Q. What is Python ?

Python is an interpreted, object-oriented, high level and Dynamic typing language

Inventor of Python :

Guido Van Rossum (FEB 20, 1991)

Features of Python :

- Easy to learn, read and maintain
- Easy to use
- Broad Standard Library
- Open source
- Portable
- Object - Oriented

Difference between interpreters & Compilers :

Interpreter:

1. Translates program one statement at a time
2. Continue translating the program until the first error is met, in which case is stop hence debugging is easy.
3. Ex: PYthon, Ruby use interpreters.

Compiler:

1. Scans the entire program and translate it as a whole into machine code.
2. It generate the error message only after scanning the whole program. Hence debugging is comparatively hard.
3. Ex. C, C++, Java use compilers

Q. Why to learn Python ?

Python syntax are easy compared to other language.

The print function in Python is a function that outputs to your console window whatever you say you want to print out.

In [ ]:

```
print("Hello World")
```

Variable

Variables only start with alphabets

In [ ]:

```
a=10
print(a)
```

In [ ]:

```
b=10
```

In [ ]:

```
print(a)
```

In [ ]:

```
print(a,b)
```

In [ ]:

```
print(id(a))
```

In [ ]:

```
print(id(b))
```

In [ ]:

```
c=20
print(id(c))
```

In [ ]:

```
A='Hello'
```

In [ ]:

```
print(A)
```

In [ ]:

```
_ = 10
print(_)
```

In [ ]:

```
a1=5
print(a1)
```

In [ ]:

```
1a=6
print(1a)
```

In [ ]:

```
a_b=15
print(a_b)
```

In [ ]:

```
print('a_b')
```

In [ ]:

```
@a=25
print(@a)
```

In [ ]:

```
_a=25
print(_a)
```

String Concatenation/ Join two string variables

In [ ]:

```
a="Hello"
```

In [ ]:

```
b= "NetTech India"
```

In [ ]:

```
print(a+b)
```

In [ ]:

```
print(a+" "+b)
```

In [ ]:

```
c=20
```

In [ ]:

```
print(c+30)
```

In [ ]:

```
print(a+c)
```

In [ ]:

```
d='50'
```

In [ ]:

```
print(a+d)
```

In [ ]:

```
print(c+d)
```

In [ ]:

```
a=10
print(a)
b="sanvee"
print(b)
c="hello"
print(c)
d = "NetTech"
print(d)
```

Keyword

Keywords are the reserved words in python. These reserved words cannot be used as function name, variable name or any other identifier.

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

and	A logical operator
as	To create an alias
assert	For debugging
break	To break out of a loop
class	To define a class
continue	To go to the next iteration of a loop
def	To define a function
del	To delete an object
elif	A conditional statements, like else if
else	A conditional statements
except	Used with exceptions, what to do when an exception occurs
False	Boolean value
finally	Used with exceptions, will be executed no matter if there is an exception or not
for	To create a for loop
from	To import specific parts of a module
global	To declare a global variable
if	To make a conditional statement
import	To import a module
in	To check if a value is in a list, tuple
is	To test if two variables are equal
lambda	To create an anonymous function
None	Represents a null value
nonlocal	To declare a non-local variable
not	A logical operator
or	A logical operator
pass	A statement that will do nothing (null)
raise	To raise an exception
return	To exit a function and return a value
True	Boolean value
try	To make a try...except statement
while	To create a while loop
with	Used to simplify exception handling
yield	To end a function, returns a generator

In [ ]:

```
help("keywords")
```

Statements

Instructions that you write in your code and that a Python interpreter can execute are called statements.

In [ ]:

```
a= 1+3+5
print(a)
```

Comments

Comments are nothing but the sentences that the python interpreter ignores.

In [ ]:

```
#this is a comment
print("Hello World!")
```

In [ ]:

```
'''I start my comment from this line...
still the comment...
still the comment...
okay finished :)'''
print("Hello World!")
```

In [ ]:

```
"""I start my comment from this line...
still the comment...
still the comment...
okay finished :)"""
print("Hello World!")
```

Indentation

Indentation in Python refers to the (spaces and tabs) that are used at the beginning of a statement or a code we write.

In [ ]:

```
#using indentation for a bunch of codes is extremely elegant and contributes a lot to the c
#our code starts looking well organised and more readable.
```

Datatypes

1. Integer
2. Float
3. Complex

4. String
5. List
6. Tuple
7. Set
8. Dictionary
9. Boolean
10. Range

In [ ]:

```
#we can use type function to get the type information of a value.
a=10
type(a)
```

In [ ]:

```
b=10.0
print(type(b))
```

In [ ]:

```
#A complex number is a number with a real and an imaginary component represented as a+bj wh
c=10j
print(type(c))
```

In [ ]:

```
var = "Welcome to NetTech India"
type(var)
```

In [ ]:

```
#The Python List is an ordered collection (also known as a sequence ) of elements.
List = ["apple", "banana", "cherry",10]
type(List)
```

In [ ]:

```
#Tuples are ordered collections of elements that are unchangeable.
Tuple = ("apple", "banana", "cherry")
print(type(Tuple))
```

In [ ]:

```
#a set is an unordered collection of data items that are unique.
Set = {"apple", "banana", "cherry"}
type(Set)
```

In [ ]:

```
#dictionaries are unordered collections of unique values stored in (Key-Value) pairs.
Dictionary = {"Name":"Sanvee"}
print(type(Dictionary))
```

In [ ]:

```
#to represent boolean values (True and False) we use the bool data type.
print(20>10)
```

In [ ]:

```
x = 25
y = 20

z = x > y
print(z)
print(type(z))
```

In [ ]:

```
#The built-in function  range() used to generate a sequence of numbers from a start number
numbers = range(10, 15, 1)
print(type(numbers))
```

In [ ]:

```
numbers = range(10, 15, 2)
print(type(numbers))
```

Dynamic typing and Static Typing

In python, variables are the reserved memory locations to store values. Python is a dynamically typed language which is not required to mention the type of variable while declaring. It performs the type checking at run time.

In statically typed programming languages which is required to mention the type of variable while declaring. It performs the type checking at compile time.

In [ ]:

```
msg="Hello World"
print(msg)
```

Here, the variable msg contains a string value “Hello World”. It is not mandatory to mention the type of msg as string which will decided at runtime.

Input and Output

Until now the values were defined to the variables. In some cases user might want to input values to variables, which allows flexibility.

Python has input() function to perform this.

In [ ]:

```
a = input()
print(a)
```

In [ ]:

```
my_name = input("Enter Name:")
print("My name is", my_name)
```

In [ ]:

```
my_number = input("Enter Number:")
print("Number is", my_number)
```

In [ ]:

```
print(type(my_number))
```

In [ ]:

```
my_number1 = int(input("Enter Number:"))
print("Number is", my_number)
```

In [ ]:

```
print(type(my_number1))
```

In [ ]:

```
n = input("enter name : ")
print("My name is "+ n)
```

Operators

Arithmatic Operator :

1. Addition(+)
2. Subtraction(-)
3. Multiplication(*)
4. Division(/)
5. Floor division(//)
6. Modulus(%)
7. Exponent(**)

In [ ]:

```
#Addition
x = 10
y = 40
print(x + y)
```

In [ ]:

```
#Subtraction
x = 10
y = 40
print(y - x)
```

In [ ]:

```
#Multiplication
x = 2
y = 4
print(x * y)
```

In [ ]:

```
#Division
x = 2
y = 4
print(y / x)
```

In [ ]:

```
#Floor Division(It returns the quotient (the result of division) in which the digits after
x = 2.2
y = 4
# normal division
print(y / x)
#floor division
print(y // x)
```

In [ ]:

```
#Modulus(The remainder of the division)
x = 15
y = 4

print(x % y)
```

In [ ]:

```
#Exponent(power of a number)
num = 2
# 2*2
print(num ** 2)
```

In [ ]:

```
num1 = 3
# 3*3
print(num1 ** 2)
```

Relational Operator

1. Greater than (>)
2. Less than (<)
3. Equal to (==)
4. Not equal to (!=)
5. Greater than equal to (>=)
6. Less than equal to (<=)

In [ ]:

```
x = 10
y = 5
print(x > y)
print(x < y)
```

In [ ]:

```
print(x == y)
print(x == 10)
```

In [ ]:

```
print(x != y)
print(10 != x)
```

In [ ]:

```
print(x >= y)
print(10 >= x)
```

In [ ]:

```
print(x <= y)
print(10 <= x)
```

Assignment operator

1. Assign (=)
2. Add and assign(+=)
3. Subtract and assign(-=)
4. Multiply and assign(*=)
5. Divide and assign(/=)
6. Floor divide and assign(//=)
7. Modulus and assign(%=)
8. Exponent and assign(**=)

In [ ]:

```
a = 4
b = 2

a += b
print(a)
```

In [ ]:

```
a = 4
a -= 2
print(a)
```

In [ ]:

```
a = 4
a *= 2
print(a)
```

In [ ]:

```
a = 4
a /= 2
print(a)
```

In [ ]:

```
a = 4
a **= 2
print(a)
```

In [ ]:

```
a = 5
a %= 2
print(a)
```

a = 4 a //= 2 print(a)

Logical Operator

1. and : The logical and operator returns True if both expressions are True.
2. or : The logical or the operator returns a boolean True if one expression is true.
3. not : The logical not operator returns boolean True if the expression is false.

In [ ]:

```
x = 5

print(x > 3 and x < 10)
```

```
True
```

In [ ]:

```
#In the case of arithmetic values, Logical and always returns the second value
print(10 and 20)
print(10 and 5)
print(100 and 300)
```

```
20
5
300
```

In [ ]:

```
x = 5

print(x > 3 or x < 4)
```

```
True
```

In [ ]:

```
#In the case of arithmetic values, Logical or it always returns the first value
print(10 or 20)
print(10 or 5)
print(100 or 300)
```

```
10
10
100
```

In [ ]:

```
x = 5

print(not(x > 3))
```

```
False
```

In [ ]:

```
#In the case of arithmetic values, Logical not always return False for nonzero value.
print(not 10) # False. Non-zero value
print(not 0)  # True. Non-zero value
```

```
False
True
```

Membership Operator

In Python, there are two membership operator in and not in

In [ ]:

```
#in operator
x = ["apple", "banana"]

print("cherry" in x)
```

In [ ]:

```
#Not in operator
x = ["apple", "banana"]

print("pineapple" not in x)
```

Identity Operator

Use the Identity operator to check whether the value of two variables is the same or not.

Python has 2 identity operators is and is not.

In [ ]:

```
#The is operator returns Boolean True or False.
x = 10
y = 11
z = 10
print(x is y) # it compare memory address of x and y
print(x is z) # it compare memory address of x and z
```

```
False
True
```

In [ ]:

```
x = 10
y = 11
z = 10
print(x is not y) # it campare memory address of x and y
print(x is not z) # it campare memory address of x and z
```

```
True
False
```

Bitwise Operator

1. & Bitwise and
2. | Bitwise or
3. ^ Bitwise xor

In Python, bitwise operators are used to performing bitwise operations on integers. To perform bitwise, we first need to convert integer value to binary (0 and 1) value.

In [ ]:

```
#AND
a = 7
b = 4
c = 5
print(a & b)#5
print(a & c)
print(b & c)
```

```
4
5
4
```

In [ ]:

```
#OR
a = 7
b = 4
c = 5
print(a | b)
print(a | c)
print(b | c)
print(13|17)
```

```
7
7
5
29
```

In [ ]:

```
a = 4
b = 3
print(a ^ b)
```

```
7
```

In [ ]:

```
a = 45
b = 35
print(a | b)
```

```
47
```

In [ ]:

```
a = 4
b = 5
print(a | b)
```

```
5
```

In [ ]:

```
a= 5
b= 4
print(a or b)
```

```
5
```

In [ ]:

```
a=55
b=21
print(a ^ b)
```

```
34
```

In [ ]:

```
a = 34
b = 34
print(a & b)
```

```
34
```

In [ ]:

```

```