

Python

# What is Python ?

Python is a clear and powerful object-oriented programming language, comparable to Perl, Ruby, or Java.

Python is a programming language that combines features of C and Java.

<https://www.python.org/>

# History

- Python was developed by *Guido Van Rossum* in the year *1990* at Stichting Mathematisch Centrum in the Netherlands as a successor of a language called ABC.
- Name “Python” picked from TV Show Monty Python’s Flying Circus.

<https://docs.python.org/3/license.html>

# Version

- Python 0.9.0 - February, 1991
- Python 1.0 - January 1994
- Python 2.0 - October, 2000
- Python 3.0 - December, 2008
- Python 3.1 - June, 2009
- Python 3.2 - February, 2011
- Python 3.3 - September, 2012
- Python 3.4 - March, 2014
- Python 3.5 - September, 2015
- Python 3.6 - December, 2016
- Python 3.7 - June, 2018

Python 3.5+ cannot be used on Windows XP or earlier.

<https://www.python.org/doc/versions/>

# Features

- Easy to Learn
- High Level Language
- Interpreted Language
- Platform Independent
- Procedure and Object Oriented
- Huge Library
- Scalable

# Application for Python

- Web Application - Django, Pyramid, Flask, Bottle
- Desktop GUI Application – Tkinter
- Console Based Application
- Games and 3D Application
- Mobile Application
- Scientific and Numeric <https://www.python.org/about/apps/>
- Data Science
- Machine Learning - scikit-learn and TensorFlow
- Data Analysis - Matplotlib, Seaborn
- Business Application

Byte Code – Byte Code represents the fixed set of instruction created by Python developers representing all type of operations like arithmetic operations, comparison operation, memory related operation etc.

The size of each byte code instruction is 1 byte or 8 bits.

We can find byte code instruction in the .pyc file.

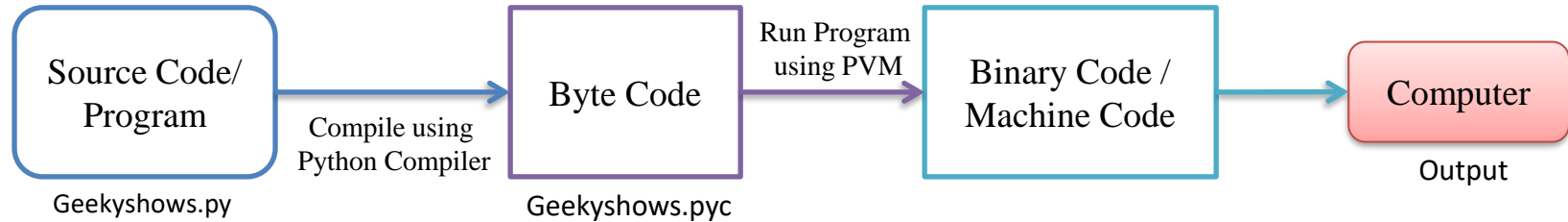
Python Compiler – A Python Compiler converts the program source code into byte code.

Type of Python Compilers :-

- CPython
- Jpython/ Jython
- PyPy
- RubyPython
- IronPython
- StacklessPython
- Pythonxy
- AnacondaPython



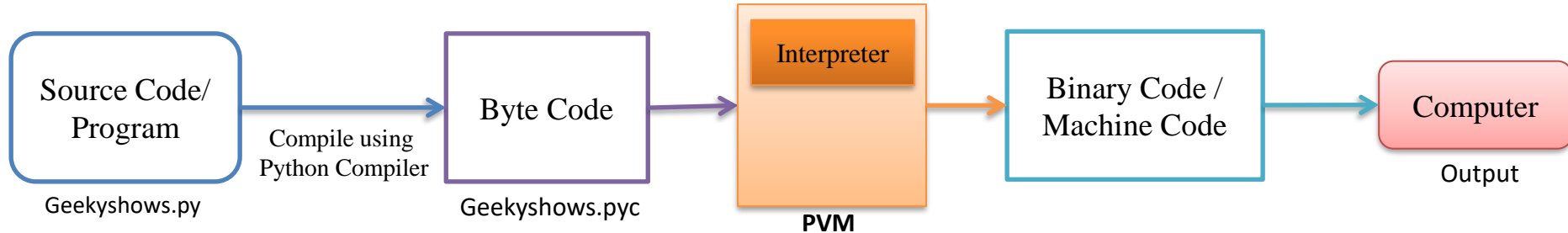
- Write Source Code / Program
- Compile the Program using Python Compiler
- Compiler Converts the Python Program into byte Code
- Computer/Machine Can not understand Byte Code so we convert it into Machine Code using PVM
- PVM uses an interpreter which understands the byte code and convert it into machine code
- Machine Code instructions are then executed by the processor and results are displayed



# Python Virtual Machine

Python Virtual Machine (PVM) is a program which provides programming environment. The role of PVM is to convert the byte code instructions into machine code so the computer can execute those machine code instructions and display the output.

Interpreter converts the byte code into machine code and sends that machine code to the computer processor for execution.



# Identifier

An identifier is a name having a few letters, numbers and special characters \_ (underscore).

It should always start with a non-numeric character.

It is used to identify a variable, function, symbolic constant, class etc.

Ex : -

X2

PI

Sigma

matadd

full\_name

- Python is case sensitive programming language.

d is not equal to D

t is not equal to T

rahul is not equal to Rahul

rahul is not equal to RAHUL

# Keywords or Reserved Words

Python language uses the following keywords which are not available to users to use them as identifiers.

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

# Constants

A constant is an identifier whose value cannot be changed throughout the execution of a program whereas the variable value keeps on changing.

There are no constants in Python, the way they exist in C and Java.

In Python, It is not possible to define constant whose value can not be changed.

In Python, Constants are usually defined on a module level and written in all capital letters with underscores separating words but remember its value can be changed.

Ex:-

PI

TOTAL

MIN\_VALUE

# Variable

In C, Java or some other programming languages, a variable is an identifier or a name, connected to memory location.

**a = 30**

a



12114

b = 10

b



12115

c = 20

c



12117

y = a

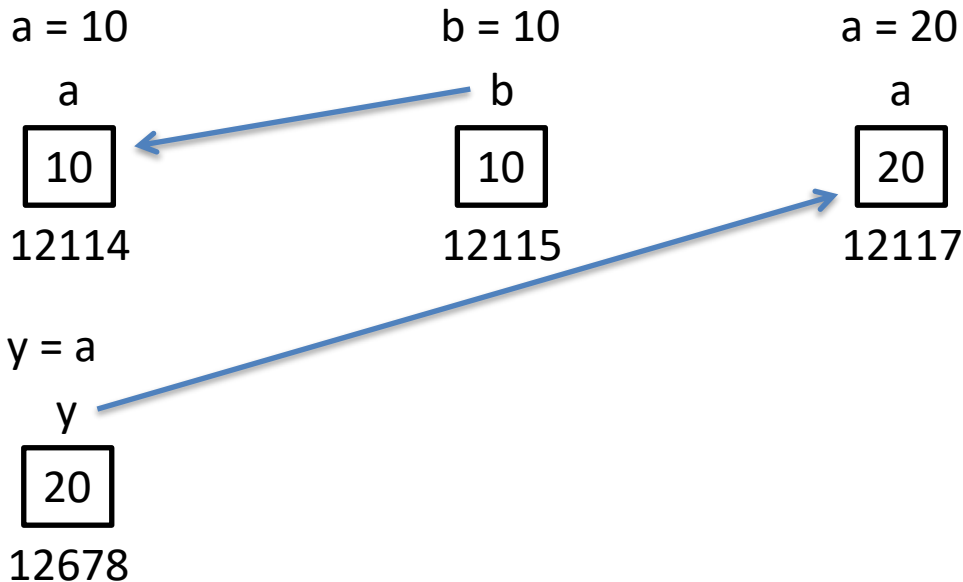
y



12678

# Variable

In Python, a variable is considered as tag that is tied to some value. Python considers value as objects.





# Variable

In Python, a variable is considered as tag that is tied to some value. Python considers value as objects.

a = 10

a

10


12114

a = 20

a

20

12115



Since value 10 becomes unreferenced object,  
it is removed by garbage collector.

# Rules

- Every variable name should start with alphabets or underscore (\_).
- No spaces are allowed in variable declaration.
- Except underscore ( \_ ) no other special symbol are allowed in the middle of the variable declaration
- A variable is written with a combination of letters, numbers and special characters \_ (underscore)
- No Reserved keyword

# Examples

## Do

- A
- a
- name
- name15
- \_city
- Full\_name
- FullName

## Don't

- and
- 15name
- \$city
- Full\$Name
- Full Name

# Data Type

Datatype represents the type of data stored into a variable or memory.

Type of Data type :-

- Built-in Data type
- User Defined Data type

# **Built-in Datatype**

These datatypes are provided by Python Language.

Following are the built-in data type:-

- None Type
- Numeric Types
- Sequences
- Sets
- Mappings

# User Defined Data type

- Array
- Class
- Module

# None Type

None datatype represents an object that doesn't contain any value.

# Numeric Type / Number

Following are the Numeric Data type:-

Int

Float

Complex



# Numeric Type / Number

Int – The int datatype represents an integer number. An integer number without any decimal point or fraction part. In Python, It is possible to store very large integer number as there is no limit for the size of an int datatype.

Ex:-

20, 10, -50, -1002

y = 10

pin\_code = 564512

int type variable



pin\_code = 564512

int value

# Numeric Type / Number

Float – The float data type represents floating point numbers. A floating point number is a number that contains a decimal point.

Ex:-

25.56, 10.5, -45.69, -0.8

price = 25.56

run\_rate = -0.8

value = 5.1e5

$5.1 \times 10^5$

float type variable

run\_rate = -0.8

float value

**5.1e5**

It's scientific notation where e or E represents exponentiation which represents the power of 10

# Numeric Type / Number

Complex – A complex number is a number that is written in the form of  $a + bj$  or  $a + bJ$ . Where,

$a$  = Real Part of the number

$b$  = Imaginary part of the number

$j$  or  $J$  = Square root value of  $-1$

$a$  and  $b$  may contain integer or float number.

Ex:-  $5+7j$ ,  $0.8+2j$

$com = 5+7j$

complex type variable



$com = 5+7j$

complex number



# Bool type

The bool datatype represents boolean value True or False. Python internally represents True as 1 and False as 0.

Ex:- True, False

$\text{True} + \text{True} = 2$

$\text{True} - \text{False} = 1$

# Sequence Type

Following are sequence type:-

String

List

Tuple

Range

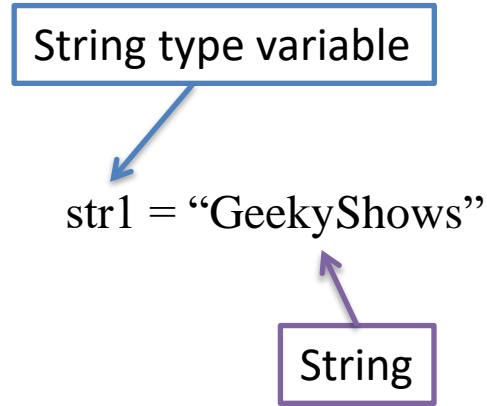
# Sequence Type

String – String represents group of characters. Strings are enclosed in double quotes or single quotes.

Ex:- “Hello”, “GeekyShows”, ‘Rahul’

str1 = “GeekyShows”

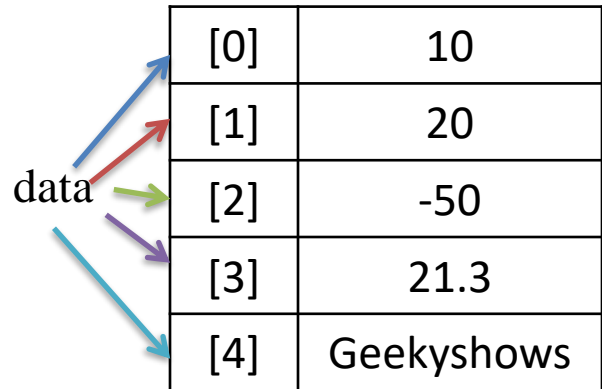
str1 = ‘GeekyShows’



# Sequence Type

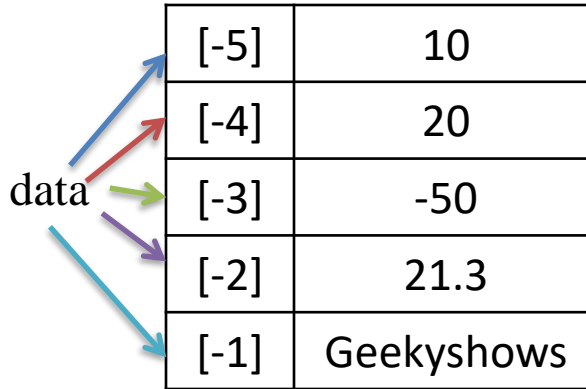
List – A list represents a group of elements. A list can store different types of elements which can be modified. Lists are dynamic which means size is not fixed. Lists are represented using square bracket [ ].

Ex:- `data = [10, 20, -50, 21.3, 'Geekyshows']`



A diagram illustrating list indexing. On the left, the word 'data' is written. Five colored arrows point from 'data' to the index column of a table: a blue arrow to [0], a red arrow to [1], a green arrow to [2], a purple arrow to [3], and a cyan arrow to [4].

[0]	10
[1]	20
[2]	-50
[3]	21.3
[4]	Geekyshows



A diagram illustrating list indexing. On the left, the word 'data' is written. Five colored arrows point from 'data' to the index column of a table: a blue arrow to [-5], a red arrow to [-4], a green arrow to [-3], a purple arrow to [-2], and a cyan arrow to [-1].

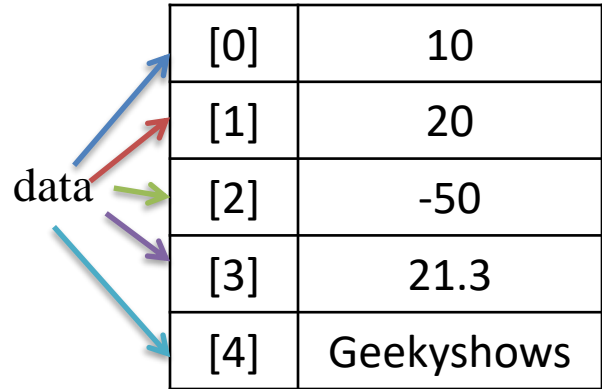
[-5]	10
[-4]	20
[-3]	-50
[-2]	21.3
[-1]	Geekyshows

`data[1] = 40`

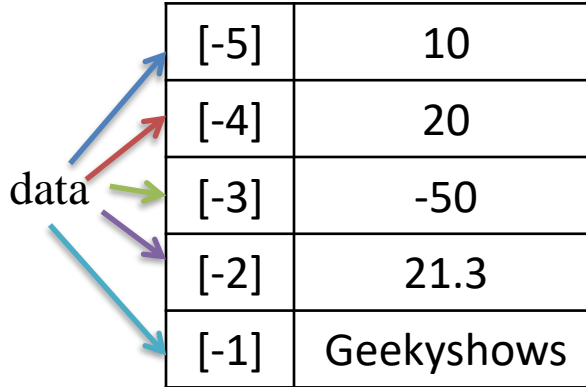
# Sequence Type

Tuple – A tuple contains a group of elements which can be different types. It is similar to List but Tuples are read-only which means we can not modify its element. Tuples are represented using parentheses ( ).

Ex:- data = (10, 20, -50, 21.3, 'Geekyshows')



[0]	10
[1]	20
[2]	-50
[3]	21.3
[4]	Geekyshows



[-5]	10
[-4]	20
[-3]	-50
[-2]	21.3
[-1]	Geekyshows

data[1] = 40



# Sequence Type

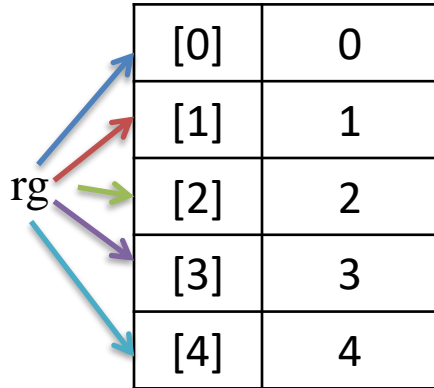
Range – Range represents a sequence of numbers. The numbers in the range are not modifiable.

Ex:- `rg = range(5)`

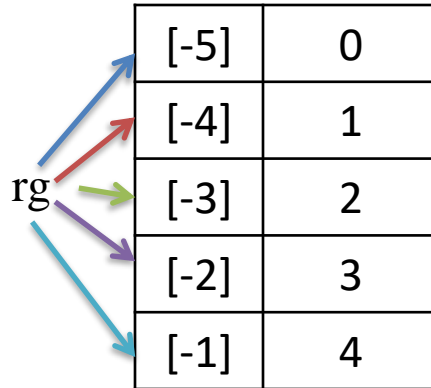
`rg = range(10, 20, 2)`

0 1 2 3 4

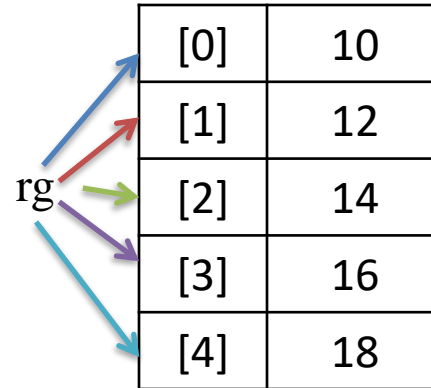
10 12 14 16 18



[0]	0
[1]	1
[2]	2
[3]	3
[4]	4



[-5]	0
[-4]	1
[-3]	2
[-2]	3
[-1]	4



[0]	10
[1]	12
[2]	14
[3]	16
[4]	18

# Set Type

A set is an unordered collection of elements much like a set in mathematics.

The order of elements is not maintained in the sets. It means the elements may not appear in the same order as they are entered into the set.

A set does not accept duplicate elements.

Sets are unordered so we can not access its element using index.

`data[0] = 10`

Sets are represented using curly brackets { }.

Ex:-

`data = {10, 20, 30, "GeekyShows", "Raj", 40}`

`data = {10, 20, 30, "GeekyShows", "Raj", 40, 10, 20}`

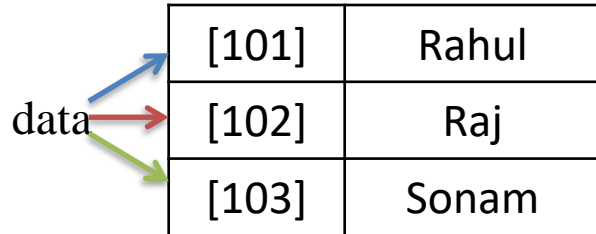
# Mapping Type/ dict / Dictionary

A map represents a group of elements in the form of key value pairs.

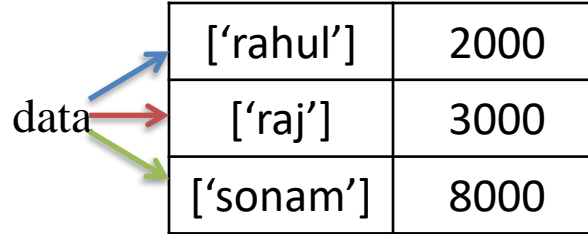
Ex:-

```
data = {101: 'Rahul', 102: 'Raj', 103: 'Sonam' }
```

```
data = {'rahul':2000, 'raj':3000, 'sonam':8000, }
```



[101]	Rahul
[102]	Raj
[103]	Sonam



['rahul']	2000
['raj']	3000
['sonam']	8000

# Character

There is no concept of char data type in Python to represent individual character.

# Operators

An operator is a symbol that performs an operation.

- Arithmetic Operators
- Relational Operators / Comparison Operators
- Logical Operators
- Assignment Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

# Arithmetic Operators

Arithmetic Operators are used to perform basic arithmetic operations like addition, subtraction, division etc.

Operators	Meaning	Example	Result
+	Addition	$4 + 2$	6
-	Subtraction	$4 - 2$	2
*	Multiplication	$4 * 2$	8
/	Division	$4 / 2$	2
%	Modulus operator to get remainder in integer division	$5 \% 2$	1
**	Exponent	$5**2 = 5^2$	25
//	Integer Division/ Floor Division	$5//2$ $-5//2$	2 -3

# Relational/ Comparison Operators

Relational operators are used to compare the value of operands (expressions) to produce a logical value. A logical value is either True or False.

Operators	Meaning	Example	Result
<	Less than	5<2	False
>	Greater than	5>2	True
<=	Less than or equal to	5<=2	False
>=	Greater than or equal to	5>=2	True
==	Equal to	5==2	False
!=	Not equal to	5!=2	True

# Logical Operators

Logical operators are used to connect more relational operations to form a complex expression called logical expression. A value obtained by evaluating a logical expression is always logical, i.e. either True or False.

Operator	Meaning	Example	Result
and	Logical and	$(5 < 2)$ and $(5 > 3)$	False
or	Logical or	$(5 < 2)$ or $(5 > 3)$	True
not	Logical not	not $(5 < 2)$	True



# and

Operand 1	Operand 2	Result
True	True	True
True	False	False
False	True	False
False	False	False
True	Expression	Expression
False	Expression	False

True and Expression1 and Expression2 = Expression2

False and Expression1 and Expression2 = False

# or

Operand 1	Operand 2	Result
True	True	True
True	False	True
False	True	True
False	False	False
True	Expression	True
False	Expression	Expression

True or Expression1 or Expression2 = True

False or Expression1 or Expression2 = Expression1

# not

Operand	Result
False	True
True	False

# Assignment Operators

Assignment operators are used to perform arithmetic operations while assigning a value to a variable.

Operator	Example	Equivalent Expression (m=15)	Result
=	y = a+b	y = 10 + 20	30
+=	m +=10	m = m+10	25
-=	m -=10	m = m-10	5
*=	m *=10	m = m*10	150
/=	m /=10	m = m/10	1.5
%=	m %=10	m = m%10	5
**=	m **=2	m = m**2 or $m = m^2$	225
//=	m //=10	m = m//10	1

# Bitwise Operators

Bitwise operators are used to perform operations at binary digit level. These operators are not commonly used and are used only in special applications where optimized use of storage is required.

Operator	Meaning
&	Bitwise AND
	Bitwise OR
^	Bitwise exclusive OR / Bitwise XOR
~	Bitwise inversion (one's complement)
<<	Shifts the bits to left / Bitwise Left Shift
>>	Shifts the bits to right / Bitwise Right Shift

# Bitwise AND &

Operand 1	Operand 2	Result (operand1 & operand2)
True 1	True 1	True 1
True 1	False 0	False 0
False 0	True 1	False 0
False 0	False 0	False 0

a = 10      0 0 0 0 1 0 1 0  
b = 15      0 0 0 0 1 1 1 1

# Bitwise OR |

Operand 1	Operand 2	Result (operand1   operand2)
True 1	True 1	True 1
True 1	False 0	True 1
False 0	True 1	True 1
False 0	False 0	False 0

a = 10      0 0 0 0 1 0 1 0  
b = 15      0 0 0 0 1 1 1 1

# Bitwise XOR ^

Operand 1	Operand 2	Result (operand1 ^ operand2)
True 1	True 1	False 0
True 1	False 0	True 1
False 0	True 1	True 1
False 0	False 0	False 0

a = 10      0 0 0 0 1 0 1 0  
b = 15      0 0 0 0 1 1 1 1



# Bitwise NOT ~

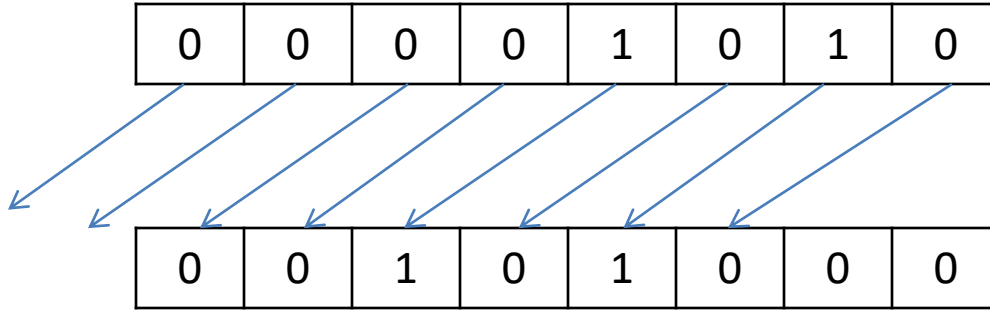
Operand	Result (~ operand)
True 1	False 0
False 0	True 1

a = 10

0 0 0 0 1 0 1 0

# Bitwise Left Shift <<

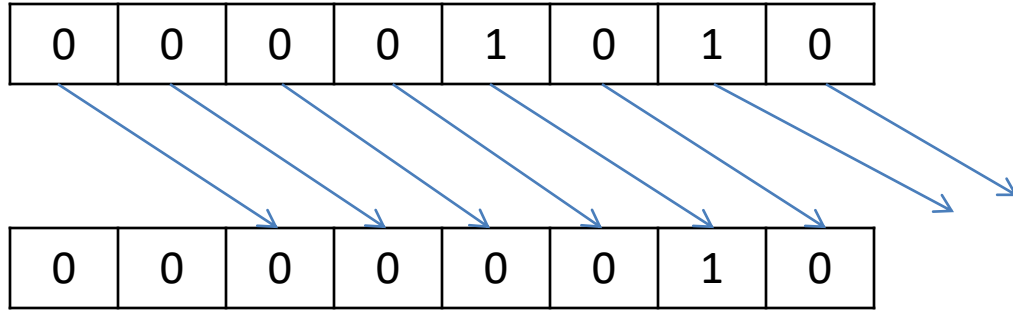
$a = 10$



$a \ll 2$

# Bitwise Right Shift >>

$a = 10$



$a \gg 2$

# Membership Operators

The membership operators are useful to test for membership in a sequence such as string, lists, tuples and dictionaries.

There are two type of Membership operator:-

- in
- not in

# in

This operators is used to find an element in the specified sequence.

It returns True if element is found in the specified sequence else it returns False.

Ex:-

```
st1 = "Welcome to geekyshows"
```

```
"to" in st1  True
```

```
st2 = "Welcome top geekyshows"
```

```
"to" in st2  True
```

```
st3 = "Welcome to geekyshows"
```

```
"subs" in st3  False
```

# not in

This operators works in reverse manner for in operator.

It returns True if element is not found in the specified sequence and if element is found, then it returns False.

Ex:-

```
st1 = "Welcome to geekyshows"
```

```
"subs" not in st1    True
```

```
st2 = "Welcome to geekyshows"
```

```
"to" not in st2      False
```

```
st3 = "Welcome top geekyshows"
```

```
"to" not in st3      False
```

# Identity Operators

The identity operators compare the memory locations of two objects. Hence, it is possible to know whether two objects are same or not.

There are two types of Identity operator:-

- is
- is not

# is

This operator is used to compare whether two objects are same or not.

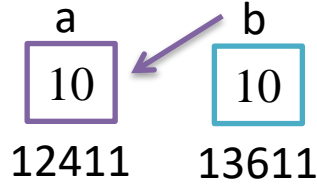
It returns True if memory location of two objects are same else it returns False.

Ex:-

a = 10

b = 10

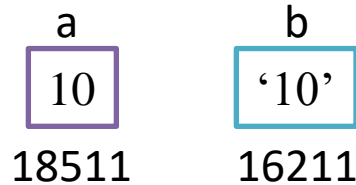
a is b    True



a = 10

b = '10'

a is b    False





# is not

This operator works in reverse manner for *is* operator.

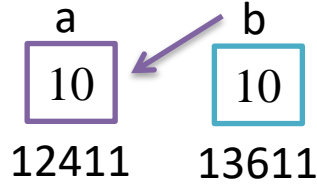
It returns True if memory location of two objects are not same and if they are same it returns False.

Ex:-

a = 10

b = 10

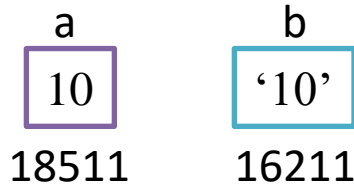
a is not b    False



a = 10

b = '10'

a is not b    True



# **Operator Precedence and Associativity**

The computer scans an expression which contains the operators from left to right and performs only one operation at a time. The expression will be scanned many times to produce the result. The order in which various operations are performed is known as hierarchy of operations or operator precedence. Some of the operators of the same level of precedence are evaluated from left to right or right to left. This is referred to as associativity.

Order	Operator	Meaning
1	()	Parentheses
2	**	Exponentiation
3	+, -, ~	Unary Plus, Unary Minus, Bitwise Not
4	*, /, //, %	Multiplication, Division, Floor Division, Modulus
5	+, -	Addition, Subtraction
6	<<, >>	Bitwise Left Shift, Bitwise Right Shift
7	&	Bitwise AND
8	^	Bitwise XOR
9	>, >=, <, <=, ==, !=	Relational Operators
10	=, %=, /=, //=, -=, +=, *=, **=	Assignment Operators
11	is, is not	Identity Operators
12	in, not in	Membership Operators
13	not	Logical NOT
14	or	Logical OR
15	and	Logical AND

- Parentheses
- Exponentiation
- Multiplication, Division, Modulus and Floor Division
- Addition and Subtraction
- Assignment

value = (1+1)\*2\*\*4//3+4-1  
2\*2\*\*4//3+4-1  
2\*16//3+4-1  
32//3+4-1  
10+4-1  
14-1  
13

# Type Conversion

Converting one data type into another data type is called *Type Conversion*.

Type of *Type Conversion*:-

- Implicit Type Conversion
- Explicit Type Conversion

# Implicit Type Conversion

In the Implicit type conversion, python automatically converts one data type into another data type.

Ex:-

```
a = 5
```

```
b = 2
```

```
value = a / b
```

```
print(value)
```

```
print(type(value))
```

# Explicit Type Conversion

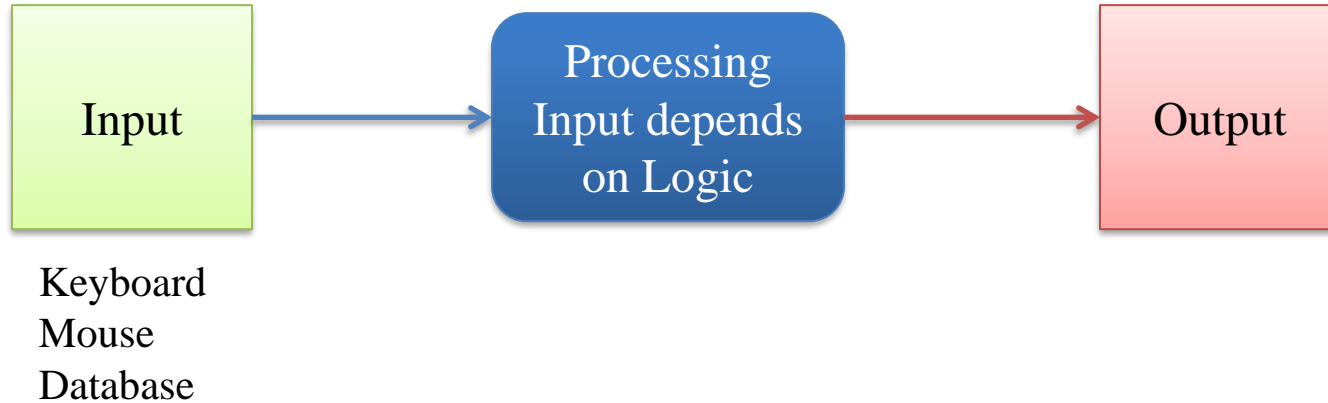
In the Cast/Explicit Type Conversion, Programmer converts one data type into another data type.

- `int (n)`
- `float (n)`
- `complex (n)`
- `complex (x, y)` where x is real part and y is imaginary part
- `str (n)`
- `list(n)`
- `tuple(n)`
- `bin (n)`
- `oct (n)`
- `hex (n )`

# Input and Output

Input - The data given to the computer is called input.

Output – The results returned by the computer are called output.





# Output Statements

`print ( )` Function - The `print()` function is used to print the specified message to the output screen/device. The message can be a string, or any other object.

Syntax:- `print(objects, sep='character', end='character', file=sys.stdout, flush=False)`

`sep` - Separate the objects by given character. Character can be any string. Default is ' ' or can write none.

`end` – It indicates ending character for the line. Default is '\n' or can write none.

`file` - An object with a write method. Default is `sys.stdout` or can write none.

`flush` - A Boolean, specifying if the output is flushed (True) or buffered (False). Default is False

# Output Statements

`print ( )` – This function is used to display a blank line.

`print(“string”)` - When a string is passed to the function, the string is displayed as it is.

Ex:-

```
print(“Welcome to Geeky Shows”)
```

```
print(‘Welcome to Geeky Shows’)
```

```
print(“Like”, “Share”, “Subscribe”)
```

```
print(10)
```

```
print(“Welcome”)
```

```
print(“to”)
```

```
print(“Geeky Shows”)
```

# Output Statements

`print(object)` - We can pass objects like list, tuples and dictionaries to display the elements of those objects.

Ex:-

```
data = [10, 20, -50, 21.3, 'Geekyshows']
```

```
print(data)
```



List

# Output Statements

`print("string" sep='')` – It separates string with given sep character. Character can be any string. Default is ' ' or can write none.

Ex:-

```
print("Like", "Share", "Subscribe", sep='')
```

```
print("Like", "Share", "Subscribe", sep='***')
```

# Output Statements

`print("string" end='')` – When ending character is passed. It prints given character at the end. Default is `'\n'` or can write none.

Ex:-

```
print("Welcome", end='\n')
```

```
print("Welcome", end='')
```

```
print("to", end='')
```

```
print("GeekyShows")
```

```
print("Welcome", end='\t')
```

```
print("to", end='\t')
```

```
print("GeekyShows")
```

# Output Statements

`print(variable list)` – This is used to display the value of a variable or a list of variable.

Ex:-

```
a = 10
```

```
print(a)
```


Output: 10

```
x = 20
```

```
y = 30
```

```
print(x, y)
```

Output: 20 30



List of variable's value in the output screen, are separated by a space by default. we can change this by sep

```
print(x, y, sep=',')
```

Output: 20, 30

# Output Statements

print(“String”, variable list) – This is used to display the string along with variable.

Ex:-

```
m = 40
```

```
print(“Value: ”, m)
```

Output: Value: 40

```
name = “Rahul”
```

```
age = 62
```

```
print(“My Name is ”, name, “and My age is”, age)
```

Output: My Name is **Rahul** and My age is **62**

# Input Statements

`input( )` – This function is used to accept input from keyboard.

This function will stop the program flow until the user gives an input and end the input with the return key.

Whatever user gives as input, input function convert it into a string. If user enters an integer value still `input()` function convert it into a string.

So if you need an integer you have to use type conversion.

Syntax:- `input([prompt])`

`prompt` is a string or message, representing a default message before input. It is optional

Ex:-

```
name = input( )
```

```
name = input("Your Name: ")
```

```
mobile = input("Enter Your Mobile Number: ")
```



# Input Statements

Whatever user gives as input, input function convert it into a string. If user enters an integer value still input() function convert it into a string.

So if you need an integer you have to use type conversion.

Ex:-

```
mobile = input("Enter Your Mobile Number: ")
```

```
mb = int(mobile)
```

```
mobile = int ( input ("Enter Your Mobile Number: ") )
```

```
price = float ( input ("Total Price: ") )
```

```
mobile = complex ( input ("Enter Complex Number: ") )
```

# Escape Sequence

**Escape sequences** – Escape sequences are control character used to move the cursor and print characters such as ‘, “. \ and so on.

Escape Sequence	Meaning
\\	Backslash
\'	Single Quote
\"	Double Quote

Escape Sequence	Meaning
\a	Bell
\b	Backspace
\f	Formfeed
\n	NewLine
\r	Carriage Return
\t	Horizontal Tab
\v	Vertical Tab
\newline	Backslash and NewLine Ignored

# Comment

Comments are non-executable statements. A Comment is used to describe the feature of a program. Comment helps to understand our program, not only ourselves but also other programmer.

There are two type of programs:-

- Single Line Comment
- Multi line Comment

# Single Line Comment

These comments start with a hash symbol (#).

Ex:-

```
# I am single Line Comment
```

```
# This is my first Python Program
```

```
# Adding two numbers
```

# Multi Line Comment

There is no concept of multi line comment in python but we can create string starting and ending with triple double quotes (""" ) or triple single quotes (''' ) which can be used as block of comments.

Since strings are not assigned to any variable, then they are removed from memory by the garbage collector and hence these can be used as comments.

It is not recommended to use triple double quotes or triple single quotes for writing comments as it internally occupy memory and would waste time of the interpreter since the interpreter has to check them.

Ex:-

"""

Comment Line 1

Comment Line 2

Comment Line 3

"""

'''

Comment Line 1

Comment Line 2

Comment Line 3

'''

# If Statement

It is used to execute an instruction or block of instructions only if a condition is fulfilled.

Syntax: -

if (condition):

statement

Rest of the Code

First the condition is tested, If the condition is True

then the statements given after  
colon (:) are executed

if (condition):

statement1

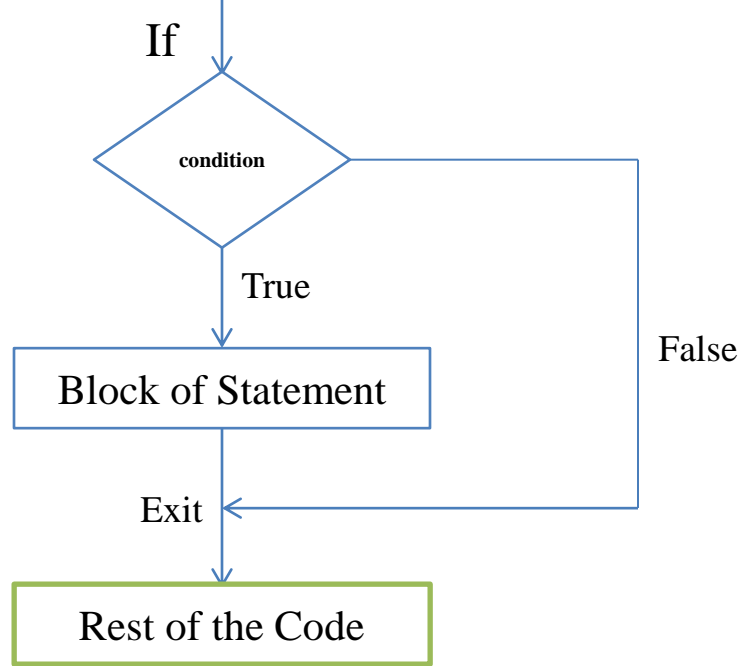
statement2

Rest of the Code

Block of statement/ Group of statements/ Suite

If there is single statement it can be written in one line.

Ex:- if (condition): Statement



# Nested If Statement

Syntax :

```
if (condition):
```

```
    block of statements
```

```
    if(condition):
```

```
        block of statements
```

```
    if(condition):
```

```
        block of statements
```

```
if(condition):
```

```
    block of statements
```

```
Rest of the code
```



# If Statement with Logical Operator

if ( (condition1) and (condition2) ):

Statement

Rest of the Code

if ( (condition1) and (condition2) ):

Block of Statements

Rest of the Code

if ( (condition1) or (condition2) ):

Statement

Rest of the Code

# Indentation

Indentation refers to spaces that are used in the beginning of a statement. By default python puts 4 spaces but it can be changed by programmers.

```
if (condition):  
    Statement  
if(condition):  
    statement 1  
    statement 2  
if(condition):  
    Statement  
if(condition):  
    Statement 1  
    Statement 2  
Rest of the code
```

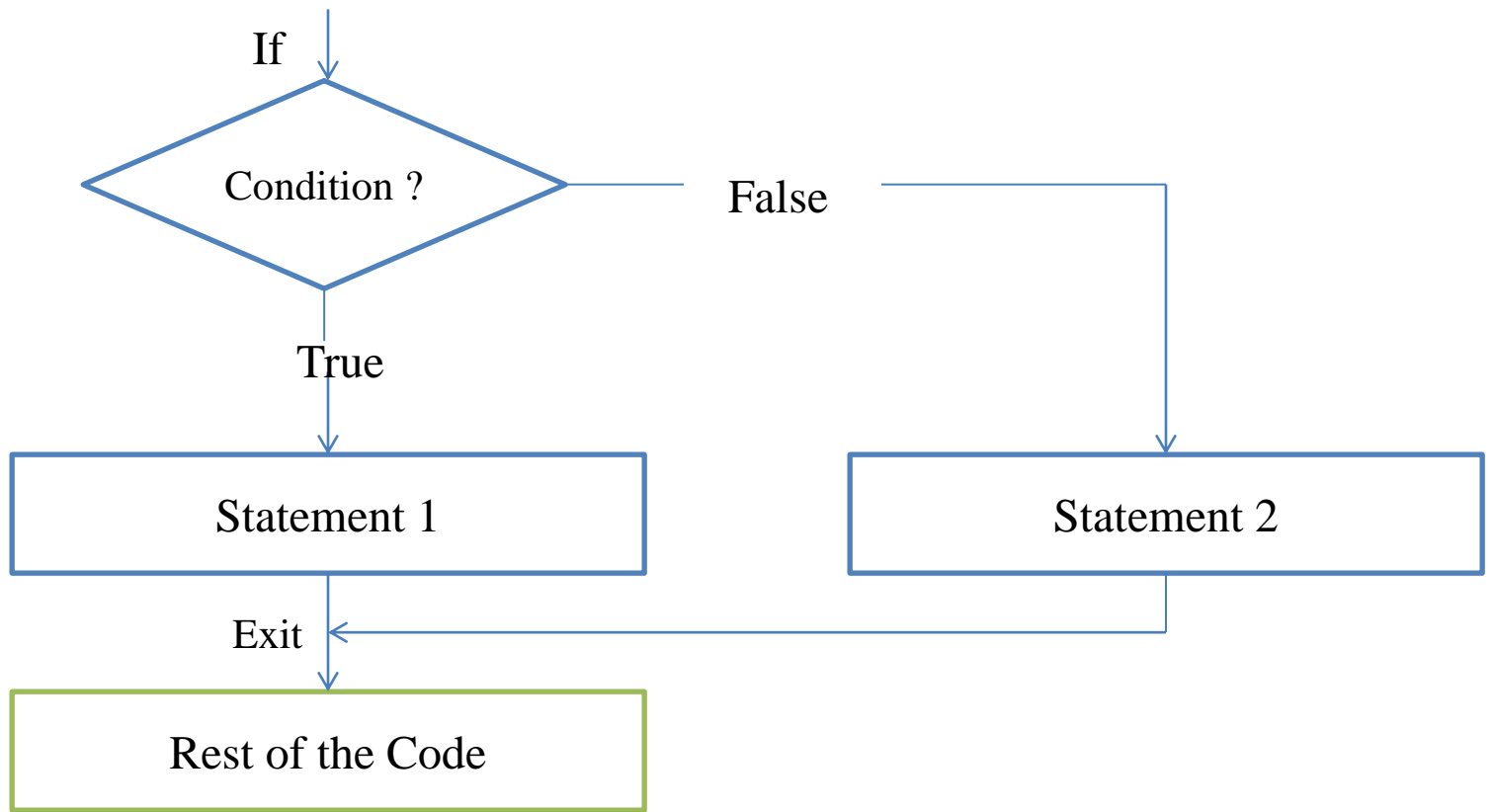
# If Else Statement

if... else statement is used when a different sequence of instructions is to be executed depending on the logical value(True/False) of the condition evaluated.

Syntax: -

```
if(condition):  
    Statement 1  
else:  
    Statement 2  
Rest of the Code
```

```
if(condition):  
    Statement 1  
    Statement 2  
else:  
    Statement 3  
    Statement 4  
Rest of the Code
```



# **Nested If Else Statement**

In nested if.... else statement, an entire if... else construct is written within either the body of the if statement or the body of an else statement.

# Nested If Else Statement

```
if(condition_1):  
    if(condition_2):  
        Statement 1  
    else:  
        Statement 2  
else:  
    Statement 3  
Rest of the Code
```

```
if(condition_1):  
    if(condition_2):  
        Statement 1  
    else:  
        Statement 2  
else:  
    if(condition_3):  
        Statement 3  
    else:  
        Statement 4  
Rest of the Code
```

# if elif Statement

To show a multi-way decision based on several conditions, we use if elif statement.

```
if (condition_1):
```

```
    Statement 1
```

```
elif (condition_2):
```

```
    Statement 2
```

```
elif (condition_3):
```

```
    Statement 3
```

```
elif (condition_n):
```

```
    Statement n
```

```
Rest of the Code
```

# if elif else Statement

To show a multi-way decision based on several conditions, we use if elif else statement.

```
if (condition_1):
```

```
    Statement 1
```

```
elif (condition_2):
```

```
    Statement 2
```

```
elif (condition_3):
```

```
    Statement 3
```

```
elif (condition_n):
```

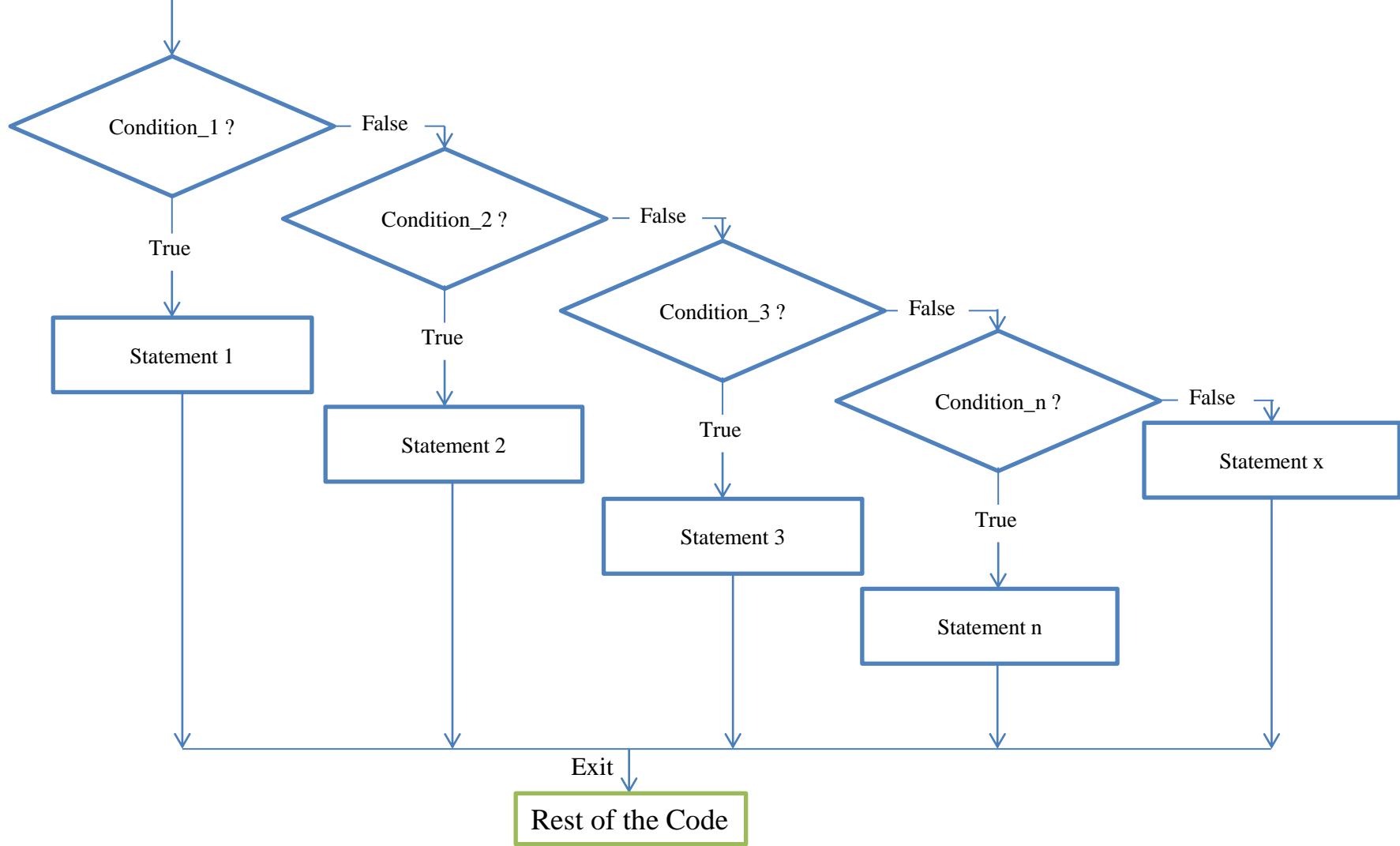
```
    Statement n
```

```
else:
```

```
    Statements x
```

```
Rest of the Code
```





# **Loop Control Statements**

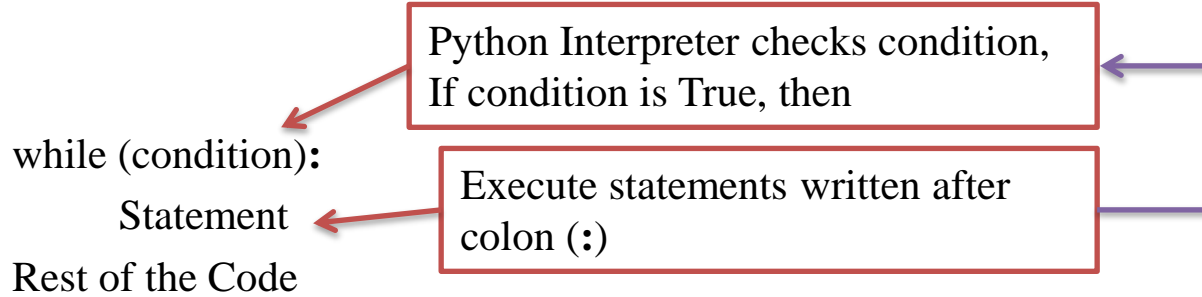
Loop control statements are used when a section of code may either be executed a fixed number of times, or while some condition is true.

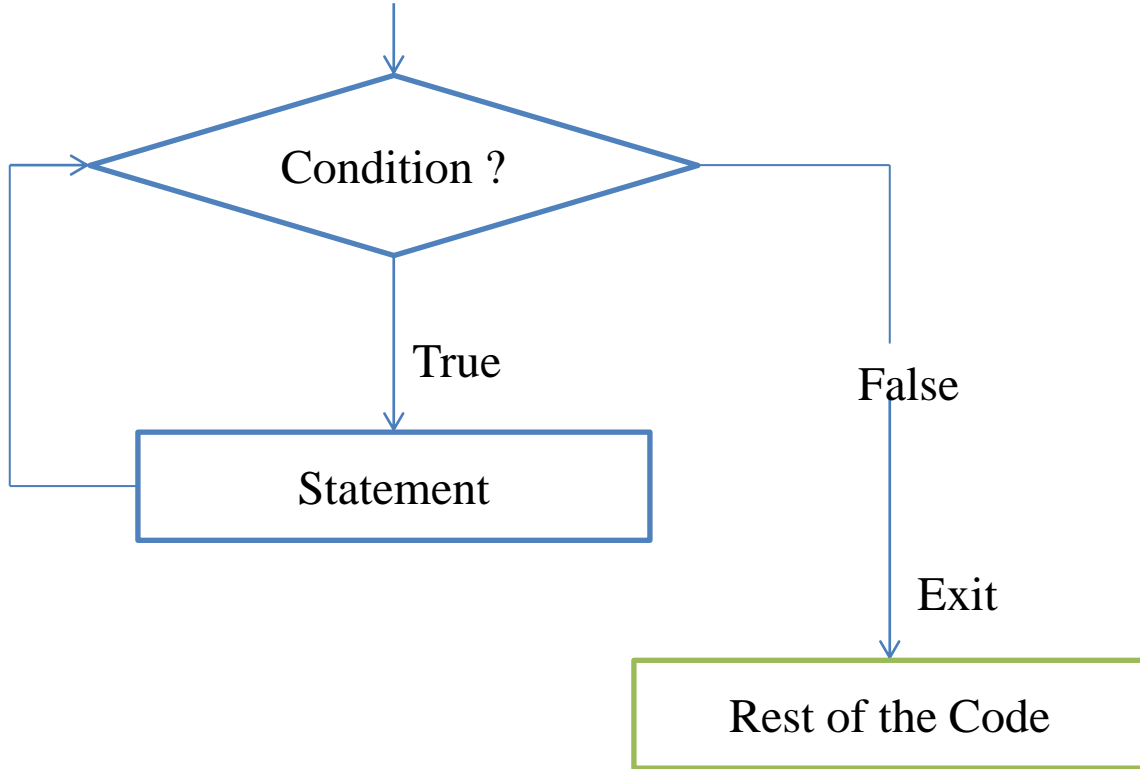
- While
- For

# while Loop

The while loop keeps repeating an action until an associated condition returns false.

Syntax:



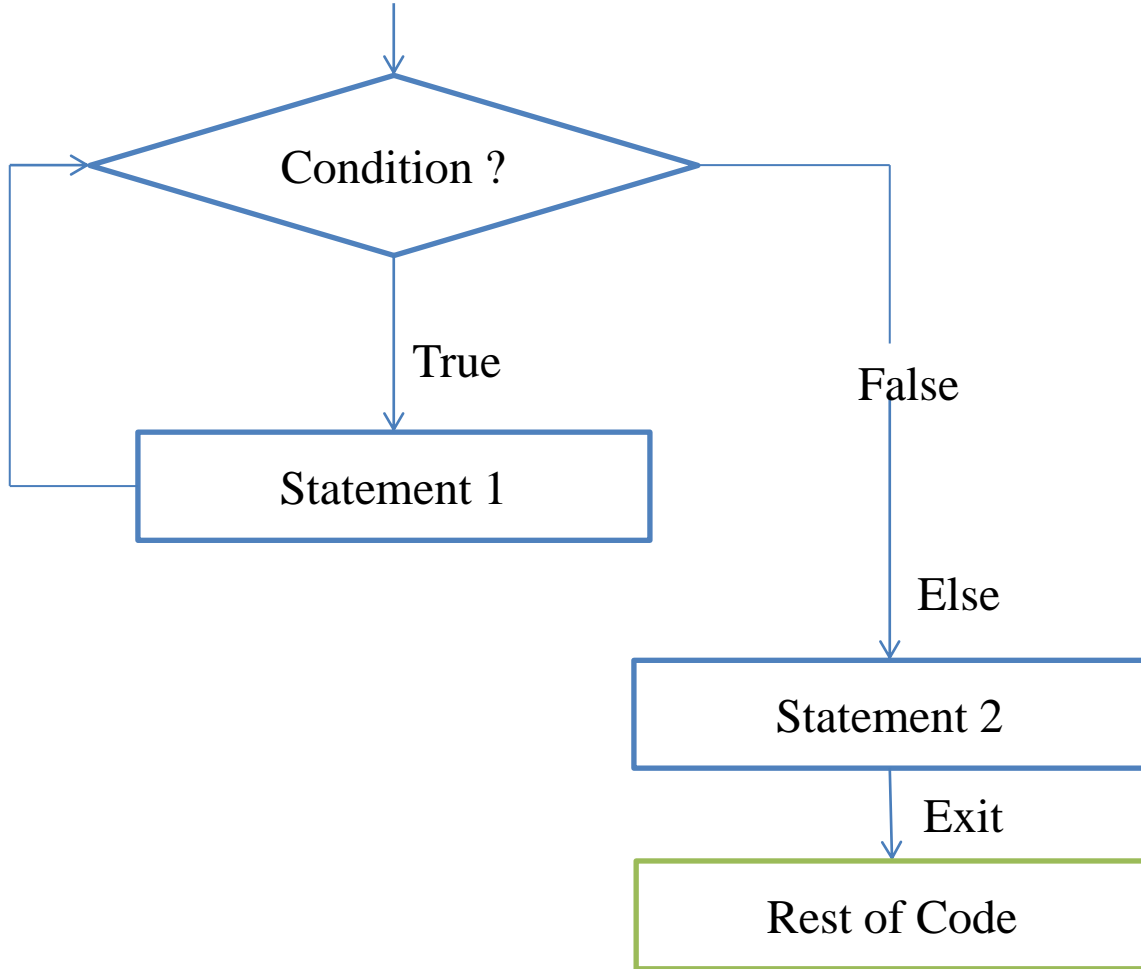


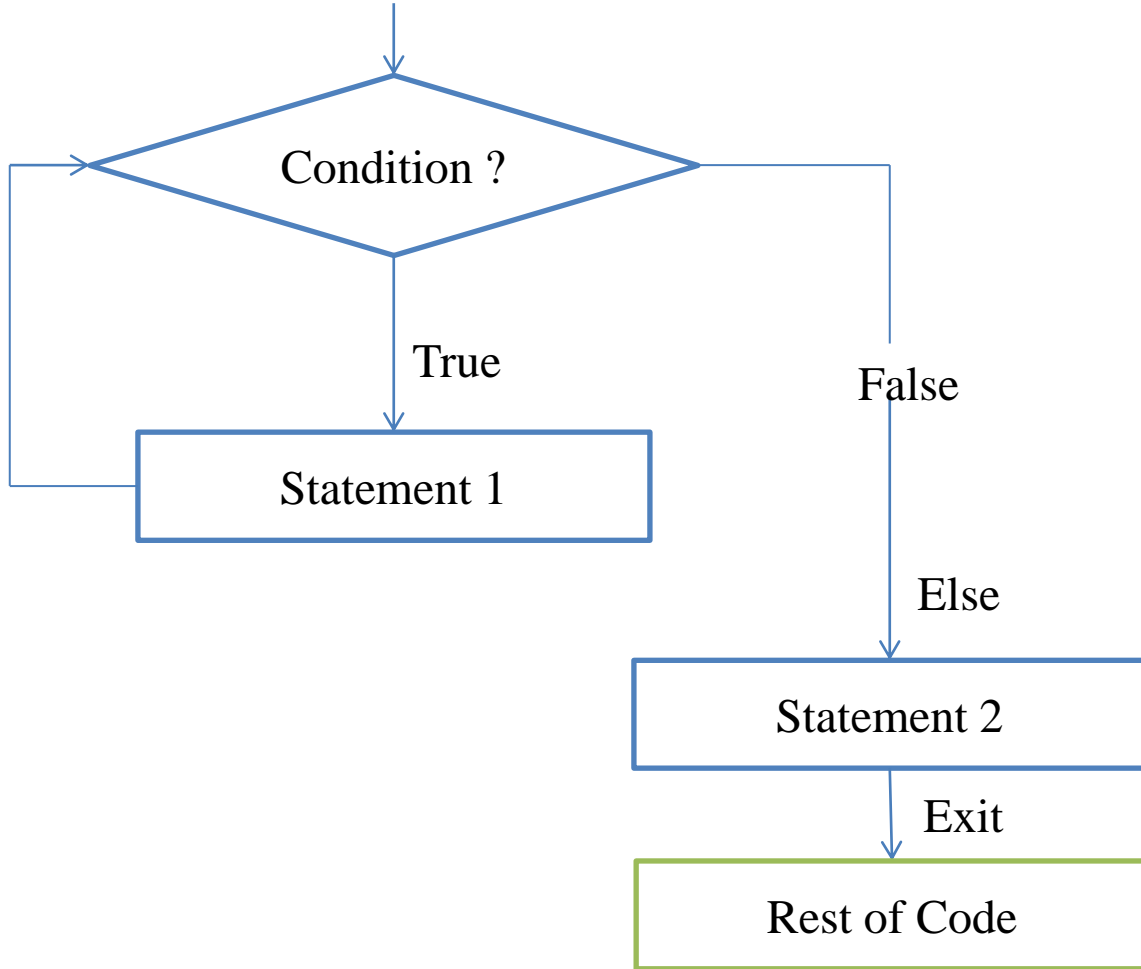
# while Loop with else

This repeatedly tests the condition and, if it is True, executes the Statement 1; if the condition is False (which may be the first time it is tested) the Statement 2 of the else clause, is executed and the loop terminates. The else suite will be always executed irrespective of the statements in the loop are executed or not.

Syntax:

```
while (condition):  
    Statement 1  
else:  
    Statement 2  
Rest of the Code
```





# Infinite while Loop

Syntax:

```
while (True):
```

```
    Statement
```

```
Rest of the Code
```

```
while (True):
```

```
    Statement
```

```
    if(condition):
```

```
        break
```

```
Rest of the Code
```



# Nested While Loop

while(condition):

    Statements

        while(condition):

            Statements

        Statements

Rest of Code

# range() Function

range() function is used to generate a sequence of integers starting from 0 by default, and increments by 1 by default, till j-1.

Syntax:-

range(start, stop, stepsize)

Start – Starting position. If we do not mention start by default it's 0

\*Stop – Ending position. The range of integers stops one element prior to stop. If stop is j then it will stop at exact j-1

Stepsize – Increment by stepsize. If we do not mention start by default it's 1

# range() Function

Syntax:- range( j )                      0, 1, 2, 3, 4,....., j-1

Ex:- range(10)                              0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Syntax:- range(i, j)                      i, i+1, i+2, i+3,....., j-1

Ex:- range(1, 10)                              1, 2, 3, 4, 5, 6, 7, 8, 9

Syntax: - range(i, j, k)                      i, i+k, i+2k, i+3k, i+4k,....., j-1

range(1, 10, 2)                              1, 3, 5, 7, 9

range(-1, -10, -2)      -1 -3 -5 -7 -9

range(10, 0, -1)      10 9 8 7 6 5 4 3 2 1

# **Rules:-**

- All argument must be integers, whether its positive or negative
- You can not pass a string or float number or any other type in a start, stop and stepsize.
- The stepsize value should not be zero.

# for Loop

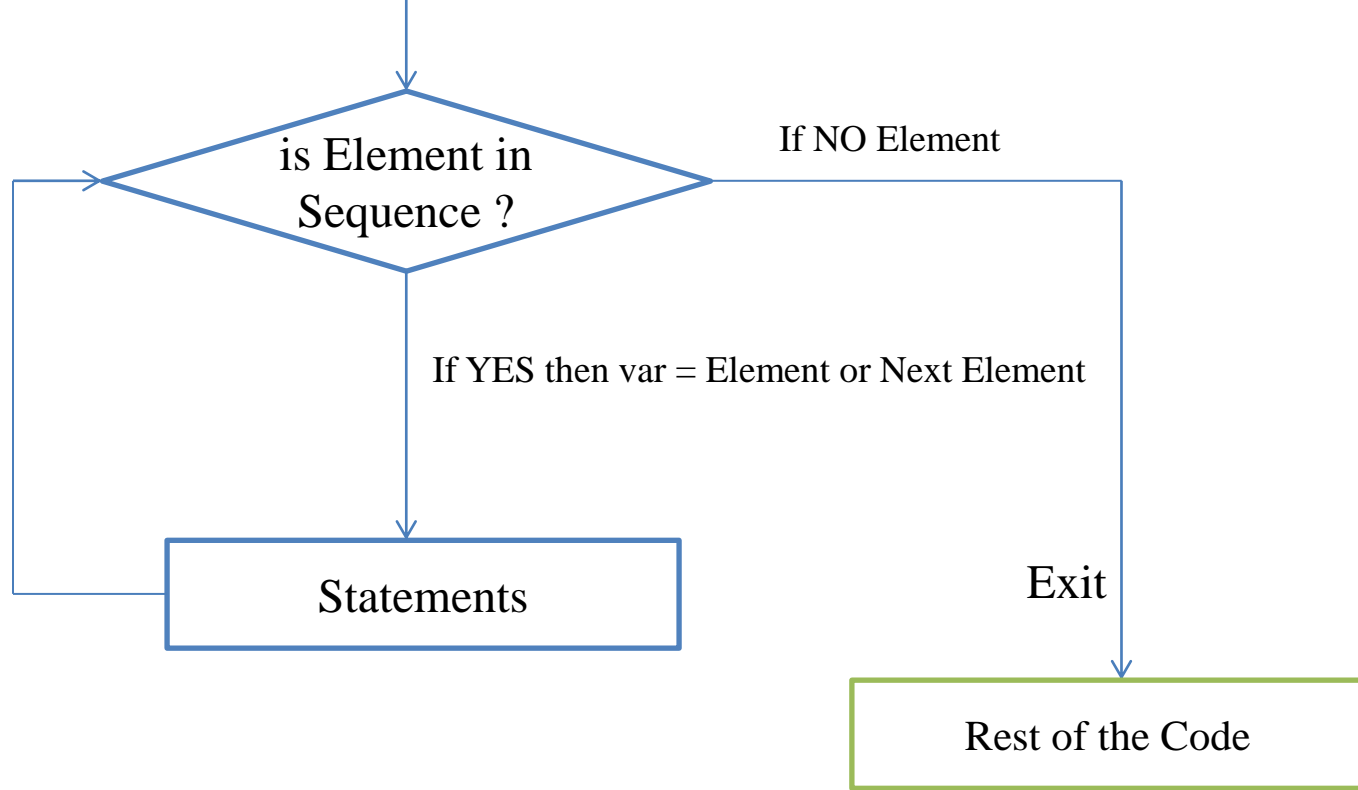
The for loop is useful to iterate over the elements of sequence such as string, list, tuple etc.

Syntax:

```
for var in sequence:
```

```
    Statements
```

```
Rest of the Code
```



# for Loop with Range

```
a = range(5)
```

```
for i in a :
```

```
    print(i)
```

```
for i in range(5) :
```

```
    print(i)
```

# for Loop with else

The for loop is useful to iterate over the elements of sequence such as string, list, tuple etc. The else suite will be always executed irrespective of the statements in the loop are executed or not.

Syntax:

for var in sequence:

Statements

else:

Statements

Rest of the Code



# Nested for loop

For loop inside another for loop is known as nested for loop.

```
for i in range(n):
```

```
    for j in range(y):
```

```
        Statements
```

```
    Statements
```

```
Rest of the Code
```