UNIT II

DATA, EXPRESSIONS, STATEMENTS

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

1. INTRODUCTION TO PYTHON:

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language.

It was created by Guido van Rossum during 1985- 1990.

Python got its name from "Monty Python's flying circus". Python was released in the year 2000.

- **Python is interpreted**: Python is processed at runtime by the interpreter. You do not need to compile your program before executing it.
- **Python is Interactive**: You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- **Python is Object-Oriented**: Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- **Python is a Beginner's Language:** Python is a great language for the beginner-level programmers and supports the development of a wide range of applications.

1.1. Python Features:

- **Easy-to-learn:** Python is clearly defined and easily readable. The structure of the program is very simple. It uses few keywords.
- **Easy-to-maintain:** Python's source code is fairly easy-to-maintain.
- **Portable:** Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **Interpreted:** Python is processed at runtime by the interpreter. So, there is no need to compile a program before executing it. You can simply run the program.
- **Extensible:** Programmers can embed python within their C,C++,Java script ,ActiveX, etc.
- **Free and Open Source:** Anyone can freely distribute it, read the source code, and edit it.
- **High Level Language:** When writing programs, programmers concentrate on solutions of the current problem, no need to worry about the low level details.
- **Scalable:** Python provides a better structure and support for large programs than shell scripting.

1.2. Applications:

- Bit Torrent file sharing
- Google search engine, Youtube
- Intel, Cisco, HP, IBM
- i–Robot
- NASA

❖ Facebook, Drop box

1.3. Python interpreter:

Interpreter: To execute a program in a high-level language by translating it one line at a time.

Compiler: To translate a program written in a high-level language into a low-level language all at once, in preparation for later execution.

Compiler	Interpreter
Compiler Takes **Entire** program as input	Interpreter Takes **Single** instruction as input
Intermediate Object Code is **Generated**	**No** Intermediate Object Code is **Generated**
Conditional Control Statements are Executes **faster**	Conditional Control Statements are Executes **slower**
Memory Requirement is **More**(Since Object Code is Generated)	**Memory Requirement** is **Less**
Program need not be **compiled** every time	Every time higher level program is converted into lower level program
Errors are displayed after **entire program** is checked	**Errors** are displayed for **every instruction** interpreted (if any)
Example : C Compiler	**Example** : PYTHON

1.4 MODES OF PYTHON INTERPRETER:

Python Interpreter is a program that reads and executes Python code. It uses 2 modes of Execution.

1. Interactive mode
2. Script mode

Interactive mode:

- ❖ Interactive Mode, as the name suggests, allows us to interact with OS.
- ❖ When we type Python statement, **interpreter displays the result(s) immediately.**

Advantages:

- ❖ Python, in interactive mode, is good enough to learn, experiment or explore.
- ❖ Working in interactive mode is convenient for beginners and for testing small pieces of code.

Drawback:

- ❖ We cannot save the statements and have to retype all the statements once again to re-run them.

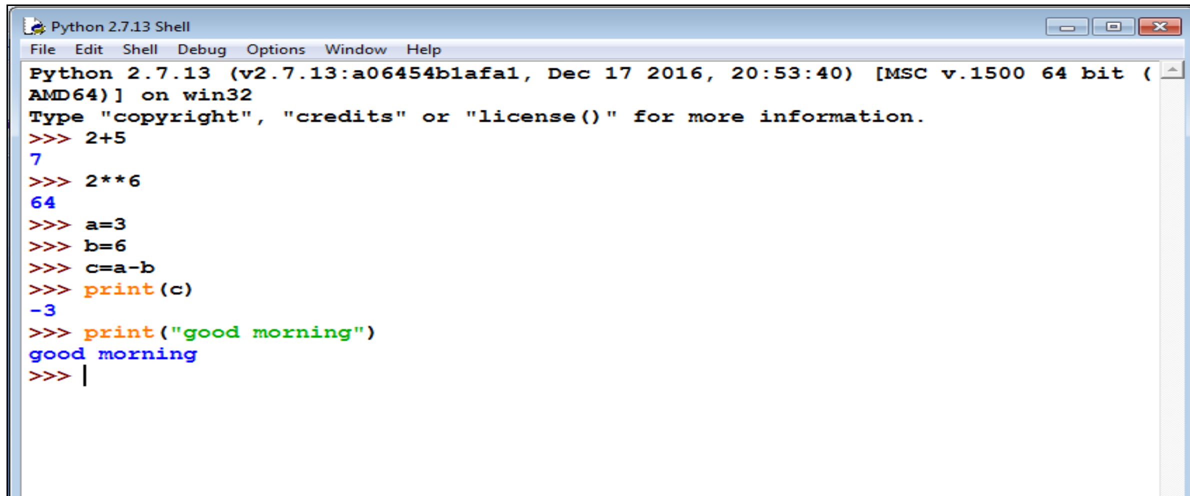
In interactive mode, you type Python programs and the interpreter displays the result:

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

The chevron, >>>, is the prompt the interpreter uses to indicate that it is ready for you to enter code. If you type 1 + 1, the interpreter replies 2.

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

This is an example of a print statement. It displays a result on the screen. In this case, the result is the words.



<u>Script mode:</u>

- In script mode, we type python program in a file and then use interpreter to execute the content of the file.
- Scripts can be <u>saved to disk for future use</u>. **Python scripts have the extension .py**, meaning that the filename ends with **.py**
- Save the code with **filename.py** and run the interpreter in script mode to execute the script.

<u>Example1:</u>

```
print(1)
x = 2
print(x)
```

Output:

```
>>>1
   2
```

Interactive mode	Script mode
A way of using the Python interpreter by typing commands and expressions at the prompt.	A way of using the Python interpreter to read and execute statements in a script.
Cant save and edit the code	Can save and edit the code
If we want to experiment with the code, we can use interactive mode.	If we are very clear about the code, we can use script mode.
we cannot save the statements for further use and we have to retype all the statements to re-run them.	we can save the statements for further use and we no need to retype all the statements to re-run them.
We can see the results immediately.	We cant see the code immediately.

Integrated Development Learning Environment (IDLE):

- Is a **graphical user interface** which is completely written in Python.
- It is bundled with the default implementation of the python language and also comes with optional part of the Python packaging.

Features of IDLE:

- Multi-window text editor with syntax highlighting.

- Auto completion with **smart indentation**.
- **Python shell** to display output with syntax highlighting.

2.VALUES AND DATA TYPES

Value:

Value can be any letter ,number or string.

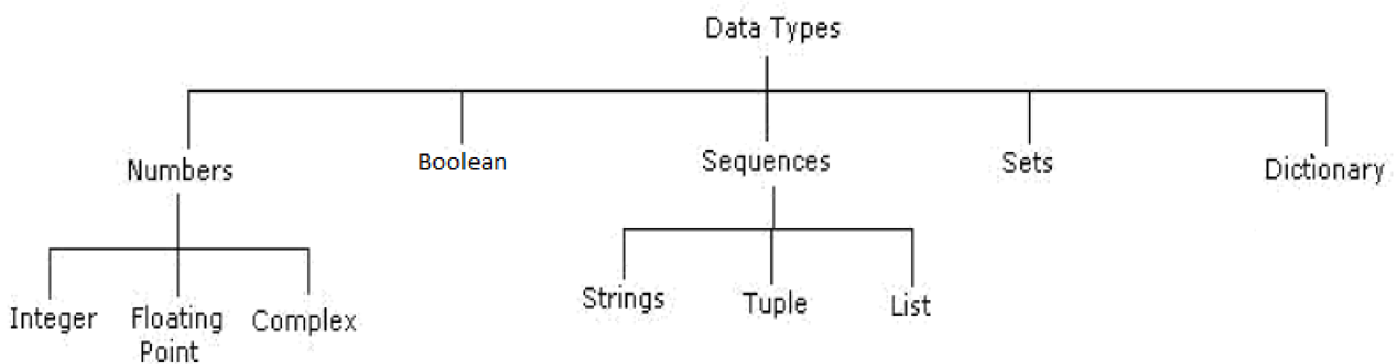
Eg, Values are 2, 42.0, and 'Hello, World!'. (These values belong to different datatypes.)

Data type:

Every value in Python has a data type.

It is a set of values, and the allowable operations on those values.

Python has four standard data types:



2.1Numbers:

- Number data type stores **Numerical Values.**
- This data type is immutable [i.e. values/items cannot be changed].
- Python supports integers, floating point numbers and complex numbers. They are defined as,

Integers	Long	Float	Complex
- They are often called just integers or **int**. - They are positive or negative whole numbers with no decimal point.	-They are long integers. -They can also be represented in octal and hexadecimal representation.	-They are written with a decimal point dividing the integer and the fractional parts.	-They are of the form **a + bj**, where a and b are floats and j represents the square root of -1 (which is an imaginary number). -The real part of the number is a, and the imaginary part is b.
Eg, 56	**Eg, 5692431L**	**Eg, 56.778**	**Eg, square root of-1 is a complex number**

2.2 Sequence:

- A sequence is an **ordered collection of items**, indexed by positive integers.
- It is a combination **of mutable** (value can be changed) **and immutable** (values cannot be changed) data types.

- There are three types of sequence data type available in Python, they are
  1. **Strings**
  2. **Lists**
  3. **Tuples**

2.2.1 Strings:

- A String in Python consists of a series or sequence of characters - letters, numbers, and special characters.
- Strings are marked by quotes:
  - single quotes (' ') Eg, 'This a string in single quotes'
  - double quotes (" ") Eg, "'This a string in double quotes'"
  - triple quotes(""" """) Eg, This is a paragraph. It is made up of multiple lines and sentences."""
- Individual character in a string is accessed using a subscript (index).
- Characters can be accessed using indexing and slicing operations

Strings are immutable i.e. the contents of the string cannot be changed after it is created.

Indexing:

String A	H	E	L	L	O
Positive Index	0	1	2	3	4
Negative Index	-5	-4	-3	-2	-1

- Positive indexing helps in accessing the string from the beginning
- Negative subscript helps in accessing the string from the end.
- Subscript 0 or –ve n(where n is length of the string) displays the first element.
  Example: A[0] or A[-5] will display "H"
- Subscript 1 or –ve (n-1) displays the second element.
  Example: A[1] or A[-4] will display "E"

Operations on string:

i. Indexing
ii. Slicing
iii. Concatenation
iv. Repetitions
v. Member ship

Creating a string	>>> s="good morning"	Creating the list with elements of different data types.

Indexing	>>> **print(s[2])**
	o
	>>> **print(s[6])**
	O
		❖ Accessing the item in the position 0
		❖ Accessing the item in the position 2

Slicing(ending position -1)	>>> **print(s[2:])** od morning	- Displaying items from 2nd till last.

Slice operator is used to extract part of a data type	>>> **print(s[:4])** Good	- Displaying items from 1st position till 3rd .
Concatenation	>>>**print(s+"friends")** good morningfriends	-Adding and printing the characters of two strings.
Repetition	**>>>print(s*2)** good morninggood morning	Creates new strings, concatenating multiple copies of the same string
in, not in (membership operator)	>>> s="good morning" **>>>"m" in s** **True** **>>> "a" not in s** **True**	Using membership operators to check a particular character is in string or not. Returns true if present.

2.2.2 Lists

- List is an ordered sequence of items. Values in the list are called elements / items.
- It can be written as a list of comma-separated items (values) between **square brackets[ ]**.
- Items in the lists can be of different data types.

Operations on list:

Indexing
Slicing
Concatenation
Repetitions
Updation, Insertion, Deletion

Creating a list	>>>**list1=["python", 7.79, 101, "hello"]** >>>**list2=["god",6.78,9]**	Creating the list with elements of different data types.
Indexing	>>>**print(list1[0])** python >>> **list1[2]** 101	- Accessing the item in the position 0 - Accessing the item in the position 2
Slicing(ending position -1) Slice operator is used to extract part of a string, or some part of a list Python	>>> **print(list1[1:3])** [7.79, 101] >>>**print(list1[1:])** [7.79, 101, 'hello']	- Displaying items from 1st till 2nd. - Displaying items from 1st position till last.
Concatenation	>>>**print(list1+list2)** ['python', 7.79, 101, 'hello', 'god',	-Adding and printing the items of two lists.

	6.78, 9]	
Repetition	>>> **list2*3** **['god', 6.78, 9, 'god', 6.78, 9, 'god', 6.78, 9]**	Creates new strings, concatenating multiple copies of the same string
Updating the list	>>> **list1[2]=45** >>>print(list1) ['python', 7.79, 45, 'hello']	Updating the list using index value

Inserting an element	>>> **list1.insert(2,"program")**
	>>> print(list1)
	['python', 7.79, 'program', 45, 'hello']
		Inserting an element in 2nd position

Removing an element	>>> **list1.remove(45)**
	>>> print(list1)
	['python', 7.79, 'program', 'hello']
		Removing an element by giving the element directly

2.2.4Tuple:

- A tuple is same as list, except that the set of elements is **enclosed in parentheses** instead of square brackets.
- **A tuple is an immutable list.** i.e. once a tuple has been created, you can't add elements to a tuple or remove elements from the tuple.
- Benefit of Tuple:
- Tuples are faster than lists.
- If the user wants to protect the data from accidental changes, tuple can be used.
- Tuples can be used as keys in dictionaries, while lists can't.

Basic Operations:

Creating a tuple	>>>**t=("python", 7.79, 101, "hello")**	Creating the tuple with elements of different data types.
Indexing	>>>**print(t[0])** python >>> **t[2]** 101	❖ Accessing the item in the position 0 ❖ Accessing the item in the position 2
Slicing(ending position -1)	>>>**print(t[1:3])** (7.79, 101)	❖ Displaying items from 1st till 2nd.
Concatenation	>>> **t+("ram", 67)** ('python', 7.79, 101, 'hello', 'ram', 67)	❖ Adding tuple elements at the end of another tuple elements
Repetition	>>>**print(t*2)** ('python', 7.79, 101, 'hello', 'python', 7.79, 101, 'hello')	❖ Creates new strings, concatenating multiple copies of the same string

Altering the tuple data type leads to error. Following error occurs when user tries to do.

```
>>> t[0]="a"
Trace back (most recent call last):
  File "<stdin>", line 1, in <module>
Type Error: 'tuple' object does not support item assignment
```

2.3 Mapping

-This data type is unordered and mutable.

-Dictionaries fall under Mappings.

2.3.1Dictionaries:

- Lists are ordered sets of objects, whereas **dictionaries are unordered sets**.
- Dictionary is created by using **curly brackets**. i,e. {}
- Dictionaries **are accessed via keys** and not via their position.
- A dictionary is an associative array (also known as hashes). Any key of the dictionary is associated (or mapped) to a value.
- The values of a dictionary can be any Python data type. So dictionaries are **unordered key-value-pairs(**The association of a key and a value is called a key-value pair)

Dictionaries don't support the sequence operation of the sequence data types like strings, tuples and lists.

Creating a dictionary	**>>> food = {"ham":"yes", "egg" : "yes", "rate":450 }** >>>print(food) {'rate': 450, 'egg': 'yes', 'ham': 'yes'}	Creating the dictionary with elements of different data types.
Indexing	>>>> **print(food["rate"])** 450	Accessing the item with keys.
Slicing(ending position -1)	>>>**print(t[1:3])** (7.79, 101)	Displaying items from 1st till 2nd.

If you try to access a key which doesn't exist, you will get an error message:

```
>>> words = {"house" : "Haus", "cat":"Katze"}
>>> words["car"]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'car'
```

Data type	Compile time	Run time
int	a=10	a=int(input("enter a"))
float	a=10.5	a=float(input("enter a"))
string	a="panimalar"	a=input("enter a string")
list	a=[20,30,40,50]	a=list(input("enter a list"))
tuple	a=(20,30,40,50)	a=tuple(input("enter a tuple"))

3.Variables,Keywords Expressions, Statements, Comments, Docstring ,Lines And Indentation, Quotation In Python, Tuple Assignment:

3.1VARIABLES:

- A variable allows us to store a value by assigning it to a name, which can be used later.
- Named memory locations to store values.
- Programmers generally choose names for their variables that are meaningful.
- It can be of any length. No space is allowed.
- We don't need to declare a variable before using it. In Python, we simply assign a value to a variable and it will exist.

Assigning value to variable:

Value should be given on the right side of assignment operator(=) and variable on left side.

```
>>>counter =45
print(counter)
```

Assigning a single value to several variables simultaneously:

```
>>> a=b=c=100
```

Assigning multiple values to multiple variables:

```
>>> a,b,c=2,4,"ram"
```

3.2KEYWORDS:

- Keywords are the reserved words in Python.
- We cannot use a keyword as variable name, function name or any other identifier.
- They are used to define the syntax and structure of the Python language.
- Keywords are case sensitive.

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

3.3IDENTIFIERS:

Identifier is the name given to entities like class, functions, variables etc. in Python.

- Identifiers can be a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore (_).

- all are valid example.
- An identifier cannot start with a digit.
- Keywords cannot be used as identifiers.
- Cannot use special symbols like !, @, #, $, % etc. in our identifier.
- Identifier can be of any length.

Example:

Names like myClass, var_1, and **this_is_a_long_variabl**e

Valid declarations	**Invalid declarations**
Num	Number 1
Num	num 1
Num1	addition of program
_NUM	1Num
NUM_temp2	Num.no
IF	if
Else	else

3.4 STATEMENTS AND EXPRESSIONS:

3.4.1 **Statements**:

-Instructions that a Python interpreter can executes are called statements.

-A statement is a unit of code like creating a variable or displaying a value.

```
>>> n = 17
>>> print(n)
```

Here, The first line is an assignment statement that gives a value to n.
The second line is a print statement that displays the value of n.

3.4.2**Expressions**:

-An expression is a **combination of values, variables, and operators.**

- A value all by itself is considered an expression, and also a variable.
- So the following are all legal expressions:

```
>>> 42
42
>>> a=2
>>> a+3+2
7
>>> z=("hi"+"friend")
>>> print(z)
hifriend
```

3.5 INPUT AND OUTPUT

INPUT: Input is data entered by user (end user) in the program.
In python, **input () function** is available for input.

Syntax for input() is:
variable = input (“data”)

Example:

```
>>> x=input("enter the name:")
enter the name: george
```

```
>>>y=int(input("enter the number"))
enter the number 3
```

#python accepts string as default data type. conversion is required for type.

OUTPUT: Output can be displayed to the user using Print statement .

Syntax:

print (expression/constant/variable)

Example:

```
>>> print ("Hello")
Hello
```

3.6 COMMENTS:

- A **hash sign (#)** is the beginning of a comment.
- Anything written after # in a line is ignored by interpreter.

  Eg:percentage = (minute * 100) / 60 **# calculating percentage of an hour**
- Python **does not have multiple-line commenting feature.** You have to comment each line individually as follows :

Example:

```
# This is a comment.
# This is a comment, too.
# I said that already.
```

3.7 DOCSTRING:

- Docstring is short for documentation string.
- It is a string that occurs as the first statement in a module, function, class, or method definition. We must write what a function/class does in the docstring.
- **Triple quotes** are used while writing docstrings.

Syntax:

functionname__doc.__

Example:

```
def double(num):
    """Function to double the value"""
    return 2*num
>>> print(double.__doc__)
Function to double the value
```

3.8 LINES AND INDENTATION:

- Most of the programming languages like C, C++, Java use braces { } to define a block of code. But, python uses indentation.
- Blocks of code are denoted by line indentation.
- It is a space given to the block of codes for class and function definitions or flow control.

Example:

```
a=3
b=1
if a>b:
   print("a is greater")
else:
   print("b is greater")
```

3.9 QUOTATION IN PYTHON:

Python accepts single ('), double (") and triple (''' or """) quotes to denote string literals. Anything that is represented using quotations are considered as string.

- single quotes (' ') Eg, 'This a string in single quotes'
- double quotes (" ") Eg, "'This a string in double quotes'"
- triple quotes(""" """) Eg, This is a paragraph. It is made up of multiple lines and sentences."""

3.10 TUPLE ASSIGNMENT

- An assignment to all of the elements in a tuple using a single assignment statement.
- Python has a very powerful **tuple assignment** feature that allows a tuple of variables on the left of an assignment to be assigned values from a tuple on the right of the assignment.
- The left side is a tuple of variables; the right side is a tuple of values.
- Each value is assigned to its respective variable.
- All the expressions on the right side are evaluated before any of the assignments. This feature makes tuple assignment quite versatile.
- Naturally, the number of variables on the left and the number of values on the right have to be the same.

```
>>> (a, b, c, d) = (1, 2, 3)
ValueError: need more than 3 values to unpack
```

Example:

-It is useful to swap the values of two variables. With **conventional assignment statements**, we have to use a temporary variable. For example, to swap a and b:

Swap two numbers	Output:
a=2;b=3 print(a,b) temp = a a = b b = temp print(a,b)	 (2, 3) (3, 2) >>>

-Tuple assignment solves this problem neatly:

```
(a, b) = (b, a)
```

-One way to think of tuple assignment is as <u>tuple packing/unpacking</u>.
In tuple packing, the values on the left are 'packed' together in a tuple:

```
>>> b = ("George", 25, "20000")    # tuple packing
```

-In tuple unpacking, **<u>the values in a tuple on the right are 'unpacked'</u> <u>into the variables/names on the right:</u>**

```
>>> b = ("George", 25, "20000")      # tuple packing
>>> (name, age, salary) = b    # tuple unpacking
>>> name
'George'
>>> age
25
>>> salary
'20000'
```

-The right side can be any kind of sequence (string,list,tuple)
<u>Example:</u>
-To split an email address in to user name and a domain

```
>>> mailid='god@abc.org'
>>> name,domain=mailid.split('@')
>>> print name
god
>>> print (domain)
abc.org
```

4.OPERATORS:

- ❖ Operators are the constructs which can manipulate the value of operands.
- ❖ Consider the **expression *4 + 5 = 9.*** Here, **4 and 5 are called operands** and **<u>+ is called operator</u>**
- ❖ Types of Operators:

  -Python language supports the following types of operators
  - Arithmetic Operators
  - Comparison (Relational) Operators
  - Assignment Operators
  - Logical Operators
  - Bitwise Operators
  - Membership Operators
  - Identity Operators

4.1 Arithmetic operators:

They are used to perform **mathematical operations** like addition, subtraction, multiplication etc. **Assume, a=10 and b=5**

Operator	Description	Example
+ Addition	Adds values on either side of the operator.	a + b = 30
- Subtraction	Subtracts right hand operand from left hand operand.	a – b = -10
* Multiplication	Multiplies values on either side of the operator	a * b = 200
/ Division	Divides left hand operand by right hand operand	b / a = 2
% Modulus	Divides left hand operand by right hand operand and returns remainder	b % a = 0
** Exponent	Performs exponential (power) calculation on operators	a**b =10 to the power 20
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed	5//2=2

Examples

```
a=10
b=5
print("a+b=",a+b)
print("a-b=",a-b)
print("a*b=",a*b)
print("a/b=",a/b)
print("a%b=",a%b)
print("a//b=",a//b)
print("a**b=",a**b)
```

Output:

```
a+b= 15
a-b= 5
a*b= 50
a/b= 2.0
a%b= 0
a//b= 2
a**b= 100000
```

4.2 Comparison (Relational) Operators:

- Comparison operators are used to compare values.
- It either returns True or False according to the condition. **Assume, a=10 and b=5**

Operator	Description	Example
==	If the values of two operands are equal, then the condition	(a == b) is

	becomes true.	not true.
!=	If values of two operands are not equal, then condition becomes true.	(a!=b) is true
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	(a > b) is not true.
<	If the value of left operand is less than the value of right operand, then condition becomes true.	(a < b) is true.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	(a >= b) is not true.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) is true.

Example

```
a=10
b=5
print("a>b=>",a>b)
print("a>b=>",a<b)
print("a==b=>",a==b)
print("a!=b=>",a!=b)
print("a>=b=>",a<=b)
print("a>=b=>",a>=b)
```

Output:

```
a>b=> True
a>b=> False
a==b=> False
a!=b=> True
a>=b=> False
a>=b=> True
```

4.3 Assignment Operators:

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

Operator	Description	Example
=	Assigns values from right side operands to left side operand	c = a + b assigns value of a + b into c
+= Add AND	It adds right operand to the left operand and assign the result to left operand	c += a is equivalent to c = c + a
-= Subtract AND	It subtracts right operand from the left operand and assign the result to left operand	c -= a is equivalent to c = c - a

*= Multiply AND	It multiplies right operand with the left operand and assign the result to left operand	c *= a is equivalent to c = c * a
/= Divide AND	It divides left operand with the right operand and assign the result to left operand	c /= a is equivalent to c = c / ac /= a is equivalent to c = c / a
%= Modulus AND	It takes modulus using two operands and assign the result to left operand	c %= a is equivalent to c = c % a
**= Exponent AND	Performs exponential (power) calculation on operators and assign value to the left operand	c **= a is equivalent to c = c ** a
//= Floor Division	It performs floor division on operators and assign value to the left operand	c //= a is equivalent to c = c // a

Example

```
a = 21
b = 10
c = 0
c = a + b
print("Line 1 - Value of c is ", c)
c += a
print("Line 2 - Value of c is ", c)
c *= a
print("Line 3 - Value of c is ", c)
c /= a
print("Line 4 - Value of c is ", c)
c  = 2
c %= a
print("Line 5 - Value of c is ", c)
c **= a
print("Line 6 - Value of c is ", c)
c //= a
print("Line 7 - Value of c is ", c)
```

Output

```
Line 1 - Value of c is  31
Line 2 - Value of c is  52
Line 3 - Value of c is  1092
Line 4 - Value of c is  52.0
Line 5 - Value of c is  2
Line 6 - Value of c is  2097152
Line 7 - Value of c is  99864
```

4.4 Logical Operators:

-Logical operators are the and, or, not operators.

Operator	Meaning	Example
and	True if both the operands are true	x and y
or	True if either of the operands is true	x or y
not	True if operand is false (complements the operand)	not x

Example

```
a = True
b = False
print('a and b is',a and b)
print('a or b is',a or b)
print('not a is',not a)
```

Output

```
x and y is False
x or y is True
not x is False
```

4.5 Bitwise Operators:

- A **bitwise operation** operates on one or more **bit** patterns at the level of individual bits

Example: Let x = 10 (0000 1010 in binary) and
y = 4 (0000 0100 in binary)

Operator	Meaning	Example
&	Bitwise AND	x& y = 0 (0000 0000)
|	Bitwise OR	x | y = 14 (0000 1110)
~	Bitwise NOT	~x = -11 (1111 0101)
^	Bitwise XOR	x ^ y = 14 (0000 1110)
>>	Bitwise right shift	x>> 2 = 2 (0000 0010)
<<	Bitwise left shift	x<< 2 = 40 (0010 1000)

Example

```
a = 60          # 60 = 0011 1100
b = 13          # 13 = 0000 1101
c = 0
c = a & b;       # 12 = 0000 1100
print "Line 1 - Value of c is ", c
c = a | b;      # 61 = 0011 1101
print "Line 2 - Value of c is ", c
c = a ^ b;       # 49 = 0011 0001
print "Line 3 - Value of c is ", c
c = ~a;         # -61 = 1100 0011
```

Output

```
Line 1 - Value of c is 12
Line 2 - Value of c is 61
Line 3 - Value of c is 49
Line 4 - Value of c is -61
Line 5 - Value of c is 240
Line 6 - Value of c is 15
```

```
print "Line 4 - Value of c is ", c
c = a << 2;      # 240 = 1111 0000
print "Line 5 - Value of c is ", c
c = a >> 2;      # 15 = 0000 1111
print "Line 6 - Value of c is ", c
```

4.6 Membership Operators:

- Evaluates to find a value or a variable is in the specified sequence of string, list, tuple, dictionary or not.
- Let, **x=[5,3,6,4,1].** To check particular item in list or not**, in and not in** operators are used.

Operator	Meaning	Example
in	True if value/variable is found in the sequence	5 in x
not in	True if value/variable is not found in the sequence	5 not in x

Example:

x=[5,3,6,4,1]
>>> **5 in x**
True
>>> **5 not in x**
False

4.7 Identity Operators:

- They are used to check if two values (or variables) are located on the same part of the
memory.

Operator	Meaning	Example
is	True if the operands are identical (refer to the same object)	x is True
is not	True if the operands are not identical (do not refer to the same object)	x is not True

Example

```
x = 5
y = 5
x2 = 'Hello'
y2 = 'Hello'
print(x1 is not y1)
print(x2 is y2)
```

Output
False
True

5.OPERATOR PRECEDENCE:

When an expression contains **more than one operator, the order of evaluation** depends on the order of operations.

Operator	Description
**	Exponentiation (raise to the power)
~ + -	Complement, unary plus and minus (method names for the last two are +@ and -@)
* / % //	Multiply, divide, modulo and floor division
+ -	Addition and subtraction
>> <<	Right and left bitwise shift
&	Bitwise 'AND'
^ |	Bitwise exclusive `OR' and regular `OR'
<= < > >=	Comparison operators
<> == !=	Equality operators
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators

-For mathematical operators, Python follows mathematical convention.

-The acronym **PEMDAS** (Parentheses, Exponentiation, Multiplication, Division, Addition, Subtraction**)** is a useful way to remember the rules:

- Parentheses have the highest precedence and can be used to force an expression to evaluate in the order you want. Since expressions in parentheses are evaluated first, 2 * (3-1)is 4, and (1+1)**(5-2) is 8.
- You can also use parentheses to make an expression easier to read, as in (minute * 100) / 60, even if it doesn't change the result.
- Exponentiation has the next highest precedence, so 1 + 2**3 is 9, not 27, and 2 *3**2 is 18, not 36.
- Multiplication and Division have higher precedence than Addition and Subtraction. So 2*3-1 is 5, not 4, and 6+4/2 is 8, not 5.
- Operators with the same precedence are evaluated from left to right (except exponentiation).

Example:

<table>
<tr>
<td>a=9-12/3+3*2-1
a=?
a=9-4+3*2-1
a=9-4+6-1
a=5+6-1
a=11-1
a=10</td>
<td>A=2*3+4%5-3/2+6
A=6+4%5-3/2+6
A=6+4-3/2+6
A=6+4-1+6
A=10-1+6
A=9+6
A=15</td>
<td>find m=?
m=-43||8&&0||-2
m=-43||0||-2
m=1||-2
m=1</td>
</tr>
<tr>
<td>a=2,b=12,c=1
d=a<b>c
d=2<12>1
d=1>1
d=0</td>
<td>a=2,b=12,c=1
d=a<b>c-1
d=2<12>1-1
d=2<12>0
d=1>0
d=1</td>
<td>a=2*3+4%5-3//2+6
a=6+4-1+6
a=10-1+6
a=15</td>
</tr>
</table>

6.Functions, Function Definition And Use, Function call, Flow Of Execution, Function Prototypes, Parameters And Arguments, Return statement, Argumentstypes,Modules

6.1 FUNCTIONS:

- **Function is a sub program which consists of set of instructions used to perform a specific task. A large program is divided into basic building blocks called function.**

Need For Function:

- When the program is too complex and large they are divided into parts. Each part is separately coded and combined into single program. Each subprogram is called as function.
- Debugging, Testing and maintenance becomes easy when the program is divided into subprograms.
- Functions are used to avoid rewriting same code again and again in a program.
- Function provides code re-usability
- The length of the program is reduced.

Types of function:

Functions can be classified into two categories:

i) user defined function

ii) Built in function

i) Built in functions

- Built in functions are the functions that are already created and stored in python.
- These built in functions are always available for usage and accessed by a programmer. It cannot be modified.

Built in function	Description

>>>**max(3,4)** 4	# returns largest element
>>>**min(3,4)** 3	# returns smallest element
>>>**len("hello")** 5	#returns length of an object
>>>**range(2,8,1)** [2, 3, 4, 5, 6, 7]	#returns range of given values
>>>**round(7.8)** 8.0	#returns rounded integer of the given number
>>>**chr(5)** \x05'	#returns a character (a string) from an integer
>>>**float(5)** 5.0	#returns float number from string or integer
>>>**int(5.0)** 5	# returns integer from string or float
>>>**pow(3,5)** 243	#returns power of given number
>>>**type(5.6)** <type 'float'>	#returns data type of object to which it belongs
>>>**t=tuple([4,6.0,7])** (4, 6.0, 7)	# to create tuple of items from list
>>>**print("good morning")** Good morning	# displays the given object
>>>**input("enter name: ")** enter name : George	# reads and returns the given string

ii)**User Defined Functions:**

- User defined functions are the functions that programmers create for their requirement and use.
- These functions can then be **combined to form module** which can be used in other programs by importing them.
- Advantages of user defined functions:
  - Programmers working on large project can divide the workload by making different functions.
  - If repeated code occurs in a program, function can be used to include those codes and execute when needed by calling that function.

6.2 Function definition: (Sub program)

- def keyword is used to define a function.
- Give the function name after def keyword followed by parentheses in which arguments are given.
- End with colon (:)
- Inside the function add the program statements to be executed
- End with or without return statement

Syntax:

```
def  fun_name(Parameter1,Parameter2…Parameter n):
        statement1
        statement2…
        statement n
        return[expression]
```

Example:

```
def my_add(a,b):
      c=a+b
      return c
```

6.3Function Calling: (Main Function)

- Once we have defined a function, we can call it from another function, program or even the Python prompt.
- To call a function we **simply type the function name with appropriate arguments.**

Example:

```
x=5
y=4
my_add(x,y)
```

6.4 Flow of Execution:

- The order in which statements are executed is called the **flow of execution**
- Execution always begins at the first statement of the program.
- Statements are executed one at a time, in order, from top to bottom.
- Function definitions do not alter the flow of execution of the program, but remember that statements inside the function are not executed until the function is called.
- Function calls are like a bypass in the flow of execution. Instead of going to the next statement, the flow jumps to the first line of the called function, executes all the statements there, and then comes back to pick up where it left off.

Note: When you read a program, don't read from top to bottom. Instead, follow the flow of execution. This means that you will read the **def** statements as you are scanning from top to bottom, but you should skip the statements of the function definition until you reach a point where that function is called.

6.5 Function Prototypes:

i. Function without arguments and without return type
ii. Function with arguments and without return type
iii. Function without arguments and with return type
iv. Function with arguments and with return type

i) Function without arguments and without return type

- In this type no argument is passed through the function call and no output is return to main function
- The sub function will read the input values perform the operation and print the result in the same block

ii) Function with arguments and without return type

- Arguments are passed through the function call but output is not return to the main function

iii) Function without arguments and with return type

- In this type no argument is passed through the function call but output is return to the main function.

iv) Function with arguments and with return type

- In this type arguments are passed through the function call and output is return to the main function

Without Return Type

Without argument	With argument

def add():
    a=int(input("enter a"))
    b=int(input("enter b"))
    c=a+b
    print(c)
add()
	def add(a,b):
	    c=a+b
	    print(c)
	a=int(input("enter a"))
	b=int(input("enter b"))
	add(a,b)

OUTPUT: enter a 5 enter b 10 15	OUTPUT: enter a 5 enter b 10 15

With return type

Without argument	With argument

def add():
    a=int(input("enter a"))
    b=int(input("enter b"))
    c=a+b
    return c
c=add()
print(c)
	def add(a,b):
	    c=a+b
	    return c
	a=int(input("enter a"))
	b=int(input("enter b"))
	c=add(a,b)
	print(c)

OUTPUT: enter a 5 enter b 10 15	OUTPUT: enter a 5 enter b 10 15

6.6 Parameters And Arguments:

Parameters:

- Parameters are the value(s) provided in the parenthesis when we write function header.
- These are the values required by function to work.
- If there is more than one value required, all of them will be listed in parameter list separated by **comma.**
- Example: def my_add(a,b):

Arguments :

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

6.7 RETURN STATEMENT:

- The **return statement is used to exit a function** and go back to the place from where it was called.
- If the return statement has no arguments, then it will not return any values. But exits from function.

Syntax:

```
return[expression]
```

Example:

```
def my_add(a,b):
   c=a+b
   return c
x=5
y=4
print(my_add(x,y))
```

Output:

```
9
```

6.8 ARGUMENTS TYPES:

1. Required Arguments
2. Keyword Arguments
3. Default Arguments
4. Variable length Arguments

❖ **Required Arguments:** The number of arguments in the function call should match exactly with the function definition.

```
def my_details( name, age ):
  print("Name: ", name)
  print("Age ", age)
  return
my_details("george",56)
```

Output:

```
Name:  george
Age   56
```

❖ **Keyword Arguments:**

Python interpreter is able to use the keywords provided to match the values with parameters even though if they are arranged in out of order.

```
def my_details( name, age ):
  print("Name: ", name)
  print("Age ", age)
  return
my_details(age=56,name="george")
```

Output:

```
Name:  george
Age  56
```

❖ **Default Arguments:**

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

```
def my_details( name, age=40 ):
  print("Name: ", name)
  print("Age ", age)
  return
my_details(name="george")
```

Output:

```
Name:  george
Age  40
```

❖ **Variable length Arguments**

If we want to specify more arguments than specified while defining the function, variable length arguments are used. It is denoted by <u>* symbol</u> before parameter.

```
def my_details(*name ):
    print(*name)
my_details("rajan","rahul","micheal",
ärjun")
```

Output:

```
rajan rahul micheal ärjun
```

6.9 MODULES:

- **A module is a file containing Python definitions ,functions, statements and instructions.**
- <u>Standard library of Python</u> is extended as modules.
- **To use these modules in a program, <u>programmer needs to import the module.</u>**

- Once we import a module, we can reference or use to any of its functions or variables in our code.
  - There is large number of standard modules also available in python.
  - Standard modules can be imported the same way as we import our user-defined modules.
  - Every module contains many function.
  - *To access one of the function , you have to specify the name of the module and the name of the function separated by dot .* **This format is called dot notation.**

Syntax:

```
import module_name
module_name.function_name(variable)
```

Importing Builtin Module:	Importing User Defined Module:
import math x=math.sqrt(25) print(x)	import cal x=cal.add(5,4) print(x)

There are **four ways to import a module** in our program, they are

Import: It is simplest and most common way to use modules in our code. **Example:** import math x=math.pi print("The value of pi is", x) Output: The value of pi is 3.141592653589793	**from import** : It is used to get a specific function in the code instead of complete file. **Example:** from math import pi x=pi print("The value of pi is", x) Output: The value of pi is 3.141592653589793
import with renaming: We can import a module by renaming the module as our wish. Example: import math as m x=m.pi print("The value of pi is", x) Output: The value of pi is 3.141592653589793	**import all:** We can import all names(definitions) form a module using * Example: from math import * x=pi print("The value of pi is", x) Output: The value of pi is 3.141592653589793

Built-in python modules are,

1.math – mathematical functions:

some of the functions in math module is,

- math.ceil(x) - Return the ceiling of *x*, the smallest integer greater

than or equal to *x*

- math.floor(x) - Return the floor of *x*, the largest integer less than or equal to *x*.
- math.factorial(x) -Return *x* factorial. math.gcd(x,y)- Return the greatest common divisor of the integers *a* and *b*
- math.sqrt(x)- Return the square root of *x*
- math.log(x)- return the natural logarithm of *x*
- math.log10(x) – returns the base-10 logarithms
- math.log2(x) - Return the base-2 logarithm of *x*.
- math.sin(x) – returns sin of x radians
- math.cos(x)- returns cosine of x radians
- math.tan(x)-returns tangent of x radians
- math.pi - The mathematical constant π = 3.141592
- math.e – returns The mathematical constant e = 2.718281

2 .**random**-Generate pseudo-random numbers

- random.**randrange**(*stop*)
- random.**randrange**(*start*, *stop*[, *step*])
- random.**uniform**(*a*, *b*)
- -Return a random floating point number

ILLUSTRATIVE PROGRAMS

Program for SWAPPING(Exchanging)of values	**Output**
a = int(input("Enter a value ")) b = int(input("Enter b value ")) c = a a = b b = c print("a=",a,"b=",b,)	Enter a value 5 Enter b value 8 a=8 b=5
Program to find distance between two points	**Output**
import math x1=int(input("enter x1")) y1=int(input("enter y1")) x2=int(input("enter x2")) y2=int(input("enter y2")) distance =math.sqrt((x2-x1)**2)+((y2-y1)**2) print(distance)	enter x1 7 enter y1 6 enter x2 5 enter y2 7 2.5
Program to circulate n numbers	**Output:**
a=list(input("enter the list"))	enter the list '1234'

print(a) for i in range(1,len(a),1): print(a[i:]+a[:i])	['1', '2', '3', '4'] ['2', '3', '4', '1'] ['3', '4', '1', '2'] ['4', '1', '2', '3']

Part A:

1. What is interpreter?
2. What are the two modes of python?
3. List the features of python.
4. List the applications of python
5. List the difference between interactive and script mode
6. What is value in python?
7. What is identifier? and list the rules to name identifier.
8. What is keyword?
9. How to get data types in compile time and runtime?
10. What is indexing and types of indexing?
11. List out the operations on strings.
12. Explain slicing?
13. Explain below operations with the example
    (i)Concatenation (ii)Repetition
14. Give the difference between list and tuple
15. Differentiate Membership and Identity operators.
16. Compose the importance of indentation in python.
17. Evaluate the expression and find the result
    (a+b)*c/d
    a+b*c/d
18. Write a python program to print ‘n’ numbers.
19. Define function and its uses
20. Give the various data types in Python
21. Assess a program to assign and access variables.
22. Select and assign how an input operation was done in python.
23. Discover the difference between logical and bitwise operator.
24. Give the reserved words in Python.
25. Give the operator precedence in python.
26. Define the scope and lifetime of a variable in python.
27. Point out the uses of default arguments in python
28. Generalize the uses of python module.
29. Demonstrate how a function calls another function. Justify your answer.
30. List the syntax for function call with and without arguments.
31. Define recursive function.
32. What are the two parts of function definition? give the syntax.
33. Point out the difference between recursive and iterative technique.
34. Give the syntax for variable length arguments.

Part B

1. Explain in detail about various data types in Python with an example?
2. Explain the different types of operators in python with an example.
3. Discuss the need and importance of function in python.
4. Explain in details about function prototypes in python.
5. Discuss about the various type of arguments in python.
6. Explain the flow of execution in user defined function with example.
7. Illustrate a program to display different data types using variables and literal constants.
8. Show how an input and output function is performed in python with an example.
9. Explain in detail about the various operators in python with suitable examples.
10. Discuss the difference between tuples and list
11. Discuss the various operation that can be performed on a tuple and Lists (minimum 5)with an example program
12. What is membership and identity operators.
13. Write a program to perform addition, subtraction, multiplication, integer division, floor division and modulo division on two integer and float.
14. Write a program to convert degree Fahrenheit to Celsius
15. Discuss the need and importance of function in python.
16. Illustrate a program to exchange the value of two variables with temporary variables
17. Briefly discuss in detail about function prototyping in python. With suitable example program
18. Analyze the difference between local and global variables.
19. Explain with an example program to circulate the values of n variables
20. Analyze with a program to find out the distance between two points using python.
21. Do the Case study and perform the following operation in tuples i) Maxima minima iii)sum of two tuples iv) duplicate a tuple v)slicing operator vi) obtaining a list from a tuple vii) Compare two tuples viii)printing two tuples of different data types
22. Write a program to find out the square root of two numbers.