Introduction:-

Python is a high-level, interpreted, interactive and object-oriented scripting language. Designed to be highly readable. It uses English keywords frequently where as other languages use punctuation.

History of Python

Python is a fairly old language created by Guido Van Rossum. The design began in the late 1980s and was first released in February 1991 and the current version is 3.5.3 released in january 2017.

Why Python was created?

In late 1980s, Guido Van Rossum was working on the Amoeba distributed operating system group. He wanted to use an interpreted language like ABC that could access the Amoeba system calls. So, he decided to create a language that was extensible. This led to a design of new language which was later named Python.

Why the name Python?

It wasn't named after a dangerous snake. Rossum was fan of a comedy series from late seventies. The name "Python" was adopted from the same series "Monty Python's Flying Circus".

Features of Python Programming

1. **A simple language which is easier to learn**
   Python has a very simple and elegant syntax. It's much easier to read and write Python programs compared to other languages like: C++, Java, C#.
2. **Free and open-source**
   You can freely use and distribute Python, even for commercial use. Not only can you use and distribute softwares written in it, you can even make changes to the Python's source code.
3. **Portability**
   You can move Python programs from one platform to another, and run it without any changes. It runs seamlessly on almost all platforms including Windows, Mac OS X and Linux.
4. **Extensible and Embeddable**
   Suppose an application requires high performance. You can easily combine pieces of C/C++ or other languages with Python code. This will give your application high performance as well as scripting capabilities which other languages may not provide out of the box.
5. **A high-level, interpreted language**
   Unlike C/C++, you don't have to worry about difficult tasks like memory management, garbage collection and so on. Likewise, when you run Python code, it automatically converts your code to the language your computer understands. You don't need to worry about any lower-level operations.
6. **Large standard libraries to solve common tasks**
   Python has a number of standard libraries which makes life of a programmer much easier since you don't have to write all the code yourself.

7. **Object-oriented**
   Everything in Python is an object. Object oriented programming (OOP) helps you solve a complex problem very easily With OOP, you are able to divide these complex problems into smaller sets by creating objects.

Applications of Python

1. **Web Applications**
   You can create scalable Web Apps using frameworks and CMS (Content Management System) that are built on Python. Sites like Mozilla,Instagram are written in Python.
2. **Scientific and Numeric Computing**
   There are numerous libraries available in Python for scientific and numeric computing. There are specific libraries like: EarthPy for earth science, AstroPy for Astronomy and so on.
3. **Creating software Prototypes**
   Python is a great language for creating prototypes. For example: You can use Pygame (library for creating games) to create your game's prototype first. If you like the prototype, you can use language like C++ to create the actual game.

Definitions:

What is a program: A **program** is a sequence of instructions that specifies how to perform a computation. The computation might be something mathematical, such as solving a system of equations or finding the roots of a polynomial, but it can also be a symbolic computation, such as searching and replacing text in a document or (strangely enough) compiling a program.

Debugging: Programming errors are called bugs and the process of tracking them down is called debugging. Three kinds of errors can occur in a program: syntax errors, runtime errors, and semantic errors.

Problem solving: The process of formulating a problem, finding a solution, and expressing the solution.

Portability: A property of a program that can run on more than one kind of computer.

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

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

Script: A program stored in a file (usually one that will be interpreted).

algorithm: A general process for solving a category of problems.

bug: An error in a program.

syntax: The structure of a program.

Python Variables

A variable is a location in memory used to store some data (value). They are given unique names to differentiate between different memory locations.

- No need to declare a variable before using it.
- In Python, Simply assign a value to a variable and it will exist.
- No need to declare the type of the variable.

```
#!/usr/bin/python

counter = 100          # An integer assignment

miles   = 1000.0       # A floating point

name    = "John"       # A string

print counter

print miles

print name
```

Here, 100, 1000.0 and "John" are the values assigned to *counter*, *miles*, and *name* variables, respectively. This produces the following result –

Output:
100

1000.0

John

Multiple assignments

Python allows you to assign a single value to several variables simultaneously. For example –
a = b = c = 1

Here, an integer object is created with the value 1, and all three variables are assigned to the same memory location. You can also assign multiple objects to multiple variables. For example –

Example:
a,b,c = 1,2,"john"

Here, two integer objects with values 1 and 2 are assigned to variables a and b respectively, and one string object with the value "john" is assigned to the variable c.

Keywords in Python

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.

In Python, keywords are case sensitive. There are 33 keywords in Python. All the keywords except True, False and None are in lowercase and they must be written as it is. The list of all the keywords are given below.

Keywords in Python programming language

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	

Python Identifiers

Identifier is the name given to entities like class, functions, variables etc. in Python. It helps differentiating one entity from another.

Rules for writing identifiers

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 (_). Names like myClass, var_1 and print_this_to_screen, all are valid example.

An identifier cannot start with a digit. 1variable is invalid, but variable1 is perfectly fine.

Keywords cannot be used as identifiers.

```
>>> global = 1
File "<interactive input>", line 1
global = 1
^
SyntaxError: invalid syntax
```

Python Statement

Instructions that a Python interpreter can execute are called statements. For example, a = 1 is an assignment statement.

Multi-line statement

In Python, end of a statement is marked by a newline character. But we can make a statement extend over multiple lines with the line continuation character (\). For example:

```
a = 1 + 2 + 3 + \
4 + 5 + 6 + \
7 + 8 + 9
```

Expressions:

An **expression** is a combination of values, variables, and operators. A value all by itself is considered an expression, and so is a variable, so the following are all legal expressions (assuming that the variable x has been assigned a value):

Python Indentation

Most of the programming languages like C, C++, Java use braces { } to define a block of code. Python uses indentation.

A code block (body of a function, loop etc.) starts with indentation and ends with the first unindented line. The amount of indentation is up to you, but it must be consistent throughout that block.Generally four whitespaces are used for indentation and is preferred over tabs.

Example

```
for i in range(1,11):
    print(i)
    if i == 5:
        break
```

Python Comments

Comments are very important while writing a program. It describes what's going on inside a program so that a person looking at the source code does not have a hard time figuring it out. In Python, we use the hash (#) symbol to start writing a comment.It extends up to the newline character. Comments are for programmers for better understanding of a program. Python Interpreter ignores comment.

Example

```
#This is a comment
#print out Hello
print('Hello')
```

Multi-line comments

If we have comments that extend multiple lines, We can use triple quotes, either ''' or """.
These triple quotes are generally used for multi-line strings. But they can be used as multi-line comment as well.

Example

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

Data types in Python

Every value in Python has a datatype. Since everything is an object in Python programming, data types are actually classes and variables are instance (object) of these classes.
There are various data types in Python. Some of the important types are listed below.

1. Python Numbers
2. Python List
3. Python tuple
4. Python string
5. Python set
6. Python dictionary

1) **Python Numbers:** Number data types store numeric values. Number objects are created when you assign a value to them. For example:

```
var1 = 1
```

var2 = 10

2) **Python Strings:** Strings in Python are identified as a contiguous set of characters represented in the quotation marks. Python allows for either pairs of single or double quotes. Subsets of strings can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.
The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator. **For example:**

```
str = 'Hello World!'
print str # Prints complete string
print str[0] # Prints first character of the string
print str[2:5] # Prints characters starting from 3rd to 5th
print str[2:] # Prints string starting from 3rd character
print str * 2 # Prints string two times
print str + "TEST" # Prints concatenated string
```

3) List is an ordered sequence of items. It is one of the most used datatype in Python and is very flexible. All the items in a list do not need to be of the same type. Declaring a list is pretty straight forward. Items separated by commas are enclosed within brackets [ ].
example:

```
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tinylist = [123, 'john']
print list # Prints complete list
print list[0] # Prints first element of the list
print list[1:3] # Prints elements starting from 2nd till 3rd
print list[2:] # Prints elements starting from 3rd element
print tinylist * 2 # Prints list two times
print list + tinylist # Prints concatenated lists
```

4) **A tuple** is another sequence data type that is similar to the list. A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.
For example:

```
tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )
tinytuple = (123, 'john')
print tuple # Prints complete list
print tuple[0] # Prints first element of the list
print tuple[1:3] # Prints elements starting from 2nd till 3rd
print tuple[2:] # Prints elements starting from 3rd element
print tinytuple * 2 # Prints list two times
print tuple + tinytuple # Prints concatenated lists
```

5) **Dictionary** is an unordered collection of key-value pairs. It is generally used when we have a huge amount of data. Dictionaries are optimized for retrieving data. We must know the key to retrieve the value.
Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]).
For example:

```
dict = {}
```

```
dict['one'] = "This is one"
dict[2] = "This is two"
tinydict = {'name': 'john','code':6734, 'dept': 'sales'}
print dict['one'] # Prints value for 'one' key
print dict[2] # Prints value for 2 key
print tinydict # Prints complete dictionary
print tinydict.keys() # Prints all the keys
print tinydict.values() # Prints all the values
```

Python Operators

Operators are special symbols in Python that carry out arithmetic or logical computation. The value that the operator operates on is called the operand.

1. Arithmetic Operators
2. Comparison (Relational) Operators
3. Assignment Operators
4. Logical Operators
5. Bitwise Operators
6. Special Operators

Arithmetic operators

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

Arithmetic operators in Python

Operator	Meaning	Example
+	Add two operands or unary plus	x + y +2
-	Subtract right operand from the left or unary minus	x - y -2
*	Multiply two operands	x * y
/	Divide left operand by the right one (always results into float)	x / y
%	Modulus - remainder of the division of left operand by the right	x % y (remainder of x/y)
//	Floor division - division that results into whole number adjusted to the left in the number line	x // y
**	Exponent - left operand raised to the power of right	x**y (x to the power y)

Example

```
x = 15
y = 4
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)
```

Comparison operators (Relational Operators)

Comparison operators are used to compare values. It either returns True or False according to the condition.

Comparision operators in Python

Operator	Meaning	Example
>	Greater that - True if left operand is greater than the right	x > y
<	Less that - True if left operand is less than the right	x < y
==	Equal to - True if both operands are equal	x == y
!=	Not equal to - True if operands are not equal	x != y
>=	Greater than or equal to - True if left operand is greater than or equal to the right	x >= y
<=	Less than or equal to - True if left operand is less than or equal to the right	x <= y

Example

```
x = 10
y = 12
print('x > y  is',x>y)
print('x < y  is',x<y)
```

Ex: #!/usr/bin/python

```
a = 21
b = 10
c = 0

c = a + b
print "Line 1 - Value of c is ", c

c = a - b
print "Line 2 - Value of c is ", c

c = a * b
print "Line 3 - Value of c is ", c

c = a / b
print "Line 4 - Value of c is ", c

c = a % b
print "Line 5 - Value of c is ", c
```

```
a = 2
b = 3
c = a**b
print "Line 6 - Value of c is ", c

a = 10
b = 5
c = a//b
print "Line 7 - Value of c is ", c
```

Output:

```
Line 1 - Value of c is 31
Line 2 - Value of c is 11
Line 3 - Value of c is 210
Line 4 - Value of c is 2
Line 5 - Value of c is 1
Line 6 - Value of c is 8
Line 7 - Value of c is 2
```

Logical operators

Logical operators are the and, or, not operators.

Logical operators in Python

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

```
x = True
y = False
# Output: x and y is False
print('x and y is',x and y)
# Output: x or y is True
print('x or y is',x or y)
# Output: not x is False
print('not x is',not x)
```

Assignment operators

Assignment operators are used in Python to assign values to variables. a = 5 is a simple assignment operator that assigns the value 5 on the right to the variable a on the left.

Operator	Example	Equivatent to
=	x = 5	x = 5
+=	x += 5	x = x + 5
-=	x -= 5	x = x - 5
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5
%=	x %= 5	x = x % 5
//=	x //= 5	x = x // 5
**=	x **= 5	x = x ** 5
&=	x &= 5	x = x & 5
|=	x |= 5	x = x | 5
^=	x ^= 5	x = x ^ 5
>>=	x >>= 5	x = x >> 5
<<=	x <<= 5	x = x << 5

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
Line 5 - Value of c is 2
Line 6 - Value of c is 2097152
Line 7 - Value of c is 99864

Special operators

Python language offers some special type of operators like the identity operator or the membership operator.

Identity operators

is and is not are the identity operators in Python. They are used to check if two values (or variables) are located on the same part of the memory. Two variables that are equal does not imply that they are identical.

Identity operators in Python		
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

```
x1 = 5
y1 = 5
x2 = 'Hello'
y2 = 'Hello'
x3 = [1,2,3]
y3 = [1,2,3]
# Output: False
print(x1 is not y1)
# Output: True
print(x2 is y2)
# Output: False
print(x3 is y3)
```

Here, we see that x1 and y1 are integers of same values, so they are equal as well as identical. Same is the case with x2 and y2 (strings).

But x3 and y3 are list. They are equal but not identical. Since list are mutable (can be changed), interpreter locates them separately in memory although they are equal.

Membership operators

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

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 = 'Hello world'
y = {1:'a',2:'b'}
# Output: True
print('H' in x)
# Output: True
print('hello' not in x)
# Output: True
print(1 in y)
# Output: False
print('a' in y)
```

Here, 'H' is in x but 'hello' is not present in x (remember, Python is case sensitive). Similary, 1 is key and 'a' is the value in dictionary y. Hence, 'a' in y returns False.

Python Bitwise Operators

Bitwise operator works on bits and performs bit by bit operation. Assume if a = 60; and b = 13; Now in binary format they will be as follows –

a = 0011 1100
b = 0000 1101
---
a&b = 0000 1100
a|b = 0011 1101
a^b = 0011 0001
~a = 1100 0011

There are following Bitwise operators supported by Python language

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	It is unary and has the effect of 'flipping' bits.	(~a) = -61

Complement		(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)

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
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
```

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
```

Python Output Using print() function

We use the print() function to output data to the standard output device (screen).

Example

```
print('This sentence is output to the screen')
# Output: This sentence is output to the screen
a = 5
print('The value of a is', a)
# Output: The value of a is 5
```

Output formatting

Sometimes we would like to format our output to make it look attractive. This can be done by using the str.format() method. This method is visible to any string object.

Example:

```
x = 5; y = 10
print('The value of x is {} and y is {}'.format(x,y))
```

Output:- The value of x is 5 and y is 10

Here the curly braces {} are used as placeholders. We can specify the order in which it is printed by using numbers (tuple index).

Example

```
print('I love {0} and {1}'.format('bread','butter'))
# Output: I love bread and butter
print('I love {1} and {0}'.format('bread','butter'))
# Output: I love butter and bread
```

We can even use keyword arguments to format the string.

```
print('Hello {name}, {greeting}'.format(greeting = 'Goodmorning', name = 'John'))
Hello John, Goodmorning
```

We can even format strings like the old sprintf() style used in C programming language. We use the % operator to accomplish this.

```
x = 12.3456789
print('The value of x is %3.2f' %x)
Output: The value of x is 12.35
print('The value of x is %3.4f' %x)
Output: The value of x is 12.3457
```

Python Input

To allow flexibility we might want to take the input from the user. In Python, we have the input() function to allow this. The syntax for input() is

```
input([prompt])
```

where prompt is the string we wish to display on the screen. It is optional.

Example: num = input('Enter a number: ')

Python Import

When our program grows bigger, it is a good idea to break it into different modules.

A module is a file containing Python definitions and statements. Python modules have a filename and end with the extension .py.

Definitions inside a module can be imported to another module or the interactive interpreter in Python. We use the import keyword to do this.
For example, we can import the math module by typing in import math.

Example

```
import math
print(math.pi)
```

Exercises:

1. Assume that we execute the following assignment statements:
width = 17
height = 12.0
delimiter = '.'
For each of the following expressions, write the value of the expression and the type (of the value of the expression).
1. width/2
2. width/2.0
3. height/3
4. 1 + 2 * 5
5. delimiter * 5

2. The volume of a sphere with radius *r* is (4/3) π*r*3. What is the volume of a sphere with radius 5? Hint: 392.6 is wrong!

3. Python Program to Compute Simple Interest Given all the Required Values.

4. Python Program to Exchange the Values of Two Numbers Without Using a Temporary Variable

5. Write a Python program which accepts the user's first and last name and print them in reverse order with a space between them.

6. Write a Python program to check whether a specified value is contained in a group of values.
Ex: *Test Data* :
3 -> [1, 5, 8, 3] : True
-1 -> [1, 5, 8, 3] : False

7. Write a Python program that will accept the base and height of a triangle and compute the area.

8. Write a Python program to solve (x + y) * (x + y)

9. Write a python program to sum of the first n positive integers

10. Given variables x=30 and y=20, write a Python program to print t "30+20=50"

11. Write a Python program to swap two variables.