UNIT - 2

DATA, EXPRESSIONS, STATEMENTS

2.1. INTRODUCTION TO PYTHON

Python is a high level programming language. There are some other programming languages like C, C++, Perl and Java. Still, python is a general purpose language and it is easier to read and write programs. Since the high level programs are portable, it can run on different kinds of computers with some little modifications. There are two different types of processes which are used to convert high level programming language into low level programming language. They are interpreters and compilers. They are explained below in this text.

2.2. PYTHON INTERPRETER AND INTERACTIVE MODE

Interpreter

Python is an interpreted language because they are executed by an interpreter. Interpreter take high level program as input and executes what the program says. It processes the source program without translating it into a machine language at a minimum time. It read lines and performs computations alternatively. The figure 2.1 explains the structure of an interpreter.

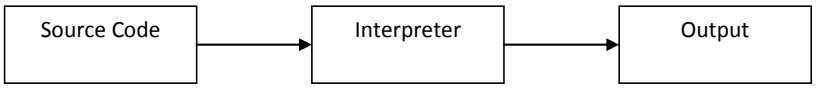

Fig. 2.1 Function of Interpreter

Compiler

A compiler reads the program and translates it completely to machine readable form called object code or executable code before the program starts running. Once a program in compiled, the program can be executed repeatedly without further translations. The figure 2.2 shows the structure of a compiler.

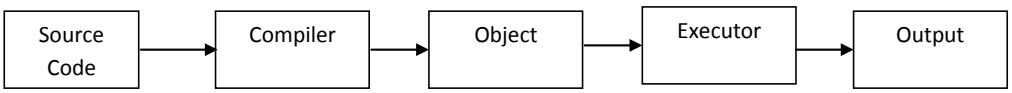

Fig. 2.2 Function of Compiler

To execute the program code, the interpreter can be used. There are two different modes to use the interpreter. 1. Interactive mode, 2. Script mode. In interactive mode, the program statements can be typed in prompt, so the interpreter displays the result.

```
>>>1+1
2
```

The prompt >>> (chevron) indicates that the interpreter is ready.

Interactive Mode

There are two modes in Python. They are Interactive mode and script mode. In Interactive mode, execution is convenient for smaller programs.

In script mode, the program can be stored in a file and the interpreter can be used to execute the file. The python scripts filename ends with .py extension.

To execute the script, in UNIX and windows type,

- python filename.py # To execute the file

For larger programs, the program code can be stored as a script and execute it in the future.

Example:

```
# first simple program

# To print Hello, World!

    print  'Hello, World!'
```

It prints Hello, World. In python printing statement uses the keyword print as in the above mentioned format.

In Python 3, the syntax for printing is slightly different like this,

```
print  ('Hello, World!')
```

The parenthesis indicates that the print is a function. The single quotation represents the beginning and end of the text to be displayed.

2.3. VALUES AND TYPES

A value is a basic thing that a program works with, like a letter or a number. Ex: 1,2 and 'Hello, World!'.

The values belong to different data types. In Python, the standard data types available are:

(1) Numbers

(2) String

(3) List

(4) Tuple

(5) Dictionary

(i) Number data type

Number data type stores numerical values. It supports four numerical types.

(1) int (signed numbers like 10, -20)

(2) long (long integers represented in octal & hexadecimal like, ox3245 and 234L)

(3) float (Floating point real values like 3.45)

(4) complex (Complex numbers like, 7.32e-3j) [ordered pairs of real floating point numbers represented by x ± jy, where x & y are real numbers and j is imaginary part.

(5) Boolean data type takes the two values: True and False.

Example:

print True # True is a Boolean value print False # False is a Boolean value.

(ii) String data type

String are the sequence of characters represented within quotation marks. It allows either pairs of single or double quotes. The substring access is possible through the slicing operator ([] or [:]). The string index 0 represents beginning of the string whereas, index -1 represents ending of the string. The following examples illustrate the string and substring accesses.

Example:

#str – string variable (explained later in this chapter) assigned with a value

str= 'Hello, World!'

Code	**Comment**	**Result**
print str	# prints complete string	Hello, World!
print str[0]	# prints first character of the string	H
print str[-1]	#prints last character of the string	!
print str[1:5]	# prints character starting from index 1 to 4 # prints str[start_at : end_at-1]	ello
print str[2:]	#prints string starting at index 2 till end of the string	llo, World!
print str * 2	# asterisk (*) -is the repetition operator. Prints the string two times.	Hello, World! Hello, World!
print str + "Hai"	# prints concatenated string	Hello, World! Hai

(iii) List data type

Lists are the most significant compound data types contain elements of various types. A List can hold items of different data types. The list is enclosed by square brackets [] where the items are separated by commas. Like string data type, the list values can be accessed using the slice operator

([] or [:]). The index 0 represents beginning of the list whereas, index -1 represents ending of the list. The following example illustrates list accesses.

Example:

list1=[‘abcd’, 345, 3.2,’python’, 3.14]

list2=[234, ‘xyz’]

Code	Comment	Result
print list1	# prints complete list	[‘abcd’, 345, 3.2,’python’, 3.14]
print list1[0]	# prints first element of the list	abcd
print list1[-1]	#prints last element of the list	3.14
print list1[1:3]	# prints elements starting from index 1 to 2 # prints list1[start_at : end_at-1]	[345, 3.2]
print list1[2:]	#prints list starting at index 2 till end of the list	[3.2, ‘python’, 3.14]
print list 2 * 2	# asterisk (*) -is the repetition operator. Prints the list two times.	[‘abcd’, 345, 3.2, ‘python’, 3.14, 234, ‘xyz’]
print list1 + list2	# prints concatenated lists	[‘abcd’, 345, 3.2,’python’, 3.14, 234, ‘xyz’]

(iv) Tuple data type

Tuple is another sequence data type similar to list. A tuple consists of a number of values separated by commas and enclosed within parentheses. Unlike list, the tuple values cannot be updated. They are treated as read-only lists. The following example explains the tuple element access.

Example:

tuple1= (‘abcd’, 345 , 3.2,’python’, 3.14)

tuple2= (234, ‘xyz’)

Code	Comment	Result
print tuple1	# prints complete tuple1	(‘abcd’, 345, 3.2, ’python’, 3.14)
print tuple1[0]	# prints first element of the tuple1	abcd
print tuple1[-1]	#prints last element of the tuple1	3.14

print tuple1[1:3]	# prints elements starting from index 1 to 2 # prints tuple1[start_at : end_at-1]	(345, 3.2)
print tuple1[2:]	#prints tuple1 starting at index 2 till the end	(3.2, ‘python’, 3.14)
print tuple 2 * 2	# asterisk (*) -is the repetition operator. Prints the tuple2 two times.	(234, 'xyz', 234, 'xyz')
print tuple1 + tuple2	# prints concatenated tuples	(‘abcd’, 345, 3.2,’python’, 3.14, 234, ‘xyz’)

(v) Dictionary data type

Dictionary data type is a kind of hash table. It contains key-value pairs. A dictionary key can be almost any python type, usually numbers or strings. Values can be arbitrary python object. Dictionaries are enclosed by curly braces {} and values can be assigned and accessed using square brackets []. The following example explains the dictionary element access.

dict1= {‘name’: ‘ABCD’ , ‘code’ : 6734 , ‘dept’ : ‘Engg’}

dict2= {}

dict2 [‘rollno’] = “II-ITA24”

Code	**Comment**	**Result**
print dict1	# prints complete dictionary	{ ‘dept’: ‘Engg’, ‘code’:6734, ‘name’ : ‘ABCD’}
print dict1.keys()	# prints all keys of dictionary	{ ‘dept’, ‘code’, ‘name’}
print dict1.values	#prints all values of dictionary	{‘Engg’, 6734, ‘ABCD’}
print dict2[‘rollno’]	# print the value for the key rollno	II-ITA24

2.4 VARIABLE

Variable is an identifier that refers to a value. While creating a variable, memory space is reserved in memory. Based on the data type of a variable, the interpreter allocates memory. The assignment statements are used to create new variables and assign values to them.

>>>msg= ‘ Hello, World!’

>>> n=12

>>> pi=3.14

The first statement creates a variable, msg and assigns ‘Hello, World!’ to it. The second statement creates the variable, n assigns 12 to it. The third statement creates the variable, pi and assigns the value 3.14.

Variable names can contain both letters and numbers. But do not start with a number. The underscore character (_) can be used in variable names . In addition to variables, python has 31 keywords.

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	

Table 2.1. Python keywords

2.5 EXPRESSIONS AND STATEMENTS

An expression is a combination of values, variables and operators. A value and a variable, itself considered as an expression.

Example:

```
17          #expression
X           #expression
X+17        #expression
```

A statement is a code that ends with a new line. So far, we have seen two kids of statements: print and assignment. Python allows the use of line continuation character (\) to denote that the line should continue.

Example:

```
Total= mark1+ \
       mark2+ \
       mark3
```

But the statements contained within [], {} or () brackets do not need to use line continuation characters. For example,

```
Item= [‘item1’, item2’, ‘item3’, ‘item4’, ‘item5’]
```

2.5.1. Assigning Values in Python

In Python both numerals and strings are considered as values. For e.g. 5, 28, 3.76 ,“Hai” are all values. They are also called as literals. Literals can be of any type such int, float or string.

For example,

5,28 are of type int.

3.76 is float

"Hai" is of type string.

2.5.2. Variable Declaration

In python, interpreter automatically detects the type by the data to which it is assigned. For assigning values "=" is used.

Examples of Variable Declaration

```
>>>X= 10                          # x is an integer
>>>Y=15.7                         # y is float
>>>Z="Welcome to Python"          # Z is a string
```

2.5.3 Multiple Assignments

Multiple assignments are allowed in a single statement in Python language.

For example,

```
>>> a, b, c  =  2, 4, 6, "Hai"
```

If all the variables possess same value, then it can be represented as follows:

```
>>>a=b=c = "5"
```

2.6 OPERATORS

An operator is a special symbol that asks the compiler to perform particular mathematical or logical computations like addition, multiplication, comparison and so on. The values the operator is applied to are called operands. For eg, in the expression 4 + 5, 4 and 5 are operands and + is an operator.

The following tokens are operators in python:

```
+     –     *     **    /     //    %
<<    >>    &     |     ^     ~
<     >     <=    >=    ==    !=    <>
```

2.6.1 Types of Operator

Python language supports the following types of operators.

- Arithmetic Operators
- Comparison (Relational) Operators
- Assignment Operators

- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators
- Unary arithmetic Operators

Arithmetic Operators

Operator	Description
+ Addition	Adds two operands.
- Subtraction	Subtracts second operand from first operand.
* Multiplication	Multiplies two operands.
/ Division	Divides first operand by second operand.
% Modulus	Divides first operand by second operand and returns the remainder
** Exponent	Performs exponential (power) calculation
// Floor Division (Integer Division)	Division of operands in which the quotient without fraction is returned as a result.

Sample Code:

```
a = 21
b = 10
c = 0
c = a + b
print “Result of addition “, c
c = a - b
print “Result of subtraction “, c
c = a * b
print “Result of multiplication “, c
c = a / b
print “Result of division “, c
c = a % b
print “Result of modulus “, c
a = 2
b = 3
c = a**b
print “Result of exponentiation “, c
```

```
a = 10
b = 5
c = a//b
print "Result of floor division ", c
```

Sample Output:

```
Result of addition 31
Result of subtraction 11
Result of multiplication 210
Result of division 2
Result of modulus 1
Result of exponentiation 8
Result of floor division 2
```

Comparison (Relational) Operators

These operators compare the values of their two operands and determine the relationship among them.

Operator	Description
==	If the values of two operands are equal, then this operator returns true.
!=	If values of two operands are not equal, then this operator returns true.
>	If the value of first operand is strictly greater than the value of second operand, then this operator returns true.
<	If the value of first operand is strictly smaller than the value of second operand, then this operator returns true.
>=	If the value of first operand is greater than or equal to the value of second operand, then this operator returns true.
<=	If the value of first operand is smaller than or equal to the value of second operand, then this operator returns true.

Sample Code and Output:

```
>>>4 == 4
 True
>>>4 != 4
False
>>>4 < 5
 True
>>>4 >= 3
```

```
True
>>>"A" < "B"
True
>>>
```

Assignment Operators

Operator	Description
=	Assigns values from right side operand to left side operand.
+=	Performs addition using two operands and assigns the result to left side operand.
-=	Subtracts right side operand from the left side operand and assigns the result to left side operand.
*=	Performs multiplication using two operands and assigns the result to left side operand.
/=	Divides left side operand by the right side operand and assigns the result to left side operand.
%=	Finds modulus using two operands and assigns the result to left side operand.
**=	Performs exponential calculation and assigns the result to the left side operand.
//=	Performs floor division and assigns the result to the left side operand

Sample Code:

```
a = 10
b = 5
c = a
print "c is ",c
c = a + b
print "c is ", c
c += a                  #  c = c + a
print "c is ", c
c -=a                   #  c = c - a
print "c is ", c
c *= a                  #  c = c * a
print "c is ", c
c /= a                  #  c = c / a
print "c is ", c
c  = 2
```

```
c %= a              #  c = c % a
print "c is ", c
c **= a             #  c = c ** a
print "c is ", c
c //= a             #  c = c // a
print "c is ", c
```

Sample Output:

```
c is  10
c is  15
c is  25
c is  15
c is  150
c is  15
c is  2
c is  1024
c is  102
```

Logical Operators

Following table shows all the logical operators supported by python:

Operator	Description
and	Logical AND returns true, if and only if both operands are true.
or	Logical OR returns true, if any of the two operands is true.
not	Logical NOT returns the logical negation of its operand.

Here, any nonzero number is interpreted as true and zero is interpreted as false. Both the ***and*** operator and the ***or*** operator expect two operands. ***not*** operator operates on a single operand.

The behaviour of each logical operator is specified in a truth table for that operator.

Sample Code:

```
a=True
b=False
print a and b
print a or b
print not a
```

Sample Output:

False
True
False

Bitwise Operators

Bitwise operators perform bit by bit operations and explained as follows:

Operator	Description
&	Performs Bitwise AND operation between two the operands.
|	Performs Bitwise OR operation between two the operands.
^	Performs Bitwise XOR (exclusive OR) operation between two the operands.
~	Performs Bitwise 1's complement on a single operand.
<<	Shifts the first operand left by the number of bits specified by the second operand (Bitwise Left Shift).
>>	Shifts the first operand right by the number of bits specified by the second operand (Bitwise Right Shift).

Sample Code:

```
a = 60          # 60 = 0011 1100
b = 26          # 13 = 0001 1010
c = a & b;      # 24 = 0001 1000
print "Result of Bitwise AND is ", c
c = a | b;      # 62 = 0011 1110
print "Result of Bitwise OR is ", c
c = a ^ b;      # 38 = 0010 0110
print "Result of Bitwise XOR is ", c
c = ~a;         # -61 = 1100 0011
print "Result of Bitwise Ones Complement is ", c
c = a << 2;     # 240 = 1111 0000
print "Result of Bitwise Left Shift is ", c
c = a >> 2;     # 15 = 0000 1111
print "Result of Bitwise Right Shift is ", c
```

Sample Output:

Result of Bitwise AND is 24
Result of Bitwise OR is 62

```
Result of Bitwise XOR is  38
Result of Bitwise Ones Complement is  -61
Result of Bitwise Left Shift is  240
Result of Bitwise Right Shift is  15
```

Membership Operators

Membership operators test for membership in a sequence, such as strings, lists, or tuples and explained below:

Operator	Description
in	Evaluates to true if it finds a variable in the specified sequence and false otherwise.
not in	Evaluates to true if it does not finds a variable in the specified sequence and false otherwise.

Sample Code:

```
a = 6
b = 2
list = [1, 2, 3, 4, 5 ];
print a in list
print a not in list
print b in list
print b not in list
```

Sample Output:

```
False
True
True
False
```

Identity Operators

Identity operators compare the memory locations of two objects. They are explained below:

Operator	Description
is	Returns true if both operands point to the same object and false otherwise.
is not	Returns false if both operands point to the same object and true otherwise.

Sample Code:

```
a = 20
b = 20
print a is b
```

```
print id(a) == id(b)
print a is not b
b=30
print a is b
print a is not b
print id(a) == id(b)
```

Sample Output:

```
True
True
False
False
True
False
```

Unary arithmetic operators

Operator	Description
+	Returns its numeric argument without any change.
-	Returns its numeric argument with its sign changed.

Sample Code:

```
a = 10
b = +a
print b
c = -a
print c
```

Sample Output:

```
10
–10
```

2.6.2 Operator precedence

When more than one operator appears in an expression, the order of evaluation depends on the rules of precedence. For mathematical operators, Python follows mathematical convention. The acronym PEMDAS (Parentheses, Exponentiation, Multiplication, Division, Addition, Subtraction) is a useful way to remember the rules.

The following table summarizes the operator precedence in Python, from the highest precedence to the lowest precedence. Operators in the same box have the same precedence and group from left to right (except for comparisons statements).

Table 2.6. Operator Precedence

Operator	Description	Associativity
(expressions...) [expressions...] {key: value...} 'expressions...'	Binding or tuple display list display dictionary display string conversion	left to right
x[index] x[index:index] x(arguments...) x.attribute	Subscription Slicing Call Attribute reference	left to right
**	Exponentiation	right-to-left
+x -x ~x	Unary plus Unary minus Bitwise NOT	left to right
* / // %	Multiplication Division Floor division Remainder	left to right
+ -	Addition Subtraction	left to right
<<, >>	Bitwise Left Shift and Right Shift	left to right
&	Bitwise AND	left to right
^	Bitwise XOR	left to right
|	Bitwise OR	left to right
in, not in is, is not <, <=, >, >=, <>, !=, ==	Membership tests Identity tests Comparisons	Chain from left to right
not	Boolean NOT	left to right
and	Boolean AND	left to right
or	Boolean OR	left to right

Examples:

4 * (6-3) is 12, and (1+2)**(6-3) is 27.

3**1+1 is 4, not 9.

2*1**4 is 2, not 16.

4*6-2 is 22, not 16.

4+2/2 is 5, not 3.

4/2*2 is 4, not 1.

2.7 COMMENTS

Comments are the non-executable statements explain what the program does. For large programs it often difficult to understand what is does. The comment can be added in the program code with the symbol #.

Example:

```
print 'Hello, World!' # print the message Hello, World!; comment
v=5     # creates the variable v and assign the value 5; comment
```

2.8. MODULES AND FUNCTIONS

2.8.1 Modules

A Module holds some definitions and statements.Python puts some statements and definitions in a file and that can be used in the interpreter. Such a file is defined as module and contains a name with a suffix .py extension. A module can also be imported using the statement "import module name". A module name is considered as a global variable value as __name__. Python also uses some built in modules.

Eg. >>>Import math

>>>Print (math.sqrt (81))

2.8.2 Function Definition and Use

A function is a group of statements that perform a specific task. If a program is large, it is difficult to understand the steps involved in it. Hence, it is subdivided into a number of smaller programs called subprograms or modules. Each subprogram specifies one or more actions to be performed for the larger program. Such subprograms are called as functions. Functions may or may not take arguments and may or may not produce results.

In Python ,there are two types of functions in python. They are :

(i) Built –in function: These are predefined functions usually a part of Python packages and libraries such as raw_input (), type (), float (), int() etc.,, are some of the built-in functions. Built-in functions are treated as reserved words (not used as variable names)

(ii) User defined function: Functions that are defined by the users are treated as user defined functions.

(i) Built-in/Pre-defined function

Built-in functions are functions already built into Python interpreter and are readily available for use.

Example:

input()	Reads a line from input, converts it to a string (stripping a trailing newline), and returns that.
print()	Print objects to the stream.

abs()	Return the absolute value of a number.
len()	Return the length (the number of items) of an object.

Program to find the ASCII value of the given character.

```
c = input("Enter a character")

print("ASCII value of " +c+ "is"+ord(c))
```

Sample Input/Output:

```
p

ASCII value of p is 112
```

ord() function convert a character to an integer (ASCII value). It returns the Unicode code point of that character.

Type conversion functions

Python provides built-in functions that convert values from one type to another.

Function	Converting what to what	Example

int()	string, floating point → integer	>>> int('2014')
		2014
		>>> int(3.141592)
		3

float()	string, integer → floating point number	>>> float('1.99')
		1.99
		>>> float(5)
		5.0

str()	integer, float, list, tuple, dictionary → string	>>> str(3.141592)
		'3.141592'
		>>> str([1,2,3,4])
		'[1, 2, 3, 4]'

list()	string, tuple, dictionary → list	>>> list('Mary') # list of characters in 'Mary'
		['M', 'a', 'r', 'y']
		>>> list((1,2,3,4)) # (1,2,3,4) is a tuple
		[1, 2, 3, 4]

tuple()	string, list → tuple	>>> tuple('Mary')
		('M', 'a', 'r', 'y')
		>>> tuple([1,2,3,4]) # [ ] for list, ( ) for tuple
		(1, 2, 3, 4)

```
>>> age = 21

>>> sign = 'You must be ' + age + 'Years old'
```

Many Python functions are sensitive to the type of data. For example, you cannot concatenate a string with an integer. If you try, it will result in following error.

```
Traceback (most recent call last):
  File "<pyshell#71>", line 1, in <module>
    sign = 'You must be ' + age + 'years old'
TypeError: cannot concatenate 'str' and 'int' objects type
```

For the example above, use the str() conversion function to convert integer to string data.

```
>>> age = 21
>>> sign = "You must be " + str(age) + "Years old"
>>>sign
```

Sample Output:

```
You must be 21 Years old
```

Examples using Built-in functions for type conversion

Program Code	Output
Converting float to int >>>print(3.14, int(3.14)) >>>print(3.9999, int(3.9999)) >>>print(3.0, int(3.0)) >>>print(-3.999, int(–3.999))	 3.14 3 3.9999 3 3.0 3 –3.999 –3
Converting string to int >>>print("2345", int("2345")) >>>print(int("23bottles"))	 2345 2345 Error : ValueError: invalid literal for int() with base 10: '23bottles'
Converting int to string >>>print(str(17))	 17
Converting float to string >>>print(str(123.45)) >>>print(type(str(123.45)))	 123.45

Converting list to tuple >>>fruits = ['apple', 'orange', 'grapes', 'pineapple'] >>>print(tuple(fruits)) >>>print(tuple('Python'))	<class 'str'> ('apple', 'orange', 'grapes', 'pineapple') ('P', 'y', 't', 'h', 'o', 'n')
Converting tuple to list >>>print(list('Python'))	['P', 'y', 't', 'h', 'o', 'n']

Math functions

Math and cmath are mathematical modules available in Python to support familiar mathematical functions. A module is a file that contains a collection of related functions. Before using built-in math functions, import math module.

```
>>>import math
```

It will create a module object named math which contains functions and variables defined in the module. Some of the familiar math functions are listed in Table.

Function	Description	Example	Output
abs(n)	Return the absolute value of a number(n)	abs(–99)	99
round(n,d)	Round a number(n) to a number of decimal points(d)	round(3.1415,2)	3.14
floor(n)	Round down to nearest integer	math.floor(4.7)	4.0
ceil(n)	Round up to nearest integer	math.ceil(4.7)	5.0
pow(n,d)	Return n raised to the power d	math.pow(10,3)	1000.0
sqrt(n)	Returns the square root of number(n)	math.sqrt(256)	16.0
fsum(iterable)	Return an accurate floating point sum of values in the iterable	sum([.1, .1, .1, .1, .1, .1, .1, .1, .1, .1]) fsum([.1, .1, .1, .1, .1, .1, .1, .1, .1, .1])	0.99999999999999 1.0
factorial(n)	Return n factorial	math.factorial(5)	120

gcd(n,m)	Return greatest common divisior of (n,m)	math.gcd(10,125)	5
trunc(x)	Return the real value *x* truncated to an integral	math.trunc(1.999)	1
sin(x), cos(x), tan(x)	Return the arc sine, cosine, tangent of *x*, in radians.	math.sin(math.pi/4) math.cos(math.pi) math.tan(math.pi/6)	0.7071067811865476 –1.0 0.5773502691896257

Mathematical Constants

- math.pi

The mathematical constant π = 3.141592..., to available precision.

- math.e

The mathematical constant e = 2.718281..., to available precision.

(ii) User defined function

The functions defined by the users according to their requirements are called user-defined functions. The users can modify the function according to their requirements.

Example:

multiply(), sum_of_numbers(), display()

2.8.3 Defining a Function

Functions in python are defined using the block keyword def followed by the function name and parentheses (( )). Function definition includes:

(1) A **header**, which begins with a keyword def and ends with a colon.

(2) A **body** consisting of one or more Python statements each indented the same amount – 4 spaces is the Python standard – from the header.

Syntax:

```
def function_name( parameters ):
  "function_docstring"
  function_suite
  return [expression]
```

The code block within every function starts with a colon (:) and is indented. The first statement of a function can be an optional statement - the documentation string of the function or docstring. The statement return [expression] exits a function, and optionally it returns the result of the function. The rules for function names are the same as for variable names: letters, numbers and some punctuation marks are legal, but the first character can't be a number. Keywords should not be used as function name. To avoid confusion, use different names for functions and variables.

2.8.4 Calling a Function

A function can be executed by calling it from another function or directly from the Python prompt by its name.

Syntax:

function_name(parameters)

Function to display Welcome message.

```
def display():
print "Welcome!!!"
>>>display()
```

Sample output:

Welcome!!!

The first line of the function definition is called the header; the rest is called the body. The header has to end with a colon and the body has to be indented. By convention, the indentation is always four spaces. In this example, the function name is *display()*.The empty parentheses after the name indicate that this function doesn't take any arguments.

Function to find the biggest of three numbers.

```
def great(no1,no2,no3):
    if (no1>no2) and (no1>no3):
            return no1
    elif (no2>no3):
            return no2
    else:
            return no3
n1=input("Enter first number")
n2=input("Enter second number")
n3= input("Enter third number")
result=great(n1,n2,n3)
print result, "is bigger"
```

Sample Input/Output:

```
Enter first number 10
Enter second number 5
Enter third number 25
25 is bigger
```

In this example, the function name is *great()*.It takes three parameters and return the greatest of three. *input()* reads input value from the user. The function is invoked by calling great(n1,n2,n3). *print()* displays the output. The strings in the print statements are enclosed in double quotes.

2.8.5 Uses of Function

```
def add(a,b):
        return (a+b)
def sub(a,b):
        return (a-b)
def calculate():
a=input("Enter first number")
b=input("Enter second number")
result=add(a,b)
print result
result=sub(a,b)
print result
return
```

This program contains three function definitions: *add()*, *sub()*, and *calculate()*. Function definitions are executed to create function objects. The statements inside the function do not get executed until the function is called, and the function definition generates no output. Function must be created before execution i.e. function definition has to be executed before the first time it is called.

2.8.6 Advantages of Functions

- Decomposing larger programs in to smaller functions makes program easy to understand, maintain and debug.
- Functions developed for one program can be reused with or without modification when needed.
- Reduces program development time and cost.
- It is easy to locate and isolate faulty function.

2.9 FLOW OF EXECUTION

Flow of execution specifies the order in which statements are executed. Program execution starts from first statement of the program. One statement is executed at a time from top to bottom. Function definitions do not alter the flow of execution of the program, and the statements inside the function are not executed until the function is called. When a function is called, the control flow jumps to the body of the function, executes the function, and return back to the place in the program

where the function call was made. Python is good at keeping track of where it is, so each time a function completes, the program picks up where it left off in the function that called it. When it gets to the end of the program, it terminates.

2.10 PARAMETERS AND ARGUMENTS

Arguments are the values provided to the function during the function call. Parameters are name used inside the function definition to refer to the value passed as an argument. Inside the function, the value of arguments passed during function call is assigned to parameters.

Function to raise a number to given power

```
import math   # This will import math module
>>>def raise(no,power):
        print math.pow(no,power)
>>>a=100
>>>b=2
>>>raise(a,b)
>>>10000
```

The function raise() assign the argument a, b to parameters no, power.

```
>>>raise(30,3)
>>>27000
```

The function raise() assign the values 30, 3 to parameters no, power.

In this example, the arguments used in function call *raise()* are *a*, *b* and their values are assigned to parameters *no*, *power*.

2.10.1 Functions with no arguments

The empty parentheses after the function name indicate that this function doesn't take any arguments.

Function to display PI value.

```
import math
>>>def show_PI():
        print math.pi
>>>show_PI()
```

Sample output:

```
3.141592653589793
```

2.10.2 Functions with arguments

Functions may also receive arguments (variables passed from the caller to the function). Arguments in function call are assigned to function parameters.

Function to calculate area of a circle.

```
import math
>>>def area_of_circle(r):
a = r * r * math.pi
print "Area of Circle:", a
>>>radius=input("Enter Radius:")
>>>area_of_circle(radius)
```

Sample output:

```
Enter Radius:6
Area of Circle:113.04
```

2.10.3 Functions with return value

Functions may return a value to the caller, using the keyword- 'return'.

Function to calculate factorial of a given number.

```
def factorial(num):
        fact=1
        i=1
if num==0:
        return 1
else:
        for i in range(1,num+1):
                fact=fact*i
        return fact
no=input("Enter a number")
print(factorial(no))
```

Sample input/output:

```
Enter a number:5
120
```

2.11 ILLUSTRATIVE PROGRAMS

2.11.1 Function to exchange the values of two variables

```
def swap(a,b):                          # function definition for swapping
        tmp=a
        a=b
        b=tmp
        print 'After Swap: n1=",a, "n2=",b
        return
n1=input("Enter first number")
n2=input("Enter second number")
print 'Before Swap: n1=",n1, "n2=",n2
swap(n1,n2)                             #  function call
```

Sample input/output:

```
Enter first number1
Enter second number11
Before Swap: n1= 1 n2= 11
After Swap: n1= 11 n2= 1
```

2.11.2 Python program to test for leap year

Note: A leap year is divisible by four, but not by one hundred, unless it is divisible by four hundred.

```
def leapyr(yr):                                  # function definition
    if yr%4==0 and yr%100!=0 or yr%400==0:       # condition check for leap year
        print 'Leap Year'
    else:
        print 'Not a Leap Year'
    return
year=input("Enter a year")
leapyr(year)                                     # function call
```

Sample input/output:

```
Enter a year1900
Not a Leap Year
```

2.11.3 Python program to test for leap year using calendar module

Python provides this functionality already in the library module 'calendar'.

```
import calendar
def leapyr(yr):
        if(calendar.isleap(yr)):        #  Using built-in function to check for leap year
                print 'Leap Year'
        else:
                print 'Not a Leap Year'
        return
year=input("Enter a year")
leapyr(year)
```

2.11.4 Python function to sum all the numbers in a list

```
def sum(numbers):
        total=0
        for x in numbers:               # Loop through number
                total+=x
        return total
print(sum((1,2,3,4,5)))
```

Sample output:

15

2.7.5 Python program to reverse a string

```
def string_reverse(str1):
        rstr1= ""
        index=len(str1)
        while index>0:
                rstr1+=str1[index-1]
                index=index-1
        return rstr1
print "Reverse String:", string_reverse("Python")
```

Sample output:

Reverse String:nohtyP

2.11.6 Python function to check whether a number is in a given range

```
def test_range(n):
    if n in range(1,10):            # Loop through range of number from 1 upto 10
        print "No. is between 1 and 10"
    else:
        print "No. is not between 1 and 10"
no=input("Enter a no.")
test_range(no)
```

Sample input/output:

```
Enter a no. 10
No. is not between 1 and 10
Enter a no. 4
No. is between 1 and 10
```

2.11.7 Python program to print the even numbers from a given list

```
def print_even(l):        # l represents list of values
for n in l:
    if n % 2 == 0:
            print '\n', n
return
print_even([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Sample output:

```
2
4
6
8
```

2.7.8 Function that circulates/rotates the list values for given number of times (order)

```
def rotate(l,order):
        for i in range(0,order):
            j=len(l)-1
            while j>0:
                tmp=l[j]
```

```
                    l[j]=l[j-1]
                    l[j-1]=tmp
                    j=j-1
                print i, 'rotation ',l
        return
    l=[1,2,3,4,5]
    rotate(l,3)
```

Sample output:

```
0 rotation: [5, 1, 2, 3, 4]
1 rotation: [4, 5, 1, 2, 3]
2 rotation: [3, 4, 5, 1, 2]
```

2.11.9 Find the distance between two points (x_c,y_c) and (x_p,y_p).

```
import math
def distance(x1, y1, x2, y2):
    dx = x2 - x1
    dy = y2 - y1
    dsquared = dx**2 + dy**2
    result = math.sqrt(dsquared)
    return result
xc=int(input("Enter xc"))
yc=int(input("Enter yc"))
xp=int(input("Enter xp"))
yp=int(input("Enter yp"))
print (distance(xc,yc,xp,yp))
```

Sample input/output:

```
Enter xc 2
Enter yc 2
Enter xp 4
Enter yp 4
2.82
```

SOME MORE EXAMPLES:

Function that takes a number as a parameter and check the number is prime or not

Note : A prime number (or a prime) is a natural number greater than 1 and that has no positive divisors other than 1 and itself.

```
def test_prime(n):              # function that returns boolean value
    if(n==1):
        return False
    elif(n==2):
        return True
    else:
        for x in range(2,n):
            if(n%x==0):
                return False
        return True
no=input("Enter a number")
if(test_prime(no)):
    print no, 'is Prime'
else:
    print no, 'is not Prime'
```

Sample output:

```
Enter a number 9
9 is not Prime
Enter a number 11
11 is Prime
```

Function to check whether a number is perfect or not

Example : The first perfect number is 6, because 1, 2, and 3 are its proper positive divisors, and 1 + 2 + 3 = 6. Equivalently, the number 6 is equal to half the sum of all its positive divisors: (1 + 2 + 3 + 6) / 2 = 6.

```
def perfect_number(n):          # function definition
   sum = 0
   for x in range(1, n):
      if n % x == 0:
         sum += x
   return sum
no=input("Enter a number")
```

```
sum=perfect_number(no)        # function call
if(sum==no):
        print 'Perfect number'
else:
        print 'Not a Perfect number'
```

Sample input/output:

```
Enter a number5
Not a Perfect number
Enter a number6
Perfect number
```

Function that checks whether a passed string is palindrome or not

```
def isPalindrome(string):
        left_pos = 0
        right_pos = len(string) - 1
        while right_pos >= left_pos:
                if string[left_pos] != string[right_pos]:
                        return False
                left_pos += 1
                right_pos -= 1
        return True
str=raw_input("Enter a string:") #use raw_input() to read verbatim string entered by user
if(isPalindrome(str)):
        print str, 'is a Palindrome'
else:
        print str, ' is not a Palindrome'
```

Sample input/output:

```
Enter a string:madam
madam is a Palindrome
```

Function that checks whether a number is palindrome or not

```
def Palindrome_Number():
   no = input("Enter a Number:")
   q=no
```

```
rev = 0
while(q!=0):                    # loop for reverse
   rev = q % 10 + rev * 10
   q = q / 10
if( rev == no):
   print('%d is a palindrome number' %no)
else:
   print('%d is not a palindrome number' %no)
```

Sample input/output:

```
Enter a Number121
121 is a palindrome number
```

TWO MARKS QUESTION & ANSWER

1. What is Python?

Python is a general-purpose, interpreted, interactive, object-oriented, and high-level programming language. It is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, and Unix shell and other scripting languages.

2. What are the features of the Python language?

- Easy-to-learn: Python has few keywords, simple structure, and a clearly defined syntax
- Easy-to-read: Python code is more clearly defined
- Easy-to-maintain: Python's source code is fairly easy-to-maintain.
- Interactive and extendable: Python allows interactive testing and debugging and allows user to add low-level modules to the Python Interpreter.
- Portable and scalable: Python can run on a wide variety of hardware platforms and support for large programs than shell scripting.

3. What are the comment lines in Python?

A comment is a programmer-readable explanation or annotation in the source code of a computer program. All characters available inside any comment are ignored by interpreter.

Single-line comments are created simply by beginning a line with the hash (#) character, and they are automatically terminated by the end of line.

Example:

#This would be a comment in Python

Multi-line comments are created by adding a triple quoted string (“””) on each end of the comment.

Example:

“””

This is a

multilline comment

“””

4. What are the advantages and disadvantages of Python?

Advantages:

- Python is easy to learn for even a novice developer
- Supports multiple systems and platforms.
- Object Oriented Programming-driven
- Allows to scale even the most complex applications with ease.
- A large number of resources are available for Python.

Disadvantages:

- Python is slow.
- Has limitations with database access.
- Python is not good for multi-processor/multi-core work.

5. Define operator.

An operator is a special symbol that asks the compiler to perform particular mathematical or logical computations like addition, multiplication, comparison and so on. The values the operator is applied to are called operands.

6. Categorize the operators of Python.

Python language supports the following types of operators.

- Arithmetic Operators
- Comparison (Relational) Operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators
- Unary arithmetic Operators

7. What is the difference between * operator and ** operator?

* is a multiplication operator which multiplies two operands.

Eg. 2*3 returns 6.

** is an exponent operator that performs exponential (power) calculation.

Eg. 2**3 returns 8.

8. What is the difference between = and == operator?

= is an assignment operator and == is a relational operator.

== operator returns true, if the values of two operands are equal.

= operator assigns values from right side operand to left side operand.

9. Give the use of identity operators in Python?

Identity operators compare the memory locations of two objects. There are two identity operators in Python:

- is operator returns true if both operands point to the same object.
- Is not operator returns false if both operands point to the same object.

10. What is the difference between unary operator and binary operator?

unary operator	binary operator
Used with single operand.	Used with two operands.
Eg: +, -, <, is.	Eg: +, -.

11. What is the need of operator precedence?

When more than one operator appears in an expression, the order of evaluation is determined by the rules of operator precedence.

12. What is the output of the following Python program?

```
x = 6
y = 4
print(x/y)
```

Output: 1.5

13. Define keyword. List few Python keywords.

Keywords are certain reserved words that have standard and pre-defined meaning in Python. We cannot use a keyword as variable name, function name or any other identifier.

Example. False, class, finally, nonlocal, yield, lambda, assert.

14. What is a variable?

Variables are reserved memory locations to store values. A variable is an identifier for memory location.

Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables.

Example:

```
counter = 100          # An integer assignment
miles   = 1000.0       # A floating point
name    = "John"       # A string
```

15. What ate the data types available in Python?

Python has five standard data types :

- Numbers (int, long, float, complex)
- String
- List
- Tuple
- Dictionary

16. Write any 6 escape sequences in Python.

\n – move to next line

\t – move one space horizontally

\v- move in space vertically

\f – move to next page

\r – move to starting of that line

\b – backspace

17. What is type casting?

Type casting is the process of converting the value of an expression to a particular data type. This is, in Python, done with functions such as int() or float() or str().

Example:

```
x = '100'
y = '-90'
print x + y
```

Output: 100-90

```
print int(x) + int(y)
```

Output: 10

18. Write in short about relational operators.

Relational operators are used to compare any two values. An expression which uses a relational operator is called a relational expression. The value of a relational expression is either true or false.

Example: = =, !=,>, <

```
3>4
```

Output: False

19. What are the Bitwise operators available in Python?

& - Bitwise AND

| - Bitwise OR

~ - One's Complement

>> - Right shift

<< - Left shift

^ - Bitwise XOR are called bit field operators

Example: k=~j; where ~ take one's complement of j and the result is stored in k.

20. Define function.

A function is a self-contained program, or a sub-program of one or more statements which is used to do some particular task.

Types of functions: i)User defined, ii)pre-defined

21. What are pre-defined functions? Give example.

Pre-defined functions or Built-in functions are functions already built into Python interpreter and are readily available for use.

Example: print(), abs(), len()

22. What is RETURN statement?

It is used to return the control from calling function to the next statement in the program. It can also return some values.

Example: return, return 0, return (a+b)