Knowledge flow

PYTHON PROGRAMMING

Knowledge flow

"LEARNING STARTS WITH VIEWING THE WORLD DIFFERENTLY."

Knowledge flow- A mobile learning platform provides apps, eBooks and video tutorials.

Knowledge flow brings you a learning eBook of ***Python Programming***. This eBook is for all information technology, computer science and engineering students and professionals across the world.

Follow us on

Facebook

Google plus

Twitter

For more information visit us at

Knowledgeflow.in

knowledgeflowapps.blogspot.in

Thank you for using Knowledge flow eBooks

PYTHON PROGRAMMING

1. Introduction

2. Pythons Basic Syntax

3. Variable types

4. Basic Operators

5. Decision making of Python

6. Pythons dictionary

7. Python Date and Time

8. Python functions

9. Python Files I/O

10. Operating System methods

11. More eBooks and Apps

12. Our eBooks on IT and Computer Science

Disclaimer

This eBook contents is for informational and study purposes only. The ***Knowledge flow*** makes no claims, promises, or guarantees about the accuracy, completeness, or adequacy of the contents of this eBook and no legal liability or other responsibility is accepted by ***Knowledge flow*** for any errors, omissions, or statements on this eBook.

Introduction

- Python is a commonly used high-level programming language.
- Its design is characterized as code readability, and its syntax allows programmers to express concepts in very less lines of code than would be possible in languages such as C++ or Java.
- Python entertains multiple programming paradigms also being an object-oriented as well as functional programming.
- It features a automatic memory management that is also a dynamic type of system.
- It also has a wide ranged standard library.
- Python has its own interpreters which are used for installation on an operating systems.
- Third-party tool such as Py2exe/Pyinstaller can be used as an executable program for some of the operating systems.

Logo of Python

History

- The idea of Python was given in the late 1980s.
- Its implementation started by Guido van Rossum in December 1989 at CWI in the Netherlands.
- Van Rossum is known as Python's principal author, and plays a central role in deciding the direction of Python

- Python 2.0 got released on date 16 of October 2000.

Features

- multi-paradigm
- object-oriented programming
- structured programming
- support functional programming
- logic programming
- Uses dynamic typing
- cycle-detecting garbage collector for memory management
- Dynamic name resolution

Official repository of software for Python contains more than 54,000 packages with a wide range of functionality like.

- graphical user interfaces
- web frameworks
- multimedia
- databases
- networking
- communications
- test frameworks
- automation
- web scraping
- documentation tools
- system administration
- scientific computing
- text processing
- image processing

Development environment

- Python implements function like a command line interpreter the user enters the statement in a sequenced manner and receives the results instantly.
- Python shell implement features like auto-completion, retention of session state and syntax highlighting.

Implementation

- Google started a project in 2009 termed ***Unladen Swallow*** whose aim was increasing the speed of Python interpreter as well as improving its multithreading ability to scale to thousands of cores. Later the project lost Google's back up as well as its main developers.
- Integrated *CPython* transforms byte code to machine code at runtime and also produces codes specialized for certain data types that is faster than the standard Python code.

Pythons Basic Syntax

Python sample program-

$ Python

Python 2.4.3 (#1, Nov 11 2010, 13:34:43)

[GCC 4.1.2 20080704 (Red Hat 4.1.2-48)] on linux2

Type"help","copyright","credits" or"license" for more information

≫≫

Following should be written on the command prompt and the enter key should be pressed-

>>> print "Hello, Python!";

Output

Hello, Python!

Identifiers

Here are following identifiers conventions declared for python.

- All the identifiers are in lower case except the class name which would start with an upper case.
- Identifiers are used for privacy, starting with a single leading underscore.
- If two leading underscore are contained then it contains a strong privacy.
- If it ends with two different underscores then the language is defined special name.

Lines and Indentation

Various spaces in indentation are variables, and almost whole statement must be inside the block must be indented the similar amount.

Example

```
if True:

print "True"

else:

print "False"
```

Second block that would produce an error is as follows.

```
if True:

print "Answer"

print "True"

else:

print "Answer"

print "False"
```

Following is an example for blocks with various statements.

```
#!/usr/bin/python

import sys

try:

# open file stream
```

```
file = open(file_name, "w")
except IOError:
print "There was an error writing to", file_name
sys.exit()
print "Enter '", file_finish,
print "' When finished"
while file_text != file_finish:
file_text = raw_input("Enter text: ")
if file_text == file_finish:
# close the file
file.close
break
file.write(file_text)
file.write("\n")
file.close
file_name = raw_input("Enter filename: ")
if len(file_name) == 0:
print "Next time please enter something"
sys.exit()
try:
```

```
file = open(file_name, "r")

except IOError:

print "There was an error reading file"

sys.exit()

file_text = file.read()

file.close()

print file_text
```

Command line argument

Basic information’s are provided through this so that one can know how to run these programs.

Example

```
$ python -h

usage: python [option] ... [-c cmd | -m mod | file | -] [arg] ...

Options and arguments (and corresponding environment variables):

-c cmd : program passed in as string (terminates option list)

-d : debug output from parser (also PYTHONDEBUG=x)

-E : ignore environment variables (such as PYTHONPATH)

-h : print this help message and exit.
```

Accessing command line argument

It provides a getopt module for accessing a command line.

Syntax

$ python test.py arg1 arg2 arg3

There are two purpose of it which is as follow.

- List of command line arguments are provided through sys.argv.
- And the total number of command line is Len (sys.argv).

Example

```
#!/usr/bin/python

import sys

print 'Number of arguments:', len(sys.argv), 'arguments.'

print 'Argument List:', str(sys.argv)
```

Next step is to run above script as follows.

$ python test.py arg1 arg2 arg3

Output

Number of arguments: 4 arguments.

Argument List: ['test.py', 'arg1', 'arg2', 'arg3']

Variable type

There are five main Data types which are as follows.

- Numbers
- String
- List
- Tuple
- Dictionary

Numbers

It contains numeric values and is created when value is assigned to them.

Example

var1 = 1

var2 = 10

Syntax of delete statement is as follows.

del var1[,var2[,var3[....,varN]]]]

Syntax for deleting multiple or single objects are as follows.

del var

del var_a, var_b

It supports four different numerical styles which are as follows.

- Int
- Long
- Float

- Complex

Strings

- It is identified as the set of characters.
- The plus sign is represented as the string concatenation operator.
- Similarly the asterisk is represented as the repetition operator.

Example

```
#!/usr/bin/python

str = 'Hello World!'

print str # Prints complete string

print str[0] # Prints first character of the string

print str[2:5] # Prints characters starting from 3rd to 5th

print str[2:] # Prints string starting from 3rd character

print str * 2 # Prints string two times

print str + "TEST" # Prints concatenated string.
```

Output

```
Hello World!

H

llo

llo World!

Hello World!Hello World!
```

Hello World!TEST.

List

These are written in between the brackets and with commas.

Example

```
#!/usr/bin/python

list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]

tinylist = [123, 'john']

print list # Prints complete list

print list[0] # Prints first element of the list

print list[1:3] # Prints elements starting from 2nd till 3rd

print list[2:] # Prints elements starting from 3rd element

print tinylist * 2 # Prints list two times

print list + tinylist # Prints concatenated lists
```

Output

```
['abcd', 786, 2.23, 'john', 70.200000000000003]

abcd

[786, 2.23]

[2.23, 'john', 70.200000000000003]

[123, 'john', 123, 'john']

['abcd', 786, 2.23, 'john', 70.200000000000003, 123, 'john']
```

Tuples

It is also a similar sequenced data type like list.

Example

```
#!/usr/bin/python

tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )

tinytuple = (123, 'john')

print tuple # Prints complete list

print tuple[0] # Prints first element of the list

print tuple[1:3] # Prints elements starting from 2nd till 3rd

print tuple[2:] # Prints elements starting from 3rd element

print tinytuple * 2 # Prints list two times

print tuple + tinytuple # Prints concatenated lists
```

Output

```
('abcd', 786, 2.23, 'john', 70.200000000000003)

Abcd

(786, 2.23)

(2.23, 'john', 70.200000000000003)

(123, 'john', 123, 'john')

('abcd', 786, 2.23, 'john', 70.200000000000003, 123, 'john'
```

Dictionary

It is a kind of hash table and the values are enclosed in the curly bracket.

Example

```
#!/usr/bin/python

dict = {}

dict['one'] = "This is one"

dict[2] = "This is two"

tinydict = {'name': 'john','code':6734, 'dept': 'sales'}

print dict['one'] # Prints value for 'one' key

print dict[2] # Prints value for 2 key

print tinydict # Prints complete dictionary

print tinydict.keys() # Prints all the keys

print tinydict.values() # Prints all the values
```

Output

```
This is one

This is two

{'dept': 'sales', 'code': 6734, 'name': 'john'}

['dept', 'code', 'name']

['sales', 6734, 'john']
```

Basic Operators

The basic operators for python language are as follows.

- Arithmetic Operators
- Comparison Operators
- Logical Operators
- Membership Operators
- Identity Operators

Arithmetic operators

Addition- Adds values on either side of the operator. Represented by ‘+’.

Example

a + b will give 30.

Subtraction- This subtracts right hand operand from left hand operand. Represented by ‘-‘.

Example

a - b will give -10.

Multiplication- Multiplies values on either side of the operator. Represented by ‘*‘.

Example

a * b will give 200.

Division- Divides left hand operand by right hand operand. Represented by ‘/’.

Example

b / a will give 2.

Modulus- Divides left hand operand by right hand operand and returns remainder. It is represented by ‘%’.

Example

b % a will give 0.

Exponent- It performs power calculation on operators. It is represented by ‘**’.

Example

a**b will give 10 to the power 20.

Floor Division- This is a kind of division of operands where the result will be the quotient where the digits after the decimal point are deducted.

Example

9//2 is equal to 4 and

9.0//2.0 is equal to

4.0.

Program

#!/usr/bin/python

a = 21

b = 10

c = 0

```
c = a + b
print "Line 1 - Value of c is ", c
c = a - b
print "Line 2 - Value of c is ", c
c = a * b
print "Line 3 - Value of c is ", c
c = a / b
print "Line 4 - Value of c is ", c
c = a % b
print "Line 5 - Value of c is ", c
a = 2
b = 3
c = a**b
print "Line 6 - Value of c is ", c
a = 10
b = 5
c = a//b
print "Line 7 - Value of c is ", c
```

Output

```
Line 1 - Value of c is 31
```

Line 2 - Value of c is 11

Line 3 - Value of c is 210

Line 4 - Value of c is 2

Line 5 - Value of c is 1

Line 6 - Value of c is 8

Line 7 - Value of c is 2

Comparison operator

==- Checks if the value of two operands is equal or not, if yes then only the condition becomes true.

Example

(a == b) is not true.

!=- Checks if the value of two operands is equal or not, if values are not equal. Then only the condition becomes true.

Example

(a != b) is true.

<>- Checks whether the value of two operands is same or not, if values are not same then only condition becomes true.

Example

(a <> b) is true.

>=- Checks if the value of left operand is larger than or equal to the value of

right operand, if yes then condition becomes true.

Example

(a >= b) is not true.

<=- Checks if the value of left operand is smaller than or equal to the value of right operand, if yes then condition becomes true.

Example

(a <= b) is true.

Program

```
#!/usr/bin/python

a = 21

b = 10

c = 0

if ( a == b ):

print "Line 1 - a is equal to b"

else:

print "Line 1 - a is not equal to b"

if ( a != b ):

print "Line 2 - a is not equal to b"

else:

print "Line 2 - a is equal to b"
```

```
if ( a <> b ):
print "Line 3 - a is not equal to b"
else:
print "Line 3 - a is equal to b"
if ( a < b ):
print "Line 4 - a is less than b"
else:
print "Line 4 - a is not less than b"
if ( a > b ):
print "Line 5 - a is greater than b"
else:
print "Line 5 - a is not greater than b"
a = 5;
b = 20;
if ( a <= b ):
print "Line 6 - a is either less than or equal to b"
else:
print "Line 6 - a is neither less than nor equal to b"
if ( b >= a ):
print "Line 7 - b is either greater than or equal to b"
```

else:

print "Line 7 - b is neither greater than nor equal to b"

Output

Line 1 - a is not equal to b

Line 2 - a is not equal to b

Line 3 - a is not equal to b

Line 4 - a is not less than b

Line 5 - a is greater than b

Line 6 - a is either less than or equal to b

Line 7 - b is either greater than or equal to b.

Logical operator

And- Termed Logical AND operator, both the values are true,therefore the condition becomes true.

Example

(a and b) is true.

Or- Termed Logical OR Operator any of the two values are non zero and then the condition becomes true.

Example

(a or b) is true.

Not- Termed as Logical NOT Operator that function is to reverse the logical situation of its operand.

Example

not(a and b) is false.

Membership operator

In- Its evaluation is true only if it finds a particular variable in the specified sequence and falseif not.

Example

x in y, here in results in a 1 if x is a member of sequence y.

Not in- If it's evaluates is true and it does not finds a variable in the specified sequence and false if not.

Example

x not in y, here not in results in a 1 if x is not a member of sequence y.

Identify operator

Is- Its evaluation is true if the variables on either side of the operator do point to the same object and falseif not.

Example

x is y, here is results in 1 if id(x) equals id(y).

Is not- its evaluation is false if the variables on either side of the given operator points to the same object and true if not.

Example

x is not y, here is not results in 1 if id(x) is not equal to id(y).

Program

```
#!/usr/bin/python

a = 20

b = 20

if ( a is b ):

print "Line 1 - a and b have same identity"

else:

print "Line 1 - a and b do not have same identity"

if ( id(a) == id(b) ):

print "Line 2 - a and b have same identity"

else:

print "Line 2 - a and b do not have same identity"

b = 30

if ( a is b ):

print "Line 3 - a and b have same identity"

else:

print "Line 3 - a and b do not have same identity"

if ( a is not b ):

print "Line 4 - a and b do not have same identity"

else:

print "Line 4 - a and b have same identity"
```

Output

Line 1 - a and b have same identity

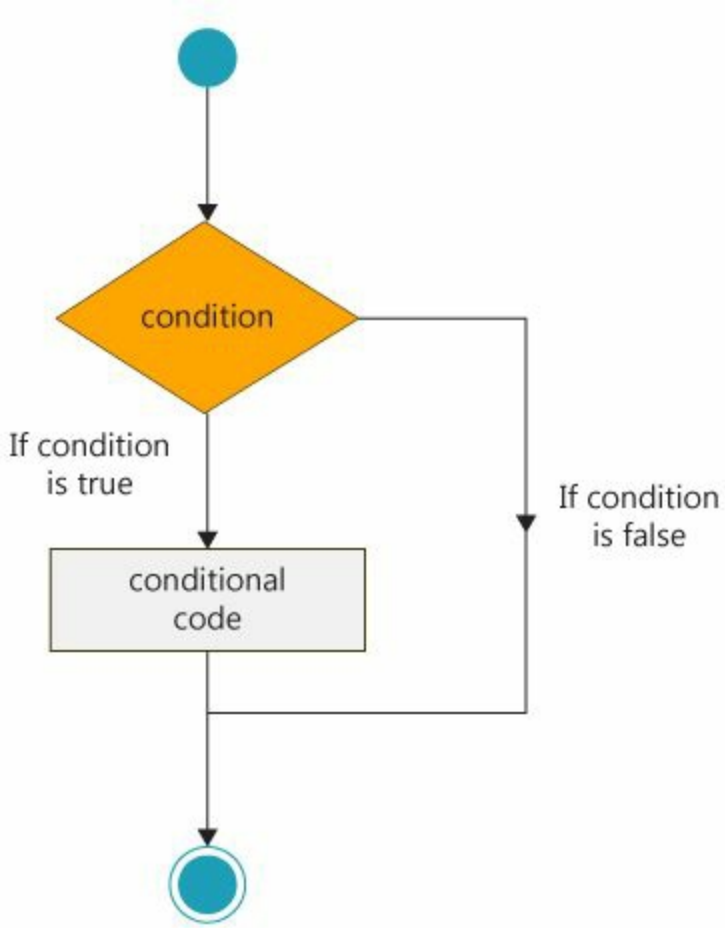
Line 2 - a and b have same identity

Line 3 - a and b do not have same identity

Line 4 - a and b do not have same identity.

Decision making of Python

- Decision making is one of the important aspects of any programming languages.
- Similarly it goes with the python the various structures are as follows.



Flow diagram

- In this programming language it assumes non-zero and non-null value as true.
- But if it is either zero or null it is assumed as false.

Following are the three decision making statement.

- If statement.
- If else statement.
- Nested if statement.

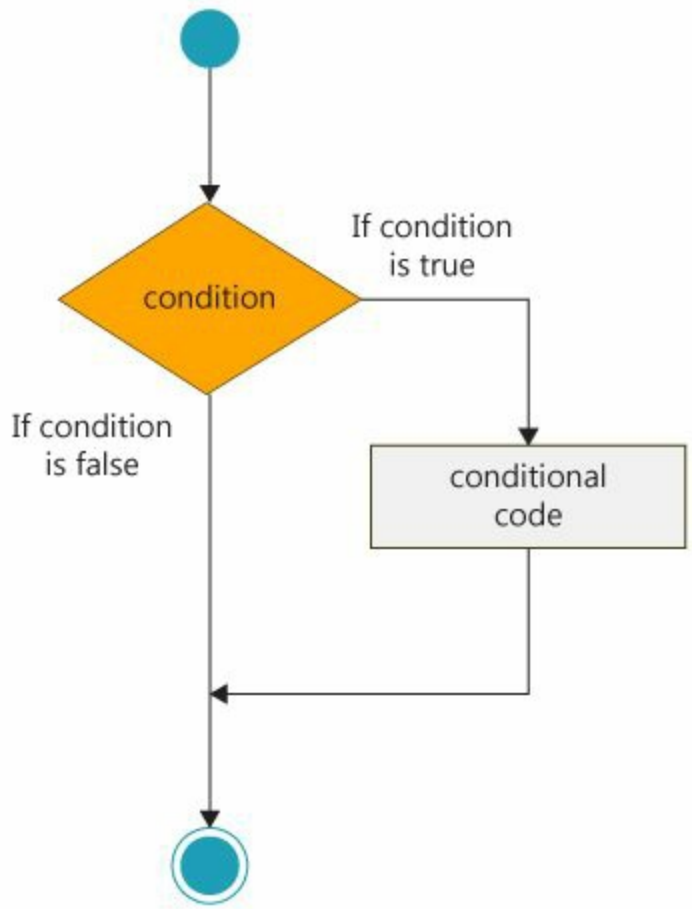
If statement

- Contains logical expression.
- Utilizes these expressions and converts them into a result after comparison.

Syntax

If expression:

Statement



If flow table

Program

```
#!/usr/bin/python

var1 = 100

if var1:

print "1 - Got a true expression value"

print var1

var2 = 0

if var2:

print "2 - Got a true expression value"

print var2

print "Good bye!"
```

Output

```
1 - Got a true expression value

100

Good bye!
```

If else statement

- If and else are combined.
- Else provide an option to the if statement.

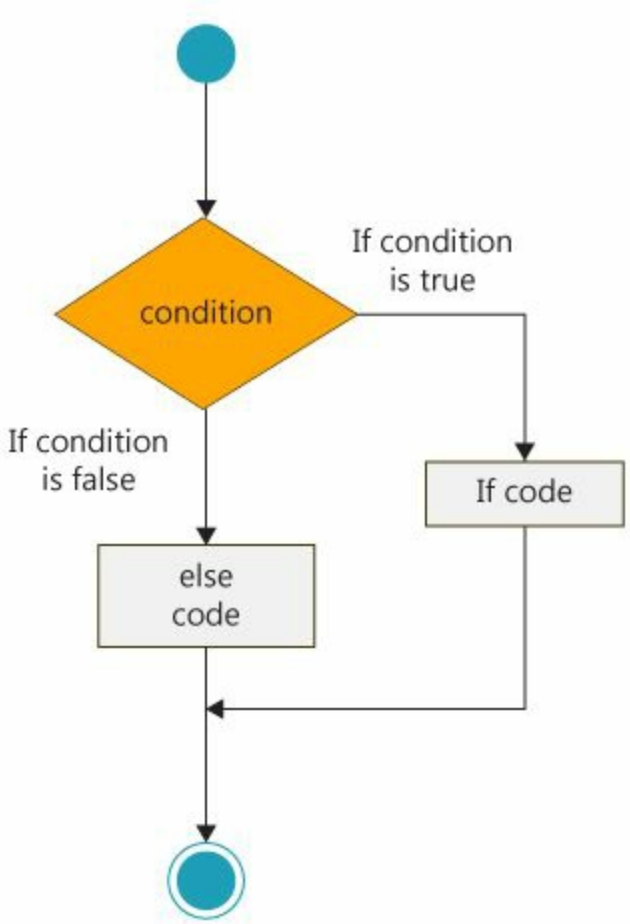
Syntax

if expression:

statement(s)

else:

statement(s)



If else flow table

Program

#!/usr/bin/python

var1 = 100

if var1:

```
print "1 - Got a true expression value"

print var1

else:

print "1 - Got a false expression value"

print var1

var2 = 0

if var2:

print "2 - Got a true expression value"

print var2

else:

print "2 - Got a false expression value"

print var2

print "Good bye!".
```

Output

```
1 - Got a true expression value

100

2 - Got a false expression value

0

Good bye!
```

Nested if statement

- It is having various conditions one by one in sequence.
- It’s used to resolve one condition after one.

Syntax

if expression1:

statement(s)

if expression2:

statement(s)

elif expression3:

statement(s)

else

statement(s)

elif expression4:

statement(s)

else:

statement(s).

Program

#!/usr/bin/python

var = 100

if var < 200:

print "Expression value is less than 200"

```
if var == 150:
print "Which is 150"
elif var == 100:
print "Which is 100"
elif var == 50:
print "Which is 50"
elif var < 50:
print "Expression value is less than 50"
print "Could not find true expression"
print "Good by!".
```

Output

```
Expression value is less than 200
Which is 100
Good bye!
```

Pythons dictionary

Dictionary is a container type facility that is used to save or contain various objects or data. General syntax for dictionary is as follows.

dict = {'Alice': '2341', 'Beth': '9102', 'Cecil': '3258'}

A dictionary can be created as follows.

dict1 = { 'abc': 456 };

dict2 = { 'abc': 123, 98.6: 37 };

Accessing value in dictionary

To access a data or element one has to use the square brackets and the key value.

Example

#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'};

print "dict['Name']: ", dict['Name'];

print "dict['Age']: ", dict['Age'];

Output

dict['Name']: Zara

dict['Age']: 7

Updating dictionary

Used to modify or add a new existing entry.

Example

```
#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'};

dict['Age'] = 8; # update existing entry

dict['School'] = "DPS School"; # Add new entry

print "dict['Age']: ", dict['Age'];

print "dict['School']: ", dict['School'].
```

Output

```
dict['Age']: 8

dict['School']: DPS School
```

Delete dictionary

It is used to remove the selected dictionary element or the entire data in dictionary elements.

Example

```
#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'};

del dict['Name']; # remove entry with key 'Name'

dict.clear(); # remove all entries in dict

del dict ; # delete entire dictionary

print "dict['Age']: ", dict['Age'];
```

print "dict['School']: ", dict['School'];

dict['Age']:

Traceback (most recent call last):

File "test.py", line 8, in <module>

print "dict['Age']: ", dict['Age'];

TypeError: 'type' object is unsubscriptable.

Built in dictionary functions and methods

- ***cmp(di1, di2)-*** This is to compares elements of both di.
- ***len(di)-*** This provide the total length of the di which would be equal to the number of items in the di.
- ***str(di)-*** This produces a printable string representation of a di (dictionary).
- ***type(variable)-*** This returns the type of a pass variable and also If pass variable is di, then it would return a di type.

Example

#!/usr/bin/python

dict1 = {'Name': 'Zara', 'Age': 7};

dict2 = {'Name': 'Mahnaz', 'Age': 27};

dict3 = {'Name': 'Abid', 'Age': 27};

dict4 = {'Name': 'Zara', 'Age': 7};

print "Return Value : %d" % cmp (dict1, dict2)

print "Return Value : %d" % cmp (dict2, dict3)

print "Return Value : %d" % cmp (dict1, dict)

Output

Return Value : -1

Return Value : 1

Return Value : 0

Python Date and Time

There are various methods for handling date and time in python. Therefore has various calendar modules by which the actual time and date are tracked.

Tick

The interval of time in python is in the time instance of seconds. These seconds are in floating point format.

Example

#!/usr/bin/python

import time; # This is required to include time module.

ticks = time. time ()

print "Number of ticks since 12:00am, January 1, 1970:", ticks

Output

Number of ticks since 12:00am, January 1, 1970: 7186862.73399.

Python handles time with Tuple of 9 numbers.

S. No.	***Field***	***Values***
0	4 DIGITED YEAR	2008
1	MONTH	1 TO 12
2	DAYS	1 TO 31
3	HOURS	0 TO 23

4	MINUTES	0 TO 59
5	SECONDS	0 TO 61
6	DAY OF WEEK	0 TO 6
7	DAY OF YEAR	1 TO 366
8	DAYLIGHT SAVING	-1, 0, 1,-1

Table

The above tuple are considered equal to the STRUC_TIME structure. Therefore these also contain some attributes which are as follows.

S. No.	***Attributes***	***Value***
0	tm_year	2008
1	tm_mon	1 to 12
2	tm_mday	1 to 31
3	tm_hour	0 to 23
4	tm_min	0 to 59
5	tm_sec	0 to 61
6	tm_wday	0 to 6
7	tm_yday	1 to 366
8	tm_isdst	-1,0,1,-1

Table

Getting current time

This passes the particular required floating point to the function. It returns all the valid nine element to the time Tuple.

Example

```
#!/usr/bin/python

import time;

localtime = time.localtime(time.time())

print "Local current time :", localtime
```

Output

```
Local current time : time.struct_time(tm_year=2008, tm_mon=5,
tm_mday=15,

tm_hour=12, tm_min=55, tm_sec=32, tm_wday=0, tm_yday=136,
tm_isdst=1.
```

Getting formatted time

When particular output we have is lagging or is wrong. Therefore there is a need of formatting the time that we have.

Example

```
#!/usr/bin/python

import time;

localtime = time.asctime( time.localtime(time.time()) )

print "Local current time :", localtime.
```

Output

Local current time : Tue Jan 13 10:17:09 2009.

Getting calendar for a month

It helps us to perform functionality with already existing yearly as well as monthly calendar.

Example

```
#!/usr/bin/python

import calendar

cal = calendar.month(2008, 1)

print "Here is the calendar:"

print cal;
```

Output

```
Here is the calendar-

January 2008

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
```

Python functions

- These function block with key words and followed by parenthesis.
- Input should position between theses parenthesis.
- First statement might be or can be optional statements.
- Every function code block starts with semi colon (:).
- It is optional passing back function to the user.

Syntax

def functionname(parameters):

"function_docstring"

function_suite

return [expression].

Example

def printme(str):

"This prints a passed string into this function"

print str

return.

Calling of a function

Declaring function gives it a name, specifies the parameters which are to be included in the function as well as structures of the blocks of that of the code. When the basic structure of a function is decided, one can execute it just by calling it from another function or even by just directly from the Python prompt.

Example

```
#!/usr/bin/python

# Function definition is here

def printme( str ):

"This prints a passed string into this function"

print str;

return;

# Now you can call printme function

printme("I'm first call to user defined function!");

printme("Again second call to the same function");
```

Output

```
I'm first call to user defined function!

Again second call to the same function
```

Pass by reference vs. the value

In python all the parameter are passed by through references. Parameters are refereed within the function and the therefore the changes also reflects back the calling functions.

Example

```
#!/usr/bin/python

# Function definition is here
```

```
def changeme( mylist ):
"This changes a passed list into this function"
mylist.append([1,2,3,4]);
print "Values inside the function: ", mylist
return
# Now you can call changeme function
mylist = [10,20,30];
changeme( mylist );
print "Values outside the function: ", mylist
```

Output

```
Values inside the function: [10, 20, 30, [1, 2, 3, 4]]
Values outside the function: [10, 20, 30, [1, 2, 3, 4]]
```

There is various another kinds where the arguments are passed within the functions. And usual the reference is overwritten inside the function.

Example

```
#!/usr/bin/python
# Function definition is here
def changeme( mylist ):
"This changes a passed list into this function"
mylist = [1,2,3,4]; # This would assign new reference in mylist
```

```
print "Values inside the function: ", mylist

return

# Now you can call changeme function

mylist = [10,20,30];

changeme( mylist );

print "Values outside the function: ", mylist
```

Output

```
Values inside the function: [1, 2, 3, 4]

Values outside the function: [10, 20, 30]
```

Python Files I/O

One of the basic methods to produce output is by using print statement where one can pass zero and more expressions.

Example

#!/usr/bin/python

print "Python is really a great language,", "isn't it?";

Output

Python is really a great language, isn't it?

Reading keyboard input

It is a built in function with a line of text to perform various functions. These functions are as follows.

- raw_input
- input

raw_input

It reads a single line as string from the input.

Example

#!/usr/bin/python

str = raw_input("Enter your input: ");

print "Received input is : ", str

Output

Enter your input: Hello Python

Received input is: Hello Python

Input function

Its functionality is similar to the raw_input function but it has an exception that is it considers it as a python operation in whole.

Example

#!/usr/bin/python

str = input("Enter your input: ");

print "Received input is : ", str

Output

Enter your input: [x*5 for x in range(2,10,2)]

Recieved input is: [10, 20, 30, 40

File position

- The tell() tells you the present position within the file.
- The seek() changes the present file position. The from argument specifies the position to be referred from where the bytes are to be transferred.
- If it is set to 0, that means use of the beginning of the file as the particular reference position and 1 means to use the current position as the referenced position and if set to 2 then its end of the file which would be taken to be the reference position.

Example

```
#!/usr/bin/python

# Open a file

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 opened file

fo.close()
```

Output

```
Read String is : Python is

Current file position : 10

Again read String is : Python is
```

Operating System methods

os.fchdir(fd)- Change the current working directory to the directory represented by the file descriptor fd.

os.fchmod(fd, mode)- Change the mode of the file given by fd to the numeric mode.

os.fchown(fd, uid, gid)- Change the owner and group id of the file given by fd to the numeric uid and gid.

os.fdatasync(fd)- Force write of file with filedescriptor fd to disk.

os.fdopen(fd[, mode[, bufsize]])- Return an open file object connected to the file descriptor fd.

os.fpathconf(fd, name)- Return system configuration information relevant to an open file. name specifies the configurationvalue to retrieve.

os.fstat(fd)- Return status for file descriptor fd.

os.fstatvfs(fd)- Return information about the filesystem containing the file associated with file descriptor fd.

os.fsync(fd)- Force write of file with filedescriptor fd to disk.

os.ftruncate(fd, length)- Truncate the file corresponding to file descriptor so that it is at most length bytes in size.

os.getcwd()- Return a string representing the current working directory.

os.getcwdu()- Return a Unicode object representing the current working directory.

os.isatty(fd)- Return True if the file descriptor fd is open and connected to

device, else False.

os.lchflags(path, flags)- Set the flags of path to the numeric flags but do not follow symbolic links.

os.lchmod(path, mode)- Change the mode of path to the numeric mode.

os.lchown(path, uid, gid)- Change the owner and group id of path to the numeric. This function will not follow symbolic links.

os.link(src, dst)- Create a hard link pointing to src named dst.

os.listdir(path)- Return a list containing the names of the entries in the directory given by path.

os.lseek(fd, pos, how)- Set the current position of file descriptor fd to position modified.

os.lstat(path)- Like stat(), but do not follow symbolic links.

os.major(device)- Extract the device major number from a raw device number.

os.makedev(major, minor)- Compose a raw device number from the major and minor device numbers.

os.makedirs(path[, mode])- Recursive directory creation function.

os.minor(device)- Extract the device minor number from a raw device number .

os.mkdir(path[, mode])- Create a directory named path with numeric mode.

os.mkfifo(path[, mode])- Create a FIFO which is a named pipe named path with numeric mode. The default mode is octal.

os.mknod(filename[, mode, device])- Create a filesystem node that is a file, device special file or named pipe that is named filename.

os.open(file, flags[, mode])- Open the file and set various flags according to flags and possibly its mode according to mode.

os.openpty()- Open a new pseudo-terminal pair. Return a pair of file descriptors (master, slave).

os.pathconf(path, name)- Return system configuration information relevant to a named file.

os.pipe()- Create a pipe. Return a pair of file descriptors (r, w) usable for reading and writing.

os.popen(command[, mode[, bufsize]])- Open a pipe to or from command.

os.read(fd, n)- Read at most n bytes from file descriptor that would return a string containing the bytes read. If the end of the file referred to be reached so an empty string is returned.

os.readlink(path)- Return a string representing the path to which the symbolic link points.

os.remove(path)- Remove the file path.

os.removedirs(path)- Remove directories recursively.

os.rename(src, dst)- Rename the file or directory src to dst.

os.renames(old, new)- Recursive directory or file renaming function.

os.rmdir(path)- Remove the directory path

os.stat(path)- Perform a stat system call on the given path.

os.stat_float_times([newvalue])- Determine whether stat_result represents time stamps as float objects.

os.statvfs(path)- Perform a statvfs system call on the given path.

Knowledge flow: more eBooks and Apps

- *Get more Google Play Books*
- *Get more eBooks*
- *Get more apps from Google Play store*
- *Get more apps from Amazon app store*

Our eBooks on IT and Computer Science

- *Compiler Design*
- *Java Programming for Beginners*
- *Data Structures and Algorithms*
- *Database Management System*
- *Learning PHP and MySQL*
- *Learn C++ Programming*
- *Learn C Programming*
- *Beginning Linux Programming*
- *Beginning ASP.NET*
- *Information and Communication Technology*