UNIT 3

Introduction to Python

UNIT 3

Chapter 1

Getting Started

After studying this lesson, students will be able to:

- *Appreciate the use of Graphical User interface and Integrated Development Environment for creating Python programs.*
- *Work in interactive & Script mode for programming.*
- *Create and assign values to variables.*
- *Understand the concept and usage of different data types in python.*
- *Appreciate the importance and usage of different types of operator (arithmetic, Relation and logical)*
- *Create Python expression(s) and statement(s).*

Introduction

In order to tell the computer 'what you want to do', we write a program in a language which computer can understand. Though there are many different programming languages such as BASIC, Pascal, C, C++, Java, Haskell, Ruby, Python, etc. but we will study Python in this course.

Before learning the technicalities of Python, let's get familiar with it.

Python was created by Guido Van Rossum when he was working at CWI (Centrum Wiskunde & Informatica) which is a National Research Institute for Mathematics and Computer Science in Netherlands. The language was released in I991. Python got its name from a BBC comedy series from seventies- "Monty Python's Flying Circus". Python can be used to follow both Procedural approach and Object Oriented approach of programming. It is free to use.

Some of the features which make Python so popular are as follows:

- It is a general purpose programming language which can be used for both scientific and non scientific programming.
- It is a platform independent programming language.

- It is a very simple high level language with vast library of add-on modules.
- It is excellent for beginners as the language is interpreted, hence gives immediate results.
- The programs written in Python are easily readable and understandable.
- It is suitable as an extension language for customizable applications.
- It is easy to learn and use.

The language is used by companies in real revenue generating products, such as:

- In operations of Google search engine, youtube, etc.
- Bit Torrent peer to peer file sharing is written using Python
- Intel, Cisco, HP, IBM, etc use Python for hardware testing.
- Maya provides a Python scripting API
- i–Robot uses Python to develop commercial Robot.
- NASA and others use Python for their scientific programming task.

First Step with Python

We are continuously saying that Python is a programming language but don't know what a program is? Therefore, let's start Python by understanding Program.

A program is a sequence of instructions that specifies how to perform a Computation. The Computation might be mathematical or working with text.

To write and run Python program, we need to have Python interpreter installed in our computer. IDLE (GUI integrated) is the standard, most popular Python development environment. IDLE is an acronym of Integrated Development Environment. It lets edit, run, browse and debug Python Programs from a single interface. This environment makes it easy to write programs.

We will be using version 2.7 of Python IDLE to develop and run Python code, in this course. It can be downloaded from www.python.org

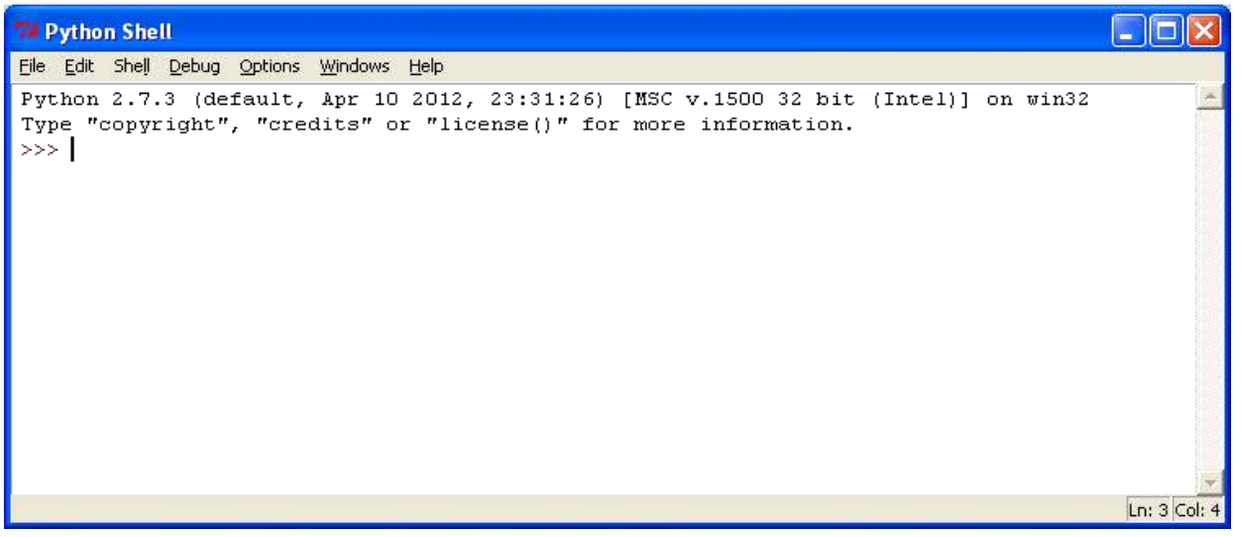
Python shell can be used in two ways, viz., interactive mode and script mode. Where Interactive Mode, as the name suggests, allows us to interact with OS; script mode let us

create and edit python source file. Now, we will first start with interactive mode. Here, we type a Python statement and the interpreter displays the result(s) immediately.

Interactive Mode

For working in the interactive mode, we will start Python on our computer. You can take the help of your Teacher.

When we start up the IDLE following window will appear:



What we see is a welcome message of Python interpreter with revision details and the Python prompt, i.e., '>>>'. This is a primary prompt indicating that the interpreter is expecting a python command. There is secondary prompt also which is '…' indicating that interpreter is waiting for additional input to complete the current statement.

Interpreter uses prompt to indicate that it is ready for instruction. Therefore, we can say, if there is prompt on screen, it means IDLE is working in interactive mode.

We type Python expression / statement / command after the prompt and Python immediately responds with the output of it. Let's start with typing print **"How are you"** after the prompt.

>>>print "How are you?"

How are you?

What we get is Python's response. We may try the following and check the response:

i) print 5+7

ii) 5+7

iii) 6*250/9

iv) print 5-7

It is also possible to get a sequence of instructions executed through interpreter.

Example 1	Example 2

>>> x=2
>>> y=6
>>> z = x+y
>>> print z
8
	>>> a=3
	>>> a+1, a-1
	(4,2) #result is tuple of 2 values

#result is tuple of 2 values, is a comment statement. We will talk about it in the later part of chapter.

Now we are good to write a small code on our own in Python. While writing in Python, remember Python is case sensitive. That means x & X are different in Python.

Note: If we want to repeat prior command in interactive window, you can use '↑' key to scroll backward through commands history and '↓' key to scroll forward. Use Enter key to select it. Using these keys, your prior commands will be recalled and displayed, and we may edit or rerun them also.

^D (Ctrl+D) or quit () is used to leave the interpreter.

^F6 will restart the shell.

Help of IDLE can be explored to know about the various menu options available for Programmer.

Apart from writing simple commands, let's explore the interpreter more.

Type **Credits** after the prompt and what we get is information about the organization involved in Python development. Similarly, **Copyright** and **Licenses** command can be used to know more about Python. Help command provides **help** on Python. It can be used as….. help() with nothing in parenthesis will allow us to enter an interactive help mode. And with a name (predefined) in bracket will give us details of the referred word.

To leave the help mode and return back to interactive mode, quit command can be used.

Script Mode

In script mode, we type Python program in a file and then use the interpreter to execute the content from the file. Working in interactive mode is convenient for beginners and for testing small pieces of code, as we can test them immediately. But for coding more than few lines, we should always save our code so that we may modify and reuse the code.

Note: Result produced by Interpreter in both the modes, viz., Interactive and script mode is exactly same.

Python, in interactive mode, is good enough to learn, experiment or explore, but its only drawback is that we cannot save the statements for further use and we have to retype all the statements to re-run them.

To create and run a Python script, we will use following steps in IDLE, if the script mode is not made available by default with IDLE environment.

1. File>Open OR File>New Window (for creating a new script file)
2. Write the Python code as function i.e. script
3. Save it (^S)
4. Execute it in interactive mode- by using RUN option (^F5)

   Otherwise (if script mode is available) start from Step 2

Note: For every updation of script file, we need to repeat step 3 & step 4

If we write Example 1 in script mode, it will be written in the following way:

Step 1: File> New Window

Step 2:

```
def test():
x=2
y=6
z = x+y
print z
```

Step 3:

Use File > Save or File > Save As - option for saving the file

(*By convention all Python program files have names which end with .py*)

Step 4:

For execution, press ^F5, and we will go to Python prompt (in other window)

```
>>> test()
8
```

Alternatively we can execute the script directly by choosing the RUN option.

Note: While working in script mode, we add 'print' statement in our program to see the results which otherwise were displayed on screen in interactive mode without typing such statements.

Variables and Types

When we create a program, we often like to store values so that it can be used later. We use objects to capture data, which then can be manipulated by computer to provide information. By now we know that object/ variable is a name which refers to a value.

Every object has:

A. An Identity, - can be known using id (object)

B. A type – can be checked using type (object) and

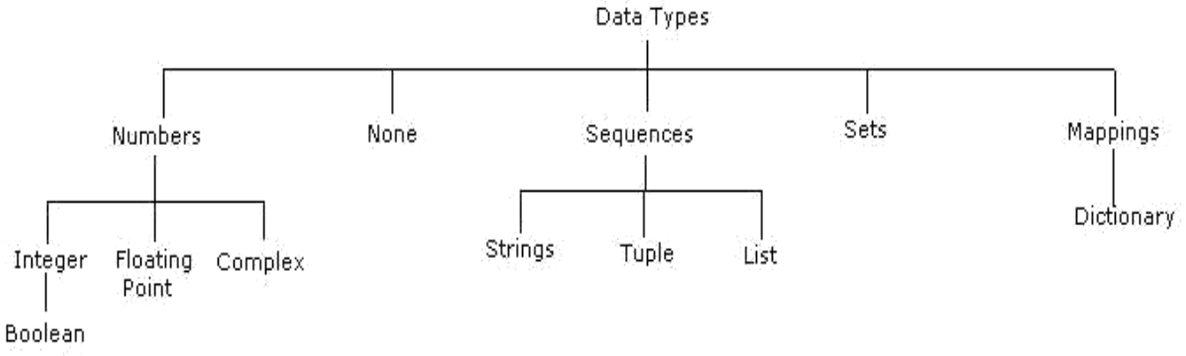
C. A value

Let us study all these in detail

A. Identity of the object: It is the object's address in memory and does not change once it has been created.

***(We would be referring to objects as variable for** now)*

B. Type (i.e data type): It is a set of values, and the allowable operations on those values. It can be one of the following:



1. Number

Number data type stores Numerical Values. This data type is immutable i.e. value of its object cannot be changed (we will talk about this aspect later). These are of three different types:

a) Integer & Long

b) Float/floating point

c) Complex

Range of an integer in Python can be from -2147483648 to 2147483647, and long integer has unlimited range subject to available memory.

1.1 Integers are the whole numbers consisting of + or – sign with decimal digits like 100000, -99, 0, 17. While writing a large integer value, don't use commas to separate digits. Also integers should not have leading zeros.

When we are working with integers, we need not to worry about the size of integer as a very big integer value is automatically handled by Python. When we want a value to be treated as very long integer value append **L** to the value. Such values are treated as long integers by python.

```
>>> a = 10
>>> b = 5192L          #example of supplying a very long value to a variable
>>> c= 4298114
>>> type(c)            # type ( ) is used to check data type of value
<type 'int'>
>>> c = c * 5669
>>> type(c)
<type 'long'>
```

We can know the largest integer in our version of Python by following the given set of commands:

```
>>> import sys
>>> print sys.maxint
```

Integers contain Boolean Type which is a unique data type, consisting of two constants, **True** & **False**. A Boolean True value is Non-Zero, Non-Null and Non-empty.

Example

```
>>> flag = True
>>> type(flag)
<type 'bool'>
```

1.2 **Floating Point:** Numbers with fractions or decimal point are called floating point numbers.

A floating point number will consist of sign (+,-) sequence of decimals digits and a dot such as 0.0, -21.9, 0.98333328, 15.2963. These numbers can also be used to represent a number in engineering/ scientific notation.

$-2.0X\ 10^5$ will be represented as -2.0e5

$2.0X10^{-5}$ will be 2.0E-5

Example

y= 12.36

A value when stored as floating point in Python will have 53 bits of precision.

1.3 **Complex:** Complex number in python is made up of two floating point values, one each for real and imaginary part. For accessing different parts of variable (object) x; we will use x.real and x.image. Imaginary part of the number is represented by 'j' instead of 'i', so 1+0j denotes zero imaginary part.

Example

```
>>> x = 1+0j
>>> print x.real,x.imag
1.0 0.0
```

Example

```
>>> y = 9-5j
>>> print y.real, y.imag
9.0 -5.0
```

2. None

This is special data type with single value. It is used to signify the absence of value/false in a situation. It is represented by **None**.

3. Sequence

A sequence is an ordered collection of items, indexed by positive integers. It is combination of mutable and non mutable data types. Three types of sequence data type available in Python are Strings, Lists & Tuples.

3.1 **String:** is an ordered sequence of letters/characters. They are enclosed in single quotes (‘ ’) or double (‘‘ “). The quotes are not part of string. They only tell the computer where the string constant begins and ends. They can have any character or sign, including space in them. These are immutable data types. We will learn about immutable data types while dealing with third aspect of object i.e. value of object.

Example

```
>>> a = 'Ram'
```

A string with length 1 represents a character in Python.

Conversion from one type to another

If we are not sure, what is the data type of a value, Python interpreter can tell us:

```
>>> type (‘Good Morning’)
<type ‘str’>
>>> type (‘3.2’)
<type ‘str’>
```

It is possible to change one type of value/ variable to another type. It is known as type conversion or type casting. The conversion can be done explicitly (programmer specifies the conversions) or implicitly (Interpreter automatically converts the data type).

For explicit type casting, we use functions (constructors):

```
int ()
float ()
str ()
bool ()
```

Example

```
>>> a= 12.34
>>> b= int(a)
>>> print b
12
```

Example

```
>>>a=25
>>>y=float(a)
>>>print y
25.0
```

3.2 **Lists:** List is also a sequence of values of any type. Values in the list are called elements / items. These are mutable and indexed/ordered. List is enclosed in square brackets.

Example

```
l = ['spam', 20.5,5]
```

3.3 **Tuples:** Tuples are a sequence of values of any type, and are indexed by integers. They are immutable. Tuples are enclosed in (). We have already seen a tuple, in Example 2 (4, 2).

4. Sets

Set is an unordered collection of values, of any type, with no duplicate entry. Sets are immutable.

Example

```
s = set ([1,2,34])
```

5. Mapping

This data type is unordered and mutable. Dictionaries fall under Mappings.

5.1 **Dictionaries:** Can store any number of python objects. What they store is a key – value pairs, which are accessed using key. Dictionary is enclosed in curly brackets.

Example

```
d = {1:'a',2:'b',3:'c'}
```

C. Value of Object (variable) – to bind value to a variable, we use assignment operator (=). This is also known as building of a variable.

Example

```
>>> pi = 31415
```

Here, value on RHS of '=' is assigned to newly created 'pi' variable.

Mutable and Immutable Variables

A mutable variable is one whose value may change in place, whereas in an immutable variable change of value will not happen in place. Modifying an immutable variable will rebuild the same variable.

Example

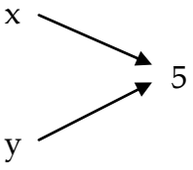
```
>>>x=5
```

Will create a value 5 referenced by x

x → 5

```
>>>y=x
```

This statement will make y refer to 5 of x




```
>>> x=x+y
```

As x being integer (immutable type) has been rebuild.

In the statement, expression on RHS will result into value 10 and when this is assigned to LHS (x), x will rebuild to 10. So now

x ⟶ 10 and

y ⟶ 5

After learning about what a variable can incorporate, let's move on with naming them. Programmers choose the names of the variable that are meaningful. A variable name:

1. Can be of any size
2. Have allowed characters, which are a-z, A-Z, 0-9 and underscore (_)
3. should begin with an alphabet or underscore
4. should not be a keyword

It is a good practice to follow these identifier naming conventions:

1. Variable name should be meaningful and short
2. Generally, they are written in lower case letters

Keywords

They are the words used by Python interpreter to recognize the structure of program. As these words have specific meaning for interpreter, they cannot be used for any other purpose.

A partial list of keywords in Python 2.7 is

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

Remember:

- Variables are created when they are first assigned a value.
- Variables must be assigned a value before using them in expression,
- Variables refer to an object and are never declared ahead of time.

Operators and Operands

Operators are special symbols which represents computation. They are applied on operand(s), which can be values or variables. Same operator can behave differently on different data types. Operators when applied on operands form an expression. Operators are categorized as Arithmetic, Relational, Logical and Assignment. Value and variables when used with operator are known as operands.

Following is the partial list of operators:

Mathematical/Arithmetic Operators

Symbol	Description	Example 1	Example 2
+	Addition	>>>55+45 100	>>> 'Good' + 'Morning' GoodMorning
-	Subtraction	>>>55-45 10	>>>30-80 -50
*	Multiplication	>>>55*45 2475	>>> 'Good'* 3 GoodGoodGood
/	Division	>>>17/5 3 >>>17/5.0 3.4 >>> 17.0/5 3.4	>>>28/3 9

%	Remainder/ Modulo	>>>17%5 2	>>> 23%2 1
**	Exponentiation	>>>2**3 8 >>>16**.5 4.0	>>>2**8 256
//	Integer Division	>>>7.0//2 3.0	>>>3/ / 2 1

Note: Division is Implementation Dependent

Relational Operators

Symbol	Description	Example 1	Example 2

<	Less than	>>>7<10
		True
		>>> 7<5
		False
		>>> 7<10<15
		True
		>>>7<10 and 10<15
		True
			>>>'Hello'< 'Goodbye'
			False
			>>>'Goodbye'< 'Hello'
			True

>	Greater than	>>>7>5 True >>>10<10 False	>>>'Hello'> 'Goodbye' True >>>'Goodbye'> 'Hello' False
<=	less than equal to	>>> 2<=5	>>>'Hello'<= 'Goodbye'

		True >>> 7<=4 False	False >>>'Goodbye' <= 'Hello' True
>=	greater than equal to	>>>10>=10 True >>>10>=12 False	>>>'Hello'>= 'Goodbye' True >>>'Goodbye' >= 'Hello' False
! =, <>	not equal to	>>>10!=11 True >>>10!=10 False	>>>'Hello'!= 'HELLO' True >>> 'Hello' != 'Hello' False
==	equal to	>>>10==10 True >>>10==11 False	>>>'Hello' == 'Hello' True >>>'Hello' == 'Good Bye' False

Note: Two values that are of different data type will never be equal to each other.

Logical Operators

Symbol	Description
or	If any one of the operand is true, then the condition becomes true.
and	If both the operands are true, then the condition becomes true.
not	Reverses the state of operand/condition.

Assignment Operators

Assignment Operator combines the effect of arithmetic and assignment operator

Symbol	Description	Example	Explanation
=	Assigned values from right side operands to left variable	>>>x=12* >>>y='greetings'	

*(*we will use it as initial value of x for following examples)*

+=	added and assign back the result to left operand	>>>x+=2	The operand/ expression/ constant written on RHS of operator is will change the value of x to 14
-=	subtracted and assign back the result to left operand	x-=2	x will become 10
*=	multiplied and assign back the result to left operand	x*=2	x will become 24
/=	divided and assign back the result to left operand	x/=2	x will become 6
%=	taken modulus using two operands and assign the result to left operand	x%=2	x will become 0
**=	performed exponential (power) calculation on operators and assign value to the left operand	x**=2	x will become 144
//=	performed floor division on operators and assign value to the left operand	x / /= 2	x will become 6

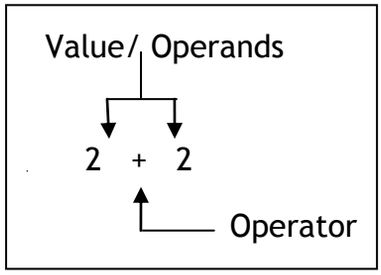
Note:

1. Same operator may perform a different function depending on the data type of the value to which it is applied.
2. Division operator '/' behaves differently on integer and float values.

Expression and Statements

An expression is a combination of value(s) (i.e. constant), variable and operators. It generates a single value, which by itself is an expression.

Example



The expression is solved by Computer and gets it value. In the above example, it will be 4, and we say the expression is evaluated.

Note: Expression values in turn can act as, Operands for Operators

We have seen many such expressions (with list of operator as example). 10+5 and 9+4+2 are two expressions which will result into value 15. Taking another example, 5.0/4+ (6-3.0) is an expression in which values of different data types are used. These type of expressions are also known as mixed type expressions.

When mixed type expressions are evaluated, Python promotes the result of lower data type to higher data type, i.e. to float in the above example. This is known as implicit type casting. So the result of above expression will be 4.25. Expression can also contain another expression. As we have already seen in 9+4+2. When we have an expression consisting of sub expression(s), how does Python decide the order of operations?

It is done based on precedence of operator. Higher precedence operator is worked on before lower precedence operator. Operator associativity determines the order of evaluation when they are of same precedence, and are not grouped by parenthesis. An operator may be Left-associative or Right –associative. In left associative, the operator falling on left side will be evaluated first, while in right assosiative operator falling on right will be evaluated first.

Note: In python '=' and '**' are Right Associative.

Precedence of operator - Listed from high precedence to low precedence.

Operator	Description
**	Exponentiation (raise to the power)
+ , -	unary plus and minus
* , /, %, //	Multiply, divide, modulo and floor division
+ , -	Addition and subtraction
<, <=, >, >=	Comparison operators
==, !=	Equality operators
% =, / =, // = , - =, + =, * =	Assignment operators
not and or	Logical operators

Using the above table, we know that 9+4 itself is an expression which evaluates to 13 and then 13+2 is evaluated to 15 by computer. Similarly, 5.0/4 + (6-3.0) will be evaluated as 5.0/4+3.0 and then to 1.25 + 3.0, and then 4.25.

If we just type 10+, we will get an error message. This happens because 10+ is not a complete expression. A complete expression will always have appropriate number of value (Operands) with each operator. '+' needs two operands and we have given just one.

Note: Remember precedence of operators is applied to find out which sub expression should be evaluated first.

Expression can be combined together to form large expressions but no matter how big the expression is, it always evaluate into a single value.

A Python statement is a unit of code that the Python interpreter can execute.

Example of statement are:

```
>>> x=5
>>> area=x**2          #assignment statement
>>>print x             #print statement
5
>>>print area
25
>>> print x, area
5  25
```

Note: To print multiple items in same line, separate them with comma.

Statements normally go to the end of a line.

X= "good morning" #comment

What we have seen as an example till now were simple statements, i.e. they do not contain a nested block. In Python, there are compound/ group statements also. They are sometimes called nested block. Statement belonging to a block are indented (usually by 4 spaces). Leading whitespace at the beginning of logical line is used to determine the indentation level of line. That means statement(s) which go together must have same indentation level.

Example of Compound Statement

Example

```
if i<0:
    print "i is negative"
else:
    print "i is non-negative"
```

Example

```
if i>0:
    print "i is positive"
else:
    print "i is equal to 0"
```

While writing Python statements, keep the following points in mind:

1. Write one python statement per line (Physical Line). Although it is possible to write two statements in a line separated by semicolon.
2. Comment starts with '#' outside a quoted string and ends at the end of a line. Comments are not part of statement. They may occur on the line by themselves or at the end of the statement. They are not executed by interpreter.
3. For a long statement, spanning multiple physical lines, we can use '/' at the end of physical line to logically join it with next physical line. Use of the '/' for joining lines is not required with expression consists of ( ), [ ], { }
4. When entering statement(s) in interactive mode, an extra blank line is treated as the end of the indented block.
5. Indentation is used to represent the embedded statement(s) in a compound/ Grouped statement. All statement(s) of a compound statement must be indented by a consistent no. of spaces (usually 4)
6. White space in the beginning of line is part of indentation, elsewhere it is not significant.

Note:

- Wrong indentation can give rise to syntax error(s).
- Most Python editor will automatically indent the statements.
- A physical line is what you see as a line when you write a program and a logical line is what Python sees as a single statement.

Input and Output

A Program needs to interact with end user to accomplish the desired task, this is done using Input-Output facility. Input means the data entered by the user (end user) of the program. While writing algorithm(s), getting input from user was represented by Take/Input. In python, we have raw-input() and input ( ) function available for Input.

raw_input()

Syntax of raw_input() is:

raw_input ([prompt])

Optional

If prompt is present, it is displayed on the monitor after which user can provide the data from keyboard. The function takes exactly what is typed from keyboard, convert it to string and then return it to the variable on LHS of '='.

Example (in interactive mode)

```
>>>x=raw_input ('Enter your name: ')
```

Enter your name: ABC

x is a variable which will get the string (ABC), typed by user during the execution of program. Typing of data for the raw_input function is terminated by 'enter' key.

We can use raw_input() to enter numeric data also. In that case we typecast, i.e., change the datatype using function, the string data accepted from user to appropriate Numeric type.

Example

y=int(raw_input("enter your roll no"))

enter your roll no. 5

will convert the accepted string i.e. 5 to integer before assigning it to 'y'.

input()

Syntax for input() is:

Input ([prompt])

Optional

If prompt is present, it is displayed on monitor, after which the user can provide data from keyboard. Input takes whatever is typed from the keyboard and evaluates it. As the input provided is evaluated, it expects valid python expression. If the input provided is not correct then either syntax error or exception is raised by python.

Example

x= input ('enter data:')

Enter data: 2+1/2.0

Will supply 2.5 to x

input ( ), is not so popular with python programmers as:

i) Exceptions are raised for non-well formed expressions.

ii) Sometimes well formed expression can wreak havoc.

Output is what program produces. In algorithm, it was represented by print. For output in Python we use print. We have already seen its usage in previous examples. Let's learn more about it.

Print Statement

Syntax:

print expression/constant/variable

Print evaluates the expression before printing it on the monitor. Print statement outputs an entire (complete) line and then goes to next line for subsequent output (s). To print more than one item on a single line, comma (,) may be used.

Example

>>> print "Hello"

Hello

>>> print 5.5

5.5

>>> print 4+6

10

Try this on the computer and evaluate the output generated

>>>print 3.14159* 7**2

>>>print "I", "am" + "class XI", "student"

>>>print "I'm",

>>>print "class XI student"

>>>print "I'm ", 16, "years old"

Comments

As the program gets bigger, it becomes difficult to read it, and to make out what it is doing by just looking at it. So it is good to add notes to the code, while writing it. These notes are known as comments. In Python, comment start with '#' symbol. Anything written after # in a line is ignored by interpreter, i.e. it will not have any effect on the program.

A comment can appear on a line by itself or they can also be at the end of line.

Example

Calculating area of a square

>>> area = side **2

or

>>>area= side**2 #calculating area of a square

For adding multi-line comment in a program, we can:

i) Place '#' in front of each line, or

ii) Use triple quoted string. They will only work as comment, when they are not being used as docstring. (A docstring is the first thing in a class/function /module, and will be taken up in details when we study functions).

The comment line "#calculating area of a rectangle" can also be written as following using triple quote:

1. """ Calculating area of a rectangle """
2. """ Calculating area

   of a rectangle """

We should use as many useful comments as we can, to explain

*Any assumptions made

*important details or decisions made in the program. This will make program more readable. We already know the importance of comments (documented in the program).

EXERCISE

1. Create following Variables

   i) 'mystring' to contain 'hello'

   ii) 'myfloat' to contain '2.5'

   iii) 'myint' to contain '10'

2. Write the value justification

   i) 2*(3+4)

   ii) 2*3+4

   iii) 2+3*4

3. What is the type of the following result:

   i) 1+2.0+3

4. Which of the following is the valid variable name:

   i) global

   ii) 99flag

   iii) sum

   iv) an$wer

5. True or False

   i) Character Data type values should be delimited by using the single quote.

   ii) None is one of the data type in python

   iii) The += operator is used to add the right hand side value to the left hand side variable.

   iv) The data type double is not a valid python data type.

   v) Python does not have any keywords

   vi) The equal to condition is written by using the == operator

6. Check all syntactically correct statements

a) Which input statements are correct

i) a = raw_input ( )

ii) a = raw_input (“enter a number”)

iii) a = raw_imput (enter your name)

b) Which print statements are correct?

i) _print “9” + “9”

ii) _print int(“nine”)

iii) _print 9+9

iv) print 9

c) Which are correct arithmetical operations?

i) a = 1*2

ii) 2 = 1+1

iii) 5 + 6 = y

iv) Seven = 3 * 4

d) Which are correct type conversions?

i) int (7.0+0.1)

ii) str (1.2 * 3.4)

iii) float (“77”+“.0”)

iv) str (9 / 0)

e) Which operations result in 8?

i) 65 // 8

ii) 17 % 9

iii) 2 * * 4

iv) 64 * * 0.5

f) Which lines are commented?

i) """This is a comment"""

ii) # This is a comment

iii) // this is a comment

iv) ' ' ' This is a comment' ' '

g) Find the matching pairs of expressions and values.

i)	1023	boolean
ii)	None	int
iii)	[2, 4, 8, 16]	tuple
iv)	True	list
v)	17.54	str
vi)	('Roger', 1952)	NoneType
vii)	"my fat cat"	float

7. MCQ

i) The _____ data type allows only True/False values

a) bool b) boolean c) Boolean d) None

ii) If the value of a = 20 and b = 20, then a+=b will assign _____ to a

a) 40 b) 30 c) 20 d) 10

iii) The _____ operator is used to find out if division of two number yields any remainder

a) / b) + c) % d) //

8. When will following statement in interpreter result into error:

>>> B+4

9. How can we change the value of 6*1-2 to -6 from 4?

10. Is python case sensitive?

11. What does 'immutable' mean; which data type in python are immutable.
12. Name four of Python's Basic data types? Why are they called so?
13. What are relational operators? Explain with the help of examples.
14. What is an integer?
15. What is a variable? What names may variable have?
16. How are keywords different from variable names?
17. Why are data types important?
18. How can you convert a string to integer and when can it be used?
19. How can text be read from the keyboard?
20. How are comments written in a program?

LAB EXERCISE

1. Record what happens when following statements are executed:

   a) print n=7

   b) print 5+7

   c) print 5.2, "this", 4-2, "that", 5/2.0

2. Use IDLE to calculate:

   a) 6+4*10

   b) (6+4)*10

3. Type following mathematical expression and record your observations:

   a) 2**500

   b) 1/0

4. What will be the output of the following code:

```
a = 3 - 4 + 10

b = 5 * 6
```

```
c = 7.0/8.0

print "These are the values:", a, b, c
```

5. Write a code to show the use of all 6 math function.

6. Write a code that prints your full name and your Birthday as separate strings.

7. Write a program that asks two people for their names; stores the names in variables called name1 and name2; says hello to both of them.

8. Calculate root of the following equation:

   a) $34x^2 + 68x - 510$

   b) $2x^2 - x - 3 = 0$

Chapter 2

Functions

After studying this lesson, students will be able to:

- *Understand and apply the concept of module programming*
- *Write functions*
- *Identify and invoke appropriate predefined functions*
- *Create Python functions and work in script mode.*
- *Understand arguments and parameters of functions*
- *Work with different types of parameters and arguments*
- *Develop small scripts involving simple calculations*

Introduction

Remember, we earlier talked about working in script mode in chapter-1 of this unit to retain our work for future usage. For working in script mode, we need to *write a function* in the Python and save it in the file having **.py** extension.

A function is a named sequence of statement(s) that performs a computation. It contains line of code(s) that are executed sequentially from top to bottom by Python interpreter. They are the most important building blocks for any software in Python.

Functions can be categorized as belonging to

i. Modules

ii. Built in

iii. User Defined

Module

A module is a file containing Python definitions (i.e. functions) and statements. Standard library of Python is extended as module(s) to a programmer. Definitions from the module can be used within the code of a program. To use these modules in the

program, a programmer needs to import the module. Once you import a module, you can reference (use), any of its functions or variables in your code. There are many ways to import a module in your program, the one's which you should know are:

i. import

ii. from

Import

It is simplest and most common way to use modules in our code. Its syntax is:

import modulename1 [,modulename2, ---------]

Example

```
>>> import math
```

On execution of this statement, Python will

(i) search for the file '**math.py**'.

(ii) Create space where modules definition & variable will be created,

(iii) then execute the statements in the module.

Now the definitions of the module will become part of the code in which the module was imported.

To use/ access/invoke a function, you will specify the module name and name of the function- separated by dot (.). This format is also known as *dot notation*.

Example

```
>>> value= math.sqrt (25)          # dot notation
```

The example uses sqrt( ) function of module **math** to calculate square root of the value provided in parenthesis, and returns the result which is inserted in the ***value***. The expression (variable) written in parenthesis is known as argument (actual argument). It is common to say that the function takes arguments and return the result.

This statement invokes the sqrt ( ) function. We have already seen many function invoke statement(s), such as

>>> type ( )

>>> int ( ), etc.

From Statement

It is used to get a specific function in the code instead of the complete module file. If we know beforehand which function(s), we will be needing, then we may use **from**. For modules having large no. of functions, it is recommended to use **from** instead of import. Its syntax is

>>> from modulename import functionname [, functionname…..]

Example

>>> from math import sqrt

value = sqrt (25)

Here, we are importing sqrt function only, instead of the complete math module. Now sqrt( ) function will be directly referenced to. These two statements are equivalent to previous example.

from modulename import *

will import everything from the file.

Note: You normally put all import statement(s) at the beginning of the Python file but technically they can be anywhere in program.

Lets explore some more functions available in **math module:**

Name of the function	Description	Example
ceil(x)	It returns the smallest integer not less than x, where *x is a numeric expression.*	math.ceil(-45.17) **-45.0** math.ceil(100.12) **101.0** math.ceil(100.72) **101.0**

floor(x)	It returns the largest integer not greater than x, where *x is a numeric expression.*	math.floor(-45.17) **-46.0** math.floor(100.12) **100.0** math.floor(100.72) **100.0**
fabs(x)	It returns the absolute value of x, *where x is a numeric value.*	math.fabs(-45.17) **45.17** math.fabs(100.12) **100.12** math.fabs(100.72) **100.72**
exp(x)	It returns exponential of x: e^x, where x is a numeric expression.	math.exp(-45.17) **2.41500621326e-20** math.exp(100.12) **3.03084361407e+43** math.exp(100.72) **5.52255713025e+43**
log(x)	It returns natural logarithm of x, for x > 0, where *x is a numeric expression.*	math.log(100.12) **4.60636946656** math.log(100.72) **4.61234438974**
log10(x)	It returns base-10 logarithm of x for x > 0, where *x is a numeric expression.*	math.log10(100.12) **2.00052084094** math.log10(100.72) **2.0031157171**
pow(x, y)	It returns the value of x^y, *where x and* ***y*** *are numeric expressions.*	math.pow(100, 2) **10000.0** math.pow(100, -2)

		0.0001 math.pow(2, 4) **16.0** math.pow(3, 0) **1.0**
sqrt (x)	It returns the square root of x for x > 0, where x is a numeric expression.	math.sqrt(100) **10.0** math.sqrt(7) **2.64575131106**
cos (x)	It returns the cosine of x in radians, *where x is a numeric expression*	math.cos(3) **-0.9899924966** math.cos(-3) **-0.9899924966** math.cos(0) **1.0** math.cos(math.pi) **-1.0**
sin (x)	It returns the sine of x, in radians, *where x must be a numeric value.*	math.sin(3) **0.14112000806** math.sin(-3) **-0.14112000806** math.sin(0) **0.0**
tan (x)	It returns the tangent of x in radians, *where x must be a numeric value.*	math.tan(3) **-0.142546543074** math.tan(-3) **0.142546543074** math.tan(0) **0.0**

degrees (x)	It converts angle x from radians to degrees, *where x must be a numeric value.*	math.degrees(3) **171.887338539** math.degrees(-3) **-171.887338539** math.degrees(0) **0.0**
radians(x)	It converts angle x from degrees to radians, *where x must be a numeric value.*	math.radians(3) **0.0523598775598** math.radians(-3) **-0.0523598775598** math.radians(0) **0.0**

Some functions from **random module** are:

Name of the function	Description	Example
random ( )	It returns a random float x, such that 0 ≤ x<1	>>>random.random ( ) **0.281954791393** >>>random.random ( ) **0.309090465205**

randint (a, b)	It returns a int x between a & b such that
	a ≤ x ≤ b
		>>> random.randint (1,10)
		5
		>>> random.randint (-2,20)
		-1

uniform (a,b)	It returns a floating point number x, such that a <= x < b	>>>random.uniform (5, 10) **5.52615217015**

randrange ([start,] stop [,step])	It returns a random item from the given range	>>>random.randrange(100,1000,3) **150**

Some of the other modules, which you can explore, are: string, time, date

Built in Function

Built in functions are the function(s) that are built into Python and can be accessed by a programmer. These are always available and for using them, we don't have to import any module (file). Python has a small set of built-in functions as most of the functions have been partitioned to modules. This was done to keep core language precise.

Name	Description	Example
abs (x)	It returns distance between x and zero, where *x is a numeric expression.*	>>>abs(-45) **45** >>>abs(119L) **119**
max(x, y, z, ...)	It returns the largest of its arguments: where x, y and z are numeric variable/expression.	>>>max(80, 100, 1000) **1000** >>>max(-80, -20, -10) **-10**

min(x, y, z, ...)	It returns the smallest of its arguments; where x, y, and z are numeric variable/expression.	>>> min(80, 100, 1000)
		80
		>>> min(-80, -20, -10)
		-80

cmp(x, y)	It returns the sign of the difference of two numbers: -1 if x < y, 0 if x == y, or 1 if x > y, *where x and y are numeric variable/expression.*	>>>cmp(80, 100) **-1** >>>cmp(180, 100) **1**

divmod (x,y)	Returns both quotient and remainder by division through a tuple, when x is divided by y; where x & y are variable/expression.	>>> divmod (14,5)
		(2,4)
		>>> divmod (2.7, 1.5)
		(1.0, 1.20000)

len (s)	Return the length (the number of items) of an object. The argument may be a sequence (string, tuple or list) or a mapping (dictionary).	>>> a= [1,2,3]
		>>>len (a)
		3
		>>> b= 'Hello'
		>>> len (b)
		5

range (*start*, *stop*[, *step*])	This is a versatile function to create lists containing arithmetic progressions. It is most often used in for loops. The arguments must be plain integers. If the *step* argument is omitted, it defaults to 1. If the *start* argument is omitted, it defaults to 0. The full form returns a list of plain integers [start, start + step, start + 2 * step, ...]. If *step* is positive, the last element is the largest start + i * step less than *stop*; if *step* is negative, the last element is the smallest start + i * step greater than *stop*. *step* must not be zero (or else Value Error is raised).	>>> range(10)
		[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
		>>> range(1, 11)
		[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
		>>> range(0, 30, 5)
		[0, 5, 10, 15, 20, 25]
		>>> range(0, 10, 3)
		[0, 3, 6, 9]
		>>> range(0, -10, -1)
		[0, -1, -2, -3, -4, -5, -6, -7, -8, -9]
		>>> range(0)
		[]
		>>> range(1, 0)
		[]

round(x [, n])	It returns float x rounded to n digits from the decimal point, *where x and n are numeric expressions.* If n is not provided then x is rounded to 0 decimal digits.	>>>round(80.23456, 2) **80.23** >>>round(-100.000056, 3) **-100.0** >>> round (80.23456) **80.0**

Apart from these functions, you have already seen the use of the following functions:

bool ( ), chr ( ), float ( ), int ( ), long (), str ( ), type ( ), id ( ), tuple ( )

Composition

Composition is an art of combining simple function(s) to build more complicated ones, i.e., result of one function is used as the input to another.

Example

Suppose we have two functions fn1 & fn2, such that

a= fn2 (x)

b= fn1 (a)

then call to the two functions can be combined as

b= fn1 (fn2 (x))

Similarly, we can have statement composed of more than two functions. In that result of one function is passed as argument to next and result of the last one is the final result.

Example

math.exp (math.log (a+1))

Example

degrees=270

math.sin (degrees/360.0 *2*math.pi)

Composition is used to package the code into modules, which may be used in many different unrelated places and situations. Also it is easy to maintain the code.

Note: Python also allow us to take elements of program and compose them.

User Defined Functions

So far we have only seen the functions which come with Python either in some file (module) or in interpreter itself (built in), but it is also possible for programmer to write their own function(s). These functions can then be combined to form a module which can then be used in other programs by importing them.

To define a function keyword **def** is used. After the keyword comes an identifier i.e. name of the function, followed by parenthesized list of parameters and the colon which ends up the line. Next follows the block of statement(s) that are the part of function.

Before learning about Function header & its body, lets explore block of statements, which become part of function body.

Block of statements

A block is one or more lines of code, grouped together so that they are treated as one big sequence of statements while executing. In Python, statements in a block are written with indentation. Usually, a block begins when a line is indented (by four spaces) and all the statements of the block should be at same indent level. A block within block begins when its first statement is indented by four space, i.e., in total eight spaces. To end a block, write the next statement with the same indentation before the block started.

Now, lets move back to function- the **Syntax** of function is:

def NAME ([PARAMETER1, PARAMETER2, …..]): #Square brackets include
statement(s) #optional part of statement

Let's write a function to greet the world:

```
def sayHello ():                # Line No. 1
    print "Hello World!"        # Line No.2
```

The first line of function definition, i.e., Line No. 1 is called **header** and the rest, i.e. Line No. 2 in our example, is known as **body**. Name of the function is ***sayHello***, and empty parenthesis indicates no parameters. Body of the function contains one Python statement, which displays a string constant on screen. So the general structure of any function is

Function Header

It begins with the keyword def and ends with colon and contains the function identification details. As it ends with colon, we can say that what follows next is, block of statements.

Function Body

Consisting of sequence of indented (4 space) Python statement(s), to perform a task.

Defining a function will create a variable with same name, but does not generate any result. The body of the function gets executed only when the function is called/invoked. Function **call** contains the name of the function (being executed) followed by the list of values (i.e. arguments) in parenthesis. These arguments are assigned to parameters from LHS.

```
>>> sayHello ()        # Call/invoke statement of this function
```

Will produce following on screen

Hello World!

Apart from this, you have already seen many examples of invoking of functions in Modules & Built-in Functions.

Let's know more about **def.** It is an executable statement. At the time of execution a function is created and a name (name of the function) is assigned to it. Because it is a statement, **def** can appear anywhere in the program. It can even be nested.

Example

```
if condition:
    def fun ( ):                # function definition one way
.
```

```
.
.
else:
        def fun ( ):                # function definition other way
.
.
.
fun ( )                 #  calls the function selected.
```

This way we can provide an alternative definition to the function. This is possible because def is evaluated when it is reached and executed.

```
def fun (a):
```

Let's explore Function body

The first statement of the function body can optionally be a string constant, **docstring,** enclosed in triple quotes. It contains the essential information that someone might need about the function, such as

- What function does (**without How it does**) i.e. summary of its purpose
- Type of parameters it takes
- Effect of parameter on behavior of functions, etc.

DocString is an important tool to document the program better, and makes it easier to understand. We can actually access docstring of a function using __ doc__ (function name). Also, when you used help(), then Python will provide you with docstring of that function on screen. So it is strongly recommended to use docstring … when you write functions.

Example

```
def area (radius):
    """ calculates area of a circle.                    docstring begins
    require an integer or float value to calculate area.
    returns the calculated value to calling function """    docstring ends
```

```
    a=radius**2
    return a
```

Function is pretty simple and its objective is pretty much clear from the docString added to the body.

The last statement of the function, i.e. return statement returns a value from the function. Return statement may contain a constant/literal, variable, expression or function, if return is used without anything, it will return **None.** In our example value of a variable **area** is returned.

Instead of writing two statements in the function, i.e.

```
    a = radius **2
    return a
```

We could have written

```
    return radius **2
```

Here the function will first calculate and then return the value of the expression.

It is possible that a function might not return a value, as sayHello( ) was not returning a value. sayHello( ) prints a message on screen and does not contain a return statement, such functions are called **void functions**.

Void functions might display something on the screen or have some other effect, but they don't have a return value. If you try to assign the result of such function to a variable, you get a special value called **None**.

Example

```
    def check (num):
        if (num%2==0):
        print "True"
  else:
        print "False"
>>> result = check (29)
```

False

>>> print result

None

DocString Conventions:

- The first line of a docstring starts with capital letter and ends with a period (.)
- Second line is left blank (it visually separates summary from other description).
- Other details of docstring start from 3rd line.

Parameters and Arguments

Parameters are the value(s) provided in the parenthesis when we write function header. These are the values required by function to work. Let's understand this with the help of function written for calculating area of circle.

radius is a parameter to function area.

If there is more than one value required by the function to work on, then, all of them will be listed in parameter list separated by comma.

Arguments are the value(s) provided in function call/invoke statement. List of arguments should be supplied in same way as parameters are listed. Bounding of parameters to arguments is done 1:1, and so there should be same number and type of arguments as mentioned in parameter list.

Example

of argument in function call

>>> area (5)

5 is an argument. An argument can be constant, variable, or expression.

Scope of Variables

Scope of variable refers to the part of the program, where it is visible, i.e., area where you can refer (use) it. We can say that scope holds the current set of variables and their values. We will study two types of scope of variables- global scope or local scope.

Global Scope

A variable, with global scope can be used anywhere in the program. It can be created by defining a variable outside the scope of any function/block.

Example

```
x=50
def test ( ):
    print "Inside test x is" , x
print "Value of x is" , x
```

on execution the above code will produce

Inside test x is 50

Value of x is 50

Any modification to global is permanent and visible to all the functions written in the file.

Example

```
x=50
def test ( ):
    x+= 10
    print "Inside test x is", x
print "Value of x is", x
```

will produce

Inside test x is 60

Value of x is 60

Local Scope

A variable with local scope can be accessed only within the function/block that it is created in. When a variable is created inside the function/block, the variable becomes local to it. A local variable only exists while the function is executing.

Example

```
X=50
def test ( ):
    y = 20
    print 'Value of x is ', X, ';  y is ' , y
print 'Value of x is ', X, ' y is ' , y
```

On executing the code we will get

Value of x is 50; y is 20

The next print statement will produce an error, because the variable **y** is not accessible outside the function body.

A global variable remains global, till it is not recreated inside the function/block.

Example

```
x=50
def test ( ):
    x=5
    y=2
    print 'Value of x & y inside the function are ' , x , y
print 'Value of x outside the function is ' , x
```

This code will produce following output:

Value of x & y inside the function are 5 2

Value of x outside the function is 50

If we want to refer to global variable inside the function then keyword global will be prefixed with it.

Example

```
x=50
```

```
def test ( ):
    global x
    x =2
    y = 2
    print 'Value of x & y inside the function are ' , x , y
print 'Value of x outside function is ' , x
```

This code will produce following output:

Value of x & y inside the function are 2 2

Value of x outside the function is 2

More on defining Functions

It is possible to provide parameters of function with some default value. In case the user does not want to provide values (argument) for all of them at the time of calling, we can provide default argument values.

Example

```
def greet (message, times=1):
    print message * times
>>> greet ('Welcome')        # calling function with one argument value
>>> greet ('Hello', 2)       # calling function with both the argument values.
```

Default value to parameter

Will result in:

Welcome

HelloHello

The function **greet ()** is used to print a message (string) given number of times. If the second argument value, is not specified, then parameter **times** work with the default value provided to it. In the first call to greet ( ), only one argument value is provided, which is passed on to the first parameter from LHS and the **string is printed only once**

as the variable times take default value 1. In the second call to greet ( ), we supply both the argument values a **string** and **2**, saying that we want to print the message twice. So now, parameter **times** get the value **2** instead of default 1 and the message is printed twice.

As we have seen functions with default argument values, they can be called in with fewer arguments, then it is designed to allow.

Note:

- The default value assigned to the parameter should be a constant only.
- Only those parameters which are at the end of the list can be given default value. You cannot have a parameter on left with default argument value, without assigning default values to parameters lying on its right side.
- The default value is evaluated only once, at the point of function definition.

If there is a function with many parameters and we want to specify only some of them in function call, then value for such parameters can be provided by using their **name**, instead of the position (order)- this is called **keyword arguments**.

```
def fun(a, b=1, c=5):
    print 'a is ', a, 'b is ', b, 'c is ', c
```

The function fun can be invoked in many ways

1. >>>fun (3)

   a is 3 b is 1 c is 5

2. >>>fun (3, 7, 10)

   a is 3 b is 7 c is 10

3. >>>fun (25, c = 20)

   a is 25 b is 1 c is 20

4. >>>fun (c = 20, a = 10)

   a is 10 b is 1 c is 20

1st and 2nd call to function is based on default argument value, and the 3rd and 4th call are using **keyword arguments.**

In the first usage, value 3 is passed on to **a, b** & **c** works with default values. In second call, all the three parameters get values in function call statement. In third usage, variable **a** gets the first value 25, due to the position of the argument. And parameter **c** gets the value 20 due to naming, i.e., keyword arguments. The parameter **b** uses the default value.

In the fourth usage, we use keyword argument for all specified value, as we have specified the value for **c** before **a**; although **a** is defined before **c** in parameter list.

Note: The function named fun ( ) have three parameters out of which first one is without default value and other two have default values. So any call to the function should have at least one argument.

While using keyword arguments, following should be kept in mind:

- An argument list must have any positional arguments followed by any keywords arguments.
- Keywords in argument list should be from the list of parameters name only.
- No parameter should receive value more than once.
- Parameter names corresponding to positional arguments cannot be used as keywords in the same calls.

Following calls to fun () would be invalid

```
fun ()                  # required argument missing
fun (5, a=5, 6)   # non keyword argument (6) following keyword argument
fun (6, a=5)            # duplicate value for argument a
fun (d=5)               # unknown parameter
```

Advantages of writing functions with keyword arguments are:

- Using the function is easier as we do not need to remember about the order of the arguments.
- We can specify values of only those parameters to which we want to, as - other parameters have default argument values.

In python, as function definition happens at run time, so functions can be bound to other names. This allow us to

(i) Pass function as parameter

(ii) Use/invoke function by two names

Example

```
def x ( ):
    print 20
>>> y=x
>>>x ( )
>>>y ( )
20
```

Example

```
def x ( ):
    print 20
    def test (fn):
    for I in range (4):
    fn( )
    >>> test (x)
    20
    20
```

20

20

Flow of Execution of program containing Function call

Execution always begins at the first statement of the program. Statements are executed one at a time, in order from top to bottom. Function definition does not alter the flow of execution of program, as the statement inside the function is not executed until the function is called.

On a function call, instead of going to the next statement of program, the control jumps to the body of the function; executes all statements of the function in the order from top to bottom and then comes back to the point where it left off. This remains simple, till a function does not call another function. Simillarly, in the middle of a function, program might have to execute statements of the other function and so on.

Don't worry; Python is good at keeping track of execution, so each time a function completes, the program picks up from the place it left last, until it gets to end of program, where it terminates.

Note:

- Python does not allow you to call a function before the function is declared.
- When you write the name of a function without parenthesis, it is interpreted as the reference, when you write the function name with parenthesis, the interpreter invoke the function (object).

EXERCISE

1. The place where a variable can be used is called its

   a) area b) block

   c) function d) Scope

2. True or False

   i. Every variable has a scope associated with it.

   ii. ! (p or q) is same as !p or !q

3. What will be the output of the following? Explain:

```
def f1 ( ):
    n = 44
def f2( ):
    n=77
    print "value of n", n
    print "value of n", n
```

4. For each of the following functions. Specify the type of its **output**. You can assume each function is called with an appropriate argument, as specified by its docstrings.

   a)
```
def a (x):
    '''
    x: int or float.
    '''
    return x+1
```

   b)
```
def b (x):
    '''
    x: int or float.
    '''
```

```
    return x+1.0
```

c)
```
def c (x, y):
    '''
    x: int or float.
    y: int or float.
    '''
    return x+y
```

d)
```
def e (x, y,z):
    '''
    x: can be of any type.
    y: can be of any type.
    z: can be of any type
    '''
    return x >= y and x <= z
```

e)
```
def d (x,y):
    '''
    x: can be of any type.
    y: can be of any type.
    '''
    return x > y
```

5. Below is a transcript of a session with the Python shell. Assume the functions in previous question (Q 4) have been defined. Provide the type and value of the expressions being evaluated.

i) a (6)

ii) a (-5. 3)

iii) a (a(a(6)))

iv) c (a(1), b(1))

v) d ('apple', 11.1)

6. Define a function **get Bigger Number (x,y)** to take in two numbers and return the bigger of them.

7. What is the difference between methods, functions & user defined functions.

8. Open help for math module

   i. How many functions are there in the module?

   ii. Describe how square root of a value may be calculated without using a math module

   iii. What are the two data constants available in math module.

9. Generate a random number *n* such that

   i. $0 \leq n < 6$

   ii. $2 \leq n < 37$ and *n* is even

LAB EXERCISE

1. Write a program to ask for following as input

   Enter your first name: Rahul

   Enter your last name: Kumar

   Enter your date of birth

   Month? March

   Day? 10

   Year? 1992

   And display following on screen

   Rahul Kumar was born on March 10, 1992.

2. Consider the following function definition:

```
def intDiv (x, a):
    """
    x: a non-negative integer argument
```

a: a positive integer argument

returns: integer, the integer division of x divided by a.

"""

while x>=a:

count +=1

x = x-a

return count

when we call

print intDiv (5, 3)

We get an error message. Modify the code so that error does not occur.

3. Write a script that asks a user for a number. Then adds 3 to that number, and then multiplies the result by 2, subtracts twice the original number, then prints the result.

4. In analogy to the example, write a script that asks users for the temperature in F and prints the temperature in C. (Conversion: Celsius = (F - 32) * 5/9).

5. Write a Python function, odd, that takes in one number and returns True when the number is odd and False otherwise. You should use the % (mod) operator, not **if**.

6. Define a function 'SubtractNumber(x,y)' which takes in two numbers and returns the difference of the two.

7. Write a Python function, fourthPower( ), that takes in one number and returns that value raised to the fourth power.

8. Write a program that takes a number and calculate and display the log, square, sin and cosine of it.

9. a) Write a program, to display a tic-tac-toe board on screen, using print statement.

   b) Write a program to display a tic-tac-toe board on screen using variables, so that you do not need to write many print statements?

10. Write a function roll_D ( ), that takes 2 parameters- the no. of sides (with default value 6) of a dice, and the number of dice to roll-and generate random roll values for each dice rolled. Print out each roll and then return one string "That's all".

    Example roll_D (6, 3)

    4

    1

    6

    That's all

Chapter 3

Conditional and Looping Construct

After studying this lesson, students will be able to:

- *Understand the concept and usage of selection and iteration statements.*
- *Know various types of loops available in Python.*
- *Analyze the problem, decide and evaluate conditions.*
- *Will be able to analyze and decide for an appropriate combination of constructs.*
- *Write code that employ decision structures, including those that employ sequences of decision and nested decisions.*
- *Design simple applications having iterative nature.*

Control Flow Structure

Such as depending on time of the day you wish Good Morning or Good Night to people. Similarly while writing program(s), we almost always need the ability to check the condition and then change the course of program, the simplest way to do so is using **if** statement

```
if x > 0:

    print 'x is positive'
```

Here, the Boolean expression written after **if** is known as condition, and if Condition is **True,** then the statement written after, is executed. Let's see the syntax of **if** statement

Option 1	Option 2
if condition: STATEMENTs- **BLOCK 1** [else:	if condition-1: STATEMENTs- **BLOCK 1** [elif condition-2:

STATEMENTs- **BLOCK 2]**	STATEMENTs- **BLOCK 2** else: STATEMENTs- **BLOCK N]**

Statement with in [ ] bracket are optional.

Let us understand the syntax, in Option 1- if the condition is **True** (i.e. satisfied), the statement(s) written after **if** (i.e. STATEMENT-BLOCK 1) is executed, otherwise statement(s) written after else (i.e. STATEMENT-BLOCK 2) is executed. Remember **else** clause is optional. If provided, in any situation, one of the two blocks get executed not both.

We can say that, 'if' with 'else' provides an alternative execution, as there are two possibilities and the condition determines which one gets executed. If there are more than two possibilities, such as based on percentage print grade of the student.

Percentage Range	**Grade**
> 85	A
> 70 to <=85	B
> 60 to <=70	C
> 45 to <=60	D

Then we need to chain the **if** statement(s). This is done using the 2nd option of **if** statement. Here, we have used **'elif'** clause instead of 'else'. **elif** combines **if else- if else** statements to one **if elif …else**. You may consider elif to be an abbreviation of *else if*. There is no limit to the number of 'elif' clause used, but if there is an 'else' clause also it has to be at the end.

Example for combining more than one condition:

```
if perc > 85:
    print 'A'
elif perc >70 and perc <=85:        #alternative to this is if 70 <perc<85
```

```
        print 'B'
elif perc > 60 and perc <=70:          #if 60 <perc <=70
        print 'C'
elif perc >45 and perc <=60:
        print 'D'
```

In the chained conditions, each condition is checked in order– if previous is F**alse** then next is checked, and so on. If one of them is **True** then corresponding block of statement(s) are executed and the statement ends i.e., control moves out of 'if statement'. If none is true, then else block gets executed if provided. *If more than one condition is true, then only the first true option block gets executed.*

If you look at the conditional construct, you will find that it has same structure as function definition, terminated by a colon. Statements like this are called compound statements. In any compound statement, there is no limit on how many statements can appear inside the body, but there has to be at least one. Indentation level is used to tell Python which statement (s) belongs to which block.

There is another way of writing a simple if else statement in Python. The complete simple if, can be written as:

Variable= variable 1 if condition else variable 2.

In above statement, on evaluation, if condition results into True then variable 1 is assigned to Variable otherwise variable 2 is assigned to Variable.

Example

```
>>> a =5
>>> b=10
>>> x = True
>>> y = False
>>>result = x if a <b else y
```

Will assign **True** to result

Sometimes, it is useful to have a body with no statements, in that case you can use **pass** statement. Pass statement does nothing.

Example

```
if condition:
    pass
```

It is possible to have a condition within another condition. Such conditions are known as **Nested Condition.**

Example

```
if x==y:
    print x, ' and ', y, ' are equal'
else:
    if x<y:
        print x, ' is less than ', y
    else:
        print x, ' is greater than ', y
```

Nested if

Here a complete **if… else** statement belongs to else part of outer if statement.

Note: The condition can be any Python expression (i.e. something that returns a value). Following values, when returned through expression are considered to be False:

None, Number Zero, A string of length zero, an empty collection

Looping Constructs

We know that computers are often used to automate the repetitive tasks. One of the advantages of using computer to repeatedly perform an identical task is that –it is done without making any mistake. Loops are used to repeatedly execute the same code in a program. Python provides two types of looping constructs:

1) While statement

2) For statement

While Statements

Its syntax is:

while condition: **# condition is Boolean expression returning True or False**

STATEMENTs BLOCK 1

[else: **# optional part of while**

STATEMENTs BLOCK 2]

We can see that while looks like if statement. The statement bExampleins with keyword **while** followed by boolean condition followed by colon (:). What follows next is block of statement(s).

The statement(s) in BLOCK 1 keeps on executing till condition in while remains **True;** once the condition becomes **False** and if the else clause is written in while, then else will get executed. While loop may not execute even once, if the condition evaluates to false, initially, as the condition is tested before entering the loop.

Example

a loop to print nos. from 1 to 10

```
i=1
while (i <=10):
    print i,
    i = i+1          #could be written as i+=1
```

You can almost read the statement like English sentence. The first statement initialized the variable (controlling loop) and then **while** evaluates the condition, which is True so the block of statements written next will be executed.

Last statement in the block ensures that, with every execution of loop, **loop control variable** moves near to the termination point. If this does not happen then the loop will keep on executing infinitely.

As soon as **i becomes 11**, condition in while will evaluate to **False** and this will terminate the loop. Result produced by the loop will be:

1 2 3 4 5 6 7 8 9 10

As there is ',' after print i all the values will be printed in the same line

Example

```
i=1
while (i <=10):
    print i,
    i+ =1
else:
    print                # will bring print control to next printing line
    print "coming out of loop"
```

Will result into

1 2 3 4 5 6 7 8 9 10

coming out of loop

Nested loops

Block of statement belonging to while can have another while statement, i.e. a while can contain another while.

Example

```
i=1
while i<=3:
    j=1
    while j<=i:
    print j,        # inner while loop
    j=j+1
```

```
        print
        i=i+1
```

will result into

1

1 2

1 2 3

For Statement

Its Syntax is

for TARGET- LIST in EXPRESSION-LIST:

STATEMENT BLOCK 1

[else: # optional block

STATEMENT BLOCK 2]

Example

```
# loop to print value 1 to 10
for i in range (1, 11, 1):
    print i,
```

Execution of the loop will result into

1 2 3 4 5 6 7 8 9 10

Let's understand the flow of execution of the statement:

The statement introduces a function range ( ), its syntax is

range(start, stop, [step]) # step is optional

range( ) generates a list of values starting from **start** till **stop-1**. Step if given is added to the value generated, to get next value in the list. *You have already learnt about it in built-in functions.*

Let's move back to the for statement: **i** is the variable, which keeps on getting a value generated by range ( ) function, and the block of statement (s) are worked on for each

value of **i**. As the last value is assigned to **i**, the loop block is executed last time and control is returned to next statement. If **else** is specified in **for** statement, then next statement executed will be else. Now we can easily understand the result of **for** statement. range( ) generates a list from 1, 2, 3, 4, 5, …., 10 as the step mentioned is 1, **i** keeps on getting a value at a time, which is then printed on screen.

Apart from range( ) i (loop control variable) can take values from string, list, dictionary, etc.

Example

```
for letter in 'Python':
    print 'Current Letter', letter
else:
    print 'Coming out of loop'
```

On execution, will produce the following:

Current Letter: P

Current Letter: y

Current Letter: t

Current Letter: h

Current Letter: o

Current Letter: n

Coming out of loop

A **for** statement can contain another **for** statement or **while** statement. We know such statement form nested loop.

Example

```
# to print table starting from 1 to specified no.
n=2
for i in range (1, n+1):
    j=1
```

```
        print "Table to ", i, "is as follows"
        while j <6:
print i, "*", j "=", i*j
        j = j+1
        print
```

Will produce the result

Table to 1 is as follows

1 * 1 = 1

1 * 2 = 2

1 * 3 = 3

1 * 4 = 4

1 * 5 = 5

Table to 2 is as follows

2 * 1 = 2

2 * 2 = 4

2 * 3 = 6

2 * 4 = 8

2 * 5 = 10

Nesting a **for loop** within **while loop** can be seen in following example :

Example

```
i = 6
while i >= 0:
        for j in range (1, i):
                print j,
        print
```

```
        i=i-1
```

will result into

1 2 3 4 5

1 2 3 4

1 2 3

1 2

1

By now, you must have realized that, Syntax of **for statement** is also same as **if statement** or **while statement.**

Let's look at the equivalence of the two looping construct:

While	**For**
i= initial value while (i <limit): statement(s) i+=step	for i in range (initial value, limit, step): statement(s)

Break Statement

Break can be used to unconditionally jump out of the loop. It terminates the execution of the loop. Break can be used in while loop and for loop. Break is mostly required, when because of some external condition, we need to exit from a loop.

Example

```
for letter in 'Python':
if letter = = 'h':
        break
        print letter
```

will result into

P

y

t

Continue Statement

This statement is used to tell Python to skip the rest of the statements of the current loop block and to move to next iteration, of the loop. Continue will return back the control to the bExampleinning of the loop. This can also be used with both while and for statement.

Example

```
for letter in 'Python':
if letter == 'h':
    continue
    print letter
```

will result into

P

y

t

o

n

EXERCISE

1) Mark True/False:

(i) While statements gets executed at least once

(ii) The break statement allows us to come out of a loop

(iii) The continue and break statement have same effect

(iv) We can nest loops

(v) We cannot write a loop that can execute forever.

(vi) Checking condition in python can be done by using the if-else statement

2) What is the difference between the following two statements:

(i)
```
if n>2:
    if n <6 :
        print 'OK'
    else:
        print 'NG'
```

(ii)
```
if n>2:
    if n<6:
        print 'OK'
    else:
        print 'NG'
```

3) Mark the correct Option(s)

(i) If there are two or more options, then we can use

a) Simple if statement b) If elif statement

c) While d) None of these

(ii) A loop that never ends is called a:

a) Continue loop b) Infinite loop

c) Circle loop d) None of these

4) Construct a logical expression to represent each of the following conditions:

(i) Score is greater than or equal to 80 but less than 90

(ii) Answer is either 'N' or 'n'

(iii) N is between 0 and 7 but not equal to 3

5) Which of the following loop will continue infinitely:

(i) a) while O: b) while 1:

c) while :1: d) while False:

(ii) We can go back to the start of the loop by using _____

a) loop b) back

c) start d) continue

6) What is the difference between selection and repetition?

7) Explain use of if statement with example.

LAB EXERCISE

1)
```
answer = raw_input("Do you like Python? ")
if answer == "yes":
print "That is great!"
else:
    print "That is disappointing!"
```

Modify the program so that it answers "That is great!" if the answer was "yes", "That is disappointing" if the answer was "no" and "That is not an answer to my question." otherwise.

2) Write a function to find whether given number is odd or even.

3) Print all multiples of 13 that are smaller than 100. Use the range function in the following manner: range (start, end, step) where "start" is the starting value of the counter, "end" is the end value and "step" is the amount by which the counter is increased each time.

4) Write a program using while loop that asks the user for a number, and prints a countdown from that number to zero. **Note:** Decide on what your program will do, if user enters a nExampleative number.

5) Using for loop, write program that prints out the decimal equivalent of ½, $^1/_3$, ¼, ----, $^1/_{10}$

6) Write a function to print the Fibonacci Series up to an Input Limit.

7) Write a function to generate and print factorial numbers up to *n* (provided by user).

8) Write a program using a for loop, that calculates exponentials. Your program should ask for base and exp. value form user. **Note:** Do not use ** operator and math module.

9) Write a program using loop that asks the user to enter an even number. If the number entered is not even then display an appropriate message and ask them to enter a number again. Do not stop until an even number is entered. Print a Congratulatory message at end.

10) Using random module, Simulate tossing a coin N times. Hint: you can use zero for head and 1 for tails.

UNIT 4

Programming with Python