**Python**

* Python is a general purpose, interpreted, object-oriented and high level programming language
* Developed by Guido Van Rossum, in late 1980’s
* Derived from many other languages like C, C++, ALGOL, Small Talk, Unix shell scripting etc

**Python features**

* Easy to learn: few keywords, simple structure, clearly defined structure
* Easy to read: readable
* Broad standard library
* Portable
* Databases
* Object oriented support
* Dynamic data types and dynamic type checking
* Automatic garbage collection
* Free and open source
* High level language

**Running Python**

* Python programs can be written in 2 modes:

1. Interactive mode i.e. at the command line
2. Scripting mode, using an editor

**Interactive mode**

* Enter the command “python” at the command prompt and we will get >>> prompt, where we can give python commands.
* To get the current version of python, we can use –v option as:

python -v

**Scripting mode**

* Create the python file using any editor and save it with the extension “.py”.
* We can now execute them by saying,

python <filename>

at the command prompt

**Python identifiers**

* Identifier is a name used to identify a variable, function, class, module or object
* It starts with a letter A to Z, a to z or underscore, followed by zero or more letters, underscores and digits.
* Symbols line @,$,% etc are not allowed.
* They are case sensitive.

**Multi line statements**

* Allows line continuation character (\) to denote that the line should continue.

print()

* Displays some data on to the screen.
* Generates a new line by default.

Ex:

a=10;

print “welcome to python”

print “value of a is:”,a

**Waiting for the user**

* To get input from the user, we have:

raw\_input()

* Data entered through raw\_input is in string format. To convert it into numbers, we can use int() and float().

**Reading input via sys.stdin.readline()**

* Read a line of input from user.

import sys

x=sys.stdin.readline()

print “x value :”,x

**Python variable types**

* Numbers: integers, floating point
* Strings
* Lists
* Tuples
* Dictionaries

**Python operators**

* Arithmetic operators: +, - \* / % \*\*
* Relational operators: >,<,>=,<=,==, !=, <>
* Assignment operators: =, +=,-=, \*=,/=,%=,\*\*=
* Logical operators: and, or, not
* Membership operators: in, not in
* Identity operators: is, is not
* String operators: +, \*

**Multiple assignment to variables**

* a=b=c=1
* a,b,c=1,2,”John”

**Strings**

* Can be embedded in single quotes, double quotes and triple quotes.

Ex:

str=‘hello world’;

print str;

print str[0];

print str[2:5];

print str[2:];

print str \*2;

print str+”hello”;

* Str[2:10] 🡪 all characters between the indices 2 and 9
* Str[2:10:] 🡪 same as above, increment value is 1
* Str[2:10:2] 🡪 all characters between 2 and 9, increment value is 2

**Creating null variables**

var=None

Where ‘None’ is the keyword indicating nothing.

To check whether a variable has got a value or not,

if var==None:

do something

(Or)

if var is None:

do this

**Control structures**

* Branching statements
* Looping statements

**Branching statements**

* if
* if..else
* if..elif..else
* Nested if

**‘if’ statement**

* if expression:

statements

* + Checks the result of ‘expression’. If true, the statements are executed. If ‘false’, nothing is there here to execute.

**if..else**

if expression:

true block

else:

false block

**if..elif..else**

if expression1:

statement block-1

elif expression2:

statement block-2

:

:

else :

false block

**Nested if**

**if cond1:**

if cond2:

st-1

else:

st-2

a=int(raw\_input("Enter first num"))

b=int(raw\_input("Enter second number"))

c=int(raw\_input("Enter third number"))

if a>b:

if a>c:

print "a is big"

else:

print "c is big"

else:

if b>c:

print "b is big"

else:

print "c is big"

**Single statement suites**

* If a suite contains a single statement, we can write it in the same line, followed by colon.

var=100

if (var==100): print “proper value”

**Python Loops**

* while loop
* for loop

**while loop**

while expression:

statements

* Statements are repeated until the expression is true.

num=int(raw\_input("Enter any number:"))

temp=num

rev=0

while(num>0):

rem=num%10

rev=(rev\*10)+rem

num=num/10

if temp==rev:

print temp,"is a Palindrome number"

else:

print temp," is not Palindrome"

**The ‘else’ statement used with ‘while’**

while condition:

statements

else:

statements

* After the ‘while’ loop is exhausted(when no 'break' statement is used), statements in ‘else’ will be executed.

for i in range(1,10):

if i==5:

break

print i

else:

print "thanks"

print "good day"

**for loop**

for counter in sequence:

statements

* ‘counter’ is assigned to every element of the list and the statements are repeated till the list is exhausted.

Ex:

for x in 1,2,3,4:

print x,”\t”

**Using range() with for**

* for x in range(1,10):

print x,”\t”

* Generates a sequence of numbers starting from 1, increments by one till it reaches 9.
* 10 is not included in the list.

**Using ‘for’ with lists**

fruits=[“banana”, “apples”, ”oranges”]

for x in range(len(fruits)):

print fruits[x]

**len(tuple\_name)**: no. of elements in the tuple

* With range(), we can even specify the value to be incremented as:

range(1,10,2): generates a list starting from 1, till 10, increments the numbers by 2.

So, the list will be:1,3,5,7,9

Ex:

for i in range(11,2,-1):

print i

🡪values are decremented here by -1

**‘break’ statement**

* Abruptly terminates executing a looping statement.

**‘continue’ statement**

* Skips the rest of the statements in the current iteration and continues the next iteration of the loop.

Ex:

i=0

while(i<=10):

i=i+1

if i==5:

continue

print i

print "end of the loop"

**‘pass’ statement**

* We use this when we don’t want to execute any command.
* A null operation; nothing happens when it executes.

**Python functions**

* A function is a block of organized, reusable code, which provides modularity and code reusability.

Syntax:

def function-name([parameters]):

[document string]

statements

[return expression]

Ex:

def printMe(str):

print str

return

Calling a function

* function\_name([arguments])

Ex:

printMe(“welcome to Python”)

printMe(10)

**Functions with no arguments, no return value.**

def add():

a=int(raw\_input("enter first num"));

b=int(raw\_input("enter second num"));

print "sum=",a+b

Whereever we want to call,

add() # function call

**Passing parameters to functions**

def add(x,y):

print "sum=",x+y

add(10,20)

**Returning values from a function**

def add(x,y):

return x+y

res=add(10,20)

print "sum=",res

**No args..Returning values**

def add():

a=int((raw\_input(“enter first no:”))

b=int((raw\_input(“enter second no:”))

return a+b

res=add()

print “Sum of the nos=“,res

**Local variables**

* Variables defined in a function are local to it. That is their scope is local to the function.

def func(x):

print “x is”, x

x=2

print “x is changed to”, x

x=50

func(x)

print “x is still”, x

**The ‘global’ statement**

* This statement makes the changes done on a variable to continue, even if the function is terminated also.

def change():

global i # i is the same one defined in main program

i=10

print "inside the fn,i=",i

i=1 #global one

print "before fn call,i=",i

change()

print "after fn call,i=",i

def withdraw():

global bal

amnt=2000

bal=bal-amnt

def deposit():

global bal

amnt=5000

bal=bal+amnt

#main

bal=10000

print "balanace=",bal

withdraw()

print"current bal after withdrawl=",bal

deposit()

print "bal after depositing=",bal

Note:

* We can’t use the same name for a variable and a function.

Ex:

def fn(x):

return x\*2

fn=fn(5)

print fn

result=fn(4) #Error, because fn is no longer a function, it #is treated as a interger now.

print result

**Types of function arguments**

* Required arguments
* Keyword arguments
* Default arguments
* variable-length arguments

**Required arguments**

* They are passed to the function in the same order as they are defined.
* All the arguments are compulsory.
* The no. of arguments should match with the no. of parameters defined in the function.

Ex: print\_str(str):

print str

print\_str(“welcome to python”)

def printme(num,name):

print "Number:",num

print "Name:",name

printme("Rama",10)

🡪 count of arguments and order of arguments should also match.

**Keyword arguments**

* They are related to function calls.
* While calling the function, if we are using keyword arguments, the caller identifies the arguments with the parameter name.
* No need to follow the order while passing the arguments, because we are using the name of the parameter explicitly during the function call.

Ex: def my\_func(str,num):

----

---

my\_func(num=10,str=“python”)

Ex:

def printme(num,name):

print "Number:",num

print "Name:",name

printme(name="Rama",num=10)

**Default argument values**

* Some times, values for the parameters are optional and use default values, if the user don’t provide any values for them.
* This can be done with default argument parameters.
* They are specified in the function definition with parameter name, followed by = and the default value.
* The default value should always be a constant.

Ex:

def say(message, times = 1):

print message \* times

say('Hello')

say('World', 5)

def add(a=10,b=20):

print a+b

add()

add(100)

add(100,200)

**Variable-length arguments**

* Accepts variable length arguments while defining the function as:

def func(var1, \*list):

print var1,”\n”

for x in list:

print x,”\n”

func(10)

func(10,20,30,40)

func(10,20)

* The extra args are stored in \*list, and used when necessary.

Ex:

def printme(\*list):

for i in list:

print i

printme()

printme(10,20,30,40)

**lambda functions: Anonymous functions**

* Functions with out any name can be created using the keyword “lambda”.
* They are defined with out the need of ‘def’ keyword.
* They can return only one value, but can take any no. of arguments.
* They can’t contain multiple expressions or commands.
* Can’t access any other variables, except the ones in the parameter list and the global variables.
* Can’t be used directly with print() as it returns a value.

syntax

* lambda [arg1 ,arg2,.....argn]:expression

x=lambda num:num\*2

sum=lambda a,b:a+b

print "sum=",sum(10,20)

**Return statement**

* The statement return [expression] exits a function, optionally passing back an expression to the caller.
* A return statement with no arguments is the same as return None .

def sum( arg1, arg2 ):

total = arg1 + arg2

print "Inside the function : ", total

return total;

total = sum( 10, 20 );

print "Outside the function : ", total

**Local and global variables**

* Variables that are defined inside a function body have a local scope, and those defined outside have a global scope.
* This means that local variables can be accessed only inside the function in which they are declared, whereas global variables can be accessed throughout the program body by all functions.
* When you call a function, the variables declared inside it are brought into scope.

total = 0;

def sum( arg1, arg2 ):

total = arg1 + arg2;

print "Inside the function local total : ", total

return total;

sum( 10, 20 );

print "Outside the function global total : ", total

**Document strings**

* DocStrings are an important tool that you should make use of, since it helps to document the program better and makes it more easy to understand.
* A string on the first logical line of a function is the *docstring for that function.*
* The convention followed for a docstring is a multi-line string where the first line starts with a capital letter and ends with a dot. Then the second line is blank followed by any detailed explanation starting from the third line.

def printMax(x, y):

'''Prints the maximum of two numbers.

The two values must be integers.''‘

x = int(x)

y = int(y)

if x > y:

print x, 'is maximum'

else:

print y, 'is maximum‘

printMax(3, 5)

print printMax.\_\_doc\_\_

We can access the docstring of the printMax function using the \_\_doc\_\_ (notice the double underscores)

attribute (name belonging to) of the function.

**sequences**

* The most basic data structure in Python is the sequence. Each element of a sequence is assigned a number - its position or index. The first index is zero, the second index is one, and so forth.
* The commonly used sequences are lists and tuples, dictionaries.

**Python Lists**

* The list is a most versatile data type available in Python which can be written as a list of comma-separated values (items) between square brackets.
* The elements of a list need not be of the same data type.

examples

* list1 = ['physics', 'chemistry', 1997, 2000];
* list2 = [1, 2, 3, 4, 5 ];
* list3 = ["a", "b", "c", "d"];

🡪Like string indices, list indices start at 0

**Accessing the complete list**

Ex:

list1=[10,20,"siri","murali"]

for i in list1:

print i

(or)

print list1

Example:

list1=[10,20,30,40]

print "Accessing elements directly"

for i in list1:

print i

print "Accessing elements via indices"

for i in range(len(list1)):

print list1[i]

**Accessing list values**

* To access list values, we can use [ ] and index number.

Ex:

list1 = ['physics', 'chemistry', 1997, 2000];

list2 = [1, 2, 3, 4, 5, 6, 7 ];

print "list1[0]: ", list1[0]

print "list2[1:5]: ", list2[1:5]

example

list1=[10,20,“tom”, ”jerry”]

for i in range(len(list1)):

print list1[i]

**Lists with negative indices**

List[-n]: gives the nth element from the last.

Ex:

list1=[10,20,"siri","murali"]

list2=["bangalore","hyderabad"]

print list1[-2]🡺 gives 2nd element from the last i.e. “siri”

**Reversing a list**

list1=[10,20,30,"rama","python"]

print list1[::-1]

* Reverses the list
* Wont effect the original array

**List ranges (:)**

list1=[10,20,"siri","murali"]

list2=["bangalore","hyderabad"]

print list1[1:3]

🡪 won’t include index 3 in the list

**Updating lists**

list = ['physics', 'chemistry', 1997, 2000];

print "Value available at index 2 : "

print list[2];

list[2] = 2001;

print "New value available at index 2 : "

print list[2];

**Deleting list elements**

We can use the ‘del()’ to delete elements of a list.

Ex:

list1 = ['physics', 'chemistry', 1997, 2000];

print list1;

del list1[2];

print "After deleting value at index 2 : "

print list1

Ex:

del list1[-1]

**Deleting range of items**

Ex:

list1=[10,20,30,"rama","python"]

del list1[2:4]

print list1

**Deleting a complete list**

list1=[10,20,30,"rama","python"]

del list1

#print list1

**+ on lists**

* Used to concatenate two lists

Ex:

list1=[10,20,"siri","murali"]

print list1+["hello"]

print list1

* Original list won’t change.

**To change the list permanently**

list1=[10,20,30,"rama","python"]

list1=list1+["welcome"]

print list1

Ex:

list1=[10,20,30,40]

list2=["hello"]

list3= list1+list2

print list3

**\* On lists**

* Repeats the list specified no. of times

Ex:

list1=[10,20,"siri","murali"]

print list1\*2

print list1

Ex:

list1=[10,20,30,40]

list2=list1\*3

print list2

**Membership operators: in, not in**

* ‘in’ and ‘not in’ checks whether an element is the member of the list or not.
* Returns true/false.

Ex:

list1=[10,20,"siri","murali"]

list2=["bangalore","hyderabad"]

print 20 in list1

**List built-in functions**

Python includes the following list functions:

* len(list): Gives the total length of the list.
* max(list): Returns item from the list with max value.
* min(list): Returns item from the list with min value.
* list(tuple\_name): Converts a tuple into list.

Ex:

list1=[10,20,30,40,"perl","python"]

print max(list1): gives python

print min(list1): gives 10

* When we have multiple data types in a list, max() and min() work based on ASCII values.

**Nested lists**

list1=[1,2,3]

list2=[10,20,30]

list3=['a',list1,'b',list2,'c']

print "complete list:",list3

print list3[1][2] # gives 3 i.e 3rd element of list1

**Check the type of list elements**

list1=[10,20,13,"welcome",12.34,"rama123",-12]

i\_c=0

f\_c=0

s\_c=0

for i in list1:

if type(i)is int:

i\_c+=1

elif type(i) is float:

f\_c+=1

else:

s\_c+=1

print "int count=",i\_c

print "Float count=",f\_c

print "strings count=",s\_c

**Clearing the screen in Python**

import os

os.system(‘clear’): in unix

os.system(‘cls’): in windows

Executing external commands

import os

os.system(‘date’)

os.system(‘ls -l’)

-------------------

(Or)

from subprocess import call

call(["date"])

call(["ls","-l"])

call(["cal"])

call(["ps","-f"])

(Or)

import subprocess

subprocess.call([‘ls’, ‘-l’])

# ‘system’ is the old fashion of invoking external commands. It is deprecated

**Execute unix commands and store the result in python variables**

* popen() is a function in ‘os’ module, which opens a pipe to or from a command.
* The return value will be a open file object connected to the pipe, which can be read or write, depending on the mode is r or w.

Syntax:

Os.popen(command, [mode])

The default mode is ‘r’.

Ex:

import os

n=os.popen(‘ls|wc –l’)

cou=n.read()

print “No.of files=“, cou

List methods

* List.append(obj): appends object to the list
* List.count(ele): counts how many times ‘ele’ occurs in the list.
* List.extend(seq): extends the contents of ‘seq’ to the list.
* List.index(obj): returns the lowest index where ‘obj’ appears in the list.
* List.insert(index,obj): inserts ‘obj’ at the specified index in the list.
* List.remove(obj): removes ‘obj’ from the list.
* List.reverse(): reverses elements of the list
* List.sort(): sorts the elements of the list.
* List.pop([index]): removes and returns the last object from the list.

Remove all occurrences of a particular element from the list

list1=[10,20,30,10,10]

ele=int(raw\_input("Enter element to be deleted"))

if ele not in list1:

print ele," is not present in the list"

else:

for i in list1:

list1.remove(ele)

print "The list is", list1

(or)

list1=[10,20,30,40,20,50,20]

cou=list1.count(20)

for i in range(cou):

list1.remove(20)

print list1

Aliases for a list

list1=[10,20,30,40]

list2=list1

.. Now, list2 acts as an alias for list1

.. Changes made on one, are reflected on the other.

.. But if one list is deleted completely, the second list remains intact.

Python Tuples

* A tuple is a sequence of immutable elements.
* Tuples can’t be changed and use parentheses to store elements.
* Tuples can store elements in optional parentheses separated by commas.

Ex:

tup1=(10,20,30,40)

Print tup1

Ex:

tup1=10,20,30,40

print tup1

* Empty tuple

t1=();

* Tuple with one element

t1=(10,)

* Tuples have an index starting with 0.

Updating tuples

* We can’t update tuples, as they are immutable. So, the assignment is invalid on tuples.

Ex:

List1[0]=1000; 🡪 invalid

Concatenating tuples

t1=(10,20,30,40,50)

t2=("siri","murali")

print t1+t2

* won’t effect the original tuples

Repetition on tuples (\*)

t1=(10,20,30,40,50)

print t1\*2

print t1

🡪 Won’t effect original tuples

Deleting tuples

* Its not possible to delete individual tuple elements.

Ex:

t1=(10,20,30,40,50)

del t1

in and not in operators on tuples

Built-in tuple functions

* len(tuple): gives the no. of elements of the tuple.
* max(tuple): gives maximum element of the tuple
* min(tuple): gives the minimum element of the tuple.
* tuple(list): converts the list into a tuple.

Python Dictionaries

* Python dictionary is mutable.
* Contains pairs of keys and their values.
* Also known as “associative arrays” (or) “hashes.

Ex:

fruits={'oranges':10, ‘ cherries':90, cherries’:90,’lemon’:90}

Accessing dictionary values

dict={'oranges':10,'cherries':20,'apples‘:90}

print dict[‘apples’]

* Use [ ] and the ‘key’ to access the value.

updating and deleting dictionary elements

Clear all dictionary contents

dict={'oranges':10,'cherries':20,'apples':90}

dict.clear()

print dict

* Clears all dictionary contents and now, dict will be empty i.e. { }

Deleting the dictionary itself

dict={'oranges':10,'cherries':20,'apples':90}

del dict

print dict

* error, as ‘dict’ is deleted.

Built-in functions on dictionaries

* len(dict): gives the length of the dictionary.
* type(var): specifies the type of the passed variable.

Dictionary methods

* Dict.clear(): clears all the contents of the dictionary.
* Dict.copy(): creates a copy of the dictionary.
* Dict.fromkeys(): creates dictionary from a sequence.

Syntax:

dict=dict.fromkeys(sequence,[value])

list1=["jones","james","clark"]

dict=dict.fromkeys(list1)

print dict

* Creates dictionary with specified indices, with value ‘none’.

* Dict.get(index,[default]): gets the value for an index.  
  ‘default’ is the value to be thrown if an index don’t exist.
* Dict.has\_key(key): returns ‘true’ if a key exists in the dictionary.
* Dict.items(): returns (key,value) pairs present in the dictionary.

Ex:

dict={'oranges':10,'apples':20,'cherries':30}

print dict.items()

(or)

Ex:

dict={'oranges':10,'apples':20,'cherries':30}

for i in dict.items():

print i

* Dict.keys():

Prints all available keys in the dictionary.

* Dict.setdefault(key,value): sets a value to the ‘key’ if it doesn’t exist.
* dict.update(dict2): adds to ‘dict’ the (key,value) pairs of ‘dict2’.
* dict.values(): returns all values of the dictionary.

Prog. to access dictionary elements

dict={'oranges':10,'apples':20,'cherries':30}

for i in dict: #only keys

print i

print dict[i]

Program to validate login credentials

import sys

log={'alice':'a123','james':'j123','clark':'c123'}

u=raw\_input("logname:")

p=raw\_input("password:")

for i in log:

if(u==i and p==log[i]):

print "valid"

sys.exit()

print "invalid"

Getting help in Python

Ex: get info. About a type

list1=[10,20,30,40,50]

dir(list1)

help(list1)

Ex: get info. About a method

import sys

help(sys)

help(sys.exit)

Ex: get info. About a function

help(len)

**Python File I/O**

Opening files

* open() is the function used to open files.
* Before reading or writing to a file, we need to open it.

Syntax:

File\_object=open(filename, [access mode], [buffering])

* ‘filename’ is a string containing the name of the file.
* ‘access mode’ determines the purpose of opening the file like reading, writing, appending etc. Default access mode is ‘read’.
* If ‘buffering’ is set to 0, no buffering happens. If buffering is 1, contents are buffered while accessing the file.

File accessing modes

* r : opens the file for reading only.
* r+ : opens the file for both reading and writing.
* w: opens the file for writing. Creates new file if it doesn’t exists and overwrites if already exists.
* w+: opens the file for both writing and reading.
* a: opens the file for appending.
* a+: opens the file for both appending and reading.

‘file’ object attributes

* file.closed: returns ‘true’ if the file is closed.
* file.mode: returns the access mode.
* file.name: returns the file name.
* file.read([size]): reads the file contents
* file.readline(): reads a complete line
* file.readlines(): reads until EOF
* file.write(): writes a string to a file.
* file.writelines(): writes a sequence of strings to the file.

close()

* Closes the opened file, flushes any unwritten information into the disk.

Syntax:

fileobject.close();

Writing to files

* write() is the method used to write any string to an open file.
* Won’t add a \n at the end of the string.

Syntax:

fileobject.write(string);

Ex:

f=open("myfile","w")

f.write("this is my file\n");

f.write("this is the second line");

f.close();

read()

* Reads a string from an open file.

Syntax:

fileobject.read([count])

* ‘count’ is the no. of bytes to read from a file.
* If ‘count’ is omitted, it reads till the end of the file.

Ex:

f=open("myfile","r")

str=f.read(10)

print "first 10 bytes are:",str

f.close();

f=open("myfile","r")

str=f.read()

print "complete file is:",str

f.close();

Reading a file till EOF

f1=open("emp","r")

for i in f1:

print i

#f1.readlines()

f1.close()

Another version

f1=open("emp","r")

while 1:

line=f1.readline()

if not line:

break

print line

f1.close()

Searching for a string in a file (grep in unix)

f1=open("dept")

for line in f1:

if "sales" in line:

print line,

f1.close()

Extract specified lines from a line

* enumerate() returns the count(starting from 0) and the value.

Ex:

for n,name in enumerate("this is python"):

... print n,"-->",name

...

0 --> t

1 --> h

2 --> i

3 --> s

4 -->

5 --> i

6 --> s

7 -->

8 --> p

9 --> y

10 --> t

11 --> h

12 --> o

13 --> n

To get the 3rd line of a file

f1=open(“dept.txt”)

for n,line in enumerate(f1):

if n==2:

print line

Another version:with out “enumerate()”

f1=open("dept")

cou=1

while True:

line=f1.readline()

if cou==3:

print line

break

cou=cou+1

f1.close()

File positions

* tell() specifies at which location, the current file pointer is.

Syntax:

Pos=Fileobject.tell()

Ex:

f=open("myfile","r")

str=f.read(10)

print "first 10 bytes are:",str

pos=f.tell();

print "file position is:",pos

f.close();

seek()

* Changes the file pointer position.

Syntax:

seek(offset,[ from])

* ‘offset’ indicates the no. of bytes to move.
* ‘from’ indicates the location from where the pointer has to be moved.

0🡪 from beginning of the file

1 🡪 current position

2🡪 end of the file.

Ex: Adding a new line at the beginning

f1=open("emp.doc","r+")

print f1.read()

f1.seek(0)

f1.write("new line written\n")

f1.seek(0)

print f1.read()

f1.close()

Ex: Adding a new record at the beginning

f1=open("emp.txt","r+")

f1.write("106\tjames\n")

f1.seek(0)

print f1.read()

f1.close()

Renaming files

* Python ‘os’ module helps us to perform file processing operations like renaming and deleting files.
* We need to import this module first and can use any related functions.
* rename() is the method used to rename files.

Syntax:

os.rename(old name, new name)

Ex:

import os

os.rename("myfile","myfile\_new")

remove()

* Deletes the specified file.

Syntax:

os.remove(filename)

os.unlink(filename)

Directories in Python

* The ‘os’ module has several methods to handle directories.

mkdir():

* Creates new directories in the current directory.

Ex:

import os

os.mkdir("siri")

chdir()

* Used to change the current directory.

os.chdir(“dir-name”);

getcwd()

* Displays the current working directory.

Ex:

import os

print os.getcwd();

rmdir()

* Deletes the specified directory.

Ex:

import os

os.rmdir("siri")

File object methods

* File.close(): closes the opened file.
* File.next(): returns the next line from the file.
* File.read(): reads the complete file.
* File.read(size): reads the ‘size’ no. of bytes
* File.readline(): reads the complete line of a file.
* File.seek(offset,[from]): sets the file pointer position.
* File.tell(): returns the current position of the file pointer.
* File.write(str): writes the string ‘str’ to the file.
* File.writelines(strings): writes a sequence of strings to the file.
* File.fileno(): returns the file descriptor.

Ex: reading a single line of a file

f=open("myfile","r")

print f.readline()

f.close()

Reading complete file

f=open("myfile","r")

print f.read()

f.close()

Ex: to get file ids

f=open("myfile","r")

s=open("file2","w")

print f.fileno()

print s.fileno()

f.close()

**os object methods**

Methods of os module

* os.chdir(path): changes the current working directory to the specified path.
* os.getcwd(): returns a string representing the current working directory.
* os.link(source,dest): creates a hard link to the file ‘src’
* os.symlink(src,dest): creates a symbolic link for ‘src’.
* os.listdir(path): lists all the contents of the directory specified in the path.
* os.makedirs(path,[mode]): recursively creates the directories.
* os.mkdir(path,[mode]): creates a directory.
* os.remove(path): removes the file
* os.removedirs(path): recursively removes the directories.
* os.rename(src,dest): renames file/dir ‘src’ to ‘dest’.
* os.rmdir(path): removes a empty directory.
* os.unlink(path): removes the file in the specified path.
* os.walk(path):gives path,directories and files in the given path.

examples

* To print current working directory:

import os

print os.getcwd()

* To get contents of a directory:

import os

print os.listdir("c:/python27")

* Create directory

import os

os.mkdir("c:/babloo")

* Create direcories recursively

import os

os.makedirs("c:/nani/babloo/bobby")

* Removing files:

import os

os.unlink("prog1.py")

os.remove("myfile")

* Remove dirs. Recursively

import os

os.removedirs("c:/nani/babloo/bobby")

* Remove a dir.

import os

os.rmdir("c:/siri/murali")

* Renaming a file/dir:

import os

os.rename("test.py","testing.py")

Assignments:

1. Display the properties of a file "sample"

under windows,

import os

n=os.popen('dir /N')

for i in n:

if "sample" in i:

print i

output will be in the following columns:

1. date

2.time

3. am/pm

4. type of the file

5. size

6. filename

To get the file attributes individually,

import os,string

n=os.popen('dir /N')

for i in n:

if "emp.txt" in i:

list1=string.split(i)

print "date created=",list1[0]

print "time=",list1[1]+list1[2]

under linux:

import os,string

n=os.popen('ls -l e1.txt')

for i in n:

list1=string.split(i)

print list1[5]+list1[6]+list1[7]

Ex: display the files created on a particular date

import os,string

n=os.popen('dir /N')

for i in n:

if "07-08-2015" in i:

print i

Ex: searching for a file

import os,re

for (path,dir,files) in os.walk("c:\\tests"):

for i in files:

if "file1" in i:

print "found at", path ,"\\", dir

To find address of a variable,

id(var)

To get address in hexa decimal notation,

hex(id(var))

Implementing stacks in Python