue
6.          print a
7.      print "End of Loop"
```

Output:

```
1.      >>>
2.      1
3.      3
4.      5
5.      End of Loop
6.      >>>
```

Flow chart of continue:-



4.

PYTHON PASS

When you do not want any code to execute, pass Statement is used. It is same as the name refers to. It just makes the control to pass by without executing any code. If we want to bypass any code pass statement can be used.

Syntax:

1. **pass**

eg:

```

1. for i in [1,2,3,4,5]:
2.     if i==3:
3.         pass
4.         print "Pass when value is",i
5.     print i,
  
```

Output:

```

1. >>>
2. 1 2 Pass when value is 3
3. 3 4 5
  
```

4. >>>

PYTHON STRINGS

Strings are the simplest and easy to use in Python.

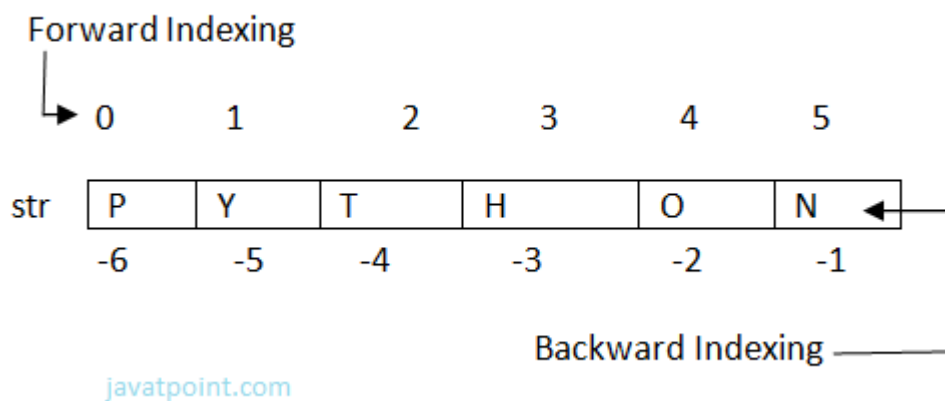
String python are immutable.

We can simply create Python String by enclosing a text in single as well as double quotes. Python treat both single and double quotes statements same.

ACCESSING STRINGS:

- In Python, Strings are stored as individual characters in a contiguous memory location.
- The benefit of using String is that it can be accessed from both the directions in forward and backward.
- Both forward as well as backward indexing are provided using Strings in Python.
 - Forward indexing starts with 0,1,2,3,....
 - Backward indexing starts with -1,-2,-3,-4,....

eg:



1. `str[0]='P'=str[-6] , str[1]='Y' = str[-5] , str[2] = 'T' = str[-4] , str[3] = 'H' = str[-3]`
2. `str[4] = 'O' = str[-2] , str[5] = 'N' = str[-1].`

Simple program to retrieve String in reverse as well as normal form.

```
1.     name="Rajat"
2.     length=len(name)
3.     i=0
4.     for n in range(-1,(-length-1),-1):
5.         print name[i],"\t",name[n]
6.         i+=1
```

Output:

```
>>>
R      t
a      a
j      j
a      a
t      R
>>>
```

STRINGS OPERATORS

There are basically 3 types of Operators supported by String:

1. Basic Operators.
2. Membership Operators.
3. Relational Operators.

BASIC OPERATORS:

There are two types of basic operators in String. They are "+" and "*".

STRING CONCATENATION OPERATOR :(+)

The concatenation operator (+) concatenate two Strings and forms a new String.

eg:

```
>>> "ratan" + "jaiswal"
```

Output:

```
'ratanjaiswal'
>>>
```

Expression	Output
'10' + '20'	'1020'
"s" + "007"	's007'
'abcd123' + 'xyz4'	'abcd123xyz4'

NOTE: Both the operands passed for concatenation must be of same type, else it will show an error.

Eg:

```
'abc' + 3
>>>
```

output:

```
Traceback (most recent call last):
  File "", line 1, in
    'abc' + 3
TypeError: cannot concatenate 'str' and 'int' objects
>>>
```

REPLICATION OPERATOR: (*)

Replication operator uses two parameter for operation. One is the integer value and the other one is the String.

The Replication operator is used to repeat a string number of times. The string will be repeated the number of times which is given by the integer value.

Eg:

1. >>> 5*"Vimal"

Output:

```
'VimalVimalVimalVimalVimal'
```

Expression	Output
"soono"*2	'soonosoono'
3*'1'	'111'
'\$'*5	'\$\$\$\$\$'

NOTE: We can use Replication operator in any way i.e., int * string or string * int. Both the parameters passed cannot be of same type.

MEMBERSHIP OPERATORS

Membership Operators are already discussed in the Operators section. Let see with context of String.

There are two types of Membership operators:

1) in: "in" operator return true if a character or the entire substring is present in the specified string, otherwise false.

2) not in: "not in" operator return true if a character or entire substring does not exist in the specified string, otherwise false.

Eg:

```

1.      >>> str1="javatpoint"
2.      >>> str2='sssit'
3.      >>> str3="seomount"
4.      >>> str4='java'
5.      >>> st5="it"
6.      >>> str6="seo"
7.      >>> str4 in str1
8.      True
9.      >>> str5 in str2
10.     >>> st5 in str2
11.     True
12.     >>> str6 in str3

```


13. True
14. >>> str4 **not in** str1
15. False
16. >>> str1 **not in** str4
17. True

RELATIONAL OPERATORS:

All the comparison operators i.e., (<,>,<=,>=,==,!=,<>) are also applicable to strings. The Strings are compared based on the ASCII value or Unicode(i.e., dictionary Order).

Eg:

1. >>> "RAJAT"=="RAJAT"
2. True
3. >>> "afsha">='Afsha'
4. True
5. >>> "Z"<>"z"
6. True

Explanation:

The ASCII value of a is 97, b is 98, c is 99 and so on. The ASCII value of A is 65,B is 66,C is 67 and so on. The comparison between strings are done on the basis on ASCII value.

SLICE NOTATION:

String slice can be defined as substring which is the part of string. Therefore further substring can be obtained from a string.

There can be many forms to slice a string. As string can be accessed or indexed from both the direction and hence string can also be sliced from both the direction that is left and right.

Syntax:

1. <string_name>[startIndex:endIndex],
2. <string_name>[:endIndex],
3. <string_name>[startIndex:]

Example:

1. >>> str="Nikhil"

```
2.      >>> str[0:6]
3.      'Nikhil'
4.      >>> str[0:3]
5.      'Nik'
6.      >>> str[2:5]
7.      'khi'
8.      >>> str[:6]
9.      'Nikhil'
10.     >>> str[3:]
11.     'hil'
```

Note: startIndex in String slice is inclusive whereas endIndex is exclusive.

String slice can also be used with Concatenation operator to get whole string.

Eg:

```
1.      >>> str="Mahesh"
2.      >>> str[:6]+str[6:]
3.      'Mahesh'
```

//here 6 is the length of the string.

STRING FUNCTIONS AND METHODS:

There are many predefined or built in functions in String. They are as follows:

capitalize()	It capitalizes the first character of the String.
count(string,begin,end)	Counts number of times substring occurs in a String between begin and end.
endswith(suffix,begin=0,end=n)	Returns a Boolean value if the string terminates with given suffix.
find(substring,beginIndex,endIndex)	It returns the index value of the string where substring is found between begin and end index.

index(substring, beginIndex, endIndex)	Same as find() except it raises an exception if string is not found.
isalnum()	It returns True if characters in the string are alphanumeric i.e., al and there is at least 1 character. Otherwise it returns False.
isalpha()	It returns True when all the characters are alphabets and the character, otherwise False.
isdigit()	It returns True if all the characters are digit and there is at least 1 character, otherwise False.
islower()	It returns True if the characters of a string are in lower case, otherwise False.
isupper()	It returns True if characters of a string are in Upper case, otherwise False.
isspace()	It returns True if the characters of a string are whitespace, otherwise False.
len(string)	len() returns the length of a string.
lower()	Converts all the characters of a string to Lower case.
upper()	Converts all the characters of a string to Upper Case.
startswith(str ,begin=0,end=n)	Returns a Boolean value if the string starts with given str between the given range.
swapcase()	Inverts case of all characters in a string.
lstrip()	Remove all leading whitespace of a string. It can also be used to remove a specific character from leading.
rstrip()	Remove all trailing whitespace of a string. It can also be used to remove a specific character from trailing.

Examples:

1) capitalize()

```
1.      >>> 'abc'.capitalize()
```

Output:

```
'Abc '
```

2) count(string)

```
1.      msg = "welcome to sssit";
2.      substr1 = "o";
3.      print msg.count(substr1, 4, 16)
4.      substr2 = "t";
5.      print msg.count(substr2)
```

Output:

```
>>>
2
2
>>>
```

3) endswith(string)

```
1.      string1="Welcome to SSSIT";
2.      substring1="SSSIT";
3.      substring2="to";
4.      substring3="of";
5.      print string1.endswith(substring1);
6.      print string1.endswith(substring2,2,16);
7.      print string1.endswith(substring3,2,19);
8.      print string1.endswith(substring3);
```

Output:

```
>>>
True
False
False
False
>>>
```

4) find(string)

```

1.     str="Welcome to SSSIT";
2.     substr1="come";
3.     substr2="to";
4.     print str.find(substr1);
5.     print str.find(substr2);
6.     print str.find(substr1,3,10);
7.     print str.find(substr2,19);

```

Output:

```

>>>
3
8
3
-1
>>>

```

5) index(string)

```

1.     str="Welcome to world of SSSIT";
2.     substr1="come";
3.     substr2="of";
4.     print str.index(substr1);
5.     print str.index(substr2);
6.     print str.index(substr1,3,10);
7.     print str.index(substr2,19);

```

Output:

```

>>>
3
17
3
Traceback (most recent call last):
  File "C:/Python27/fin.py", line 7, in
    print str.index(substr2,19);
ValueError: substring not found
>>>

```

6) isalnum()

```

1.     str="Welcome to sssit";
2.     print str.isalnum();
3.     str1="Python47";
4.     print str1.isalnum();

```

Output:

```
>>>
False
True
>>>
```

7) isalpha()

1. string1="HelloPython"; # Even space is not allowed
2. **print** string1.isalpha();
3. string2="This is Python2.7.4"
4. **print** string2.isalpha();

Output:

```
>>>
True
False
>>>
```

8) isdigit()

1. string1="HelloPython";
2. **print** string1.isdigit();
3. string2="98564738"
4. **print** string2.isdigit();

Output:

```
>>>
False
True
>>>
```

9) islower()

1. string1="Hello Python";
2. **print** string1.islower();
3. string2="welcome to "
4. **print** string2.islower();

Output:

```
>>>
False
True
```

```
>>>
```

10) isupper()

```
1.      string1="Hello Python";
2.      print string1.isupper();
3.      string2="WELCOME TO"
4.      print string2.isupper();
```

Output:

```
>>>
False
True
>>>
```

11) isspace()

```
1.      string1="  ";
2.      print string1.isspace();
3.      string2="WELCOME TO WORLD OF PYT"
4.      print string2.isspace();
```

Output:

```
>>>
True
False
>>>
```

12) len(string)

```
1.      string1="  ";
2.      print len(string1);
3.      string2="WELCOME TO SSSIT"
4.      print len(string2);
```

Output:

```
>>>
4
16
>>>
```

13) lower()

```
1.      string1="Hello Python";
2.      print string1.lower();
3.      string2="WELCOME TO SSSIT"
4.      print string2.lower();
```

Output:

```
>>>
hello python
welcome to sssit
>>>
```

14) upper()

```
1.      string1="Hello Python";
2.      print string1.upper();
3.      string2="welcome to SSSIT"
4.      print string2.upper();
```

Output:

```
>>>
HELLO PYTHON
WELCOME TO SSSIT
>>>
```

15) startswith(string)

```
1.      string1="Hello Python";
2.      print string1.startswith('Hello');
3.      string2="welcome to SSSIT"
4.      print string2.startswith('come',3,7);
```

Output:

```
>>>
True
True
>>>
```

16) swapcase()

```
1.      string1="Hello Python";
2.      print string1.swapcase();
3.      string2="welcome to SSSIT"
```


4. `print string2.swapcase();`

Output:

```
>>>
hELLO pYTHON
WELCOME TO sssit
>>>
```

17) lstrip()

```
1. string1=" Hello Python";
2. print string1.lstrip();
3. string2="@@@@@@@welcome to SSSIT"
4. print string2.lstrip('@');
```

Output:

```
>>>
Hello Python
welcome to world to SSSIT
>>>
```

18) rstrip()

```
1. string1=" Hello Python ";
2. print string1.rstrip();
3. string2="@welcome to SSSIT!!!"
4. print string2.rstrip('!');
```

Output:

```
>>>
Hello Python
@welcome to SSSIT
>>>
```

PYTHON LIST

1).Python lists are the data structure that is capable of holding different type of data.

2).Python lists are mutable i.e., Python will not create a new list if we modify an element in the list.

3).It is a container that holds other objects in a given order. Different operation like insertion and deletion can be performed on lists.

4).A list can be composed by storing a sequence of different type of values separated by commas.

5).A python list is enclosed between square([]) brackets.

6).The elements are stored in the index basis with starting index as 0.

eg:

```
1.      data1=[1,2,3,4];
2.      data2=['x','y','z'];
3.      data3=[12.5,11.6];
4.      data4=['raman','rahul'];
5.      data5=[];
6.      data6=['abhinav',10,56.4,'a'];
```

ACCESSING LISTS

A list can be created by putting the value inside the square bracket and separated by comma.

Syntax:

```
1.      <list_name>=[value1,value2,value3,...,valuen];
```

For accessing list :

```
1.      <list_name>[index]
```

Different ways to access list:

Eg:

```
1.      data1=[1,2,3,4];
2.      data2=['x','y','z'];
3.      print data1[0]
4.      print data1[0:2]
5.      print data2[-3:-1]
6.      print data1[0:]
7.      print data2[:2]
```

Output:

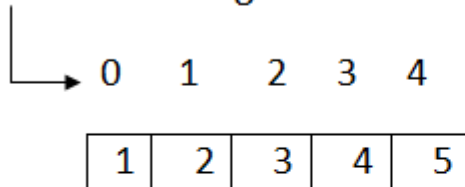
```
>>>
```

```
>>>
1
[1, 2]
['x', 'y']
[1, 2, 3, 4]
['x', 'y']
>>>
```

ELEMENTS IN A LISTS:

1. Data=[1,2,3,4,5];

Forward indexing



-5 -4 -3 -2 -1

Backward indexing

javatpoint.com

1. Data[0]=1=Data[-5] , Data[1]=2=Data[-4] , Data[2]=3=Data[-3] ,
2. =4=Data[-2] , Data[4]=5=Data[-1].

Note: Internal Memory Organization:
List do not store the elements directly at the index. In fact a reference is stored at each index which subsequently refers to the object stored somewhere in the memory. This is due to the fact that some objects may be large enough than other objects and hence they are stored at some other memory location.

LIST OPERATIONS:

Various Operations can be performed on List. Operations performed on List are given as:

a) Adding Lists:

Lists can be added by using the concatenation operator(+) to join two lists.

Eg:

1. list1=[10,20]
2. list2=[30,40]

```
3.         list3=list1+list2
4.         print list3
```

Output:

```
1.         >>>
2.         [10, 20, 30, 40]
3.         >>>
```

Note: '+' operator implies that both the operands passed must be list else error will be shown.

Eg:

```
1.         list1=[10,20]
2.         list1+30
3.         print list1
```

Output:

```
1.         Traceback (most recent call last):
2.             File "C:/Python27/lis.py", line 2, in <module>
3.                 list1+30
```

b) Replicating lists:

Replicating means repeating . It can be performed by using '*' operator by a specific number of time.

Eg:

```
1.         list1=[10,20]
2.         print list1*1
```

Output:

```
1.         >>>
2.         [10, 20]
3.         >>>
```

c) List slicing:

A subpart of a list can be retrieved on the basis of index. This subpart is known as list slice.

Eg:

```
1. list1=[1,2,4,5,7]
2. print list1[0:2]
3. print list1[4]
4. list1[1]=9
5. print list1
```

Output:

```
1. >>>
2. [1, 2]
3. 7
4. [1, 9, 4, 5, 7]
5. >>>
```

Note: If the index provided in the list slice is outside the list, then it raises an IndexError exception.

OTHER OPERATIONS:

Apart from above operations various other functions can also be performed on List such as Updating, Appending and Deleting elements from a List:

a) Updating elements in a List:

To update or change the value of particular index of a list, assign the value to that particular index of the List.

Syntax:

```
1. <list_name>[index]=<value>
```

Eg:

```
1. data1=[5,10,15,20,25]
2. print "Values of list are: "
3. print data1
4. data1[2]="Multiple of 5"
5. print "Values of list are: "
6. print data1
```

Output:

```
1.      >>>
2.      Values of list are:
3.      [5, 10, 15, 20, 25]
4.      Values of list are:
5.      [5, 10, 'Multiple of 5', 20, 25]
6.      >>>
```

b) Appending elements to a List:

append() method is used to append i.e., add an element at the end of the existing elements.

Syntax:

```
1.      <list_name>.append(item)
```

Eg:

```
1.      list1=[10,"rahul",'z']
2.      print "Elements of List are: "
3.      print list1
4.      list1.append(10.45)
5.      print "List after appending: "
6.      print list1
```

Output:

```
1.      >>>
2.      Elements of List are:
3.      [10, 'rahul', 'z']
4.      List after appending:
5.      [10, 'rahul', 'z', 10.45]
6.      >>>
```

c) Deleting Elements from a List:

del statement can be used to delete an element from the list. It can also be used to delete all items from startIndex to endIndex.

Eg:

```
1. list1=[10,'rahul',50.8,'a',20,30]
2. print list1
3. del list1[0]
4. print list1
5. del list1[0:3]
6. print list1
```

Output:

```
1. >>>
2. [10, 'rahul', 50.8, 'a', 20, 30]
3. ['rahul', 50.8, 'a', 20, 30]
4. [20, 30]
5. >>>
```

FUNCTIONS AND METHODS OF LISTS:

There are many Built-in functions and methods for Lists. They are as follows:

There are following List functions:

Function	Description
min(list)	Returns the minimum value from the list given.
max(list)	Returns the largest value from the given list.
len(list)	Returns number of elements in a list.
cmp(list1,list2)	Compares the two list.
list(sequence)	Takes sequence types and converts them to lists.

1) min(list):

Eg:

```
1. list1=[101,981,'abcd','xyz','m']
```

```
2. list2=['aman','shekhar',100.45,98.2]
3. print "Minimum value in List1: ",min(list1)
4. print "Minimum value in List2: ",min(list2)
```

Output:

```
1. >>>
2. Minimum value in List1: 101
3. Minimum value in List2: 98.2
4. >>>
```

2) max(list):

Eg:

```
1. list1=[101,981,'abcd','xyz','m']
2. list2=['aman','shekhar',100.45,98.2]
3. print "Maximum value in List : ",max(list1)
4. print "Maximum value in List : ",max(list2)
```

Output:

```
1. >>>
2. Maximum value in List : xyz
3. Maximum value in List : shekhar
4. >>>
```

3) len(list):

Eg:

```
1. list1=[101,981,'abcd','xyz','m']
2. list2=['aman','shekhar',100.45,98.2]
3. print "No. of elements in List1: ",len(list1)
4. print "No. of elements in List2: ",len(list2)
```

Output:

```
1. >>>
2. No. of elements in List1 : 5
3. No. of elements in List2 : 4
```


4. >>>

4) cmp(list1,list2):

Explanation: If elements are of the same type, perform the comparison and return the result. If elements are different types, check whether they are numbers.

- If numbers, perform comparison.
- If either element is a number, then the other element is returned.
- Otherwise, types are sorted alphabetically .

If we reached the end of one of the lists, the longer list is "larger." If both list are same it returns 0.

Eg:

```
1. list1=[101,981,'abcd','xyz','m']
2. list2=['aman','shekhar',100.45,98.2]
3. list3=[101,981,'abcd','xyz','m']
4. print cmp(list1,list2)
5. print cmp(list2,list1)
6. print cmp(list3,list1)
```

Output:

```
1. >>>
2. -1
3. 1
4. 0
5. >>>
```

5) list(sequence):

Eg:

```
1. seq=(145,"abcd",'a')
2. data=list(seq)
3. print "List formed is : ",data
```

Output:

```
1. >>>
```

2. List formed **is** : [145, 'abcd', 'a']
3. >>>

There are following built-in methods of List:

Methods	Description
index(object)	Returns the index value of the object.
count(object)	It returns the number of times an object is repeated in list.
pop()/pop(index)	Returns the last object or the specified indexed object. It removes the popped object.
insert(index,object)	Insert an object at the given index.
extend(sequence)	It adds the sequence to existing list.
remove(object)	It removes the object from the given List.
reverse()	Reverse the position of all the elements of a list.
sort()	It is used to sort the elements of the List.

1) index(object):

Eg:

1. data = [786,'abc','a',123.5]
2. **print** "Index of 123.5:", data.index(123.5)
3. **print** "Index of a is", data.index('a')

Output:

1. >>>
2. Index of 123.5 : 3
3. Index of a **is** 2
4. >>>

2) count(object):

Eg:

```
1. data = [786,'abc','a',123.5,786,'rahul','b',786]
2. print "Number of times 123.5 occurred is", data.count(123.5)
3. print "Number of times 786 occurred is", data.count(786)
```

Output:

```
1. >>>
2. Number of times 123.5 occurred is 1
3. Number of times 786 occurred is 3
4. >>>
```

3) pop()/pop(int):

Eg:

```
1. data = [786,'abc','a',123.5,786]
2. print "Last element is", data.pop()
3. print "2nd position element:", data.pop(1)
4. print data
```

Output:

```
1. >>>
2. Last element is 786
3. 2nd position element:abc
4. [786, 'a', 123.5]
5. >>>
```

4) insert(index,object):

Eg:

```
1. data=['abc',123,10.5,'a']
2. data.insert(2,'hello')
3. print data
```

Output:

```
1.      >>>
2.      ['abc', 123, 'hello', 10.5, 'a']
3.      >>>
```

5) extend(sequence):

Eg:

```
1.      data1=['abc',123,10.5,'a']
2.      data2=['ram',541]
3.      data1.extend(data2)
4.      print data1
5.      print data2
```

Output:

```
1.      >>>
2.      ['abc', 123, 10.5, 'a', 'ram', 541]
3.      ['ram', 541]
4.      >>>
```

6) remove(object):

Eg:

```
1.      data1=['abc',123,10.5,'a','xyz']
2.      data2=['ram',541]
3.      print data1
4.      data1.remove('xyz')
5.      print data1
6.      print data2
7.      data2.remove('ram')
8.      print data2
```

Output:

```
1.      >>>
2.      ['abc', 123, 10.5, 'a', 'xyz']
3.      ['abc', 123, 10.5, 'a']
4.      ['ram', 541]
5.      [541]
```

6. >>>

7) reverse():

Eg:

```
1. list1=[10,20,30,40,50]
2. list1.reverse()
3. print list1
```

Output:

```
1. >>>
2. [50, 40, 30, 20, 10]
3. >>>
```

8) sort():

Eg:

```
1. list1=[10,50,13,'rahul','aakash']
2. list1.sort()
3. print list1
```

Output:

```
1. >>>
2. [10, 13, 50, 'aakash', 'rahul']
3. >>>
```

PYTHON TUPLE

A tuple is a sequence of immutable objects, therefore tuple cannot be changed.

The objects are enclosed within parenthesis and separated by comma.

Tuple is similar to list. Only the difference is that list is enclosed between square bracket, tuple between parenthesis and List have mutable objects whereas Tuple have immutable objects.

eg:

```
1. >>> data=(10,20,'ram',56.8)
```

```
2.      >>> data2="a",10,20.9
3.      >>> data
4.      (10, 20, 'ram', 56.8)
5.      >>> data2
6.      ('a', 10, 20.9)
7.      >>>
```

NOTE: If Parenthesis is not given with a sequence, it is by default treated as Tuple.

There can be an empty Tuple also which contains no object.

eg:

```
1.      tuple1=()
```

For a single valued tuple, there must be a comma at the end of the value.

eg:

```
1.      Tuple1=(10,)
```

Tuples can also be nested.

eg:

```
1.      tupl1='a','mahesh',10.56
2.      tupl2=tupl1,(10,20,30)
3.      print tupl1
4.      print tupl2
```

Output:

```
1.      >>>
2.      ('a', 'mahesh', 10.56)
3.      (('a', 'mahesh', 10.56), (10, 20, 30))
4.      >>>
```

ACCESSING TUPLE

Tuple can be accessed in the same way as List.

Some examples are given below:

eg:

```
1. data1=(1,2,3,4)
2. data2=('x','y','z')
3. print data1[0]
4. print data1[0:2]
5. print data2[-3:-1]
6. print data1[0:]
7. print data2[:2]
```

Output:

```
1. >>>
2. 1
3. (1, 2)
4. ('x', 'y')
5. (1, 2, 3, 4)
6. ('x', 'y')
7. >>>
```

ELEMENTS IN A TUPLE

Data=(1,2,3,4,5,10,19,17)

Forward indexing

0 1 2 3 4 5 6 7

1	2	3	4	5	10	19	17
---	---	---	---	---	----	----	----

-8 -7 -6 -5 -4 -3 -2 -1

javatpoint.com

Backward index

```
1. Data[0]=1=Data[-8] , Data[1]=2=Data[-7] , Data[2]=3=Data[-6] ,
2. Data[3]=4=Data[-5] , Data[4]=5=Data[-4] , Data[5]=10=Data[-3],
3. Data[6]=19=Data[-2],Data[7]=17=Data[-1]
```

TUPLE OPERATIONS

Various Operations can be performed on Tuple. Operations performed on Tuple are given as:

a) Adding Tuple:

Tuple can be added by using the concatenation operator(+) to join two tuples.

eg:

1. data1=(1,2,3,4)
2. data2=('x','y','z')
3. data3=data1+data2
4. **print** data1
5. **print** data2
6. **print** data3

Output:

```
>>>
(1, 2, 3, 4)
('x', 'y', 'z')
(1, 2, 3, 4, 'x', 'y', 'z')
>>>
```

Note: The new sequence formed is a new Tuple.

b) Replicating Tuple:

Replicating means repeating. It can be performed by using '*' operator by a specific number of time.

Eg:

1. tuple1=(10,20,30);
2. tuple2=(40,50,60);
3. **print** tuple1*2
4. **print** tuple2*3

Output:

```
>>>
(10, 20, 30, 10, 20, 30)
(40, 50, 60, 40, 50, 60, 40, 50, 60)
>>>
```


c) Tuple slicing:

A subpart of a tuple can be retrieved on the basis of index. This subpart is known as tuple slice.

Eg:

1. data1=(1,2,4,5,7)
2. `print data1[0:2]`
3. `print data1[4]`
4. `print data1[:-1]`
5. `print data1[-5:]`
6. `print data1`

Output:

```
>>>
(1, 2)
7
(1, 2, 4, 5)
(1, 2, 4, 5, 7)
(1, 2, 4, 5, 7)
>>>
```

Note: If the index provided in the Tuple slice is outside the list, then it raises an `IndexError` exception.

OTHER OPERATIONS:

a) Updating elements in a List:

Elements of the Tuple cannot be updated. This is due to the fact that Tuples are immutable. Whereas the Tuple can be used to form a new Tuple.

Eg:

1. data=(10,20,30)
2. data[0]=100
3. `print data`

Output:

```
>>>
Traceback (most recent call last):
  File "C:/Python27/t.py", line 2, in
```

```
data[0]=100
TypeError: 'tuple' object does not support item assignment
>>>
```

Creating a new Tuple from existing:

Eg:

1. data1=(10,20,30)
2. data2=(40,50,60)
3. data3=data1+data2
4. **print** data3

Output:

```
>>>
(10, 20, 30, 40, 50, 60)
>>>
```

b) Deleting elements from Tuple:

Deleting individual element from a tuple is not supported. However the whole of the tuple can be deleted using the del statement.

Eg:

1. data=(10,20,'rahul',40.6,'z')
2. **print** data
3. **del** data #will delete the tuple data
4. **print** data #will show an error since tuple data is already deleted

Output:

```
>>>
(10, 20, 'rahul', 40.6, 'z')
Traceback (most recent call last):
  File "C:/Python27/t.py", line 4, in
    print data
NameError: name 'data' is not defined
>>>
```

FUNCTIONS OF TUPLE:

There are following in-built Type Functions:

Function	Description
min(tuple)	Returns the minimum value from a tuple.
max(tuple)	Returns the maximum value from the tuple.
len(tuple)	Gives the length of a tuple
cmp(tuple1,tuple2)	Compares the two Tuples.
tuple(sequence)	Converts the sequence into tuple.

1) min(tuple):

Eg:

1. data=(10,20,'rahul',40.6,'z')
2. **print** min(data)

Output:

```
>>>
10
>>>
```

2) max(tuple):

Eg:

1. data=(10,20,'rahul',40.6,'z')
2. **print** max(data)

Output:

```
>>>
z
>>>
```

3) len(tuple):

Eg:

1. data=(10,20,'rahul',40.6,'z')
2. **print** len(data)

Output:

```
>>>
5
>>>
```

4) cmp(tuple1,tuple2):

Explanation: If elements are of the same type, perform the comparison and return the result. If elements are different types, check whether they are numbers.

- If numbers, perform comparison.
- If either element is a number, then the other element is returned.
- Otherwise, types are sorted alphabetically .

If we reached the end of one of the lists, the longer list is "larger." If both list are same it returns 0.

Eg:

1. data1=(10,20,'rahul',40.6,'z')
2. data2=(20,30,'sachin',50.2)
3. **print** cmp(data1,data2)
4. **print** cmp(data2,data1)
5. data3=(20,30,'sachin',50.2)
6. **print** cmp(data2,data3)

Output:

```
>>>
-1
1
0
>>>
```

5) tuple(sequence):

Eg:

1. dat=[10,20,30,40]
2. data=tuple(dat)
3. **print** data

Output:

```
>>>
(10, 20, 30, 40)
>>>
```

WHY USE TUPLE?

1. Processing of Tuples are faster than Lists.
2. It makes the data safe as Tuples are immutable and hence cannot be changed.
3. Tuples are used for String formatting.

PYTHON DICTIONARY

Dictionary is an unordered set of key and value pair.

It is an container that contains data, enclosed within curly braces.

The pair i.e., key and value is known as item.

The key passed in the item must be unique.

The key and the value is separated by a colon(:). This pair is known as item. Items are separated from each other by a comma(,). Different items are enclosed within a curly brace and this forms Dictionary.

eg:

1. data={100:'Ravi' ,101:'Vijay' ,102:'Rahul'}
2. **print** data

Output:

1. >>>
2. {100: 'Ravi', 101: 'Vijay', 102: 'Rahul'}
3. >>>

Dictionary is mutable i.e., value can be updated.

Key must be unique and immutable. Value is accessed by key. Value can be updated while key cannot be changed.

Dictionary is known as Associative array since the Key works as Index and they are decided by the user.

eg:

```
1.     plant={}
2.     plant[1]='Ravi'
3.     plant[2]='Manoj'
4.     plant['name']='Hari'
5.     plant[4]='Om'
6.     print plant[2]
7.     print plant['name']
8.     print plant[1]
9.     print plant
```

Output:

```
>>>
Manoj
Hari
Ravi
{1: 'Ravi', 2: 'Manoj', 4: 'Om', 'name': 'Hari'}
>>>
```

ACCESSING VALUES

Since Index is not defined, a Dictionaries value can be accessed by their keys.

Syntax:

```
[key]
```

Eg:

```
data1={'Id':100, 'Name':'Suresh', 'Profession':'Developer'}
data2={'Id':101, 'Name':'Ramesh', 'Profession':'Trainer'}
print "Id of 1st employer is",data1['Id']
print "Id of 2nd employer is",data2['Id']
print "Name of 1st employer:",data1['Name']
print "Profession of 2nd employer:",data2['Profession']
```

Output:

```
>>>
Id of 1st employer is 100
Id of 2nd employer is 101
Name of 1st employer is Suresh
Profession of 2nd employer is Trainer
>>>
```

UPDATION

The item i.e., key-value pair can be updated. Updating means new item can be added. The values can be modified.

Eg:

```
data1={'Id':100, 'Name':'Suresh', 'Profession':'Developer'}
data2={'Id':101, 'Name':'Ramesh', 'Profession':'Trainer'}
data1['Profession']='Manager'
data2['Salary']=20000
data1['Salary']=15000
print data1
print data2
```

Output:

```
>>>
{'Salary': 15000, 'Profession': 'Manager', 'Id': 100, 'Name': 'Suresh'}
{'Salary': 20000, 'Profession': 'Trainer', 'Id': 101, 'Name': 'Ramesh'}
>>>
```

DELETION

del statement is used for performing deletion operation.

An item can be deleted from a dictionary using the key.

Syntax:

```
del [key]
```

Whole of the dictionary can also be deleted using the del statement.

Eg:

```
data={100:'Ram', 101:'Suraj', 102:'Alok'}
del data[102]
print data
del data
print data    #will show an error since dictionary is deleted.
```

Output:

```
>>>
{100: 'Ram', 101: 'Suraj'}

Traceback (most recent call last):
  File "C:/Python27/dict.py", line 5, in
    print data
NameError: name 'data' is not defined
>>>
```

FUNCTIONS AND METHODS

Python Dictionary supports the following Functions:

Dictionary Functions:

Functions	Description
<code>len(dictionary)</code>	Gives number of items in a dictionary.
<code>cmp(dictionary1,dictionary2)</code>	Compares the two dictionaries.
<code>str(dictionary)</code>	Gives the string representation of a string.

Dictionary Methods:

Methods	Description
<code>keys()</code>	Return all the keys element of a dictionary.
<code>values()</code>	Return all the values element of a dictionary.
<code>items()</code>	Return all the items(key-value pair) of a dictionary.
<code>update(dictionary2)</code>	It is used to add items of dictionary2 to first dictionary.
<code>clear()</code>	It is used to remove all items of a dictionary. It returns an empty dictionary.
<code>fromkeys(sequence,value1)/ fromkeys(sequence)</code>	It is used to create a new dictionary from the sequence where each element forms the key and all keys share the values ?value1?. In case if value1 is not provided, it set the values of keys to be none.
<code>copy()</code>	It returns an ordered copy of the data.
<code>has_key(key)</code>	It returns a boolean value. True in case if key is present in dictionary, else false.

get(key)	Returns the value of the given key. If key is not present it returns None.
----------	--

FUNCTIONS:

1) len(dictionary):

Eg:

```
data={100:'Ram', 101:'Suraj', 102:'Alok'}
print data
print len(data)
```

Output:

```
>>>
{100: 'Ram', 101: 'Suraj', 102: 'Alok'}
3
>>>
```

2) cmp(dictionary1,dictionary2):

Explanation:

The comparison is done on the basis of key and value.

```
If, dictionary1 == dictionary2, returns 0.
    dictionary1 < dictionary2, returns -1.
    dictionary1 > dictionary2, returns 1.
```

Eg:

```
data1={100:'Ram', 101:'Suraj', 102:'Alok'}
data2={103:'abc', 104:'xyz', 105:'mno'}
data3={'Id':10, 'First':'Aman','Second':'Sharma'}
data4={100:'Ram', 101:'Suraj', 102:'Alok'}
print cmp(data1,data2)
print cmp(data1,data4)
print cmp(data3,data2)
```

Output:

```
>>>
-1
0
1
>>>
```

3) str(dictionary):

Eg:

```
data1={100:'Ram', 101:'Suraj', 102:'Alok'}  
print str(data1)
```

Output:

```
>>>  
{100: 'Ram', 101: 'Suraj', 102: 'Alok'}  
>>>
```

METHODS:

1) keys():

Eg:

```
data1={100:'Ram', 101:'Suraj', 102:'Alok'}  
print data1.keys()
```

Output:

```
>>>  
[100, 101, 102]  
>>>
```

2) values():

Eg:

```
data1={100:'Ram', 101:'Suraj', 102:'Alok'}  
print data1.values()
```

Output:

```
>>>  
['Ram', 'Suraj', 'Alok']  
>>>
```

3) items():

Eg:

```
data1={100:'Ram', 101:'Suraj', 102:'Alok'}  
print data1.items()
```

Output:

```
>>>  
[(100, 'Ram'), (101, 'Suraj'), (102, 'Alok')]  
>>>
```

4) update(dictionary2):

Eg:

```
data1={100:'Ram', 101:'Suraj', 102:'Alok'}
data2={103:'Sanjay'}
data1.update(data2)
print data1
print data2
```

Output:

```
>>>
{100: 'Ram', 101: 'Suraj', 102: 'Alok', 103: 'Sanjay'}
{103: 'Sanjay'}
>>>
```

5) clear():

Eg:

```
data1={100:'Ram', 101:'Suraj', 102:'Alok'}
print data1
data1.clear()
print data1
```

Output:

```
>>>
{100: 'Ram', 101: 'Suraj', 102: 'Alok'}
{}
>>>
```

6) fromkeys(sequence)/ fromkeys(seq,value):

Eg:

```
sequence=('Id' , 'Number' , 'Email')
data={}
data1={}
data=data.fromkeys(sequence)
print data
data1=data1.fromkeys(sequence,100)
print data1
```

Output:

```
>>>
{'Email': None, 'Id': None, 'Number': None}
{'Email': 100, 'Id': 100, 'Number': 100}
>>>
```

7) copy():

Eg:

```
data={'Id':100 , 'Name':'Aakash' , 'Age':23}
data1=data.copy()
print data1
```

Output:

```
>>>
{'Age': 23, 'Id': 100, 'Name': 'Aakash'}
>>>
```

8) has_key(key):

Eg:

```
data={'Id':100 , 'Name':'Aakash' , 'Age':23}
print data.has_key('Age')
print data.has_key('Email')
```

Output:

```
>>>
True
False
>>>
```

9) get(key):

Eg:

```
data={'Id':100 , 'Name':'Aakash' , 'Age':23}
print data.get('Age')
print data.get('Email')
```

Output:

```
>>>
23
None
>>>
```

PYTHON INPUT AND OUTPUT

Python can be used to read and write data. Also it supports reading and writing data to Files.

"PRINT" STATEMENT:

"print" statement is used to print the output on the screen.

print statement is used to take string as input and place that string to standard output.

Whatever you want to display on output place that expression inside the inverted commas. The expression whose value is to be printed place it without inverted commas.

Syntax:

1. **print** "expression" or **print** expression.

eg:

1. a=10
2. **print** "Welcome to the world of Python"
3. **print** a

Output:

1. >>>
2. Welcome to the world of Python
3. 10
4. >>>

INPUT FROM KEYBOARD:

Python offers two in-built functions for taking input from user. They are:

1) input()

2) raw_input()

1) input() function input() function is used to take input from the user. Whatever expression is given by the user, it is evaluated and result is returned back.

Syntax:

1. input("Expression")

eg:

1. n=input("Enter your expression ");
2. **print** "The evaluated expression is ", n

Output:

1. >>>
2. Enter your expression 10*2
3. The evaluated expression **is** 20
4. >>>

2) raw_input() raw_input() function is used to take input from the user. It takes the input from the Standard input in the form of a string and reads the data from a line at once.

Syntax:

1. raw_input(?statement?)

eg:

1. n=raw_input("Enter your name ");
2. **print** "Welcome ", n

Output:

1. >>>
2. Enter your name Rajat
3. Welcome Rajat
4. >>>

raw_input() function returns a string. Hence in case an expression is to be evaluated, then it has to be type casted to its following data type. Some of the examples are given below:

Program to calculate Simple Interest.

1. prn=int(raw_input("Enter Principal"))
2. r=int(raw_input("Enter Rate"))
3. t=int(raw_input("Enter Time"))
4. si=(prn*r*t)/100
5. **print** "Simple Interest is ",si

Output:

1. >>>
2. Enter Principal1000
3. Enter Rate10
4. Enter Time2

```
5.      Simple Interest is 200
6.      >>>
```

Program to enter details of an user and print them.

```
1.      name=raw_input("Enter your name ")
2.      math=float(raw_input("Enter your marks in Math"))
3.      physics=float(raw_input("Enter your marks in Physics"))
4.      chemistry=float(raw_input("Enter your marks in Chemistry"))
5.      rollno=int(raw_input("Enter your Roll no"))
6.      print "Welcome ",name
7.      print "Your Roll no is ",rollno
8.      print "Marks in Maths is ",math
9.      print "Marks in Physics is ",physics
10.     print "Marks in Chemistry is ",chemistry
11.     print "Average marks is ",(math+physics+chemistry)/3
```

Output:

```
1.      >>>
2.      Enter your name rajat
3.      Enter your marks in Math76.8
4.      Enter your marks in Physics71.4
5.      Enter your marks in Chemistry88.4
6.      Enter your Roll no0987645672
7.      Welcome  rajat
8.      Your Roll no is  987645672
9.      Marks in Maths is  76.8
10.     Marks in Physics is  71.4
11.     Marks in Chemistry is  88.4
12.     Average marks is  78.8666666667
13.     >>>
```

FILE HANDLING:

Python provides the facility of working on Files. A File is an external storage on hard disk from where data can be stored and retrieved.

Operations on Files:

1) Opening a File: Before working with Files you have to open the File. To open a File, Python built in function `open()` is used. It returns an object of File which is used with other functions. Having opened the file now you can perform read, write, etc. operations on the File.

Syntax:

1. `obj=open(filename , mode , buffer)`

here,

filename:It is the name of the file which you want to access.

mode:It specifies the mode in which File is to be opened. There are many types of mode. Mode depends the operation to be performed on File. Default access mode is read.

2) Closing a File: Once you are finished with the operations on File at the end you need to close the file. It is done by the `close()` method. `close()` method is used to close a File.

Syntax:

1. `fileobject.close()`

3) Writing to a File:`write()` method is used to write a string into a file.

Syntax:

1. `fileobject.write(string str)`

4) Reading from a File:`read()` method is used to read data from the File.

Syntax:

1. `fileobject.read(value)`

here, value is the number of bytes to be read. In case, no value is given it reads till end of file is reached.

Program to read and write data from a file.

1. `obj=open("abcd.txt","w")`
2. `obj.write("Welcome to the world of Python")`
3. `obj.close()`
4. `obj1=open("abcd.txt","r")`
5. `s=obj1.read()`
6. `print s`


```
7.      obj1.close()
8.      obj2=open("abcd.txt","r")
9.      s1=obj2.read(20)
10.     print s1
11.     obj2.close()
```

Output:

```
1.      >>>
2.      Welcome to the world of Python
3.      Welcome to the world
4.      >>>
```

ATTRIBUTES OF FILE:

There are following File attributes.

Attribute	Description
Name	Returns the name of the file.
Mode	Returns the mode in which file is being opened.
Closed	Returns Boolean value. True, in case if file is closed else false.

Eg:

```
1.      obj = open("data.txt", "w")
2.      print obj.name
3.      print obj.mode
4.      print obj.closed
```

Output:

```
1.      >>>
2.      data.txt
3.      w
4.      False
```

5. >>>

MODES OF FILE:

There are different modes of file in which it can be opened. They are mentioned in the following table.

A File can be opened in two modes:

1) Text Mode.

2) Binary Mode.

Mode	Description
R	It opens in Reading mode. It is default mode of File. Pointer is at beginning of the file.
rb	It opens in Reading mode for binary format. It is the default mode. Pointer is at beginning of file.
r+	Opens file for reading and writing. Pointer is at beginning of file.
rb+	Opens file for reading and writing in binary format. Pointer is at beginning of file.
W	Opens file in Writing mode. If file already exists, then overwrite the file else create a new file.
wb	Opens file in Writing mode in binary format. If file already exists, then overwrite the file else create a new file.
w+	Opens file for reading and writing. If file already exists, then overwrite the file else create a new file.
wb+	Opens file for reading and writing in binary format. If file already exists, then overwrite the file else create a new file.
a	Opens file in Appending mode. If file already exists, then append the data at the end of existing file. If file does not exist, then create a new file.
ab	Opens file in Appending mode in binary format. If file already exists, then append the data at the end of existing file. If file does not exist, then create a new file.

	file, else create a new file.
a+	Opens file in reading and appending mode. If file already exists, then append the data at the end of existing file, else create a new file.
ab+	Opens file in reading and appending mode in binary format. If file already exists, then append the data at the end of existing file, else create a new file.

METHODS:

There are many methods related to File Handling. They are given in the following table:

There is a module "os" defined in Python that provides various functions which are used to perform various operations on Files. To use these functions 'os' needs to be imported.

Method	Description
rename()	It is used to rename a file. It takes two arguments, existing_file_name and new_file_name.
remove()	It is used to delete a file. It takes one argument. Pass the name of the file which is to be deleted as the argument of method.
mkdir()	It is used to create a directory. A directory contains the files. It takes one argument which is the name of the directory.
chdir()	It is used to change the current working directory. It takes one argument which is the name of the directory.
getcwd()	It gives the current working directory.
rmdir()	It is used to delete a directory. It takes one argument which is the name of the directory.
tell()	It is used to get the exact position in the file.

1) rename():

Syntax:

```
1.      os.rename(existing_file_name, new_file_name)
```

eg:

```
1.      import os
2.      os.rename('mno.txt', 'pqr.txt')
```

2) remove():

Syntax:

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

eg:

```
1.      import os
2.      os.remove('mno.txt')
```

3) mkdir()

Syntax:

```
os.mkdir("file_name")
```

eg:

```
1.      import os
2.      os.mkdir("new")
```

4) chdir()

Syntax:

```
os.chdir("file_name")
```

eg:

```
1.      import os
2.      os.chdir("new")
```

5) getcwd()

Syntax:

```
os.getcwd()
```

eg:

1. `import os`
2. `print os.getcwd()`

6) rmdir()**Syntax:**

```
os.rmdir("directory_name")
```

eg:

1. `import os`
2. `os.rmdir("new")`

NOTE: In order to delete a directory, it should be empty. In case directory is not empty first delete the files.

PYTHON MOUDULE

Modules are used to categorize code in Python into smaller part. A module is simply a file, where classes, functions and variables are defined. Grouping similar code into a single file makes it easy to access.

Have a look over example:

If the content of a book is not indexed or categorized into individual chapters, then the book might have turned boring and hectic. Hence, dividing book into chapters made it easy to understand.

In the same sense python modules are the files which have similar code. Thus module is simplify a python code where classes, variables and functions are defined.

Advantage:

Python provides the following advantages for using module:

1) Reusability: Module can be used in some other python code. Hence it provides the facility of code reusability.

2) Categorization: Similar type of attributes can be placed in one module.

IMPORTING A MODULE:

There are different ways by which you we can import a module. These are as follows:

1) USING IMPORT STATEMENT:

"import" statement can be used to import a module.

Syntax:

1. **import** <file_name1, file_name2,...file_name(n)="">
2. </file_name1,>

Have a look over an example:

eg:

1. **def** add(a,b):
2. c=a+b
3. **print** c
4. **return**

Save the file by the name addition.py. To import this file "import" statement is used.

1. **import** addition
2. addition.add(10,20)
3. addition.add(30,40)

Create another python file in which you want to import the former python file. For that, import statement is used as given in the above example. The corresponding method can be used by file_name.method (). (Here, addition. add (), where addition is the python file and add () is the method defined in the file addition.py)

Output:

1. >>>
2. 30
3. 70
4. >>>

NOTE: You can access any function which is inside a module by module name and function name separated by dot. It is also known as period. Whole notation is known as dot notation.

Example of importing multiple modules:

Eg:

1) msg.py:

```
1.      def msg_method():
2.          print "Today the weather is rainy"
3.          return
```

2) display.py:

```
1.      def display_method():
2.          print "The weather is Sunny"
3.          return
```

3) multiimport.py:

```
1.      import msg,display
2.      msg.msg_method()
3.      display.display_method()
```

Output:

```
1.      >>>
2.      Today the weather is rainy
3.      The weather is Sunny
4.      >>>
```

2) USING FROM.. IMPORT STATEMENT:

from..import statement is used to import particular attribute from a module. In case you do not want whole of the module to be imported then you can use from ?import statement.

Syntax:

```
1.      from <module_name> import <attribute1,attribute2,attribute3,...attributen>
2.      </attribute1,attribute2,attribute3,...attributen></module_name>
```

Have a look over the example:

1) area.py

Eg:

```
1.     def circle(r):
2.         print 3.14*r*r
3.         return
4.
5.     def square(l):
6.         print l*l
7.         return
8.
9.     def rectangle(l,b):
10.        print l*b
11.        return
12.
13.    def triangle(b,h):
14.        print 0.5*b*h
15.        return
```

2) area1.py

```
1.     from area import square,rectangle
2.     square(10)
3.     rectangle(2,5)
```

Output:

```
1.     >>>
2.     100
3.     10
4.     >>>
```

3) TO IMPORT WHOLE MODULE:

You can import whole of the module using "from? import *"

Syntax:

1. `from <module_name> import *`
2. `</module_name>`

Using the above statement all the attributes defined in the module will be imported and hence you can access each attribute.

1) area.py

Same as above example

2) area1.py

1. `from area import *`
2. `square(10)`
3. `rectangle(2,5)`
4. `circle(5)`
5. `triangle(10,20)`

Output:

1. `>>>`
2. `100`
3. `10`
4. `78.5`
5. `100.0`
6. `>>>`

BUILT IN MODULES IN PYTHON:

There are many built in modules in Python. Some of them are as follows:

math, random , threading , collections , os , mailbox , string , time , tkinter etc..

Each module has a number of built in functions which can be used to perform various functions.

Let's have a look over each module:

1) math:

Using math module , you can use different built in mathematical functions.

Functions:

Function	Description
ceil(n)	Returns the next integer number of the given number
sqrt(n)	Returns the Square root of the given number.
exp(n)	Returns the natural logarithm e raised to the given number
floor(n)	Returns the previous integer number of the given number.
log(n,baseto)	Returns the natural logarithm of the number.
pow(baseto, exp)	Returns baseto raised to the exp power.
sin(n)	Returns sine of the given radian.
cos(n)	Returns cosine of the given radian.
tan(n)	Returns tangent of the given radian.

Useful Example of math module:

Eg:

```

1.      import math
2.      a=4.6
3.      print math.ceil(a)
4.      print math.floor(a)
5.      b=9
6.      print math.sqrt(b)
7.      print math.exp(3.0)
8.      print math.log(2.0)
9.      print math.pow(2.0,3.0)
10.     print math.sin(0)
11.     print math.cos(0)
12.     print math.tan(45)

```

Output:

```
1.      >>>
2.      5.0
3.      4.0
4.      3.0
5.      20.0855369232
6.      0.69314718056
7.      8.0
8.      0.0
9.      1.0
10.     1.61977519054
11.     >>>
```

Constants:

The math module provides two constants for mathematical Operations:

Constants	Descriptions
Pi	Returns constant $\pi = 3.14159...$
ceil(n)	Returns constant $e = 2.71828...$

Eg:

```
1.      import math
2.
3.      print math.pi
4.      print math.e
```

Output:

```
1.      >>>
2.      3.14159265359
3.      2.71828182846
4.      >>>
```

2) random:

The random module is used to generate the random numbers. It provides the following two built in functions:

Function	Description
random()	It returns a random number between 0.0 and 1.0 where 1.0 is exclusive.
randint(x,y)	It returns a random number between x and y where both the numbers are inclusive

Eg:

```
1. import random
2.
3. print random.random()
4. print random.randint(2,8)
```

Output:

```
1. >>>
2. 0.797473843839
3. 7
4. >>>
```

Other modules will be covered in their respective topics.

PACKAGE

A Package is simply a collection of similar modules, sub-packages etc..

Steps to create and import Package:

- 1) Create a directory, say Info
- 2) Place different modules inside the directory. We are placing 3 modules msg1.py, msg2.py and msg3.py respectively and place corresponding codes in respective modules. Let us place msg1() in msg1.py, msg2() in msg2.py and msg3() in msg3.py.
- 3) Create a file __init__.py which specifies attributes in each module.
- 4) Import the package and use the attributes using package.

Have a look over the example:

1) Create the directory:

1. `import os`
2. `os.mkdir("Info")`

2) Place different modules in package: (Save different modules inside the Info package)

msg1.py

1. `def msg1():`
2. `print "This is msg1"`

msg2.py

1. `def msg2():`
2. `print "This is msg2"`

msg3.py

1. `def msg3():`
2. `print "This is msg3"`

3) Create `__init__.py` file:

1. `from msg1 import msg1`
2. `from msg2 import msg2`
3. `from msg3 import msg3`

4) Import package and use the attributes:

1. `import Info`
2. `Info.msg1()`
3. `Info.msg2()`
4. `Info.msg3()`

Output:

1. `>>>`
2. `This is msg1`
3. `This is msg2`

4. This **is** msg3
5. >>>

What is `__init__.py` file? `__init__.py` is simply a file that is used to consider the directories on the disk as the package of the Python. It is basically used to initialize the python packages.

EXCEPTION HANDLING

Exception can be said to be any abnormal condition in a program resulting to the disruption in the flow of the program.

Whenever an exception occurs the program halts the execution and thus further code is not executed. Thus exception is that error which python script is unable to tackle with.

Exception in a code can also be handled. In case it is not handled, then the code is not executed further and hence execution stops when exception occurs.

Hierarchy Of Exception:

1. ZeroDivisionError: Occurs when a number is divided by zero.
2. NameError: It occurs when a name is not found. It may be local or global.
3. IndentationError: If incorrect indentation is given.
4. IOError: It occurs when Input Output operation fails.
5. EOFError: It occurs when end of file is reached and yet operations are being performed

etc..

EXCEPTION HANDLING:

The suspicious code can be handled by using the try block. Enclose the code which raises an exception inside the try block. The try block is followed except statement. It is then further followed by statements which are executed during exception and in case if exception does not occur.

Syntax:

1. **try**:
2. malicious code
3. **except** Exception1:
4. execute code

```
5.     except Exception2:
6.         execute code
7.     ....
8.     ....
9.     except ExceptionN:
10.        execute code
11.     else:
12.        In case of no exception, execute the else block code.
```

eg:

```
1.     try:
2.         a=10/0
3.         print a
4.     except ArithmeticError:
5.         print "This statement is raising an exception"
6.     else:
7.         print "Welcome"
```

Output:

```
1.     >>>
2.     This statement is raising an exception
3.     >>>
```

Explanation:

1. The malicious code (code having exception) is enclosed in the try block.
2. Try block is followed by except statement. There can be multiple except statement with a single try block.
3. Except statement specifies the exception which occurred. In case that exception is occurred, the corresponding statement will be executed.
4. At the last you can provide else statement. It is executed when no exception is occurred.

EXCEPT WITH NO EXCEPTION:

Except statement can also be used without specifying Exception.

Syntax:

1. **try:**
2. code
3. **except:**
4. code to be executed **in** case exception occurs.
5. **else:**
6. code to be executed **in** case exception does **not** occur.

eg:

1. **try:**
2. a=10/0;
3. **except:**
4. **print** "Arithmetic Exception"
5. **else:**
6. **print** "Successfully Done"

Output:

1. >>>
2. Arithmetic Exception
3. >>>

DECLARING MULTIPLE EXCEPTION

Multiple Exceptions can be declared using the same except statement:

Syntax:

1. **try:**
2. code
3. **except** Exception1,Exception2,Exception3,...,ExceptionN
4. execute this code **in** case any Exception of these occur.
5. **else:**
6. execute code **in** case no exception occurred.

eg:

1. **try:**
2. a=10/0;


```
3.     except ArithmeticError,StandardError:
4.         print "Arithmetic Exception"
5.     else:
6.         print "Successfully Done"
```

Output:

```
1.     >>>
2.     Arithmetic Exception
3.     >>>
```

FINALLY BLOCK:

In case if there is any code which the user want to be executed, whether exception occurs or not then that code can be placed inside the finally block. Finally block will always be executed irrespective of the exception.

Syntax:

```
1.     try:
2.         Code
3.     finally:
4.         code which is must to be executed.
```

eg:

```
1.     try:
2.         a=10/0;
3.         print "Exception occurred"
4.     finally:
5.         print "Code to be executed"
```

Output:

```
1.     >>>
2.     Code to be executed
3.     Traceback (most recent call last):
4.       File "C:/Python27/noexception.py", line 2, in <module>
5.         a=10/0;
6.     ZeroDivisionError: integer division or modulo by zero
7.     >>>
```

In the above example finally block is executed. Since exception is not handled therefore exception occurred and execution is stopped.

RAISE AN EXCEPTION:

You can explicitly throw an exception in Python using `raise` statement. `raise` will cause an exception to occur and thus execution control will stop in case it is not handled.

Syntax:

```
1.      raise Exception_class,<value>
```

eg:

```
1.      try:
2.          a=10
3.          print a
4.          raise NameError("Hello")
5.      except NameError as e:
6.          print "An exception occurred"
7.          print e
```

Output:

```
1.      >>>
2.      10
3.      An exception occurred
4.      Hello
5.      >>>
```

Explanation:

- i) To raise an exception, `raise` statement is used. It is followed by exception class name.
- ii) Exception can be provided with a value that can be given in the parenthesis. (here, Hello)
- iii) To access the value "as" keyword is used. "e" is used as a reference variable which stores the value of the exception.

CUSTOM EXCEPTION:

Refer to this section after visiting Class and Object section:

Creating your own Exception class or User Defined Exceptions are known as Custom Exception.

eg:

```
1.      class ErrorInCode(Exception):
2.          def __init__(self, data):
3.              self.data = data
4.          def __str__(self):
5.              return repr(self.data)
6.
7.      try:
8.          raise ErrorInCode(2000)
9.      except ErrorInCode as ae:
10.         print "Received error:", ae.data
```

Output:

```
1.      >>>
2.      Received error : 2000
3.      >>>
```

DATE AND TIME

Python is very useful in case of Date and Time. We can easily retrieve current date and time using Python.

RETRIEVE TIME

To retrieve current time a predefined function `localtime()` is used. `localtime()` receives a parameter `time.time()`. Here,

`time` is a module,

`time()` is a function that returns the current system time in number of ticks since 12:00 am, January 1, 1970. It is known as epoch.

Tick is simply a floating point number in seconds since epoch.

eg:

```
1.      import time;
2.      localtime = time.localtime(time.time())
```

3. `print "Current Time is :", localtime`

Output:

```
1. >>>
2. Current Time is :time.struct_time(tm_year=2014, tm_mon=6, tm_mday=18, tm_hour=12,
3. tm_min=35, tm_sec=44, tm_wday=2, tm_yday=169, tm_isdst=0)
4. >>>
```

Explanation:

The time returned is a time structure which includes 9 attributes. These are summoned in the table given below.

Attribute	Description
tm_year	Returns the current year
tm_mon	Returns the current month
tm_mday	Returns the current month day
tm_hour	Returns the current hour.
tm_min	Returns the current minute
tm_sec	Returns current seconds
tm_wday	Returns the week day
tm_yday	Returns the year day.
tm_isdst	It returns -1,0 or 1.

FORMATTED TIME

Python also support formatted time. Proceed as follows:

1. Pass the time structure in a predefined function `asctime()`. It is a function defined in `time` module.
2. It returns a formatted time which includes Day ,month, date, time and year.
3. Print the formatted time.

eg:

1. `import time;`
- 2.
3. `localtime = time.asctime(time.localtime(time.time()))`
4. `print "Formatted time :", localtime`

Output:

1. `>>>`
2. `Formatted time : Sun Jun 22 18:54:20 2014`
3. `>>>`

TIME MODULE:

There are many built in functions defined in `time` module which are used to work with time.

Methods	Description
<code>time()</code>	Returns floating point value in seconds since epoch i.e.,12 1970
<code>asctime(time)</code>	It takes the tuple returned by <code>localtime()</code> as parameter and returns a character string.
<code>sleep(time)</code>	The execution will be stopped for the given interval of time.
<code>strptime(String,format)</code>	It returns an tuple with 9 time attributes. It receives an S format.

gtime()/gtime(sec)	It returns struct_time which contains 9 time attributes. In case no time is specified it takes current second from epoch.
mktime()	Returns second in floating point since epoch.
strftime(format)/strftime(format,time)	Returns time in particular format. If time is not given, current time is fetched.

time()

eg:

1. `import time`
2. `printtime.time()`

Output:

1. `>>>`
2. `1403700740.39`
3. `>>>`

asctime(time)

1. `import time`
2. `t = time.localtime()`
3. `printtime.asctime(t)`

Output:

1. `>>>`
2. `Wed Jun 25 18:30:25 2014`
3. `>>>`

sleep(time)

Eg:

1. `import time`
2. `time.sleep(1)`
3. `localtime = time.asctime(time.localtime(time.time()))`

```

4.      printlocaltime
5.      time.sleep( 10 )
6.      localtime = time.asctime( time.localtime(time.time()) )
7.      printlocaltime

```

Output:

```

1.      >>>
2.      Wed Jun 25 18:15:30 2014
3.      Wed Jun 25 18:15:40 2014
4.      >>>

```

strptime(String str,format f)

Eg:

```

1.      import time
2.
3.      timerequired = time.strptime("26 Jun 14", "%d %b %y")
4.      printtimerequired

```

Output:

```

1.      >>>
2.      time.struct_time(tm_year=2014, tm_mon=6, tm_mday=26, tm_hour=0, tm_min=0
,
3.      tm_sec=0, tm_wday=3, tm_yday=177, tm_isdst=-1)
4.      >>>

```

Explanation:

The `strptime()` takes a String and format as argument. The format refers to String passed as an argument. "%a %b %d %H:%M:%S %Y" are the default directives. There are many other directives which can be used. In the given example we have used three directives: %d %b %y which specifies day of the month, month in abbreviated form and year without century respectively. Some of them are given as:

%a	weekday name.
%b	month name

%c	date and time
%e	day of a month
%m	month in digit.
%n	new line character.
%S	second
%t	tab character

etc...

gmtime()

Eg:

1. `import time`
2. `printtime.gmtime()`

Output:

1. `>>>`
2. `time.struct_time(tm_year=2014, tm_mon=6, tm_mday=28, tm_hour=9, tm_min=38, tm_sec=0,`
3. `tm_wday=5, tm_yday=179, tm_isdst=0)`
4. `>>>`

mktime()

Eg:

1. `import time`
2. `t = (2014, 2, 17, 17, 3, 38, 1, 48, 0)`
3. `second = time.mktime(t)`
4. `print second`

Output:


```
1.      >>>
2.      1392636818.0
3.      >>>
```

strftime()

Eg:

```
1.      import time
2.      t = (2014, 6, 26, 17, 3, 38, 1, 48, 0)
3.      t = time.mktime(t)
4.      printtime.strftime("%b %d %Y %H:%M:%S", time.gmtime(t))
```

Output:

```
1.      >>>
2.      Jun 26 2014 11:33:38
3.      >>>
```

CALENDAR

Python provides calendar module to display Calendar.

Eg:

```
1.      import calendar
2.      print "Current month is:"
3.      cal = calendar.month(2014, 6)
4.      printcal
```

Output:

```
1.      >>>
2.      Current month is:
3.      June 2014
4.      Mo Tu We Th Fr Sa Su
5.      1
6.      2 3 4 5 6 7 8
7.      9 10 11 12 13 14 15
8.      16 17 18 19 20 21 22
9.      23 24 25 26 27 28 29
```

10. 30
11. >>>

CALENDAR MODULE:

Python provides calendar module which provides many functions and methods to work on calendar. A list of methods and function used is given below:

Methods	Description
prcal(year)	Prints the whole calendar of the year.
firstweekday()	Returns the first week day. It is by default 0 which specifies Monday
isleap(year)	Returns a Boolean value i.e., true or false. True in case given year is
monthcalendar(year,month)	Returns the given month with each week as one list.
leapdays(year1,year2)	Return number of leap days from year1 to year2
prmonth(year,month)	Print the given month of the given year

prcal(year)

Eg:

1. `import calendar`
2. `calendar.prcal(2014)`

Output:

1. `>>> ===== RESTART =====`
`=====`
2. `>>>`

2014

January	February	March
Mo Tu We Th Fr Sa Su	Mo Tu We Th Fr Sa Su	Mo Tu We Th Fr Sa Su
1 2 3 4 5	1 2	1 2
6 7 8 9 10 11 12	3 4 5 6 7 8 9	3 4 5 6 7 8 9
13 14 15 16 17 18 19	10 11 12 13 14 15 16	10 11 12 13 14 15 16
20 21 22 23 24 25 26	17 18 19 20 21 22 23	17 18 19 20 21 22 23
27 28 29 30 31	24 25 26 27 28	24 25 26 27 28 29 30
	31	

April	May	June
Mo Tu We Th Fr Sa Su	Mo Tu We Th Fr Sa Su	Mo Tu We Th Fr Sa Su
1 2 3 4 5 6	1 2 3 4	1
7 8 9 10 11 12 13	5 6 7 8 9 10 11	2 3 4 5 6 7 8
14 15 16 17 18 19 20	12 13 14 15 16 17 18	9 10 11 12 13 14 15
21 22 23 24 25 26 27	19 20 21 22 23 24 25	16 17 18 19 20 21 22
28 29 30	26 27 28 29 30 31	23 24 25 26 27 28 29
	30	

July	August	September
Mo Tu We Th Fr Sa Su	Mo Tu We Th Fr Sa Su	Mo Tu We Th Fr Sa Su
1 2 3 4 5 6	1 2 3	1 2 3 4 5 6 7
7 8 9 10 11 12 13	4 5 6 7 8 9 10	8 9 10 11 12 13 14
14 15 16 17 18 19 20	11 12 13 14 15 16 17	15 16 17 18 19 20 21
21 22 23 24 25 26 27	18 19 20 21 22 23 24	22 23 24 25 26 27 28
28 29 30 31	25 26 27 28 29 30 31	29 30

October	November	December
Mo Tu We Th Fr Sa Su	Mo Tu We Th Fr Sa Su	Mo Tu We Th Fr Sa Su
1 2 3 4 5	1 2	1 2 3 4 5 6 7
6 7 8 9 10 11 12	3 4 5 6 7 8 9	8 9 10 11 12 13 14
13 14 15 16 17 18 19	10 11 12 13 14 15 16	15 16 17 18 19 20 21
20 21 22 23 24 25 26	17 18 19 20 21 22 23	22 23 24 25 26 27 28
27 28 29 30 31	24 25 26 27 28 29 30	29 30 31

>>>

javatpoint.com

firstweekday()

Eg:

1. `import calendar`
2. `printcalendar.firstweekday()`

Output:

1. `>>>`

2. 0
3. >>>

isleap(year)

Eg:

1. `import calendar`
2. `printcalendar.isleap(2000)`

Output:

1. >>>
2. True
3. >>>

monthcalendar(year,month)

Eg:

1. `import calendar`
2. `printcalendar.monthcalendar(2014,6)`

Output:

1. >>>
2. `[[0, 0, 0, 0, 0, 0, 1], [2, 3, 4, 5, 6, 7, 8], [9, 10, 11, 12, 13, 14, 15],`
3. `[16, 17, 18, 19, 20, 21, 22],`
4. `[23, 24, 25, 26, 27, 28, 29], [30, 0, 0, 0, 0, 0, 0]]`
5. >>>

prmonth(year,month)

Eg:

1. `import calendar`
2. `printcalendar.prmonth(2014,6)`

Output:

1. >>>
2. June 2014
3. Mo Tu We ThFrSa Su

```
4.          1
5.      2 3 4 5 6 7 8
6.      9 10 11 12 13 14 15
7.     16 17 18 19 20 21 22
8.     23 24 25 26 27 28 29
9.     30
10.    None
11.    >>>
```

PYTHON PROGRAMS

There can be various python programs on many topics like basic python programming, conditions and loops, functions and native data types. A list of top python programs are given below which are widely asked by interviewer.

BASIC PYTHON PROGRAMS

- [Python program to print "Hello Python"](#)
- [Python program to do arithmetical operations](#)
- [Python program to find the area of a triangle](#)
- [Python program to solve quadratic equation](#)
- [Python program to swap two variables](#)
- [Python program to generate a random number](#)
- [Python program to convert kilometers to miles](#)
- [Python program to convert Celsius to Fahrenheit](#)
- [Python program to display calendar](#)

PYTHON PROGRAMS WITH CONDITIONS AND LOOPS

- [Python Program to Check if a Number is Positive, Negative or Zero](#)
- [Python Program to Check if a Number is Odd or Even](#)
- [Python Program to Check Leap Year](#)
- [Python Program to Check Prime Number](#)
- [Python Program to Print all Prime Numbers in an Interval](#)
- [Python Program to Find the Factorial of a Number](#)

- [Python Program to Display the multiplication Table](#)
- [Python Program to Print the Fibonacci sequence](#)
- [Python Program to Check Armstrong Number](#)
- [Python Program to Find Armstrong Number in an Interval](#)
- [Python Program to Find the Sum of Natural Numbers](#)

PYTHON FUNCTION PROGRAMS

- [Python Program to Find LCM](#)
- [Python Program to Find HCF](#)
- [Python Program to Convert Decimal to Binary, Octal and Hexadecimal](#)
- [Python Program To Find ASCII value of a character](#)
- [Python Program to Make a Simple Calculator](#)
- [Python Program to Display Calendar](#)
- [Python Program to Display Fibonacci Sequence Using Recursion](#)
- [Python Program to Find Factorial of Number Using Recursion](#)

PYTHON NATIVE DATA TYPE PROGRAMS

- [Python Program to Add Two Matrices](#)
- [Python Program to Multiply Two Matrices](#)
- [Python Program to Transpose a Matrix](#)
- [Python Program to Sort Words in Alphabetic Order](#)
- [Python Program to Remove Punctuation From a String](#)

PYTHON PROGRAMS WITH CONDITIONS AND LOOPS

There can be various programs on conditions and loops. A list of top useful condition and loop programs are given below:

- [Python Program to Check if a Number is Positive, Negative or Zero](#)
- [Python Program to Check if a Number is Odd or Even](#)
- [Python Program to Check Leap Year](#)
- [Python Program to Check Prime Number](#)

- [Python Program to Print all Prime Numbers in an Interval](#)
- [Python Program to Find the Factorial of a Number](#)
- [Python Program to Display the multiplication Table](#)
- [Python Program to Print the Fibonacci sequence](#)
- [Python Program to Check Armstrong Number](#)
- [Python Program to Find Armstrong Number in an Interval](#)
- [Python Program to Find the Sum of Natural Numbers](#)

PYTHON FUNCTION PROGRAMS

There can be many useful function programs in python. A list of top python function programs are given below:

- [Python Program to Find LCM](#)
- [Python Program to Find HCF](#)
- [Python Program to Convert Decimal to Binary, Octal and Hexadecimal](#)
- [Python Program To Find ASCII value of a character](#)
- [Python Program to Make a Simple Calculator](#)
- [Python Program to Display Calendar](#)
- [Python Program to Display Fibonacci Sequence Using Recursion](#)
- [Python Program to Find Factorial of Number Using Recursion](#)

PYTHON PROGRAM TO FIND LCM

LCM: Least Common Multiple/ Lowest Common Multiple

LCM stands for Least Common Multiple. It is a concept of arithmetic and number system. The LCM of two integers a and b is denoted by LCM (a,b). It is the smallest positive integer that is divisible by both "a" and "b".

For example: We have two integers 4 and 6. Let's find LCM

Multiples of 4 are:

1. 4, 8, 12, 16, 20, 24, 28, 32, 36,... **and** so on...

Multiples of 6 are:

1. 6, 12, 18, 24, 30, 36, 42,... **and** so on....

Common multiples of 4 and 6 are simply the numbers that are in both lists:

1. 12, 24, 36, 48, 60, 72,.... **and** so on....

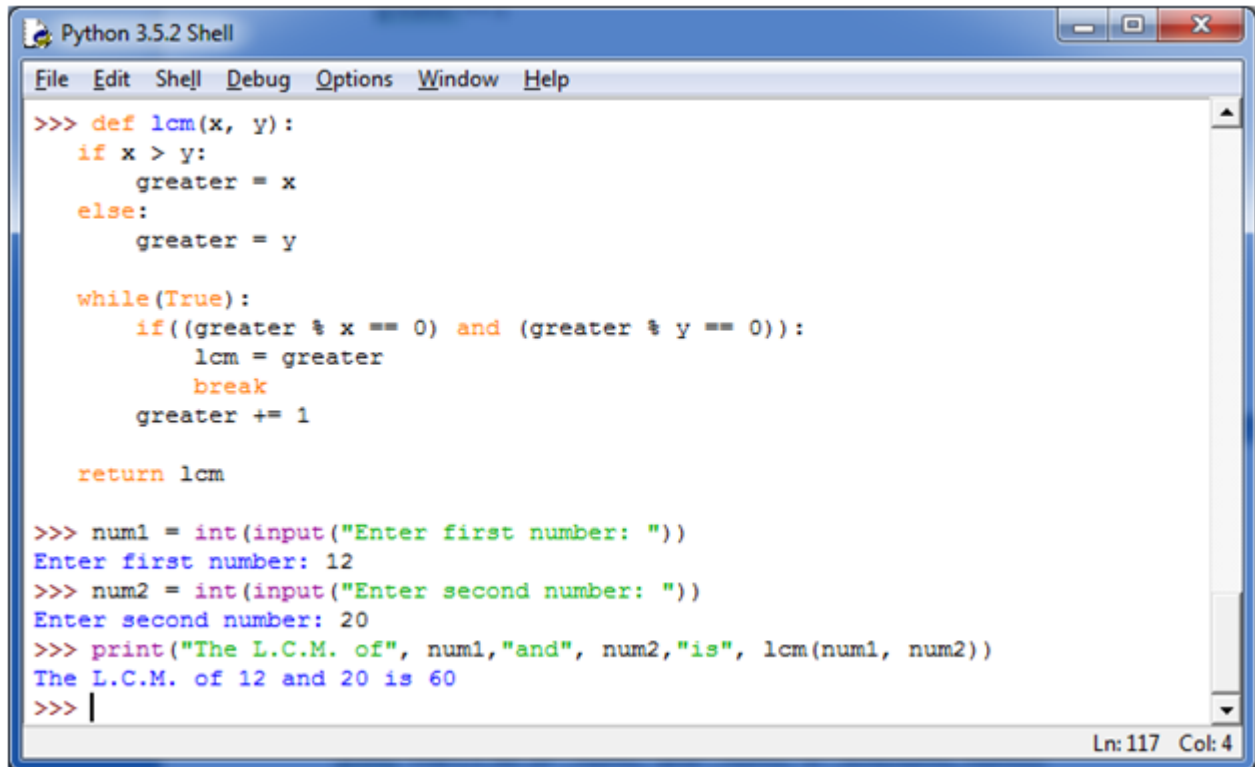
LCM is the lowest common multiplier so it is 12.

See this example:

```
1. def lcm(x, y):
2.     if x > y:
3.         greater = x
4.     else:
5.         greater = y
6.     while(True):
7.         if((greater % x == 0) and (greater % y == 0)):
8.             lcm = greater
9.             break
10.        greater += 1
11.    return lcm
12.
13.
14.    num1 = int(input("Enter first number: "))
15.    num2 = int(input("Enter second number: "))
16.    print("The L.C.M. of", num1,"and", num2,"is", lcm(num1, num2))
```

The following example will show the LCM of 12 and 20 (according to the user input)

Output:



```
Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
>>> def lcm(x, y):
    if x > y:
        greater = x
    else:
        greater = y

    while(True):
        if((greater % x == 0) and (greater % y == 0)):
            lcm = greater
            break
        greater += 1

    return lcm

>>> num1 = int(input("Enter first number: "))
Enter first number: 12
>>> num2 = int(input("Enter second number: "))
Enter second number: 20
>>> print("The L.C.M. of", num1,"and", num2,"is", lcm(num1, num2))
The L.C.M. of 12 and 20 is 60
>>> |
```

Ln: 117 Col: 4

PYTHON PROGRAM TO PRINT "HELLO PYTHON"

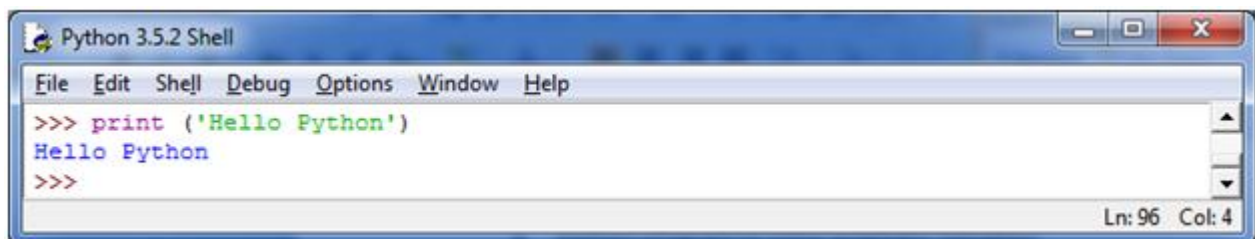
This is the most basic Python program. It specifies how to print any statement in Python.

In old python (up to Python 2.7.0), print command is not written in parenthesis but in new python software (Python 3.4.3), it is mandatory to add a parenthesis to print a statement.

See this example:

1. `print ('Hello Python')`

Output:



```
Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
>>> print ('Hello Python')
Hello Python
>>>
```

Ln: 96 Col: 4

PYTHON PROGRAM TO DO ARITHMETICAL OPERATIONS

The arithmetic operations are performed by calculator where we can perform addition, subtraction, multiplication and division. This example shows the basic arithmetic operations i.e.

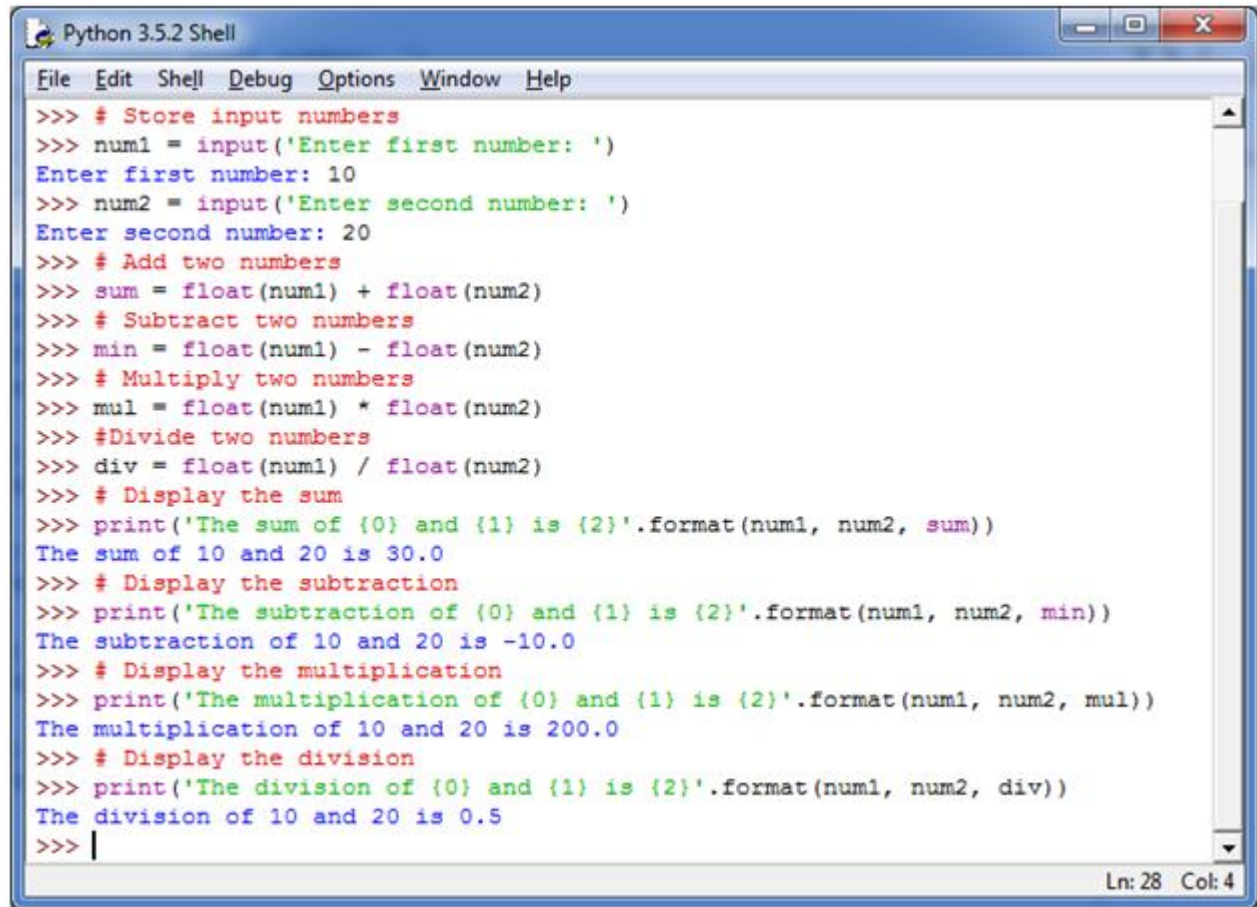
- Addition
- Subtraction
- Multiplication
- Division

See this example:

```
1.      # Store input numbers:
2.      num1 = input('Enter first number: ')
3.      num2 = input('Enter second number: ')
4.
5.      # Add two numbers
6.      sum = float(num1) + float(num2)
7.      # Subtract two numbers
8.      min = float(num1) - float(num2)
9.      # Multiply two numbers
10.     mul = float(num1) * float(num2)
11.     #Divide two numbers
12.     div = float(num1) / float(num2)
13.     # Display the sum
14.     print('The sum of {0} and {1} is {2}'.format(num1, num2, sum))
15.
16.     # Display the subtraction
17.     print('The subtraction of {0} and {1} is {2}'.format(num1, num2, min))
18.     # Display the multiplication
19.     print('The multiplication of {0} and {1} is {2}'.format(num1, num2, mul))
20.     # Display the division
21.     print('The division of {0} and {1} is {2}'.format(num1, num2, div))
```

Note: Here input numbers are 10 and 20.

Output:



```
Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
>>> # Store input numbers
>>> num1 = input('Enter first number: ')
Enter first number: 10
>>> num2 = input('Enter second number: ')
Enter second number: 20
>>> # Add two numbers
>>> sum = float(num1) + float(num2)
>>> # Subtract two numbers
>>> min = float(num1) - float(num2)
>>> # Multiply two numbers
>>> mul = float(num1) * float(num2)
>>> # Divide two numbers
>>> div = float(num1) / float(num2)
>>> # Display the sum
>>> print('The sum of {0} and {1} is {2}'.format(num1, num2, sum))
The sum of 10 and 20 is 30.0
>>> # Display the subtraction
>>> print('The subtraction of {0} and {1} is {2}'.format(num1, num2, min))
The subtraction of 10 and 20 is -10.0
>>> # Display the multiplication
>>> print('The multiplication of {0} and {1} is {2}'.format(num1, num2, mul))
The multiplication of 10 and 20 is 200.0
>>> # Display the division
>>> print('The division of {0} and {1} is {2}'.format(num1, num2, div))
The division of 10 and 20 is 0.5
>>> |
```

PYTHON PROGRAM TO FIND THE AREA OF A TRIANGLE

Mathematical formula:

$$\text{Area of a triangle} = (s \cdot (s-a) \cdot (s-b) \cdot (s-c))^{-1/2}$$

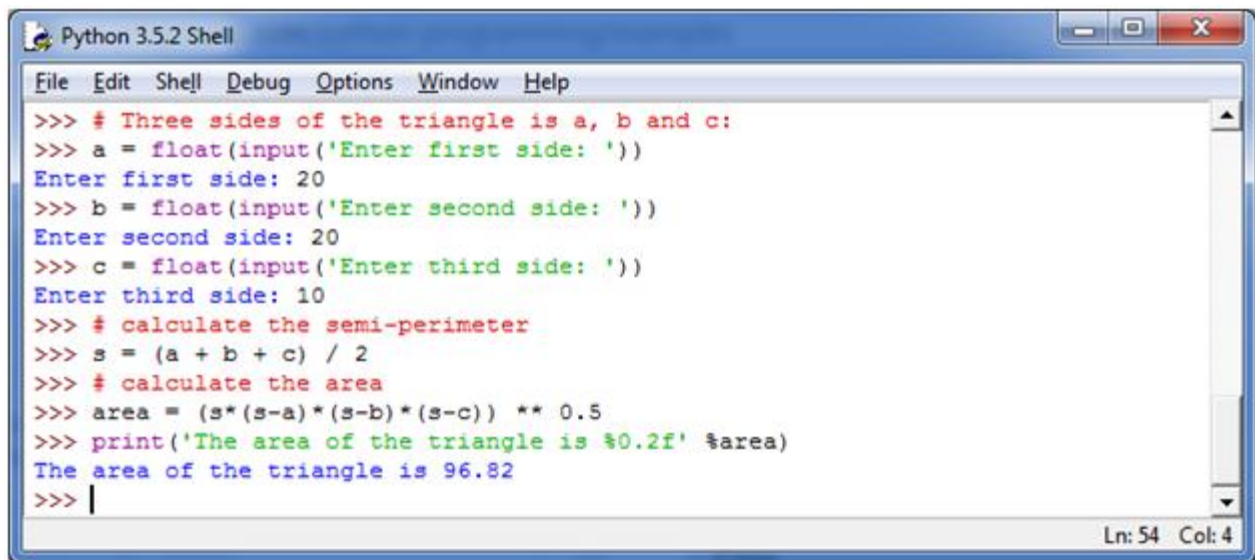
Here s is the semi-perimeter and a, b and c are three sides of the triangle.

See this example:

1. `# Three sides of the triangle is a, b and c:`
2. `a = float(input('Enter first side: '))`
3. `b = float(input('Enter second side: '))`
4. `c = float(input('Enter third side: '))`
- 5.

6. `# calculate the semi-perimeter`
7. `s = (a + b + c) / 2`
- 8.
9. `# calculate the area`
10. `area = (s*(s-a)*(s-b)*(s-c)) ** 0.5`
11. `print('The area of the triangle is %0.2f' %area)`

Output:

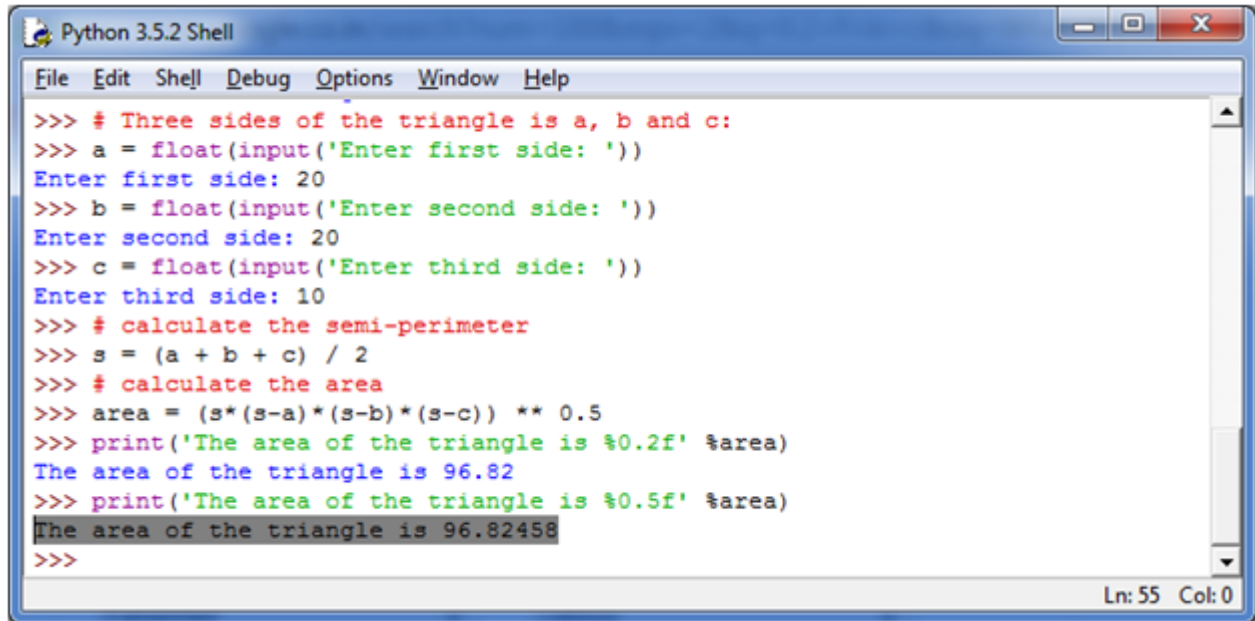


```
Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
>>> # Three sides of the triangle is a, b and c:
>>> a = float(input('Enter first side: '))
Enter first side: 20
>>> b = float(input('Enter second side: '))
Enter second side: 20
>>> c = float(input('Enter third side: '))
Enter third side: 10
>>> # calculate the semi-perimeter
>>> s = (a + b + c) / 2
>>> # calculate the area
>>> area = (s*(s-a)*(s-b)*(s-c)) ** 0.5
>>> print('The area of the triangle is %0.2f' %area)
The area of the triangle is 96.82
>>> |
```

Ln: 54 Col: 4

Note: %0.2f floating point specifies at least 0 wide and 2 numbers after decimal. If you use %0.5f then it will give 5 numbers after decimal.

See this example:

A screenshot of a Python 3.5.2 Shell window. The window has a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area contains the following code and output:

```
>>> # Three sides of the triangle is a, b and c:
>>> a = float(input('Enter first side: '))
Enter first side: 20
>>> b = float(input('Enter second side: '))
Enter second side: 20
>>> c = float(input('Enter third side: '))
Enter third side: 10
>>> # calculate the semi-perimeter
>>> s = (a + b + c) / 2
>>> # calculate the area
>>> area = (s*(s-a)*(s-b)*(s-c)) ** 0.5
>>> print('The area of the triangle is %0.2f' %area)
The area of the triangle is 96.82
>>> print('The area of the triangle is %0.5f' %area)
The area of the triangle is 96.82458
>>>
```

The status bar at the bottom right shows 'Ln: 55 Col: 0'.

PYTHON PROGRAM TO SOLVE QUADRATIC EQUATION

Quadratic equation:

Quadratic equation is made from a Latin term "quadrates" which means square. It is a special type of equation having the form of:

$$ax^2+bx+c=0$$

Here, "x" is unknown which you have to find and "a", "b", "c" specifies the numbers such that "a" is not equal to 0. If a = 0 then the equation becomes liner not quadratic anymore.

In the equation, a, b and c are called coefficients.

Let's take an example to solve the quadratic equation $8x^2 + 16x + 8 = 0$

See this example:

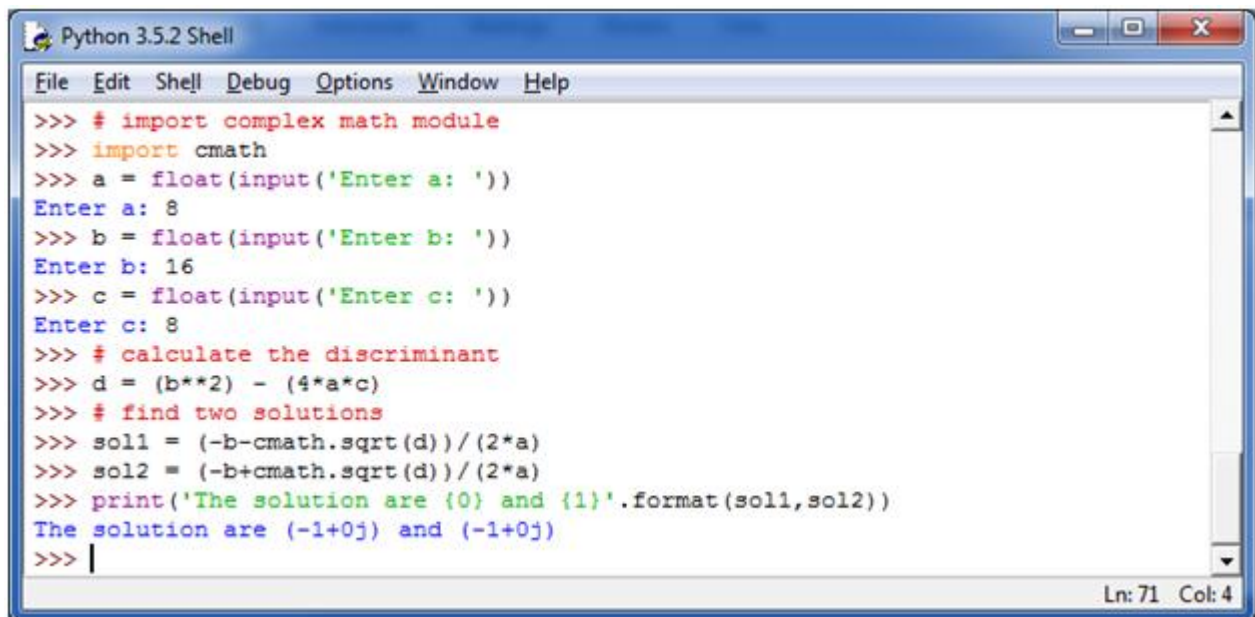
1. `# import complex math module`
2. `import cmath`
3. `a = float(input('Enter a: '))`
4. `b = float(input('Enter b: '))`
5. `c = float(input('Enter c: '))`
- 6.
7. `# calculate the discriminant`

```

8.      d = (b**2) - (4*a*c)
9.
10.     # find two solutions
11.     sol1 = (-b-cmath.sqrt(d))/(2*a)
12.     sol2 = (-b+cmath.sqrt(d))/(2*a)
13.     print('The solution are {0} and {1}'.format(sol1,sol2))

```

Output:



```

Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
>>> # import complex math module
>>> import cmath
>>> a = float(input('Enter a: '))
Enter a: 8
>>> b = float(input('Enter b: '))
Enter b: 16
>>> c = float(input('Enter c: '))
Enter c: 8
>>> # calculate the discriminant
>>> d = (b**2) - (4*a*c)
>>> # find two solutions
>>> sol1 = (-b-cmath.sqrt(d))/(2*a)
>>> sol2 = (-b+cmath.sqrt(d))/(2*a)
>>> print('The solution are {0} and {1}'.format(sol1,sol2))
The solution are (-1+0j) and (-1+0j)
>>> |
Ln: 71 Col: 4

```

PYTHON PROGRAM TO SWAP TWO VARIABLES

Variable swapping:

In computer programming, swapping two variables specifies the mutual exchange of values of the variables. It is generally done by using a temporary variable.

For example:

```

1.      data_item x := 1
2.      data_item y := 0
3.      swap (x, y)

```

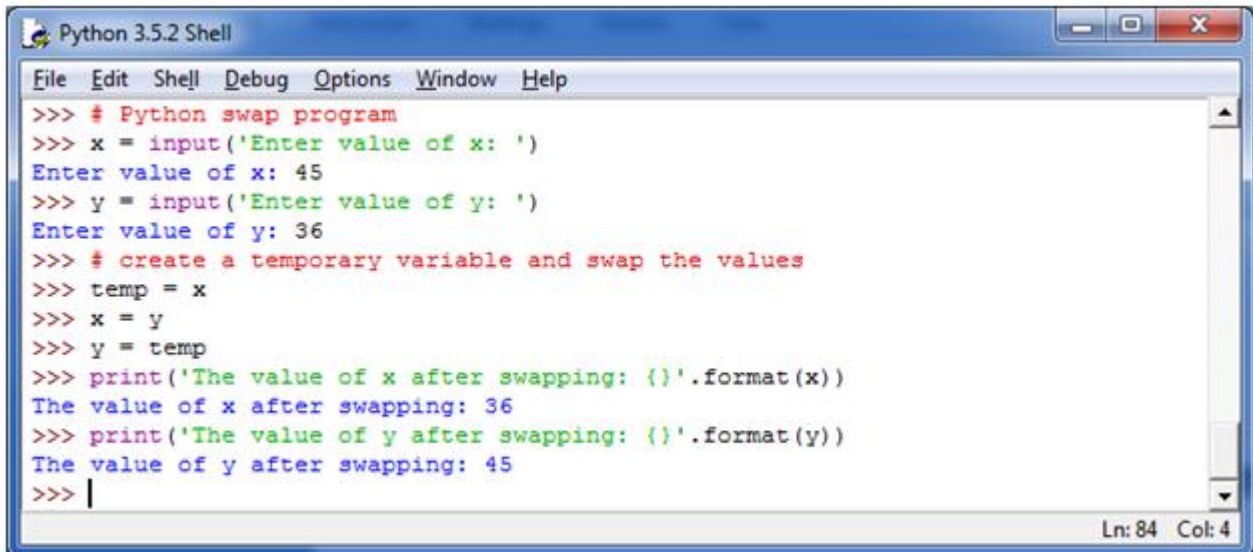
After swapping:

1. data_item x := 0
2. data_item y := 1

See this example:

1. # Python swap program
2. x = input('Enter value of x: ')
3. y = input('Enter value of y: ')
- 4.
5. # create a temporary variable and swap the values
6. temp = x
7. x = y
8. y = temp
- 9.
10. print('The value of x after swapping: {}'.format(x))
11. print('The value of y after swapping: {}'.format(y))

Output:



```
Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
>>> # Python swap program
>>> x = input('Enter value of x: ')
Enter value of x: 45
>>> y = input('Enter value of y: ')
Enter value of y: 36
>>> # create a temporary variable and swap the values
>>> temp = x
>>> x = y
>>> y = temp
>>> print('The value of x after swapping: {}'.format(x))
The value of x after swapping: 36
>>> print('The value of y after swapping: {}'.format(y))
The value of y after swapping: 45
>>> |
```

PYTHON PROGRAM TO GENERATE A RANDOM NUMBER

In Python programming, you can generate a random integer, doubles, longs etc . in various ranges by importing a "random" class.

Syntax:

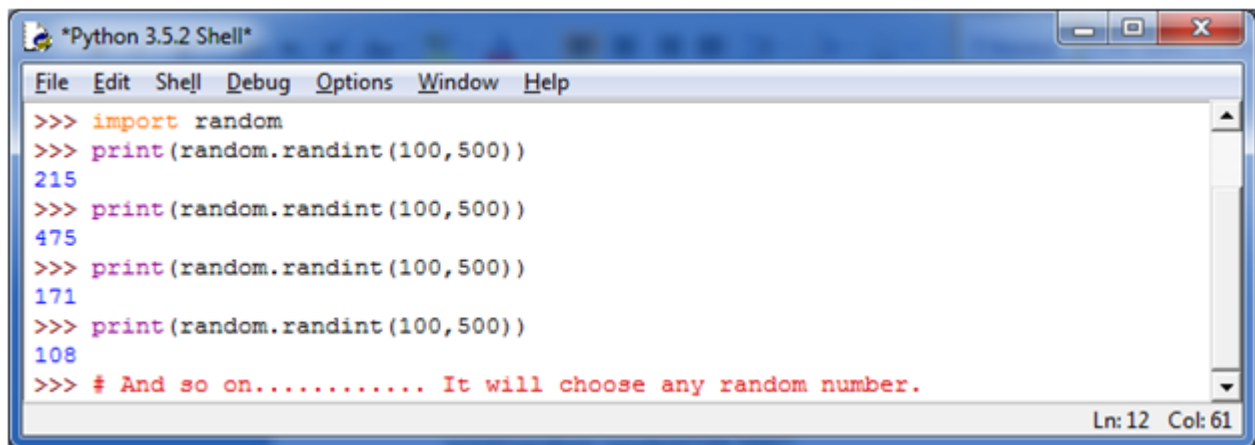
First you have to import the random module and then apply the syntax:

1. `import random`
2. `random.randint(a,b)`

See this example:

1. `import random`
2. `print(random.randint(100,500))`

Output:

A screenshot of a Python 3.5.2 Shell window. The window has a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area contains the following code:

```
>>> import random
>>> print(random.randint(100,500))
215
>>> print(random.randint(100,500))
475
>>> print(random.randint(100,500))
171
>>> print(random.randint(100,500))
108
>>> # And so on..... It will choose any random number.
```

The status bar at the bottom right shows 'Ln: 12 Col: 61'.

PYTHON PROGRAM TO CONVERT KILOMETERS TO MILES

Here, we are going to see the python program to convert kilometers to miles. Let's understand kilometers and miles first.

Kilometer:

The kilometer is a unit of length in the metric system. It is equivalent to 1000 meters.

Miles:

Mile is also the unit of length. It is equal to 1760 yards.

Conversion formula:

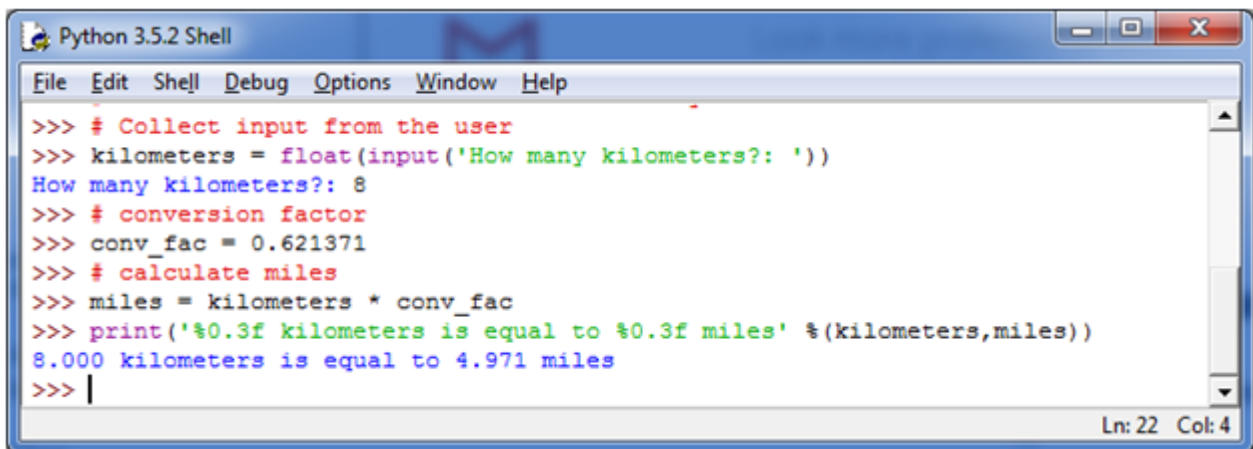
1 kilometer is equal to 0.62137 miles.

1. $\text{Miles} = \text{kilometer} * 0.62137$
2. $\text{Kilometer} = \text{Miles} / 0.62137$

See this example:

1. `# Collect input from the user`
2. `kilometers = float(input('How many kilometers?: '))`
3. `# conversion factor`
4. `conv_fac = 0.621371`
5. `# calculate miles`
6. `miles = kilometers * conv_fac`
7. `print('%0.3f kilometers is equal to %0.3f miles' %(kilometers,miles))`

Output:



```
Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
>>> # Collect input from the user
>>> kilometers = float(input('How many kilometers?: '))
How many kilometers?: 8
>>> # conversion factor
>>> conv_fac = 0.621371
>>> # calculate miles
>>> miles = kilometers * conv_fac
>>> print('%0.3f kilometers is equal to %0.3f miles' %(kilometers,miles))
8.000 kilometers is equal to 4.971 miles
>>> |
```

PYTHON PROGRAM TO CONVERT CELSIUS TO FAHRENHEIT

Celsius:

Celsius is a unit of measurement for temperature. It is also known as centigrade. It is a SI derived unit used by most of the countries worldwide.

It is named after the Swedish astronomer Anders Celsius.

Fahrenheit:

Fahrenheit is also a temperature scale. It is named on Polish-born German physicist Daniel Gabriel Fahrenheit. It uses degree Fahrenheit as a unit for temperature.

Conversion formula:

$$T(^{\circ}\text{F}) = T(^{\circ}\text{C}) \times 9/5 + 32$$

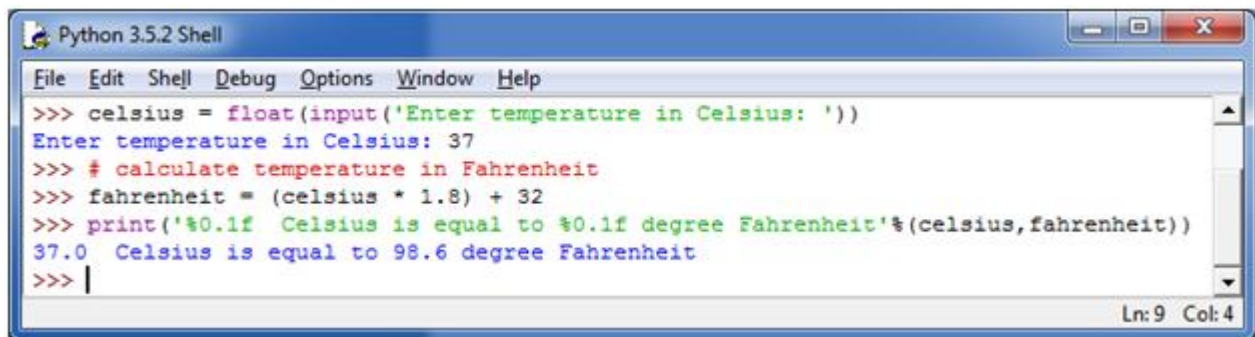
Or,

$$T(^{\circ}\text{F}) = T(^{\circ}\text{C}) \times 1.8 + 32$$

See this example:

1. `# Collect input from the user`
2. `celsius = float(input('Enter temperature in Celsius: '))`
- 3.
4. `# calculate temperature in Fahrenheit`
5. `fahrenheit = (celsius * 1.8) + 32`
6. `print('%0.1f Celsius is equal to %0.1f degree Fahrenheit'%(celsius,fahrenheit))`

Output:



```
Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
>>> celsius = float(input('Enter temperature in Celsius: '))
Enter temperature in Celsius: 37
>>> # calculate temperature in Fahrenheit
>>> fahrenheit = (celsius * 1.8) + 32
>>> print('%0.1f Celsius is equal to %0.1f degree Fahrenheit'%(celsius,fahrenheit))
37.0 Celsius is equal to 98.6 degree Fahrenheit
>>> |
```

PYTHON PROGRAM TO DISPLAY CALENDAR

It is simple in python programming to display calendar. To do so, you need to import the calendar module which comes with Python.

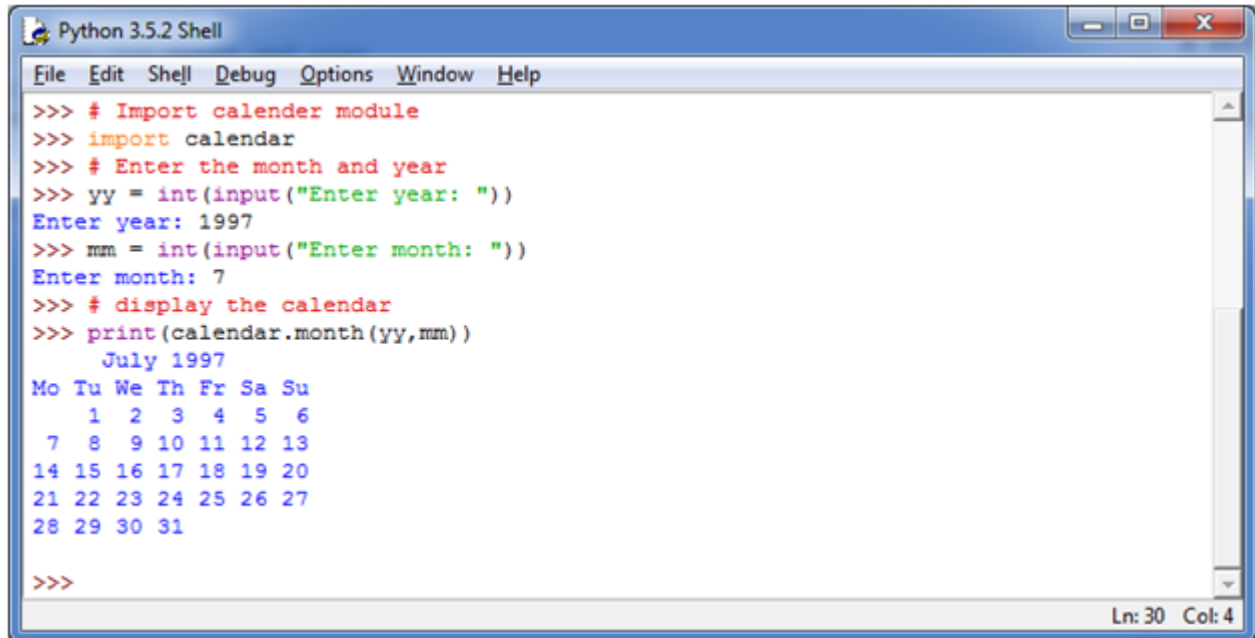
1. `import calendar`
2. And then apply the syntax
3. `(calendar.month(yy,mm))`

See this example:

1. `import calendar`
2. `# Enter the month and year`
3. `yy = int(input("Enter year: "))`
4. `mm = int(input("Enter month: "))`
- 5.

6. `# display the calendar`
7. `print(calendar.month(yy,mm))`

Output:



```

Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
>>> # Import calender module
>>> import calendar
>>> # Enter the month and year
>>> yy = int(input("Enter year: "))
Enter year: 1997
>>> mm = int(input("Enter month: "))
Enter month: 7
>>> # display the calendar
>>> print(calendar.month(yy,mm))
      July 1997
Mo Tu We Th Fr Sa Su
   1  2  3  4  5  6
  7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31

>>>
Ln: 30 Col: 4

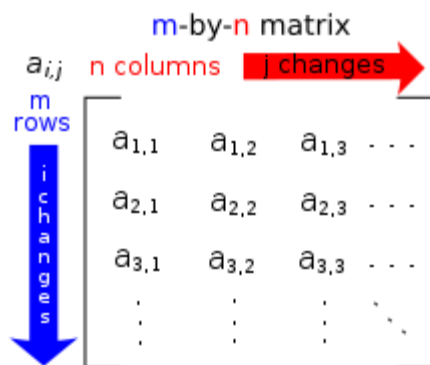
```

t is Matrix?

In mathematics, matrix is a rectangular array of numbers, symbols or expressions arranged in the form of rows and columns. For example: if you take a matrix A which is a 2x3 matrix then it can be shown like this:

1. 2 3 5
2. 8 12 7

Image representation:



In Python, matrices can be implemented as nested list. Each element of the matrix is treated as a row. For example $X = [[1, 2], [3, 4], [5, 6]]$ would represent a 3x2 matrix. First row can be selected as $X[0]$ and the element in first row, first column can be selected as $X[0][0]$.

Let's take two matrices X and Y, having the following value:

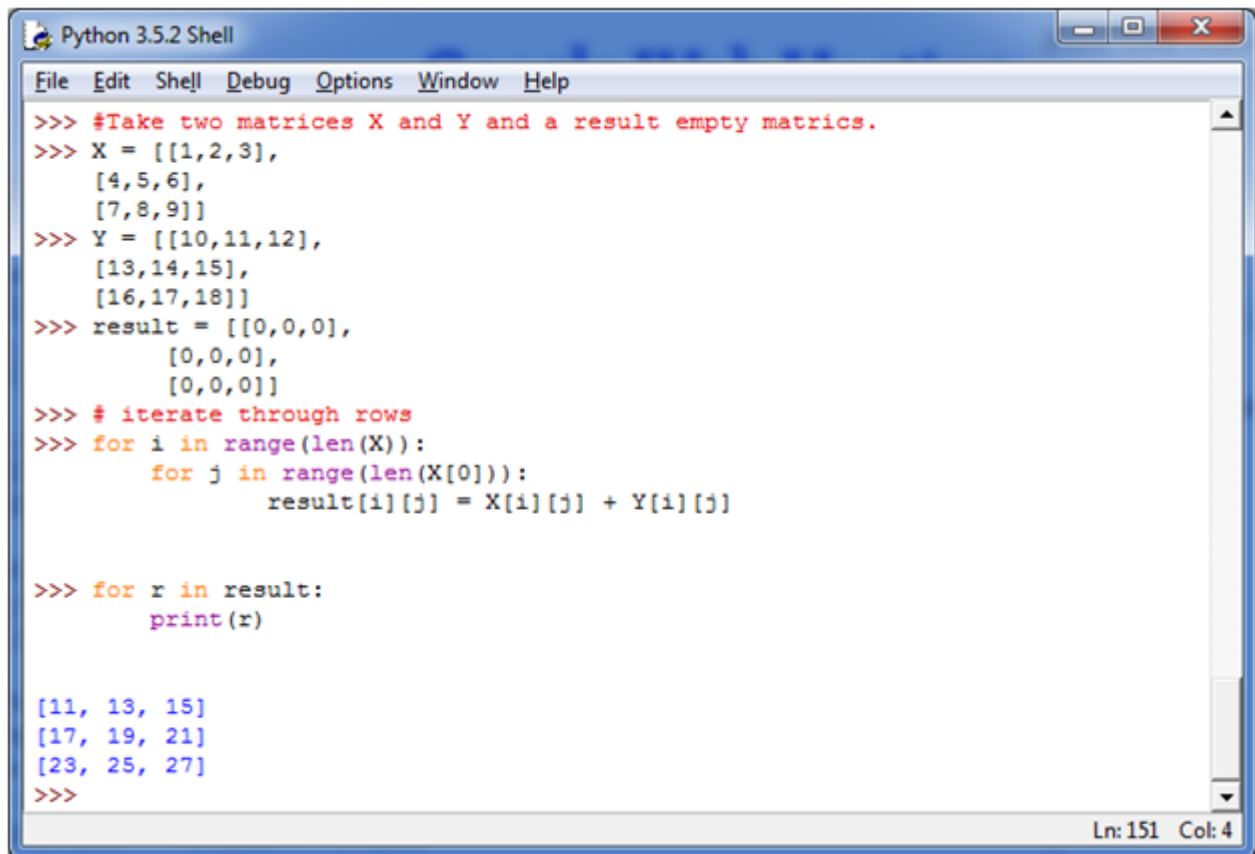
```
1.      X = [[1,2,3],
2.          [4,5,6],
3.          [7,8,9]]
4.
5.      Y = [[10,11,12],
6.          [13,14,15],
7.          [16,17,18]]
```

Create a new matrix result by adding them.

See this example:

```
1.      X = [[1,2,3],
2.          [4,5,6],
3.          [7,8,9]]
4.
5.      Y = [[10,11,12],
6.          [13,14,15],
7.          [16,17,18]]
8.
9.      Result = [[0,0,0],
10.              [0,0,0],
11.              [0,0,0]]
12.      # iterate through rows
13.      for i in range(len(X)):
14.          # iterate through columns
15.          for j in range(len(X[0])):
16.              result[i][j] = X[i][j] + Y[i][j]
17.      for r in result:
18.          print(r)
```

Output:



```
Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
>>> #Take two matrices X and Y and a result empty matrices.
>>> X = [[1,2,3],
        [4,5,6],
        [7,8,9]]
>>> Y = [[10,11,12],
        [13,14,15],
        [16,17,18]]
>>> result = [[0,0,0],
              [0,0,0],
              [0,0,0]]
>>> # iterate through rows
>>> for i in range(len(X)):
>>>     for j in range(len(X[0])):
>>>         result[i][j] = X[i][j] + Y[i][j]
>>>
>>> for r in result:
>>>     print(r)

[11, 13, 15]
[17, 19, 21]
[23, 25, 27]
>>>
```

PYTHON PROGRAM TO MULTIPLY TWO MATRICES

This Python program specifies how to multiply two matrices, having some certain values.

Matrix multiplication:

Matrix multiplication is a binary operation that uses a pair of matrices to produce another matrix. The elements within the matrix are multiplied according to elementary arithmetic.

See this example:

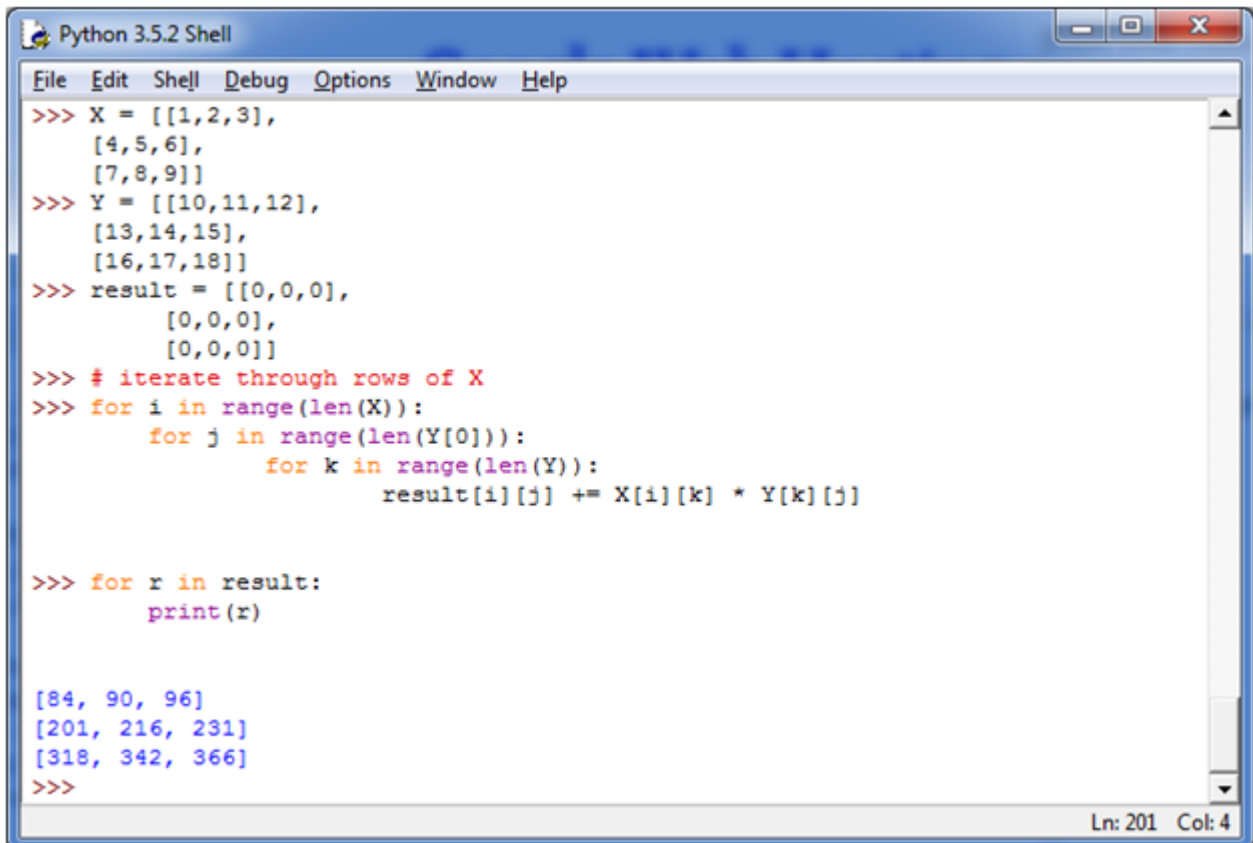
- 1.
2. `X = [[1,2,3],`
3. `[4,5,6],`
4. `[7,8,9]]`
- 5.
6. `Y = [[10,11,12],`
7. `[13,14,15],`

```

8.         [16,17,18]]
9.
10.    Result = [[0,0,0],
11.              [0,0,0],
12.              [0,0,0]]
13.
14.    # iterate through rows of X
15.    for i in range(len(X)):
16.        for j in range(len(Y[0])):
17.            for k in range(len(Y)):
18.                result[i][j] += X[i][k] * Y[k][j]
19.    for r in result:
20.        print(r)

```

Output:



```

Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
>>> X = [[1,2,3],
          [4,5,6],
          [7,8,9]]
>>> Y = [[10,11,12],
          [13,14,15],
          [16,17,18]]
>>> result = [[0,0,0],
               [0,0,0],
               [0,0,0]]
>>> # iterate through rows of X
>>> for i in range(len(X)):
>>>     for j in range(len(Y[0])):
>>>         for k in range(len(Y)):
>>>             result[i][j] += X[i][k] * Y[k][j]
>>>
>>> for r in result:
>>>     print(r)

[84, 90, 96]
[201, 216, 231]
[318, 342, 366]
>>>
Ln: 201 Col: 4

```

PYTHON PROGRAM TO TRANSPOSE A MATRIX

Transpose Matrix:

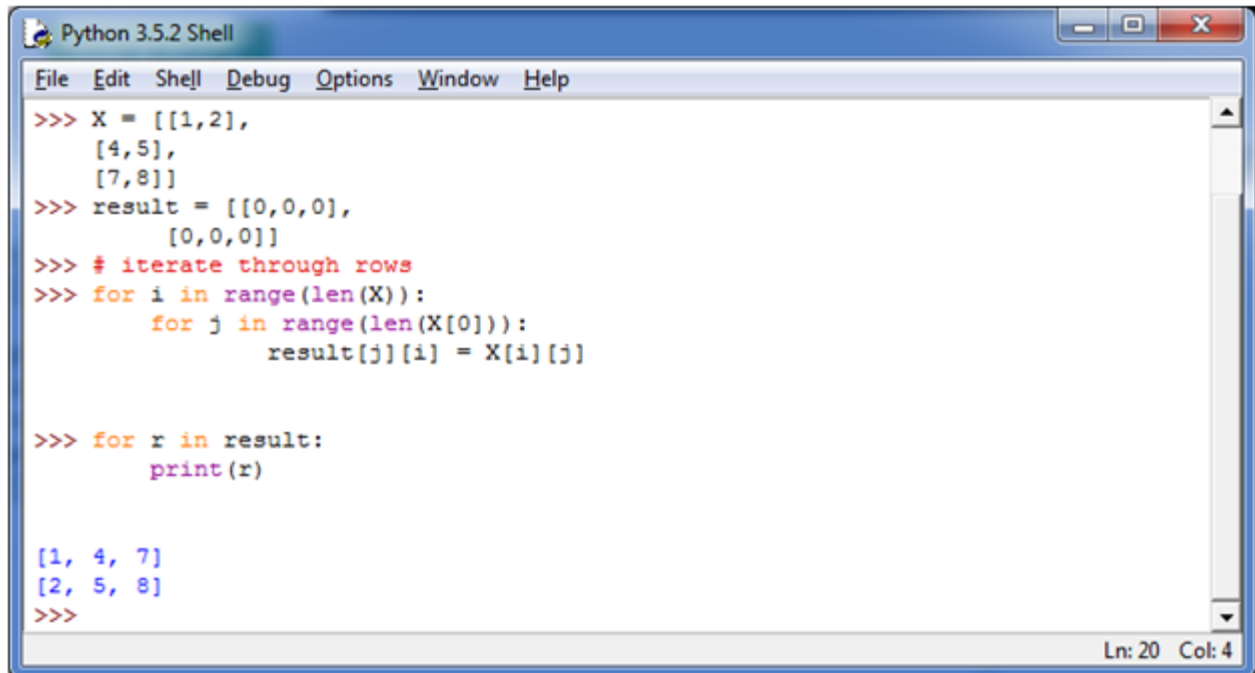
If you change the rows of a matrix with the column of the same matrix, it is known as transpose of a matrix. It is denoted as X' . **For example:** The element at i^{th} row and j^{th} column in X will be placed at j^{th} row and i^{th} column in X' .

Let's take a matrix X , having the following elements:

1. $X = [[1,2],$
2. $[4,5],$
3. $[7,8]]$

See this example:

1. $X = [[1,2],$
2. $[4,5],$
3. $[7,8]]$
- 4.
5. $\text{Result} = [[0,0,0],$
6. $[0,0,0]]$
- 7.
8. `# iterate through rows`
9. `for i in range(len(X)):`
10. `for j in range(len(X[0])):`
11. `result[j][i] = X[i][j]`
- 12.
13. `for r in result:`
14. `print(r)`
15. **Output:**



```
Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
>>> X = [[1,2],
         [4,5],
         [7,8]]
>>> result = [[0,0,0],
              [0,0,0]]
>>> # iterate through rows
>>> for i in range(len(X)):
>>>     for j in range(len(X[0])):
>>>         result[j][i] = X[i][j]
>>>
>>> for r in result:
>>>     print(r)

[1, 4, 7]
[2, 5, 8]
>>>
Ln: 20 Col: 4
```

PYTHON PROGRAM TO SORT WORDS IN ALPHABETIC ORDER

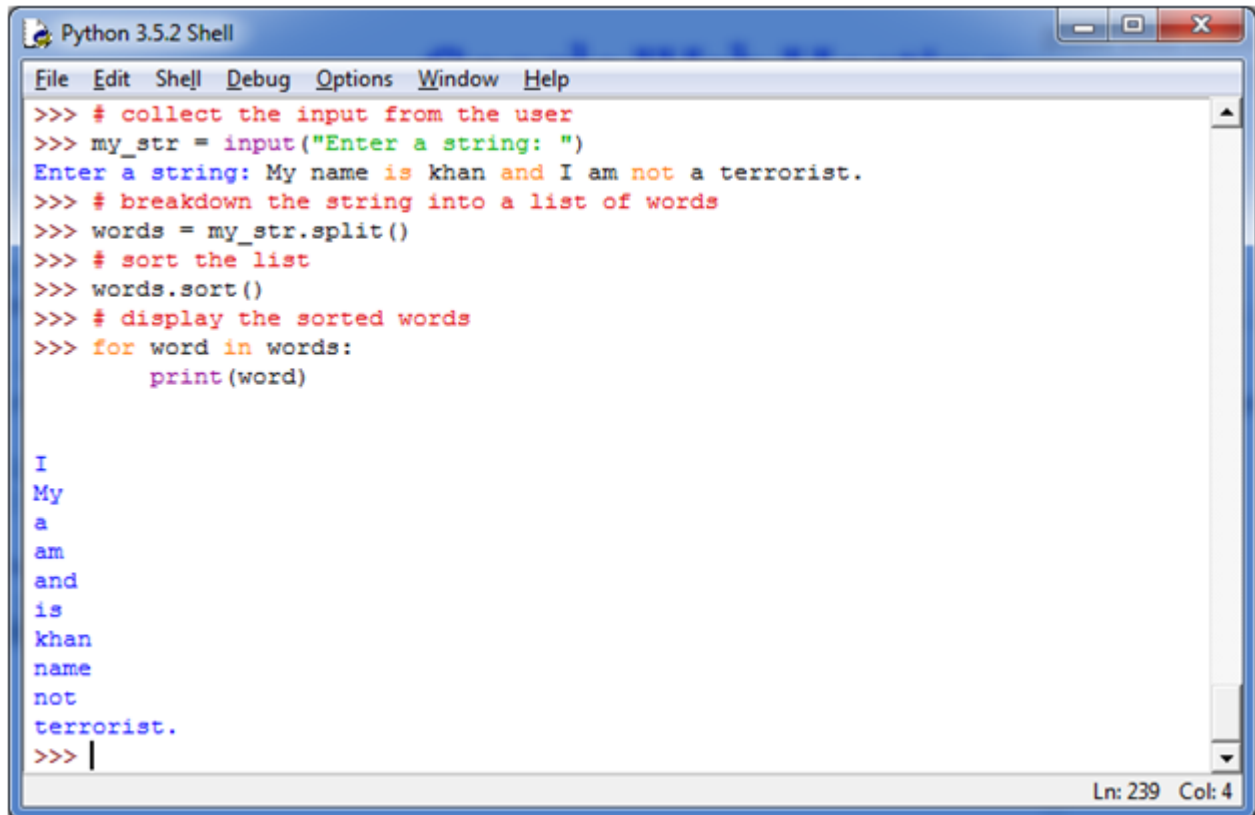
Sorting:

Sorting is a process of arrangement. It arranges data systematically in a particular format. It follows some algorithm to sort data.

See this example:

1. `my_str = input("Enter a string: ")`
2. `# breakdown the string into a list of words`
3. `words = my_str.split()`
4. `# sort the list`
5. `words.sort()`
6. `# display the sorted words`
7. `for word in words:`
8. `print(word)`

Output:



```
Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
>>> # collect the input from the user
>>> my_str = input("Enter a string: ")
Enter a string: My name is khan and I am not a terrorist.
>>> # breakdown the string into a list of words
>>> words = my_str.split()
>>> # sort the list
>>> words.sort()
>>> # display the sorted words
>>> for word in words:
>>>     print(word)

I
My
a
am
and
is
khan
name
not
terrorist.
>>> |
```

Ln: 239 Col: 4

PYTHON PROGRAM TO REMOVE PUNCTUATION FROM A STRING

Punctuation:

The practice, action, or system of inserting points or other small marks into texts, in order to aid interpretation; division of text into sentences, clauses, etc., is called punctuation. - Wikipedia

Punctuation are very powerful. They can change the entire meaning of a sentence.

See this example:

- "Woman, without her man, is nothing" (the sentence boasting about men's importance.)
- "Woman: without her, man is nothing" (the sentence boasting about women's importance.)

This program is written to remove punctuation from a statement.

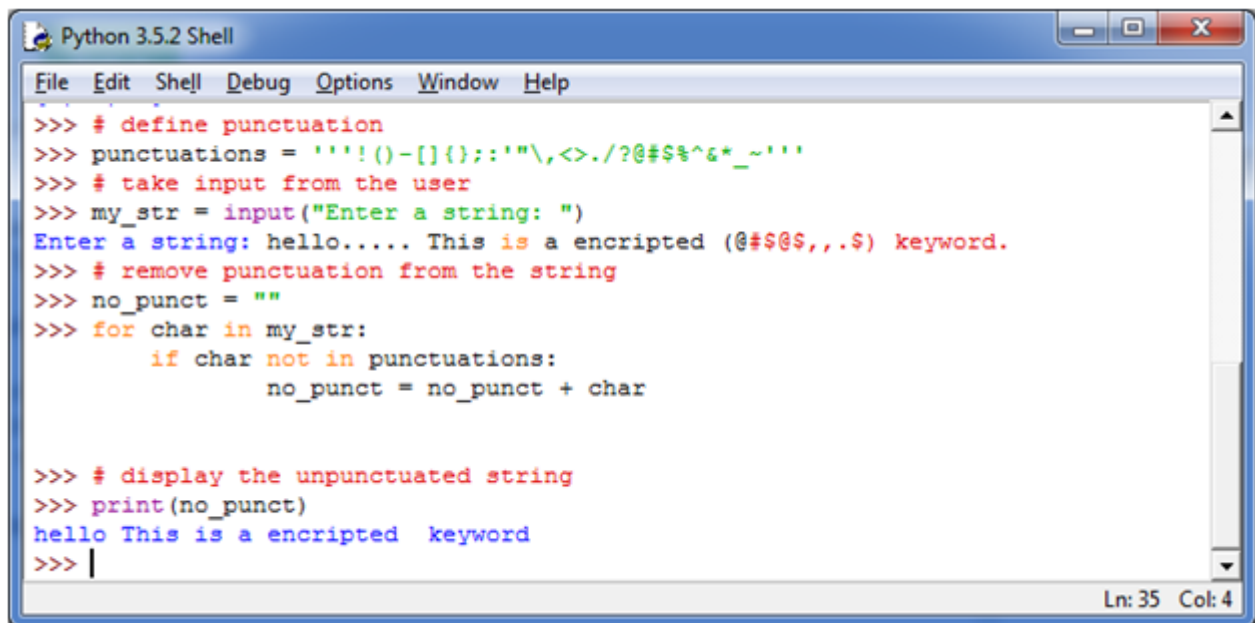
See this example:

```

1.     # define punctuation
2.     punctuation = ""!"()-[]{};:'"\,<>./?@#$$%^&*~_'"
3.     # take input from the user
4.     my_str = input("Enter a string: ")
5.     # remove punctuation from the string
6.     no_punct = ""
7.     for char in my_str:
8.         if char not in punctuation:
9.             no_punct = no_punct + char
10.    # display the unpunctuated string
11.    print(no_punct)

```

Output:



```

Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
>>> # define punctuation
>>> punctuations = ""!"()-[]{};:'"\,<>./?@#$$%^&*~_'"
>>> # take input from the user
>>> my_str = input("Enter a string: ")
Enter a string: hello..... This is a encrypted (@#$$$,,$) keyword.
>>> # remove punctuation from the string
>>> no_punct = ""
>>> for char in my_str:
>>>     if char not in punctuations:
>>>         no_punct = no_punct + char
>>>
>>> # display the unpunctuated string
>>> print(no_punct)
hello This is a encrypted keyword
>>> |
Ln: 35 Col: 4

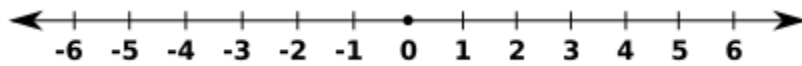
```

PYTHON PROGRAM TO CHECK IF A NUMBER IS POSITIVE, NEGATIVE OR ZERO

We can use a Python program to distinguish that if a number is positive, negative or zero.

Positive Numbers: A number is known as a positive number if it has a greater value than zero. i.e. 1, 2, 3, 4 etc.

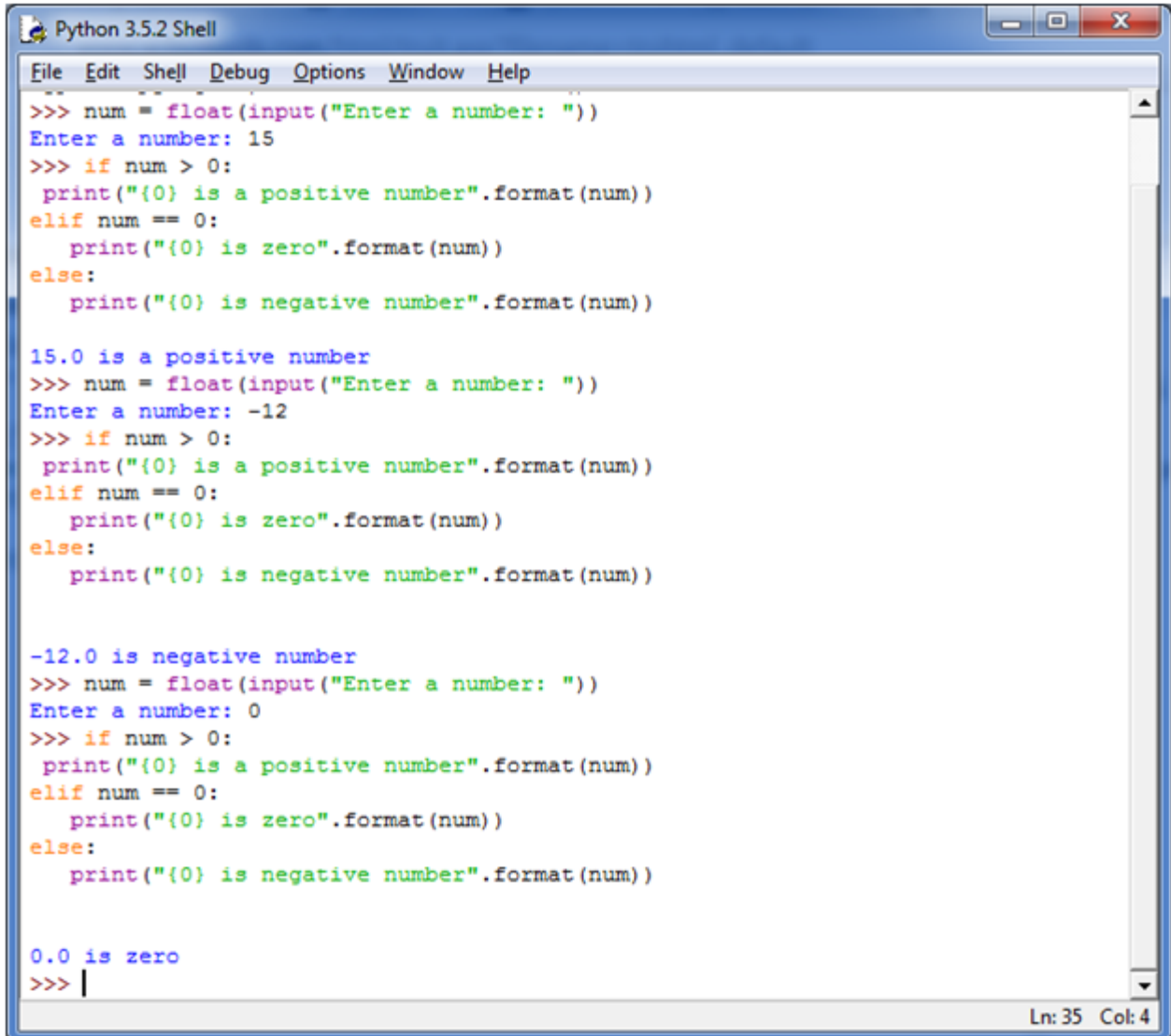
Negative Numbers: A number is known as a negative number if it has a lesser value than zero. i.e. -1, -2, -3, -4 etc.



See this example:

```
1.     num = float(input("Enter a number: "))
2.
3.     if num > 0:
4.         print("{0} is a positive number".format(num))
5.     elif num == 0:
6.         print("{0} is zero".format(num))
7.     else:
8.         print("{0} is negative number".format(num))
```

Output:



```
Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
>>> num = float(input("Enter a number: "))
Enter a number: 15
>>> if num > 0:
    print("{0} is a positive number".format(num))
elif num == 0:
    print("{0} is zero".format(num))
else:
    print("{0} is negative number".format(num))

15.0 is a positive number
>>> num = float(input("Enter a number: "))
Enter a number: -12
>>> if num > 0:
    print("{0} is a positive number".format(num))
elif num == 0:
    print("{0} is zero".format(num))
else:
    print("{0} is negative number".format(num))

-12.0 is negative number
>>> num = float(input("Enter a number: "))
Enter a number: 0
>>> if num > 0:
    print("{0} is a positive number".format(num))
elif num == 0:
    print("{0} is zero".format(num))
else:
    print("{0} is negative number".format(num))

0.0 is zero
>>> |
```

Ln: 35 Col: 4

Note:In the above example, elif statement is used. The elif statement is used to check multiple expressions for TRUE and executes a block of code when one of the conditions becomes TRUE.

It is always followed by if statement.

PYTHON PROGRAM TO CHECK IF A NUMBER IS ODD OR EVEN

Odd and Even numbers:

If you divide a number by 2 and it gives a remainder of 0 then it is known as even number, otherwise an odd number.

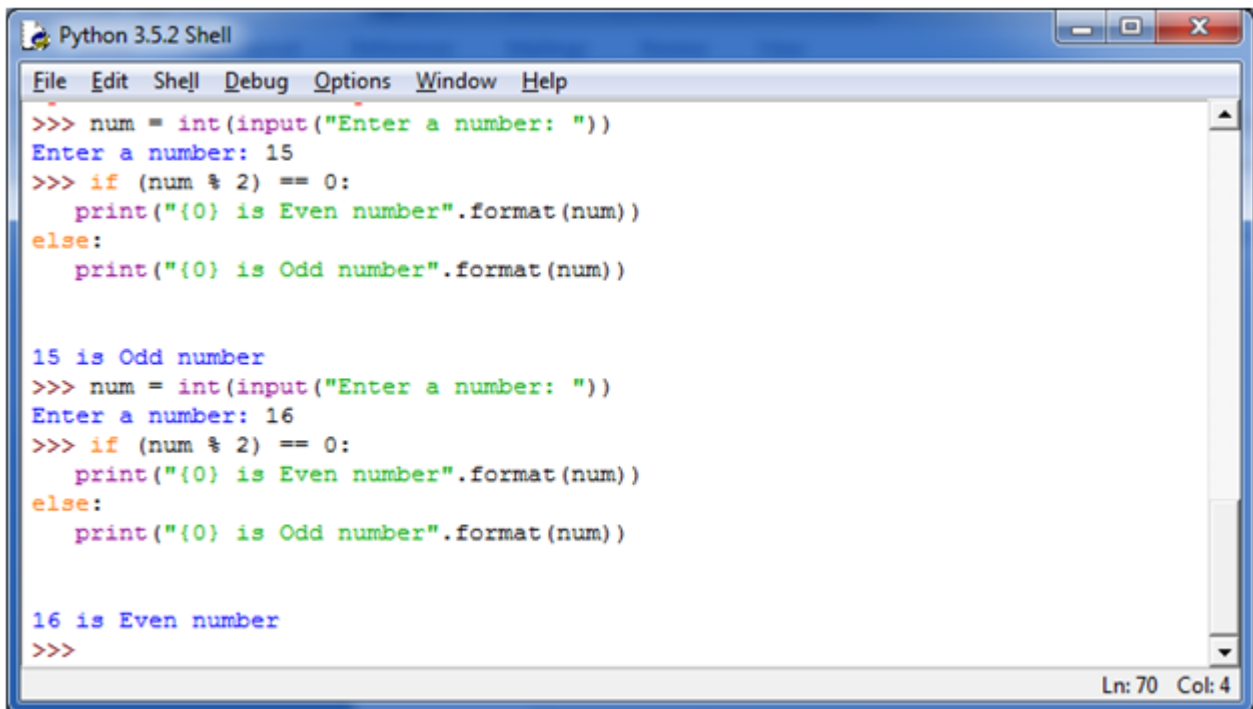
Even number examples: 2, 4, 6, 8, 10, etc.

Odd number examples: 1, 3, 5, 7, 9 etc.

See this example:

```
1.     num = int(input("Enter a number: "))
2.     if (num % 2) == 0:
3.         print("{0} is Even number".format(num))
4.     else:
5.         print("{0} is Odd number".format(num))
```

Output:



```
Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
>>> num = int(input("Enter a number: "))
Enter a number: 15
>>> if (num % 2) == 0:
    print("{0} is Even number".format(num))
else:
    print("{0} is Odd number".format(num))

15 is Odd number
>>> num = int(input("Enter a number: "))
Enter a number: 16
>>> if (num % 2) == 0:
    print("{0} is Even number".format(num))
else:
    print("{0} is Odd number".format(num))

16 is Even number
>>>
```

PYTHON PROGRAM TO CHECK LEAP YEAR

Leap Year:

A year is called a leap year if it contains an additional day which makes the number of the days in that year is 366. This additional day is added in February which makes it 29 days long.

A leap year occurred once every 4 years.

How to determine if a year is a leap year?

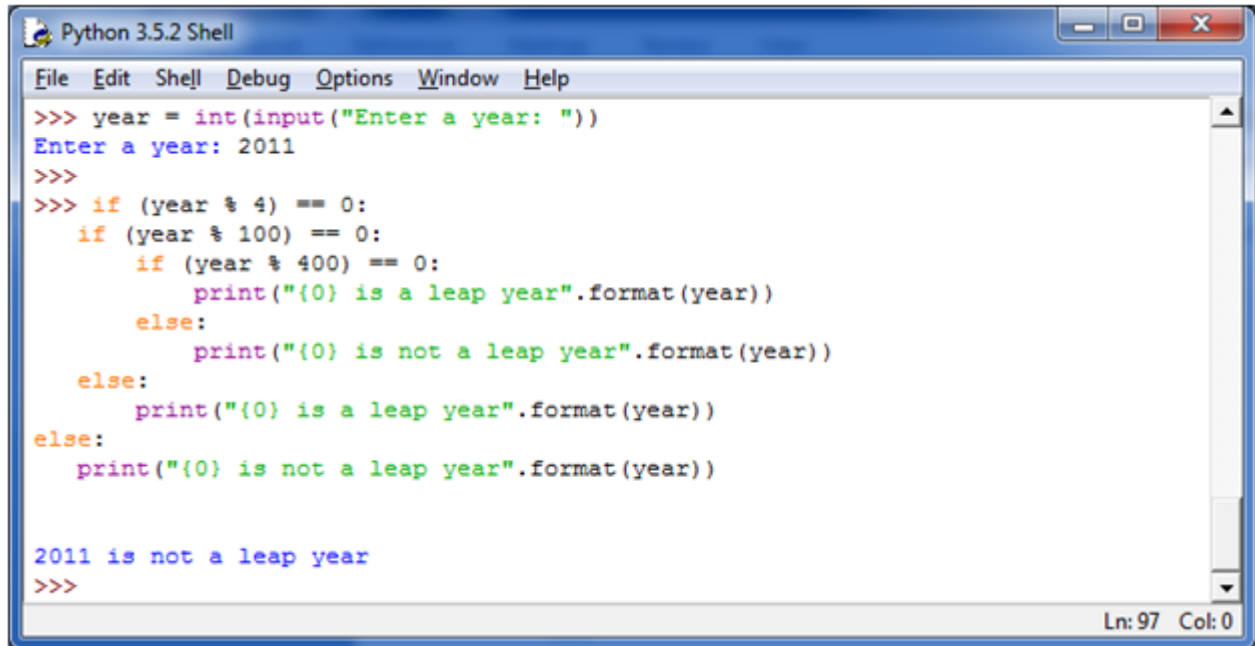
You should follow the following steps to determine whether a year is a leap year or not.

1. If a year is evenly divisible by 4 means having no remainder then go to next step. If it is not divisible by 4. It is not a leap year. For example: 1997 is not a leap year.
2. If a year is divisible by 4, but not by 100. For example: 2012, it is a leap year. If a year is divisible by both 4 and 100, go to next step.
3. If a year is divisible by 100, but not by 400. For example: 1900, then it is not a leap year. If a year is divisible by both, then it is a leap year. So 2000 is a leap year.

See this example:

```
1.     year = int(input("Enter a year: "))
2.     if (year % 4) == 0:
3.         if (year % 100) == 0:
4.             if (year % 400) == 0:
5.                 print("{0} is a leap year".format(year))
6.             else:
7.                 print("{0} is not a leap year".format(year))
8.         else:
9.             print("{0} is a leap year".format(year))
10.    else:
11.        print("{0} is not a leap year".format(year))
```

Output:



```
Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
>>> year = int(input("Enter a year: "))
Enter a year: 2011
>>>
>>> if (year % 4) == 0:
    if (year % 100) == 0:
        if (year % 400) == 0:
            print("{0} is a leap year".format(year))
        else:
            print("{0} is not a leap year".format(year))
    else:
        print("{0} is a leap year".format(year))
else:
    print("{0} is not a leap year".format(year))

2011 is not a leap year
>>>
```

PYTHON PROGRAM TO CHECK PRIME NUMBER

Prime numbers:

A prime number is a natural number greater than 1 and having no positive divisor other than 1 and itself.

For example: 3, 7, 11 etc are prime numbers.

Composite number:

Other natural numbers that are not prime numbers are called composite numbers.

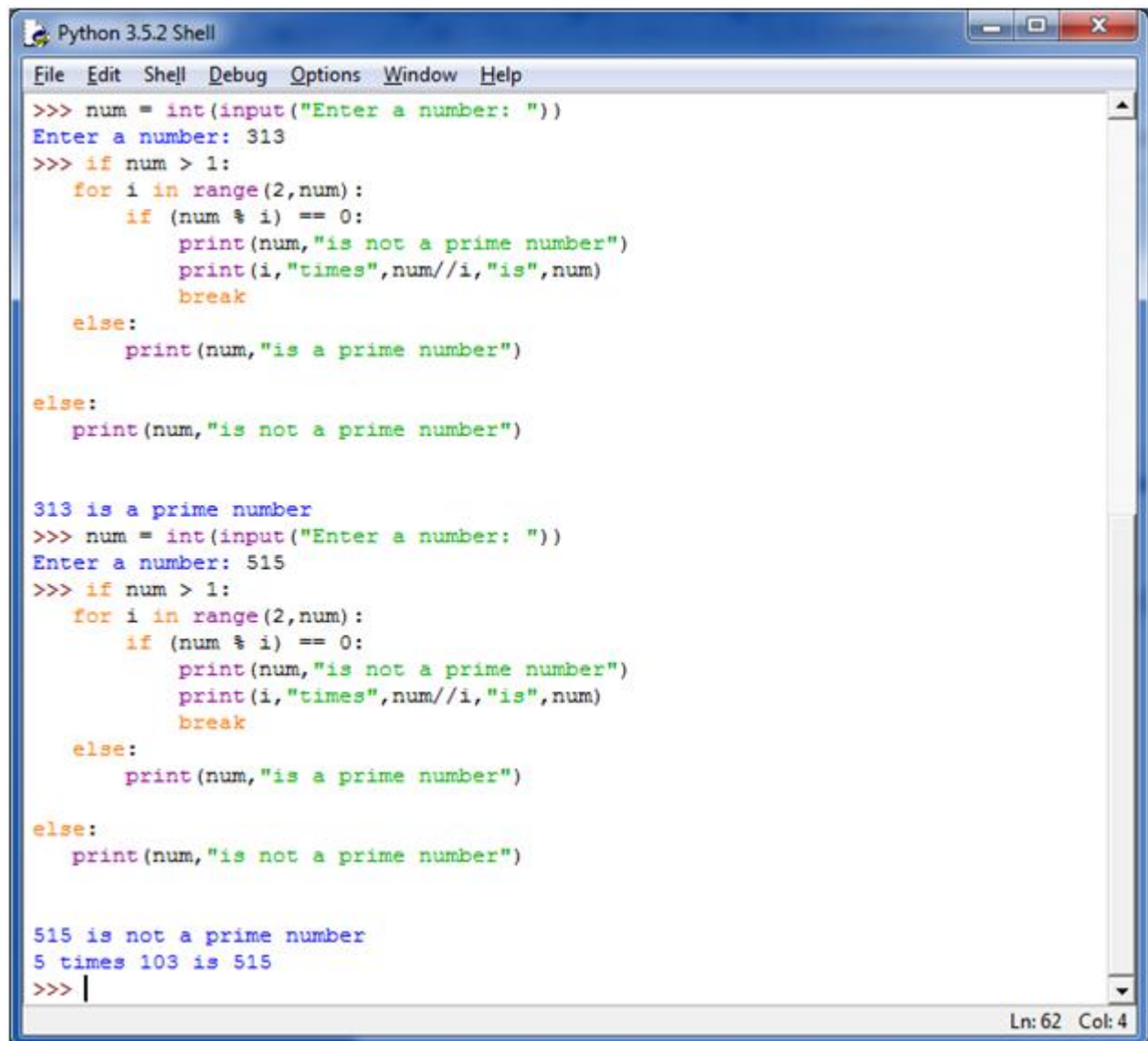
For example: 4, 6, 9 etc. are composite numbers.

See this example:

1. `num = int(input("Enter a number: "))`
- 2.
3. `if num > 1:`
4. `for i in range(2,num):`
5. `if (num % i) == 0:`
6. `print(num,"is not a prime number")`
7. `print(i,"times",num//i,"is",num)`

8. **break**
9. **else:**
10. **print**(num,"is a prime number")
- 11.
12. **else:**
13. **print**(num,"is not a prime number")

Output:



```
Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
>>> num = int(input("Enter a number: "))
Enter a number: 313
>>> if num > 1:
    for i in range(2,num):
        if (num % i) == 0:
            print(num,"is not a prime number")
            print(i,"times",num//i,"is",num)
            break
        else:
            print(num,"is a prime number")
    else:
        print(num,"is not a prime number")

313 is a prime number
>>> num = int(input("Enter a number: "))
Enter a number: 515
>>> if num > 1:
    for i in range(2,num):
        if (num % i) == 0:
            print(num,"is not a prime number")
            print(i,"times",num//i,"is",num)
            break
        else:
            print(num,"is a prime number")
    else:
        print(num,"is not a prime number")

515 is not a prime number
5 times 103 is 515
>>> |
```

Ln: 62 Col: 4

[next](#) → ← [prev](#)

PYTHON PROGRAM TO PRINT ALL PRIME NUMBERS BETWEEN AN INTERVAL

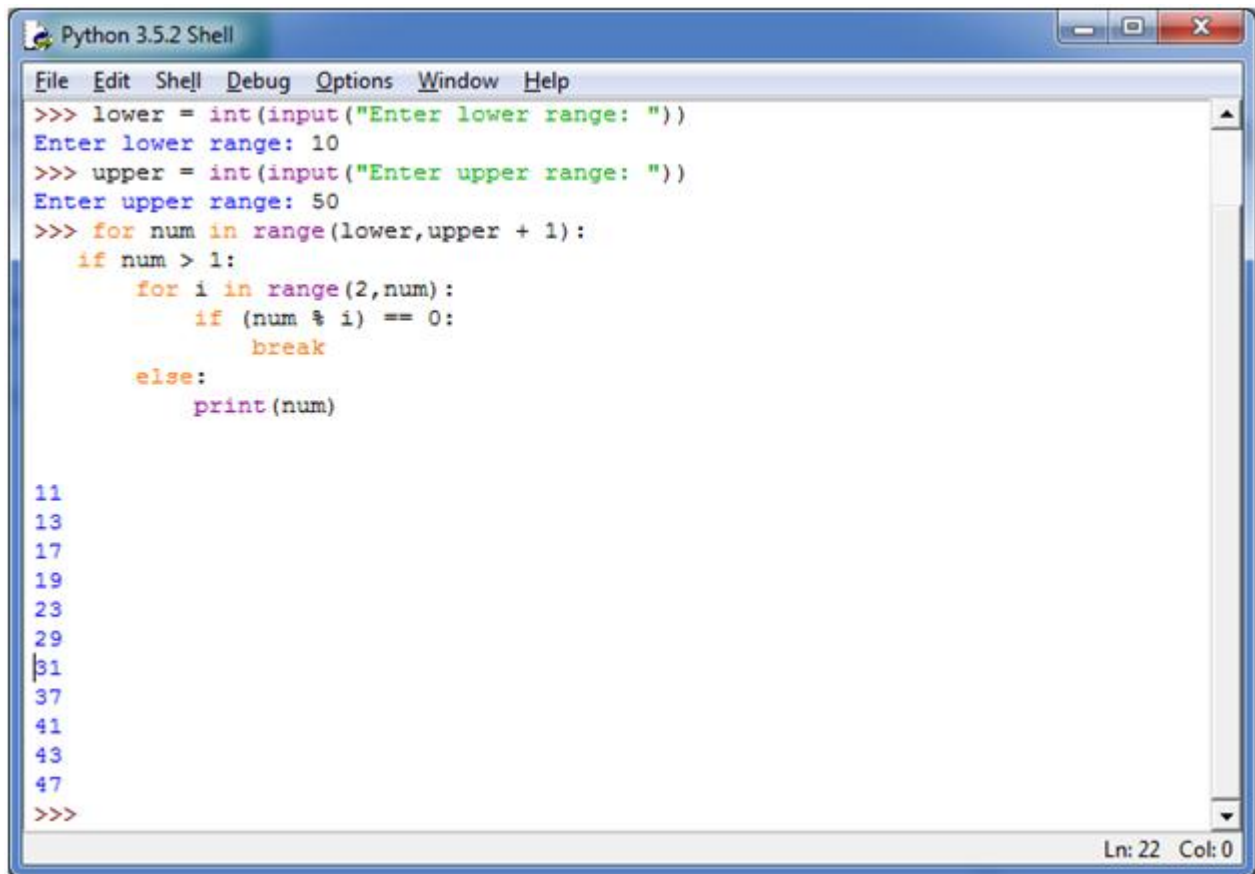
We have already read the concept of prime numbers in the previous program. Here, we are going to print the prime numbers between given interval.

See this example:

```
1.     #Take the input from the user:
2.     lower = int(input("Enter lower range: "))
3.     upper = int(input("Enter upper range: "))
4.
5.     for num in range(lower,upper + 1):
6.         if num > 1:
7.             for i in range(2,num):
8.                 if (num % i) == 0:
9.                     break
10.            else:
11.                print(num)
```

This example will show the prime numbers between 10 and 50.

Output:



```
Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
>>> lower = int(input("Enter lower range: "))
Enter lower range: 10
>>> upper = int(input("Enter upper range: "))
Enter upper range: 50
>>> for num in range(lower, upper + 1):
    if num > 1:
        for i in range(2, num):
            if (num % i) == 0:
                break
        else:
            print(num)

11
13
17
19
23
29
31
37
41
43
47
>>>
```

Ln: 22 Col: 0

PYTHON PROGRAM TO FIND THE FACTORIAL OF A NUMBER

What is factorial?

Factorial is a non-negative integer. It is the product of all positive integers less than or equal to that number for which you ask for factorial. It is denoted by exclamation sign (!).

For example:

1. $4! = 4 \times 3 \times 2 \times 1 = 24$

The factorial value of 4 is 24.

Note: The factorial value of 0 is 1 always. (Rule violation)

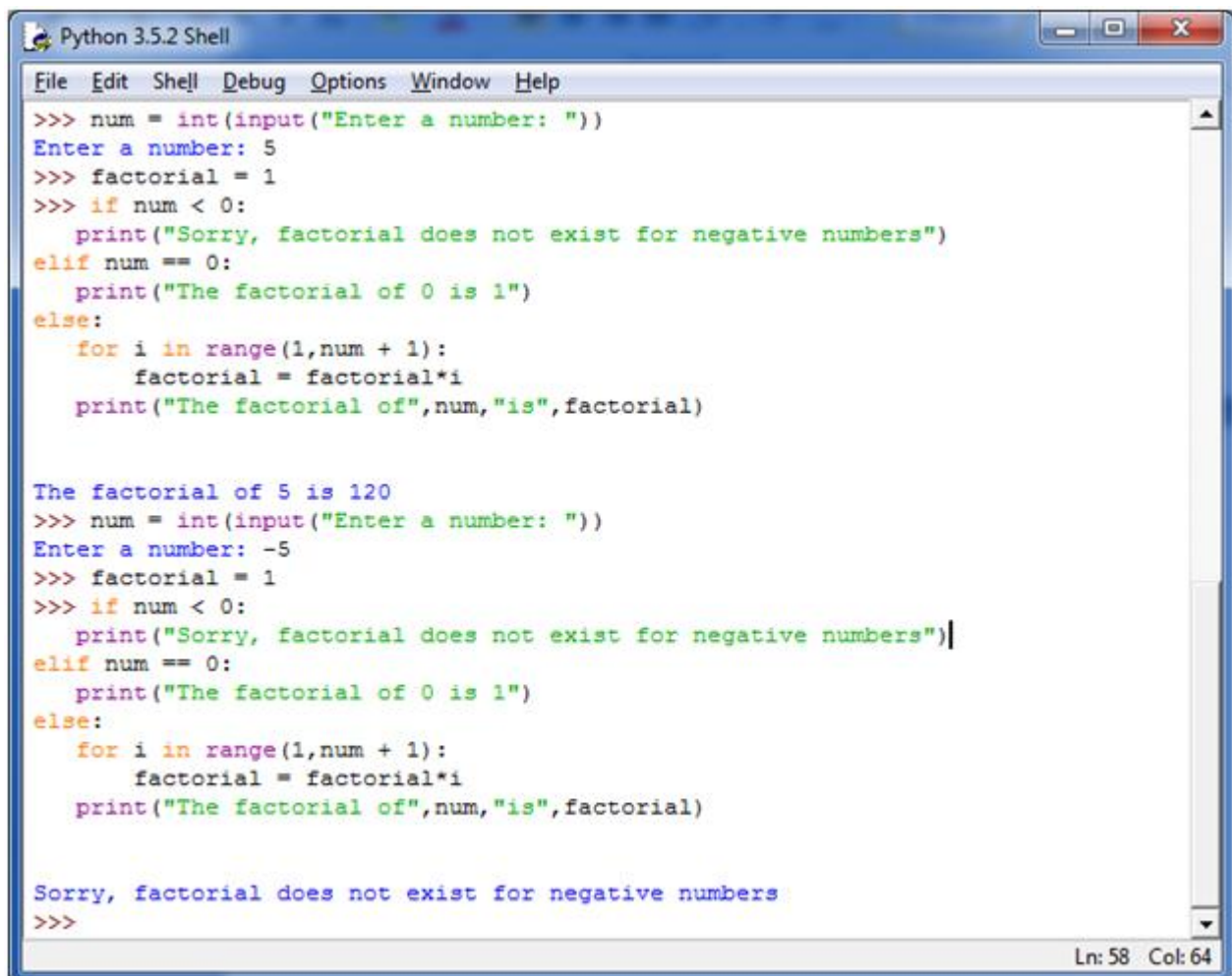
See this example:

1. `num = int(input("Enter a number: "))`
2. `factorial = 1`

```
3.     if num < 0:
4.         print("Sorry, factorial does not exist for negative numbers")
5.     elif num == 0:
6.         print("The factorial of 0 is 1")
7.     else:
8.         for i in range(1,num + 1):
9.             factorial = factorial*i
10.        print("The factorial of",num,"is",factorial)
```

The following example displays the factorial of 5 and -5.

Output:



```
Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
>>> num = int(input("Enter a number: "))
Enter a number: 5
>>> factorial = 1
>>> if num < 0:
    print("Sorry, factorial does not exist for negative numbers")
elif num == 0:
    print("The factorial of 0 is 1")
else:
    for i in range(1,num + 1):
        factorial = factorial*i
    print("The factorial of",num,"is",factorial)

The factorial of 5 is 120
>>> num = int(input("Enter a number: "))
Enter a number: -5
>>> factorial = 1
>>> if num < 0:
    print("Sorry, factorial does not exist for negative numbers")
elif num == 0:
    print("The factorial of 0 is 1")
else:
    for i in range(1,num + 1):
        factorial = factorial*i
    print("The factorial of",num,"is",factorial)

Sorry, factorial does not exist for negative numbers
>>>
```

Ln: 58 Col: 64

PYTHON PROGRAM TO DISPLAY THE MULTIPLICATION TABLE

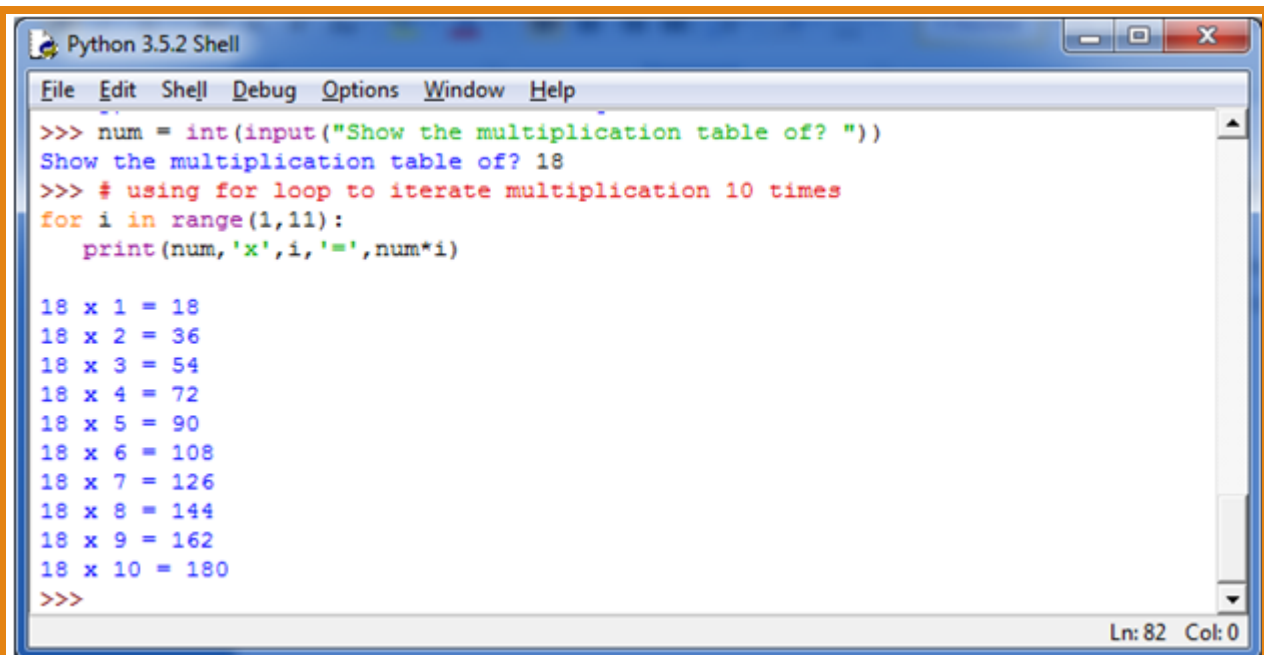
In Python, you can make a program to display the multiplication table of any number. The following program displays the multiplication table (from 1 to 10) according to the user input.

See this example:

1. `num = int(input("Show the multiplication table of? "))`
2. `# using for loop to iterate multiplication 10 times`
3. `for i in range(1,11):`
4. `print(num,'x',i,'=',num*i)`

The following example shows the multiplication table of 18.

Output:



```
Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
>>> num = int(input("Show the multiplication table of? "))
Show the multiplication table of? 18
>>> # using for loop to iterate multiplication 10 times
for i in range(1,11):
    print(num,'x',i,'=',num*i)

18 x 1 = 18
18 x 2 = 36
18 x 3 = 54
18 x 4 = 72
18 x 5 = 90
18 x 6 = 108
18 x 7 = 126
18 x 8 = 144
18 x 9 = 162
18 x 10 = 180
>>>
```

PYTHON PROGRAM TO PRINT THE FIBONACCI SEQUENCE

Fibonacci sequence:

The Fibonacci sequence specifies a series of numbers where the next number is found by adding up the two numbers just before it.

For example:

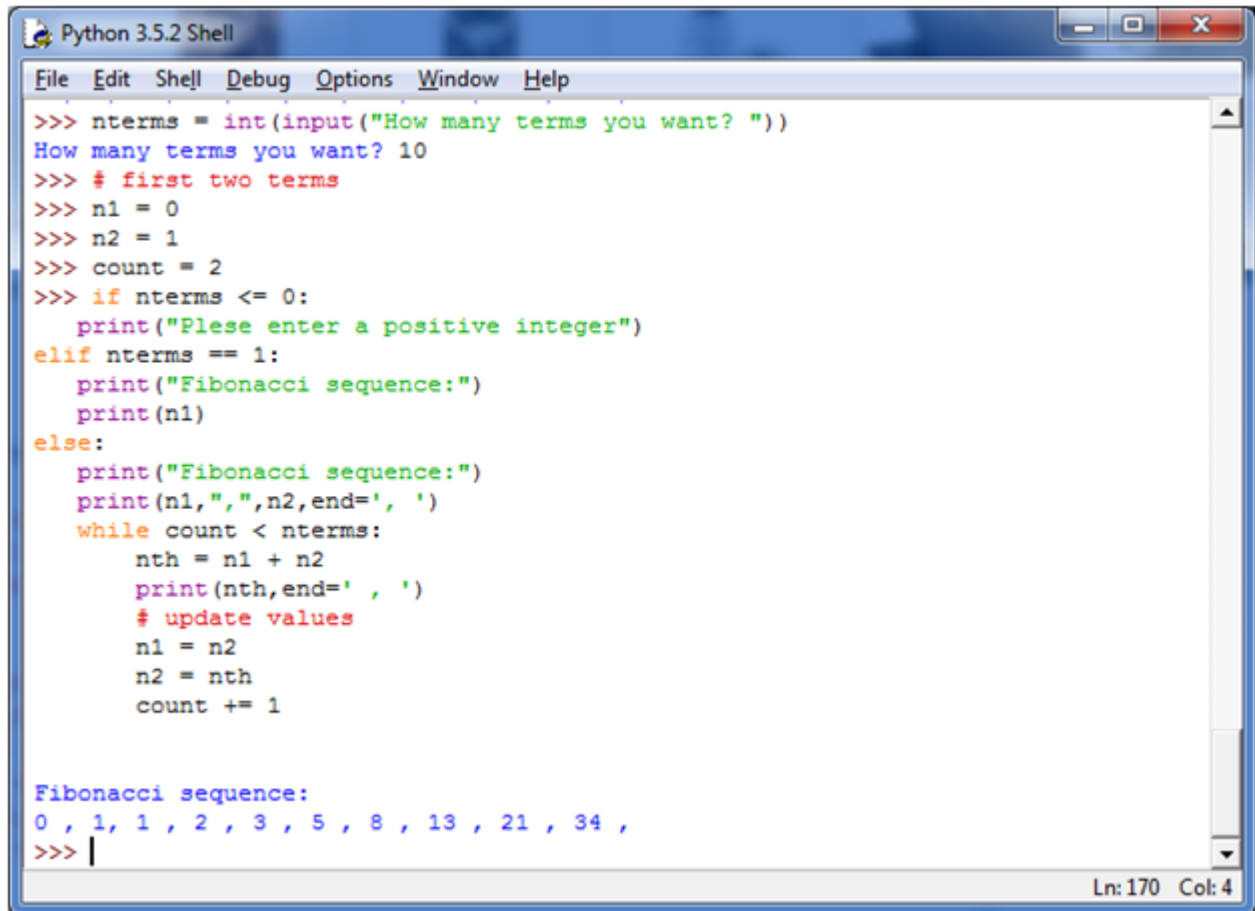
- 1.

2. 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, **and** so on....

See this example:

```
1.       nterms = int(input("How many terms you want? "))
2.       # first two terms
3.       n1 = 0
4.       n2 = 1
5.       count = 2
6.       # check if the number of terms is valid
7.       if nterms <= 0:
8.           print("Plese enter a positive integer")
9.       elif nterms == 1:
10.          print("Fibonacci sequence:")
11.          print(n1)
12.       else:
13.          print("Fibonacci sequence:")
14.          print(n1, ",", n2, end=', ')
15.          while count < nterms:
16.              nth = n1 + n2
17.              print(nth, end=', ')
18.              # update values
19.              n1 = n2
20.              n2 = nth
21.              count += 1
```

Output:

A screenshot of a Python 3.5.2 Shell window. The window has a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area contains the following code:

```
>>> nterms = int(input("How many terms you want? "))
How many terms you want? 10
>>> # first two terms
>>> n1 = 0
>>> n2 = 1
>>> count = 2
>>> if nterms <= 0:
    print("Plese enter a positive integer")
elif nterms == 1:
    print("Fibonacci sequence:")
    print(n1)
else:
    print("Fibonacci sequence:")
    print(n1, ",", n2, end=', ')
    while count < nterms:
        nth = n1 + n2
        print(nth, end=', ')
        # update values
        n1 = n2
        n2 = nth
        count += 1
```

The output of the program is displayed at the bottom:

```
Fibonacci sequence:
0 , 1 , 1 , 2 , 3 , 5 , 8 , 13 , 21 , 34 ,
>>> |
```

The status bar at the bottom right shows 'Ln: 170 Col: 4'.

PYTHON PROGRAM TO CHECK ARMSTRONG NUMBER

Armstrong number:

A number is called Armstrong number if it is equal to the sum of the cubes of its own digits.

For example: 153 is an Armstrong number since $153 = 1*1*1 + 5*5*5 + 3*3*3$.

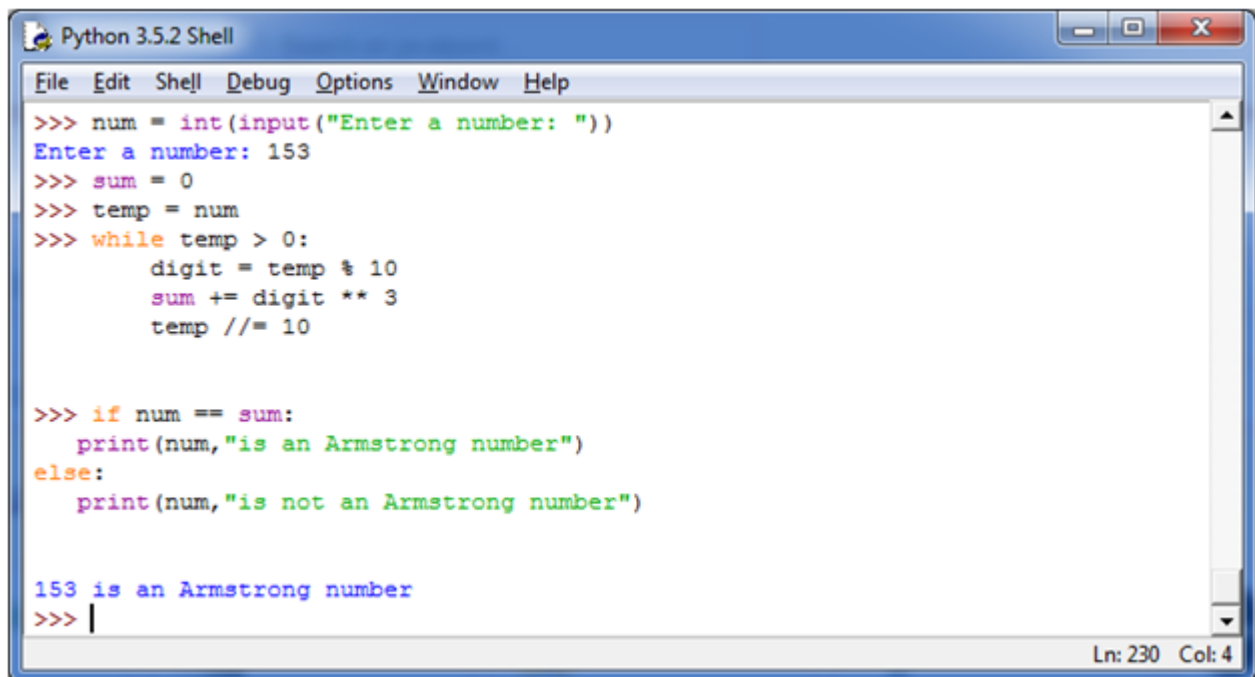
The Armstrong number is also known as **narcissistic** number.

See this example:

1. `num = int(input("Enter a number: "))`
2. `sum = 0`
3. `temp = num`
- 4.
5. `while temp > 0:`

```
6.         digit = temp % 10
7.         sum += digit ** 3
8.         temp //= 10
9.
10.        if num == sum:
11.            print(num,"is an Armstrong number")
12.        else:
13.            print(num,"is not an Armstrong number")
```

Output:



```
Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
>>> num = int(input("Enter a number: "))
Enter a number: 153
>>> sum = 0
>>> temp = num
>>> while temp > 0:
>>>     digit = temp % 10
>>>     sum += digit ** 3
>>>     temp //= 10
>>>
>>> if num == sum:
>>>     print(num,"is an Armstrong number")
>>> else:
>>>     print(num,"is not an Armstrong number")
>>>
153 is an Armstrong number
>>> |
```

Ln: 230 Col: 4

PYTHON PROGRAM TO FIND ARMSTRONG NUMBER BETWEEN AN INTERVAL

We have already read the concept of Armstrong numbers in the previous program. Here, we print the Armstrong numbers within a specific given interval.

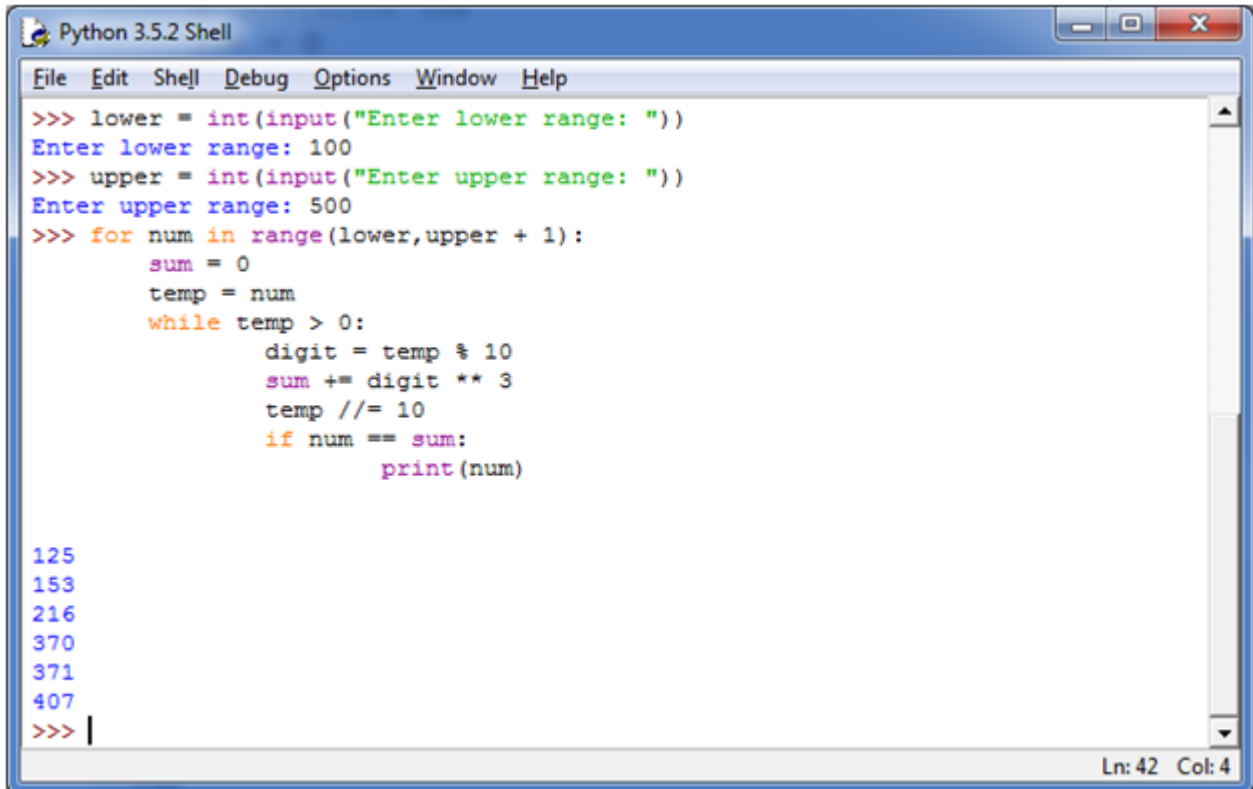
See this example:

```
1.         lower = int(input("Enter lower range: "))
2.         upper = int(input("Enter upper range: "))
```

```
3.
4.     for num in range(lower,upper + 1):
5.         sum = 0
6.         temp = num
7.         while temp > 0:
8.             digit = temp % 10
9.             sum += digit ** 3
10.            temp //= 10
11.            if num == sum:
12.                print(num)
```

This example shows all Armstrong numbers between 100 and 500.

Output:



```
Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
>>> lower = int(input("Enter lower range: "))
Enter lower range: 100
>>> upper = int(input("Enter upper range: "))
Enter upper range: 500
>>> for num in range(lower,upper + 1):
    sum = 0
    temp = num
    while temp > 0:
        digit = temp % 10
        sum += digit ** 3
        temp //= 10
    if num == sum:
        print(num)

125
153
216
370
371
407
>>> |
```

Ln: 42 Col: 4

PYTHON PROGRAM TO FIND THE SUM OF NATURAL NUMBERS

Natural numbers:

As the name specifies, a natural number is the number that occurs commonly and obviously in the nature. It is a whole, non-negative number.

Some mathematicians think that a natural number must contain 0 and some don't believe this theory. So, a list of natural number can be defined as:

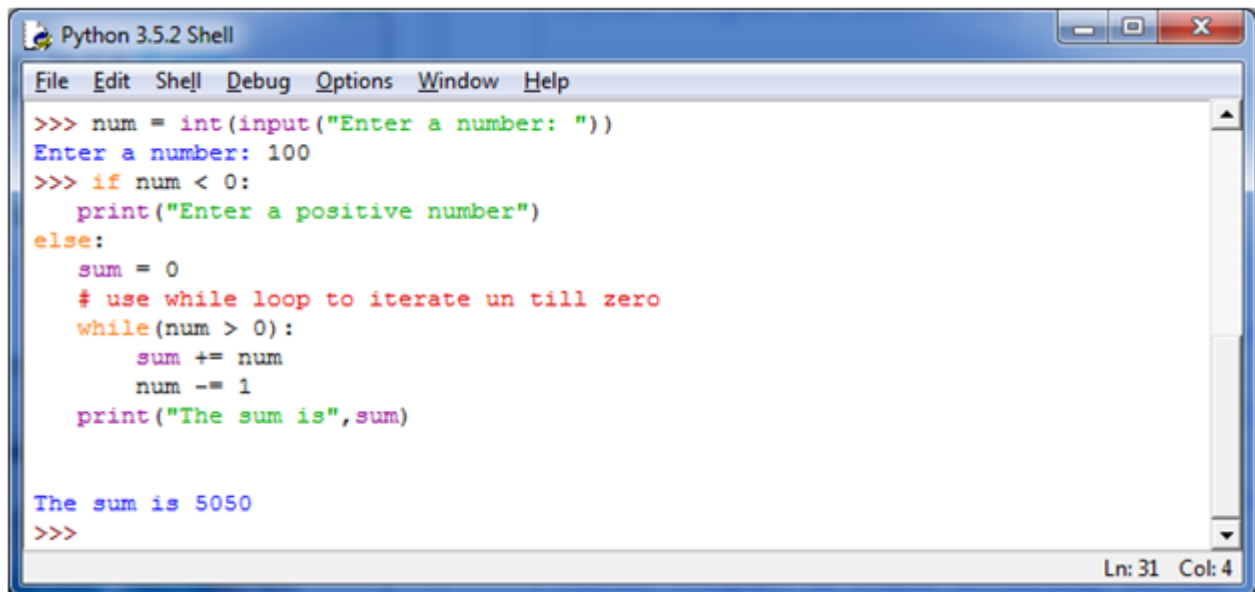
- 1.
2. $N = \{0, 1, 2, 3, 4, \dots \text{ and so on}\}$
3. $N = \{1, 2, 3, 4, \dots \text{ and so on}\}$

See this example:

```
1.     num = int(input("Enter a number: "))
2.
3.     if num < 0:
4.         print("Enter a positive number")
5.     else:
6.         sum = 0
7.         # use while loop to iterate un till zero
8.         while(num > 0):
9.             sum += num
10.            num -= 1
11.        print("The sum is",sum)
```

This example shows the sum of the first 100 positive numbers (0-100)

Output:



```
Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
>>> num = int(input("Enter a number: "))
Enter a number: 100
>>> if num < 0:
    print("Enter a positive number")
else:
    sum = 0
    # use while loop to iterate un till zero
    while(num > 0):
        sum += num
        num -= 1
    print("The sum is",sum)

The sum is 5050
>>>
```

PYTHON PROGRAM TO FIND HCF

HCF: Highest Common Factor

Highest Common Factor or Greatest Common Divisor of two or more integers when at least one of them is not zero is the largest positive integer that evenly divides the numbers without a remainder. For example, the GCD of 8 and 12 is 4.

For example:

We have two integers 8 and 12. Let's find the HCF.

The divisors of 8 are:

1. 1, 2, 4, 8

The divisors of 12 are:

1. 1, 2, 3, 4, 6, 12

HCF /GCD is the greatest common divisor. So HCF of 8 and 12 are 4.

See this example:

1. **def** hcf(x, y):
2. **if** x > y:
3. **smaller** = y
4. **else:**

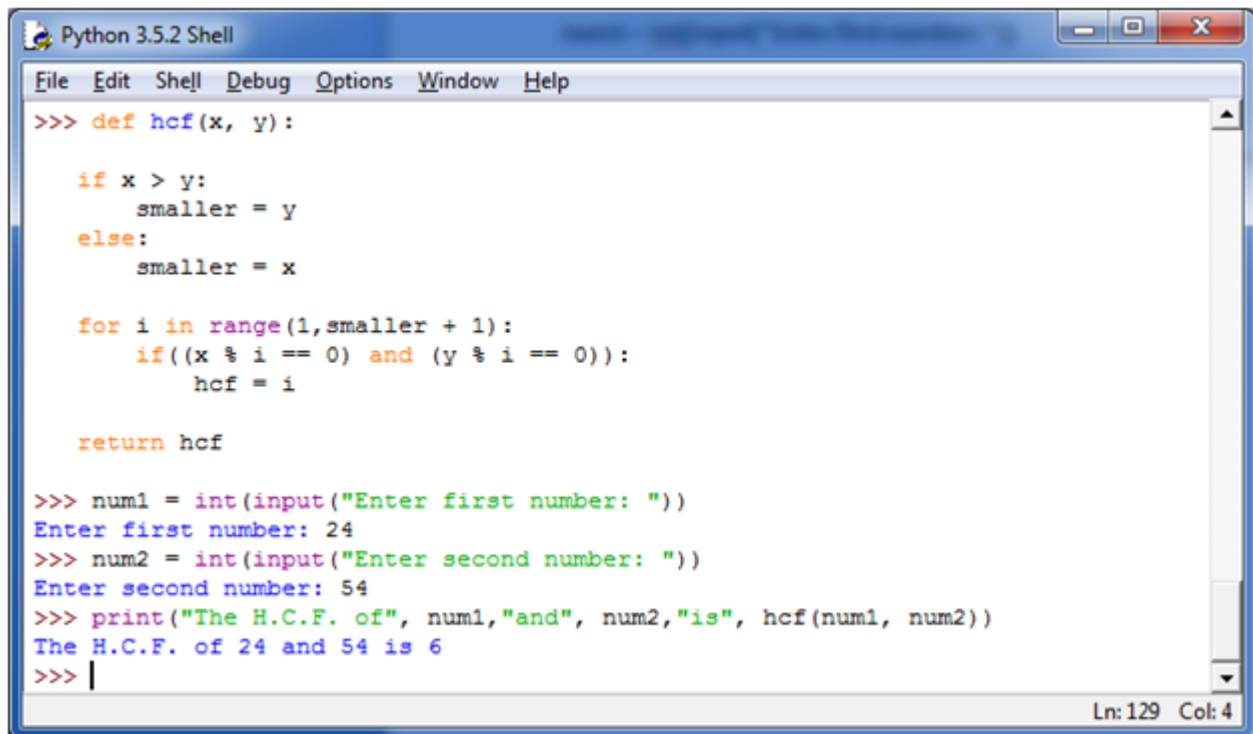
```

5.         smaller = x
6.         for i in range(1,smaller + 1):
7.             if((x % i == 0) and (y % i == 0)):
8.                 hcf = i
9.         return hcf
10.
11.     num1 = int(input("Enter first number: "))
12.     num2 = int(input("Enter second number: "))
13.     print("The H.C.F. of", num1,"and", num2,"is", hcf(num1, num2))

```

The following example shows the HCF of 24 and 54. (according to user input)

Output:



```

Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
>>> def hcf(x, y):
    if x > y:
        smaller = y
    else:
        smaller = x
    for i in range(1,smaller + 1):
        if((x % i == 0) and (y % i == 0)):
            hcf = i
    return hcf
>>> num1 = int(input("Enter first number: "))
Enter first number: 24
>>> num2 = int(input("Enter second number: "))
Enter second number: 54
>>> print("The H.C.F. of", num1,"and", num2,"is", hcf(num1, num2))
The H.C.F. of 24 and 54 is 6
>>> |
Ln: 129 Col: 4

```

PYTHON PROGRAM TO CONVERT DECIMAL TO BINARY, OCTAL AND HEXADECIMAL

Decimal System: The most widely used number system is decimal system. This system is base 10 number system. In this system, ten numbers (0-9) are used to represent a number.

Binary System: Binary system is base 2 number system. Binary system is used because computers only understand binary numbers (0 and 1).

Octal System: Octal system is base 8 number system.

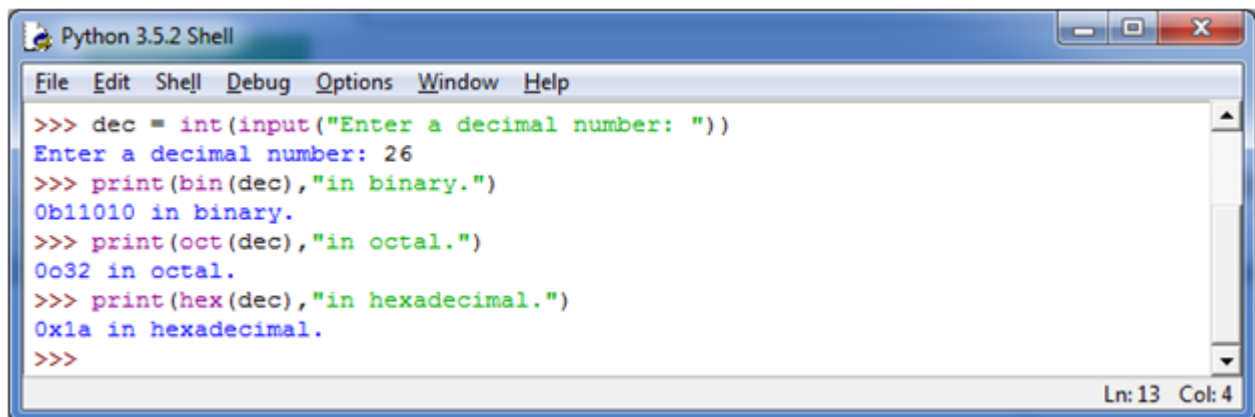
Hexadecimal System: Hexadecimal system is base 16 number system.

This program is written to convert decimal to binary, octal and hexadecimal.

See this example:

1. `dec = int(input("Enter a decimal number: "))`
- 2.
3. `print(bin(dec),"in binary.")`
4. `print(oct(dec),"in octal.")`
5. `print(hex(dec),"in hexadecimal.")`

Output:



```
Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
>>> dec = int(input("Enter a decimal number: "))
Enter a decimal number: 26
>>> print(bin(dec),"in binary.")
0b11010 in binary.
>>> print(oct(dec),"in octal.")
0o32 in octal.
>>> print(hex(dec),"in hexadecimal.")
0x1a in hexadecimal.
>>>
```

PYTHON PROGRAM TO FIND ASCII VALUE OF A CHARACTER

ASCII: ASCII is an acronym stands for **American Standard Code for Information Interchange**. In ASCII, a specific numerical value is given to different characters and symbols, for computers to store and manipulate.

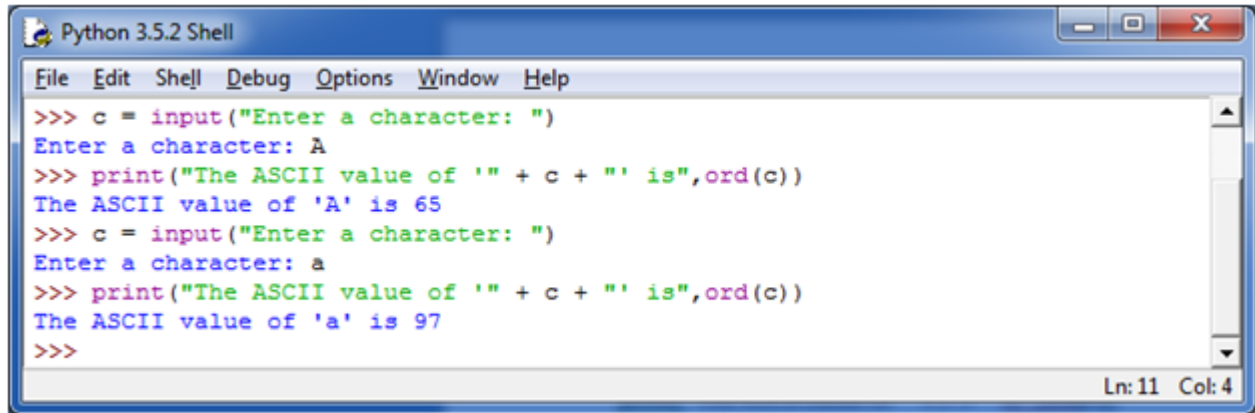
It is case sensitive. Same character, having different format (upper case and lower case) has different value. For example: The ASCII value of "A" is 65 while the ASCII value of "a" is 97.

See this example:

1. `c = input("Enter a character: ")`

- 2.
3. `print("The ASCII value of '" + c + "' is",ord(c))`

Output:



```
Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
>>> c = input("Enter a character: ")
Enter a character: A
>>> print("The ASCII value of '" + c + "' is",ord(c))
The ASCII value of 'A' is 65
>>> c = input("Enter a character: ")
Enter a character: a
>>> print("The ASCII value of '" + c + "' is",ord(c))
The ASCII value of 'a' is 97
>>>
```

PYTHON PROGRAM TO MAKE A SIMPLE CALCULATOR

In Python, you can create a simple calculator, displaying the different arithmetical operations i.e. addition, subtraction, multiplication and division.

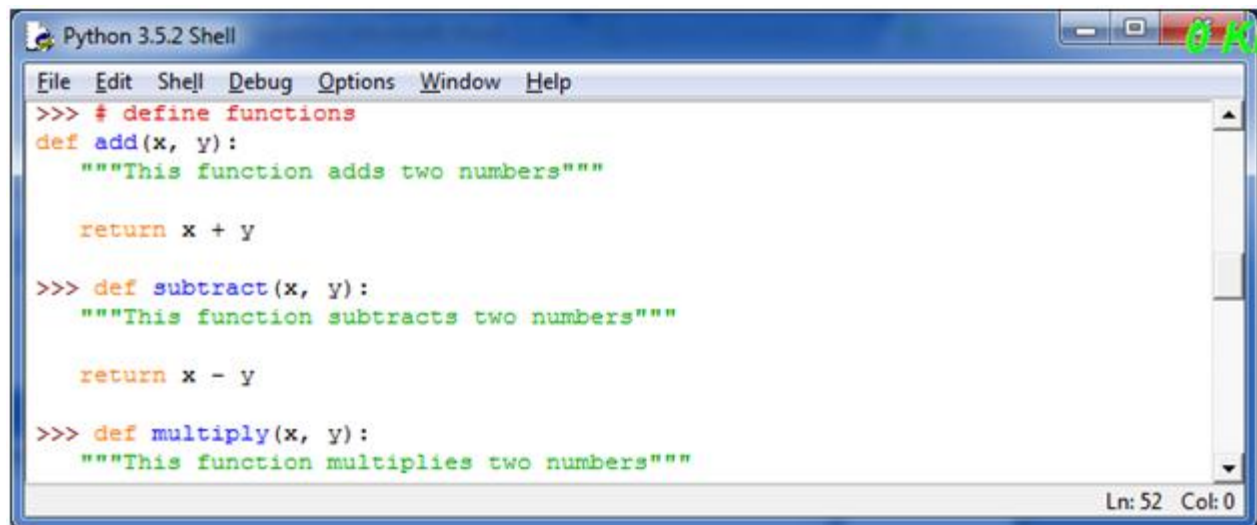
The following program is intended to write a simple calculator in Python:

See this example:

1. `# define functions`
2. `def add(x, y):`
3. `"""This function adds two numbers"""`
4. `return x + y`
5. `def subtract(x, y):`
6. `"""This function subtracts two numbers"""`
7. `return x - y`
8. `def multiply(x, y):`
9. `"""This function multiplies two numbers"""`
10. `return x * y`
11. `def divide(x, y):`
12. `"""This function divides two numbers"""`
13. `return x / y`
14. `# take input from the user`

```
15.     print("Select operation.")
16.     print("1.Add")
17.     print("2.Subtract")
18.     print("3.Multiply")
19.     print("4.Divide")
20.
21.     choice = input("Enter choice(1/2/3/4):")
22.
23.     num1 = int(input("Enter first number: "))
24.     num2 = int(input("Enter second number: "))
25.
26.     if choice == '1':
27.         print(num1,"+",num2,"=", add(num1,num2))
28.
29.     elif choice == '2':
30.         print(num1,"-",num2,"=", subtract(num1,num2))
31.
32.     elif choice == '3':
33.         print(num1,"*",num2,"=", multiply(num1,num2))
34.     elif choice == '4':
35.         print(num1,"/",num2,"=", divide(num1,num2))
36.     else:
37.         print("Invalid input")
```

Output:



A screenshot of a Python 3.5.2 Shell window. The window has a blue title bar with the text "Python 3.5.2 Shell" and standard window controls. Below the title bar is a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The main area contains Python code defining three functions: `add`, `subtract`, and `multiply`. Each function has a docstring. The code is color-coded: red for comments, blue for function definitions, green for docstrings, and orange for return statements. A status bar at the bottom right shows "Ln: 52 Col: 0".

```
>>> # define functions
def add(x, y):
    """This function adds two numbers"""

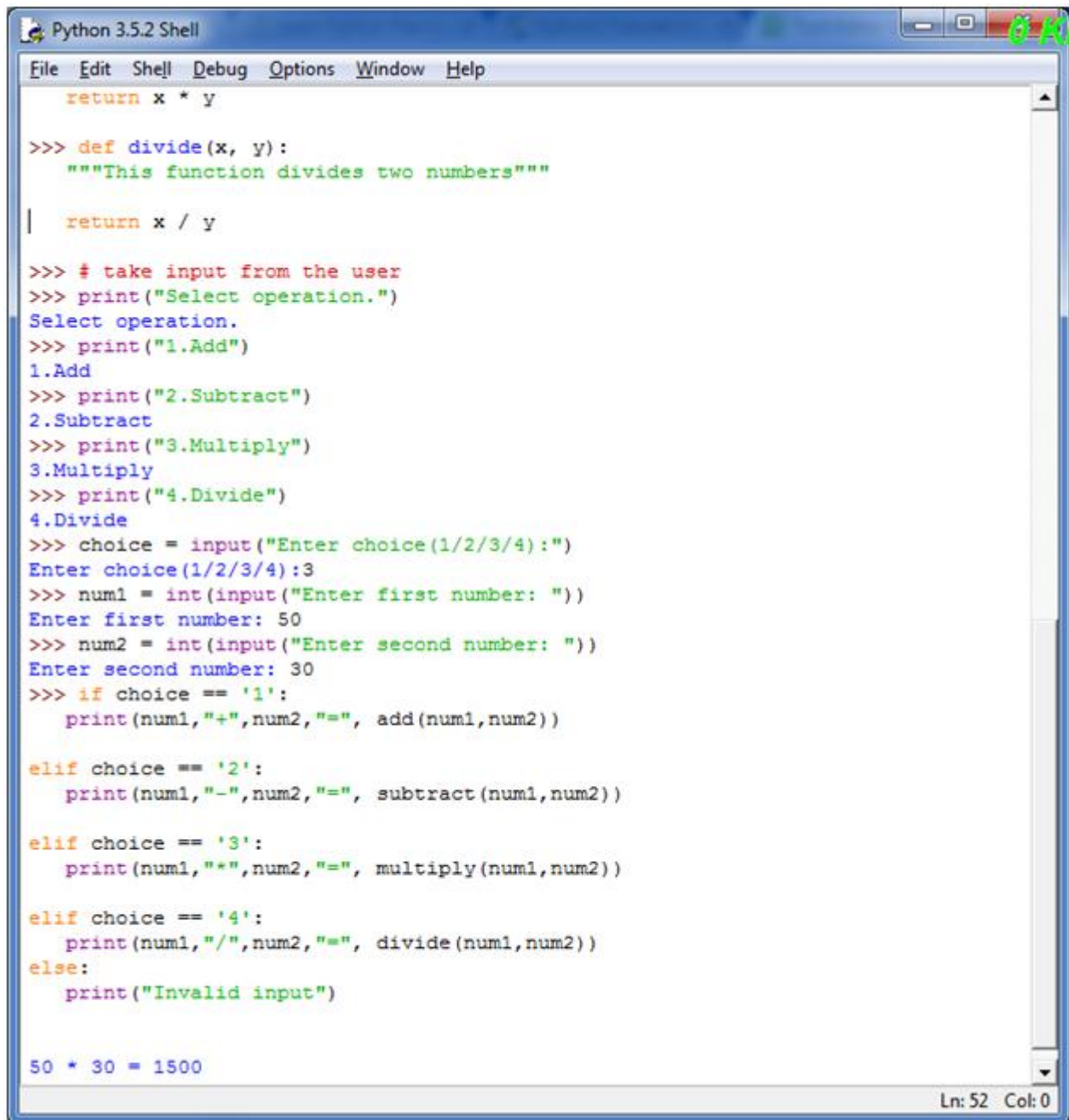
    return x + y

>>> def subtract(x, y):
    """This function subtracts two numbers"""

    return x - y

>>> def multiply(x, y):
    """This function multiplies two numbers"""
```

Ln: 52 Col: 0



```
Python 3.5.2 Shell
File Edit Shell Debug Options Window Help

return x * y

>>> def divide(x, y):
    """This function divides two numbers"""
    return x / y

>>> # take input from the user
>>> print("Select operation.")
Select operation.
>>> print("1.Add")
1.Add
>>> print("2.Subtract")
2.Subtract
>>> print("3.Multiply")
3.Multiply
>>> print("4.Divide")
4.Divide
>>> choice = input("Enter choice(1/2/3/4):")
Enter choice(1/2/3/4):3
>>> num1 = int(input("Enter first number: "))
Enter first number: 50
>>> num2 = int(input("Enter second number: "))
Enter second number: 30
>>> if choice == '1':
    print(num1,"+",num2,"=", add(num1,num2))

elif choice == '2':
    print(num1,"-",num2,"=", subtract(num1,num2))

elif choice == '3':
    print(num1,"*",num2,"=", multiply(num1,num2))

elif choice == '4':
    print(num1,"/",num2,"=", divide(num1,num2))
else:
    print("Invalid input")

50 * 30 = 1500

Ln: 52 Col: 0
```

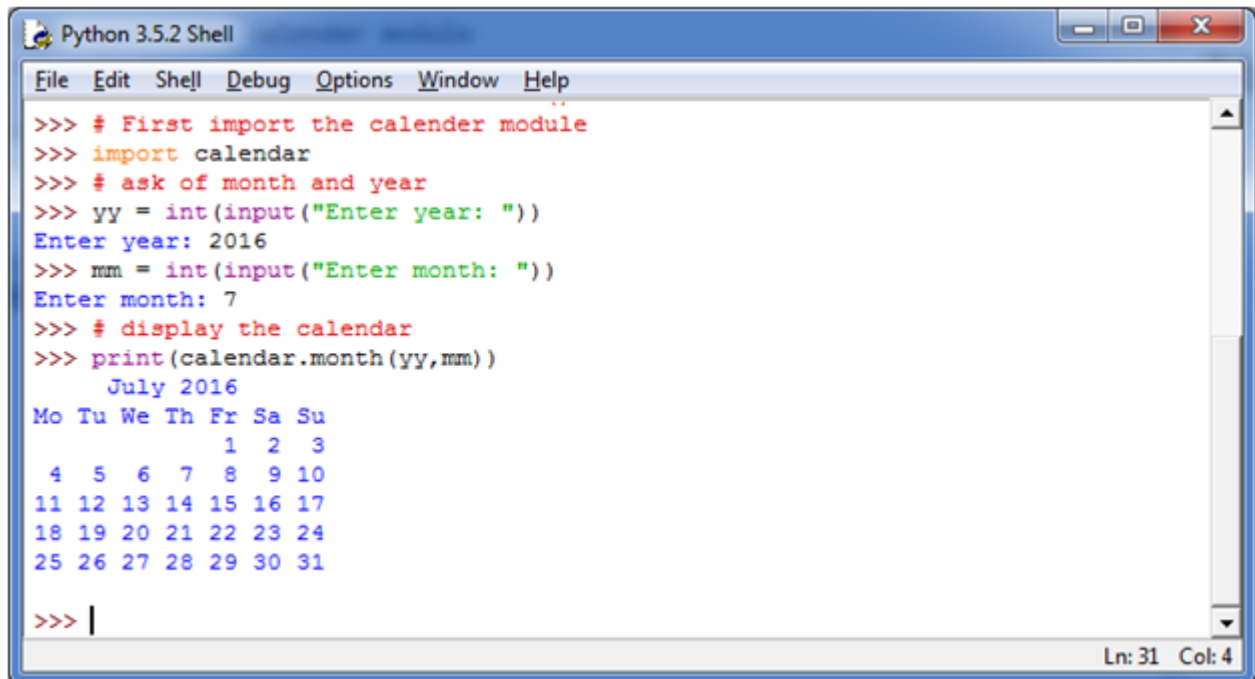
PYTHON FUNCTION TO DISPLAY CALENDAR

In Python, we can display the calendar of any month of any year by importing the calendar module.

See this example:

1. `# First import the calendar module`
2. `import calendar`
3. `# ask of month and year`
4. `yy = int(input("Enter year: "))`
5. `mm = int(input("Enter month: "))`
6. `# display the calendar`
7. `print(calendar.month(yy,mm))`

Output:



```
Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
>>> # First import the calendar module
>>> import calendar
>>> # ask of month and year
>>> yy = int(input("Enter year: "))
Enter year: 2016
>>> mm = int(input("Enter month: "))
Enter month: 7
>>> # display the calendar
>>> print(calendar.month(yy,mm))
    July 2016
Mo Tu We Th Fr Sa Su
                1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31

>>> |
```

Ln: 31 Col: 4

PYTHON PROGRAM TO DISPLAY FIBONACCI SEQUENCE USING RECURSION

Fibonacci sequence:

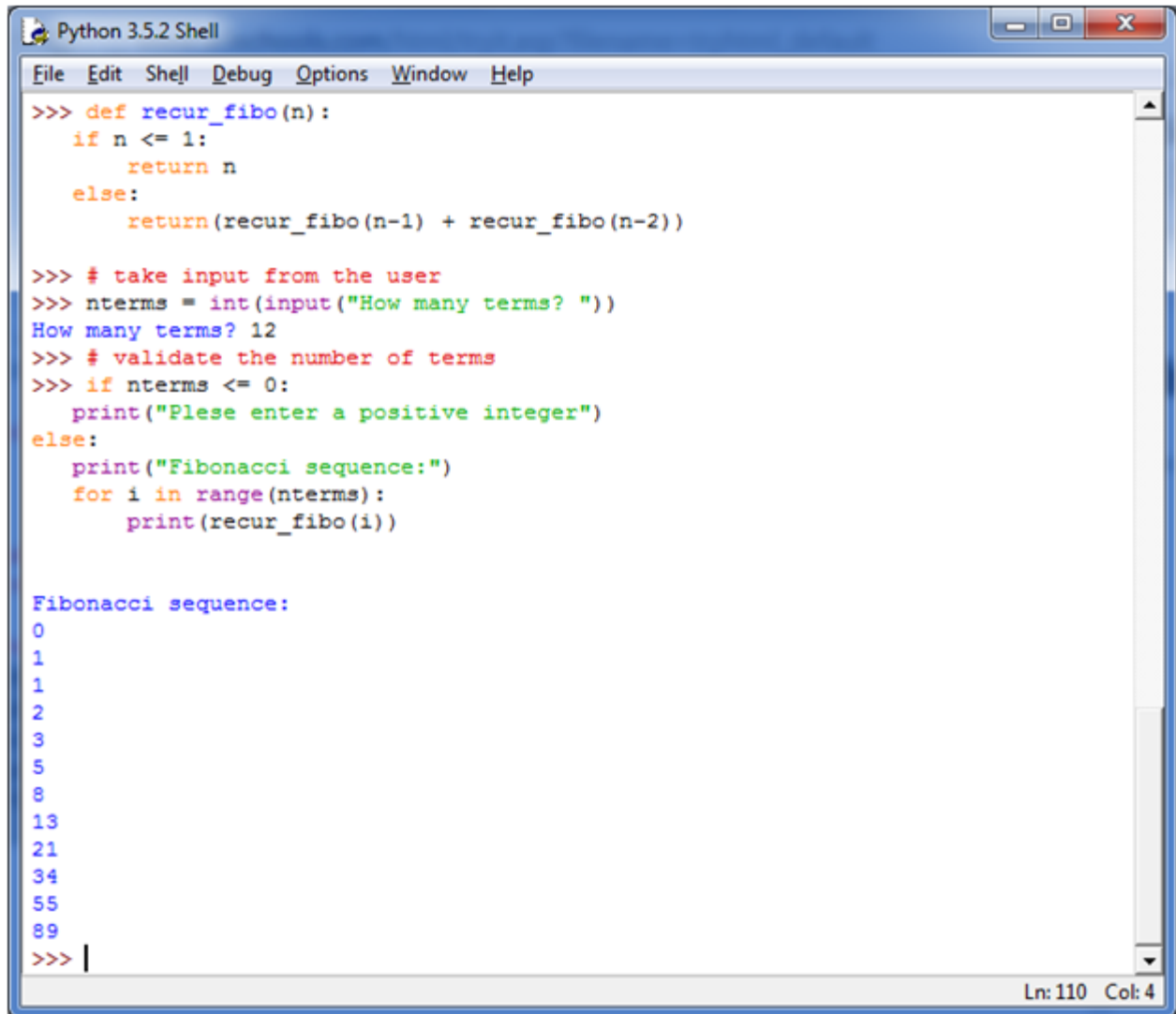
A Fibonacci sequence is a sequence of integers which first two terms are 0 and 1 and all other terms of the sequence are obtained by adding their preceding two numbers.

For example: 0, 1, 1, 2, 3, 5, 8, 13 and so on...

See this example:

```
1.     def recur_fibo(n):
2.         if n <= 1:
3.             return n
4.         else:
5.             return(recur_fibo(n-1) + recur_fibo(n-2))
6.     # take input from the user
7.     nterms = int(input("How many terms? "))
8.     # check if the number of terms is valid
9.     if nterms <= 0:
10.        print("Plese enter a positive integer")
11.    else:
12.        print("Fibonacci sequence:")
13.        for i in range(nterms):
14.            print(recur_fibo(i))
```

Output:

A screenshot of a Python 3.5.2 Shell window. The window has a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The code inside defines a recursive function 'recur_fibo(n)' which returns 'n' if 'n' is less than or equal to 1, and 'recur_fibo(n-1) + recur_fibo(n-2)' otherwise. The program then takes user input for the number of terms, validates it, and prints the Fibonacci sequence. The output shows the sequence from 0 to 89. The status bar at the bottom right indicates 'Ln: 110 Col: 4'.

```
>>> def recur_fibo(n):
    if n <= 1:
        return n
    else:
        return(recur_fibo(n-1) + recur_fibo(n-2))

>>> # take input from the user
>>> nterms = int(input("How many terms? "))
How many terms? 12
>>> # validate the number of terms
>>> if nterms <= 0:
    print("Plese enter a positive integer")
else:
    print("Fibonacci sequence:")
    for i in range(nterms):
        print(recur_fibo(i))

Fibonacci sequence:
0
1
1
2
3
5
8
13
21
34
55
89
>>> |
```

PYTHON PROGRAM TO FIND FACTORIAL OF NUMBER USING RECURSION

Factorial: Factorial of a number specifies a product of all integers from 1 to that number. It is defined by the symbol explanation mark (!).

For example: The factorial of 5 is denoted as $5! = 1*2*3*4*5 = 120$.

See this example:

1. `def recur_factorial(n):`
2. `if n == 1:`
3. `return n`
4. `else:`

```
5.         return n*recur_factorial(n-1)
6.     # take input from the user
7.     num = int(input("Enter a number: "))
8.     # check is the number is negative
9.     if num < 0:
10.         print("Sorry, factorial does not exist for negative numbers")
11.     elif num == 0:
12.         print("The factorial of 0 is 1")
13.     else:
14.         print("The factorial of",num,"is",recur_factorial(num))
```

Output: