5 GETTING STARTED WITH PYTHON (NCERT CLASS 11)

INTRODUCTION

An ordered set of instructions to be executed by acomputer to carry out a specific task is called a program, and the language used to specify this set of instructions to the computer is called a programming language.

Computers understand the language of 0s and 1s which is called machine language or low level language. However, it is difficult for humans to write or comprehend instructions using 0s and 1s. This led to the advent of high-level programming languages like Python, C++, Visual Basic, PHP, Java that are easier to manage by humans but are not directly understood by the computer.

A program written in a high-level language is called source code. Language translators like compilers and interpreters are needed to translate the source code into machine language. Python uses an interpreter to convert its instructions into machine language, so that it can be understood by the computer. An interpreter processes the program statements one by one, first translating and then executing. This process is continued until an error is encountered or the whole program is executed successfully. In both the cases, program execution will stop. On the contrary, a compiler translates theentire source code, as a whole, into the object code. After scanning the whole program, it generates error messages, if any.

- **Features of Python**

(I) Python is a high level language. It is a free and open source language.

(II) It is an interpreted language, as Python programs are executed by an interpreter.

(III) Python programs are easy to understand as they have a clearly defined syntax and relatively simple structure.

(IV) Python is case-sensitive. For example, NUMBER and number are not same in Python.

(V) Python is portable and platform independent, means it can run on various operating systems and hardware platforms.

(VI) Python has a rich library of predefined functions.

(VII) Python is also helpful in web development. Many popular web services and applications are built using Python.

(VIII) Python uses indentation for blocks and nested blocks.

- **Working with Python :** To write and run (execute) a Python program, we needto have a Python interpreter installed on our computer or we can use any online Python interpreter. The interpreter is also called Python shell.

The symbol >>> is the Python prompt, which indicates that the interpreter is ready to take instructions. We can type commands or statements on this prompt to execute them using a Python interpreter.

- **Execution Modes:** There are two ways to use the Python interpreter:

(I) **Interactive mode:** To work in the interactive mode, we can simply type a Python statement on the >>> prompt directly. As soon as we press enter, the interpreter executes the statement and displays the result(s). Interactive mode allows execution of individual statement instantaneously. Working in the interactive mode is convenient for testing a single line code for instant execution. But in the interactive mode, we cannot save the statements for future use and we have to retype the statements to run them again.

(II) **Script mode:** we can write a Python program in a file, save it and then use the interpreter to execute it. Python scripts are saved as files where file name has extension ".py". By default, the Python scripts are saved in the Python installation folder. To execute a script, we can either:

(I) Type the file name along with the path at the prompt.

(II) While working in the script mode, after saving the file, click [Run]->[Run Module] from the menu,

(III) The output appears on shell.

PYTHON KEYWORDS

Keywords are reserved words. Each keyword has a specific meaning to the Python interpreter, and we can use a keyword in our program only for the purpose for which it has been defined.

False	Class	finally	Is	Return
None	continue	for	lamda	try
True	Def	from	nonlocal	while
and	Del	global	not	with
As	Elif	if	or	yield
Assert	Else	import	Pass	
Break	except	in	raise	

IDENTIFIERS

In programming languages, identifiers are names used to identify a variable, function, or other entities in a program. The rules for naming an identifier in Python are as follows:

(I) The name should begin with an uppercase or a lowercase alphabet or an underscore sign (_). This may be followed by any combination of characters a–z, A–Z, 0–9 or underscore (_). Thus, an identifier cannot start with a digit.

(II) It can be of any length. (However, it is preferred to keep it short and meaningful).

(III) It should not be a keyword or reserved word given in Table above.

(IV) We cannot use special symbols like !, @, #, $, %,etc., in identifiers.

For example, to find the average of marks obtained by a student in three subjects, we can choose the identifiers as marks1, marks2, marks3 and avg rather than a, b, c, or A, B, C.

avg = (marks1 + marks2 + marks3)/3

VARIABLES

A variable in a program is uniquely identified by a name(identifier). Variable in Python refers to an object — an item or element that is stored in the memory. Value of a variable can be a string (e.g., 'b', 'Global Citizen'), numeric (e.g., 345) or any combination of alphanumeric characters (CD67). In Python we can use an assignment statement to create new variables and assign specific values to them.

Variable declaration is implicit in Python, means variables are automatically declared and defined when they are assigned a value the first time. Variables must always be assigned values before they are used in expressions as otherwise it will lead to an error in the program. Wherever a variable name occurs in an expression, the interpreter replaces it with the value of that particular variable.

COMMENTS

(I) Comments are used to add a remark or a note in the source code. Comments are not executed by interpreter.

(II) Comments are added with the purpose of making the source code easier for humans to understand.

(III) Comments are used primarily to document the meaning and purpose of source code and its input and output requirements, so that we can remember later how it functions and how to use it.

(IV) For large and complex software, comments may require programmers to work in teams and sometimes, a program written by one programmer is required to be used or maintained by another programmer. In such situations, documentations in the form of comments are needed to understand the working of the program.

(V) In Python, a comment starts with # (hash sign).Everything following the # till the end of that line is treated as a comment and the interpreter simply ignores it while executing the statement.

EVERYTHING IS AN OBJECT

Python treats every value or data item whether numeric, string, or other type as an object in the sense that it can be assigned to some variable or can be passed to a function as an argument.

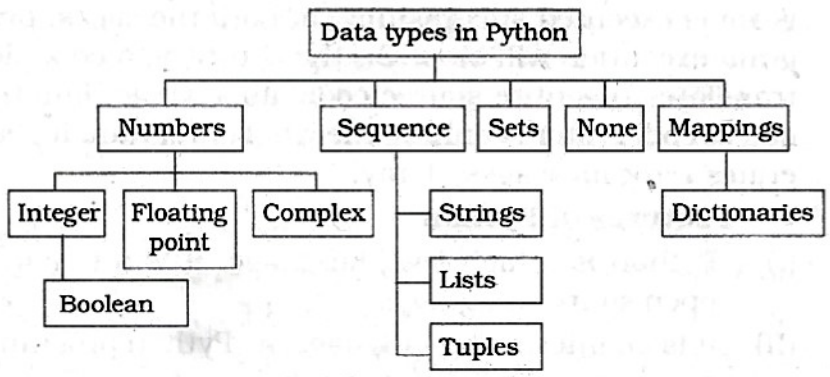
Every object in Python is assigned a unique identity(ID) which remains the same for the lifetime of that object. This ID is akin to the memory address of the object. The function id() returns the identity of an object.

NOTE: In the context of Object Oriented Programming (OOP), objects are a representation of the real world, such as employee, student, vehicle, box, book, etc. In any object oriented programming language like C++, JAVA, etc., each object has two things associated with it: (i) data or attributes and (ii) behaviour or methods. Further there are concepts of class and class hierarchies from which objects can be instantiated.

Python also comes under the category of object oriented programming. However, in Python, the definition of object is loosely casted as some objects may not have attributes or others may not have methods.

DATA TYPES

Every value belongs to a specific data type in Python. Data type identifies the type of data values a variable can hold and the operations that can be performed on that data. Following Figure enlists the data types available in Python.



- **NUMBER:** Number data type stores numerical values only. It isclassified into three different types: int, float and complex.

Type/ Class	Description	Examples
Int	integer numbers	-11, -2, 0, 122, 3
Float	real or floating point numbers	-1.03, 3.0, 15.22
Complex	complex numbers	3 + 4j, 2 – 2j

Boolean data type (bool) is a subtype of integer. Itis a unique data type, consisting of two constants, True and False. Boolean True value is non-zero, non-null and non-empty. Boolean False is the value zero.

Variables of simple data types like integers, float,boolean, etc., hold single values. But such variables are not useful to hold a long list of information, for example, names of the months in a year, names of students in a class, names and numbers in a phone

book or the list of artefacts in a museum. For this, Python provides data types like tuples, lists, dictionaries and sets.

- **Sequence :** A Python sequence is an ordered collection of items,where each item is indexed by an integer. The three types of sequence data types available in Python are Strings, Lists and Tuples.

(I) **String:** It is a group of characters. These characters may be alphabets, digits or special characters including spaces. String values are enclosed either in single quotation marks (e.g., 'Hello') or in double quotation marks (e.g., "Hello"). The quotes are not a part of the string, they are used to mark the beginning and end of the string for the interpreter.

(II) **List:** List is a sequence of items separated by commas and the items are enclosed in square brackets [ ].

(III) **Tuple:** Tuple is a sequence of items separated by commas and items are enclosed in parenthesis ( ). This is unlike list, where values are enclosed in brackets [ ]. Once created, we cannot change the tuple.

- **Set:** Set is an unordered collection of items separated by commas and the items are enclosed in curly brackets { }. A set is similar to list, except that it cannot have duplicate entries. Once created, elements of a set cannot be changed.
- **None:** None is a special data type with a single value. It is used to signify the absence of value in a situation. None supports no special operations, and it is neither same as False nor 0 (zero).
- **Mapping:** Mapping is an unordered data type in Python. Currently, there is only one standard mapping data type in Python called dictionary.

Dictionary in Python holds data items in key-value pairs. Items in a dictionary are enclosed in curly brackets { }. Dictionaries permit faster access to data. Every key is separated from its value using a colon (:) sign. The key: value pairs of a dictionary can be accessed using the key. The keys are usually strings and their values can be any data type. In order to access any value in the dictionary, we have to specify its key in square brackets [ ].

- **Mutable and Immutable Data Types:** Variables whose values can be changed after they are created and assigned are called mutable. Variables whose values cannot be changed after they are created and assigned are called immutable. When an attempt is made to update the value of an immutable variable, the old variable is destroyed and a new variable is created by the same name in memory.
- **Deciding Usage of Python Data Types:**

(I) It is preferred to use lists when we need a simple iterable collection of data that may go for frequent modifications.

(II) Tuples are used when we do not need any change in the data.

(III) When we need uniqueness of elements and to avoid duplicacy it is preferable to use sets.

(IV) If our data is being constantly modified or we need a fast lookup based on a custom key or we need a logical association between the key : value pair, it is advised to use dictionaries.

OPERATORS

An operator is used to perform specific mathematical or logical operation on values. The values that the operators work on are called operands. For example, in the expression 10 + num, the value 10, and the variable num are operands and the + (plus) sign is an operator.

(I) **Arithmetic Operators :** Python supports arithmetic operators that are used to perform the four basic arithmetic operations as well as modular division, floor division and exponentiation.

Operator	Operation	Description	Example

+	Addition	Adds the two numeric values on either side of the operator.
		
		This operator can also be used to concatenate two strings on either side of the operator.
			>>> num1 = 5
			>>> num2 = 6
			>>> num1 + num2
			11
			>>> str1 = "Hello"
			>>> str2 = "India"
			>>> str1 + str2
			'HelloIndia

-	Subtraction	Subtracts the operand on the right from the operand on the left	>>> num1 = 5
			>>> num2 = 6
			>>> num1 - num2
			-1

*	Multiplication	Multiplies the two values on both side of the operator
		
		Repeats the item on left of the operator if first operand is a string and second operand is an integer value
			>>> num1 = 5
			>>> num2 = 6
			>>> num1 * num2
			30
			>>> str1 = 'India'
			>>> str1 * 2
			'IndiaIndia'

/	Division	Divides the operand on the left by the operand on the right and returns the quotient	>>> num1 = 8
			>>> num2 = 4
			>>> num2 / num1
			0.5

%	Modulus	Divides the operand on the left by the operand on the right and returns the remainder.	>>> num1 = 13
			>>> num2 = 5
			>>> num1 % num2
			3

//	Floor Division	Divides the operand on the left by the operand on the right and returns the quotient by removing the decimal part. It is sometimes also called integer division.	>>> num1 = 13
			>>> num2 = 4
			>>> num1 // num2
			3
			>>> num2 // num1
			0

**	Exponent	Performs exponential (power) calculation on operands. That is, raise the operand on the left to the power of the operand on the right	>>> num1 = 3
			>>> num2 = 4
			>>> num1 ** num2
			81

(II) **Relational Operators :** Relational operator compares the values of the operands on its either side and determines the relationship among them.

Operator	Operation	Description	Example
==	Equals to	If the values of two operands are equal, then the condition is True, otherwise it is False	>>> num1 == num2 False >> str1 == str2 False

!=	Not equal to	If values of two operands are not equal, then condition is True, otherwise it is False	>>> num1 != num2
			True
			>>> str1 != str2
			True
			>>> num1 != num3
			False

>	Greater than	If the value of the left-side operand is greater than the value of the rightside operand, then condition is True, otherwise it is False	>>> num1 > num2
			True
			>>> str1 > str2
			True

<	Less than	If the value of the left-side operand is less than the value of the rightside operand, then condition is True,otherwise it is False	>>> num1 < num3
			False
			>>> str2 < str1
			True

>=	Greater than or equal to	If the value of the left-side operand is greater than or equal to the value of the right-side operand, then condition is True, otherwise it is False	>>> num1 >= num2
			True
			>>> num2 >= num3
			False
			>>> str1 >= str2
			True

<=	Less than or equal to	If the value of the left operand is less than or equal to the value of the right operand, then is True otherwise it is False	>>> num1 <= num2
			False
			>>> num2 <= num3
			True
			>>> str1 <= str2
			False

(III) **Assignment Operators:** Assignment operator assigns or changes the value of the variable on its left.

Operator	Description	Example

=	Assigns value from right-side operand to left side operand	>>> num1 = 2
		>>> num2 = num1
		>>> num2
		2
		>>> country = 'India'
		>>> country
		'India'

+=	It adds the value of right-side operand to the left-side operand and assigns the result to the left-side operand
	Note: x += y is same as x = x + y
		>>> num1 = 10
		>>> num2 = 2
		>>> num1 += num2
		>>> num1
		12
		>>> num2
		2
		>>> str1 = 'Hello'
		>>> str2 = 'India'
		>>> str1 += str2
		>>> str1
		'HelloIndia'

-=	It subtracts the value of right-side operand from the left-side operand and assigns the result to left-side operand
	Note: x -= y is same as x = x - y
		>>> num1 = 10
		>>> num2 = 2
		>>> num1 -= num2
		>>> num1
		8

*=	It multiplies the value of right-side operand with the value of left-side operand and assigns the result to left-side operand
	Note: x *= y is same as x = x * y
		>>> num1 = 2
		>>> num2 = 3
		>>> num1 *= 3
		>>> num1
		6
		>>> a = 'India'
		>>> a *= 3
		>>> a
		'IndiaIndiaIndia'

/=	It divides the value of left-side operand by the value of right-side operand and assigns the result to left-side operand
	Note: x /= y is same as x = x / y
		>>> num1 = 6
		>>> num2 = 3
		>>> num1 /= num2
		>>> num1
		2.0

%=	It performs modulus operation using two operands and assigns the result to left-side operand
	Note: x %= y is same as x = x % y
		>>> num1 = 7
		>>> num2 = 3
		>>> num1 %= num2
		>>> num1

//=	It performs floor division using two operands and assigns the result to left-side operand
	Note: x //= y is same as x = x // y
		>>> num1 = 7
		>>> num2 = 3
		>>> num1 //= num2
		>>> num1
		2

**=	It performs exponential (power) calculation on operators and assigns value to the left-side operand
	Note: x **= y is same as x = x ** y
		>>> num1 = 2
		>>> num2 = 3
		>>> num1 **= num2
		>>> num1
		8

(IV) **Logical operator:** There are three logical operators supported by Python.These operators (and, or, not) are to be written in lower case only. The logical operator evaluates to either True or False based on the logical operands on either side. Every value is logically either True or False. By default, all values are True except None, False, 0 (zero), empty collections "", (), [], {}, and few other special values. So if we say num1 = 10, num2 = -20, then both num1 and num2 are logically True.

(V) **Identity Operators :** Identity operators are used to determine whether the value of a variable is of a certain type or not. Identity operators can also be used to determine whether two variables are referring to the same object or not. There are two identity operators.

Operator	Description	Example

is	Evaluates True if the variables on either side of the operator point towards the same memory location and False otherwise. var1 is var2 results to True if id(var1) is equal to id(var2)	>>> num1 = 5
		>>> type(num1) is int
		True
		>>> num2 = num1
		>>> id(num1)
		1433920576
		>>> id(num2)
		1433920576
		>>> num1 is num2
		True

is not	Evaluates to False if the variables on either side of the operator point to the same memory location and True otherwise. var1 is not var2 results to True if id(var1) is not equal to id(var2)	>>> num1 is not num2 False

(VI) **Membership operator:** Membership operators are used to check if a value is a member of the given sequence or not.

Operator	Description	Example

In	Returns True if the variable/value is found in the specified sequence and False otherwise	>>> a = [1,2,3]
		>>> 2 in a
		True
		>>> '1' in a
		False

not in	Returns True if the variable/value is not found in the specified sequence and False otherwise	>>> a = [1,2,3]
		>>> 10 not in a
		True
		>>> 1 not in a
		False

EXPRESSIONS

(I) An expression is defined as a combination of constants, variables, and operators.

(II) An expression always evaluates to a value.

(III) A value or a standalone variable is also considered as an expression but a standalone operator is not an expression.

- **Precedence of Operators :** When an expression contains different kinds of operators, precedence determines which operator should be applied first. Higher precedence operator is evaluated before the lower precedence operator. Most of the operators studied till now are binary operators. Binary operators are operators with two operands. The unary operators need only one operand, and they have a higher precedence than the binary operators. The minus (-) as well as + (plus) operators can act as both unary and binary operators, but not is a unary logical operator.

```
#Depth is using - (minus) as unary operator
Value = -Depth
#not is a unary operator, negates True
print(not(True))
```

- **Precedence of all operators in Python :**

Order of Precedence	Operators	Description
1.	**	Exponentiation (raise to the power)
2.	~ ,+, -	Complement, unary plus and unary minus
3.	* ,/, %, //	Multiply, divide, modulo and floor division
4.	+, -	Addition and subtraction
5.	<= , < , > , >=, == , !=	Relational and Comparison operators
6.	=, %=, /=, //=, -=, +=, *=, **=	Assignment operators
7.	is, is not	Identity operators
8.	in, not in	Membership operators
9.	not	Logical operators
10.	And	Logical operators
11.	or	Logical operators

Note:

(I) Parenthesis can be used to override the precedence of operators. The expression within () is evaluated first.

(II) For operators with equal precedence, the expression is evaluated from left to right.

STATEMENT

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

INPUT AND OUTPUT

In Python, wehave the input() function for taking the user input. The input() function prompts the user to enter data. It accepts all user input as string. The user may enter a number or a string but the input() function treats them as strings only. The syntax for input() is:

input ([Prompt])

Prompt is the string we may like to display on the screen prior to taking the input, and it is optional. When a prompt is specified, first it is displayed on the screen after which the user can enter data. The input() takes exactly what is typed from the keyboard, converts it into a string and assigns it to the variable on left-hand side of the assignment operator (=). Entering data for the input function is terminated by pressing the enter key.

Python uses the print() function to output data tostandard output device — the screen.

The syntax for print() is:

```
print(value [, ..., sep = ' ', end = '\n'])
```

- sep: The optional parameter sep is a separator between the output values. We can use a character, integer or a string as a separator. The default separator is space.
- end: This is also optional and it allows us to specify any string to be appended after the last value. The default is a new line.

TYPE CONVERSION

(I) **Explicit Conversion :** Explicit conversion, also called type casting happens when data type conversion takes place because the programmer forced it in the program. The general form of an explicit data type conversion is: (new_data_type) (expression)

Following are some of the functions in Python that are used for explicitly converting an expression or a variable to a different type.

Function	Description
int(x)	Converts x to an integer
float(x)	Converts x to a floating-point number
str(x)	Converts x to a string representation
chr(x)	Converts ASCII value of x to character
ord(x)	returns the character associated with the ASCII code x

The interpreter cannot convert an integer value to string implicitly. It may appear quite intuitive that the program should convert the integer value to a string depending upon the usage. However, the interpreter may not decide on its own when to convert as there is a risk of loss of information. Python provides the mechanism of the explicit type conversion so that one can clearly state the desired outcome.

Type casting is needed to convert float to string. In Python, one can convert string to integer or float values whenever required.

(II) **Implicit Conversion :**Implicit conversion, also known as coercion, happens when data type conversion is done automatically by Python and is not instructed by the programmer.

The float value not converted to an integer due to type promotion that allows performing operations (whenever possible) by converting data into a wider-sized data type without any loss of information.

DEBUGGING

The process of identifying and removing mistakes also known as bugs or errors, from a program which a programer make during writing a program is called debugging. Errors occurring in programs can be categorised as:

(i) Syntax errors

(ii) Logical errors

(iii) Runtime errors

- **Syntax errors :** Like other programming languages, Python has its own rules that determine its syntax. The interpreter interprets the statements only if it is syntactically (as per the rules of Python) correct. If any syntax error is present, the interpreter shows error message(s) and stops the execution there.
- **Logical errors :** A logical error is a bug in the program that causes it to behave incorrectly. A logical error produces an undesired output but without abrupt termination of the execution of the program.
- **Runtime errors :** A runtime error causes abnormal termination of program while it is executing. Runtime error is when the statement is correct syntactically, but the interpreter cannot execute it. Runtime errors do not appear until after the program starts running or executing.

□□□

6 FLOW OF CONTROL (NCERT CLASS 11)

INTRODUCTION

The order of execution of the statements in a program is known as flow of control. The flow of control can be implemented using control structures.

CONTROL STATEMENTS

Flow control statements are used to control the flow of execution depending upon the specified condition/logic.

Sequential control statement - Sequential execution is when statements are executed one after another in order. We don't need to do anything more for this to happen as python compiler itself do it.

There are three types of control statements:-

(I) Decision Making Statements/If control statement.

(II) Iteration Statements (Loop control statement).

(III) Jump Statements (break, continue, pass).

Decision Making Statement :

Decision making statement used to control the flow of execution of program depending upon condition.

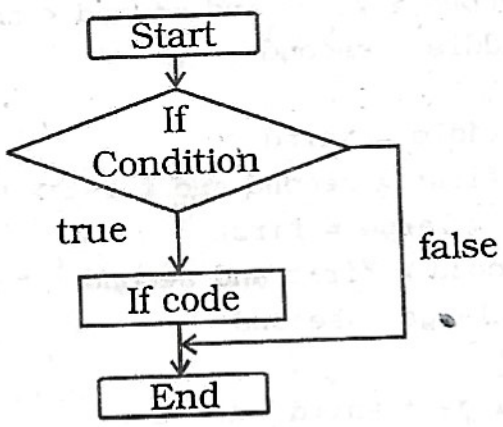
There are three types of decision making statemen:-

(I) if statements.

(II) if-else statements.

(III) Nested if-else statement.

- **if statements :** An if statement is a programming conditional statement that, if proved true, performs a function or displays information.



if statements

Syntax:

```
if(condition):
statement
[statements]
e.g.
noofbooks = 2
if (noofbooks == 2):
print('You have ')
print('two books')
print('outside of if statement')
Output
You have two books.
```

Note : To indicate a block of code in Python, you must indent each line of the block by the same amount. In above e.g. both print statements are part of if condition because of both are at same level indented but not the third print statement.

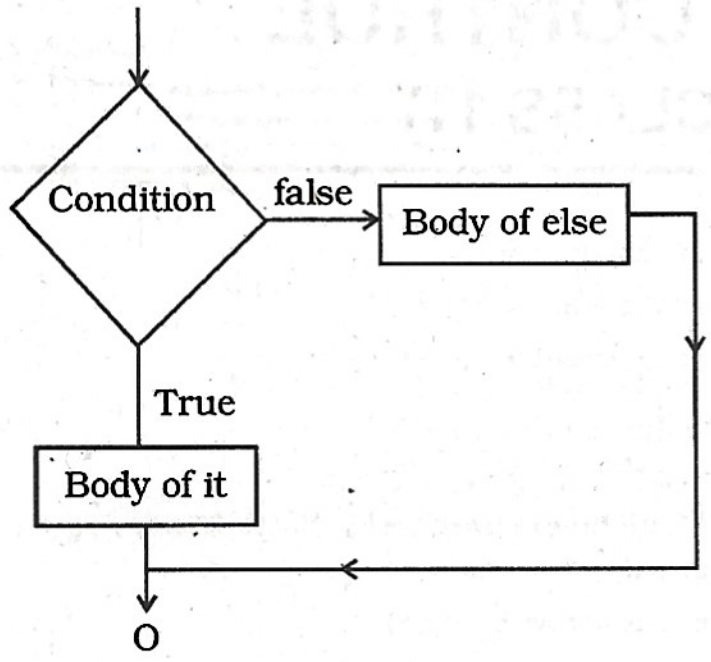
- **if-else statements :**

```
#find absolute value
a=int(input("enter a number"))
if(a<0):
a=a*-1
print(a)
#it will always return value in positive.
```

- **if statements :**

```
Using logical operator in if statement
x=1
 y=2
 if(x==1 and y==2):
print('condition matcing the criteria')
Output :-
condition matcing the criteria
.....
a=100
if not(a == 20):
print('a is not equal to 20')
Output :-
a is not equal to 20.
```

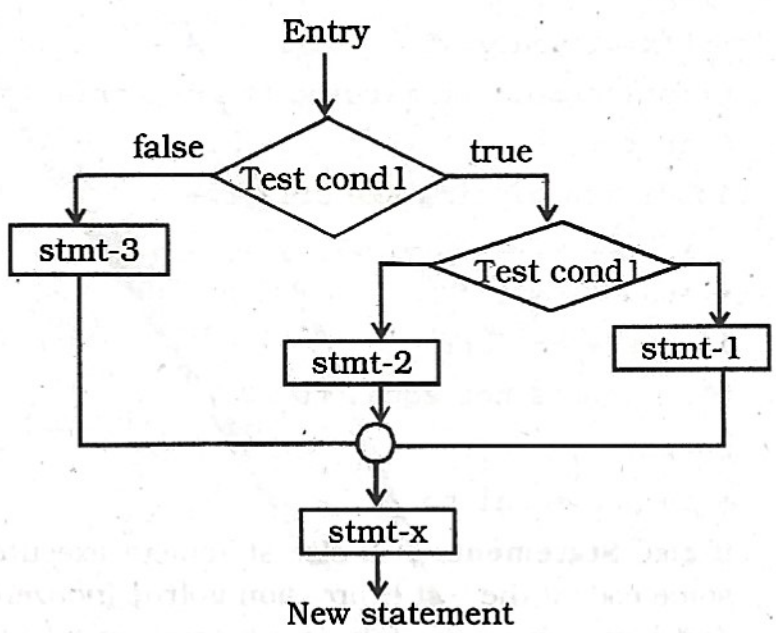
- **if-else Statements :** If-else statement executes some code if the test expression is true (nonzero) and some other code if the test expression is false.



if-else Statements

```
Syntax:
 if(condition):
                    statements
else:
                    statements
e.g.
a=10
 if(a < 100):
    print('less than 100')
 else:
     print('more than equal 100')
 OUTPUT
 less than 100.
```

(III) **Nested if-else statement :** The nested if...else statement allows you to check for multiple test expressions and execute different codes for more than two conditions.



Nested if-else statement:

```
Syntax:
If (condition):
                    statements
elif (condition):
                    statements
else:
                    statements
e.g.
num = float(input("Enter a number: "))
if num >= 0:
   if num == 0:
   print("Zero")
else:
    print("Positive number")
else:
    print("Negative number")
OUTPUT
Enter a number: 5
Positive number
Nested if-else Statements (sort 3 numbers)
first = int(input("Enter the first number:
"))
second = int(input("Enter the second number:
"))
third = int(input("Enter the third number:
"))
small = 0
middle = 0
large = 0
if first < third and first < second:
   small = first
if second < third and second < first:
   small = second
else:
   small = third
elif first < second and first < third:
     middle = first
if second > first and second < third:
   middle = second
else:
    middle = third
elif first > second and first > third:
        large = first
if second > first and second > third:
      large = second
else:
    large = third
print("The numbers in accending order are:
", small, middle, large).
Nested if-else Statements (Check leap year
/ divisibility):-
year = int(input("Enter a year: "))
```

```
if (year % 4) == 0:
   if (year % 100) == 0:
   if (year % 400) == 0:
   print("{0} is a leap year".format(year))
else:
 print("{0} is not a leap year".format(year))
else:
 print("{0} is a leap year".format(year))
else:
 print("{0} is not a leap year".format(year)).
```

ITERATION STATEMENTS (LOOPS)

Iteration statements(loop) are used to execute a block of statements as long as the condition is true. Loops statements are used when we need to run same code again and again.

Python Iteration (Loops) statements are of three type :

(I) While Loop

(II) For Loop

(III) Nested For Loops

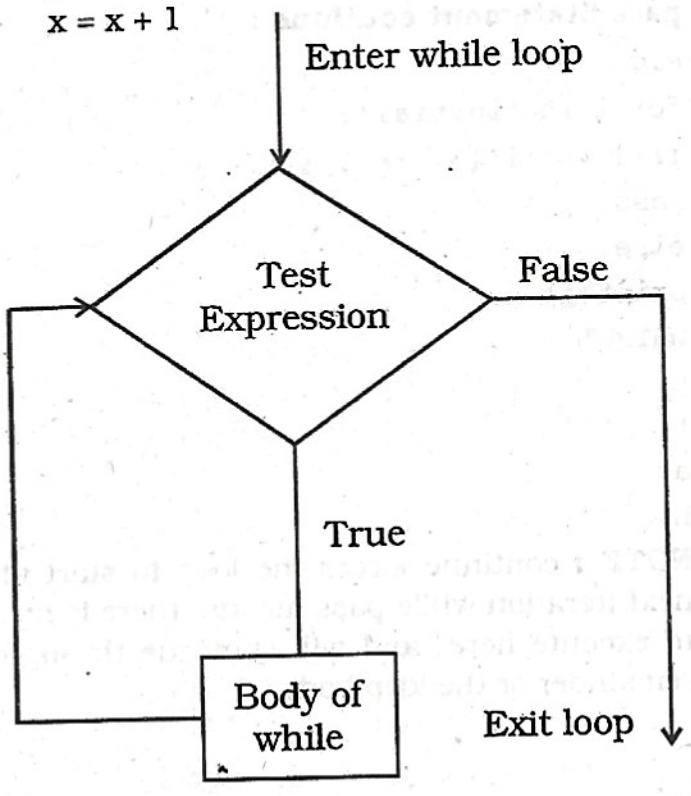
(I) **While Loop :** It is used to execute a block of statement as long as a given condition is true. And when the condition become false, the control will come out of the loop. The condition is checked every time at the beginning of the loop.

Syntax

```
while (condition):
        statement
       [statements]
```

e.g.

```
x = 1
while (x <= 4):
print(x)
x = x + 1
```



- **While Loop continue (While Loop With Else) :**

e.g.

```
x = 1
while (x < 3):
 print('inside while loop value of x is ',x)
x = x + 1
else:
   print('inside else value of x is ', x)
```

Output

inside while loop value of x is 1

inside while loop value of x is 2

inside else value of x is 3

- **While Loop continue (Infinite While Loop) :**

```
e.g.
x = 5
while (x == 5):
print('inside loop')
```

Output

Inside loop

Inside loop

...

...

(II) **For Loop :** It is used to iterate over items of any sequence, such as a list or a string.

Syntax

```
for val in sequence:
   statements
```

e.g.

```
for i in range(3,5):
    print(i)
```

Output

3

4

- For Loop continue:- Example programs

```
for i in range(5,3,-1):
    print(i)
```

Output

5

4

range() Function Parameters

start: Starting number of the sequence.

stop: Generate numbers up to, but not including this number.

step(Optional): Determines the increment between each numbers in the sequence.

- **For Loop continue :** Example programs with range() and len() function

```
fruits = ['banana', 'apple', 'mango']
```

```
for index in range(len(fruits)):
print ('Current fruit :', fruits[index] )
```

range() with len() Function Parameters.

- **For Loop continue :** For Loop With Else

e.g.

```
for i in range(1, 4):
print(i)
else: # Executed because no break in for
print("No Break")
```

Output

1

2

3

No Break

- For Loop continue:- Nested For Loop

e.g.

```
for i in range(1,3):
    for j in range(1,11):
    k=i*j
    print (k,.end=' ')
print()
```

Output

1 2 3 4 5 6 7 8 9 10

2 4 6 8 10 12 14 16 18 20

- **For Loop continue :** Compound Interest calculation:-

```
n=int(input("Enter the principle amount:"))
rate=int(input("Enter the rate:"))
years=int(input("Enter the number of years:"))
for i in range(years):
n=n+((n*rate)/100)
print(n)
```

(III) **Jump Statements :** Jump statements are used to transfer the program's control from one location to another. Means these are used to alter the flow of a loop like - to skip a part of a loop or terminate a loop.

There are three types of jump statements used in python.

(I) break

(II) continue

(III) pass

- break : it is used to terminate the loop.

e.g.

```
for val in "string":
    if val == "i":
     break
    print(val)
print("The end")
```

Output

s

t

r

The end

- **continue :** It is used to skip all the remaining statements in the loop and move controls back to the top of the loop.

e.g.

```
for val in "init":
    if val == "i":
    continue
print(val)
print("The end")
```

Output

n

t

The end

- **pass Statement :** This statement does nothing. It can be used when a statement is required syntactically but the program requires no action.

Use in loop

while True:

pass # Busy-wait for keyboard interrupt (Ctrl+C)

In function : It makes a controller to pass by without executing any code.

e.g.

```
def myfun():
```

pass #if we don't use pass here then error message will be shown print('my program')

OUTPUT

My program

- **pass Statement continue :**

e.g.

```
for i in 'initial':
if(i == 'i'):
pass
else:
print(i)
```

OUTPUT

n

t

a

L

NOTE : continue forces the loop to start at the next iteration while pass means "there is no code to execute here" and will continue through the remainder or the loop body.

❑❑❑

7 FUNCTIONS
(NCERT CLASS 11)

INTRODUCTION

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

Functions can be categorized as belonging to

(I) Modules

(II) Built in

(III) User Defined

- **Module:-**A module is a file containing Python definitions (i.e. functions) and statements. Standard library of Python is extended as module(s) to a programmer. Definitions from the module can be used within the code of a program. To use these modules in the program, a programmer needs to import the module. Once you import a module, you can reference (use), any of its functions or variables in your code. There are many ways to import a module in your program, the one's which you should know are: import and from.

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

```
modulename1 [,modulename2, ——]
```

Example

```
>>> import math
```

On execution of this statement, Python will

(i) search for the file "math.py".

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

(iii) then execute the statements in the module.

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

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

Example

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

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

This statement invokes the sqrt ( ) function. Many function invoke statement(s), such as

```
>>> type ( )
>>> int ( ), etc.
```

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

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

Example

```
>>> from math import sqrt
value = sqrt (25)
```

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

```
from modulename import *
```

will import everything from the file.

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

Functions available in math module:

Name of the function	Description	Example
ceil(x)	It returns the smallest integer not less than x, where *x is a numeric expression.*	math.ceil(-45.17) **-45.0** math.ceil(100.12) **101.0** math.ceil(100.72) **101.0**
floor(x)	It returns the largest integer not greater than x, where *x is a numeric expression.*	math.floor(-45.17) **-46.0** math.floor(100.12) **100.0** math.floor(100.72) **100.0**
fabs(x)	It returns the absolute value of x, *where x is a numeric value.*	math.fabs(-45.17) **45.17** math.fabs(100.12) **100.12** math.fabs(100.72) **100.72**
exp(x)	It returns exponential of x: ex, where x is a numeric expression.	math.exp(-45.17) **2.41500621326e-20** math.exp(100.12) **3.03084361407e+43** math.exp(100.72) **5.52255713025e+43**
log(x)	It returns natural logarithm of x, for x > 0, where *x is a numeric expression.*	math.log(100.12), **4.60636946656** math.log(100.72) **4.61234438974**
log10(x)	It returns base-10 logarithm of x for x > 0, where *x is a numeric expression.*	math.log10(100.12) **2.00052084094** math.log10(100.72) **2.0031157171**
pow(x, y)	It returns the value of xy; *where x and y are numeric expressions.*	math.pow(100, 2) **10000.0** math.pow(100, -2) **0.0001** math.pow(2, 4) **16.0** math.pow(3, 0) **1.0**
sqrt (x)	It returns the square root of x for x > 0, where x is a numeric expression.	math.sqrt(100) **10.0** math.sqrt(7) **2.64575131106**

cos (x)	It returns the cosine of x in radians, *where x is a numeric expression.*	math.cos(3) **-0.9899924966** math.cos(-3) **-0.9899924966** math.cos(0) **1.0** math.cos(math.pi) **-1.0**
sin (x)	It returns the sine of x, in radians, *where x must be a numeric value.*	math.sin(3) **0.14112000806** math.sin(-3) **-0.14112000806** math.sin(0) **0.0**
tan (x)	It returns the tangent of x in radians, *where x must be a numeric value.*	math.tan(3) **-0.142546543074** math.tan(-3) **0.142546543074** math.tan(0) **0.0**
degrees (x)	It converts angle x from radians to degrees, *where x must be a numeric value.*	math.degrees(3) **171.887338539** math.degrees(-3) **-171.887338539** math.degrees(0) **0.0**
radians(x)	It converts angle x from degrees to radians, *where x must be a numeric value.*	math.radians(3) **0.0523598775598** math.radians(-3) **-0.0523598775598** math.radians(0) **0.0**

Some functions from random module are :

Name Of Function	Description	Example
random ( )	It returns a random float x, such that $0 \leq x<1$	>>>random.random ( ) **0.281954791393** >>>random.random ( ) **0.309090465205**

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

uniform (a,b)	It returns a floating point number x, such that a <= x < b	>>>random.uniform (5, 10) **5.52615217015**
randrange ([start,] stop [,step])	It returns a random item from the given range ,1000,3)	>>>random.randrange(100

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

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

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

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

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

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

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

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

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

Example :

Suppose we have two functions fn1 & fn2, such that

a= fn2 (x)

b= fn1 (a)

then call to the two functions can be combined as

b= fn1 (fn2 (x))

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

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

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

- **User Defined Functions:-**

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

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

BLOCK OF STATEMENTS

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

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

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

Let's write a function to greet the world:

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

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

- **Function Header :** It begins with the keyword def and ends with colon and contains the function identification details. As it ends with colon, we can say that what follows next is, block of statements.
- **Function Body :**Consisting of sequence of indented (4 space) Python statement(s), to perform a task. Defining a function will create a variable with same name, but does not generate any result. The body of the function gets executed only when the function is called/invoked. Function call contains the name of the function (being executed) followed by the list of values (i.e. arguments) in parenthesis. These arguments are assigned to parameters from LHS.

```
>>> sayHello ()  # Call/invoke statement of this function
Will produce following on screen
Hello World!
```

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

Example

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

fun ( ) # calls the function selected.

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

def fun (a):

- **Let's explore Function body :-** The first statement of the function body can optionally be a string constant, docstring, enclosed in triple quotes. It contains the essential information that someone might need about the function, such as:-
  (I) What function does (without How it does) i.e. summary of its purpose.
  (II) Type of parameters it takes
  (III) Effect of parameter on behavior of functions, etc.

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

Example:

```
def area (radius):
""" calculates area of a circle. docstring begins
require an integer or float value to calculate area.
returns the calculated value to calling function
""" docstring ends
a=radius**2
        return a
```

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

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

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

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

We could have written

```
radius **2
```

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

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

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

DocString Conventions:

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

PARAMETERS AND ARGUMENTS

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

Radius is a parameter to function area.

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

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

Example :

```
of argument in function call
>>> area (5)
```

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

SCOPE OF VARIABLES

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

(I) **Global Scope:-** A variable, with global scope can be used anywhere in the program. It can be created by defining a variable outside the scope of any function/block.
(II) **Local Scope:-**A variable with local scope can be accessed only within the function/block that it is created in. When a variable is created inside the function/block, the variable becomes local to it. A local variable only exists while the function is executing.

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

More on defining Functions

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

Example :

→ Default value to parameter

?

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

The function greet () is used to print a message (string) given number of times. If thesecond argument value, is not specified, then parameter times work with the default value provided to it. In the first call to greet (), only one argument value is provided,which is passed on to the first parameter from LHS and the string is printed only onceas the variable times take default value 1. In the second call to greet (), we supply both the argument values a string and 2, saying that we want to print the message twice. So now, parameter times get the value 2 instead of default 1 and the message is printed twice.

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

Note:

(I) The default value assigned to the parameter should be a constant only.

(II) Only those parameters which are at the end of the list can be given default value.

(III) You cannot have a parameter on left with default argument value, without assigning default values to parameters lying on its right side.

(IV) The default value is evaluated only once, at the point of function definition.

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

- While using keyword arguments, following should be kept in mind:
  - (I) An argument list must have any positional arguments followed by any keywords arguments.
  - (II) Keywords in argument list should be from the list of parameters name only.
  - (III) No parameter should receive value more than once.
  - (IV) Parameter names corresponding to positional arguments cannot be used as keywords in the same calls.
- Advantages of writing functions with keyword arguments are:
  - (I) Using the function is easier as we do not need to remember about the order of the arguments.
  - (II) We can specify values of only those parameters to which we want to, as – other parameters have default argument values.

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

(I) Pass function as parameter

(II) Use/invoke function by two names.

Example :

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

Example :

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

FLOW OF EXECUTION OF PROGRAM CONTAINING FUNCTION CALL

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

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

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

Note :

(I) Python does not allow you to call a function before the function is declared.

(II) When you write the name of a function without parenthesis, it is interpreted as the reference, when you write the function name with parenthesis, the interpreter invoke the function (object). □□□