What is Python?

- Python is free and simple to learn.
- Its primary features are that it is high-level, dynamically typed and interpreted.
- This makes debugging of errors easy and encourages the rapid development of application prototypes, marking itself as the language to code with.
- Python was developed in **1989** by **Guido Van Rossum** and emphasizes on the DRY (Don't Repeat Yourself) principle and readability.

Python Features/Advantages:

1. Easy Syntax – No semicolons, No Curly Braces
2. Readable - Easy to use & Learn
3. Platform Independent – Can be used cross-platform
4. Dynamically Typed – No need to specify data type of variable
5. Free and Open Source – We don't have to pay money to any organization to use Python
6. Follows Multiple Paradigms – like Procedure Oriented, Functional, Imperative
7. Supports OOP as well
8. Extensible/Integrated – Can be used with C,C++
9. Large Library Collection – Plenty of third party modules & Predefined functions
10. Powerful Web Frameworks – like Flask, Django.

Limitations:

- Backward Incompatible - Python 2.x program can't be executed in Python 3.x
- Slow – Compared to C/C++
- Weak in case of mobile development.

History of Python:

- Conceived by Guido Van Rossum in 1989.
- Influenced by Languages like – ABC, Modula-3
- Python 1.x -1994
- Python 2.x -2000
- Python 3.x -2008
- Python 3.8.0 -2019

Python Applications:

Web Development

Game Development

Machine Learning and AI

Data Science and Data Visualization

Desktop GUI

Business Applications

Embedded Applications

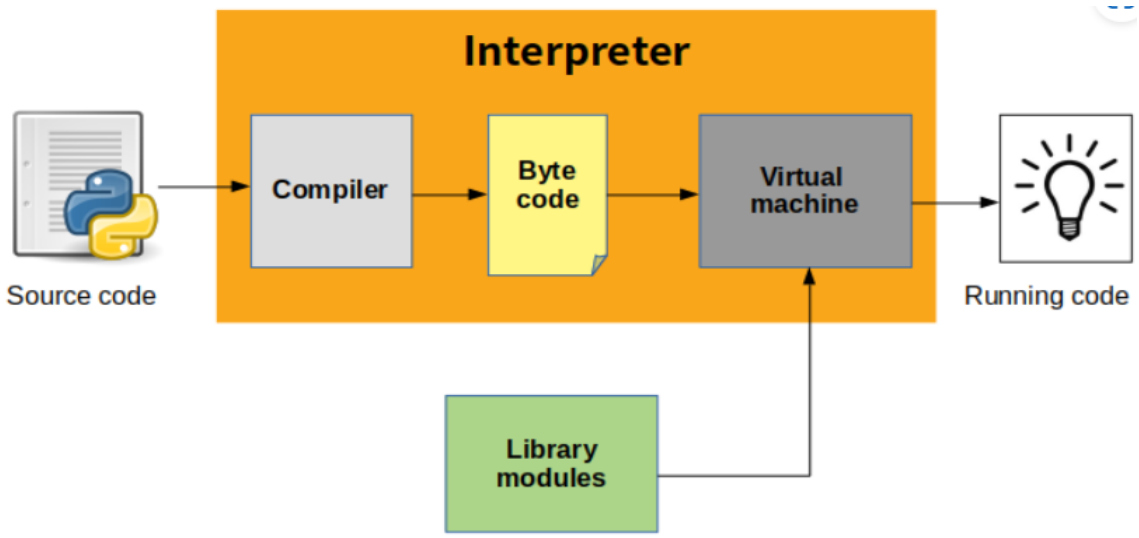
Where Python is used in Industry?

Whether Python is Interpreted or Compiled?

- **Compiler:**
  - A compiler is a computer program that transforms (translates) source code of a programming language (the source language) into another computer language (the target language).
  - In most cases compilers are used to transform source code into executable program, i.e. they translate code from high-level programming languages into low (or lower) level languages, mostly assembly or machine code.

- **Interpreter:**
  - An interpreter is a computer program that executes instructions written in a programming language.

– It can either execute the source code directly or translate the source code in a first step into a more efficient representation and execute this code.



Steps to compile .py file:

1) Python code is translated into intermediate code
2) Python virtual machine run intermediate code.
3) We can compile manually python program.
   ----------cmd----------
   python -m py_compile .py
   python -m compile

Keywords in Python:

– These are the reserved words in Python.
– We cannot use a keyword as a identifier or as variable.

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	async
break	except	in	raise	await

Python - Data Types:

- Python Data Types are used to define the type of a variable.
- It defines what type of data we are going to store in a variable.
- The data stored in memory can be of many types.
- For example: a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters.
- Python has various built-in data types which we will discuss with in this tutorial:

    - Numeric - int, float, complex
    - String - str
    - Sequence - list, tuple, range
    - Binary - bytes, bytearray, memoryview
    - Mapping - dict
    - Boolean - bool
    - Set - set, frozenset
    - None - NoneType

Variables:

- Python variables are the reserved memory locations used to store values with in a Python Program.
- This means that when you create a variable you reserve some space in the memory.

Python Variable Names: (Rules to define Variable)

- Every Python variable should have a unique name like a, b, c.
- A variable name can be meaningful like color, age, name etc.
- There are certain rules which should be taken care while naming a Python variable:

    1. A variable name must start with a letter or the underscore character
    2. A variable name cannot start with a number or any special character like $, (, * % etc.
    3. A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
    4. Python variable names are case-sensitive which means Name and NAME are two different variables in Python.
    5. Python reserved keywords cannot be used naming the variable.

Python – Comments:

- Python comments are programmer-readable explanation or annotations in the Python source code.
- They are added with the purpose of making the source code easier for humans to understand, and are ignored by Python interpreter.
- Comments enhance the readability of the code and help the programmers to understand the code very carefully.
- There are three types of comments available in Python

    - Single line Comments (#)

- Multiline Comments (# OR ’’’_____’’’)

Python Operators:

- Python operators are the constructs which can manipulate the value of operands.
- These are symbols used for the purpose of logical, arithmetic and various other operations.
- Consider the expression 4 + 5 = 9. Here, 4 and 5 are called **operands** and + is called **operator**.

Types of Python Operators:

Python language supports the following types of operators.

- Arithmetic Operators
- Comparison (Relational) Operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

A. Arithmetic Operators:

- Python arithmetic operators are used to perform mathematical operations on numerical values.
- These operations are Addition, Subtraction, Multiplication, Division, Modulus, Exponents and Floor Division.

Operator	Name	Example
+	Addition	10 + 20 = 30
-	Subtraction	20 – 10 = 10
*	Multiplication	10 * 20 = 200
/	Division	20 / 10 = 2
%	Modulus	22 % 10 = 2
**	Exponent	4**2 = 16
//	Floor Division	9//2 = 4

B. Comparison Operators

- Python comparison operators compare the values on either sides of them and decide the relation among them.
- They are also called relational operators.
- These operators are equal, not equal, greater than, less than, greater than or equal to and less than or equal to.

Operator	Name	Example
==	Equal	4 == 5 is not true.
!=	Not Equal	4 != 5 is true.
>	Greater Than	4 > 5 is not true.
<	Less Than	4 < 5 is true.
>=	Greater than or Equal to	4 >= 5 is not true.
<=	Less than or Equal to	4 <= 5 is true.

C. Assignment Operators:

- Python assignment operators are used to assign values to variables.
- These operators include simple assignment operator, addition assign, subtraction assign, multiplication assign, division and assign operators etc.

Operator	Name	Example
=	Assignment Operator	a = 10
+=	Addition Assignment	a += 5 (Same as a = a + 5)
-=	Subtraction Assignment	a -= 5 (Same as a = a - 5)
*=	Multiplication Assignment	a *= 5 (Same as a = a * 5)
/=	Division Assignment	a /= 5 (Same as a = a / 5)
%=	Remainder Assignment	a %= 5 (Same as a = a % 5)
**=	Exponent Assignment	a **= 2 (Same as a = a ** 2)
//=	Floor Division Assignment	a //= 3 (Same as a = a // 3)

D. Bitwise Operators

- Bitwise operator works on bits and performs bit by bit operation.

- Assume if a = 60; and b = 13; Now in the binary format their values will be 0011 1100 and 0000 1101 respectively.
- Following table lists out the bitwise operators supported by Python language with an example each in those, we use the above two variables (a and b) as operands −

a = 0011 1100

b = 0000 1101

---

a&b = 12 (0000 1100)

a|b = 61 (0011 1101)

a^b = 49 (0011 0001)

~a = -61 (1100 0011)

a << 2 = 240 (1111 0000)

a>>2 = 15 (0000 1111)

Operator	Name	Example
&	Binary AND	Sets each bit to 1 if both bits are 1
|	Binary OR	Sets each bit to 1 if one of two bits is 1
^	Binary XOR	Sets each bit to 1 if only one of two bits is 1
~	Binary Ones Complement	Inverts all the bits
<<	Binary Left Shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Binary Right Shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

E. Logical Operators:

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.

F. Membership Operator:

- Python's membership operators test for membership in a sequence, such as strings, lists, or tuples.
- There are two membership operators as explained below –

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.

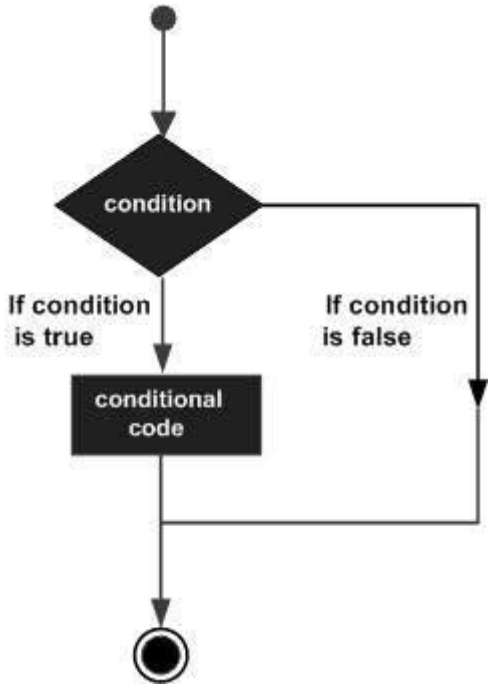
G. Identity Operator:

- Identity operators compare the memory locations of two objects.
- There are two Identity operators explained below –

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).

Decision making Statements:

- Decision making is anticipation of conditions occurring while execution of the program and specifying actions taken according to the conditions.
- Decision structures evaluate multiple expressions which produce TRUE or FALSE as outcome.
- You need to determine which action to take and which statements to execute if outcome is TRUE or FALSE otherwise.
- Following is the general form of a typical decision-making structure found in most of the programming languages −
- Python programming language assumes any **non-zero** and **non-null** values as TRUE, and if it is either **zero** or **null**, then it is assumed as FALSE value.



– Python programming language provides following types of decision-making statements.

Sr.No.	Statement & Description
1	if statements An **if statement** consists of a boolean expression followed by one or more statements.
2	if...else statements An **if statement** can be followed by an optional **else statement**, which executes when the boolean expression is FALSE.
3	nested if statements You can use one **if** or **else if** statement inside another **if** or **else if** statement(s).

a. IF statement:

– The **if** statement contains a logical expression using which data is compared and a decision is made based on the result of the comparison.

– Syntax:

If expression:

Statements:

– If the boolean expression evaluates to TRUE, then the block of statement(s) inside the if statement is executed.

– If boolean expression evaluates to FALSE, then the first set of code after the end of the if statement(s) is executed.

b. IF…ELSE Statement:

– An **else** statement can be combined with an **if** statement.

– An **else** statement contains the block of code that executes if the conditional expression in the if statement resolves to 0 or a FALSE value.

– The *else* statement is an optional statement and there could be at most only one **else** statement following **if**.

– Syntax:

If expression:

Statements

Else:

Statements

c. ELIF Statements:

- The **elif** statement allows you to check multiple expressions for TRUE and execute a block of code as soon as one of the conditions evaluates to TRUE.
- Similar to the **else**, the **elif** statement is optional.
- However, unlike **else**, for which there can be at most one statement, there can be an arbitrary number of **elif** statements following an **if**.
- Syntax:

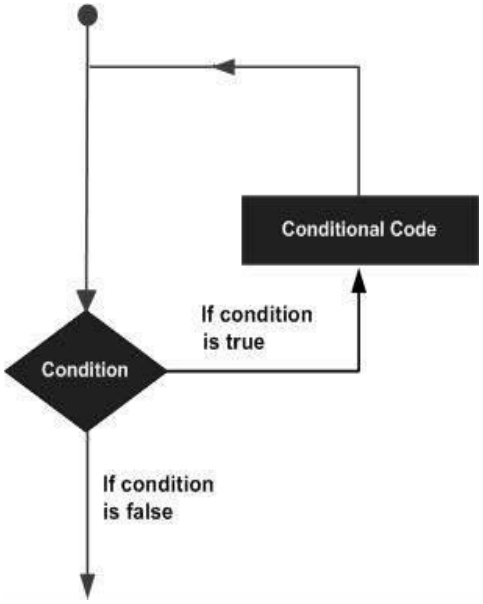
```
If expression:
        Statements
Elif expression:
        Statements
Elif expression:
        Statements
Else:
        Statements:
```

d. Nested IF Statements:

- There may be a situation when you want to check for another condition after a condition resolves to true.
- In such a situation, you can use the nested **if** construct.
- In a nested **if** construct, you can have an **if...elif...else** construct inside another **if...elif...else** construct.
- **Syntax:**

```
If expression1:
        Statements…..
        If expression2:
                Statements….
Else:
        Statements….
```

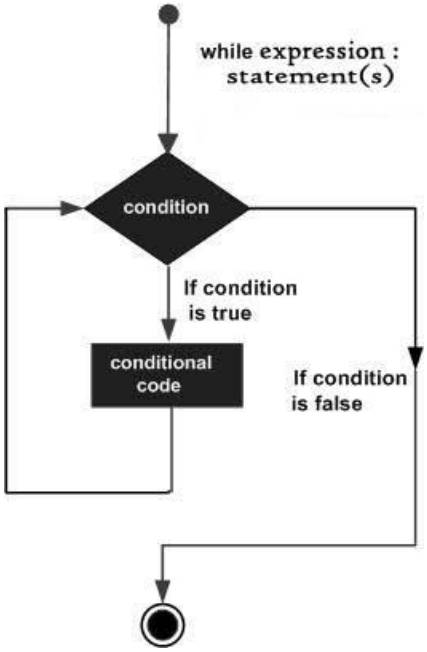
Python – Loops:



- In general, statements are executed sequentially:
- The first statement in a function is executed first, followed by the second, and so on.
- There may be a situation when you need to execute a block of code several number of times.
- Programming languages provide various control structures that allow for more complicated execution paths.
- A loop statement allows us to execute a statement or group of statements multiple times.
- Python programming language provides following types of loops to handle looping requirements.:

Sr.No.	Loop Type & Description
1	while loop Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.
2	for loop Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
3	nested loops You can use one or more loop inside any another while, for or do..while loop.

A. WHILE LOOP:



- While loop is know as indefinite or conditional loop.
- They will keep iterating till certain conditions are met.
- There is no guarantee ahead of time regarding how many time the loop will iterate.
- This is used when you don't know how many iterations that is required for execution.
- SYNTAX:

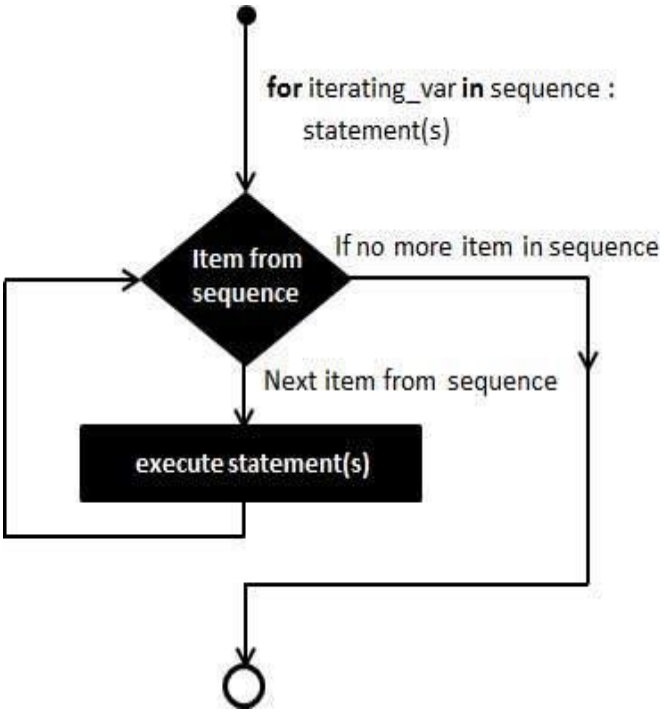
```
initialization
while <Condition>:
    statements
    increment/Decrement
```

* E.G.Print 0 to 10 numbers

```
i=0
while i<10:
   print(i)
   i=i+1
```

B. FOR LOOP:

- For loop repeats the group of statements a specified number of time.
- The syntax of for loop contains:
  - i. Boolean Condition
  - ii. Initial Value of counting variable
  - iii. Final Value of counting variable
  - iv. Increment / Decrement



- SYNTAX:

```
for <variable> in range(range):
   stmnt1...
   stmnt2...
   .
   .
   stmnt n
```

C. NESTED LOOPS:

- Python programming language allows to use one loop inside another loop.
- Syntax of NSETED FOR LOOP:

```
For expression:
        For expression:
                Statements….
Statements….
```

- Syntax of NESTED WHILE LOOP:

```
While expression:
        While expression:
                Statements:
Statements:
```

Loop Control Statements:

- Loop control statements change execution from its normal sequence.
- When execution leaves a scope, all automatic objects that were created in that scope are destroyed.
- Python supports the following control statements.

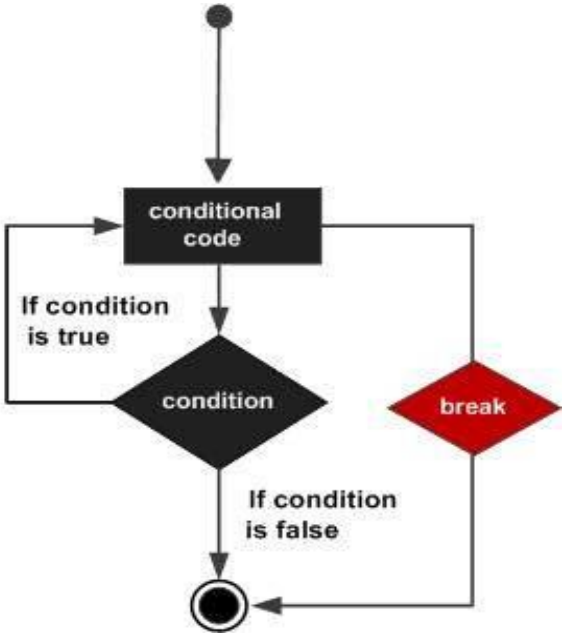
Sr.No.	Control Statement & Description
1	break statement Terminates the loop statement and transfers execution to the statement immediately following the loop.
2	continue statement Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
3	pass statement The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.

a. Break Statement:

- It terminates the current loop and resumes execution at the next statement.
- **Syntax**:

  Break

- **Flowchart for Break Statement**:



- **Example for Break statement** :

---------- Example 1 using FOR ----------

```
for letter in 'Python':
    if letter == 'h':
        break
    print("Current Letter:", latter)
```

---------- Example 2 using While ----------

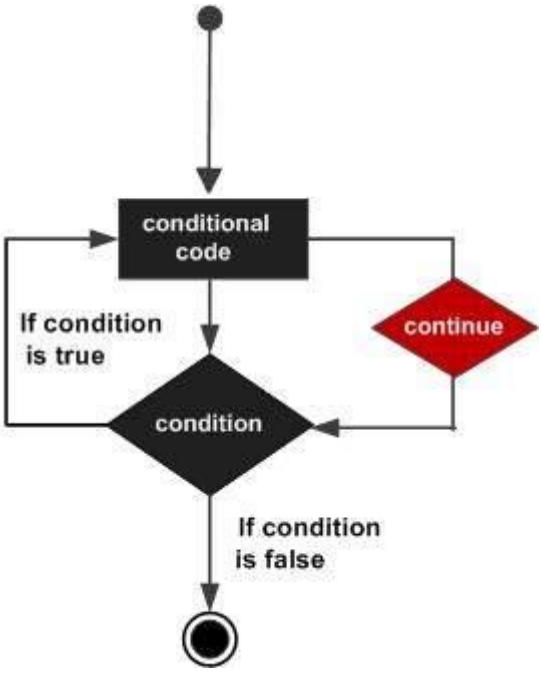
```
var = 10
while var > 0:
    print("The Current Letter :", var)
    var = var -1
    if var == 5 :
        break
```

b. Continue Statement:

- It returns the control to the beginning of the while loop.
- The **continue** statement rejects all the remaining statements in the current iteration of the loop and moves the control back to the top of the loop.
- **Syntax**:

```
Continue
```

- **Flowchart** for Continue Statement:



c. Pass Statement:

- It is used when a statement is required syntactically but you do not want any command or code to execute.
- The **pass** statement is a *null* operation; nothing happens when it executes.
- The **pass** is also useful in places where your code will eventually go, but has not been written yet.
- **Syntax**:

```
Pass
```

- **Example** of Pass Statement:

```
For letter in  'Python':
    If letter == 't':
        Pass
        Print("Pass of statement……")
    Print("The current Letter:", letter)
```

Difference Between FOR and WHILE LOOP:

Basis of Comparison	For Loop	While Loop
Keyword	Uses **for** keyword	Uses **while** keyword
Used	For loop is used when the number of iterations is already known.	While loop is used when the number of iterations is already Unknown.
absence of condition	The loop runs infinite times in the absence of condition	Returns the compile time error in the absence of condition
Nature of Initialization	Once done, it cannot be repeated	In the while loop, it can be repeated at every iteration.
Functions	To iterate, the range or xrange function is used.	There is no such function in the while loop.
Initialization based on iteration	To be done at the beginning of the loop.	In the while loop, it is possible to do this anywhere in the loop body.
Generator Support	Python's for loop can iterate over generators.	While loops cannot be directly iterated on Generators.
Speed	The for loop is faster than the while loop.	While loop is relatively slower as compared to for loop.

Python Functions:

- A function is a block of organized, reusable code that is used to perform a single, related action.
- Functions provide better modularity for your application and a high degree of code reusing.
- A function is a block of organized, reusable code that is used to perform a single, related action.
- Functions provide better modularity for your application and a high degree of code reusing.

Defining a Function:

- You can define functions to provide the required functionality.
- Here are simple rules to define a function in Python.
  - Function blocks begin with the keyword **def** followed by the function name and parentheses (( )).
  - Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
  - The first statement of a function can be an optional statement - the documentation string of the function or *docstring*.
  - The code block within every function starts with a colon (:) and is indented.
  - The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.
  - **Syntax:**

    Def function_name(parameters):

        Statements……

        Statements……

        Return [Expression]

  - **Example**:

    Def demo(str):

        Print str

        Return

Calling a Function:

- Defining a function only gives it a name, specifies the parameters that are to be included in the function and structures the blocks of code.
- Once the basic structure of a function is finalized, you can execute it by calling it from another function or directly from the Python prompt.
- **Example**:

```
Def demo(str):
        Print str
Demo(“This is Python Program”)
```

TYPES OF FUNCTION:

A. Parameterized Function

B. Non-Parameterized Function

A. Non-Parameterized Function:

- Function with no parameters.

E.G.

```
def add():
    result=4+5
    print("Addition:" result)
add()
```

B. Parameterized Function:

- Function with parameters.

ARGUMENTS:

-We use them at the time of function call.

Types:

a. Default Arguments

b. Positional Arguments

c. Keyword Arguments

FUNCTIONS WITH RETURN STATEMENTS:

- The return statement is special statement that can use inside function to send function's result back to caller.
- The return statement consists of return keyword followed by specific value.
- It returns values as string, int, list , tuple.
- When function doesn't have return statement then return type of function is NONE.

SYNTAX:

```
def function_name():
    //statements
```

```
        return [expression]
```

EXAMPLE:

```
    def add():
        total=5+4
        return total
    res=add()
    print("Addition is:",res)
```

RETURN STATEMENT WITH OR WITHOUT PARAMETERS:

A. Return with no parameters:

```
    def add():
        total=5+4
        return total
    res=add()
    print("Addition is:",res)
```

B. Return with parameters:

```
    def add(a,b):
        total=a+b
        return total
    res=add(5,4)
    print("Addition is:",res)
```

C. Function without return:

```
    def add(a,b):
        total=a+b

    res=add(5,4)
    print("Addition is:",res)
```

*** It doesnot throws error. It runs code and Print value NONE ***