VELAMMAL ENGINEERING COLLEGE

Department of Computer Science and Engineering

19CS103T /Programming for Problem Solving using Python

UNIT-I BASICS OF PYTHON PROGRAMMING

Introduction-Features and History of Python - Python Interpreter- Interactive and Script mode–Python IDEs–Variables and Identifiers - Data types – Operators – Expressions – Statements – Operator precedence - Multiple assignments - Comments. Illustrative programs: Simple arithmetic programs, Temperature conversion.

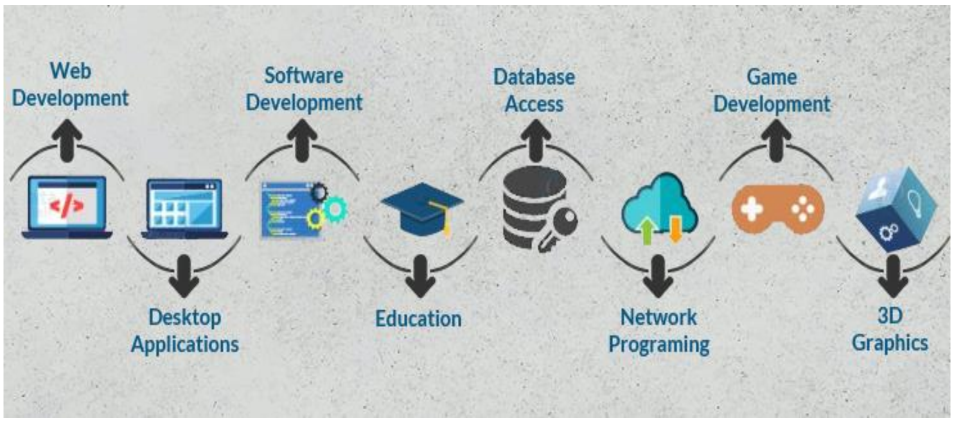
1. Python Introduction:

- ➢ Python is a High-Level , Interpreted, Interactive, Object Oriented , reliable and scripting language which can be used for a wide variety of text processing, system administration and internet-related tasks.
- ➢ Initially designed by Guido van Rossum in 1991 and developed by Python Software Foundation.
- ➢ Python has a vast library of modules to support integration of complex solutions from pre-built components.
- ➢ It was mainly developed for emphasis on code readability, and its syntax allows programmers to express concepts in fewer

2. Why Python?

- ❖ Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- ❖ Simple syntax similar to English language.
- ❖ Syntax that allows developers to write programs with fewer lines
- ❖ Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- ❖ Python can be treated in a procedural way, an object-oriented way or a functional way.

<u>Applications:</u>



3. FEATURES:

1. **Simple:** Python is simple and small language.
2. **Easy to learn:** Python is clearly defined and easily readable. It uses few keywords and clearly defined syntax.
3. **Versatile:** Python supports a development of wide range ofapplications.
4. **Free and open source:** Anyone can freely distribute it, read the source code, edit it, and even use the code to write a new program.
5. **High – level language:** The programmers don't have to worry about the low – level details like managing the memory used by the program.
6. **Interactive:** Programmers can easily interact with the interpreterdirectly at the python prompt to write their programs.
7. **Portable:** Python is portable, hence the programs behaves same on a wide variety of hardware platforms and has the same interface on all platforms.
8. **Object – oriented:** OO technique encapsulates data and functionalities within the objects.
9. **Interpreted:** Python is processed at run time by the interpreter. There is noneed of compiling the program before execution, hence it runs faster. It directly converts the source code into byte codes that is understandable by thecomputer.
10. **Dynamic:** Python executes dynamically. Python programs can be copied and used for flexible development applications.
11. **Extensible:** Anyone can add low-level modules to the python interpreter.
12. **Embeddable:** Programmers can embed the python within other programs to provide scripting capabilities to the user.
13. **Extensive libraries:** Python library has a huge librarythat is easily

portable across different programs.

14. **Easy maintenance:** Python Code is easy to maintain.
15. **Robust:** Programmers cannot manipulate the memorydirectly. Errors are raised as exceptions that can be catchand handled by the program code.
16. **Secure:** The python language environment is secured from tampering.
17. **Multithreaded:** Python can execute one process of a program simultaneously.
18. **Garbage Collection:** Python handles garbage collection of all python objects.
19. **GUI Programming and Databases** : It supports GUI application that can be created and ported to many libraries and window systems

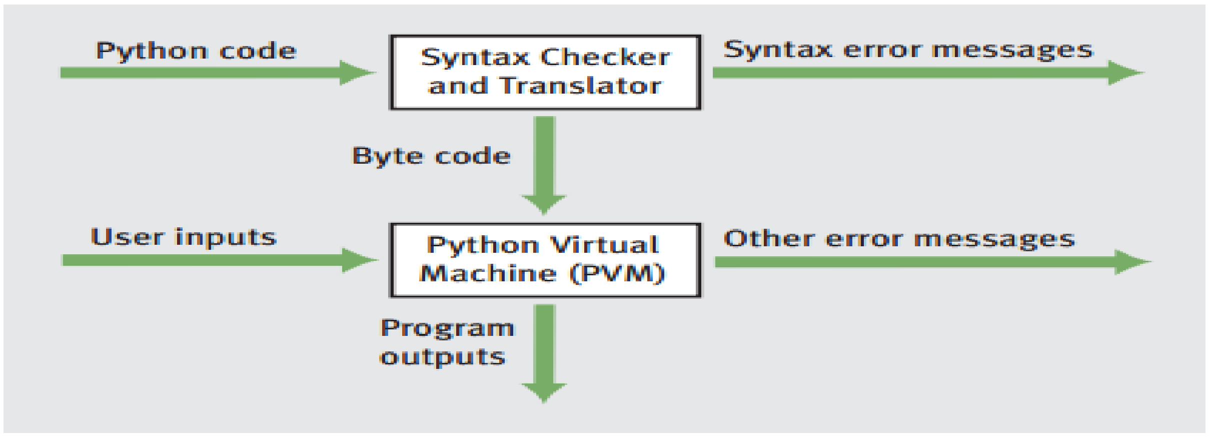
4. Python History and Versions:

- ✓ Python laid its foundation in the late 1980s.
- ✓ The implementation of Python was started in December 1989 by **Guido Van Rossum** at CWI inNetherland.
- ✓ In February 1991, **Guido Van Rossum** published the code (labelled version 0.9.0).
- ✓ In 1994, Python 1.0 : released with new features like lambda, map, filter.
- ✓ Python 2.0 : added with new features such as list comprehensions, garbage collection systems.
- ✓ On December 3, 2008, Python 3.0 (also called "Py3K") : designed to rectify thefundamental flaw of the language.
- ✓ *ABC programming language* is said to be the predecessor of Python language, which was capable ofException Handling and interfacing with the Amoeba Operating System.
- ✓ The following programming languages influence Python:
  - ABC language
  - Modula-33
- ✓ There is a fact behind choosing the name Python. **Guido van Rossum** was fond of reading the script of a popular BBC comedy series "**Monty Python's Flying Circus**".
- ✓ Van Rossum wanted to select a name which unique, sort, and little-bit mysterious. So he decided to select naming Python after the **"Monty Python's Flying Circus"** for their newlycreated programming language.
- ✓ Python programming language is being updated regularly with new features and supports.There are lots of update in Python versions, started from 1994 to current release 3.9

5. Python Interpreter:

- ✓ Python is called an interpreted language and programs are executed by an interpreter
- ✓ Python interpreter converts code into byte code and not into machine code, something that hardware can understand.
- ✓ It is into byte code and this byte code can't be understood by CPU.
- ✓ So an interpreter is needed called the python virtual machine which executes the byte codes.

Functions of Python Interpreter



<u>The Python interpreter performs following tasks to execute a Python program :</u>

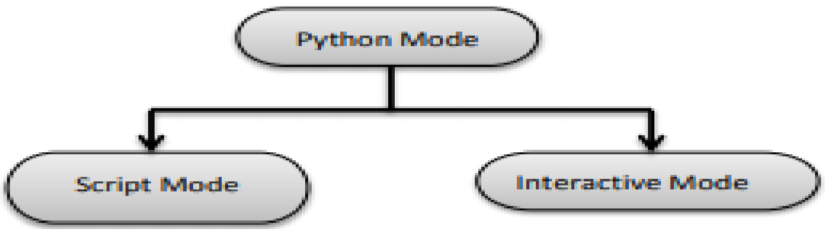
Step 1 : The interpreter reads a python code or instruction. Then it verifies that the instruction is well formatted, i.e. it checks the syntax of each line. If it encounters any error, it immediately halts the translation and shows an error message.

Step 2 : If there is no error, i.e. if the python instruction or code is well formatted then the interpreter translates it into its equivalent form in intermediate language called "Byte code". Thus, after successful execution of Python script or code, it is completely translated into Byte code.

Step 3 : Byte code is sent to the Python Virtual Machine(PVM).Here again the byte code is executed on PVM.If an error occurs during this execution then the execution is halted with an error message.

Python Interpreter uses 2 modes of Execution.

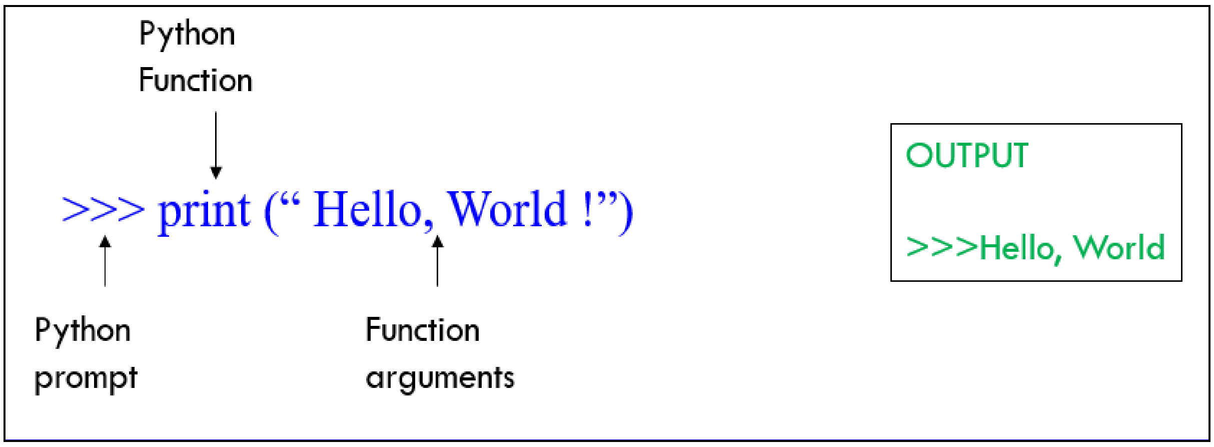
1) Interactive mode
2) Script mode



5.1 Interactive mode

- ✓ Interactive Mode, as the name suggests, allows us to interact with OS.
- ✓ It is command line shell
- ✓ Gives immediate feedback for each statement while running previously fed statements in active memory.
- ✓ > > > prompt indicates interactive mode

Example: Interactive mode in Python shell



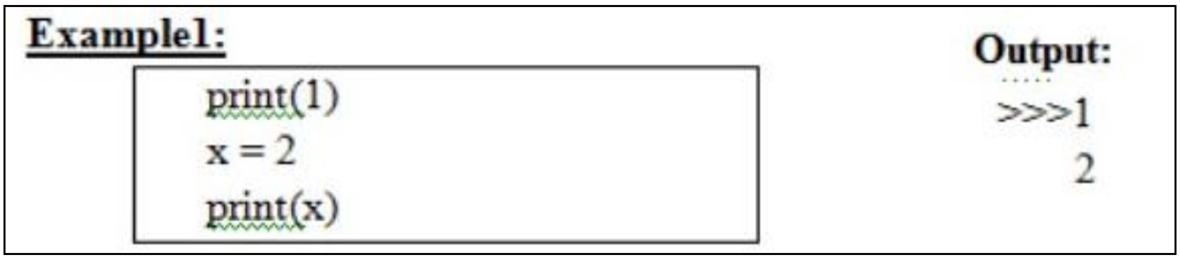
```
>>> print (" Hello, World !")
```

```
>>>Hello, World
```

5.2 Script mode

- In script mode, python program is created in a file and then interpreter is used to execute the file.
- Scripts can be saved to disk for future use. Python scripts have the extension **.py**
- Save the code with filename.py and run the interpreter in script mode to execute the script.

Ex1. Simple.py created in Script mode



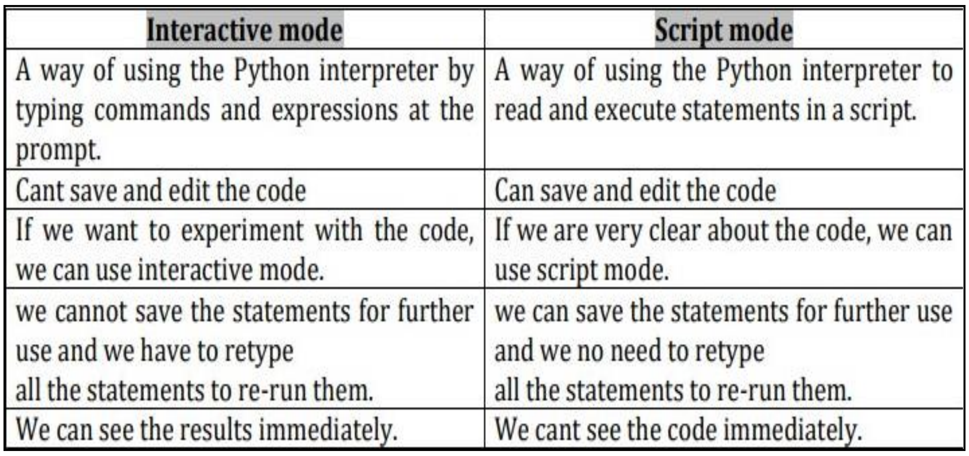
Example1:

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

Output:

```
>>>1
   2
```

Difference between Interactive mode & Script mode



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.

6. Python IDEs - Integrated Development Environment

- ❖ 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.
- ❖ Python IDLE is the standard most popular Python environment
- ❖ PyScripter IDE is also one of the Open source IDE

Integrated Development and Learning Environment (IDLE)

Efficient platform to write code and work interactively with python

Features of IDLE

1. Multi-window text editor with syntax highlighting.
2. 2 main window types, the Shell window and the Editor window
3. Auto completion with smart indentation.
4. On clicking IDLE icon Python shell window is opened.

5. Python Shell : Allows users to access different features of IDLE through its dropdown menu.
6. Used as basic calculator by entering mathematical expression on python prompt and output immediately
7. Python Editor window: Used to create python script files and save with .py extension

7. Variables and Identifiers

- Python is not “statically typed”.
- Do not need to declare variables before using them or declare their type.
- A variable is created the moment a value is assigned to it.
- Python variable is also known as an identifier and used to hold value which can be changed during program execution.
- A variable is only a name given to a memory location, all the operations done on the variable effects that memory location.
- Identifiers :Name used to identify a variable, function, class, module or any other object which helps in differentiating entities

Rules for creating variables in Python

- ❑ A variable name must start with a letter or the underscore character.
- ❑ A variable name cannot start with a number.
- ❑ A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _).
- ❑ Variable names are case-sensitive **(name, Name and NAME are three different variables).**
- ❑ The reserved words (keywords) cannot be used naming the variable.

Assigning Values to Variables

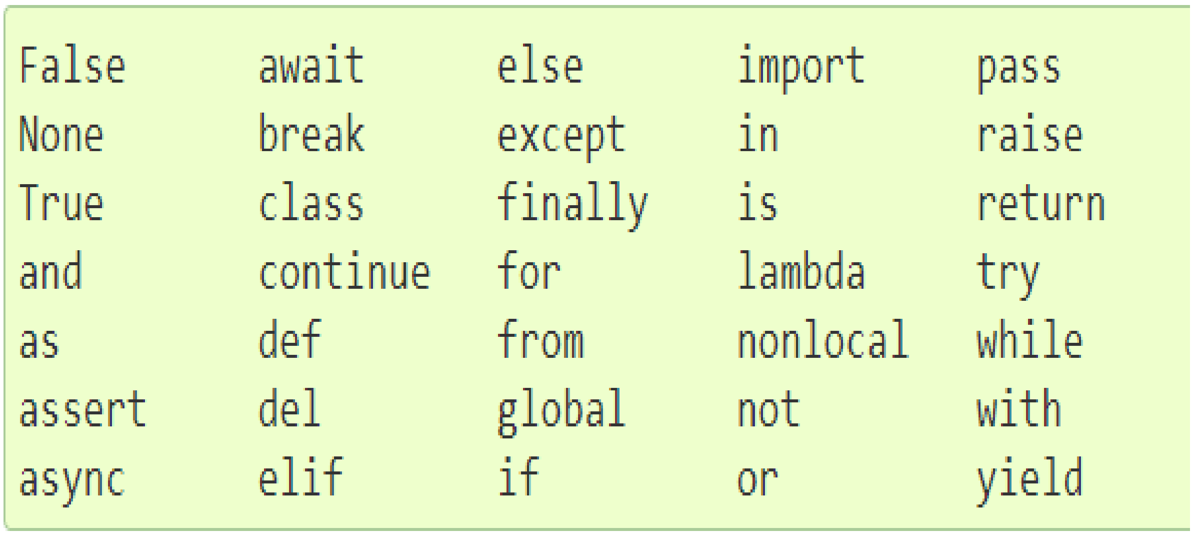
- ❖ Python variables do not need explicit declaration to reserve memory space.
- ❖ The declaration happens automatically when you assign a value to a variable.
- ❖ The equal sign (=) is used to assign values to variables.

- ❖ The operand to the left of the = operator is the name of the variable and the operand to the right of the = operator is the value stored in the variable.
  Eg: (i) a = 50
  The variable a refers to an integer object.
  (ii) a = 50,b = a
  The variable b refers to the same object that a points to because Python does not create another object.

Python Keywords

- ➢ Keywords are the reserved words in Python.
- ➢ Keyword cannot be used as a variable name, function name or any other identifier. They are used to define the syntax and structure of the Python language.
- ➢ In Python, keywords are case sensitive.
- ➢ All the keywords except True, False and None should be in lowercase.

List of Keywords



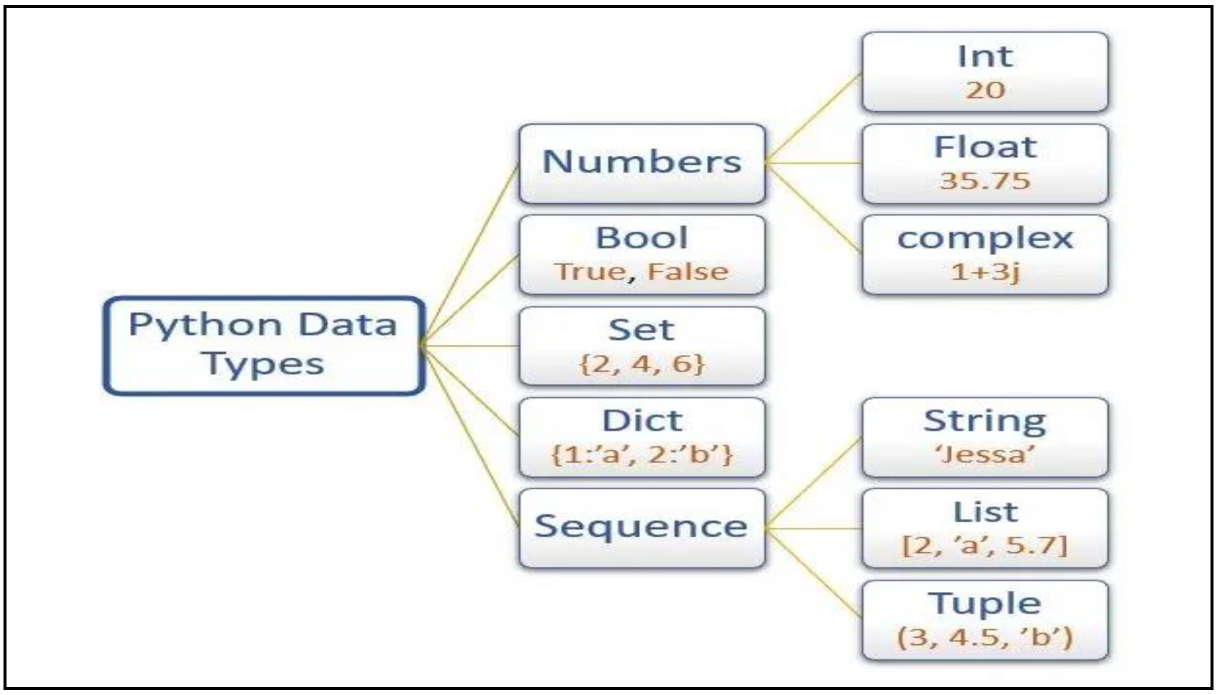
False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

8.Data Types

- ❖ A data type defines the type of data and allows languages to organise different kinds of data.

Eg.123 is an integer data while “hello” is a String type of data

List of Data types



The data types in Python are divided in two categories:

1. Immutable data types – Values cannot be changed.
2. Mutable data types – Values can be changed

❖ Immutable data types in Python are:

1. Numbers 4.Bool
2. String
3. Tuple

❖ Mutable data types in Python are:

5. List 7.Set
6. Dictionaries

Note - Python provides the **type()** function to know the data-type of the variable.

8.1. Immutable Data Types - Numbers

❖ Number stores numeric values.

❖ Numeric literal contains

1. digits (0-9)
2. optional sign character
3. possible decimal point

❖ Python creates Number objects when a number is assigned to a variable.

Python supports the following numeric data:

1. Int (signed integers)
2. Float
3. Complex

- **Int -** whole numbers with no fractional parts
  - Python has no restriction on the length of an integer.
  - eg. 10, 2, 29, - 20, -150
  - It is possible to represent an integer in binary , hexa-decimal or octal.

Binary literal	Hexa decimal literal	Octal literal

`>>> number=0b1010`
`>>> number`
`>>>10`
	`>>> number = 0xA0F`
	`>>> number`
	`>>>2575`
		`>>> number = 0o37`
		`>>> number`
		`>>>31`

- **Float -** Numbers with fractions or decimal points
  - consists of sign (+), (-)
  - It is accurate up to 15 decimal points.
  - eg. 1.9, 9.902, 15.2

```
Eg . >>> weight =67.2
     >>> type(weight)
Output : >>> <class 'float'>
```

- **complex -** A complex number contains an ordered pair, i.e., x + iy, where x and y denote the real and imaginary parts, respectively.
  - eg.2.14j, 2.0 + 2.3j

```
Eg . >>> x =4 + 5j
     >>> y= 2 - 2j
     >>> z= x + y
     >>> print (z)
Output : >>> 6 + 3 j
```

8.2. Immutable Data Types - String

A string is a sequence of characters

- ✓ Strings in Python are either enclosed with single quotes or double quotes.
- ✓ Triple double quotes “”” and triple single quotes ’’’ are used for creating multi-line strings in Python.
- ✓ A string is nothing but an array of characters so indexes can be used to access the characters

Example:

```str = 'beginnersbook'```	**Output**
```print(str)```	
```str2 = "Chaitanya"```	```>>>beginnersbook```
```print(str2)```	```>>>Chaitanya```
```# multi-line string```	```>>> Welcome to```
```str3 = """Welcome to```	```Beginnersbook.com```
```Beginnersbook.com"""```	```>>>This is a tech```
```print(str3)```	```blog```
```str4 = '''This is a tech```	```>>>e```
```blog'''```	
```print(str4)```	
```str[1]```	

8.3.Immutable Data Types - Tuple

- Tuple is an ordered collection of elements enclosed in round brackets and separated by commas.
- Tuple is a immutable ordered collection of elements enclosed in round brackets and separated by commas
- A tuple can have heterogeneous data items (i.e) a tuple can have string and list as data items as well.
- Indexed by integers
- Tuples is a read-only lists

```
Eg. >>> numtup=(1,3,5,9)
    >>> mixtup = (“Python  “, 3.6, 2)
    >>> numtup[1]=4    # error as tuple is immutable
```

- Tuple with only single element:
- Note: When a tuple has only one element, we must put a comma after the element, otherwise Python will not treat it as a tuple and consider as int
- Eg: # a tuple with single data item

  my_data = (99,)

8.4. Mutable data types – List:

- A list can be defined as an ordered collection of values of different data types
- Python lists are mutable type as elements can be modified after it created.
- Values in the list are called elements/items and are indexed ordered
- The items in the list are separated with the comma (,) and enclosed with the square brackets [ ].
- The list has the following characteristics:
  - The lists are ordered.
  - The element of the list can access by index.
  - The lists are the mutable type.
  - A list can store the number of various elements

Eg:

```
>>>L1 = ["John", 102, "USA"]
>>>L2 = [1, 2, 3, 4, 5, 6]
>>>print(type(L1))
>>> <class 'list'>
>>>print(L2[4])
>>> 5
```

8.5. Mutable data types - Dictionaries :

- Dictionary is an unordered collection of key-value pairs separated by a colon (:), enclosed in curly braces {}.
- It is a mutable data type in Python.
- Dictionary is defined into element Keys and values
- Left side of the colon (:) is the key and right side of the (:) is the value.
- Keys must be a single element
- Value can be any type such as list, tuple, integer, etc.
- Keys must be unique in dictionary, duplicate values are allowed.
- A dictionary is said to be empty if it has no key value pairs.
- An empty dictionary is denoted like this: {}.

Eg:

```
         Dict = {"Name": "Tom", "Age": 22}
Note - The keys Name and Age are the string that is
an immutable object
```

Accessing dictionary values using keys:

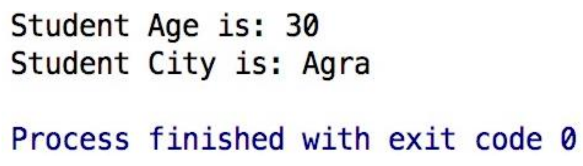
Eg:

mydict = {'StuName': 'Ajeet', 'StuAge': 30, 'StuCity': 'Agra'}

print("Student Age is:", mydict['StuAge'])

print("Student City is:", mydict['StuCity'])

Output:



```
Student Age is: 30
Student City is: Agra

Process finished with exit code 0
```

Change values in Dictionary:

To Update the values for the Existing key-value pairs corresponding key are used

Eg:

mydict = {'StuName': 'Ajeet', 'StuAge': 30, 'StuCity': 'Agra'}

print("Student Age before update is:", mydict['StuAge'])

print("Student City before update is:", mydict['StuCity'])

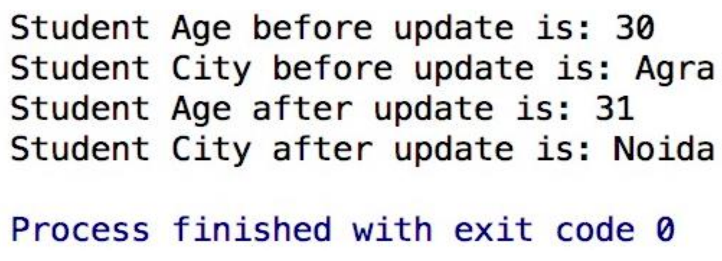
mydict['StuAge'] = 31

mydict['StuCity'] = 'Noida'

print("Student Age after update is:", mydict['StuAge'])

print("Student City after update is:", mydict['StuCity'])

Output:



```
Student Age before update is: 30
Student City before update is: Agra
Student Age after update is: 31
Student City after update is: Noida

Process finished with exit code 0
```

8.6. Mutable data types - Set :

- A set is an unordered collection with no duplicate elements
- Elements are inside curly braces { }.
- It is a mutable - Can easily add and remove items using methods like add(),update(),remove()

- 'set' object is not subscriptable-cannot access or change an element of a set using indexing or slicing
- Note:To create an empty set, use s=set() and not s={ } as the later creates empty dictionary

```
Eg.  >>> s= { 1, 2.0, “abc”}
     >>> print (s)
     >>>( [ 1, 2.0, “abc”] )
```

9. 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 + is called operator.

Types of Operator

1) Arithmetic Operators
2) Comparison (Relational) Operators
3) Assignment Operators
4) Logical Operators
5) Unary Operator
6) Bitwise Operators
7) Membership Operators
8) Identity Operators

9.1. Arithmetic Operators:

❑ Arithmetic operators are used to perform arithmetic operations between two operands.

Operator	Description	Example
+	Addition - Adds values on either side of the operator	a + b will give 30
-	Subtraction - Subtracts right hand operand from left hand operand	a - b will give -10
*	Multiplication - Multiplies values on either side of the operator	a * b will give 200
/	Division - Divides left hand operand by right hand operand	b / a will give 2
%	Modulus - Divides left hand operand by right hand operand and returns remainder	b % a will give 0
**	Exponent - Performs exponential (power) calculation on operators	a**b will give 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.	9//2 is equal to 4 and 9.0//2.0 is equal to 4.0

PROGRAM

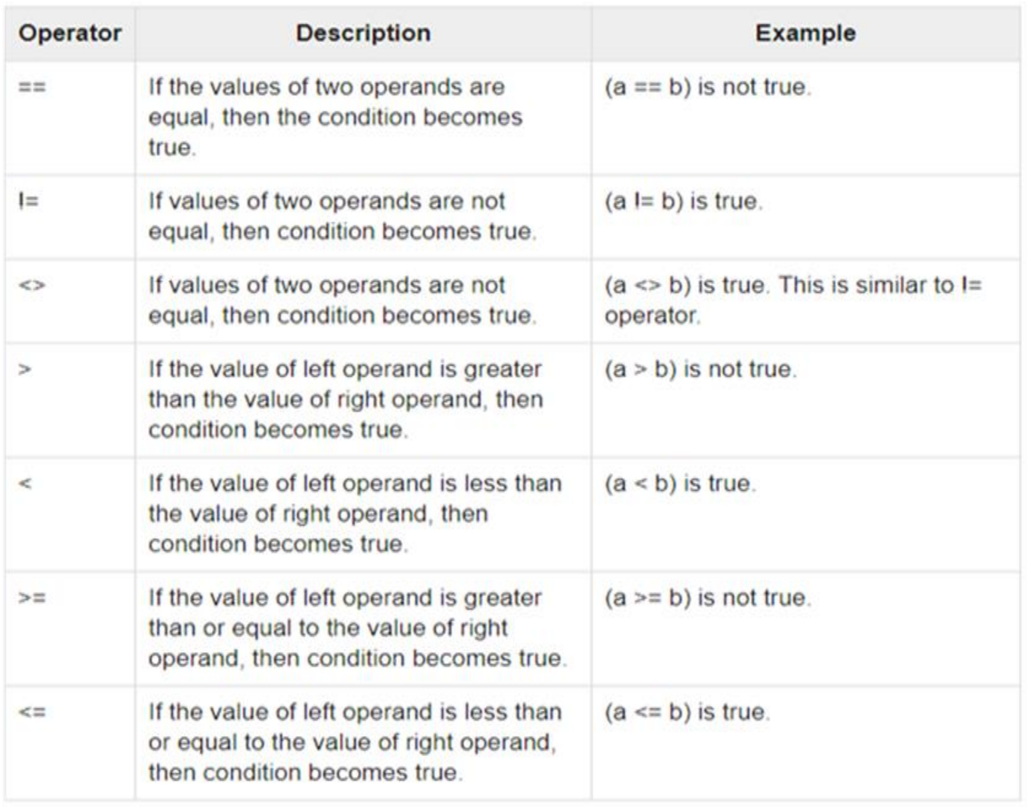
```
x,y = 10,5

print('x=',x)
print('y=',y)
print('x+y=',x+y)
print('x-y=',x-y)
print('x*y=',x*y)
print('x/y=',x/y)
print('x%y=',x%y)
print('x**y=',x**y)
print('x//y=',x//y)
```

OUTPUT

```
x= 10
y= 5
x+y= 15
x-y= 5
x*y= 50
x/y= 2.0
x%y= 0
x**y= 100000
x//y= 2
```

9.2 Comparison (Relational) Operators

❑Comparison operators are used to comparing the value of the two operands and returns Boolean true or false accordingly.



Operator	Description	Example
==	If the values of two operands are equal, then the condition becomes true.	(a == b) is not true.
!=	If values of two operands are not equal, then condition becomes true.	(a != b) is true.
<>	If values of two operands are not equal, then condition becomes true.	(a <> b) is true. This is similar to != operator.
>	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.

PROGRAM

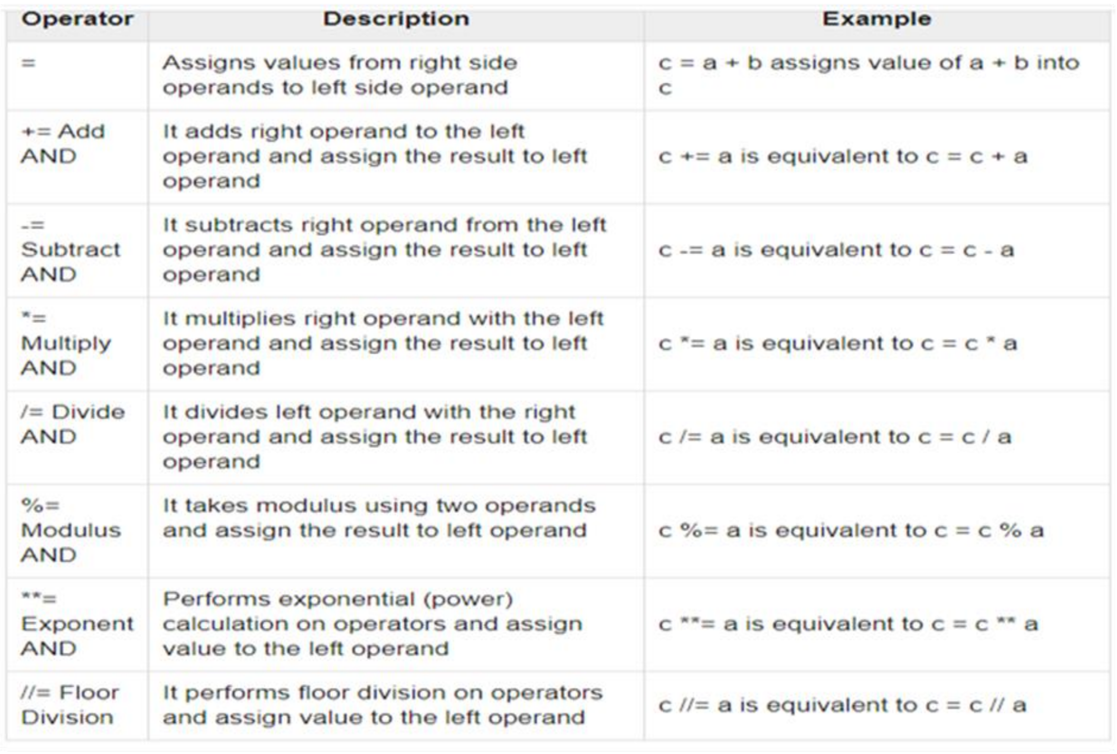
```
x,y = 10 , 5
print('x=',x)
print('y=',y)
print('x>y is',x>y)
print('x<y is',x<y)
print('x==y is',x==y)
print('x!=y is',x!=y)
print('x>=y is',x>=y)
print('x<=y is',x<=y)
```

OUTPUT

```
x= 10
y= 5
x>y is True
x<y is False
x==y is False
x!=y is True
x>=y is True
x<=y is False
```

9.3 Assignment Operators

❑The assignment operators are used to assign the value of the right expression to the left operand.

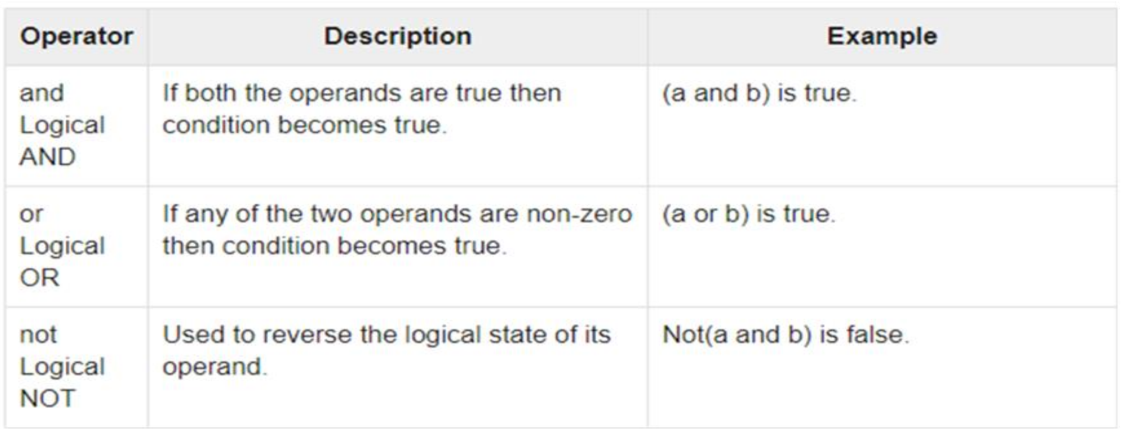


Operator	Description	Example
=	Assigns values from right side operands to left side operand	c = a + b assigns value of a + b into c
+= Add AND	It adds right operand to the left operand and assign the result to left operand	c += a is equivalent to c = c + a
-= Subtract AND	It subtracts right operand from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

PROGRAM	OUTPUT
`x,y,z = 2,8,5`	
`z=x+y`	
`print('z=x+y',z)`	`z=x+y    10`
`z+=x`	
`print('z+=x',z)`	`z+=x     12`
`z-=x`	
`print('z-=x',z)`	`z-=x     10`
`z*=x`	
`print('z*=x',z)`	`z*=x     20`
`z/=x`	
`print('z/=x',z)`	`z/=x     10.0`
`z**=x`	
`print('z**=x',z)`	`z**=x    100.0`
`z//=x`	
`print('z//=x',z)`	`z//=x    50.0`
`z%=x`	
`print('z%=x',z)`	`z%=x      0.0`

9.4 Logical operators

- The logical operators are used primarily in the expression evaluation to make a decision.



Operator	Description	Example
and Logical AND	If both the operands are true then condition becomes true.	(a and b) is true.
or Logical OR	If any of the two operands are non-zero then condition becomes true.	(a or b) is true.
not Logical NOT	Used to reverse the logical state of its operand.	Not(a and b) is false.

PROGRAM	OUTPUT
`x,y = 10,10`	
`print(x>y and x==y)`	`False`
`print(x>y or x==y)`	`True`
`print(x>=y and x==y)`	`True`
`x,y=10,5`	
`print(x>y and y>x)`	`False`
`print(x>y or y>x)`	`True`
`print(not(x>y))`	`False`

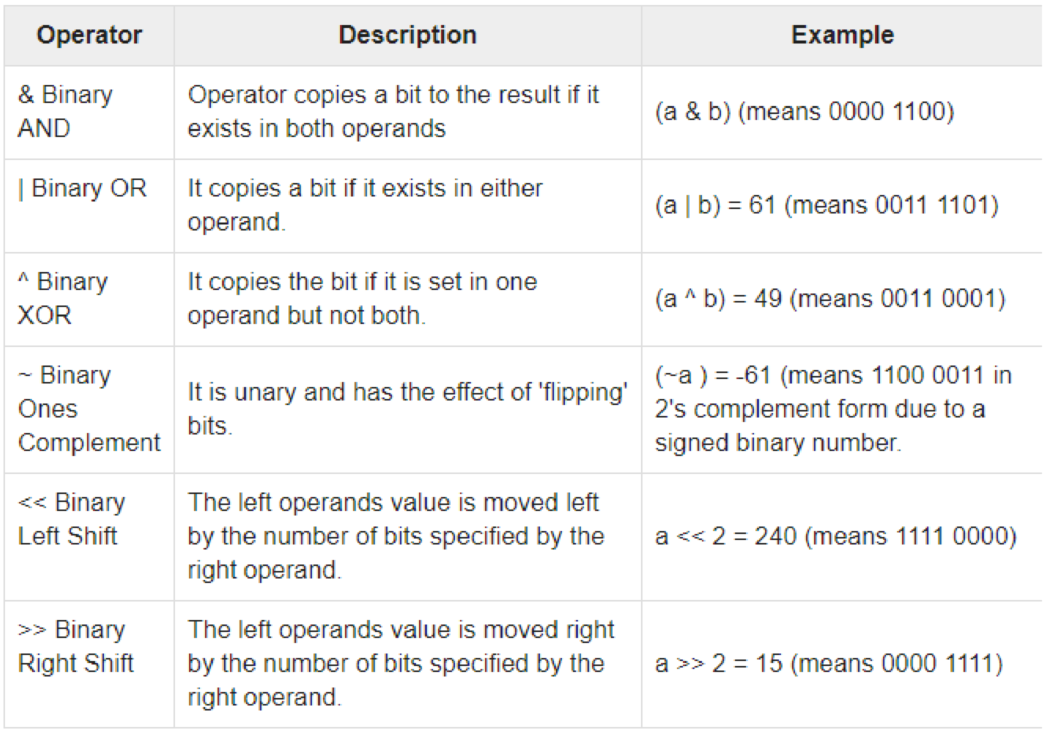
9.5. Unary operator:

- The unary operators require only one operand; they perform various operations such as incrementing/decrementing a value by one, negating an expression, or inverting the value of a boolean.
- unary + (plus) operator yields its numeric argument unchanged

- ➢ unary - (minus) operator yields the negation of its numeric argument
- ➢ The unary ~ (invert) operator yields the bitwise inversion of its integer argument. The bitwise inversion of x is defined as -(x+1). It only applies to integral numbers.
- ➢ The increment/decrement operators can be applied before (prefix) the operand but does not increment.
- ➢ The increment/decrement operators cannot be applied after (postfix) the operand. It results in Syntax error.

9.6. Bitwise operator:

The bitwise operators perform bit by bit operation on the values of the two operands.



Operator	Description	Example
& Binary AND	Operator copies a bit to the result if it exists in both operands	(a & b) (means 0000 1100)
| Binary OR	It copies a bit if it exists in either operand.	(a | b) = 61 (means 0011 1101)
^ Binary XOR	It copies the bit if it is set in one operand but not both.	(a ^ b) = 49 (means 0011 0001)
~ Binary Ones Complement	It is unary and has the effect of 'flipping' bits.	(~a) = -61 (means 1100 0011 in 2's complement form due to a signed binary number.
<< Binary Left Shift	The left operands value is moved left by the number of bits specified by the right operand.	a << 2 = 240 (means 1111 0000)
>> Binary Right Shift	The left operands value is moved right by the number of bits specified by the right operand.	a >> 2 = 15 (means 0000 1111)

Eg:

if a = 7

b = 6 then,

binary (a) = 0111

binary (b) = 0011

hence, a & b = 0011

a | b = 0111

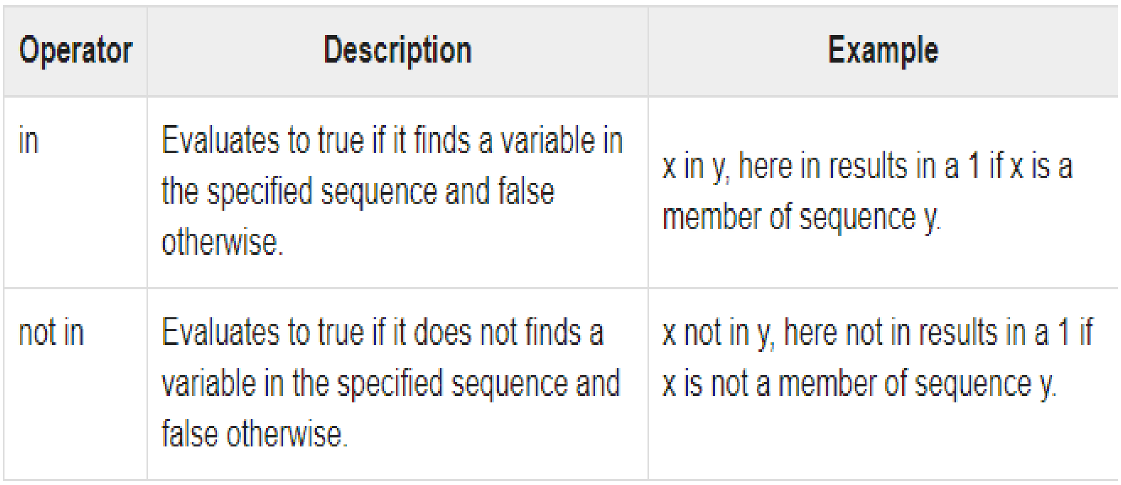
a ^ b = 0100

~ a = 1000

PROGRAM	**OUTPUT**
`x,y = 10,12`	
`print("Bitwise AND x & y=",x&y, "in binary=",bin(x&y))`	`Bitwise AND x & y= 8 in binary= 0b1000`
`print("Bitwise OR x | y=",bin(x|y))`	`Bitwise OR x | y= 0b1110`
`print("Bitwise EXOR x ^ y=",bin(x^y))`	`Bitwise EXOR x ^ y= 0b110`
`print("Bitwise one's complement of x=",bin(~x))`	`Bitwise one's complement of x= -0b1011`
`print("Left shift by 2 is ",bin(0b110001<<2))`	`Left shift by 2 is 0b11000100`
`print("Left shift by 2 is ",0b110001<<2)`	`Left shift by 2 is  196`
`print("right shift by 2 is ",bin(0b11000000>>2))`	`right shift by 2 is 0b110000`

9.7. Membership Operators:

- Python membership operators are used to check the membership of value inside a Python data structure.
- If the value is present in the data structure, then the resulting value is true otherwise it returns false.

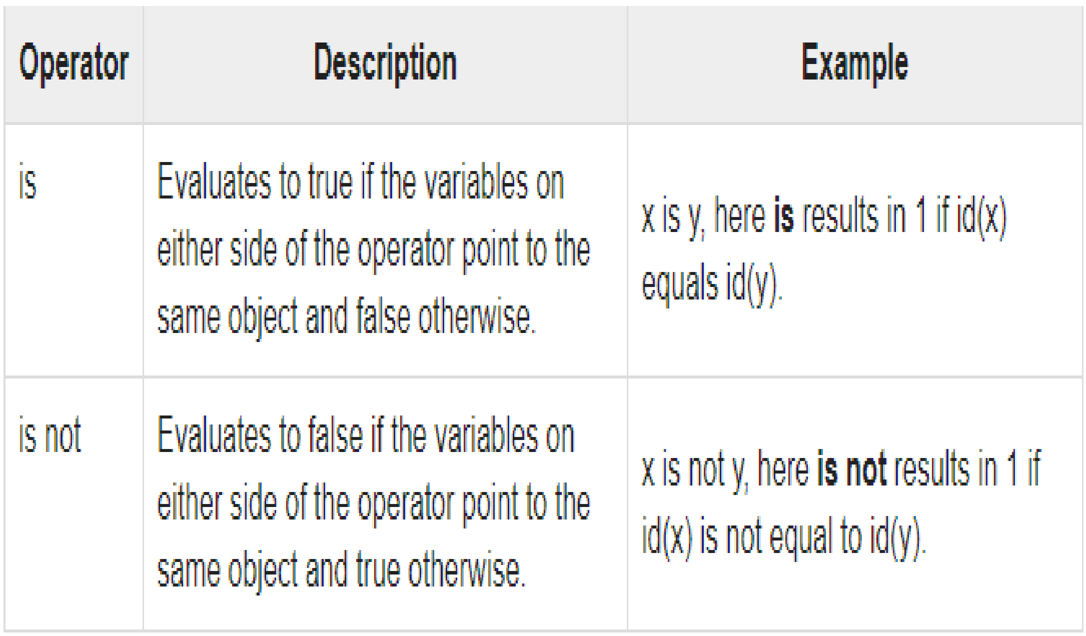


Operator	Description	Example
in	Evaluates to true if it finds a variable in the specified sequence and false otherwise.	x in y, here in results in a 1 if x is a member of sequence y.
not in	Evaluates to true if it does not finds a variable in the specified sequence and false otherwise.	x not in y, here not in results in a 1 if x is not a member of sequence y.

PROGRAM	OUTPUT
`x={"python","java","c language","php","vb"}`	
`print("java" in x)`	True
`print("c++" in x)`	False
`print("c sharp" not in x)`	True
`print(1 in x)`	False
`x={"python":1,"java":2,"clanguage":3,"php":4,"vb":5}`	
`print("java" in x)`	True
`print("c++" in x)`	False
`print("c sharp" not in x)`	True
`print(1 in x)`	False

9.8. Identity operators:

➢ Identity operators compare the memory locations of two objects.

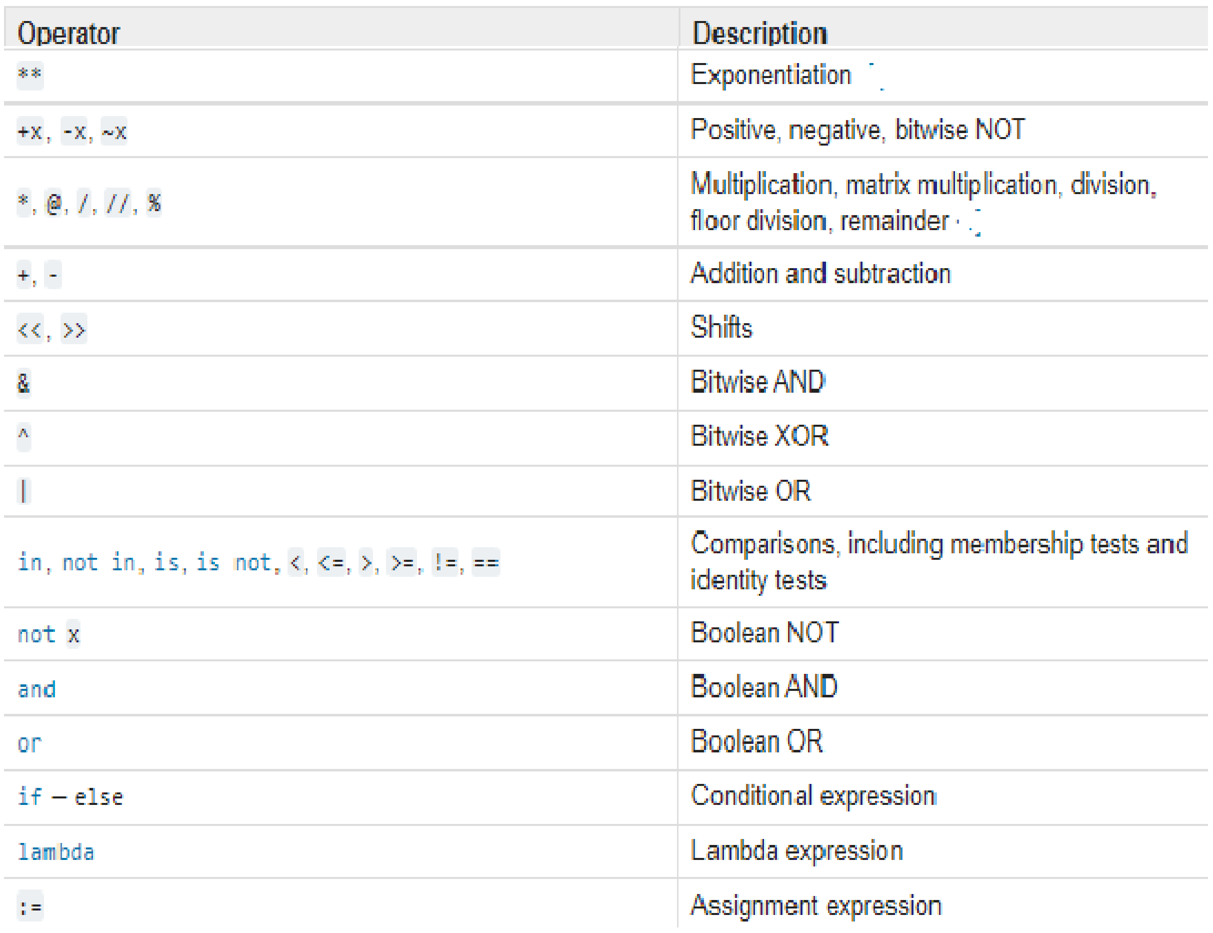


Operator	Description	Example
is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.	x is y, here **is** results in 1 if id(x) equals id(y).
is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.	x is not y, here **is not** results in 1 if id(x) is not equal to id(y).

PROGRAM	**OUTPUT**
`x,y,z=15,15,10`	
`print(x is y)`	`True`
`print(x is z)`	`False`
`print(y is z)`	`False`
`print(x is not y)`	`False`
`print(x is not z)`	`True`
`x,y,z=[1,2,3,4,5], [1,2,3,4,5], [6,7,8,9]`	
`print(x is y)`	`False`
`print(x is z)`	`False`
`print(y is z)`	`False`
`print(x is not y)`	`True`
`print(x is not z)`	`True`

10. Operator precedence

- ❖ Determines which operator needs to be executed first
- ❖ Precedence is necessary to avoid ambiguity in values
- ❖ In the Operator precedence chart
  - Operators in row have same precedence and have associativity from Left to Right except for exponentiation, which groups from right to left



Operator	Description
`**`	Exponentiation
`+x`, `-x`, `~x`	Positive, negative, bitwise NOT
`*`, `@`, `/`, `//`, `%`	Multiplication, matrix multiplication, division, floor division, remainder
`+`, `-`	Addition and subtraction
`<<`, `>>`	Shifts
`&`	Bitwise AND
`^`	Bitwise XOR
`|`	Bitwise OR
`in`, `not in`, `is`, `is not`, `<`, `<=`, `>`, `>=`, `!=`, `==`	Comparisons, including membership tests and identity tests
`not x`	Boolean NOT
`and`	Boolean AND
`or`	Boolean OR
`if – else`	Conditional expression
`lambda`	Lambda expression
`:=`	Assignment expression

11. Expressions :

❖ An expression is a combination of operators, constants and variables.
❖ An expression may consist of one or more operands, and zero or more operators to produce a value.

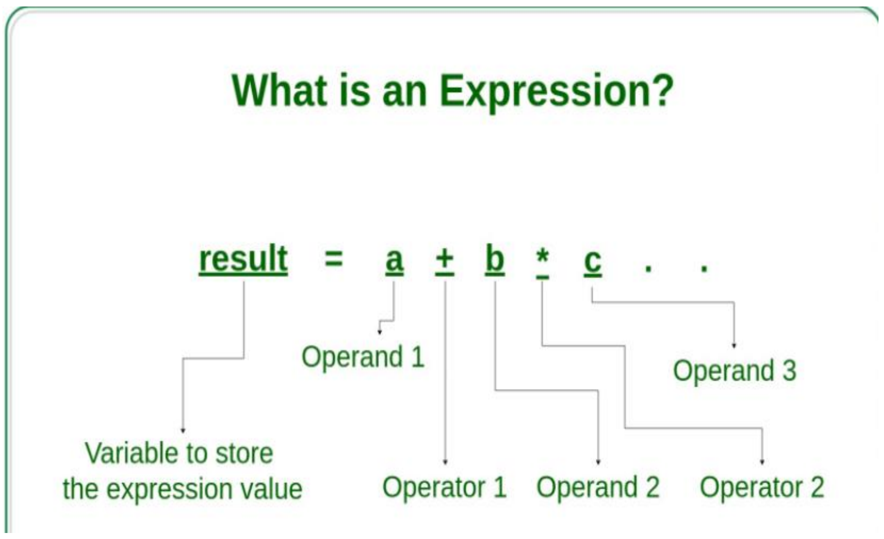
Eg:

a = 2
b = 3

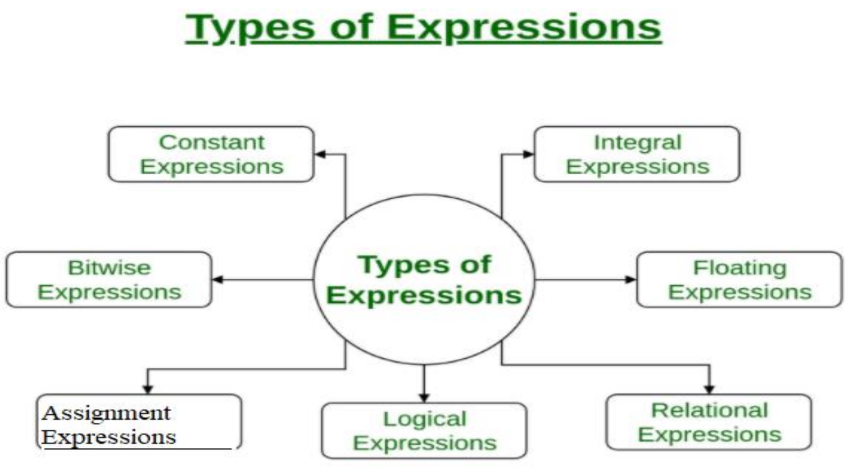
```
c = a + b
print c
d = a * b
print d
```

The output is:

```
5
6
```



Types of Expressions



<u>Types</u>

1) **Constant expressions:** Constant Expressions consists of only constant values. A constant value is one that doesn't change.

   Examples: 5, 10 + 5 / 6.0, 'x'

2) **Integral expressions:** Integral Expressions are those which produce integer results after implementing all the automatic and explicit type conversions.

   Examples: x, x * y, x + int(5.0)where x and y are integer variables.

3) **Floating expressions:** Float Expressions are which produce floating point results after implementing all the automatic and explicit type conversions.

   Examples: x + y, 10.75where x and y are floating point variables.

4) **Relational expressions:** Relational Expressions yield results of type bool which takes a value true or false. When arithmetic expressions are used on either side of a relational operator, they will be evaluated first and then the results compared. Relational expressions are also known as Boolean expressions.

   Examples: x <= y, x + y > 2

5) **Logical expressions**: Logical Expressions combine two or more relational expressions and produces bool type results.

   Examples: x > y && x == 10, x == 10 || y == 5

6) **Assignment expressions**: Assignment Expressions assigns a value to the variables.

   Examples: c=15, d=a+b+c

7) **Bitwise expressions**: Bitwise Expressions are used to manipulate data at bit level. They are basically used for testing or shifting bits.

   Examples: x << 3shifts three bit position to left

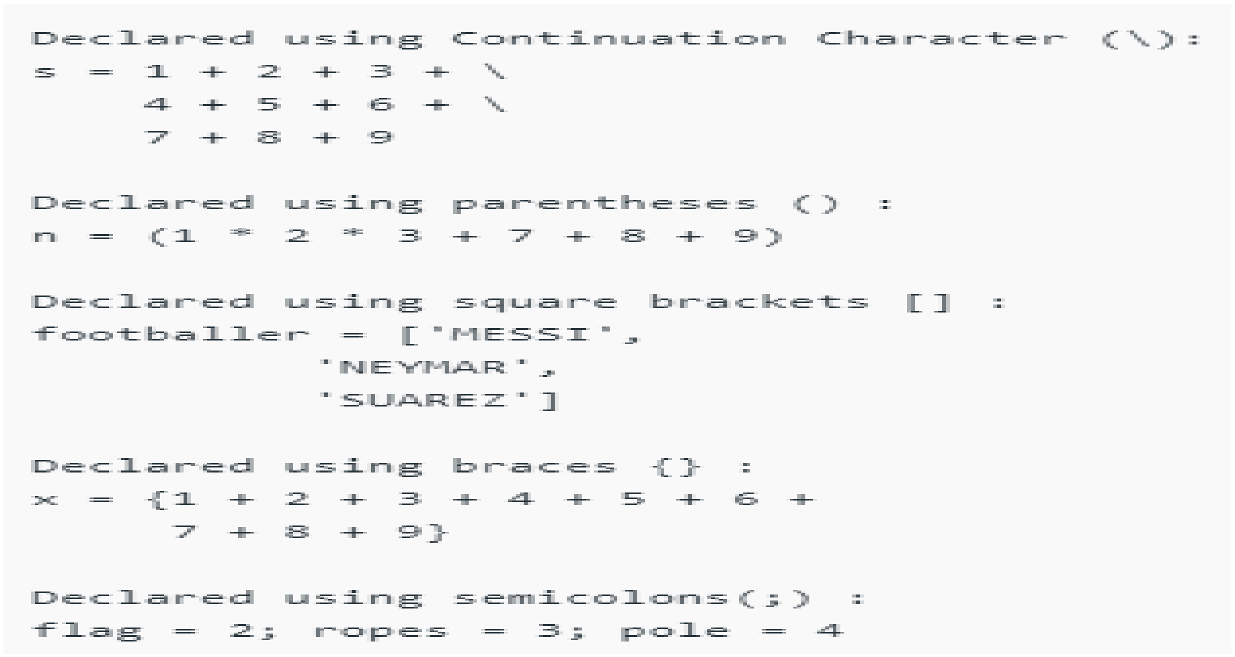
   y >> 1shifts one bit position to right. Shift operators are often used for multiplication and division by powers of two.

12. Statements :

- ❑ Instructions written in the source code for execution are called statements.
- ❑ There are different types of statements in the Python programming language like Assignment statement, Conditional statement, Looping statements etc. These all help the user to get the required output. For example, n = 50 is an assignment statement.

Multi-Line Statements:

1. Statements in Python can be extended to one or more lines using parentheses (), braces {}, square brackets [], semi-colon (;), continuation character slash (\).When the programmer needs to do long calculations and cannot fit his statements into one line, one can make use of these characters.



```
Declared using Continuation Character (\):
s = 1 + 2 + 3 + \
    4 + 5 + 6 + \
    7 + 8 + 9

Declared using parentheses () :
n = (1 * 2 * 3 + 7 + 8 + 9)

Declared using square brackets [] :
footballer = ['MESSI',
             'NEYMAR',
             'SUAREZ']

Declared using braces {} :
x = {1 + 2 + 3 + 4 + 5 + 6 +
     7 + 8 + 9}

Declared using semicolons(;) :
flag = 2; ropes = 3; pole = 4
```

Indentation:

- ➢ One of the distinctive features of Python is its use of indentation to highlight the blocks of code.
- ➢ Whitespace is used for indentation in Python.

- All statements with the same distance to the right belong to the same block of code.
- If a block has to be more deeply nested, it is simply indented further to the right.

Eg:

```
# Python program showing indentation
site = 'google'
if site = = 'google':
    print('Logging on to google...')
else:
    print('retype the URL.')
print('Welcome!')
```

Output:

```
Logging on to google...
Welcome!
```

Note : The lines print('Logging on to google …') and print('retype the URL.') are two separate code blocks. The two blocks of code in example if-statement are both indented four spaces. The final print('Welcome!') is not indented, and so it does not belong to the else-block.

Comments:

- ✓ Comments are for programmers to better understand a program.
- ✓ Python Interpreter ignores comments are not executed.
- ✓ They describe what is going on inside a program, so that a person looking at the source code does not have a hard time figuring it out.
- ✓ Hash (#) symbol is used to start writing a comment.
- ✓ It extends up to the newline character.

Eg:

#This is a comment

#print out Hello

print('Hello')

Multi-line comments:

- ✓ To extend comments up to multiple lines.
  - ➢ One way is to use the hash(#) symbol at the beginning of each line.
  - ➢ Another way of doing this is to use triple single or triple double quotes, either ''' or """. Unless they are not docstrings, they do not generate any extra code.

Eg:

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

13. Multiple assignment

- ➢ Single values can be assigned to more than one variables simultaneously
  - eg. Sum = flag = a = b =0
- ➢ Assign multiple values to multiple variables simultaneously
  - eg. Sum , a , b, mesg = 0, 3, 5, “result”
- ➢ Ex. of tuple packing
  - t = 12345, 54321, 'hello!’
    values 12345, 54321 and 'hello!' are packed together in a tuple

- ➢ Sequence unpacking and works for any sequence on the right-hand side.
- ➢ Sequence unpacking requires that there are as many variables on the left side of the equals sign as there are elements in the sequence.

```
>>> x, y, z = t
```

Note that multiple assignment is really just a combination of tuple packing and sequence unpacking.

- ➢ Definition of assignment implies that overlaps between the left-hand side and the right-hand side are ‘simultaneous’
  - ➢ Ex. a, b = b, a swaps two variables
- ➢ But overlaps *within* the collection of assigned-to variables occur left-to-right
  - ➢ Ex.

```
x = [0, 5]
i = 0
i, x[i] = 1, 2 # i is updated, then x[i] is updated
```

```
print(x)
```
Output – [0,2]

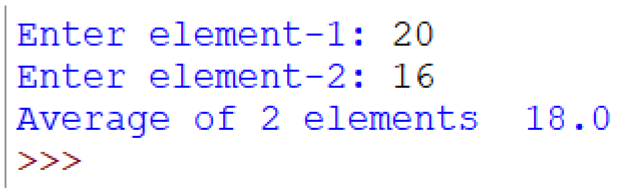
14. Illustrative programs

Simple arithmetic programs and Temperature Conversion

1. Write a program to find average of two numbers in python.

```
number1=int(input("Enter element-1: "))
number2=int(input("Enter element-2: "))
avg=(number1+number2)/2
print("Average of 2 elements ",round(avg,2))
```

OUTPUT:

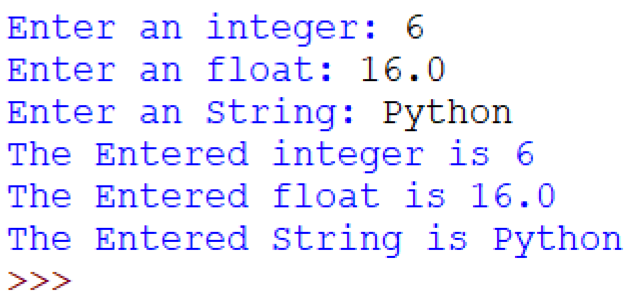


```
Enter element-1: 20
Enter element-2: 16
Average of 2 elements  18.0
>>>
```

2. Write a program to read and print the values of different data types.

```
a=int(input("Enter an integer: "))
b=float(input("Enter an float: "))
c=str(input("Enter an String: "))
print("The Entered integer is",a)
print("The Entered float is",b)
print("The Entered String is",c)
```

OUTPUT:



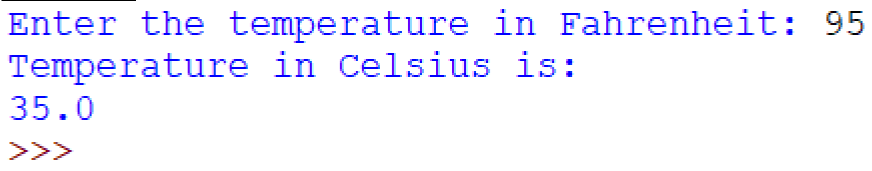
```
Enter an integer: 6
Enter an float: 16.0
Enter an String: Python
The Entered integer is 6
The Entered float is 16.0
The Entered String is Python
>>>
```

3. Write a program to convert temperature from Fahrenheit to Celsius.

```
f = int(input("Enter the temperature in Fahrenheit: "))
```

```
c = ((f-32)*5)/9
print("Temperature in Celsius is: ");
print(round(c,2));
```

OUTPUT:

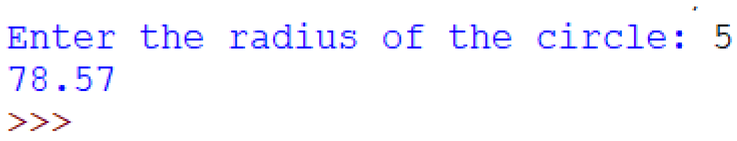


```
Enter the temperature in Fahrenheit: 95
Temperature in Celsius is:
35.0
>>>
```

4. Write a program to find area of circle.

```
pi = 22/7
r = float(input("Enter the radius of the circle: "))
area = pi * r * r
print("%.2f" %area)
```

OUTPUT:

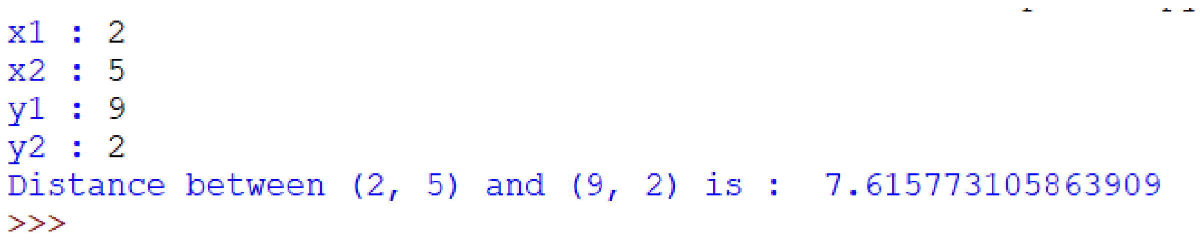


```
Enter the radius of the circle: 5
78.57
>>>
```

5. Write a program to find the distance of two points.

```
# Points are (x1,x2), (y1,y2)
x1 = int(input("x1 : "))
x2 = int(input("x2 : "))
y1 = int(input("y1 : "))
y2 = int(input("y2 : "))
dist = ((((x2 - x1 )**2) + ((y2-y1)**2) )**0.5)
print("Distance between",(x1,x2),"and",(y1,y2),"is :
",dist)
```

OUTPUT:



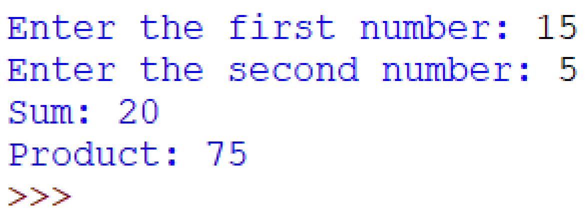
```
x1 : 2
x2 : 5
y1 : 9
y2 : 2
Distance between (2, 5) and (9, 2) is :  7.615773105863909
>>>
```

6. Write a Python program to accept two numbers, compute the sum, product and print the result.

```
a = int(input("Enter the first number: "))
b = int(input("Enter the second number: "))
add = a + b
```

```
pdt = a*b
print("Sum:", add)
print("Product:", pdt)
```

OUTPUT:

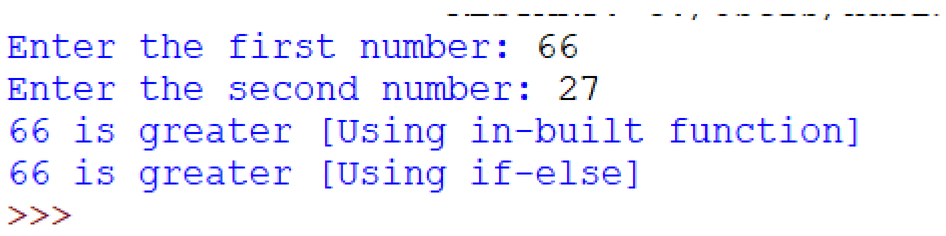


```
Enter the first number: 15
Enter the second number: 5
Sum: 20
Product: 75
>>>
```

7. Write a Python program to accept two numbers, find the greatest and print the result.

```
num1 = int(input("Enter the first number: "))
num2 = int(input("Enter the second number: "))
# Using max() function
print(max(num1, num2), "is greater")
# Using if-else statements
if(num1 >= num2):
  print(num1, "is greater")
else:
  print(num2, "is greater")
```

OUTPUT:



```
Enter the first number: 66
Enter the second number: 27
66 is greater [Using in-built function]
66 is greater [Using if-else]
>>>
```

8. Write a Python program to exchange the value of two variables.

```
x = input('Enter the value of x: ')
y = input('Enter the value of y: ')
temp = x
x = y
y = temp
print('The value of x after swapping: ',x)
print('The value of y after swapping: ',y)
```

OUTPUT:

```
Enter the value of x: 9
Enter the value of y: 5
The value of x after swapping:  5
The value of y after swapping:  9
>>>
```

9. Write a Python program to perform addition, subtraction, division, multiplication on two floating point numbers.

```
num1 = float(input('Enter the first number: '))
num2 = float(input('Enter the second number: '))
add = num1 + num2
sub = num1 - num2
mul = num1 * num2
div = num1 / num2
print('{0} + {1} = {2}'.format(num1, num2, add))
print('{0} - {1} = {2}'.format(num1, num2, sub))
print('{0} * {1} = {2}'.format(num1, num2, mul))
print('{0} / {1} = {2}'.format(num1, num2, div))
```

OUTPUT:

```
Enter the first number: 6
Enter the second number: 7
6.0 + 7.0 = 13.0
6.0 - 7.0 = -1.0
6.0 * 7.0 = 42.0
6.0 / 7.0 = 0.8571428571428571
>>>
```

10. Write a program to calculate simple interest and compound interest.

```
P = float(input('Enter principal amount : '))
R = float(input('Enter rate of interest : '))
T = float(input('Enter time : '))
N = int(input('Enter number of times interest is computed
annually : '))
SI = P + (P*R*T)/100
print('Simple interest : %.2f' % SI)
CI = P * pow(1 + (R/N),(N*T))
print('Compound interest : %.2f' % CI)
```

OUTPUT:

```
Enter principal amount : 1200
Enter rate of interest : 5
Enter time : 3
Enter number of times interest is computed
annually : 2
Simple interest : 1380.00
Compound interest : 2205918.75
```

11. Write a program to calculate salary of an employee given his basic pay, HRA=10% of Basic, TA=5% of basic. Calculate the salary of the employee.

```
basic_pay = float(input('Enter basic salary : '))
HRA = 0.1 * basic_pay
TA = 0.05 * basic_pay
salary = basic_pay + HRA + TA
print('Total Salary : %f' % salary)
```

OUTPUT:

```
Enter basic salary : 1000
Total Salary : 1150.000000
```