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

1) Easy to Use: 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: 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 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.

```
>>> a="Welcome To Python"

>>> print a

Welcome To Python
```

Explanation:Here we are using IDLE to write the Python code. 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.

There are three different ways of working in Python

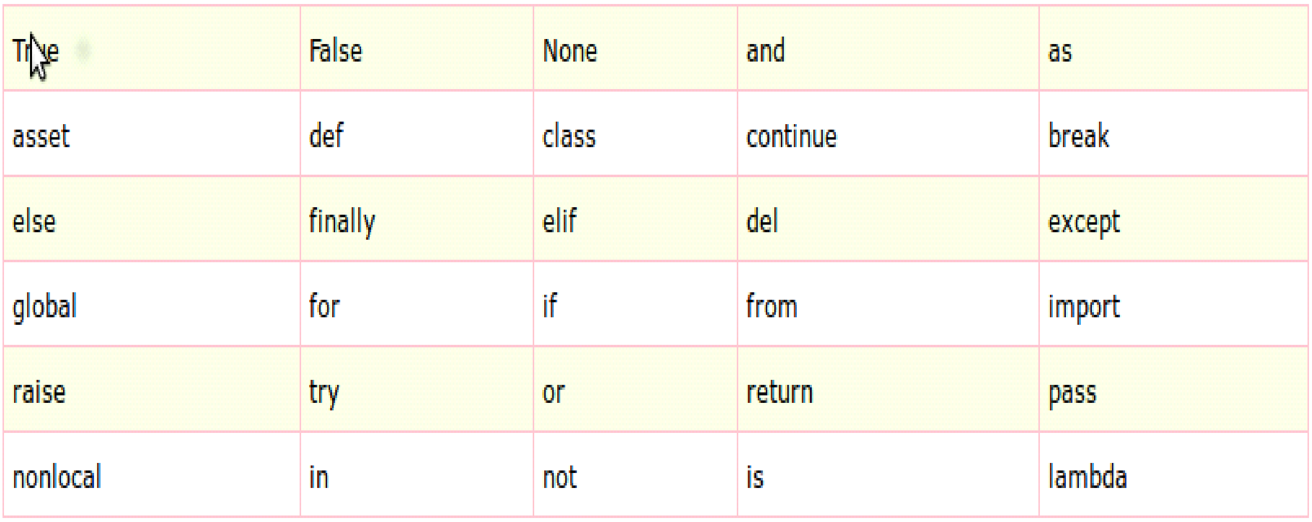
1.Interactive mode

2.Using Script Mode , you can write your Python code in a separate file using any editor of your Operating System and Save it by .py extension..

3.Using IDE: (Integrated Development Environment)

Keywords

Keywords are the reserved words in Python.
We cannot use a keyword as variable name, function name or any other identifier.
They are used to define the syntax and structure of the Python language.
In Python, keywords are case sensitive.



True	False	None	and	as
asset	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 Variables and Data types
Python Variables
A variable is a location in memory used to store some data (value). They are given unique names to differentiate between different memory locations. The rules for writing a variable name is same as the rules for writing identifiers in Python.

We don't need to declare a variable before using it. In Python, we simply assign a value to a variable and it will exist. We don't even have to declare the type of the variable. This is handled internally according to the type of value we assign to the variable.

Variable assignment
We use the assignment operator (=) to assign values to a variable. Any type of value can be assigned to any valid variable.
a = 5
b = 3.2
Multiple assignments

In Python, multiple assignments can be made in a single statement as follows:

```
a, b, c = 5, 3.2, "Hello"

If we want to assign the same value to multiple variables at once,
we can do this as

x = y = z = "same"

This assigns the "same" string to all the three variables.
```

Data types in Python

Every value in Python has a data type.
Since everything is an object in Python programming, data types are actually classes and variables are instance (object) of these classes.
There are various data types in Python. Some of the important types are as follows.

Python Numbers
Integers, floating point numbers and complex numbers falls under Python numbers category.
They are defined as int, float and complex class in Python.
We can use the type() function to know which class a variable or a value belongs to
The isinstance() function to check if an object belongs to a particular class.

```
>>> a = 5
>>> type(a)
```

```
<class 'int'>
>>> type(2.0)
<class 'float'>
>>> isinstance(1+2j,complex)
True
```

Integers can be of any length, it is only limited by the memory available.

A floating point number is accurate up to 15 decimal places.

Integer and floating points are separated by decimal points.

1 is integer, 1.0 is floating point number.
Complex numbers are written in the form, x + yj, where x is the real part and y is the imaginary part.

Here are some examples.

```
>>> a = 1234567890123456789
>>> a
1234567890123456789
>>> b = 0.1234567890123456789
>>> b
0.12345678901234568
>>> c = 1+2j
>>> c
(1+2j)
Notice that the float variable b got truncated.
```

Python Input, Output and Import
Python language provides numerous built-in functions that are readily available to us at the Python prompt.
Some of the functions like input() (raw_input()) and print() are widely used for standard input and output operations respectively

Output formatting
Sometimes we would like to format our output to make it look attractive. This can be done by using the ***str.format()*** method.
This method is visible to any string object.

Example:

```
>>> x = 5; y = 10
>>> print('The value of x is {} and y is {}'.format(x,y))
The value of x is 5 and y is 10

Here the curly braces {} are used as placeholders. We can specify
the order in which it is printed by using numbers (tuple index).


>>> print('I love {0} and {1}'.format('bread','butter'))
I love bread and butter
>>> print('I love {1} and {0}'.format('bread','butter'))
I love butter and bread
```

Input
Up till now, our programs were static. The value of variables were defined or hard coded into the source code.

To allow flexibility we might want to take the input from the user.
In Python, we have the **input() and raw_input()** function to allow this.
The syntax for input() is

```
                   input([prompt])
              raw_input([prompt]) – for strings initially
```

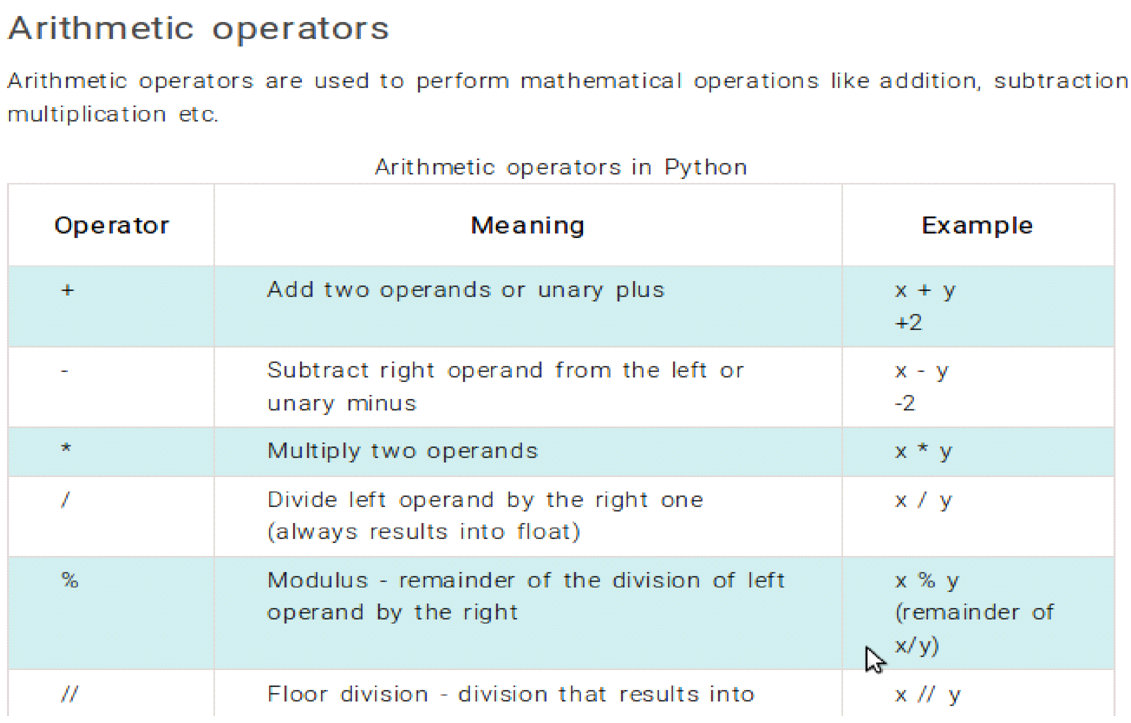
where prompt is the string we wish to display on the screen. It is optional.

```
>>> num = input('Enter a number: ')
Enter a number: 10
>>> num
10

>>> str=raw_input('enter string: ')
enter string: nitte
>>> str
'nitte'
```

Python Operators

Operators are special symbols in Python that carry out arithmetic or logical computation. The value that the operator operates on is called the operand.

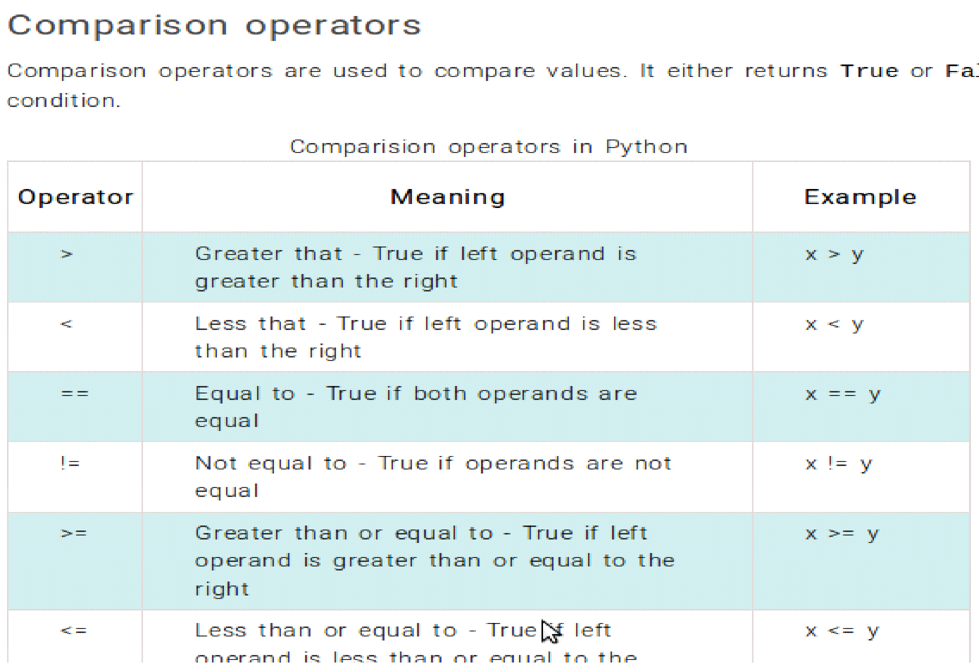


Arithmetic operators

Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication etc.

Arithmetic operators in Python

Operator	Meaning	Example
+	Add two operands or unary plus	x + y +2
-	Subtract right operand from the left or unary minus	x - y -2
*	Multiply two operands	x * y
/	Divide left operand by the right one (always results into float)	x / y
%	Modulus - remainder of the division of left operand by the right	x % y (remainder of x/y)
//	Floor division - division that results into	x // y

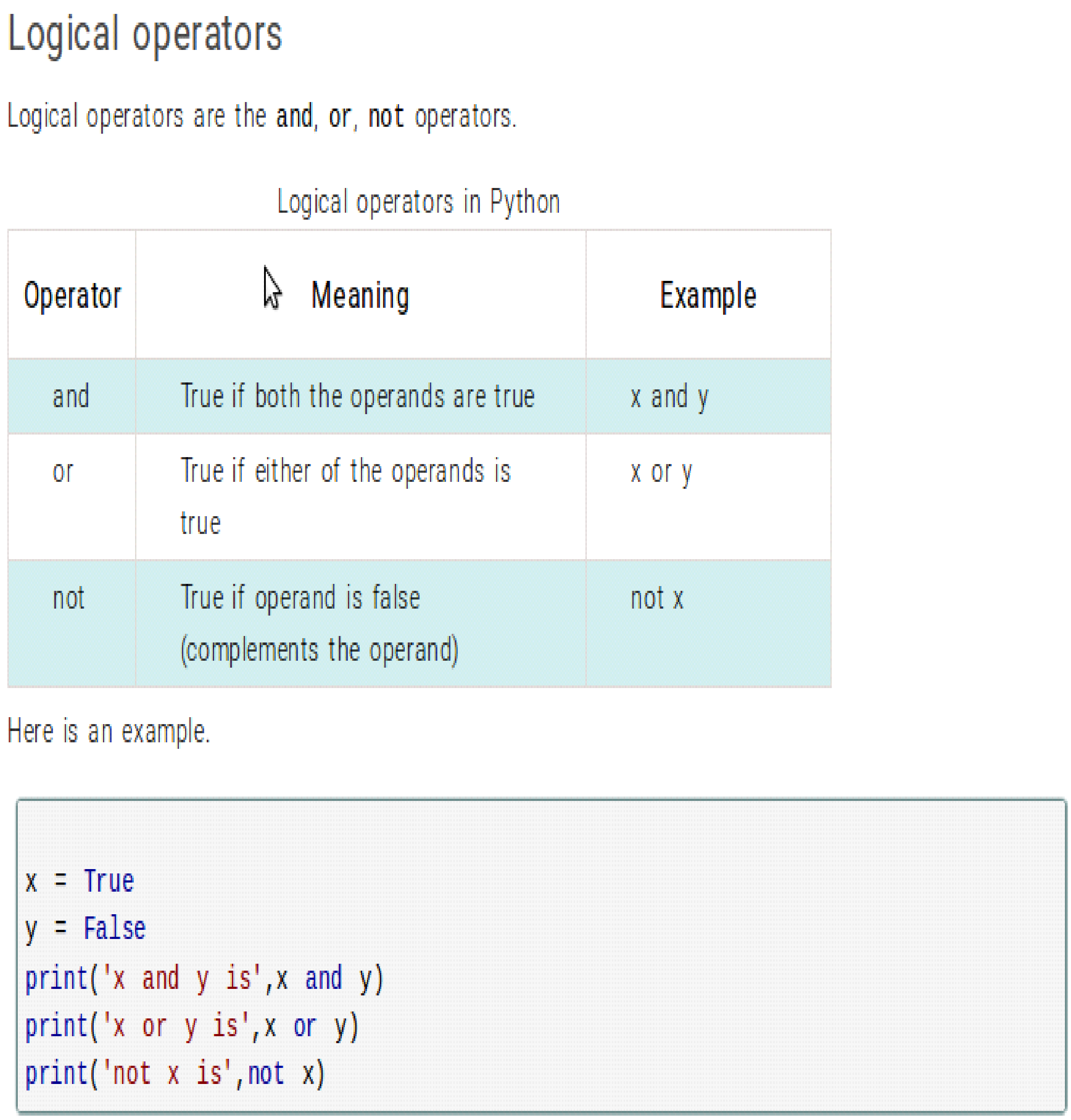


Comparison operators

Comparison operators are used to compare values. It either returns **True** or **False** according to the condition.

Comparision operators in Python

Operator	Meaning	Example
>	Greater that - True if left operand is greater than the right	x > y
<	Less that - True if left operand is less than the right	x < y
==	Equal to - True if both operands are equal	x == y
!=	Not equal to - True if operands are not equal	x != y
>=	Greater than or equal to - True if left operand is greater than or equal to the right	x >= y
<=	Less than or equal to - True if left operand is less than or equal to the	x <= y



Logical operators

Logical operators are the **and**, **or**, **not** operators.

Logical operators in Python

Operator	Meaning	Example
and	True if both the operands are true	x and y
or	True if either of the operands is true	x or y
not	True if operand is false (complements the operand)	not x

Here is an example.

```
x = True
y = False
print('x and y is',x and y)
print('x or y is',x or y)
print('not x is',not x)
```

Bitwise operators

Bitwise operators act on operands as if they were string of binary digits. It operates bit by bit, hence the name. For example, 2 is **10** in binary and 7 is **111**.

Let ***x*** = 10 (**0000 1010** in binary) and ***y*** = 4 (**0000 0100** in binary)

Bitwise operators in Python

Operator	Meaning	Example
&	Bitwise AND	x& y = 0 (**0000 0000**)
|	Bitwise OR	x | y = 14 (**0000 1110**)
~	Bitwise NOT	~x = -11 (**1111 0101**)
^	Bitwise XOR	x ^ y = 14 (**0000 1110**)
>>	Bitwise right shift	x>> 2 = 2 (**0000 0010**)
<<	Bitwise left shift	x<< 2 = 42

Assignment operators

Assignment operators are used in Python to assign values to variables. **a = 5** is a simple assignment operator that assigns the value 5 on the right to the variable ***a*** on the left. There are various compound operators in Python like **a += 5** that adds to the variable and later assigns the same. It is equivalent to **a = a + 5**.

Assignment operators in Python

Operator	Example	Equivatent to
=	x = 5	x = 5
+=	x += 5	x = x + 5
-=	x -= 5	x = x - 5
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5
%=	x %= 5	x = x % 5
//=	x //= 5	x = x // 5
**=	x **= 5	x = x ** 5
&=	x &= 5	x = x & 5

Special operators

Python language offers some special type of operators like the identity operator or the membership operator. They are described below with examples.

Identity operators

is and **is not** are the identity operators in Python. They are used to check if two values (or variables) are located on the same part of the memory. Two variables that are equal does not imply that they are identical.

Identity operators in Python

Operator	Meaning	Example
is	True if the operands are identical (refer to the same object)	x is True
is not	True if the operands are not identical (do not refer to the same object)	x is not True

Here is an example.

```
x1 = 5
y1 = 5
x2 = 'Hello'
y2 = 'Hello'
x3 = [1,2,3]
y3 = [1,2,3]
print(x1 is not y1)
print(x2 is y2)
print(x3 is y3)
```

Output

```
False
True
False
```

Here, we see that ***x1*** and ***y1*** are integers of same values, so they are equal as well as identical. Same is the case with ***x2*** and ***y2*** (strings). But ***x3*** and ***y3*** are list. They are equal but not identical. Since list are mutable (can be changed), interpreter locates them separately in memory although they are equal.

Membership operators

`in` and **`not in`** are the membership operators in Python. They are used to test whether a value or variable is found in a sequence (string, list, tuple, set and dictionary). In a dictionary we can only test for presence of key, not the value.

Operator	Meaning	Example
in	True if value/variable is found in the sequence	5 in x
not in	True if value/variable is not found in the sequence	5 not in x

Here is an example.

```
x = 'Hello world'
y = {1:'a',2:'b'}
print('H' in x)
print('hello' not in x)
print(1 in y)
print('a' in y)
```

Output

```
True
True
True
False
```

Here, **`'H'`** is in ***x*** but **`'hello'`** is not present in ***x*** (remember, Python is case sensitive). Similary, **`1`** is key and **`'a'`** is the value in dictionary ***y***. Hence, **`'a' in y`** returns **`False`**.

Python if...elif...else and Nested if

Decision making is required when we want to execute a code only if a certain condition is satisfied. The if…elif…else statement is used in Python for decision making.

Python if Statement Syntax

```
if test expression:
        statement(s)
```

Here, the program evaluates the test expression and will execute statement(s) only if the text expression is True. If the text expression is False, the statement(s) is not executed.
In Python, the body of the if statement is indicated by the indentation. Body starts with an indentation and the first unindented line marks the end.
Python interprets non-zero values as True. None and 0 are interpreted as False.

Example: Python if Statement

```
num = float(input("Enter a number: "))
if num > 0:
    print("Positive number")
print("This is always printed")
```

Python if...else
Syntax of if...else

```
if test expression:
    Body of if
else:
Body of else
```

```
num = float(input("Enter a number: "))
if num >= 0:
    print("Positive or Zero")
else:
    print("Negative number")
```

The elif is short for else if. It allows us to check for multiple expressions. If the condition for if is False, it checks the condition of the next elif block and so on.
If all the conditions are False, body of else is executed. Only one block among the several if...elif...else blocks is executed according to the condition. A if block can have only one else block. But it can have multiple elif blocks.

```
num = float(input("Enter a number: "))
if num > 0:
    print("Positive number")
elif num == 0:
    print("Zero")
else:
    print("Negative number")
```

Python Nested if statements
We can have a if...elif...else statement inside another if...elif...else statement. In fact, any number of these statements can be nested inside one another.
Indentation is the only way to figure out the level of nesting.

Python Nested if Example
```
num = float(input("Enter a number: "))
if num >= 0:
    if num == 0:
        print("Zero")
    else:
        print("Positive number")
else:
    print("Negative number")
```

Python Loops

Python for Loop

The for loop in Python is used to iterate over a sequence (list, tuple, string) or other iterable objects.

Iterating over a sequence is called traversal.

Syntax of for Loop

```
for val in sequence:
          Body of for
```

Here, val is the variable that takes the value of the item inside the sequence on each iteration. Loop continues until we reach the last item in the sequence. The body of for loop is separated from the rest of the code using indentation.

Example: Python for Loop
```
# Program to find  the sum of all number stored in a list in a
List of numbers

numbers = [6,5,3,8,4,2,5,4,11]
# variable to store the sum
sum = 0

# iterate over the list
for val in numbers:
    sum = sum+val

# print the sum
print "The sum is",sum
```

The range() function

We can generate a sequence of numbers using range() function. range(10) will generate numbers from 0 to 9 (10 numbers). We can also define the start, stop and step size as range(start,stop,step size). step size defaults to 1 if not provided. This function does not store all the values in memory, it would be inefficient. So it remembers the start, stop, step size and generates the next number on the go. To force this function to output all the items, we can use the function **list()**.

The following example will clarify this.

```
>>> range(10)
Range(0, 10)

>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> list(range(2,8))
[2, 3, 4, 5, 6, 7]

>>> list(range(2,20,3))
[2, 5, 8, 11, 14, 17]
```

We can use the range() function in for loops to iterate through a sequence of numbers.

It can be combined with the len() function to iterate though a sequence using indexing.

Here is an example.

```
# Program to iterate through a list using indexing  List of genre

genre = ['pop','rock','jazz']

# iterate over the list using index
for i in range(len(genre)):
    print("I like",genre[i])
```

for loop with else

A for loop can have an optional else block as well.The else part is executed if the items in the sequence used in for loop exhausts. break statement can be used to stop a for loop. In such case, the else part is ignored. Hence, a for loop's else part runs if no break occurs.Here is an example to illustrate this.

```
# Program to show the control flow when using else block in a for
loop

list_of_digits = [0,1,2,3,4,5,6]
input_digit = int(input("Enter a digit: "))
for i in list_of_digits:
    if input_digit == i:
            print("Digit is in the list")
        break
else:
    print("Digit not found in list")
```

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:     for  <expression>:

                       for <expression>:

                           Body

eg: for i in range(1,6):

    for j in range (1,i+1):

        print i,

    print
```

Python while Loop

The while loop in Python is used to iterate over a block of code as long as the test expression (condition) is true. We generally use this loop when we don't know beforehand, the number of times to iterate.

Syntax of while Loop

```
        while test_expression:
                 Body of while
```

In while loop, test expression is checked first.

The body of the loop is entered only if the test_expression evaluates to True. After one iteration, the test expression is checked again. This process continues untill the test_expression evaluates to False.

In Python, the body of the while loop is determined through indentation. Body starts with indentation and the first unindented line marks the end.

Python interprets any non-zero value as True. None and 0 are interpreted as False.

```
# Program to add natural numbers upto n where n is provided by the user
# sum = 1+2+3+...+n


n = int(input("Enter n: "))
sum = 0
i = 1

while i <= n:
    sum = sum + i
    i = i+1

print("The sum is",sum)
```

while loop with else

Same as that of for loop, we can have an optional else block with while loop as well.The else part is executed if the condition in the while loop evaluates to False.while loop can be terminated with a break statement. In such case, the else part is ignored. Hence, a while loop's else part runs if no break occurs and the condition is false.

Here is an example to illustrate this.

```
# Example to illustrate the use of else statement
# with the while loop

counter = 0
while counter < 3:
    print("Inside loop")
    counter = counter + 1
else:
            print("Inside else")
```

break statement

The break statement terminates the loop containing it. Control of the program flows to the statement immediately after the body of the loop. If it is inside a nested loop (loop inside another loop), break will terminate the innermost loop.

Syntax of break

break

continue statement

The continue statement is used to skip the rest of the code inside a loop for the current iteration only. Loop does not terminate but continues on with the next iteration.

Syntax of Continue

continue

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:

pass

eg:

```
for i in [1,2,3,4,5]:

    if i==3:
        pass

        print "Pass when value is",i

    print i,
```

Python File I/O

File is a named location on disk to store related information. It is used to permanently store data in a non-volatile memory (e.g. hard disk). Since, random access memory (RAM) is volatile which loses its data when computer is turned off, we use files for future use of the data.When we want to read from or write to a file we need to open it first. When we are done, it needs to be closed, so that resources that are tied with the file are freed.

1)Open a file

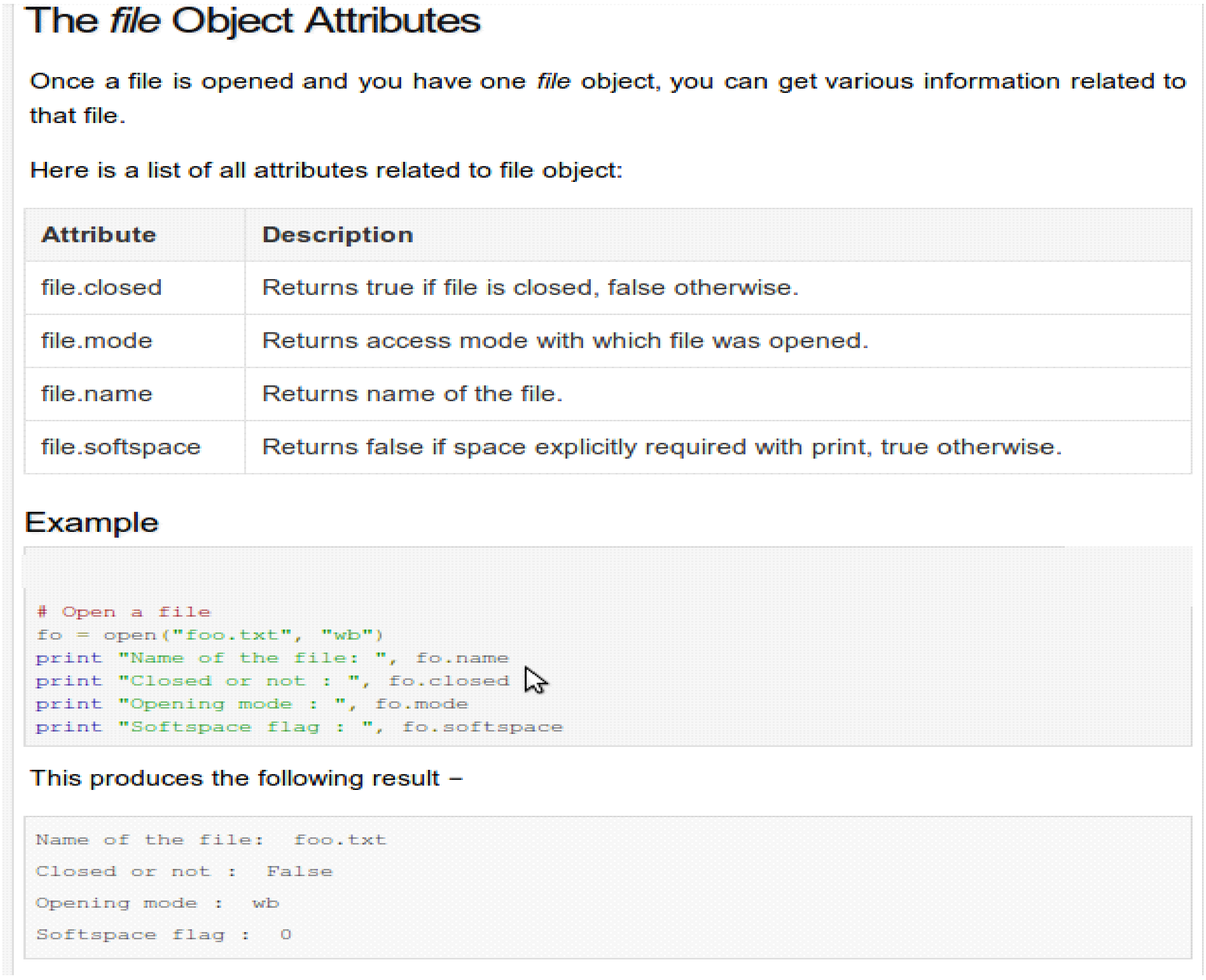
file object = open(file_name [, access_mode][, buffering])

file_name: The file_name argument is a string value that contains the name of the file that you want to access.

access_mode: The access_mode determines the mode in which the file has to be opened, i.e., read, write, append, etc.

Mode	**Description**
"r"	Opens a file for reading.
"w"	Opens a new file for writing. If the file already exists, its old contents are destroyed.
"a"	Opens a file for appending data from the end of the file.
"rb"	Opens a file for reading binary data. "wb" Opens a file for writing binary data.

This is optional parameter and the default file access mode is read (r).



The *file* Object Attributes

Once a file is opened and you have one *file* object, you can get various information related to that file.

Here is a list of all attributes related to file object:

Attribute	Description
file.closed	Returns true if file is closed, false otherwise.
file.mode	Returns access mode with which file was opened.
file.name	Returns name of the file.
file.softspace	Returns false if space explicitly required with print, true otherwise.

Example

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

This produces the following result –

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

2)
The close() Method

The close() method of a file object flushes any unwritten information and closes the file object, after which no more writing can be done.Python automatically closes a file when the reference object of a file is reassigned to another file. It is a good practice to use the close() method to close a file.

Syntax

fileObject.close();

Example

fo = open("foo.txt", "wb")
print "Name of the file: ", fo.name
# Close opened file
fo.close()

3)The write() Method

The write() method writes any string to an open file. It is important to note that Python strings can have binary data and not just text.The write() method does not add a newline character ('\n') to the end of the string .

Syntax

fileObject.write(string);

Here, passed parameter is the content to be written into the opened file.

Example 1.

```
fo = open("foo.txt", "wb")
fo.write( "Python is a great language.\nYeah its great!!\n");
fo.close()
```

The above method would create foo.txt file and would write given content in that file and finally it would close that file.

If you would open this file, it would have following content.

```
Python is a great language.
Yeah its great!!
```

4) The read() Method

The read() method reads a string from an open file. It is important to note that Python strings can have binary data. apart from text data.

Syntax

fileObject.read([count]);

Here, passed parameter is the number of bytes to be read from the opened file.

This method starts reading from the beginning of the file and if count is missing, then it tries to read as much as possible, maybe until the end of file.

Example 1 : Let's take a file foo.txt, which we created above.

```
fo = open("foo.txt", "r+")
str = fo.read(10);
print "Read String is : ", str
fo.close()
```

This produces the following result —

```
Read String is :  Python is
```

Example 2

```
infile = open("Presidents.txt", "r")
 print("(1) Using read(): ")

 print(infile.read() )

 infile.close() # Close the input file
```

5) The readline() Method

After a file is opened for reading data, you can use the read method to read a specified number of characters or all characters from the file and return them as a string, the **readline()**method to read the next line, and the **readlines()** method to read all the lines into a list of strings. When the file pointer is positioned at the end of the file, invoking readline() or read() returns an empty string ''

Example

```
infile = open("C://P.txt", "r")
print("(1) Using read(): ")
line1=infile.readline()
print(line1)
line1=infile.readline()
print(line1)
line1=infile.readline()
print(line1)

infile.close()
```

File Positions

The tell()

This method tells you the current position within the file; in other words, the next read or write will occur at that many bytes from the beginning of the file.

The seek(offset[, from])

This method changes the current file position. The offset argument indicates the number of bytes to be moved. The from argument specifies the reference position from where the bytes are to be moved.If from is set to 0, it means use the beginning of the file as the reference position and 1 means use the current position as the reference position and if it is set to 2 then the end of the file would be taken as the reference position.

Example Let us take a file foo.txt, which we created above.

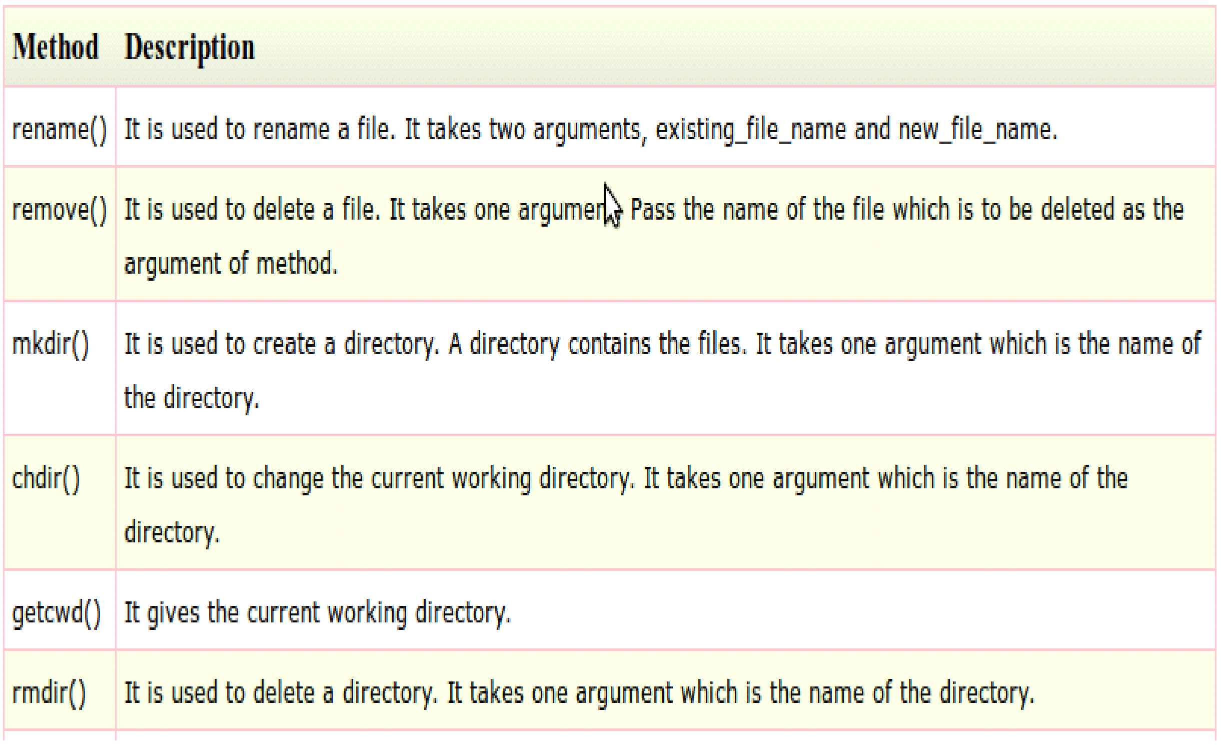
```
fo = open("foo.txt", "r+")
str = fo.read(10);
print "Read String is : ", str
# Check current position
position = fo.tell();
print "Current file position : ", position
# Reposition pointer at the beginning once again
position = fo.seek(0, 0);
str = fo.read(10);
print "Again read String is : ", str
# Close opend file
fo.close()
```

This produces the following result −

```
Read String is :  Python is
Current file position :  10
Again read String is :  Python is
```

Python Directory and Files Management

If there are large number of files in Python, we can place related files in different directories to make things more manageable. **A directory** or folder is a collection of files and sub directories. Python has the **os** module, which provides us with many useful methods to work with directories (and files as well).



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.

1) rename():

Syntax: **os.rename(existing_file_name, new_file_name)**

eg:import os

os.rename('mno.txt','pqr.txt')

2) remove():

Syntax:**os.remove(file_name)**

eg: import os

os.remove('mno.txt')

3) mkdir()

Syntax: **os.mkdir("file_name")**

eg:

```
import os

os.mkdir("new")
```

4) chdir()

Syntax: **os.chdir("file_name")**

eg:

```
import os

os.chdir("new")
```

5) getcwd()

Syntax: **os.getcwd()**

eg:

```
import os

print os.getcwd()
```

NOTE: In order to delete a directory, it should be empty. In case directory is not empty first delete the files.In order to remove a non-empty directory we can use the **rmtree()** method inside the **shutil** module.

Eg

```
import shutil

shutil.rmtree('test')
```

Testing a File’s Existence

To prevent the data in an existing file from being erased by accident, you should test to see if the file exists before opening it for writing. The **isfile** function in the **os.path** module can be used to determine whether a file exists. For example:

```
import os.path

 if os.path.isfile("Presidents.txt")

        print("Presidents.txt exists")
```

Here isfile("Presidents.txt") returns True if the file Presidents.txt exists in the current directory.

Following program illustrates a program that copies data from a source file to a target file and counts the number of lines and characters in the file.

```
 import os.path
  import sys
  def main():
        # Prompt the user to enter filenames
         f1 = input("Enter a source file: ").strip()
        f2 = input("Enter a target file: ").strip()
       # Check if target file exists
        if   os.path.isfile(f2)  :
               print(f2 + " already exists")
             sys.exit()
        # Open files for input and output
        infile = open(f1, "r")
       outfile = open(f2, "w")
       # Copy from input file to output file
        countLines = countChars = 0 20
            for line in infile:
                   countLines += 1
                     countChars += len(line)
                    outfile.write(line)
         print(countLines, "lines and", countChars, "chars copied")
         infile.close()  # Close the input file
         outfile.close() # Close the output file
  main() # Call the main function
```

Reading/Writing Numbers From/To a file

1**)Writing Numbers to a File**

The file method write expects a string as an argument. Therefore, other types of data, such as integers or floating-point numbers, must first be converted to strings before being written to an output file. In Python, the values of most data types can be converted to strings by using the str function. The resulting strings are then written to a file with a space or a newline as a separator character.The next code segment illustrates the output of integers to a text file. Five hundred random integers between 1 and 500 are generated and written to a text file named integers.txt. The newline character is the separator**.**

```
import random

ƒ=open("integers.txt",'w')

for count in x range(500):

      number=random.randint(1,500)

        f.write(str(number) + "\n")

  f.close()
```

2) Reading numbers from a file

The next code segment illustrates this technique. It opens the file of random integers written earlier, reads them, and prints their sum.Obtaining numbers from a text file in which they are separated by spaces is a bit trickier. One method proceeds by reading lines in a for loop, as before. But each line now can contain several integers separated by spaces. You can use the string method split to obtain a list of the strings representing these integers, and then process each string in this list with another for loop.

```
f=open(“integers.txt”,'r')

sum  =0

 for line in f:

   line=line.strip()

    number=int(line)

       sum+=number
print(“The sum is”,sum)
```

Creating and Reading a formatted file (csv or tab-separated).

1) Reading a formatted file (csv or tab-separated).
// Code to read data from a Comma Separated Value (CSV) file

```
import csv

csvfile= open('ex.csv')
readCSV = csv.reader(csvfile, delimiter=',')
for row in readCSV:
   print(row)
   print(row[0])
   print(row[0],row[1],row[2],)
```

2) Write data to a Comma Separated Value (CSV) file

```
import csv

data=["First_Name Last_Name City".split(","), "Shyam Rao Mangaluru".split(","),"Ram Bhat Udupi".split(",")]
csv_file = open('ex1.csv', 'w', newline='')
writer = csv.writer(csv_file, delimiter=',')
for line in data:
   writer.writerow(line)
```