Unit 2
Data, Expressions and Statements

Python interpreter and interactive mode; values and types: int, float, boolean, string, and list; variables, expressions, statements, tuple assignment, precedence of operators, comments; modules and functions, function definition and use, flow of execution, parameters and arguments; Illustrative programs: exchange the values of two variables, circulate the values of n variables, distance between two points.

1.1 Computer Languages

Computer languages are the languages through which user can communicate with the computer by writing program instructions. Computer languages can be classified into low level, middle level and high level languages.

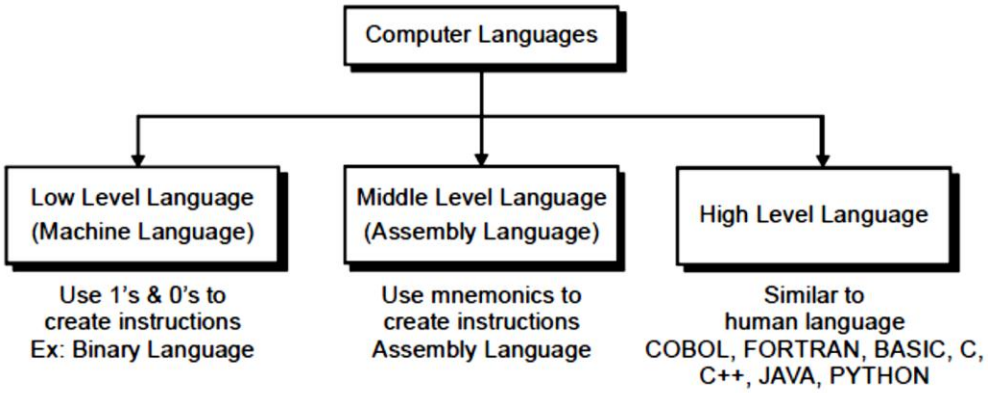


Figure 2.1 Types of Computer Languages

A high-level programming language is written in English language which normally humans can understand. To execute a program in a high level programming language, it should be converted to low level programming language. A translator is a computer program that converts a high level program into low level program.

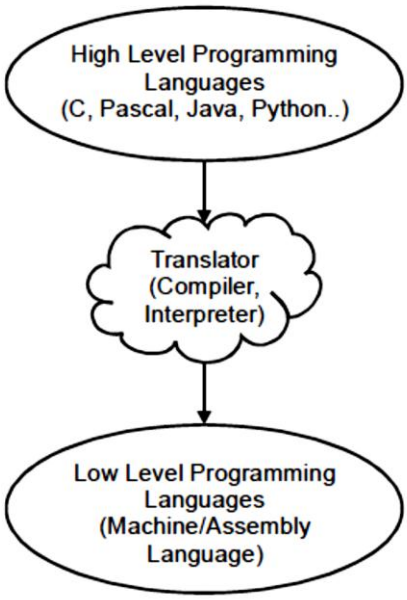


Figure 2.2 Translator

2.2 INTRODUCTION TO PYTHON

Python is an example of a high level Programming language. Python language begins its implementation in December 1989 by Guido van Rossum at Centrum Wiskunde & Informatica (CWI). Python is an interpreted, object-oriented, highlevel programming language with dynamic semantics.

The different versions of python along with the release dates are as follows:

Python 1.0 - January 1994
Python 1.5 - December 31, 1997
Python 1.6 - September 5, 2000
Python 2.0 - October 16, 2000
Python 2.1 - April 17, 2001
Python 2.2 - December 21, 2001
Python 2.3 - July 29, 2003
Python 2.4 - November 30, 2004
Python 2.5 - September 19, 2006
Python 2.6 - October 1, 2008
Python 2.7 - July 3, 2010
Python 3.0 - December 3, 2008
Python 3.1 - June 27, 2009
Python 3.2 - February 20, 2011
Python 3.3 - September 29, 2012
Python 3.4 - March 16, 2014
Python 3.5 - September 13, 2015
Python 3.6 - December 23, 2016

2.2.2 Simple Python Program

A simple program that displays "Hello, World!".

PROGRAM 1

```
1 # This program prints Hello, world!
2
3 print('Hello, world!')
```

EXECUTION

```
sh-4.3$ python program1.py
Hello, world!
```

In Python 2, the "print" statement is not a function, and therefore it is invoked without parentheses. However, in Python 3, it is a function, and must be invoked with parentheses.

2.2.3 Python Keywords

Keywords are reserved words that cannot be used as ordinary identifiers. All the keywords except True, False and None are in lowercase and they must be written as it is. The lists of all the keywords are given below.

and, del, from, not, while, as, elif, global, or, with, assert, else, if, pass, yield, break, except, import, print, class, exec, raise, in, continue, is, finally, return, def, for, lambda, try, True, False, None.

2.2.4 Python Identifiers

Identifiers are names for entities in a program, such as class, variables, functions etc.

Rules for defining Identifiers

An identifier can be composed of uppercase, lowercase letters, underscore and digits, but should start only with an alphabet or an underscore.

- Identifiers can be a combination of lowercase letters (a to z) or uppercase letters (A to Z) or digits (0 to 9) or an underscore (_).
- Identifiers cannot start with the digit.
- Keywords cannot be used as identifiers and identifiers can be any length.
- No special symbols like !, @, #, $, % etc. are used.

Valid Identifiers: sum total average _ab_ add_1 x1

Invalid Identifiers: 1x char x+y

Python is a case-sensitive language. This means, Variable and variable are not the same.

2.2.5 Python Statement

Statement is an instruction written in a high level Programming language that instructs the computer to perform a specified action. In Python, end of a statement is marked by a newline character.

```
statement = x + x**2 + x**3 + x**4 + x**5 + x**6 + x**7 + x**8
```

But statement scan be extended over multiple lines with the line continuation character (\).

```
statement = x + x**2 + x**3 \
          + x**4 + x**5 \
          + x**6 + x**7 \
          + x**8
```

We can split the above statement into multiple lines using parentheses ( ), brackets [ ] and braces { }

```
statement = (x + x**2/2 + x**3/3
          + x**4/4 + x**5/5
          + x**6/6 + x**7/7
          + x**8/8)
```

Multiple statements can be put in a single line using semicolons, as follows

```
a = 1; b = 2; c = 3
```

2.2.6 Python Indentation

Most of the programming languages like C, C++, Java use braces { } to define block of codes. Python uses indentation. Block of code starts with indentation and ends with the unintended line. Normally four whitespace characters are used for indentation and is preferred over tabs. A simple program that explains indentation.

PROGRAM 2

```
1 x=1
2 if x == 1:
3         # indented four spaces
4         print("x is 1.")
```

EXECUTION

```
sh-4.3$ python program2.py
x is 1.
```

2.2.7 Python Comments

Comments are very important while writing a program. Comment statements are used to provide a summary of the code in plain English to help future developers better understand the code.

Single Line comments

```
1 #This is a comment statement
2 print('Hello')
```

Multi-line comments

```
1 #This is a long comment
2 #and it extends
3 #to multiple lines
```

Another way of using multi-line comments is to use triple quotes, either " " "or" " ".

```
1 " " " This is also a
2 perfect example of
3 multi-line comments" " "
```

2.3 PYTHON VARIABLES

Variables are nothing but reserved memory locations to store values. Values may be numbers, text or more complicated types. Declaration of variables is not required in Python. Declaration happens automatically when a value is assigned to a variable.

```
addition = 900          # An integer assignment
amount = 1000.0         # A floating point assignment
name = "XXX"            # A string assignment
```

Multiple objects can be assigned to multiple variables as

addition, amount, name = 900,1000.0,"john"

Python allows assigning a single value to several variables.

mark1 = mark2 = mark3 = 50

Python swap values in a single line and this applies to all objects in python.

var1, var2 = var2, var1

PROGRAM 3

```
1 x = 10
2 y = 20
3 print("Before Swapping")
4 print(x)
5 print(y)
6 x,y=y,x # swap two variables
7 print("After Swapping")
8 print(x)
9 print(y)
```

EXECUTION

```
sh-4.3$ python program3.py
Before Swapping
10
20
After Swapping
20
10
```

2.4 PYTHON INPUT AND OUTPUT STATEMENTS

2.4.1 Python Output Statement

print() function is used to output data to the standard output device (screen). print statement is used where zero or more expressions are passed separated by commas. print "Python is really a great language!!!"

2.4.2 Python Input Statement

Python provides two built-in functions to read a line of text from standard input device, keyboard. These functions are:

input

- Interprets and evaluates the input ie. if user enters integer input, an integer will be returned, if user enters string input, string is returned.

raw_input

- raw_input() takes exactly what user typed and passes it back as string. It doesn't interpret the user input. Even an integer value of 10 is of string only.

PROGRAM 4

Program for simple input and output statements

```
1 # Input statement and raw_input statements
2 num1 = input("Enter the Input = ")
3 print " Simple input statement = ", num1 + 10
4 num1 = raw_input("Enter the Input = ")
5 print " Simple raw_input statement = ", num1 + 10
```

EXECUTION

```
sh-4.3$ python program4.py
Enter the Input = 6
Simple input statement = 16
Enter the Input = 6
Simple raw_input statement = print "Simple raw_input statement=",
num1 + 10
TypeError: cannot concatenate 'str' and 'int' objects
```

PROGRAM 5

```
1 # Input statement and raw_input statements
2 num1 = input("Enter the Input = ")
3 print " Simple input statement = ", num1 * 10
4 num1 = raw_input("Enter the Input = ")
5 print " Simple raw_input statement = ", num1 * 10
```

EXECUTION

```
sh-4.3$ python program4.py
```

Enter the Input = 6
Simple input statement = 60
Enter the Input = 6
Simple raw_input statement = 6666666666

2.5 PYTHON DATA TYPES

In Python, everything is represented as an object and every object has an identity, a type, and a value. Python has six standard data types:

1. Numbers
2. String
3. List
4. Tuple
5. Set
6. Dictionary

2.5.1 Numbers

Number data types store numeric values. Python has three number types: integer numbers, floating point numbers, and complex numbers. Integers can be of any length and is limited to the memory size. A number prefixed by a 0 (zero) is interpreted as an octal number. Hexadecimal numbers are prefixed either by "0x" or "0X".

PROGRAM 6

Program to explain Numbers

```
1 a = 55
2 print(a)
3 b = 010 # Octal number
4 print(b)
5 c = 0x10 # Hexadecimal number
6 print(c)
```

EXECUTION

```
sh-4.3$ python program5.py
55
8
16
```

Floating point numbers is restricted to 10 decimal places (depends on python versions). Complex numbers are of the form, x + yj, where x is the real part and y is the imaginary part. A reference to a number object is deleted by using the del statement.

The syntax of the del statement is:

del variable nam

PROGRAM 7

```
1 a = 10783724883290849032940329 40
2 print(a)
3 b = 50.0832982394832942397492374893
4 print(b)
5 c = 1+2j
6 print(c)
7
8 print(a) # print a again
9
10 del a
11 print(a)
```

EXECUTION

```
sh-4.3$ python program6.py
10783724883290849032940329 40
50.0832982395 (1+2j)
10783724883290849032940329 40
NameError: name 'a' is not defined
```

Following functions can be used on the variables of Python:

- type() function is used to check the class a variable.
- isinstance() function is used to check if an object belongs to a particular class

PROGRAM 8

Program to explain type() and isinstance() function

```
1 a = 10
2 print(a, "is of type", type(a))
3 a = 50.0
4 print(a, "is of type", type(a))
5 a = 3+4j
6 print(a, "is complex number", type(a))
7 a = 3+4j
8 print(a, "is complex number?", isinstance(3+4j,complex))
```

EXECUTION

```
sh-4.3$ python program7.py
(10, "is of type", <type 'int'>)
(50.0, "is of type", <type 'float'>)
((3+4j), "is complex number", <type 'complex'>)
((3+4j), "is complex number?", True)
```

2.5.2 Strings

String is a sequence of characters, numbers, and special characters enclosed within single-quotes(' ') or double-quote (" "). Python starts numbering at 0.

PROGRAM 9

Program to explain Strings

```
1 str = "Python Program!"
2
3 print str # Prints complete string
4 print str[0] # Indexing-Prints first character of the string
5 print str[7:10]# Slicing-Prints characters starting from 7thto 10th
6 print str[7:] # Prints string starting from
7thcharacter
7 print str * 2 # Repetition-Prints string two times
8 print str + "Ver 3.4" # concatenating - Prints concatenated string
```

EXECUTION

```
sh-4.3$ python program8.py
Python Program!
P
Pro
Program!
Python Program!Python Program!
Python Program!Ver 3.4
```

2.5.3 Lists

List is one of the most used data type in Python and is an ordered sequence of elements. All the elements in a list need not to be of the same data type. List is created by placing all the items (elements) inside a square bracket [ ], separated by commas.

PROGRAM 10

Program to explain Lists

```
1 list1 = [ ] # Empty list
2 list2 = [ 67,90,12,88] # List with integer elements
3 list3 = ['python', 6732, 45.23, 'jack', 100.2]
                    # List with mixed data types
4
5 print "List 1 =", list1 # Prints empty list
6 print "List 2 = ", list2 # Prints complete list1
7 del list2[2] # Delete 2nd element in list2
8 print "Updated List 2 =", list2
9
```

```
10 print "First element of List 3 =", list3[0]
                # Prints first element of the list3
11 print "Sliced List 3 = ", list3[1:3]
                # Prints elements starting from 2nd till 3rd
12 print "Sliced List 3 = ", list3[2:]
                # Prints elements starting from 3rd element
13 print "Repeated List 3 =", list3 * 2
                # Prints list3 two times
14 print "Concatenated List 2 & List 3 =", list2 + list3
```

EXECUTION

```
sh-4.3$ python program9.py
List 1 = []
List 2 = [67, 90, 12, 88]
Updated List 2 = [67, 90, 88]
First element of List 3 = python
Sliced List 3 = [6732, 45.23]
Sliced List 3 = [45.23, 'jack', 100.2]
Repeated List 3 = ['python', 6732, 45.23, 'jack', 100.2, 'python', 6732, 45.23, 'jack', 100.2]
Concatenated List 2 & List 3 = [67, 90, 88, 'python', 6732, 45.23, 'jack', 100.2]
```

2.5.4 Tuples

A tuple is another sequence data type similar to list. A tuple consists of a sequence of elements separated by commas. The main differences between lists and tuples are: Lists are enclosed in brackets ([ ]) and their elements and size can be changed, while tuples are enclosed in parentheses (( )) and cannot be updated.

PROGRAM 11

Program to explain Tuples

```
1 tuple2= (78,90,23,787)
2 tuple3 = ('hello', 673, 34.90, 'world')
3 list3 = ['hello', 673, 34.90, 'world']
                        # check list elements brackets[]
4 print "Tuple 2 = ", tuple2
5 print "First element of Tuple 2 =", tuple2[0]
6 print "Concatenated Tuple 2 & Tuple 3 =", tuple2 + tuple3
7 print "Tuple 3 = ", tuple3
8 print "List 3 = ", list3
9
10 list3 = [1000,89,'hi']
11 print "Changed List 3 = ", list3
12 tuple3 = (1000,89,'hi')
```

```
13 print "Changed Tuple 3 = ", tuple3
14
15 list3[2] = 5555
16 print "Changed one element of List 3 = ", list3
17 tuple3[2] = 5555
18 print "Changed one element of Tuple 3 = ", tuple3
```

EXECUTION

```
sh-4.3$ python program10.py
Tuple 2 = (78, 90, 23, 787)
First element of Tuple 2 = 78
Concatenated Tuple 2 & Tuple 3 = (78, 90, 23, 787, 'hello', 673, 34.9, 'world')
Tuple 3 = ('hello', 673, 34.9, 'world')
List 3 = ['hello', 673, 34.9, 'world']
Changed List 3 = [1000, 89, 'hi']
Changed Tuple 3 = (1000, 89, 'hi')
Changed one element of List 3 = [1000, 89, 5555]
tuple3[2] = 5555
TypeError: 'tuple' object does not support item assignment
```

PROGRAM 12

```
1 tup3 = ('hello', 23, 34.90, 'world')
2 print "Length of the Tuple = ", len(tup3)
3 print "Maximum value in the Tuple= ", max(tup3)
4 print "Minimum value in the Tuple= ", min(tup3)
5 del tuple3
```

EXECUTION

```
sh-4.3$ python program11.py
Length of the Tuple = 4
Maximum value in the Tuple= world
Minimum value in the Tuple= 23
del tuple3
NameError: name 'tuple3' is not defined
```

2.5.5 Sets

A set is an unordered collection of items. Every element is unique and cannot be changed. A set is created by placing all the items (elements) inside curly braces {}, separated by comma. It can have any number of items and they may be of different types. add() and update() functions are used to add new entries or updating existing elements in a set. Elements can be removed in a set by remove() or discard() function.

PROGRAM 13

Program to explain Sets

```
1 my_set={1,3}
2 print(my_set)
3 my_set.add(2) # add an element
4 print(my_set)
5 my_set.update([2,3,4]) # add multiple elements
6 print(my_set)
7 my_set.update([4,5],{1,6,8}) # add list and set
8 print(my_set)
9 #print my_set[0] # TypeError: 'set' object does not support indexing
10 my_set.discard(4) # discard an element in a set
11 print(my_set)
12 my_set.remove(6) # remove an element
13 print(my_set)
14 my_set.pop() # pop 1st element from a set
15 print(my_set)
```

EXECUTION

```
sh-4.3$ python program12.py
set([1, 3])
set([1, 2, 3])
set([1, 2, 3, 4])
set([1, 2, 3, 4, 5, 6, 8])
set([1, 2, 3, 5, 6, 8])
set([1, 2, 3, 5, 8])
set([2, 3, 5, 8])
```

2.5.6 Dictionary

Python dictionary is a container of unordered set of elements like lists. The elements are surrounded by curly braces { }. The elements in a dictionary are a comma-separated list of key: value pairs, where keys are usually numbers or strings and values can be any arbitrary Python data type. To access dictionary elements, use the square brackets along with the key to obtain its value.

PROGRAM 14

Program to explain Dictionary

```
1 dict = {'Name': 'John', 'SSN': 4568, 'Designation': 'Manager'}
2 print "dict['Name']: ", dict['Name']
3 print "dict['SSN']: ", dict['SSN']
4 print "dict['Designation']: ", dict['Designation']
5
6 dict['Designation'] = 'Senior Manager' # update existing entry
```

```
7 dict['Location'] = 'New Delhi' # Add new entry
8
9 print ("After Updating")
10 print "dict['Designation']: ", dict['Designation']
11 print "dict['Location']: ", dict['Location']
12 print "Length of the Dictionary =", len(dict)
```

EXECUTION

```
sh-4.3$ python program13.py
dict['Name']: John
dict['SSN']: 4568
dict['Designation']: Manager
After Updating
dict['Designation']: Senior Manager
dict['Location']: New Delhi
Length of the Dictionary = 4
```

2.6 PYTHON OPERATORS

Python language supports the following types of operators.

1. Arithmetic Operators
2. Comparison Operators
3. Logical Operators
4. Bitwise Operators
5. Assignment Operators
6. Membership Operators
7. Identity Operators

2.6.1 Arithmetic Operators

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

Operator	Name	Example	Explanation
+	Addition	x + y	Sum of x and y
–	Subtraction	x – y	Difference of x and y.
*	Multiplication	x*y	Product of x and y.
/	Division	x/y	Quotient of x divided by y
%	Modulus	x%y	Remainder of x divided by y
**	Exponentiation	x**y	x**y will give x to the power y
//	Floor Division	x//y	After division, digits after the decimal point are removed. If one of the operands is negative, the result is floored.

Table 2.1 Arithmetic Operators in Python

PROGRAM 15

Program to explain Arithmetic Operators

```
1 x = 15
2 y = 4
3 print('x + y =',x+y) # add x and y
4 print('x - y =',x-y) # subtract y from x
5 print('x * y =',x*y) # multiply x and y
6 print('x / y =',x/y) # divide x and y
7 print('x % y =',x%y) # reminder of division
8 print('x ** y =',x**y) # x to the power of y
9 print('x // y =',x//y) # division without decimal points
10
11 x = 15
12 y = -4
13 print('x // y =',x//y)
```

EXECUTION

```
sh-4.3$ python program14.py
('x + y =', 19)
('x - y =', 11)
('x * y =', 60)
('x / y =', 3.75)
('x % y =', 3)
('x ** y =', 50625)
```

('x // y =', 3)
('x // y =', -4)

2.6.2 Comparison Operators

Comparison operators are also called Relational operators. They are used to compare values and return True or False condition.

Operator	*Name*	*Example*	*Explanation*
>	Greater than	x>y	True if x is greater than y
<	Lesser than	x<y	True if x is lesser than y
==	Equal to	x==y	True if x is equal to y
!=	Not Equal to	x!=y	True if x is not equal to y
>=	Greater than or equal to	x>=y	True if x is greater than or equal to y
<=	Lesser than or equal to	x<=y	True if x is lesser than or equal to y

Table 2.2 Comparison Operators in Python

PROGRAM 16

Program to explain Comparison Operators

```
1 x = 54
2 y = 18
3 print('x > y is',x>y)
4 print('x < y is',x<y)
5 print('x == y is',x==y)
6 print('x != y is',x!=y)
7 print('x >= y is',x>=y)
8 print('x <= y is',x<=y)
```

EXECUTION

```
sh-4.3$ python program15.py
('x > y is', True)
('x < y is', False)
('x == y is', False)
('x != y is', True)
('x >= y is', True)
('x <= y is', False)
```

2.6.3 Logical Operators

Logical operators supported by python are and, or, not operators.

Operator	*Name*	*Example*	*Explanation*
and	Logical AND	x and y	True if both the operands are true
or	Logical OR	x or y	True if either of the operands is true
not	Logical NOT	x not y	True if operand is false

Table 2.3 Logical Operators in Python

PROGRAM 17

Program to explain Logical Operators

```
1 x = True
2 y = False
3 print('x and y is',x and y)
4 print('x or y is',x or y)
5 print('not x is',not x)
```

EXECUTION

```
sh-4.3$ python program16.py
('x and y is', False)
('x or y is', True)
('not x is', False)
```

2.6.4 Bitwise Operators

Bitwise operations manipulate on bits. Here, numbers are represented with bits, a series of zeros and ones.

Operator	*Name*	*Example*	*Explanation*
&	Bitwise AND	x& y	Both operands are true, then the result is true.
|	Bitwise OR	x | y	If either of the operands are true, then the result is true.
~	Bitwise NOT	~x	It complements the bits.
^	Bitwise XOR	x ^ y	Result is true, when either of the operands are true, but not both.
>>	Bitwise right shift	x>> 2	Operands are shifted right by the number of times specified.
<<	Bitwise left shift	x<< 2	Operands are shifted left by the number of times specified.

Table 2.4 Bitwise Operators in Python

PROGRAM 18

Program to explain Bitwise Operators

```
1 x = 60# In bitwise 60 = 0011 1100
2 y = 13 # In bitwise, 13 = 0000 1101
3 z = 2
4 print('x & y is',x&y) # 0011 1100 & 0000 1101 = 0000 1100 = 12
5 print('x | y is',x|y) # 0011 1100 | 0000 1101 = 0011 1101 = 61
6 print('~x is',~x) # ~ 0011 1100 = -61 in 2's complement
7 print('x ^ y is',x^y) # 0011 1100 ^ 0000 1101 = 0011 0001 = 49
8 print('x >> y is',x>>y) # 0011 1100 shifted right 2 times
9 print('x << y is',x<<y) # 0011 1100 shifted left 2 times
```

EXECUTION

```
sh-4.3$ python program17.py
('x & y is', 12)
('x | y is', 61)
('~x is', -61)
('x ^ y is', 49)
('x >> z is', 15)
('x << z is', 240)
```

2.6.5 Assignment Operators

Assignment operators are used in Python to assign values to variables.

Operator	*Name*	*Example*	*Equivalent to*
=	Equal to	x = 10	x = 10
+ =	Plus Equal to	x += 10	x = x + 10
− =	Minus Equal to	x − = 10	x = x − 10
* =	Multiplication Equal to	x *= 10	x = x * 10
/ =	Division Equal to	x/= 10	x = x / 10
% =	Modulus Equal to	x%= 10	x = x % 10
// =	Float Division Equal to	x //= 10	x = x // 10
** =	Exponent Equal to	x **= 10	x = x ** 10
& =	AND Equal to	x &= 10	x = x &10
|=	OR Equal to	x|= 10	x = x | 10
^=	XOR Equal to	x^= 10	x = x ^ 10
>>=	Right shift Equal to	x >>= 2	x = x >>2
<<=	Left shift Equal to	x >>= 2	x = x >>2

Table 2.5 Assignment Operators in Python

2.6.6 Membership Operators

Python's membership operators test are used to test whether a value or variable is found in a sequence (string, list, tuple, set and dictionary). The two membership operators are in and not in.

Operator	*Example*	*Explanation*
in	x in y	True if value/variable is found in the sequence
not in	x not in y	True if value/variable is not found in the sequence

Table 2.6 Membership Operators in Python

PROGRAM 19

Program to explain Membership Operators

```
1 x = 'Python Program'
```

```
2 y = {1:'John',2:'9090'}
3 print('y' in x)
4 print('p' in x) #case sensitive
5 print('hello' not in x)
6 print(1 in y)
7 print('a' in y)
```

EXECUTION

```
sh-4.3$ python program18.py
True
False
True
True
False
```

2.6.7 Identity Operators

Identity operators compare the memory locations of two objects. Two variables that are equal doesn't imply that they are identical. The two Identity operators are is and is not.

Operator	*Example*	*Explanation*
is	x is True	True if the operands are identical ie. they refer to the same object
is not	x is not True	True if the operands are not identical ie. they do not refer to the same object

Table 2.7 Identity Operators in Python

PROGRAM 20

Program to explain Identity Operators

```
1 x1 = 5
2 y1 = 5
3 x2 = 'Python'
4 y2 = 'python'
5 x3 = [1,2,3]
6 y3 = [1,2,3]
7 print(x1 is y1) # x1 and y1 are integer comparison
8 print(x1 is not y1)
9 print(x2 is y2) # case sensitive, x2 and y2 are string comparison
10 print(x3 is y3) # lists cannot be compared
```

EXECUTION

```
sh-4.3$ python program19.py
True
False
False
False
```

2.7 PRECEDENCE OF OPERATORS

The following table lists precedence of operators with highest precedence at top and lowest precedence at bottom. Operators in the same cell evaluate from left to right.

Operator	*Description*
**	Exponentiation
~ + -	Complement, unary plus and minus
*, /, %	Multiplication, division, remainder
+, -	Addition, subtraction
<<, >>	Bitwise shifts
&	Bitwise AND
^	Bitwise XOR
|	Bitwise OR
in, not in, is, is not, <, =, >, >=,<>, !=, ==	Comparisons, membership, identity
not x	Boolean NOT
and	Boolean AND
or	Boolean OR
lambda	Lambda expression

Table 2.8 Precedence of Operators

PROGRAM 21

Program to explain Precedence of operators

```
1 a = 20
2 b = 10
3 c = 15
```

```
4 d = 5
5 e = 0
6 e = a + b * c / d
7 print "Value of a + b * c / d is ", e
8 e = (a + b) * c / d
9 print "Value of (a + b) * c / d is ", e
10 e = a + (b * c) / d
11 print "Value of a + (b * c) / d is ", e
12 e = (a + (b * c)) / d
13 print "Value of (a + (b * c) / d is ", e
14 e = ((a + b) * c) / d
15 print "Value of ((a + b) * c) / d is ", e
16 e = a + b * (c / d);
17 print "Value of a + b * (c / d) is ", e
18 e = (a + b) * (c / d);
19 print "Value of (a + b) * (c / d) is ", e
```

EXECUTION

```
sh-4.3$ python program20.py
Value of a + b * c / d is 50
Value of (a + b) * c / d is 90
Value of a + (b * c) / d is 50
Value of (a + (b * c) / d is 34
Value of ((a + b) * c) / d is 90
Value of a + b * (c / d) is 50
Value of (a + b) * (c / d) is 90
```

2.8 PYTHON FUNCTIONS

Functions are common to all programming languages. It is defined as a block of related statements to perform a specific task. Functions help programmers to break the program into small manageable units or modules.

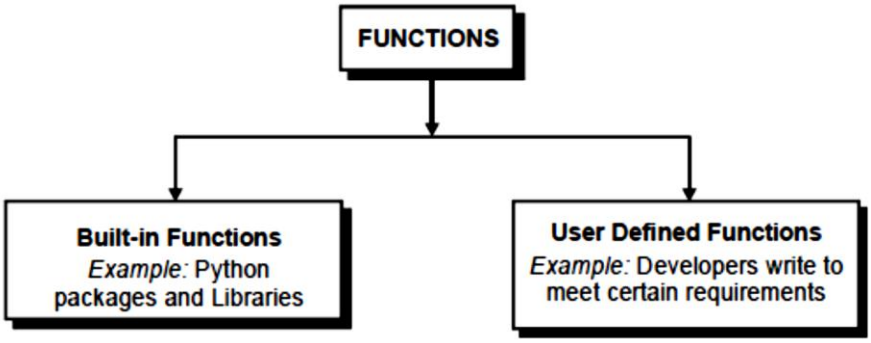


Figure 2.21 Types of Functions in Python

2.8.1 Rules to Define a Function in Python

- Every function blocks begin with the keyword def followed by the function name and parentheses ( ).
- Input parameters (arguments) through which we pass values to a function are defined inside these parentheses.
- A colon (:) is used to mark the end of function header
- Optional documentation string (docstring) to describe what the function does.
- One or more valid python statements make up the function body and all statements must have same indentation level (usually 4 spaces).
- An optional return statement to return a value from the function.

2.8.2 Syntax of Function

```
def function_name (parameters)
        """docstring"""
        statement(s)
return
```

PROGRAM 22

Simple Program to explain Function

```
1 def my_own_print( line ):
2 "This function prints a string with the function"
3 print line
4 return
5
6 my_own_print("Python program using functions")#Function call
```

EXECUTION

```
sh-4.3$ python program21.py
Python program using functions
```

2.8.3 Scope and Lifetime of Variables

Scope of a variable is the portion of a program where the variable is recognized. Variables that are defined inside a function body have a local scope, and those defined outside have a global scope.

Lifetime of a variable is the liveliness of the variable exits in the memory. The lifetime of variables inside a function live, is as long as the function executes. They are destroyed once its returned from the function.

PROGRAM 23

Simple Programs to explain Scope and Lifetime of Variables

```
1 mark1 = 86
2 mark2 = 76
```

```
3 mark3 = 89
4 total_marks = 0 #Global variable
5
6 def sum(mark1, mark2, mark3): #pass by value
7 total_marks = mark1 + mark2 + mark3
8 print "Total Marks obtained(inside function) = ", total_marks
9
10 sum(mark1, mark2, mark3) # Passing arguments in functions
11 print "Total Marks obtained = ", total_marks
```

EXECUTION

```
sh-4.3$ python program22.py
Total Marks obtained(inside function) = 251
Total Marks obtained = 0
```

PROGRAM 24

```
1 def change_content( my_data ):
2 "This function change the content in the list"
3 my_data.append([56,89,23]); # appending values in the same object
4 print "Values inside the function: ", my_data
5 return
6
7 my_data = [90,78];
8 change_content( my_data );
9 print "Values outside the function: ", my_data
```

EXECUTION

```
sh-4.3$ python program23.py
Values inside the function: [90, 78, [56, 89, 23]]
Values outside the function: [90, 78, [56, 89, 23]]
```

PROGRAM 25

```
1 def change_content(my_data):
2 "This function changes all the contents in the list"
3 my_data = [34,76,43,56];# Assign new values to the list
4 print "Values inside the function: ", my_data
5 return
6
7 my_data = [90,78];
8 change_content( my_data );
9 print "Values outside the function: ", my_data
```

EXECUTION

```
sh-4.3$ python program24.py
Values inside the function: [34, 76, 43, 56]
Values outside the function: [90, 78]
```

2.8.4 Function Arguments

Following types of formal arguments are used in Python:

1. Required arguments
2. Keyword arguments
3. Default arguments
4. Variable-length arguments or Arbitrary arguments

1. Required Arguments

In required arguments, the number of arguments passed in the function call should match exactly with the function definition.

PROGRAM 26

Program for Required arguments

```
1 def my_print( str , n1):
2 "This function prints two arguments"
3 print str,n1
4 return;
5
6 printme("Hi", 10)
7 printme(10, "Hi")
8 printme(10) #Error
```

EXECUTION

```
sh-4.3$ python program25.py
Hi 10
10 Hi
TypeError: my_print() takes exactly 2 arguments (1 given)
```

2. Keyword Arguments

In keyword arguments, the caller identifies the arguments by the parameter name. It allows skipping arguments or placing them out of order because the Python interpreter uses the keywords to match the values with parameters. Functions using keyword arguments takes the form: keyword = value

PROGRAM 27

Program for Keyword arguments

```
1 def my_details(SSN, name="John",Designation="Manager", Org="WIPRO"):
2 print(SSN,name,Designation,Organization)
```

```
3
4 my_details(1458)      # 1 positional argument
5 my_details(5656)      # 1 keyword argument
6 my_details(56868, name="julee", Org= "Google")
                                # 3 positional arguments
7 my_details('jack', Designation='HR')
                                # 1 positional, 1 keyword argument
8
9                               #Error statements
10 my_details()                 # required argument
11 my_details(56868, name="julee", "Google")
                                # non-keyword argument after a keyword argument
12 my_details(56868, SSN=89890)
                                # duplicate value for the same argument
13 my_details(city ="Bangalore") # unknown keyword argument
```

EXECUTION

```
sh-4.3$ python program26.py
(1458, 'John', 'Manager', 'WIPRO')
(5656, 'John', 'Manager', 'WIPRO')
(56868, 'julee', 'Manager', 'Google')
('jack', 'John', 'HR', 'WIPRO')
TypeError: my_details() takes at least 1 argument (0 given)
SyntaxError: non-keyword arg after keyword arg
my_details() got multiple values for keyword argument 'SSN'
TypeError: my_details() got an unexpected keyword argument 'city'
```

3. Default Arguments

Default argument assumes default value if a value is not provided in the function call for that argument.

PROGRAM 28

Program for Keyword arguments

```
1 def my_details( name, age = 55 ):
2 "This function explains about default arguments"
3 print "Name: ", name
4 print "Age ", age
5 return;
6
7 my_details( age=25, name="john" )
8 my_details( name="julee" )
```

EXECUTION

```
sh-4.3$ python program27.py
Name: john
Age 25
Name: julee
Age 55
```

4. Variable-length Arguments or Arbitrary Arguments

Sometimes, the number of arguments that will be passed into a function is not known in advance. Python allows us to handle this kind of situation through function calls with arbitrary number of arguments. Use an asterisk (*) before the parameter name to denote this kind of argument.

PROGRAM 29

Program for Arbitrary arguments

```
1 def welcome(*names):
2 "This function welcome all people"
3
4 for name in names: # names is a tuple with arguments
5 print("Hi",name)
6
7 welcome("john","jake","leo","jacky")
```

EXECUTION

```
sh-4.3$ python program28.py
('Hi', 'john')
('Hi', 'jake')
('Hi', 'leo')
('Hi', 'jacky')
```

2.8.5 The Anonymous Functions

These functions are called anonymous because they are not defined like normal functions by using the keyword def. In Python anonymous functions are defined using the keyword lambda.

Syntax:

lambda arguments: expression

1. Rules for Lambda function

- It can take any number of arguments but return just one value in the form of an expression. They cannot contain commands or multiple expressions.
- It cannot be a direct call to print because lambda requires an expression.

- Lambda functions are single line function. They are not equivalent to inline statements in C or C++, whose purpose is by passing function stack allocation during invocation for performance reasons.
- In Python, we generally use it as an argument to a higher-order function. Lambda functions are used along with built-in functions like filter(), map() etc.

PROGRAM 30

Simple program for Anonymous functions

```
1 sum = lambda arg1, arg2: arg1 + arg2;
2
3 #Now you can call sum as a function
4 print "Value of total : ", sum( 70, 20 )
5 print "Value of total : ", sum( 20, 20 )
```

EXECUTION

```
sh-4.3$ python program29.py
Value of total : 90
Value of total : 40
```

2.9 PYTHON MODULES

Module is a file containing Python definitions and statements. It defines functions, classes and variables and includes runnable code also. Functions are groups of code and modules are groups of functions.

Steps:

- Create a python file with one or more functions and save it with .py extension.
- Include the import statement.
- The import has the following syntax:
  import [module1, module2,.....module N]
    - When the interpreter comes across an import statement, it imports the module if the module is already present in the search path.
    - A search path is a list of directories that the interpreter searches before importing a module.
    - Module is loaded only once, even if multiple imports occur.
- Using the module name the function is accessed using dot (.) operation.

PROGRAM 31

Example.py

```
1 def add(a, b):
2 """This program adds two
3 numbers and return the result"""
4
```

```
5 result = a + b
6 return "Addition of two numbers", result
```

EXECUTION

```
sh-4.3$ python program30.py
sh-4.3$ python
Python 2.7.10 (default, Sep 8 2015, 17:20:17)
>>> import example
>>>example.add(89,90)
('Addition of two numbers', 179)
>>> (Ctrl+Z)
sh-4.3$
```

2.9.1 The from....import Statement

- For a module involving hundreds of functions, from.....import Statement is recommended to save loading time.

Syntax:

from python_file import function_name

- Multiple functions can be imported by separating their names with commas.

Syntax:

from python_file import_function_name1, function_name 2

- Use the asterisk symbol(*) to import everything

Syntax:

*from python_file import**

PROGRAM 32

Simple program for from...import Statement

Example.py

```
1 def add(a, b):
2 """This program adds two
3 numbers and return the result"""
4
5 result = a + b
6 return "Addition of two numbers", result
7
8 def sub(a, b):
9 """This program Subtracts two
10 numbers and return the result"""
11
12 result = a - b
13 return "subtraction of two numbers", result
```

EXECUTION

```
sh-4.3$ python program31.py
sh-4.3$ python
Python 2.7.10 (default, Sep 8 2015, 17:20:17)
>>> from example import add
>>> add(9,2)
('Addition of two numbers', 11)
>>> sub(9,2)
NameError: name 'sub' is not defined
>>> from example import add,sub
>>> add(9,2)
('Addition of two numbers', 11)
>>> sub(9,2)
('subtraction of two numbers', 7)
>>>(Ctrl+Z)
sh-4.3$
```

2.9.2 The Built-in Modules

There are plenty of built in modules, just as built in functions. The most useful ones are:

1. Random
2. Math
3. Calendar and date-time

1. Random

This module generates random numbers. If a random integer is needed, use the randint function. randint accepts two parameters: a lowest and a highest number. If a random floating point number is needed, use the random function. Choose a random element from a set such as a list, called choice.

PROGRAM 33

Program for Random function

```
1 import random
2 print random.randint(0, 5) # random integer
3 print random.random() # random floating point
4 print random.random() * 10
5 my_data = [156, 85, "John", 4.82, True]
6 print random.choice(my_data)# choose an element from the list
```

EXECUTION

```
sh-4.3$ python program32.py
1
0.698672804611
```

```
5.2069984395
4.82
sh-4.3$ python program31.py
0
0.384566916593
6.85856689662
85
```

2. Math

The math module provides access to mathematical constants and functions.

PROGRAM 34

Program for Math function

```
1 import math
2
3 print "Value of Pi = ", math.pi
4 print "Value of Eulers number = ",math.e
5 print "Value of 2 radians = ", math.degrees(2)
6 print "Value of 60 degress = ", math.radians(60)
7 print "sin(2) radians = ",math.sin(2)
8 print "cos(0.5) radians = ",math.cos(0.5)
9 print "tan(0.23) radians = ", math.tan(0.23)
10 print "Factorial of 5 = ", math.factorial(5)
11 print "Square Root of 49 = ",math.sqrt(49)
```

EXECUTION

```
sh-4.3$ python program33.py
Value of Pi = 3.14159265359
Value of Eulers number = 2.71828182846
Value of 2 radians = 114.591559026
Value of 60 degress = 1.0471975512
sin(2) radians = 0.909297426826
cos(0.5) radians = 0.87758256189
tan(0.23) radians = 0.234143362351
Factorial of 5 = 120
Square Root of 49 = 7.0
```

3. Calendar and Date-time

Python's time and calendar modules help track dates and times.

PROGRAM 35

Program for Calendar & date-time

```
1 import calendar
2 cal = calendar.month(2017, 5)
3 print "May month calendar of the year 2017:"
4 print cal
5
6 import time
7 ticks = time.time()
8 print"Number of ticks since 12:00am, January 1, 1970:", ticks
9 localtime = time.localtime(time.time())
10 print "Local current time :", localtime
11
12 #Formatted time
13 localtime = time.asctime(time.localtime(time.time()))
14 print "Local current time :", localtime
```

EXECUTION

```
sh-4.3$ python program34.py
May month calendar of the year 2017:
May 2017
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
Number of ticks since 12:00am, January 1, 1970: 1494493602.96
Local current time : time.struct_time(tm_year=2017, tm_mon=5,
tm_mday=11, tm_hour=9, tm_min=6, tm_sec=42, tm_wday=3,
tm_yday=131, tm_isdst=0)
Local current time : Thu May 11 09:06:42 2017
```

2.10 ILLUSTRATIVE PROGRAMS

2.10.1 Program to Swap Two Variables using Functions

```
1 #Swap two variables using functions
2 def swap(x,y):
3 "This function for swapping"
4 x,y=y,x # swap two variables
5 print("After Swapping")
6 print(x,y)
7
8 x = 10
```

```
9 y = 20
10 print("Before Swapping")
11 print(x, y)
12 swap(x,y)
```

EXECUTION

```
sh-4.3$ python program35.py
Before Swapping
(10, 20)
After Swapping
(20, 10)
```

2.10.2 Program to Find Fibonacci Series

```
1 Nth number of Fibonacci Series using recursive functions
2 def fib(n):
3 if n == 0:
4 return 0
5 elif n == 1:
6 return 1
7 else:
8 return fib(n-1) + fib(n-2)
9 print 'Fibonacci number in 10th place is',fib(10)
```

EXECUTION

```
sh-4.3$ python program36.py
Fibonacci number in 10th place is 55
```

2.10.3 Program to Check Whether the Given Year is Leap Year or Not

```
1 # check whether the given year is leap year or not
2
3 year = int(input("Input the year = "))
4 if year % 4 == 0 and year %100 != 0 or year % 400 == 0:
5 print ("\nGiven year is leap year")
6 else:
7 print ("\nGiven year is not leap year")
```

EXECUTION

```
sh-4.3$ python program37.py
Input the year = 2004
Given year is leap year
Input the year = 2002
Given year is not leap year
```

2.10.4 Program to Describe Usage of Global Variables

```
1 #Usage of Global Variables
2 def sample():
3 return total + 100
4
5 total = 0 # Global variable
6 print(sample())
```

EXECUTION

```
sh-4.3$ python program38.py
100
```

It is not possible to change the value of a global variable without explicitly specifying it.

```
1 def sample():
2 total = total + 100
3 print total
4 return total
5
6 total = 0
7 print(sample())
```

EXECUTION

```
sh-4.3$ python program23.py
UnboundLocalError: local variable 'total' referenced before assignment
```

2.10.5 Program to Circulate the Values of n Variables

```
1 # List Program to circulate values of n variables
2 def rotate(l, n):
3 new_list = l[n:] + l[:n]
4 return new_list
5
6 example_list = [1, 2, 3, 4, 5]
7 print "Original List = ", example_list
8 my_list = rotate(example_list, 1)
9 print "List rotated clockwise by 1 = ", my_list
10 my_list = rotate(example_list, 2)
11 print "List rotated clockwise by 2 = ", my_list
12 my_list = rotate(example_list, - 2)
13 print "List rotated anti-clockwise by 2 = ", my_list
```

EXECUTION

```
sh-4.3$ python program39.py
Original List = [1, 2, 3, 4, 5]
List rotated clockwise by 1 = [2, 3, 4, 5, 1]
List rotated clockwise by 2 = [3, 4, 5, 1, 2]
List rotated anti-clockwise by 2 = [4, 5, 1, 2, 3]
```

2.10.6 Program to Display all the Prime Numbers within an Interval

```
1 # Python program to display all the prime numbers
within an interval
2 lower = input("Enter lower range: ")
3 upper = input("Enter upper range: ")
4 print "Prime numbers between",lower,"and",upper,"are:"
5
6 for num in range(lower,upper + 1):
7 # prime numbers are greater than 1
8 if num > 1:
9 for i in range(2,num):
10 if (num % i) == 0:
11 break
12 else:
13 print(num)
```

EXECUTION

```
sh-4.3$ python program40.py
Enter lower range: 1
Enter upper range: 20
Prime numbers between 1 and 20 are:
2
3
5
7
11
13
17
19
```

2.10.7 Program to Convert Temperature in Celsius to Fahrenheit

```
1 # Python Program to convert temperature in celsius to fahrenheit
2 celsius = 37.5
3 # calculate fahrenheit
4 fahrenheit = (celsius * 1.8) + 32
```

```
5 print('%0.1f degree Celsius is equal to %0.1f degree Fahrenheit' %(celsius,fahrenheit))
```

EXECUTION

```
sh-4.3$ python program41.py
37.5 degree Celsius is equal to 99.5 degree Fahrenheit
```

2.10.8 Python Program to Check Armstrong Number

```
1 # Python program to check Armstrong number
2 # An Armstrong number of three digits integer
3 # such that the sum of the cubes of its digits is equal to the number itself.
4 # For example, 371 is an Armstrong number
5 # 3**3 + 7**3 + 1**3 = 371.
6
7 num = int(input("Enter a number: "))
8 sum = 0
9
10 # find the sum of the cube of each digit
11 temp = num
12 while temp > 0:
13 digit = temp % 10
14 sum += digit ** 3
15 temp //= 10
16
17 # display the result
18 if num == sum:
19 print(num,"is an Armstrong number")
20 else:
21 print(num,"is not an Armstrong number")
```

EXECUTION

```
sh-4.3$ python program42.py
Enter a number: 663
(663, 'is not an Armstrong number')
Enter a number: 407
(407, 'is an Armstrong number')
```

2.10.9 Python Program to find the Distance between Two Points

Distance Formula and Pythagorean Theorem

The distance formula is derived from the Pythagorean theorem. To find the distance between two points (x1, y1) and (x2, y2), the following formula is used.

$$\text{Distance} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

```
1 import math
2
3 p1 = [2 , 4] # Point p1 co-ordinates
4 p2 = [3 , 6] # Point p2 co-ordinates
5 distance = math.sqrt(((p2[0]-p1[0])**2) +((p2[1]-p1[1])**2))
6
7 print " Distance between two points ", distance
```

EXECUTION

```
sh-4.3$ python program42.py
Distance between two points 2.2360679775
```

TWO MARKS QUESTIONS WITH ANSWERS

1. What is a program?

A computer program is a collection of instructions that performs a specific task when executed by a computer.

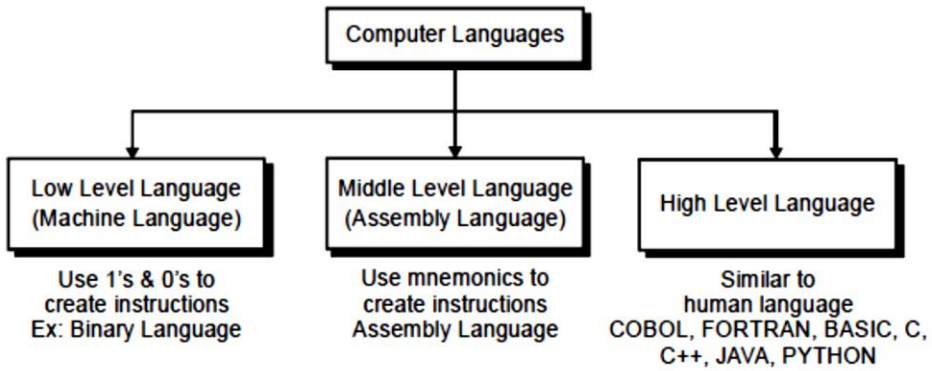
Example: Microsoft Word, Powerpoint, Excel etc.

2. Define Computer languages.

Computer languagesare the languages through which user can communicate with the computer by writing program instructions.

3. How Computer languages are classified?

Computer languages can be classified into low level, middle level and high level languages.



Types of Computer Languages

4. Define assignment statement.

The assignment statement is thata variable is always on the left hand side of an equal sign,and the right hand side it could be

- a value

- another variable
- an arithmetic expression

Syntax:

variable= expression

5. List the statements in python.

- Assignment statements
- Print statements

6. Given the strings, x = 'alpha' and y = 'beta' print the following string operations

print x + y= alphabeta

print y * 3 = betabetabeta

7. List some of the keywords in Python.

and, del, from, not, while, as, elif, global, or, with, assert, else, if, pass, yield, break, except, import, print, class, exec, raise, in, continue, is, finally, return, def, for, lambda, try, True, False, None.

8. Define identifiers in Python.

Identifiers are names for entities in a program, such as class, variables, functions etc. An identifier can be composed of uppercase, lowercase letters, underscore and digits, but should start only with an alphabet or an underscore.

i) Valid Identifiers : sum total average _ab_ add_1 x1

ii) Invalid Identifiers : 1x char x+y

9. What is the Comment statement in Python?

Comment statements are used to provide a summary of the code in plain English to help future developers better understand the code.

i) Single Line comments

#This is a commentstatement

ii) Multi-line comments

#This is a long comment
#and it extends
#to multiple lines

10. Define variables in Python.

Variables are nothing but reserved memory locations to store values. Values may be numbers, text or more complicated types. Declaration of variables is not required in Python. ***Example:*** addition = 900

11. Explain Input and Output statements in Python.

i) Input Statement:

- Python provides two built-in functions to read a line of text from standard input device, keyboard. These functions are
- input
- raw_input

ii) Output Statement

- print() function is used to output data to the standard output device (screen). print statement is used where zero or more expressions are passed separated by commas.
- print "Python is really a great language!!!"

12. List the Data types in Python.

In Python, everything is represented as an object and every object has an identity, a type, and a value. Python has five standard data types:

- Numbers
- String
- List
- Tuple
- Set
- Dictionary

13. What are the operators in Python?

Python language supports the following types of operators.

- Arithmetic Operators
- Comparison Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Membership Operators

14. Solve the mathematical expression 7 / 3 * 1.2 + 3 / 2."

7/3 *1.2+3/2
2 * 1.2 + 3 / 2
2.4 + 3 / 2
2.4 + 1 "
3.4

15. Define Functions in Python.

Functions are common to all programming languages. It is defined as a block of related statements to perform a specific task. Functions help programmers to break the program into small manageable units or modules.

Syntax of Function

def function_name (parameters)
"""docstring"""
statement(s)
return

16. Explain the precedence of operators in Python.

The following table lists precedence of operators with highest precedence at top and lowest precedence at bottom. Operators in the same cell evaluate from left to right.

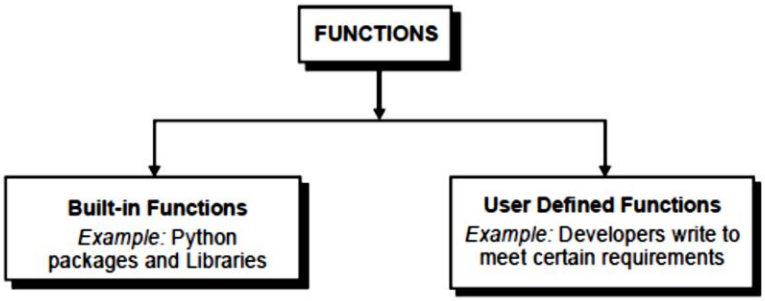
Operator	Description
**	Exponentiation
~ + -	Complement, unary plus and minus
*, /, %	Multiplication, division, remainder
+, -	Addition, subtraction
<<, >>	Bitwise shifts
&	Bitwise AND
^	Bitwise XOR
|	Bitwise OR
in, not in, is, is not, <, =, >, >=,<>, !=, ==	Comparisons, membership, identity
not x	Boolean NOT
and	Boolean AND
or	Boolean OR
lambda	Lambda expression

17. What are the Function arguments used in Python?

Following types of formal arguments are used in python:

- Required arguments
- Keyword arguments
- Default arguments
- Variable-length argument or Arbitrary argument

18. List the types of functions.



Types of Functions in Python

19. What is the Anonymous Function in Python?

Anonymous functions are the functions that are not defined like normal functions by using the keyword def. In Python anonymous functions are defined using the keyword lambda.

Syntax: lambda arguments: expression

20. Define Modules in Python

Module is a file containing Python definitions and statements. It defines functions, classes and variables and includes runnable code also. Functions are groups of code and modules are groups of functions.

21. How modules are incorporated in a Python program?

Modules are incorporated in the Python program using the from...import statement.

- For a module involving hundreds of functions, from.....import Statement is recommended to save loading time.

  Syntax: *from python_file import function_name*

- Multiple functions can be imported by separating their names with commas.

  Syntax: *from python_file import_function_name1, function_name 2*

- Use the asterisk symbol(*) to import everything

  Syntax: *from python_file import**

22. List some of the Built-in Modules in Python

There are plenty of built in modules, just as built in functions.The most useful ones are:

- Random
- Math
- Calendar and date-time

23. Write a Simple program to add two numbers in Python.

PROGRAM

```
1 # This program adds two numbers
2
3 num1 = 1.5
4 num2 = 6.3
5
6 # Add two numbers
7 sum = num1 + num2
8
9 # Display the sum
10 print 'Sum = ', sum
```

EXECUTION

```
sh-4.3$ python program.py
Sum = 7.8
```

24. Write a Simple program to convert KM/H to MPH in Python.

This program converts speed from KM/H to MPH, which may be handy for calculating speed limits when driving abroad, especially for UK and US drivers.

Formula to convert KM/H to MPHis : 1 kilometre = 0.621371192 miles

PROGRAM

```
1 kmh = int(raw_input("Enter km/h: "))
2 mph = 0.6214 * kmh
3 print "Speed:", kmh, "KM/H = ", mph, "MPH"
```

EXECUTION

```
sh-4.3$ python program.py
Enter km/h: 5
Speed: 5 KM/H = 3.107 MPH
```

25. Write a Simple program to convert decimal number into binary, octal and hexadecimal number system in Python.

PROGRAM

```
1 # Python program to convert decimal number into binary, octal and hexadecimal number system
2 dec = 344
3 print "344 in Binary =", bin(dec)
4 print "344 in Octal =", oct(dec)
5 print "344 in Hexadecimal =",hex(dec)
```

EXECUTION

```
sh-4.3$ python program.py
344 in Binary = 0b101011000
344 in Octal = 0530
344 in Hexadecimal = 0x158
```

Note: In this program, we have used built-in functions bin(), oct() and hex() to convert the given decimal number into respective number systems.

REVIEW QUESTIONS

1. Define Variables. What re the rules for defining Identifiers?
2. Explain Python Comment statement with suitable examples.
3. Explain Input statement in Python. What are the types of input?
4. Write a Python program to print the following string in a specific format:

```
Twinkle, twinkle, little star,
        How I wonder what you are!
                Up above the world so high,
                Like a diamond in the sky.
Twinkle, twinkle, little star,
        How I wonder what you are.
```

5. What are the data types in python? Explain each data type with suitable examples.
6. What are the types of operators supported by Python language? List the operators and explain them.
7. Define Function. Explain the scope and lifetime of the variables with suitable examples.
8. Explain the various function arguments in detail.
9. Explain Python modules in detail. Explain some of the built-in modules available in Python.

**** Opportunities don't happen. You create them. ****