CLASS NOTES OF
"PYTHON PROGRAMMING"

TECHNICAL CLASSES™

Technical Classes

नोट :-
1. सभी ऑफलाइन कक्षाएं Technical Classes के कैंपस तथा सभी ऑनलाइन कक्षाएं Technical Classes के एप्लीकेशन पर चलेगी।
2. यह नोट्स टेक्निकल क्लासेस के स्टूडेंट्स के लिए है, तथा क्लास करने के बाद अधिक प्रभावी होगा।

By Er. Shubham Sir

Python Programming

Basics

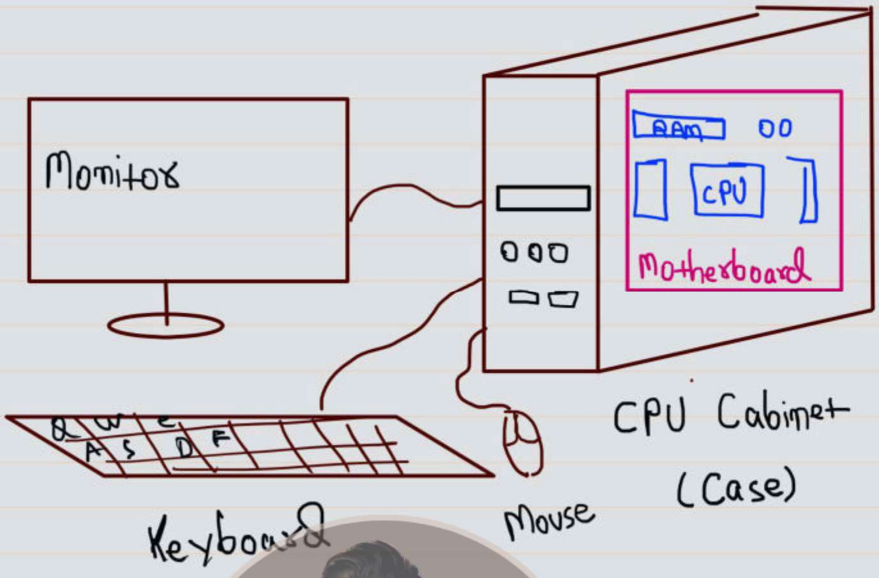


Fig → Basic Computer System

→ Computer works on binary data only.
0 1

Why Binary?

→ Computer System

↓ is made-up of

semi-conductors, transistors, wires

↳ These components have only two states → ON (1)

→ OFF (0)

Binary Language (Machine Language)

→ It has only two symbols. (0, 1)

two states → ON (1)
→ OFF (0)

Computer Instruction :-

→ An instruction is an order given to a computer processor by a computer program written in a programming language.

e.g

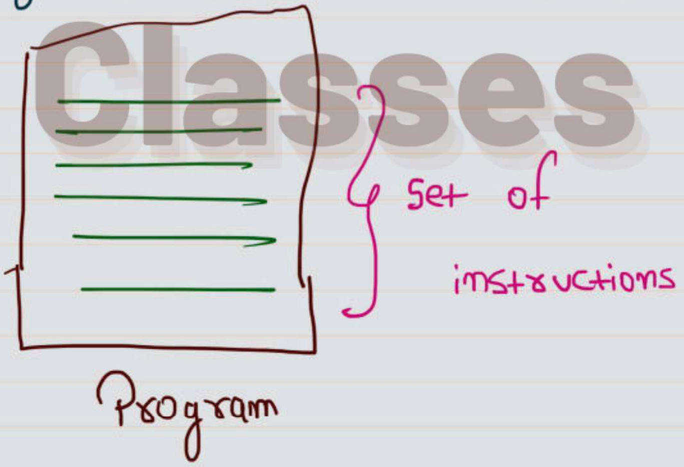
print ("Shubham") → This is an instruction written in python language.

Computer Program :-

→ A program is a set of instructions that a computer uses to perform a specific function.

→ A program is written using a programming language.

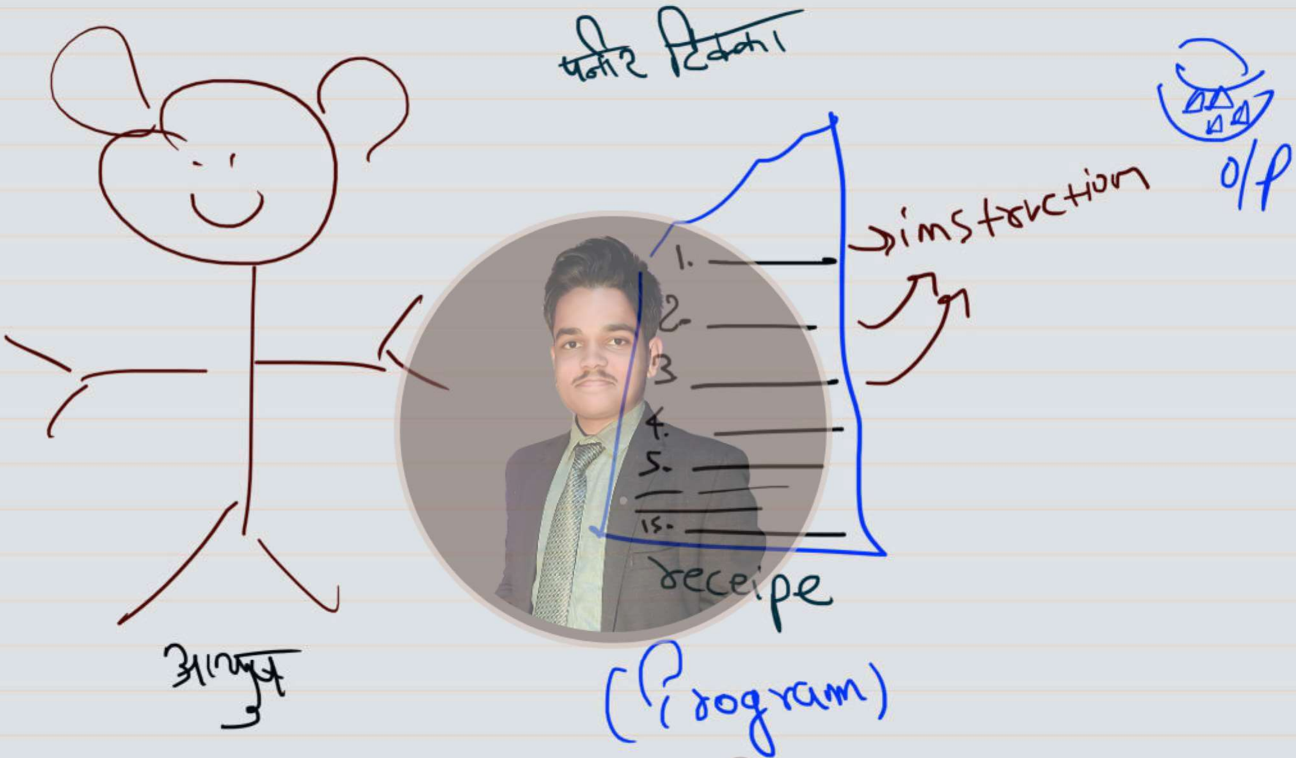
e.g a program to add two numbers



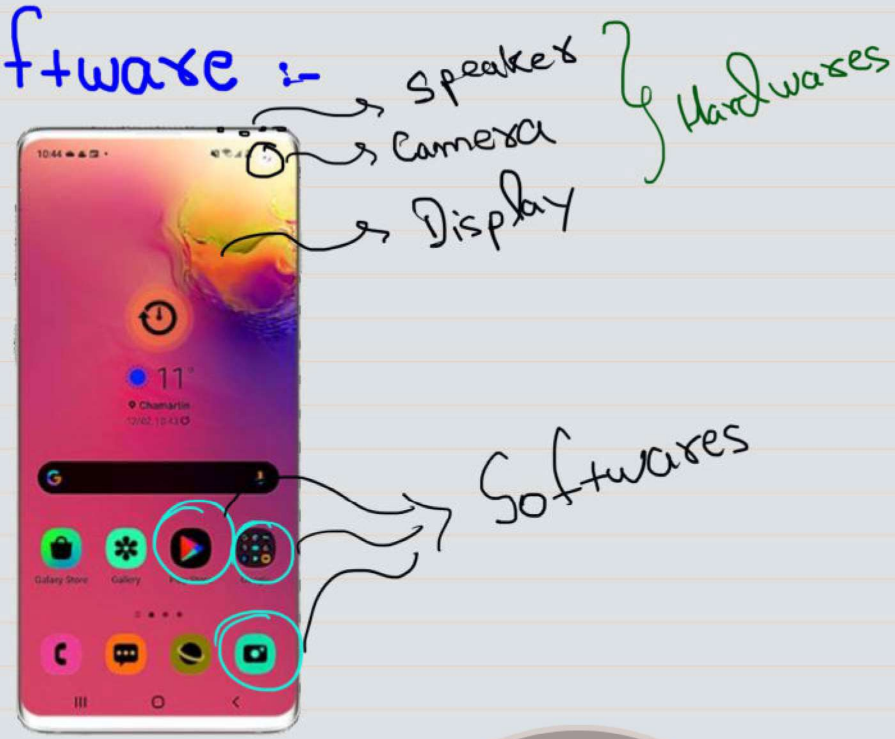
Programming Language :-

→ Computer programs are written using a particular language, known as programming language.

→ C / C++ / Java / Python etc. are the examples of programming language.



Software :-



Software :-

→ Software is a set of programs that enables the hardware to perform a specific task.

e.g.<i>

Task → Photo Capture

Camera (Hardware)

Snapchat (Software)

slide.ppt

attendance.xls

e.g.<ii>

Task → 2 + 3

Hardware

Software

(CPU) (Calculator App)

→ There are two types of software –

i) Application Software (Apps)

ii) System Software

i) Application Software :-

→ Those softwares, which are designed to perform a specific task, is known as application software.

e.g.

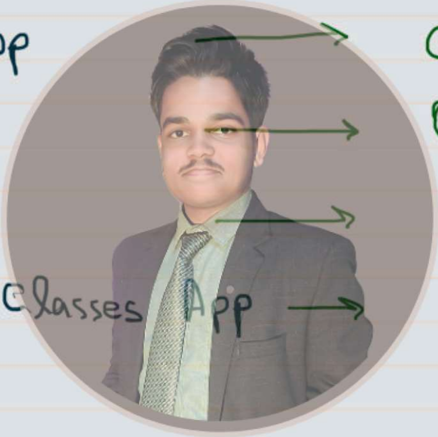
WhatsApp → Chatting

BGMI → Online Game

Spotify → Music

Technical Classes App → Online Courses

etc.



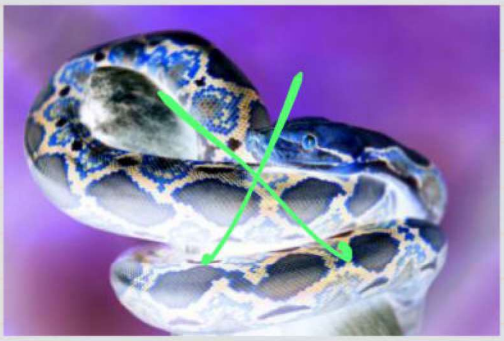
ii) System Software :-

→ A software that directly operates the computer hardware.

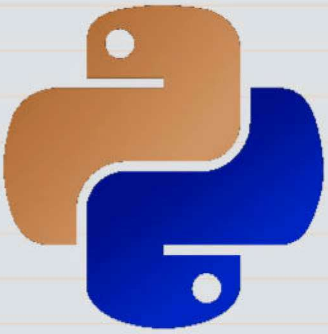
e.g. Operating System, device driver, etc.

LIVE CLASS - 4 & 5

What is Python?

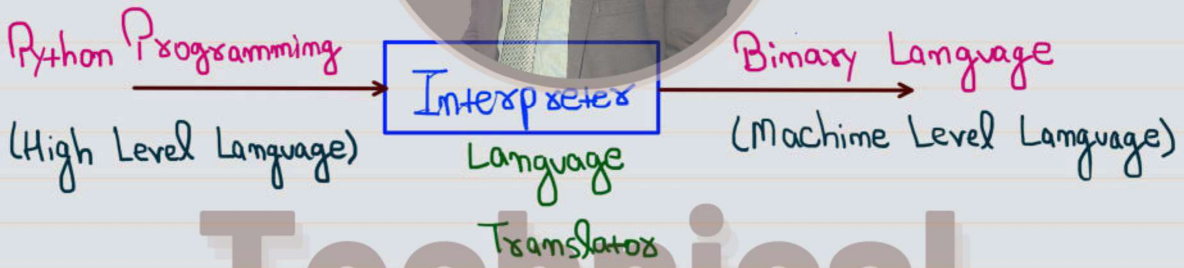


Python Snake



Python Programming

→ Python is a general purpose, dynamic, high-level and interpreted programming language.



→ Python was developed by 'Guido Van Rossum' in 1991 and further developed by the 'Python Software Foundation.'

→ It supports object oriented programming (OOP) approach to develop applications.

→ Python is simple and easy to learn. पढ़ने सीखने

→ Python is simple and easy to learn.

→ Python was designed with the aim of code readability, and its syntax allows programmers to express their concepts in fewer lines of codes.

What can Python do?

→ Python can be used on a server to create web applications.

→ Python can connect to database system. It can also read and modify files.

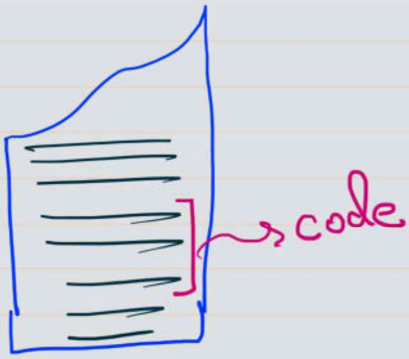
Why Python?

→ Python works on different platforms. (Windows, Linux, Mac, Rasberry Pi, etc).

→ Python has a simple syntax similar to the English Language.

→ Python runs on an interpreter system, meaning that code can be executed, as soon as it is written.

run करवाना



Python Program

अक्षर

Python Character Set :-

→ A character set in python is a collection of legal characters that is used while writing the program.

e.g.

a = 35

✓ ✓ ✓✓

→ Python is compatible with all ASCII and UNICODE characters.

Characters Encoding Techniques

Symbol	Unicode Value
B →	66
a →	97
b →	98
A →	65

→ Python character-set includes :

i) Alphabets : These includes all the lowercase (a-z) and uppercase (A-Z) alphabets.

ii) Digits : It includes all the digits (0-9)

iii) Special Symbols : It includes all types of special characters:

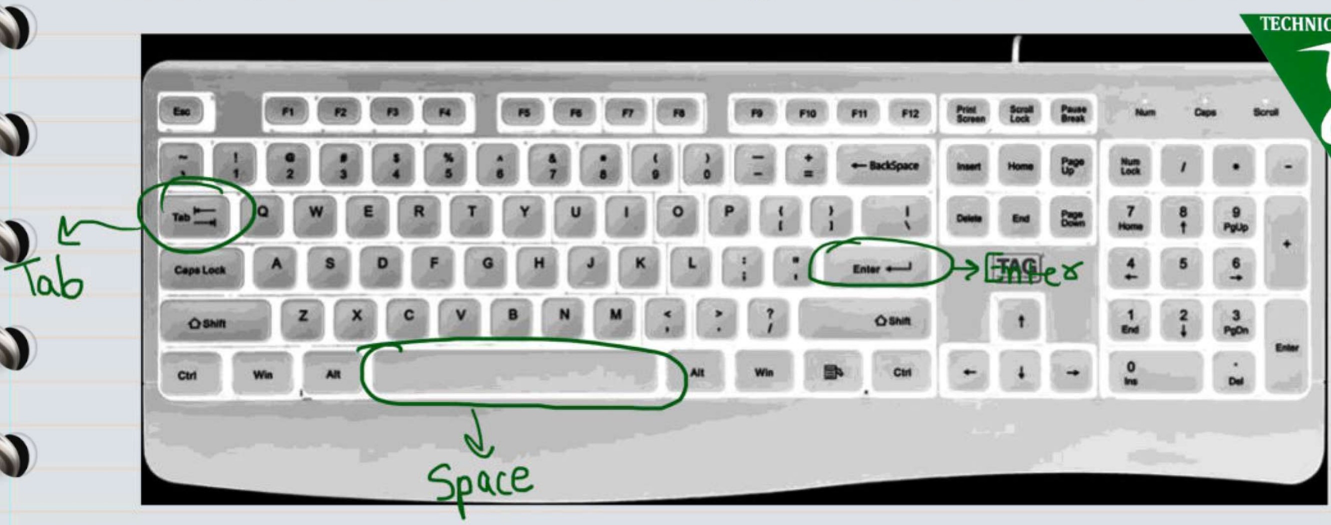
~ ! @ # $ % ^ & * ( ) _ - + = { } [ ]

< > : ; ‘ ’ “ ” ? . / \ etc.

~ → Tilde

!	→	Exclamation Mark
@	→	at the rate
#	→	Hash
$	→	Dollar
%	→	Percentage
^	→	Upper cap
&	→	Ampersand
*	→	Asterik
_	→	Underscore
:	→	Colon
;	→	Semi - colon
.	→	Dot
‘ ’	→	Single quotes
“ ”	→	Double quotes
?	→	Question Mark
/	→	forward slash
\	→	backward slash
( )	→	paranthesis
{ }	→	Curly braces
[ ]	→	Square bracket
< >	→	Angular bracket

iv) White Spaces :

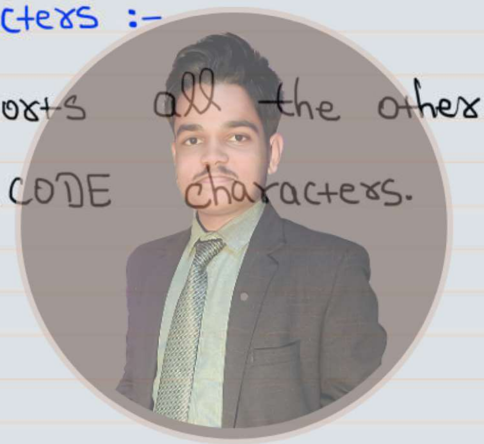


→ White spaces are also a part of the character set.

→ These are tab space, new line, blank space, etc.

v.) Other Characters :-

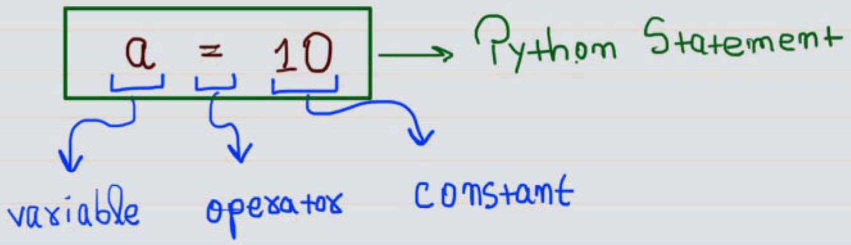
→ Python supports all the other types of ASCII and UNICODE characters.



Tokens :-

→ A token is the smallest individual unit in a python program which can't be further divided.

e.g.



→ All statements and instructions in a program are built with tokens.

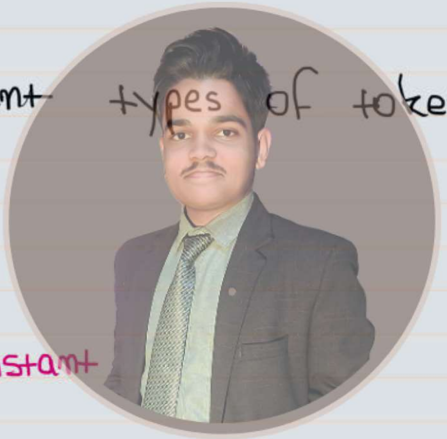
→ There are different types of tokens available in python.

i) Keywords

ii) Identifiers

iii) Literals / Constant

iv) Operators



i) Keywords :-

→ Python keywords are reserved words with predefined meaning and functions.

→ Keywords can be used for their specific function only.

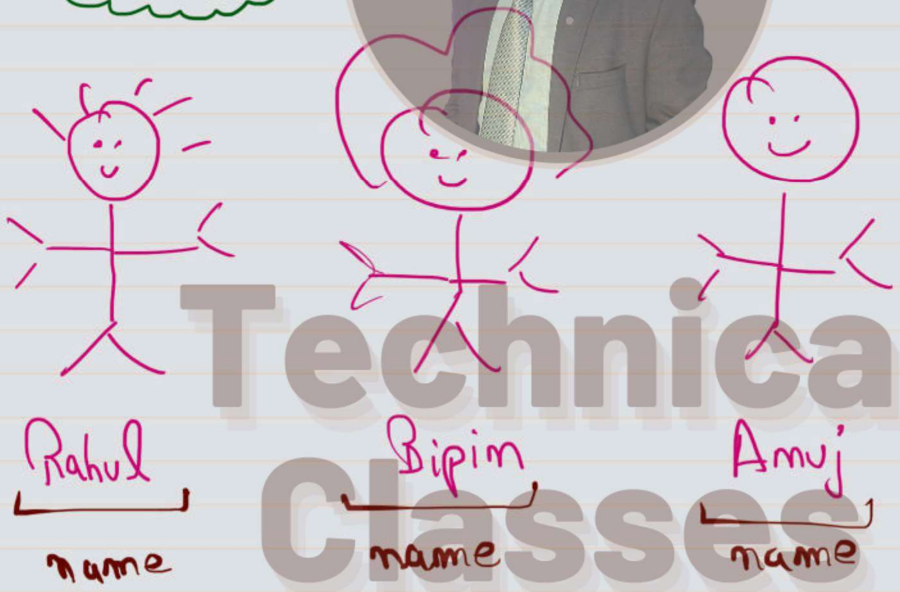
→ Python 3.11 contains total 36 keywords.

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

→ In distinct version of python, the preceding keywords might be changed.

ii) Identifiers :-

Identify → पहचानना



→ In python, an identifier is the name assigned to a variable, function, class or other object.

Rules for naming identifiers :-

i) These are the only allowed characters for a py identifier :

(a-z), (A-Z), (0-9), ___

e.g.

kajal ✓ bip012im ✓

AvinasH ✓ shu__bham ✓

r@dha ✗ sk#123 ✗

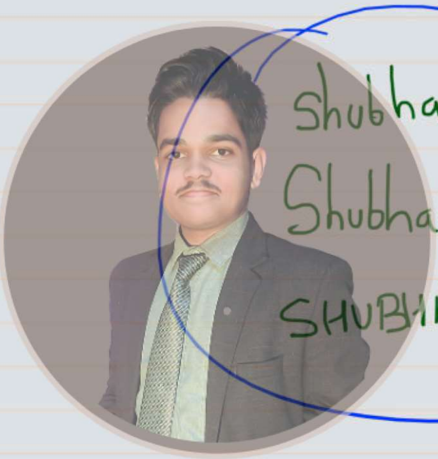
ii) An identifier name should not be a keyword.

e.g.

abc ✓

class ✗

Class ✓



shubham ✓

Shubham ✓

SHUBHAM ✓

iii) An identifier name cannot start with digit.

e.g.

shubham ✓ _ajit ✓

1shubham ✗ Kundan563 ✓

digit

iv) An identifier name in python is case-sensitive.

e.g.

Jyoti

JYOTI

& These are four identifiers.

JYOTI ←
JYOTI ←
JYOTI ←
Jyoti ←

→ These are four different identifiers.

v.) Whitespace characters are not allowed.

e.g.

omprakash ✓

om prakash ✗
   ␣ ✗

Q.) Identify the valid identifiers :-

i) jayprakash_kumar ✓

ii) @jit ✗

iii) Arunkumar ✓

iv) shubham1007 ✓

v) 5Ricky ✗

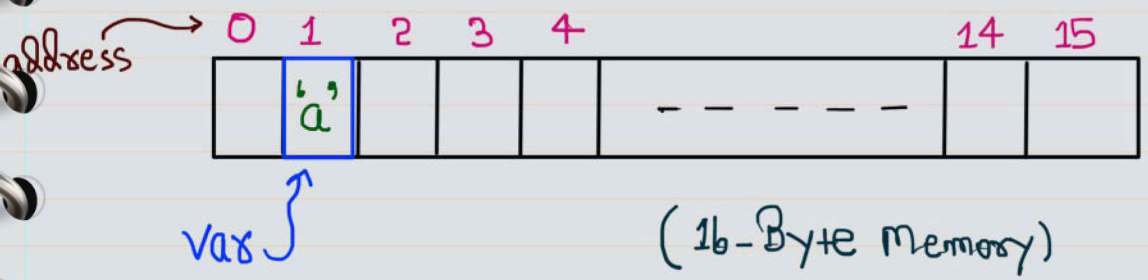
vi) technical#classes ✗

vii) _sum ✓

viii) anamika singh ✗

space not allowed

Python Variables :-



→ Variable is a name given to memory location.

→ It is used to store values.

bit → Smallest unit of memory

8 bits → 1 Byte

1024 Bytes → 1 KB (Kilo Bytes)

1024 KB → 1 MB (Mega Bytes)

1024 mB → 1 GB (Giga Bytes)

e.g.

$$2 GB = 2 \times 1024 \times 1024 \times 1024$$

→ Every value in python has a data type.

e.g.

$\underbrace{35}$ ↓ int

$\underbrace{3.14}$ ↓ float

$\underbrace{\text{"shubham"}}$ ↓ String

$\underbrace{\text{'A'}}$ ↓ character

Declaration of Variables :-

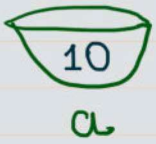
Syntax (लिखने का तरीका)

var_name = value

example

a = 10

name = "vaishnavi"



→ Data type of the variable depends upon the value which is stored in it.

var = 35

var = "Shubham"

var = 3.14

R-value and L-value :-

R → Right

L → Left

assignment operator

$\underbrace{a}_{\text{l-value}} = \underbrace{5}_{\text{r-value}}$

डब्बा = दूध

a = 4 ✓

[illegible] = [illegible]

गैलन = पानी

ड्रम = गैलन

~~पावला = कटोरा~~

~~तेल = डब्बा~~

$a = 4$ ✓

$b = a$ ✓

$~~3 = b~~$

$c = \underbrace{3+2}_{5}$ ✓

→ Tokens which are at left side of assignment operator is known as l-value.

→ Tokens which are at right side of assignment operator is known as r-value.

→ L-value should always be an assignable object.

→ ऐसा object / container जिसमें value store / रखा किया जा सके।

→ R-value can be any constant, expression or any other variable.

Types of Variables :-

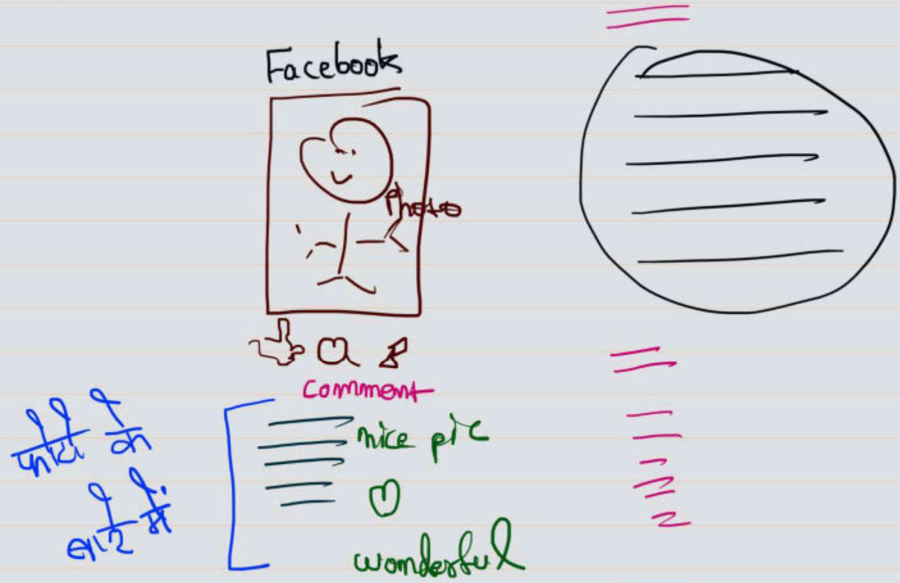
→ There are two types of variables

→ There are two types of variables

i) Global variable

ii) Local variable

Python Comments :-



→ In python, comments are used to describe our code.

→ All the comments are ignored by the python interpreter.

→ Comments increase the readability of our code i.e it can be understandable by the others.

Types of comments in python :

i) By using #

→ We can comment any line in python by adding # at the beginning of the line.

e.g.

```
# This is a comment

# a = 50
```

→ It is generally used for single-line comment.

ii) By using string literals :-
constant

→ We can write comments in python within triple quotes.

→ Triple quotes is used for string constant but it is ignored by the interpreter if not assigned to any variable.

e.g.

```
'''This is a
multi-line
comment'''
```

Example to demonstrate the use of comments in python.

```
# taking two numbers
a = 5
b = 10
# adding the values
c = a+b
# printing the result
print (c)
```

sum.py

Data Types:-

Data → raw facts & figures

e.g. name, roll no., aadhar no.

Types → प्रकार

→ In python, each variable and every literal has a data type.

literal → constant

$$\underbrace{a}_{\text{variable}} = \underbrace{20}_{\text{constant}}$$

→ Data type tells us the type of data.

→ Python is dynamically typed programming language, hence we do not need to define the type of the variable while declaring it.

dynamically → run-time

e.g.

python Language

1. x = 15
2. x = 20
3. x = "shubham"

C Language → Mother of all the languages

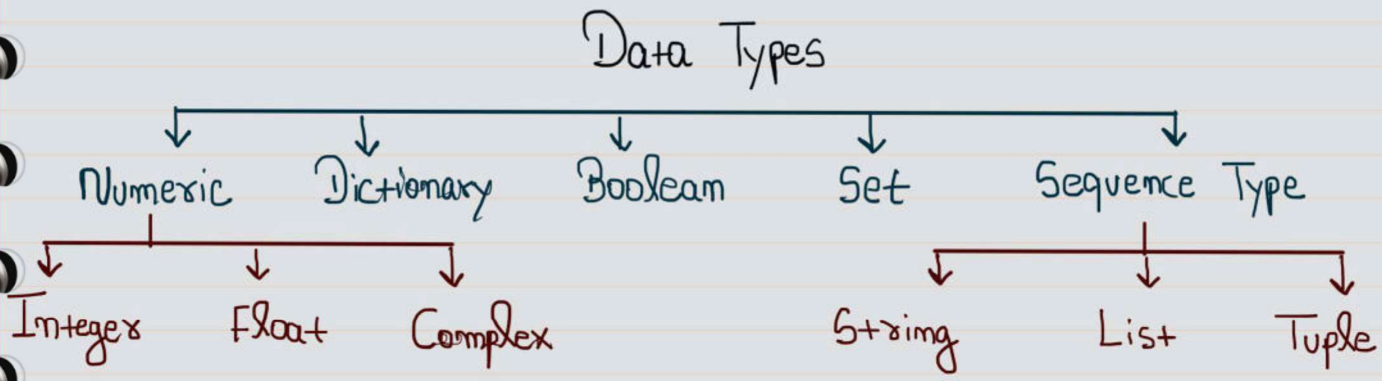
1. int x = 15;

int → data-type

2. x = 20;
3. x = "shubham"; ✗

"shubham" → compile time error

Categories of Data Types :



i) Numeric :-

→ Numeric data types store numbers.

→ The integer, float and complex number belongs to python numeric data type.

→ In numeric category there are three data types -

i) Integer (int)

→ int data type is used for integer value.

e.g.

$x = \underbrace{5}_{\text{integer number}}$

NOTE :- Python provides the type() function to know the data type of a variable.

e.g.

eg.

```
y = -5
print( type(y) )
```

o/p:- `<class 'int'>`

→ In python there is no any limit on length of the integer value.

eg.

```
value = 45              ✓
mob = 9876543210        ✓
```

ii) Float (float)

→ It is used to store floating point numbers or fractional numbers.

→ It is accurate upto 15 decimal points.

e.g.

```
z = 3.14
y = 2.12345678901234 5
x = 2.1234567890123456789
print(z)
print(y)
print(x)
print (type(z))
```

o/p:-

```
3.14
```

```
2.123456789012345
2.1234567890123457
<class 'float'>
```

iii) Complex (complex) :-

→ A complex number is written as the combination of real and imaginary number i.e $x + iy$, where the value of i is $\sqrt{-1}$.

e.g.

$[i = \sqrt{-1}]$

$$\underbrace{2}_{\text{real part}} + \underbrace{3i}_{\text{imaginary part}}$$

→ In python, complex data type is used to store complex numbers but instead of i, for representing the imaginary number $\sqrt{-1}$, j is used.

e.g

```
x = 3+2j
y = 2+7j
z = x+y
print(z)
print(type(x))
```

O/P:-

```
5+9j
```

```
<class 'complex'>
```

i.e → that is

Sequence Type Data

String :-

→ The string can be defined as the sequence (क्रम) of characters written within single, double or triple quotes.

e.g.

```
name = 'Shubham'
f_name = "Ram"
address = '''Rajeev Nagar, Road No. 0,
             Patna - 24'''

print(name)
print(type(address))
```

O/p :-

```
'Shubham'
<class 'str'>
```

→ Triple quotes is generally used for multi-line strings.

→ String belongs to class 'str' in python.

List :-

→ Lists are used to store multiple items in a single variable.

→ The list can contain data of different types.

→ The items stored in the list are separated with a comma (,)

→ The items stored in the list are separated with comma (,) and enclosed within square bracket [ ].

Syntax

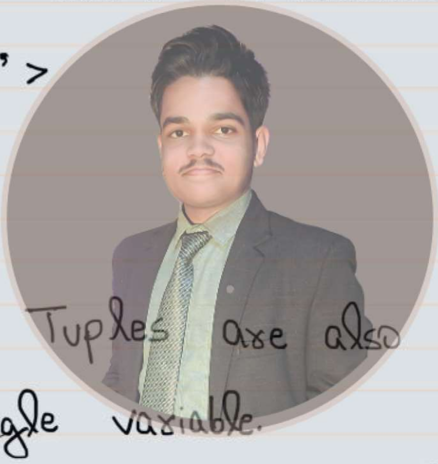
```
list_name = [ item1 , item2, item3 , - - - - ]
```

example

```
Stu1 = [ 'Shubham', 'Ram', 36, 1211818001, 96 ]
print(Stu1)
print ( type (Stu1))
```

O/p :-

```
[ 'Shubham', 'Ram', 36 , 1211818001, 96 ]
<class 'list' >
```



Tuple :-

→ Similar to List, Tuples are also used to store multiple items in a single variable.

→ But Tuple is immutable i.e read only data structure meaning that we can't modify the size and value of items in a tuple.

Syntax

```
tuple_name = (item1, item2, item3, .....)
```

or

```
tuple_name =   item1, item2, item3, .....
```

example

```
car_info = ("mercedes", 2018, '2200CC')
print (car_info)
print (type (car_info))
```

O/P:-

```
('mercedes'. 2018, '2200CC')
<class 'tuple'>
```

LIVE CLASS - 17

Boolean :-

→ Boolean type provides two built-in values – True and False.

→ These values are used to determine whether the given statement is true or false.

eg.

i) 2 > 5 ⇒ False

ii) 6 < 9 ⇒ True

→ It belongs to class 'bool'.

eg.

```
a = 5
b = 6
c = a < b
print (c)          // True
print (type (c))   // <class 'bool'>
```

Set :-

→ In python, set is the unordered collection of data type.

→ It is iterable, mutable (can be modified after creation) and has unique elements.

→ In set, the order of the elements is undefined ; it may return the changed sequence of the element.

→ The set is created using built-in function set() or

→ The set is created using built-in function or a sequence of elements is passed in the curly braces and separated by comma.

→ It can contain various types of values.

e.g.

```
a = {'Ram', 25, 5.7}
print (a)
print (type (a))
```

O/p:-

```
{25, 'Ram', 5.7}
<class 'set'>
```

Dictionary :-

→ Dictionary is an ordered collection (python 3.7 and later) of a key-value pair of items.

→ Key can hold any primitive data type, whereas value is an arbitrary python object.

→ The items in the dictionary are separated with the comma (,) and enclosed within the curly braces {}.

Syntax

```
var_name = { Key1 : value1 , Key2 : value2 , .... }
```

example

```
a = { 1 : 'Shubham' , 2 : 'Rohit' , 3 : 'Bittu' }
print (a)
print (type (a))
```

O/P:

```
{1 : 'Shubham', 2 : 'Rohit', 3 : 'Bittu'}
<class 'dict'>
```

→ Dictionary belongs to 'dict' class.

Python Literals :-

$$\underbrace{a}_{\text{variable}} = \underbrace{35}_{\text{Literal / Constant}}$$

→ Python Literals can be defined as data that is given to a variable.

→ Literals or constants are the fixed values that can't be changed.

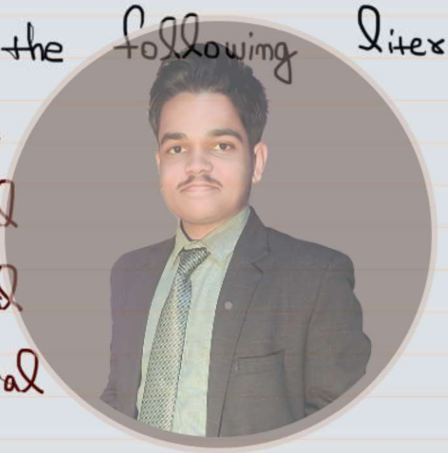
→ Python supports the following literals :

i) String Literal

ii) Numeric Literal

iii) Boolean Literal

iv) Special Literal



i) String Literal :-

→ In python, string literals can be formed by enclosing a text in the quotes.

e.g. 'shubham', 'ram', "anuj"

→ String constant can be of single-line or multi-line.

→ For single line string we use either single quotes or double quotes and for multi-line string triple quotes is used.

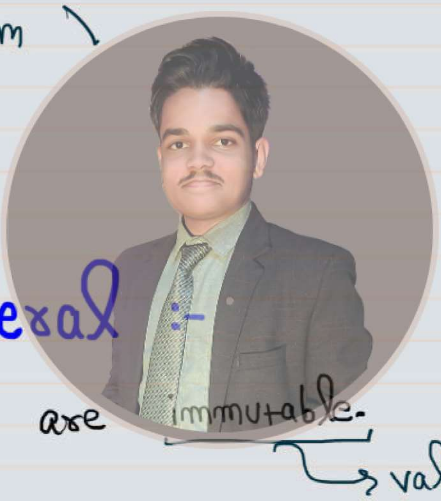
e.g. <i> 'patna'

<ii> "rajeev nagar"

<iii> """ This is a multi-line
String"""

→ We can also write multi-line string constant in single quotes. For this we have to add backslash at the end of each line.

e.g. 'My name \
is Shubham \
Kumar.'



ii) Numeric Literal :-

→ Numeric literals are immutable.
→ values can't be changed.

→ Numeric literals can be categorized in three types:

a) integer constant

→ Numbers (can be positive or negative) with no fractional part.

e.g. 100, -3, 0, 5

b) floating-point constant

→ Real numbers with both integer and fractional part.

e.g. 3.14, -5.6, 0.3

c) Complex constant

→ It is written in the form of a+bj.

e.g.

$2 + 3j$

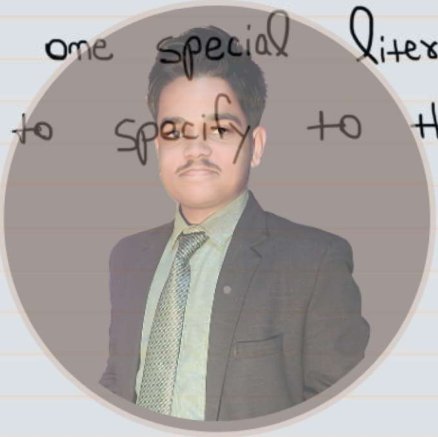
iii) Boolean Literal :-

→ 'True' and 'False' are considered as boolean literal in python.

iv) Special Literal :-

→ Python contains one special literal i.e, None.

→ 'None' is used to specify to that field that is not created.



e.g.

a = None

Typecasting :-

→ Typecasting is a process of converting one data type into another data type.

→ It is done using following constructor function:

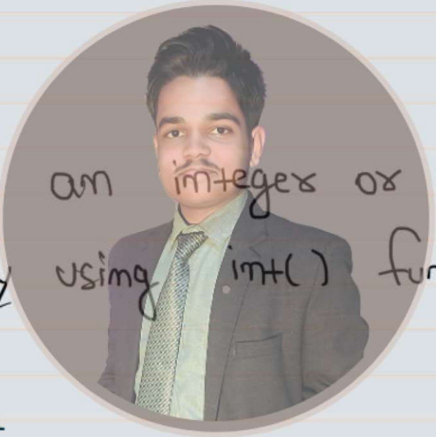
i) int ( )

ii) float( )

iii) str ( )

i) int ( ) :-

→ We can convert an integer or float or string into an integer by using int( ) function.



e.g.

```
a = 3.5
b = int(a)
print(b)
print (type(b))
```

O/p:-

```
3
<class 'int'>
```

ii) float( ) :-

→ It is used to convert an integer or real number or string into floating point value.

e.g.

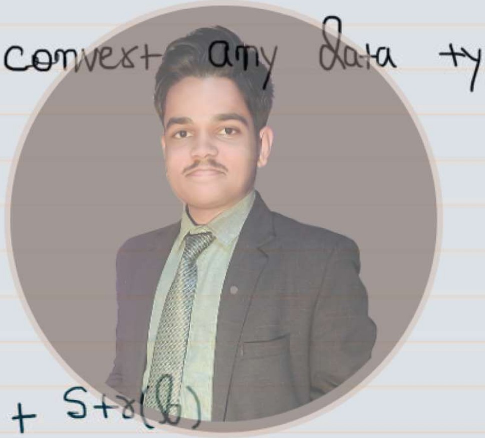
```
a = float(3)
print(a)
print(type(a))
```

O/p:-

```
3
<class 'float'>
```

iii) Str ( ) :-

→ It is used to convert any data type into string value.



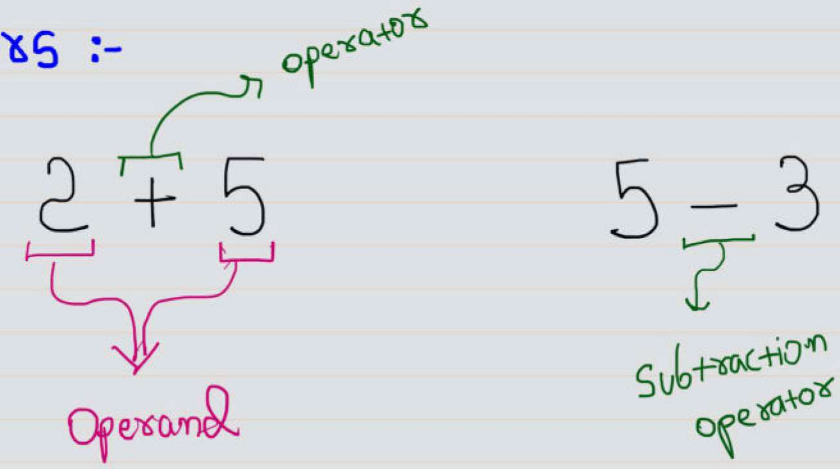
e.g.

```
a = 3
b = 5
C = Str(a) + Str(b)
print(C)
print(type(c))
```

O/p:-

```
35
<class 'Str'>
```

Operators :-



→ The operator is a symbol that performs a certain operation on operands.

→ There are many types of Operators in python :

i) Arithmetic Operators

ii) Relational Operators

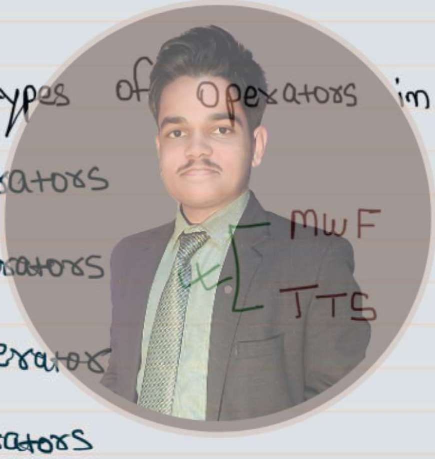
iii) Assignment Operator

iv) Logical Operators

v) Bitwise Operators

vi) Membership Operators

vii) Identity Operators



MWF 06:30 Pm — 07:30 Pm

TTS 08:30 Pm — 09:30 Pm

Arithmetic Operators :-

→ Arithmetic operators performs arithmetic operation between two operands.

i) Addition operator (+)

→ It is used to add two numbers.

e.g.

```
a = 2+3
print(a)        o/p : 5
```

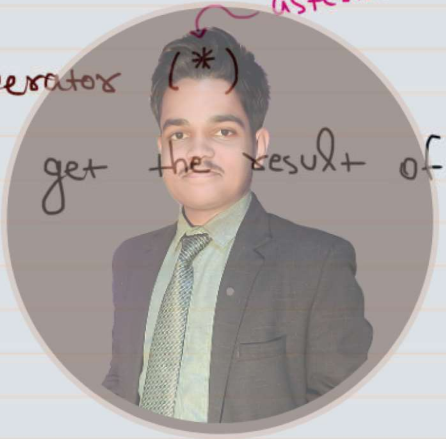
ii) Subtraction Operator (−)

→ It is used to subtract 2nd value from the first one.

e.g.

```
a = 8
b = 3
c = a-b
print(c)        o/p : 5
```

iii) Multiplication Operator (*) asterick



→ It is used to get the result of multiplication of two values.

e.g.

```
a = 3
b = 2
c = a*b
print(c)        o/p : 6
```

iv) Division Operator (/)

→ It returns the value after dividing the first operand by the second operand.

e.g.

```
a = 5
b = 2
c = a / b
```

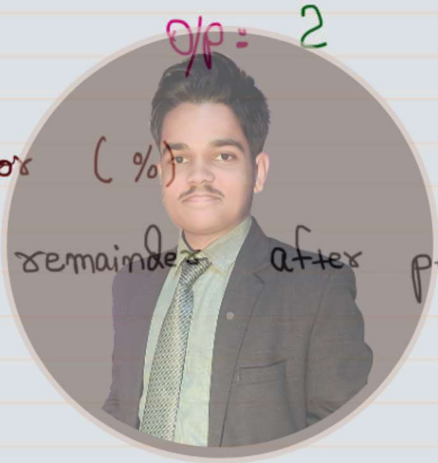
```
print(c)        O/p: 2.5
```

v) Floor Division (//)

→ It provides the quotient value after performing integral division.

e.g.

```
a = 5
b = 2
c = a//b
print(c)        O/p: 2
```



vi) Modulus Operator (%)

→ It gives the remainder after performing integral division.

e.g.

```
a = 5
b = 2
c = a % b
print(c)        O/p: 1
```

vii) Exponent Operator (**)

→ It calculates the first operand's power to the second operand.

e.g.

$a ** b \cong a^b$

```
a = 2
b = 3
```

```
c = a ** b
print(c)        O/p : 8
```

Relational Operator :-

→ Relational operators compare the value of two operands and return a True or False value according to the given relation.

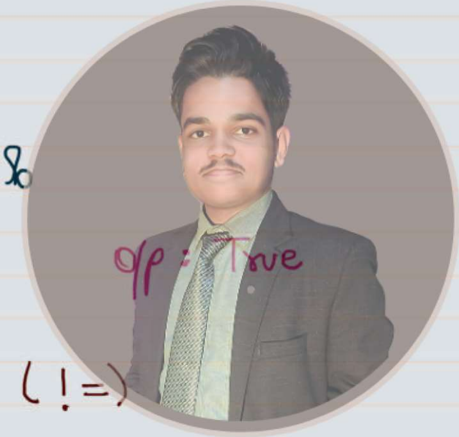
→ There are many relational operator in python:

i) Comparision operator (==)

→ It gives the output true if the value of both the operands is same otherwise it gives the output false.

e.g.

```
a = 5
b = 5
c = a == b
print (c)        o/p : True
```



ii) Not Equal to (!=)

→ It gives the output true if both the operands have different values otherwise false.

e.g.

```
print ( 5 != 6 )        o/p : True
print ( 4 != 4 )              False
```

iii) Less than (<)

→ If the first operand is less than the second operand then the output will be true otherwise false.

e.g.

```
a = 5
```

```
b = 6
print( a < b)          O/P:-   True
print (b < a)                  False
print ( a < 5)                 False
```

iv) Less than or Equal to (<=)

→ The condition becomes true if the first operand is less than or equal to the second operand.

e.g.

```
a = 5
b = 7
print ( a <= b)        True
print ( a <= 5)        True
print ( b <= a)        False
```

v) Greater than (>)

→ If the first operand is greater than the second operand then the output will be true Otherwise false.

eg

```
a = 5
b = 7
print ( a > b)       False
print ( b > a)       True
print ( b > 7)       False
```

vi) Greater than or Equal to (>=)

→ The output will be true if the first operand is greater than or equal to the second operand.

eg

```
a = 5
b = 7
print ( b >= a )        True
print( b >= 7 )         True
```

Assignment Operator :-

→ The right expression's value is assigned to the left operand using the assignment operator.

e.g

$a = 5$

assignment operator

$b = \underbrace{2+3}_{5}$

Compound Assignment Operator :-

Operator	Example	Explanation
+=	a += 5	a = a+5
-=	a -= 5	a = a-5
*=	a *= 5	a = a*5
/=	a /= 5	a = a/5
**=	a **= 5	a = a**5
%=	a %= 5	a = a%5
//=	a //= 5	a = a//5

Logical Operator :-

→ Logical operators are used to combine conditional statements.

→ Logical operators are used to combine conditional statements.

→ There are three logical operators in Python :

i> and ii> or iii> not

i> and

→ It returns true if both the statements are true.

e.g.

<i> $\underbrace{2 < 5}_{\text{True}}$ and $\underbrace{7 > 3}_{\text{True}}$ ⇒ True

<ii> $\underbrace{5 != 6}_{\text{True}}$ and $\underbrace{6 == 5}_{\text{False}}$ ⇒ False

Truth Table :-

Condition 1	Condition 2	Result
False	False	False
False	True	False
True	False	False
True	True	True

ii> or

→ It returns true if atleast one input is true.

Truth Table :-

Condition 1	Condition 2	Result
False	False	False

False	True	True
True	False	True
True	True	True

e.g.

<i> $\underbrace{2 > 1}_{\text{True}}$ or $\underbrace{5 < 3}_{\text{False}}$ ⇒ True

<ii> $\underbrace{5 == 6}_{\text{False}}$ or $\underbrace{3 != 3}_{\text{False}}$ ⇒ False

iii) not :-

→ It simply inverts i.e reverse the result.

Truth Table

Statement	Result
not True	False
not False	True

e.g.

<i> not $(\underbrace{2 > 1}_{\text{True}})$ ⇒ False

<ii> not $(\underbrace{5 != 5}_{\text{False}})$ ⇒ True

Number System :-

→ method to represent any value using digits

i) Decimal Number System

Base / radix = 10

Digits = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Place Value

_	_	_	$Base^4$	$Base^3$	$Base^2$	$Base^1$	$Base^0$
_	_	_	10^4	10^3	10^2	10^1	10^0
_	_	_	10000	1000	100	10	1

e.g.

$$\underbrace{2}\underbrace{5}\underbrace{6}$$

$$2 \times 100 + 5 \times 10 + 6 \times 1$$

ii) Binary Number System :-

Base / Radix = 2

Digits = 0, 1

Place Value

- - -	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	256	128	64	32	16	8	4	2	1

e.g

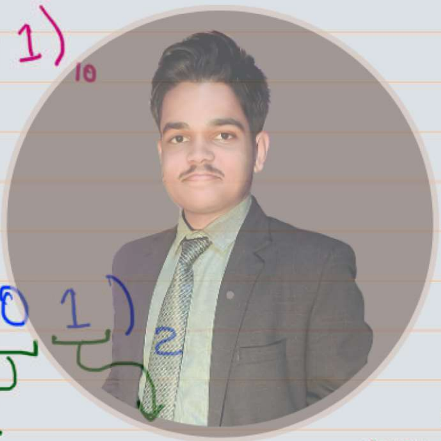
i) $(101)_2$

$1\times4 + 0\times2 + 1\times1$

$= (4+0+1)_{10}$

$= (5)_{10}$

$(\cancel{504})_2$



ii) $(1101)_2$

$1\times8 + 1\times4 + 0\times2 + 1\times1$

$= (8+4+0+1)_{10}$

$= (13)_{10}$

iii) $(\overset{32}{1}0\overset{8}{1}0\overset{2}{1}\overset{1}{1})_2$

$1\times32 + 0\times16 + 1\times8 + 0\times4 + 1\times2 + 1\times1$

$= (32+0+8+0+2+1)_{10}$

$= (43)_{10}$

Decimal to Binary

i) $(13)_{10} = (\_\_)_2$

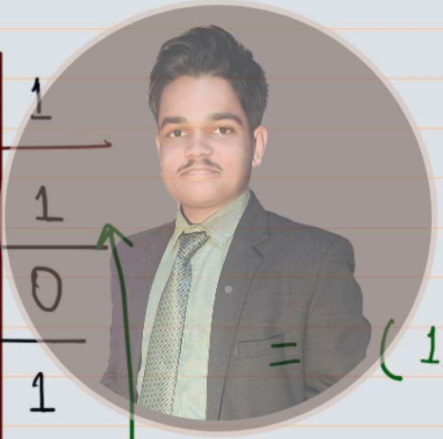
2	13	1
2	6	0
2	3	1
	1	

take remainders in reverse order

$= (1101)_2$

ii) $(43)_{10} = (\_\_\_\_)_2$

2	43	1
2	21	1
2	10	0
2	5	1
2	2	0
	1	



$= (101011)_2$

iii) $(63)_{10} = (\_\_\_\_)_2$

2	63	1
2	31	1
2	15	1
2	7	1
2	3	1
	1	

$= (111111)_2$

Bitwise Operator :-

→ Bitwise operators are used to perform bit-level manipulation.

binary-digit

→ There are six bitwise operators in python :

i) Bitwise AND (&)

ii) Bitwise OR (|)

iii) Bitwise NOT (~) / Complement

iv) Bitwise XOR (^)

v) Bitwise right shift (>>)

vi) Bitwise left shift (<<)

i) Bitwise AND (&) :-

→ Bitwise AND performs AND operation b/w bits of both the inputs.

→ It is represented by ampersand (&) symbol in python.

→ It gives the output 1 if both the operands are 1 otherwise it gives the output 0.

Truth Table :

a	b	a&b
0	0	0
0	1	0
1	0	0
1	1	1

eg (i)

```
a = 15
b = 9
c = a & b
print(c)
```

$(15)_{10} \Rightarrow 1111$

$(9)_{10} \Rightarrow 1001$

o/p :- 9

$$\begin{array}{r} 1\ 1\ 1\ 1 \\ \&\quad 1\ 0\ 0\ 1 \\ \hline (1\ 0\ 0\ 1)_2 \\ \hline \Downarrow \\ (9)_{10} \end{array}$$

eg (ii)

15 & 17

$$\begin{array}{rl} (15)_{10} = & \underline{0}\ 1\ 1\ 1\ 1 \\ (17)_{10} = & 1\ 0\ 0\ 0\ 1 \\ \hline & 0\ 0\ 0\ 0\ 1 \\ \hline & \Downarrow \\ & (1)_{10} \end{array}$$

ii) Bitwise OR (|)

→ It performs bitwise OR operation b/w two bits.

→ It gives the output 1 if any one input is 1.

eg (i)

a = 13 | 18

$(13)_{10} \Rightarrow 01101$

```
print(a)
```

o/p: 31

$(18)_{10} \Rightarrow 1\,0010$

$(1\,1111)_2$

$\Downarrow$

$(31)_{10}$

Truth Table :

a	b	a|b
0	0	0
0	1	1
1	0	1
1	1	1

e.g.(ii)

```
b = 27|33
print(b)
```

$(27)_{10} \Rightarrow 11011$

$(33)_{10} \Rightarrow 100001$

$(111011)_2$

$\Downarrow$

$(59)_{10}$

iii) Bitwise NOT (~)

→ Bitwise NOT is a unary operator because it takes only one operand as input.

→ It simply inverts all the bits i.e 1-bit to 0-bit and 0-bit to 1-bit.

iv) Bitwise XOR (^) :-

→ It gives the output 0 if the inputs are same otherwise it gives the output 1.

e.g.<i>

```
c = 20 ^ 17
print(c)
```

O/p :- 5

$(20)_{10} \Rightarrow 10100$

$(17)_{10} \Rightarrow 10001$

$(00101)_2$

⇓

$(5)_{10}$

e.g.<ii>

```
d = 7 ^ 16
print (d)
```

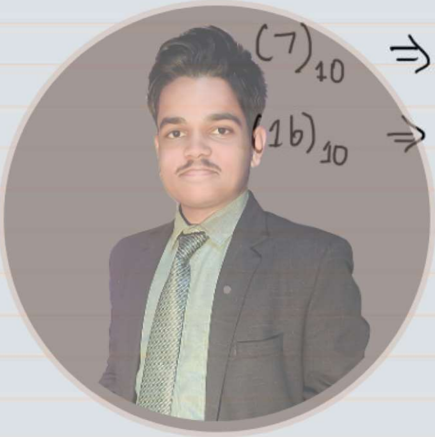
$(7)_{10} \Rightarrow 00111$

$(16)_{10} \Rightarrow 10000$

$(10111)_2$

⇓

$(23)_{10}$



v) Bitwise Right Shift Operator (>>) :-

→ It is used to shift the bits of binary value to the right.

e.g. <i>

```
a = 20 >> 1
print (a)
```

O/p :- 10

10100

$(01010)_2$ × (discard)

vacant position

⇓

$(10)_{10}$

<ii>

```
b = 20 >> 2
print (b)
```

10100

<ii> b = 20 >> 2

print(b)

O/p :- 5

$$\begin{array}{l} 1\,0\,1\,0\,0 \\ \underline{0}\,1\,0\,1\,0 \;\times \\ (\underline{0}\,0\,1\,0\,1)_2 \\ \Downarrow \\ (5)_{10} \end{array}$$

NOTE :- When we perform right shift operation, the operand gets divided by the power of 2.

$x >> n \cong x / 2^n$

$50 >> 3 \cong 50/2^3$

$\boxed{6}$ $\qquad 6.\cancel{25}$

vi) Bitwise Left Shift Operator (<<)

→ It is also a shift operator like bitwise right shift operator.

→ It shifts all the bits of a value to left.

eg <i> b = 20

c = b << 1

print (c)

O/p : 40

$$(20)_{10} \Rightarrow (1\,0\,1\,0\,0)_2$$
$$(1\,0\,1\,0\,0\,\underline{0})_2$$
$$\Downarrow$$
$$(40)_{10}$$

<ii> Z = 11 << 2 $\qquad (11)_{10} \Rightarrow (1\,0\,1\,1)_2$

(ii) Z = 11 << 2

print (z)

O/p:- 44

$(11)_{10} \Rightarrow (1011)_2$

10110

$(101100)_2$

32+ 8+4

$(44)_{10}$

NOTE :- When we perform left shift operation, the operand gets multiplied by the power of 2.

$x << n \cong x * 2^n$

$50 << 3 \cong 50 * 2^3$

$\boxed{400}$

vi) Membership Operator :-

→ Membership operators are used to test if a sequence is presented in an object.

→ There are two membership operator in python :

i) in ii) not in

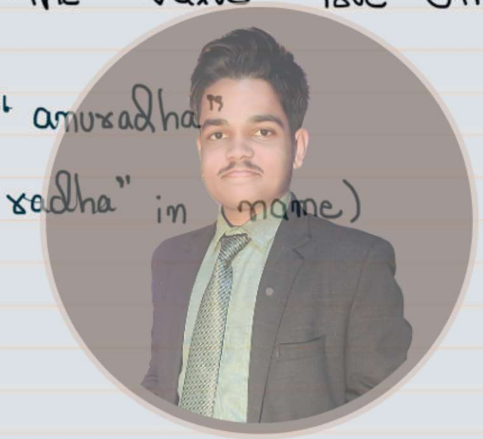
i) in :-

→ If the first operand is present in the second operand then it returns the value True otherwise False.

e.g.(i)

```
name = "anuradha"
print ("radha" in name)
```

O/p:- True



e.g.(ii)

```
num = [2, 5, 7]
print (2.0 in num)   ⇒ True
print ( 8 in num)    ⇒ False
```

ii) not in :-

→ If the first operand is not present in the second operand then it gives the output True otherwise False.

e.g.

```
name = "Anuradha"
```

```
print ('Annu' not in name)
print ('radha' not in name)
```

O/p :-
```
True
False
```

vii) Identity Operator :-

→ Identity operators are used to compare the objets, not if they are equal , but if they are actually the same object, with the same memory location.

→ There are two identity operators in python :

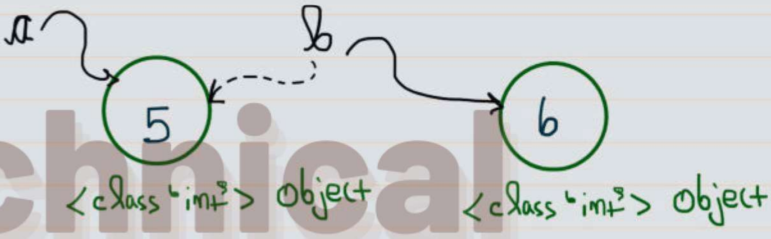
i) is   ii) is not

i) is :-

→ It returns True if both the variables are the same object.

e.g.

```
a = 5
b = 5
print(a is b)
b = 6
print(a is b)
```



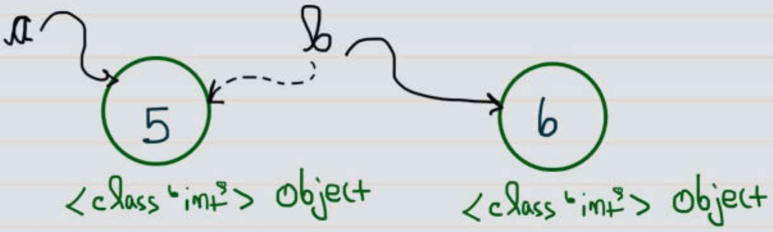
O/p :-
```
True
False
```

ii) is not :-

→ It returns True if both the variables are not same

e.g.

```
a = 5
b = 5
print(a is not b)
b = 6
print(a is not b)
```



O/P :-

```
False
True
```

Precedence and Associativity of Operators :-

e.g.(i)

2+2/2

Case-I

⇒ 2+2/2

⇒ 4/2

⇒ 2

Case-II

⇒ 2+2/2

⇒ 2+1

⇒ 3

e.g.(ii)

2 + 40 >> 2

Case-I

$\underbrace{2+40}>>2$

$\underbrace{42>>2}$

10

Case-II

$2+\underbrace{40>>2}$

$\underbrace{2+10}$

12

e.g.(iii)

40 >> 4 >> 1

Case-I

$\underbrace{40>>4}>>1$

$\underbrace{2>>1}$

1

Case-II

$40>>\underbrace{4>>1}$

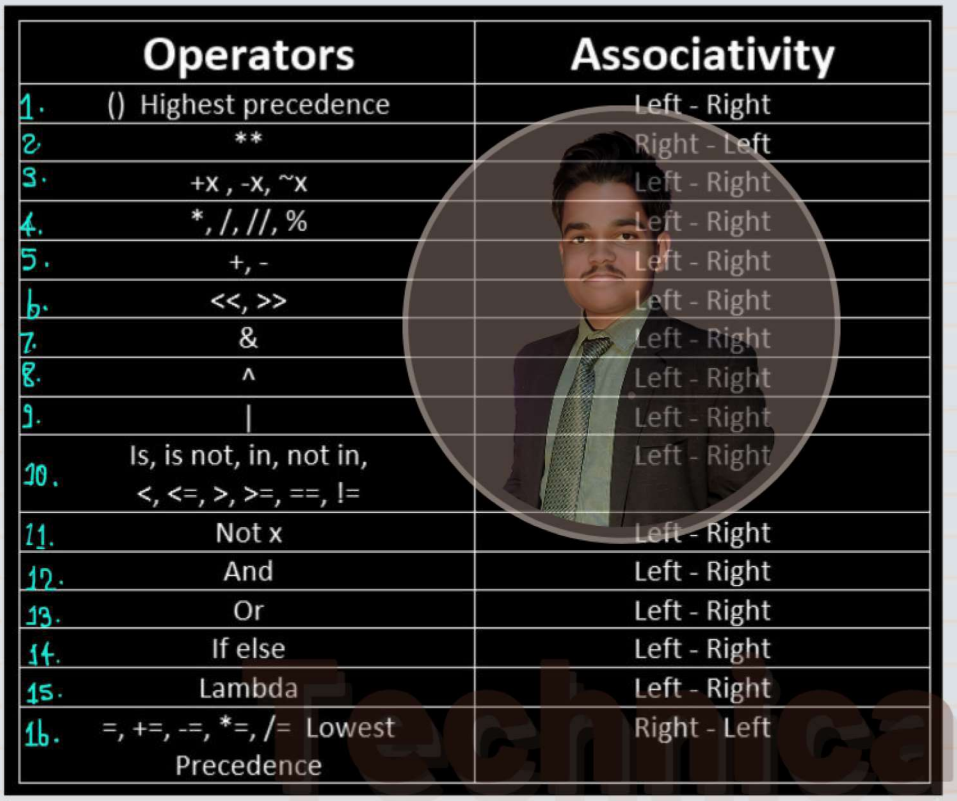
$\underbrace{40>>2}$

10

→ In python, operators have different level of precedence, which determines the order in which they are evaluated.

→ When multiple operators are present in an expression, the ones with higher precedence are evaluated first.

→ In the case of operators with same precedence, there associativity comes into play, determining the order of evaluation.



Operators	Associativity
1. () Highest precedence	Left - Right
2. **	Right - Left
3. +x , -x, ~x	Left - Right
4. *, /, //, %	Left - Right
5. +, -	Left - Right
6. <<, >>	Left - Right
7. &	Left - Right
8. ^	Left - Right
9. |	Left - Right
10. Is, is not, in, not in, <, <=, >, >=, ==, !=	Left - Right
11. Not x	Left - Right
12. And	Left - Right
13. Or	Left - Right
14. If else	Left - Right
15. Lambda	Left - Right
16. =, +=, -=, *=, /= Lowest Precedence	Right - Left

Highest Precedence

↓

Lowest Precedence

e.g. (i)

Left → Right

2 * 5 % 3 / 1 // 2

⇒ 10 % 3 / 1 // 2

⇒ 1 / 1 // 2

⇒ 1 // 2

⇒ 0

e.g.(ii)

$$2 \overset{\textcircled{2}}{+} \underbrace{5 \overset{\textcircled{1}}{**} 2} \overset{\textcircled{3}}{<<} 2$$

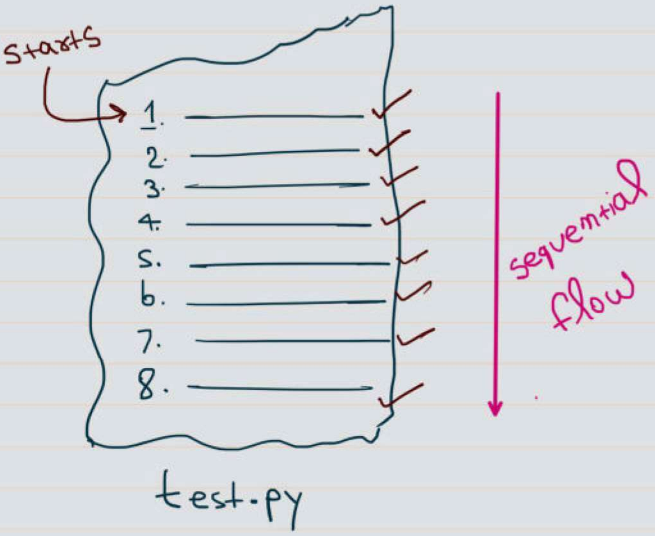
$$\Rightarrow \underbrace{2 + 25} << 2$$

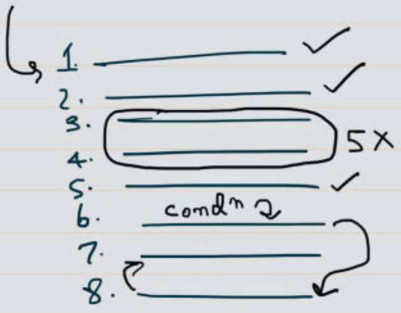
$$\Rightarrow 27 << 2$$

$$\Rightarrow 108$$

Unit - 1 : The End ✓

Unit → 2 (Control Flow Statements)





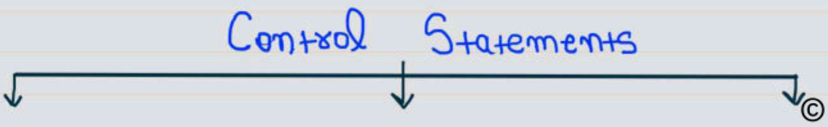
Control Flow Statement :-

→ A program's Control flow is the order in which the program's code executes.

→ By default, the flow of execution is sequential in python.

→ We can control or regulate the flow of execution of python programs by using control structures or control statements.

→ There are many control statements in python and we categorize them in three category :



Conditional	Iterative (loop)	Transfer
i) if	i) while	i) break
ii) if-else	ii) for	ii) continue
iii) elif		iii) pass
iv) Nested if-else		

Conditional Control Statements :-

→ It is used for decision-making.
↳ निर्णय लेना

→ There are four types of conditional statements in python :

i) if

ii) if-else

iii) elif

iv) Nested if-else

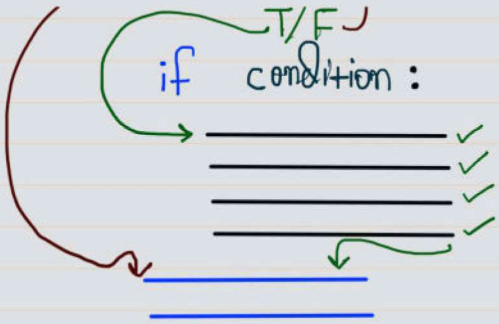
i) Simple if :-

→ The simple if statement is used to test a particular condition and if the condition is true, it executes a block of code, known as if-block.

Syntax :

T/F

if condition :



Example

<i>

```
T
if 2>1:
    print("Hii") ✓
print("Bye")
```

<ii>

```
F
if 2< 1:
    print("Hii")
print("Bye")
```

O/p:- Hii
Bye

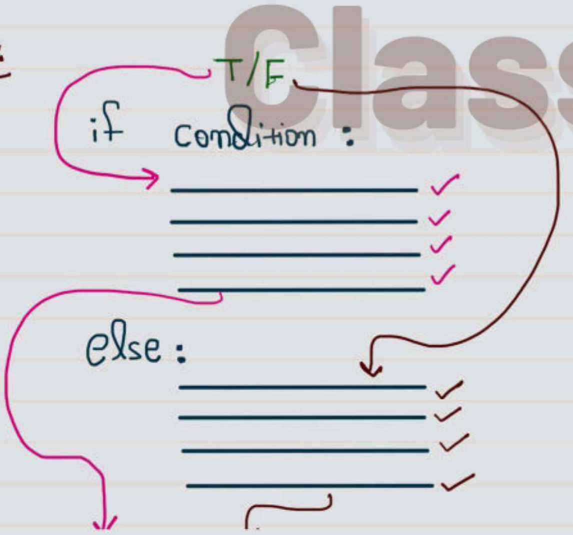
O/p:- Bye

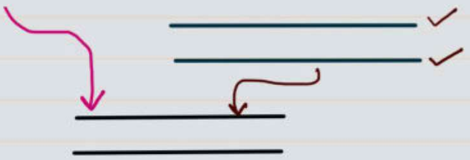
ii) if-else :-

→ The if-else statement provides an else block combined with the if statement which is executed when the condition is false.

→ If the condition is true the if block will be executed otherwise the else block will be executed.

Syntax:





Example

i)
```
a = 5
   F
if (a > 5):
   print ("Hii")
else:
   print ("Hello") ✓

print ("Byee") ✓
```

O/p:- Hello
Byee

ii)
```
a = 15
   T
if (a > 5):
   print ("Hii") ✓
else:
   print ("Hello")

print ("Byee") ✓
```

O/p:- Hii
Byee

I/O in Python:-

→ We can perform input/output operation in python by using predefined functions.

→ At runtime, to take input from user, input() function is used.

→ ** input() function reads the input given by user as string format.

→ So, to convert the input in desired format, typecasting is used.

eg.
```
a = int(input("Enter a number: "))
print(a)
```

O/p:- Enter a number : 5 ↵

5

→ To give output on monitor screen, print() function is used.

e.g.

```
a = 5
print (a)
print ("Shubham")
```

O/p:- 5

Shubham

#P1) WAP in python to check whether a given number is even or odd.

```
num = int(input('Enter a number: '))
if num%2 == 0:
    print (num, " is an even number.")
else:
    print (num, " is an odd number")
```

Dry Run :1

num

$\boxed{5}$

Enter a number : 5 ↵

5 is an odd number

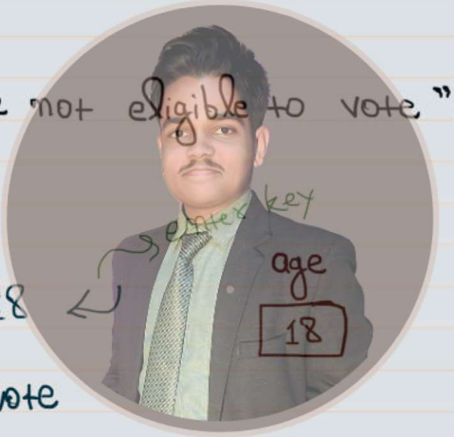
Dry Run: 2

```
Enter a number : 8 ↲
8 is an even number.
```

num $\boxed{8}$

P2) WAP in python to check whether a person is eligible to vote or not.

```
age = int(input(" Enter your age : "))
if age >= 18 :
    print ("You are eligible to vote ")
else:
    print ("You are not eligible to vote ")
```



Dry Run: 1

```
Enter your age : 18 ↲
You are eligible to vote
```

enter key

age $\boxed{18}$

Dry Run: 2

```
Enter your age : 12 ↲
You are not eligible to vote.
```

age $\boxed{12}$

iii) elif statement :-

↳ else + if

→ The elif statement enables us to check multiple condition and execute the specific block of statements depending upon the true condition among them.

Syntax :-

Syntax :-

```
if condition1 :
    ____________
    ____________

elif condition2:
    ____________
    ____________

elif condition3:
    ____________
    ____________

else:
    ____________
    ____________
```

# P3> WAP in python to check whether a given number is positive or negative.

```
num = int (input ('Enter a number: '))
if (num > 0):
    print (num, ' is a positive number.')
elif (num < 0):
    print (num, 'is a negative number.')
else:
    print (num, 'is zero.')
```

Dry Run 1:

```
Enter a number : 5
5 is a positive number
```

Dry Run 2:

```
Enter a number : -6
-6 is a negative number
```

Dry Run 3:

```
Enter a number : 0
```

0 is zero.

iv) Nested if-else :-

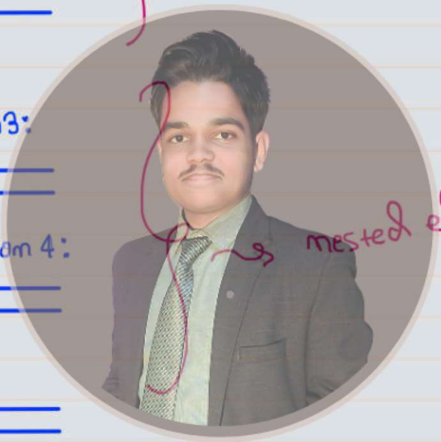
→ We can have if-else statements inside another if block or else block, and this is called nested if-else statement.

Syntax:

```
if condition1:
    if condition2:
        ____________
        ____________
    else:
        ____________
        ____________
else:
    if condition3:
        ____________
        ____________
    elif condition 4:
        ____________
        ____________
    else:
        ____________
        ____________
```

nested if

nested else



#P4 > WAP in python to check whether a year is leap or not.

```
year = int (input('Enter year :'))
if ( year % 100 == 0):
    if (year % 400 == 0):
        print (year, 'is a leap year')
    else:
        print (year, 'is not a leap year')
else:
    if (year % 4 == 0):
        print (year, 'is a leap year')
```

Dry Run 1:

Enter year : 2000 ↵

2000 is a leap year.

Dry Run 2:

Enter year : 1800 ↵

1800 is not a leap year.

Dry Run 3:

```
if (year % 4 == 0):
    print (year, 'is a leap year')
else:
    print (year, 'is not a leap year')
```

```
1800 is not a
```

Dry Run 3:

```
Enter year: 2024 ↵
2024 is a leap year.
```

Dry Run4:

```
Enter year: 2025 ↵
2025 is not a leap year.
```

#P5> WAP in python to check whether a year is leap or not Using elif construct.

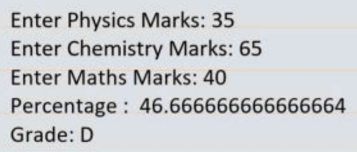
```
year = int(input('Enter year :'))
if year % 100 == 0 and year % 400 == 0:
    print(year, 'is a leap year.')
elif year % 100 != 0 and year % 4 == 0:
    print(year, 'is a leap year.')
else:
    print(year, 'is not a leap year.')
```

#P6> WAP to accept marks of three subjects Physics, Chemistry and Maths and calculate the percentage and Display grades according to the below table-

Percentage	Grade
90 - 100	A+
80 - 89	A
70 - 79	B
60 - 69	C
Less than 60	D

```
phy = float(input('Enter Physics Marks :'))
che = float(input('Enter Chemistry Marks :'))
math = float(input('Enter Maths Marks :'))
percent = (phy + che + math)/3
print("Percentage = ", percent, "%")

if (percent >= 90):
    print("Grade = A+")
elif (percent >= 80):
    print("Grade = A ")
elif (percent >= 70):
    print("Grade = B ")
elif (percent >= 60):
        print ("Grade = C  ")
    else:
        print ("Grade = D ")
```

O/P:-



```
Enter Physics Marks: 35
Enter Chemistry Marks: 65
Enter Maths Marks: 40
Percentage :  46.666666666666664
Grade: D
```

#P7) WAP in python to calculate simple interest.

#P8) WAP in python to calculate compound interest.

#P9) WAP in python to calculate surface area of a triangle by taking the length of all three sides as input.

#P10) WAP in python to convert degree celcius to degree fahrenheit.

#P11) WAP in python to print larger of two number.

#P12) WAP in python to print smaller of two number.

#P13) WAP in python to print largest of three number.

$$CI = p * \left[\left(1 + \frac{r}{100}\right)^{t} - 1 \right]$$

$$= p*(((1+r/100)**t)-1)$$

$$\sqrt{x} = x^{0.5}$$

$$\sqrt{16} = 4$$

$$16^{1/2} = 4$$

©SHUBHAM SIR

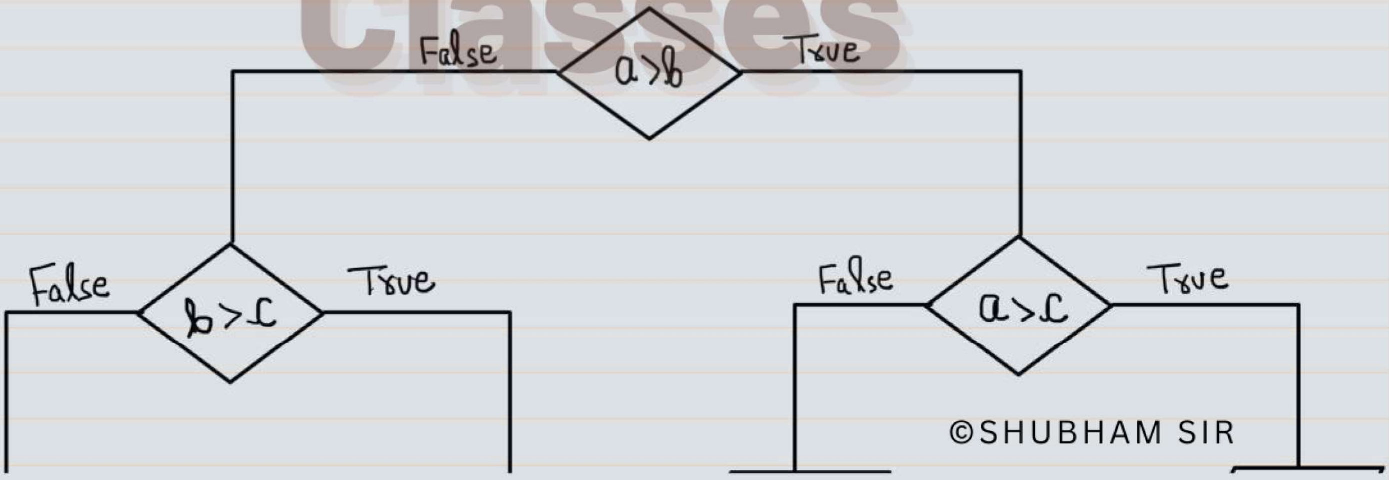
$\Delta area = \sqrt{s(s-a)(s-b)(s-c)}$ ✗

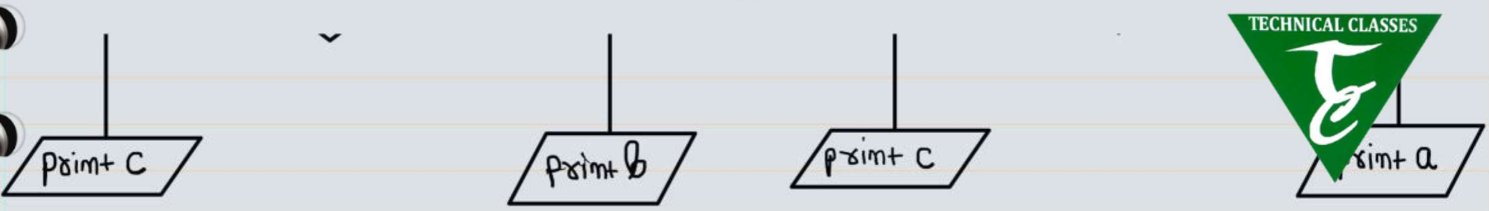
$= (s*(s-a)*(s-b)*(s-c))^{**}\ 0.5$

P13) WAP in python to print largest of three number.

```
a = int(input('Enter First Number: '))
b = int(input('Enter Second Number: '))
c = int(input('Enter Third Number: '))
if (a>b):
    if (a>c):
        print(a, 'is greatest')
    else:
        print(c, 'is greatest')
else:
    if (b>c):
        print(b, 'is greatest')
    else:
        print(c, 'is greatest')
```

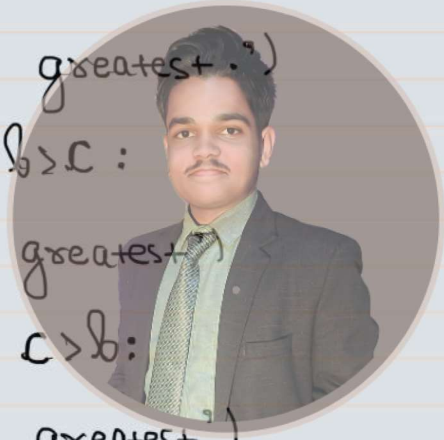
Flowchart:

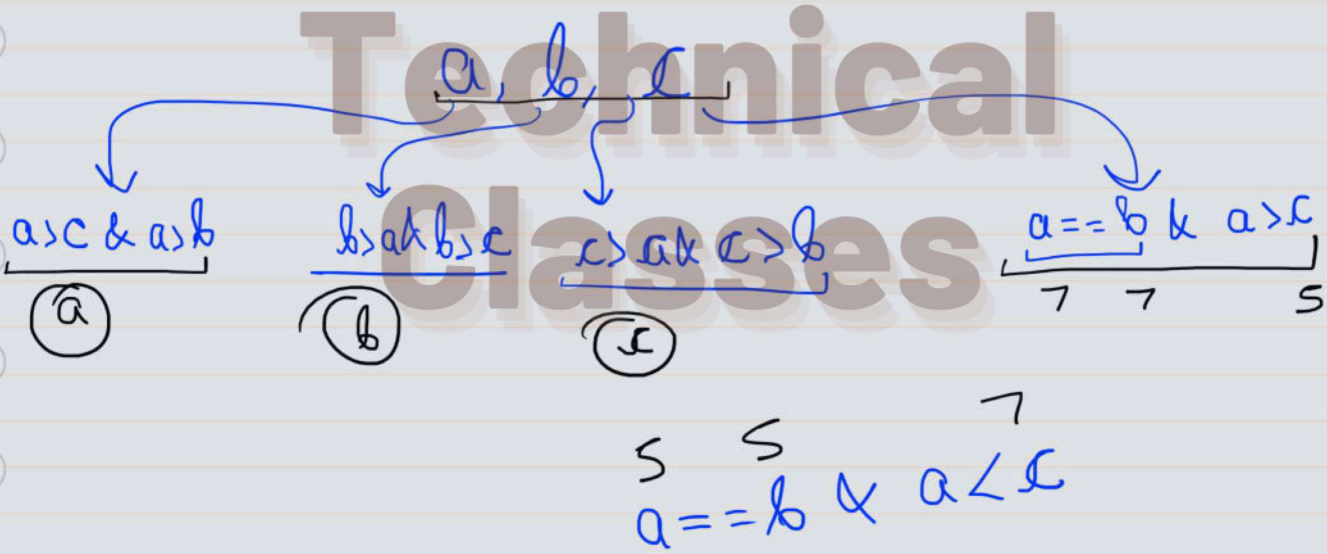




2nd Approach : By using elif

```
a = int(input('Enter First Number: '))
b = int(input('Enter Second Number: '))
c = int(input('Enter Third Number: '))
if a>b and a>c :
    print(a, 'is greatest')
elif b>a and b>c :
    print(b, 'is greatest')
elif c>a and c>b:
    print(c, 'is greatest')
```





बड़ा ?

a b c

i) 5, 6, 7

ii) 5, 5, 7

iii) 5, 7, 7 ⇒ b==c and b>a

iv) 7, 7, 5 ⇒ a==b and a>c

v) 7, 5, 7 ⇒ a==c and a>b

vi) 7, 7, 7 ⇒ else

Indentation in Python :-

→ For the ease of programming and to achieve simplicity python doesn't allow the use of curly braces for the block level code.

→ In python, indentation is used instead of curly braces.

→ If two statements are at the same indentation level, then they are the part of the same block.

→ Generally, four spaces are given to indent the statements.

```
if a>b :
----print ('Hi')
----print ('Hello')
```

Iterative Control Statements :- (Loops)

↳ Repetitive

Problem : We've to print 'Ram' 101 times.

1'st approach

```
1. print ('Ram')
2. print ('Ram')
3. print ('Ram')
   :
   :
100 print ('Ram')
```

2nd Approach

```
execute 101 times :
    print ('Ram')
```

:

100. print ('Ram')

101. print ('Ram')

#Problem2: Print even numbers from 1 to 100.

1st Approach

1 is even ?

2 is even ?

3 is even ?

4 is even ?

:

100 is even ?

2nd Approach

number ⇒ 1 to 100 → loop

num is even ?

Python		Maths
num **2	⇒	num^2
num **0.5	⇒	$\sqrt{num}$

Iterative Control Statements (Loops) :-

→ We can run a single python statement or set of statements multiple times using loop.

→ In python, we've two types of loop :-

i) while

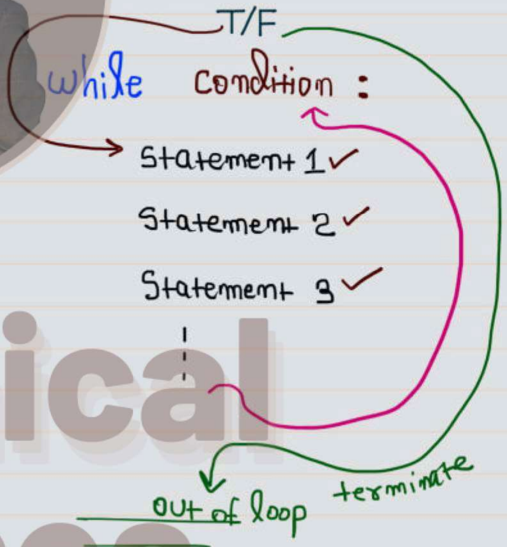
ii) for

i) while loop :-

→ While loop execute set of statements multiple times until the condition is true.

Syntax

```
while condition :
        Statement 1
        Statement 2
        Statement 3
        ⋮
```



→ The while loop will first check the condition, if the condition is true then it will execute the loop block and again it will check the condition and this task will be repeated until the condition becomes false.

→ When the condition becomes false it will terminate p and control goes to the next statement outside the loop.

NOTE :- If the condition of while loop is always true then it'll become an infinite loop.

e.g.

```
i = 5
while i< 10:
      print ('Ram')
```

O/P:-
```
Ram
Ram
Ram
 |
 |
Ram
 |
```

P14) WAP in python to print 'Ram' 5 times.

```
print ('Ram')
print ('Ram')
print ('Ram')
print ('Ram')
print ('Ram')
```

prog1.py

```
i = 1
while i< =5 :
      print ('Ram')
      i = i+1
```

prog2.py

i
~~1~~
~~2~~
~~3~~
~~4~~
~~5~~
6

O/P:-
```
Ram
Ram
Ram
Ram
Ram
```

#P15) WAP in python to print table of a given number.

```
num = int(input(" Enter a number : "))
print('Table of ', num)
i = 1
while i<=10:
    print(num,' x ', i, ' = ', num*i)
    i+=1
```

#P16) WAP in python to print all the even numbers between two given numbers.

```
m = int(input('Enter first number :'))
n = int(input('Enter second number :'))
while m<=n:
    if (m%2 == 0):
        print(m)
    m+=1
```

m	n
~~5~~	14
~~6~~	
~~7~~	
~~8~~	
~~9~~	
~~10~~	
~~11~~	
~~12~~	
~~13~~	
~~14~~	
15	

Dry Run :

```
Enter first number : 5
Enter second number : 14
6
8
10
12
14
```

#P17) WAP in python to check whether a given no. is prime or not.

1 ✓ 11 11 ✓

2, 3, 4, 5, 6, 7, 8, 9, 10

×

$\boxed{\text{num ** 0.5}}$

$\sqrt{\text{num}}$

x $\boxed{2 \text{ to } \sqrt{x}}$ ×

prime number

17 $2 \text{ to } \sqrt{17}$

2 to 4.----

2 to 9 ← 3rd approach 97 2nd approach → 2 to 48

↓ 1st approach

2 to 96

for loop :-

→ A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set or a string).

→ With the help of for loop we can execute a set of statements, once for each item in a list, tuple, set, etc.

Syntax :-

```
for value in sequence :
    ______________
    ______________
    loop body
    ______________
    ______________
```

Example :-

i)
```
x = ['apple', 'banana', 'guava', 'orange']
for fruit in x :
    print(fruit)
```

O/P :-
```
apple
banana
guava
orange
```

Transfer Control Statements :-

→ It is used to transfer the control flow from one line to another line of the program.

→ In python, we have three transfer control structures -

i) break

ii) continue

iii) pass

i) break Statement :-

→ The break is a keyword in python which is used to bring the program control out of the loop.

→ In other words, we can say that break is used to abort the current execution of program and the control goes next line after the loop.

Syntax

```
break
```

example i)

```
i = 1
while i<=5:
    print(i)
    i+=1
print("Bye")
```

i: ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ 6

```
O/P: 1
     2
     3
     4
     5
     Bye
```

```
print("Bye")
```

b

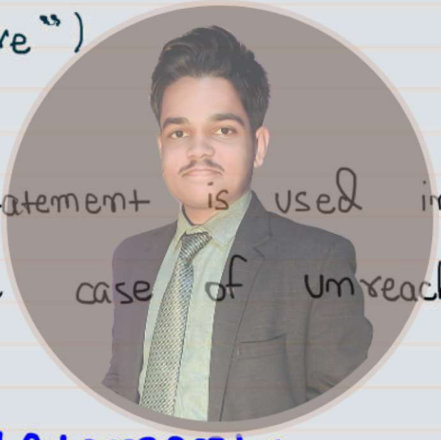
Bye

example (ii)

```
i = 1
while i<=5:
    if i==3:
        break
    print(i)
    i+=1
print("Bye")
```

i
~~1~~
~~2~~
3

O/p:- 1
2
Bye



→ Generally break statement is used inside a condition otherwise there will be the case of unreachable statement.

ii) Continue Statement :-

→ Continue returns the control to the beginning of the loop.

→ The continue statement skips all the remaining statements in the current iteration of the loop and moves the control back to the top of the loop.

→ The continue statement can be used in both while and for loop.

Syntax

```
continue
```

example

i

O/p: 1

example

```
i = 0
while i <= 5:
    i += 1
    if i == 3:
        continue
    print(i)
print('Bye')
```

i
~~0~~
~~1~~
~~2~~
~~3~~
~~4~~
~~5~~
6

O/p:-

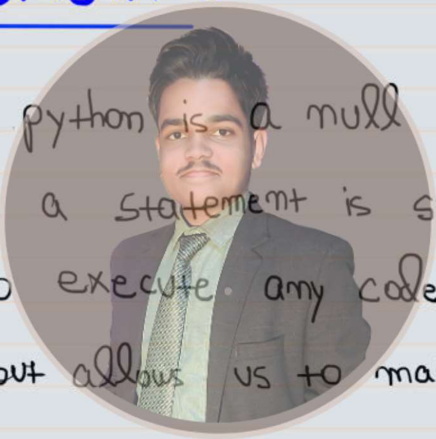
```
1
2
4
5
6
Bye
```

iii) pass Statement :-

→ Pass statement in python is a null operation or a placeholder.

→ It is used when a statement is syntactically required but we don't want to execute any code.

→ It does nothing but allows us to maintain the structure of our program.



E.g.(i)

```
i = 0
while i < 5:
    if (i == 3):
        pass
    else:
        print(i)
    i += 1
```

O/p

```
0
1
2
4
```

eg(ii)

```
num = int (input ('Enter a number'))
if (num == 5):
    pass
else :
    print ('Hi')
print ('Bye')
```

Dry Run 1

```
Enter a number 5
Bye
```

Dry Run 2

```
Enter a number 7
Hi
Bye
```

#P18) Take a list of numbers and calculate the summation of all the elements and print it.

```
x = [5, 3, 7, 2, 8]
Sum = 0
for num in x:
    sum+ = num
print(sum)
```

Unit - 2 : The End!!

Unit - 3

String :-

→ String is the collection of the sequence of characters enclosed within either single or double or triple quotes.

e.g.

```
'shubham'
" ram kumar "
""" Rajee Nagar
   Patna - 24 """
```

→ To create a String variable in python, we just have to store string value in a variable.

e.g.

```
name = 'shubham'
```

String Indexing :-

```
name = 'shubham'
print (name)
print (name[1])   #h
print ( name [3])   #b
print ( name [-2])   #a
```

name →

-7	-6	-5	-4	-3	-2	-1
s	h	u	b	h	a	m
0	1	2	3	4	5	6

Memory representation

→ In python, indexing starts from zero and goes left to right.

→ There is also negative indexes in python, which starts from -1 and goes right to left.

→ The first element or the leftmost element have index 0.

→ The last element or the rightmost element have index -1.

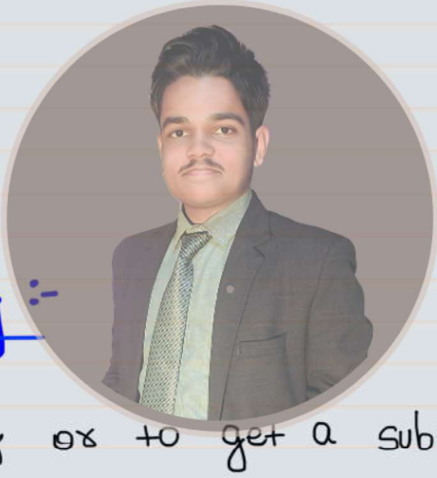
→ To access the individual character of the string slice operator [] is used.

Syntax

```
string [index]
```

example

```
name [5]
```



String Slicing :-

→ To slice the string or to get a substring, we use slice operator [] with colon (:).

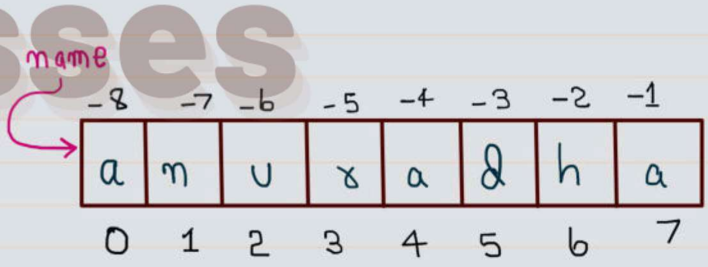
Syntax

```
str [start : end]
```

example

```
name = " anuradha "
print(name)
print ( name[0 : 3] ) # anu
```

Here, the end index



name

-8	-7	-6	-5	-4	-3	-2	-1
a	n	u	r	a	d	h	a
0	1	2	3	4	5	6	7

©SHUBHAM SIR

is always exclusive

→ While slicing the String we can leave start index as well as end index as blank.

→ If we omit the start index, the substring will be considered from first character and if we omit the end index, the substring will be considered till last character.

Example

name

s	h	u	b	h	a	m
0	1	2	3	4	5	6

```
name = 'shubham'
print (name [1:5]) #hubh
print (name [ :5]) #shubh
print (name [2: ]) #ubham
print (name [ : ]) #shubham
```

→ While slicing a string we can also use step if we want to skip characters.

Syntax

```
str [ start : end : step]
```

by default value is 1.

example

```
name = 'Sumankumari'
y = name [2:9:2]
```

```
y = name [2:9:2]
              ↑ Step
print(y)  #mmua
```

name →

		✓	✗	✓	✗	✓	✗	✓		
S	u	m	a	m	k	u	m	a	r	i
0	1	2	3	4	5	6	7	8	9	10

NOTE :- String object is immutable in python i.e the value of string can't be changed after creation.

e.g.

```
fruit = 'apple'
fruit[1] = 'k'   X error
```

String Repeatation :-

→ We can repeat a string multiple times using repeatition operator. '*'

example

```
y = 'ram'
print ( y*3)
```

O/p: ramramram

©SHUBHAM SIR

NOTE:-

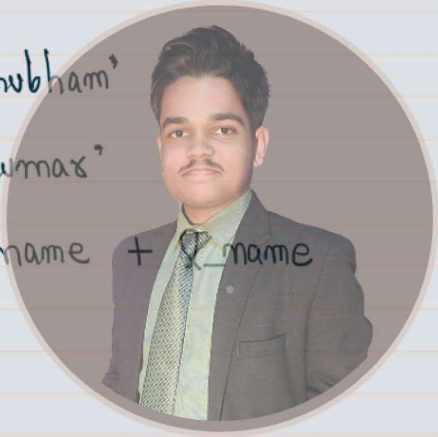
number * number	multiplication
String * number	repeatition

String Concatenation :-

→ We can join or concatenate two string using concatenation operator '+' in python.

e.g.

```
f_name = 'shubham'
l_name = 'kumar'
name   = f_name + l_name
print(name)
```



o/p:- shubhamkumar

String Membership:-

e.g.i)

```
name = 'anuradha'
print ('anu' in name)        # True
print ('krishna' in name)    # False
```

e.g.ii)

```
name = 'aryabhatta'
print ('arya' not in name)     # False
print ('ravan' not in name)        # True
```

String Built-in Methods :-

→ Python has a set of built-in methods that you can use on strings.
↳ predefined

→ All String methods returns new values. They do not change the original string as it is immutable.

i) capitalize ()

→ The capitalize() method returns a string where the first character is uppercase and the rest is lowercase.

e.g.

```
name = 'shuBhaM'
x = name.capitalize()
print(x)
```

O/P: Shubham

ii) count ()

→ The count method returns the number of times a specified value appears in the string.

Syntax

$$\texttt{str.count(value, } \underbrace{\texttt{start, end}}_{\text{optional}}\texttt{)}$$

e.g.(i)

e.g.(i)

```
txt = 'My is mine , yours is your.'
x = txt.count('is')
y = txt.count('is',6,25)
print(x)
print(y)
```

O/P: 2
1

e.g.(ii)

```
name = 'shubham'
print ( name.count ('h'))
print ( name.count ('h', 2, 5))
```

name →

s	h	u	b	h	a	m
0	1	2	3	4	5	6

O/P: 2
1

iii) find()

→ The find method finds the first occurrence of the specified value.

→ It returns -1 if the value is not found.

Syntax

$$\texttt{str.find(value , }\underbrace{\texttt{start}, \texttt{end}}_{\text{optional}}\texttt{)}$$

e.g.

name = 'shubham'

name →

E.g.

```
name = 'shubham'
x = name.find('h')
print(x)
x = name.find('h', 2, 6)
print(x)
```

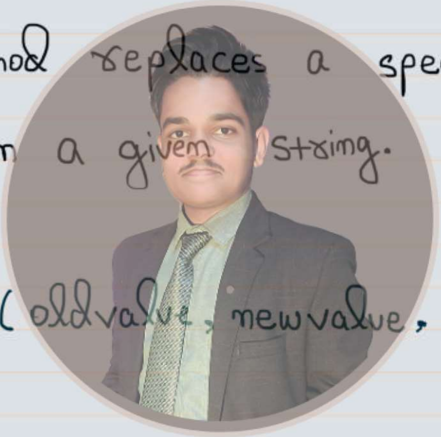
name

s	h	u	b	h	a	m
0	1	2	3	4	5	6

O/P: 1
4

iv) replace()

→ The replace() method replaces a specified phrase with another specified phrase in a given string.



Syntax

```
str.replace(oldvalue, newvalue, count)
```

$\underbrace{\text{count}}_{\text{optional}}$

E.g. i)

```
name = 'Shubham kumar'
name2 = name.replace('kumar', 'kumari')
print(name)
print(name2)
```

O/P: shubham kumar
shubham kumari

E.g. ii)

txt = ' I love you, you love me, we love us.'

E.g(ii)

```
txt = 'I love you, you love me, we love us.'
txt = txt.replace('love','hate', 2)
print(txt)
```

O/P: I hate you, you hate me, we love us.

v) lower()

→ The lower() method returns a string where all characters are in lowercase.

→ Symbols and numbers are ignored.

e.g(i)

```
name = 'GavrAV'
name2 = name.lower()
print(name2)
```

O/P: gaurav

e.g(ii)

```
name = 'adiTyA2@'
print(name.lower())
```

O/P: aditya2@

vi) upper()

→ The upper() method returns a string where all characters are in upper case.

→ Symbols and numbers are ignored.

e.g(i)

```
name = 'GavrAV'
name2 = name.upper()
print(name2)
```

e.g(ii)

```
name = 'adiTyA2@'
print(name.upper())
```

O/P: ADITYA2@

```
print (name2)
```

o/p: ADITYA2@

o/p: GAURAV

String Traversing :-

→ Traversing a String means visiting each element i.e character once.

→ We can use for loop to traverse a String in python.

e.g

```
name = 'Shubham'
for char in name:
    print(char)
```

o/p:
```
S
h
u
b
h
a
m
```

List :-

→ Lists are used to store multiple items in a single variable.

→ The list can contain data of different types.

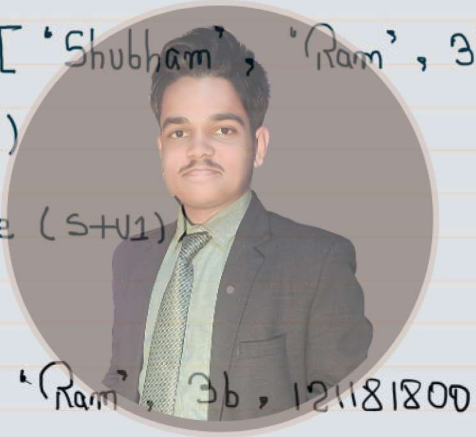
→ The items stored in the list are separated with a comma (,) and enclosed within square bracket [ ].

Syntax

```
list_name = [ item1 , item2, item3 , - - - - ]
```

example

```
stu1 = [ 'Shubham', 'Ram', 36, 1211818001, 96]
print(stu1)
print ( type (stu1))
```



O/p :-

```
[ 'Shubham', 'Ram', 36 , 1211818001, 96]
<class 'list'>
```

→ List is mutable i.e we can modify the size and elements of list.

Python List Operations :-

The different operations of list are:

i) Slicing

ii) Repetition

iii) Concatenation

iv) Length

iv) Length

v) Traversing

vi) Membership

i) List Slicing :

→ Just like string, we can perform slicing of list using slice operator [ ] and colon(:).

Syntax

```
list [start : end : step]
```

e.g.

```
l1 = [ 2 , 5.6 , 'hi' , 3 , 23 , 2.4 , 56]
l2 = l1 [1 : 5]
print(l2)                        O/p: [5.6, 'hi', 3, 23]
l2 = l1 [ : : 2]
print(l2)                             [2, 'hi', 23, 56]
```

ii) List Repetition :-

→ We use repetition operator (*) to repeat a list in python.

e.g.

```
l1 = [ 'hi' , 33 , 2.4]
l2 = l1 *2
print(l2)
```

O/p: ['hi', 33, 2.4, 'hi', 33, 2.4]

iii) List Concatenation :-

→ We can concatenate the elements of two lists by concatenation (+) operator.

e.g.

```
l1 = [ 2 , 3.14 , 'hi' ]
l2 = [ 5 , 6.5 , 'bye' ]
l3 = l1 + l2
print (l3)
```

O/p: [2 , 3.14 , 'hi' , 5 , 6.5 , 'bye']

iv) List length:-

→ To find the length of the list i.e total no. of elements in the list , len() function is used.

e.g.

```
l1 = [ 'ram' , 'sita' , 'shubham' ]
print (len (l1))
```

O/p: 3

V.) Traversing / Iterating a List :-

→ The for loop is used to iterate over the list elements.

e.g.

```
l1 = [ 2 , 5.6 , 'hi' , 3 , 23 , 2.4 , 56 ]
for x in l1:
    print(x)
```

O/p:

```
2
5.6
```

O/p:
```
2
5.6
hi
3
23
2.4
56
```

vi) Membership:-

```
l1 = [2 , 5.6 , 'hi' , 3 , 23, 2.4 , 56]
print ( 2 in l1)
print ('bye' in l1)
print ( 2 not in l1)
print ('bye' not in l1)
```

O/p:
```
True
False
False
True
```

#P18) WAP in python to traverse a list without using for loop.

```
li = [5, 2, 10, 15, 9]
i=0
while i< len(li)
    print(li[i])
    i += 1
```

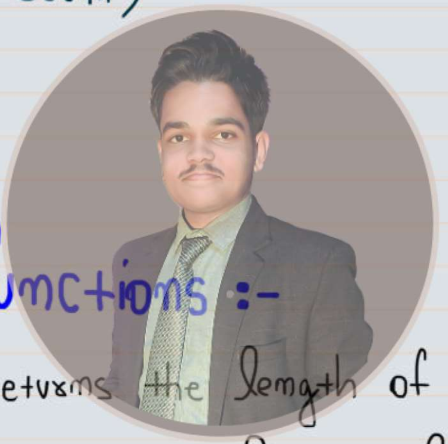
O/p:
```
5
2
```

```
10
15
9
```

#P19) WAP in python to find the length of a list without using len() function.

```
li = [5, 2, 10, 15, 9]
count = 0
for x in li:
    count += 1
print ("Length :", count)
```

o/p: Length : 5



List built-in functions :-

i) len() → It returns the length of the list.

ii) max() → It returns the largest element of the list.

iii) min() → It returns the smallest element of the list.

e.g.

```
x = [35, 'hi', 9, 2, 24]
print (len(x))
print (min(x))
print (max(x))
```

O/p: 5
    2

List is mutable :-

→ We know that list is a mutable data type and hence we can perform the following operations on list -

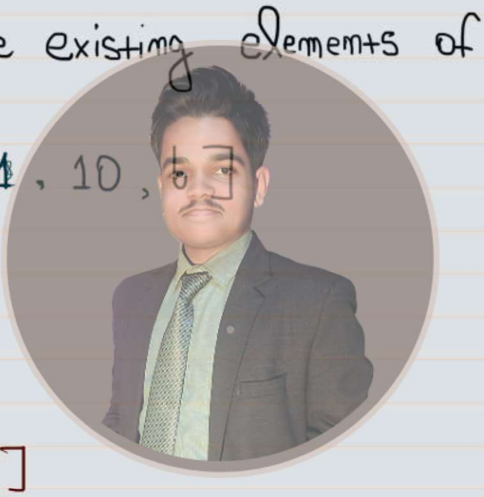
- * Modifying existing elements
- * Adding new elements
- * removing elements

Modifying existing elements :-

→ We can modify the existing elements of list by using indexes.

e.g.

```
li = [2, 5, 1, 10, 6]
li[2] = 7
print (li)
```



```
O/p: [2,5,7, 10, 6]
```

Adding new elements :-

→ We can add elements to the list by using the following methods :

i) append () → It adds an element at the end of the list.

ii) extend () → It adds multiple elements at the end of the list.

iii) insert () → It adds an element at a specific position in the list.

e.g.

```
li = [1, 8, 5]
print (li)
```

O/p :-

```
print (li)
                              O/p :-
li.append (7)
                              [1, 8, 5]
print (li)
                              [1, 8, 5, 7]
li.extend ([3, 9, 2])
                               0 1 2 3  4 5 6
print (li)                    [1, 8, 5, 7, 3, 9, 2]

li.insert (2, 15)             [1, 8, 15, 5, 7, 3, 9, 2]
           ↑   ↑
print (li) index value
```

Removing elements from the list :-

→ We can remove elements from the list by using the following function :

i) remove () → It removes the first occurrence of the element.

ii) pop() → It removes the element at specific index or the last element if no index is specified.

iii) del statement → It deletes an element at a specified position.

e.g.

```
li = [10, 2, 3, 5, 2, 6, 4]
li.remove (2)                 O/p
print (li)              [10, 3, 5, 2, 6, 4]
li.pop()
print (li)              [10, 3, 5, 2, 6]
li.pop(2)
       ↑ index
print (li)              [10, 3, 2, 6]
```

```
del li[1]
print(li)          [10,2,6]
```

Tuple :-

→ Similar to List, Tuples are also used to store multiple items in a single variable.

→ But Tuple is immutable i.e read only data structure meaning that we can't modify the size and value of items in a tuple.

Syntax

```
tuple_name = (item1, item2, item3, .....)

or

tuple_name = item1, item2, item3, .....
```

example

```
car_info = ("mercedes", 2018, '2200cc')
print(car_info)
print(type(car_info))
```

O/P:-

```
('mercedes', 2018, '2200CC')
<class 'tuple'>
```

Accessing Tuple elements :-

→ To access an element in tuple, we use the index operator [].

e.g.

```
tup = "Shubham" , 3.14 . 36
```

e.g.

```
tup = "Shubham" , 3.14 , 36
print (tup[1])
print (tup [-3])
```

O/p:-

```
3.14
Shubham
```

→ Tuples are immutable data types, which means that once they have been generated, their elements cannot be changed.

e.g.

```
data = (26, 'divakar', 'manish', 3.14)
~~data[2] = "gaurav"~~  error
```

Slicing :-

→ We can use a slicing operator [ ] and a colon (:) to access a range of tuple elements.

e.g.

```
data = ( 3, 9, 6, 10, 25, 4, 8 )
t1   =  data [1 : 5]
print (t1)
t2   =  data [ : : 3]
print (t2)          ↑ step
```

O/p: (9, 6, 10, 25)

(3, 10, 8)

Repitition of Tuple :-

→ Just like string and list we can also repeat the elements of tuples by using repitition operator (*).

e.g.

```
tp = 2, 3.14, 'mahi'
print (tp*2)
```

op:

```
(2, 3.14, 'mahi', 2, 3.14, 'mahi')
```

Traversing / Iterating :-

→ We can traverse a tuple using for loop.

e.g.

```
Data = ( 3, 9, 6, 10, 25, 4, 8 )
for x in Data:
    print (x)
```

op:

```
3
9
6
10
25
4
8
```

Tuple methods :-

Tuple methods :-

→ We have very few methods to work with tuple as it is an immutable data structure.

i) count()

ii) index()

i) count()

→ The number of times the specified element occurs in the tuple is returned by the count() function.

e.g.

```
tp = (0, 2, 8, 1, 7, 2, 3, 0, 2)
print (tp.count(2))
print (tp.count(7))
```

O/p:
```
3
1
```

ii) index() :-

→ The first instance of the requested element from the tuple is returned by the index() function.

e.g.

```
tp = (0, 2, 8, 1, 7, 2, 3, 0, 2)
print(tp.index(2))
print(tp.index(0))
```

O/p:
```
1
```

0

Technical
Classes

Set :-

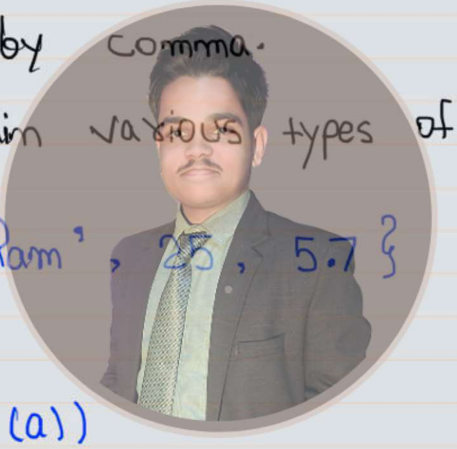
→ In python, set is the unordered collection of data type.

→ It is iterable, mutable (can be modified after creation) and has unique elements.

→ In set, the order of the elements is undefined ; it may return the changed sequence of the element.

→ The set is created using built-in function set() or a sequence of elements is passed in the curly braces and separated by comma.

→ It can contain various types of values.



eg.

```
a = {'Ram', 25, 5.7}
print(a)
print(type(a))
```

O/p:-

```
{25, 'Ram', 5.7}
<class 'set'>
```

Adding items to the set :-

→ Python provides the add() method and update() method which can be used to add some particular item to the set.

→ The add() method is used to add a single element whereas update() method is used to add multiple elements to the set.

e.g.

```
month = {'Jan', 'Feb', 'Mar', 'Apr', 'May'}
month.add('June')
print(month)
month.update(['July', 'Aug', 'Sep'])
print(month)
```

O/p:-
```
{'Jan', 'Feb', 'Mar', 'Apr', 'May', 'June'}
{'Jan', 'Feb', 'Mar', 'Apr', 'May', 'June', 'July', 'Aug', 'Sep'}
```

Removing items from set :-

→ Python provides the discard() method and remove() method which can be used to remove items from the set.

→ The difference between the function, using discard() method, if the item does not exist in the set then the set remain unchanged, whereas remove() method will throw an error.

e.g.(i)

```
month = {'Jan', 'Feb', 'Mar', 'Apr', 'May'}
month.discard('Apr')
print(month)
month.remove('Feb')
print(month)
```

O/p:-
```
{'Jan', 'Feb', 'Mar', 'May'}
{'Jan', 'Mar', 'May'}
```

e.g.(ii)

```
month = {'Jan', 'Feb', 'Mar', 'Apr', 'May'}
month.discard('Nov') ✓
month.remove('Nov') X
                    ↳ error
```

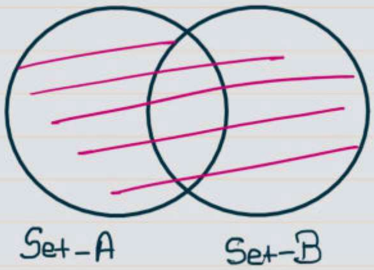
NOTE :- Set object is not subscriptable that is we can't access elements by using index.

Python Set Operations :-

→ In python, we can perform various mathmatical operations such as union, intersection and difference.

↳ Python provides the facility to carry out these operations with the help of operators or methods.

Union of two sets :-



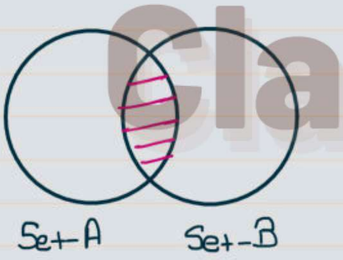
→ To combine two or more sets into one set in python, we use the union() function or union operator (|).

e.g.(i)

```
a = {2, 5, 3, 9}
b = {8, 2, 6, 7}
c = a|b
d = a.union(b)
print(c)
print(d)
```

O/P:-
```
{2, 5, 3, 9, 8, 6, 7}
{2, 5, 3, 9, 8, 6, 7}
```

Intersection of two sets :-



→ The intersection of two sets can be performed by either the intersection operator (&) or intersection() function.

e.g.(ii)

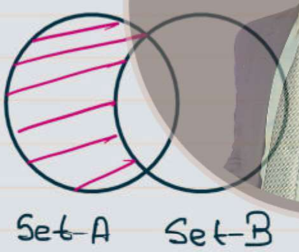
```
a = {1, 5, 6, 9 , 4, 3}
b = {1, 8, 3, 2 , 7, 5}
c = a & b
d = a.intersection(b)
print(c)
print(d)
```

O/P:-

```
{1, 5, 3}
{1, 5, 3}
```

Difference between two sets :-

A-B ⇒



→ The difference of two sets can be calculated by using the subtraction operator (-) or difference() function.

e.g.

```
a = {1, 5, 6, 9 , 4, 3}
b = {1, 8, 3, 2 , 7, 5}
c = a - b
d = a.difference(b)
print(c)
print(Q)
```

O/p:-

```
{6, 9, 4}
{6, 9, 4}
```

Traversing :-

e.g.

```
a = {5, 9, 4, 12}
for x in a:
    print(x)
```

O/p:

```
5
9
4
12
```

Dictionary :-

→ Dictionary is an ordered collection (python 3.7 and later) of a key-value pair of items.

→ Key can hold any primitive data type, whereas value is an arbitrary python object.

→ The items in the dictionary are separated with the comma (,) and enclosed within the curly braces {}.

Syntax

```
var_name = { Key1 : value1 , Key2 : value2 , .... }
```

example

```
a = { 1 : 'Shubham' , 2 : 'Rohit' , 3 : 'Bittu' }
print (a)
print (type (a))
```

O/p:

```
{ 1 : 'Shubham' , 2 : 'Rohit' , 3 : 'Bittu' }
< class 'dict' >
```

→ Dictionary belongs to 'dict' class.

→ Python dictionary is a mutable data-structure that is we can modify the size as well as the elements of a dictionary after creation.

Creating a dictionary :-

→ The simplest approach to create a python dictionary is by

→ The simplest approach to create a python Dictionary using curly brackets.

→ The dictionary can be created by using multiple key-value pairs enclosed within the curly braces {}, and each key is separated from its value by the colon (:).

Syntax

```
dict = { key1 : value1 , key2 : value2 , - - - - }
```

e.g.

```
car = { 'name' : 'mercedes' ,
        'model' : 'benz' ,
        'year' : 2023 }
```

NOTE :- To create an empty dictionary, just put an empty pair of curly braces.

e.g.

```
a = { }
print (type (a))
```

o/p:- <class 'dict'>

Accessing the dictionary values :-

→ The dictionary values can be accessed by using their keys.

e.g.

```
car = { 'name' : 'mercedes' ,
          'model' : 'benz' ,
          'year'  :  2023  }

print (car['model'])
print (car['year'])
```

O/p: benz
2023

* Duplicates Not Allowed :-

→ Dictionaries cannot have two items with the same key.

→ Duplicate values will overwrite existing values.

e.g.

```
car = {'name' : 'mercedes' ,
       'model' : 'benz' ,
       'year' : 2023 ,
       'year' : 2025 }

print (car['year'])
```

O/p: 2025

Updating / Adding Dictionary Values :-

→ The dictionary is a mutable data type, and its values can be updated or added by using the specific keys.

Syntax

Syntax

```
dict [key] = new_value
```

e.g.

```
car = {'name' : 'mercedes',
       'model' : 'benz',
       'year' : 2023 }
print (car)
car ['model'] = 'EV'
print (car)
car ['speed'] = 300
print(car)
```

o/p:-

```
{'name' : 'mercedes', 'model' : 'benz' , 'year' : 2023}
{'name' : 'mercedes', 'model' : 'EV' , 'year' : 2023}
{'name' : 'mercedes', 'model' : 'EV' , 'year' : 2023, 'speed' = 300}
```

→ We can also add new elements in Dictionary by using update() method.

Syntax

```
dict.update ({key: value})
```

e.g.

```
user = { 'id' : 2345,
         'name' : 'ram',
         'pass' : 'qwxy' }
```

```
user.update ({'age' : 25 })
print (user)
```

O/p: {'id' : 2345, 'name' : 'ram' , 'pass' : 'qwxy' , 'age' : 25 }

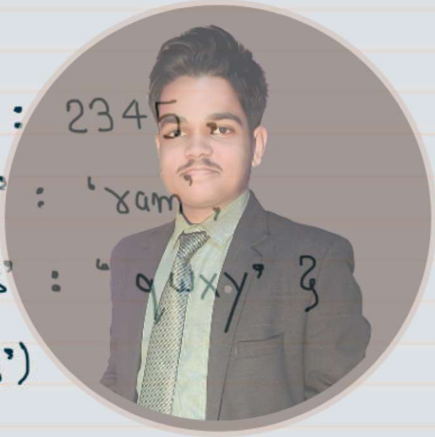
Removing elements from dictionary:-

→ There are several methods to delete elements from a dictionary.

i) pop( ) :-

→ The pop( ) method removes the item with the specified key name.

e.g.



```
user = { 'id' : 2345,
         'name' : 'ram',
         'pass' : "qwxy' }
user.pop ('name')
print (user)
```

O/p: {'id' : 2345, 'pass' : 'qwxy' }

ii) popitem( ) :-

→ The popitem( ) method removes the last inserted item.

e.g.

```
user = { 'id' : 2345,
         'name' : 'ram',
         "pass' : "qwxy' }
user.popitem()
user.popitem()
print(user)
```

O/p:- {'id': 2345, 'name': 'ram'}

iii) del keyword :-

→ The del keyword removes the item with the specified key-name.

e.g.

```
user = { 'id' : 2345,
         'name' : 'ram',
         'pass' : "qwxy" }
del user['id']
print(user)
```

O/p: {'name': 'ram', 'pass': 'qwxy'}

→ The del keyword can also delete the dictionary completely.

e.g.

```
user = { 'id' : 2345,
         'name' : 'ram',
         'pass' : "qwxy" }
```

del user → complete dictionary with all the items will be deleted

print(user['name']) → error

print (user) → error

iv) clear ( ) :-

→ The clear() function empties the dictionary.

e.g.

```
user = { 'id' : 2345,
         'name' : 'ram',
         'pass' : "qwxy' }
user.clear()
print (user)
```

o/p:- { }

Traversing Dictionary :-

→ A dictionary can be iterated using for loop as given below.

e.g

```
user = { 'id' : 2345,
         'name' : 'ram',
         'pass' : "qwxy' }
for x in user:
    print (x)
```

keys

dict [key]

value

o/p:-

```
id
name
pass
```

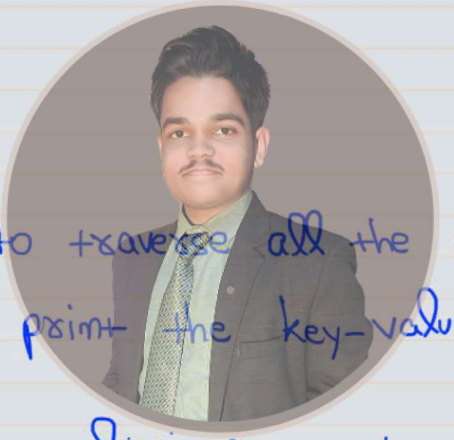
In the above example, only keys will be printed. To print the value of the corresponding key we should write like this-dict [key]

e.g

```
user = { 'id' : 2345,
         'name' : 'ram',
         'pass' : 'qwxy' }
for x in user:
    print(user[x])
```

O/p:

```
2345
ram
qwxy
```



#Q.) WAP in python to traverse all the elements of a dictionary and print the key-value pair.

Q.) Can we traverse a dictionary using while loop in python? If yes, then write the code and if not then explain why not?

Built-in Dictionary Methods:-

→ There are several predefined methods which can be used on dictionary object.

i) clear()

→ It is used to delete all the items of the dictionary.

ii) copy()

→ It returns a shallow copy of a Dictionary.

iii) keys()

→ It returns all the keys of a Dictionary.

e.g

```
user = { 'id' : 2345,
         'name' : 'ram',
         "pass' : " qwxy' }
temp = user.copy()
print (temp)
user. clear()
print (user)
print (temp.keys())
```

```
o/p:- {'id' : 2345, 'name' : 'ram', 'pass' : 'qwxy' }
      { }
      dict_keys (['id', 'name', 'pass'])
```

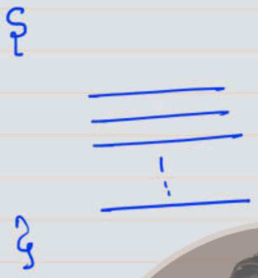
Unit-3: The End !!

Unit - 4

Functions, Modules & Packages

Function :-

→ A function is a block of code that performs a specific task.
↳ set of instructions

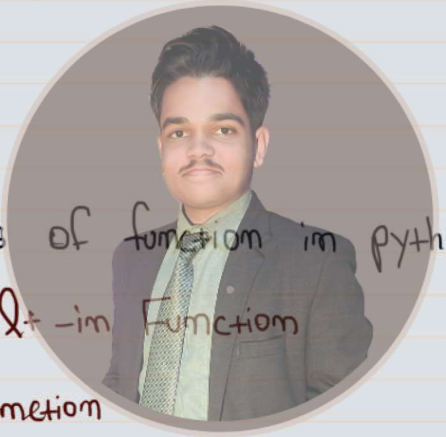


Types of function :

→ There are two types of function in python :

i) Predefined / Built-in Function

ii) User defined function



i) Predefined function :-

→ Those functions which are already present in python library and we can use them directly in our code, known as predefined or built-in function.

e.g. print() , input() , len() , dict() , union()

ii) User-defined function :-

→ Those functions which are created by the python programmer i.e user a/c to the requirement are known as user-defined functions.

Python Function Declaration :-

Syntax

```
def function_name (arguments):
        Statement 1
        Statement 2
           |
           |
        Statement n
        return
```

Here,

def → keyword used to declare a function

function_name → any name given to a function

arguments → any value passed to function

return → It is optional which is used to return a value from function.

e.g

```
def greet ():
    print ("Hello")
```

Calling a Function :-

→ To use a function, we must call it.

→ To call a function, just write the function name with arguments.

Syntax

```
function_name (arguments)
```

example

```
def greet():
    print("Hello")


greet()
```

→ function call

```
O/p: Hello
```

Python Function Example :-

function definition

```
def add():
    a = int(input("Enter a number :"))
    b = int(input("Enter another number :"))
    print("Sum =", a+b)

print("Addition Calculator")
add()
print("Thank you")
```

add() → control transfer → def add():

print("Sum =", a+b) → print("Thank you")

```
O/p:- Addition Calculator
      Enter a number : 5 ↵
      Enter another number : 2 ↵
      Sum = 7
      Thank you
```

→ When the function is called, the control of the program goes to the

→ When the function is called, the control of the program goes to the function definition. All codes inside the function are executed. then, control of the program jumps to the next statement after the function call.

Types of Variables :-

→ There are two types of variables in python :

i) Local variable

ii) Global variable

i) Local Variable :-

→ A variable created inside any function or block is known as local variable.

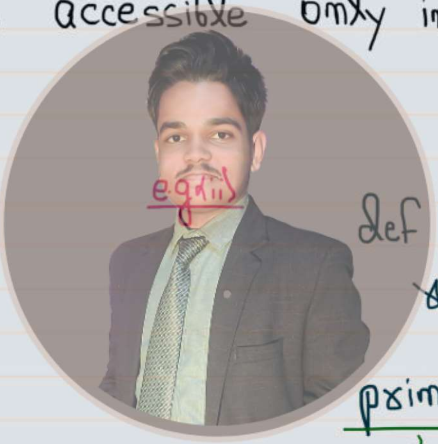
→ A local variable is accessible only inside the block in which it is declared.

e.g. i)

```
def f1():
    x = 10
    print(x)

f1()
```

local to f1()

O/p:- 10



e.g. ii)

```
def f1():
    x =10
print(x) X
```

→ error

ii) Global Variable :-

→ A variable which is declared in the main body of python code ie outside any function or block is known as global variable.

→ Global variables are accessible from everywhere.

e.g.

```
x = 5
```

global

©SHUBHAM SIR

global ←

```
x = 5
def f1():
    x = x + 1
f1()
print(x)
```

o/p :- 6

NOTE :- In python, we can declare local and global variable of same name and python will treat them as two different variables. If there will be an ambiguity in local and global variable, priority will always be given to local variable.

e.g.

```
✓ x = 5
✓ def f1():
    ✓ x = 6
    ✓ print(x)
    ✓ x = x + 1
✓ f1()
✓ print(x)
```

Global x

$\boxed{5}$

x

~~$\boxed{6}$~~

7

o/p :- 6
5

Python Function Arguments :-

→ A function can also have some arguments.

→ An argument is a value that is accepted by a function.

e.g.

```
        2       3
def sum (num1 , num2) :
    ✓ x = num1 + num2
    ✓ print (x)
sum (2, 3)
                (2,3)
```

O/p: 5

Actual Arguments & Formal Arguments :-

→ The arguments which we define in function declaration is known as formal arguments.

→ The arguments which we provide at the time of function call is known as actual arguments.

→ When we modify the value of formal arguments then it doesn't affect the value of actual arguments.

e.g.

formal arguments

```
def fun (num1, num2) :          (5,6)
    num1  += 2
    num2  += 3
    print (num1)
    print (num2)
```

num1 ~~5~~ 7

num2 ~~6~~ 9

```
a= int (input ('Enter a number:'))
b= int (input ('Enter second number:'))
fun (a, b)
print (a)
print (b)
```

actual arguments

a	b
5	6

num2 ~~6~~ 9

print (num1)   print (a)   actual arguments

print (num2)   print (b)

O/p: Enter a number : 5

Enter second number : 6

7

9

5

6

NOTE :- Any changes made to formal arguments doesn't affect the value of actual arguments i.e python function follows call by value mechanism.

return Statement in python :-

→ A python function may or may not return values.

→ If we want our function to return some value to a function call, we use the return statement.

e.g.

```
1.   def add(num1 , num2):
2.       Sum = num1 + num2
3.       return sum
4.   a = 3 , b = 2
5.   c = add (a,b)
6.   print(c)
```

3 2 (3,2) 5 5

num1 $\boxed{3}$ Sum $\boxed{5}$ num2 $\boxed{2}$

O/p: 5

a $\boxed{3}$ b $\boxed{2}$ c $\boxed{5}$

©SHUBHAM SIR

O/p: 5

a	b	c
3	2	5

NOTE :- The return statement also denotes that the function has ended. Any code after return statement is not executed. We can have multiple return statements in a single function but only one of them will be executed depending upon the condition.

Modules :-

→ A document which has many functions and various statements written in python is known as module.

Example : arithmetic.py (user_defined module)

```
def add (num1 , num2):
        sum = num1 + num2
        return sum
def sub (num1 , num2):
        s = num1 - num2
        return s
```

→ It is a python module consists of two functions.

To use this functions in another program we've to import it.

E.g. math.py (predefined module)

sqrt()

tan( )

sin()

e

How to import modules in python?

→ In python, to import functions from a module we use import keyword.

Example : test.py

```
import arithmetic
a = 5
b = 2
print (arithmetic.add (a,b))
print(arithmetic.sub (a,b))
```

O/P: 7
3

→ Modular programming is the practice of segmenting a single, complicated coding task into multiple, simpler, easier-to-manage subtask.

→ A module in python is a collection of various function definitions.

→ To use the content of module in another program, we must import that module by using import keyword.

NOTE:-

.py → extension of python file

.pyc → extension of compiled python code

Import statements in python:-

→ Using the import python keyword and the dot operator, we may import a standard module and can access the defined functions within it.

e.g.

```
import math
print(math.sqrt(16))
print(math.e)
```

module

function defined within math module

O/p: 4.0

2.718281828459045

Importing and Renaming :-

→ While importing a module, we can change its name too.

e.g.

```
import math as mt
print(mt.sqrt(25))
```

O/p: 5.0

→ After renaming a module, we can't use the original name in that particular program.

e.g.

```
import math as mt
print(math.e)
```

error

from ..... import .... Statement :-

→ We can import specific names from a module without importing the module as whole.

e.g.

```
from math import sqrt
print (sqrt(36))
```

o/p: 6.0

Importing all names :-

→ To import all the names from a module within the present namespace, use the * symbol and the from and import keyword.

e.g.

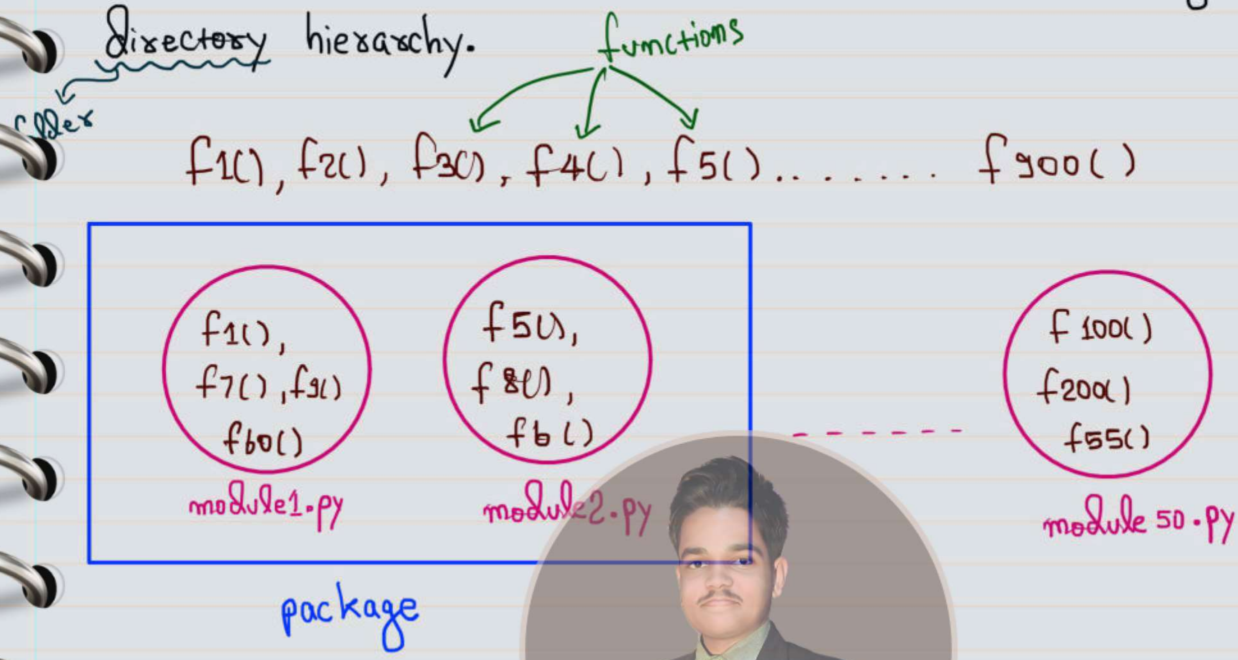
```
from math import *
print ( sqrt (81))
print (tan (pi / 6))
```

o/p: 9.0

0.577

Packages :-

→ Python packages are collection of modules that provide a set of related functionalities, and these modules are organized in a directory hierarchy.



→ In simple terms, packages in python are a way of organizing related modules in a single namespace.

→ Predefined or built-in packages are installed using a package manager pip (a tool for installing and managing python packages).

→ Each python package must contain a file named __init__.py

→ This file contains the initialization code for the corresponding package and may be empty.

→ Some popular python packages are : NumPy, Pandas, and Matplotlib, etc.

Creating Packages :-

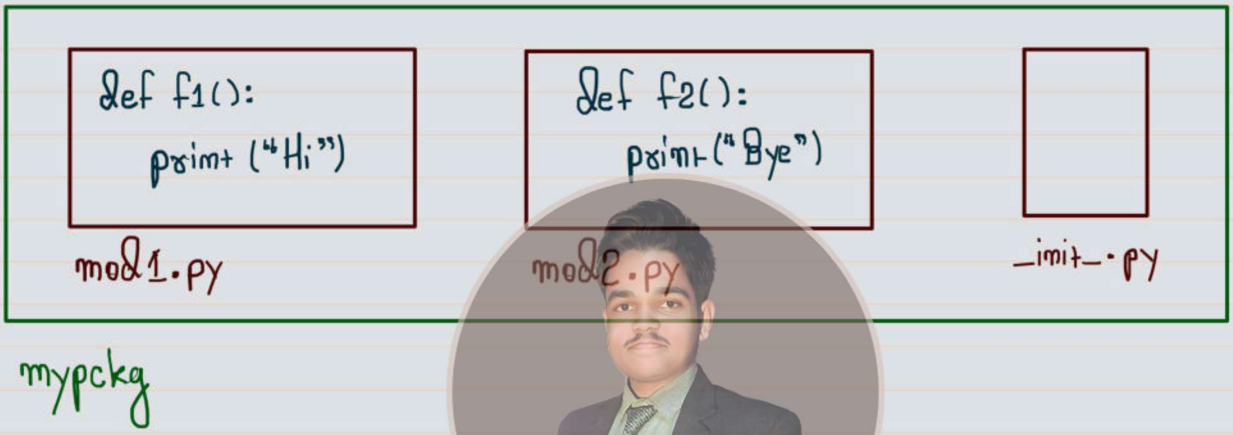
→ To create a package named 'mypckg' that will contain two modules

→ To create a package named 'mypckg' that will contain modules 'mod1' and 'mod2'.

Step 1: Create a directory named mypckg.

Step 2: Inside this directory, create an empty python file i.e `__init__.py`

Step 3: Then create two modules mod1.py and mod2.py in this folder.



```
def f1():
    print("Hi")
```

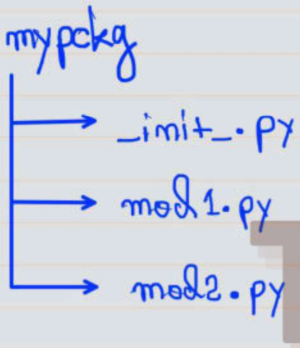
mod1.py

```
def f2():
    print("Bye")
```

mod2.py

__init__.py

mypckg



Example : Importing modules from package

```
from mypckg import mod1
from mypckg import mod2
mod1.f1()
mod2.f2()
```

test.py

test.py

o/p: Hi
 Bye

PIP and PyPI :-

→ PIP is the standard package manager available in python.

→ Although python standard's library comes with many useful packages by default. we are not limited to only those packages.

→ In python, we have a vast repository of packages at PyPI (Python Package Index), which are developed by many great contributors.

→ With the help of pip command, we can easily install and use any of these packages in our python code.

What is pip in python?

→ Python pip is the package manager for python packages.

→ We can use pip to install packages that do not come with python.

→ The basic syntax of pip command is–

```
pip 'arguments'
```

→ Python pip comes preinstalled on 3.4 or later version.

→ To check whether pip is installed or not we use the below command–

```
pip --version
```

→ We can install additional packages by using the python pip install command.

Syntax : pip install package_name

Example : pip install numpy

Syntax : `pip install package-name`

Example : `pip install numpy`

→ We can use the pip show command to display the details of a particular package.

e.g.

```
pip show numpy
```

→ The pip uninstall command uninstalls a particular existing package.

e.g.

```
pip uninstall numpy
```

Unit - 4 : The End !!

Unit - 5

"Object Oriented Programming"

OOPs :-

→ OOP stands for Object Oriented Programming.

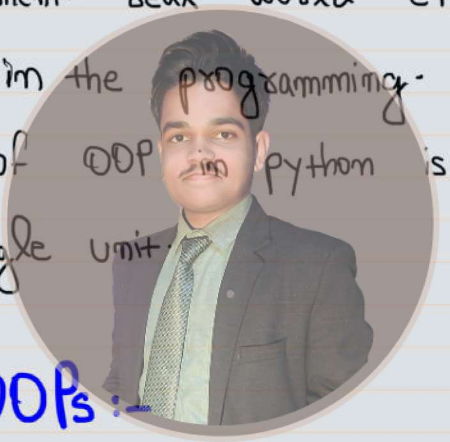
→ It is a programming paradigm which is used in python programming.

↳ way /approach

→ It uses the concept of class and objects.

→ It aims to implement real world entities inheritance, polymorphism, encaptulation, etc. in the programming.

→ The main concept of OOP in python is to bind the data and function in a single unit.



Features of OOPs :-

i) Class

ii) Object

iii) Polymorphism

iv) Encapsulation

v) Inheritance

vi) Data Abstraction

→ Pillars of OOP

i) Class :-

→ A class is just a prototype or the blueprint of objects.

→ Objects are created from class.

→ Objects are Created from class.

→ A class has some attributes and methods.

variables functions

→ For example, if we have a employee class, then it should contain attribute and methods like emp_id, email, salary, etc.

ii) Object :-

→ An object is a reference or implementation of class.

→ It is a real world entity, that has state and behaviour.

variable functions

→ It may be any real world object like the mouse, keyboard, table, chair, pen, etc.

→ For example:

chair

state
- → color
- → type

behaviour
- → sound( )
- → up()
- → down()

→ Objects are created using class in python.

→ Class does not take space in memory and when we define objects then the memory space allocated to that object.

iii) Polymorphism :-

→ It contains two words "Poly" and "morphs".

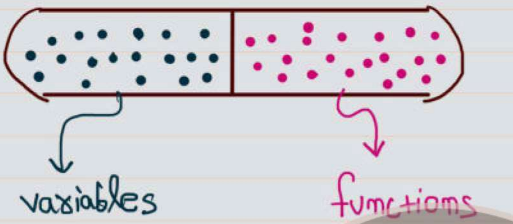
→ 'Poly' means many and 'morph' means shape.

→ By polymorphism, we understand that one task can be performed in different ways.

→ For example – we have a class 'Animal' and all animals speak but they speak differently.

iv) Encapsulation :-

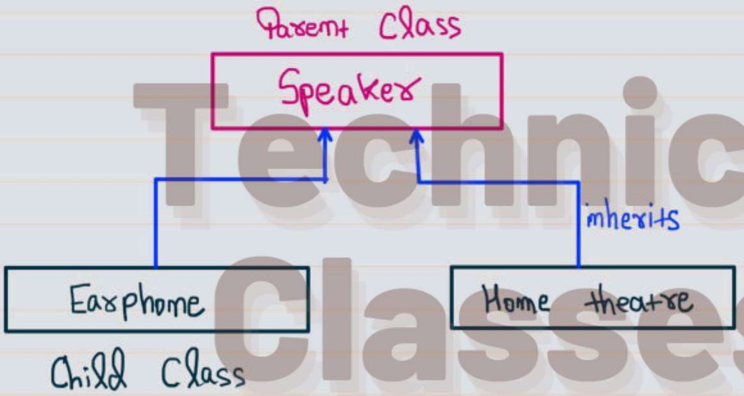
→ Encapsulation is an important concept in OOP in which code (functions) and data (variables) are wrapped together within a single unit.



v) Inheritance :-

→ In inheritance a child object acquires all the properties and behaviour of parent objects.

→ It provides the re-usability of the code.



vi) Data Abstraction :-

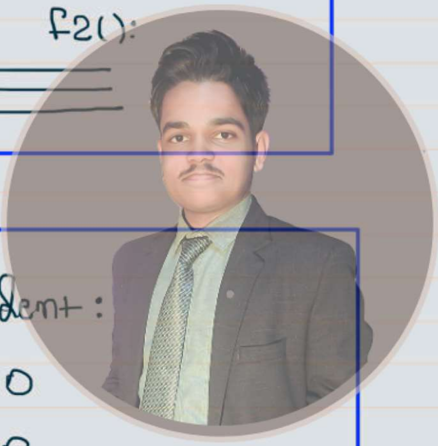
→ Data abstraction or data hiding is a very useful concept of OOP in which we hide internal details from the user and show only functionalities.

Class in Python :-

→ To create a class in python, we use the keyword 'class.'

Syntax

```
class class_name:
        variable1
        variable2
          ⋮

        def f1():
            ═══

        def f2():
            ═══
```



Example

```
class Student:
        roll = 0
        reg = 0
        name = "Shubham"
        def details(self):
            print(self.roll)
            print(self.reg)
            print(self.name)
```

variables (data)

method (code)

Object in Python :-

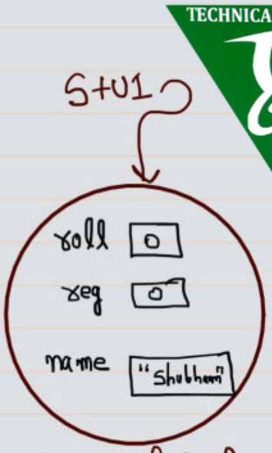
→ To create an object of a class, use the following syntax-

Syntax

reference_variable = class_name ( )

Eg.

stu1 = Student ( )



object of Student class

→ Here, stu1 is a reference variable which is pointing to an object of Student class.

→ By the help of this reference variable, we can access the members of Student object.

Syntax

reference_variable . member_name

e.g.

stu1 . details ( )

o/p: 0
0
Shubham

WAP in python to demonstrate the use of class and objects.

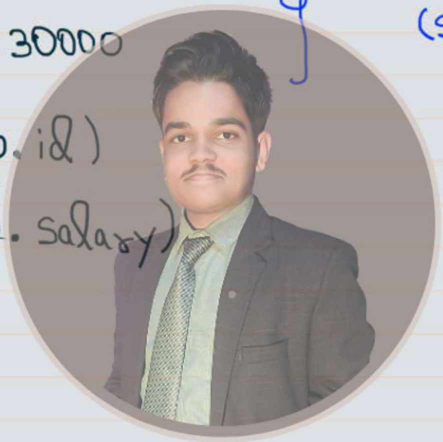
```
class Student:
    roll = 0
    reg = 0
    name = ""
    def getDetails(self):
        print(self.roll)
        print(self.reg)
        print(self.name)
stu1 = Student()
stu1.roll = 50
stu1.reg = 21121314
stu1.name = "Shubham"
stu1.getDetails()
```

Class variables in python :-

- → Class variables are also known as static variables in python.
- → Those variables which are defined inside a class but outside any function or method is known as class variable.
- → Class variables can be accessed by using class name without creating any object.

E.g.

```
class Emp:
    id = 10
    salary = 30000          } class-level variables
                              (static variable)
print(Emp.id)
print(Emp.salary)
```



O/p: 10

```
30000
```

Instance Variables in Python :-

- → Instance variables are also known as object-level variables.
- → Those variables, which are created inside any methods in class declaration are known as instance variable.

Example :

```
class Sample :
    x = 5              ——→ Static variable
    def f1(self):
```

→ Instance variable

```
    def f1(self):
        y = 6
        print(y)
```

→ Instance variable

```
print(Sample.x)
obj = Sample()
obj.f1()
```

O/P: 5
6

Methods in Class :-

There are three types of methods in class :-

i) Class method

ii) Static method

iii) Instance method

Class Method :-

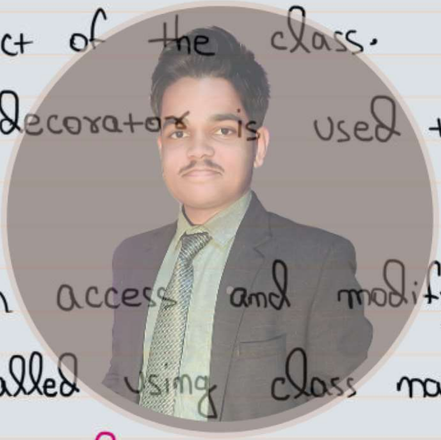
→ Class method is a method in class definition that is bound to the class and not the object of the class.

→ The @classmethod decorator is used to define a class method in python.

→ A class method can access and modify static variables using cls.

→ Class method is called using class name without object.

→ We should provide cls as a argument to class method.



Syntax

```
@classmethod
def class_name (cls) :
    ________
    ________
    ________
```

Example:

```
class Sample :
    x = 5              ]→ static variable
    @classmethod
    def f1 (cls):      ]
```

```
@classmethod
def f1(cls):
    cls.x = 6
    print(x)

def f2(self):
    y = 10
    print(y)
```

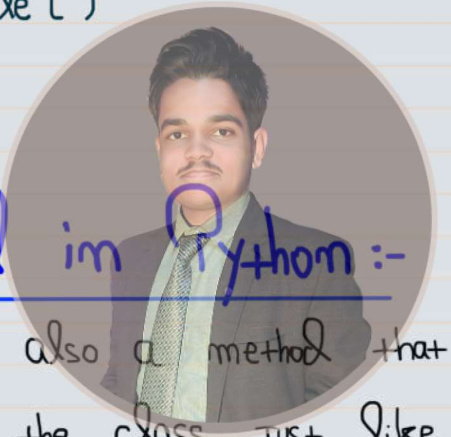
→ Class method

→ Instance method

instance variable ← y = 10

```
Sample.f1() ✓
~~Sample.f2()~~
obj = Sample()
obj.f2()
```

op : 6



Static Method in Python :-

→ A static method is also a method that is bound to a class and not the object of the class just like class method.

→ The main difference between class method and static method is that a static method can't access or modify class variables (static variables).

→ To create a static method in python we use @staticmethod decorator.

Example :

```
class Sample :
    x = 5
    @staticmethod
    def f1():
```

```
    @staticmethod
    def f1():
        print(" Hi ")
```

```
Sample.f1()
```

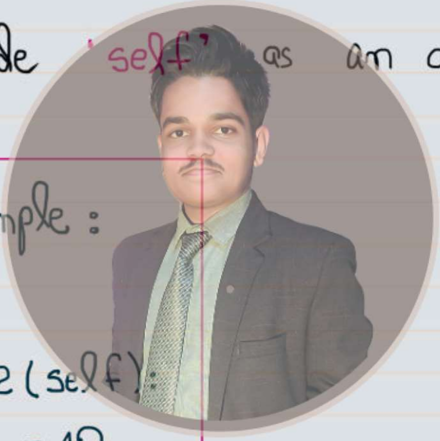
O/P: Hi

Instance Method :-

→ Those methods which are declared directly inside the class without any decorator is known as Instance method.

→ We should provide 'self' as an argument in instance methods.

e.g.



```
class Sample :
    x = 5
    def f2(self):
        y = 10
        print(y)
```

```
k = Sample()
k.f2()
```

O/P: 10

Constructors in Python :-

→ In python, a constructor is a special method which is called automatically when an object is created from a class.

→ To create a constructor in python, we use __init__() method.

$\underbrace{\_\_init\_\_()}$ → initialization method / constructor

Example:

constructor

```
class Sample :
    def __init__(self):
        print ("Hi")
    def f1(self):
        print ('Bye')

obj = Sample()
obj.f1()
```

O/p: Hi
Bye

Destructors in Python :-

→ Destructors are called automatically when an object gets destroyed.

→ In python, destructors are not needed because python has a garbage collector that handles memory management automatically.

collector that handles memory management automatically.

→ The `__del__()` method is used as a destructor in Pyth

→ `__del__()` method is called when all references to the object have been deleted.

Example:

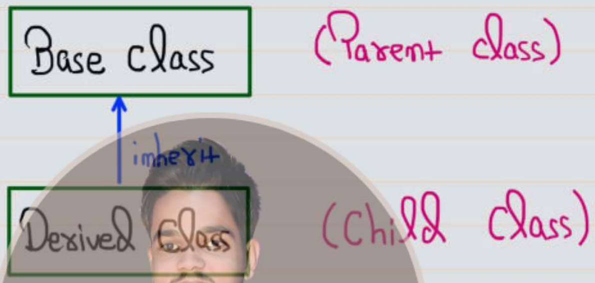
```
              class Sample :
                  def __init__(self):
constructor ←         print ("Hi")
                  def __del__ (self):
destructor            print ('Bye')

              obj = Sample()
```

O/P: Hi
     Bye

Python Inheritance :-

→ Inheritance is an important concept in object-oriented programming which provides code reusability.

→ In python, a child class acquires all the properties of parent class and this mechanism of inheriting the property from parent is known as inheritance.



→ There are many types of inheritance -

i) Single inheritance

ii) Multiple inheritance

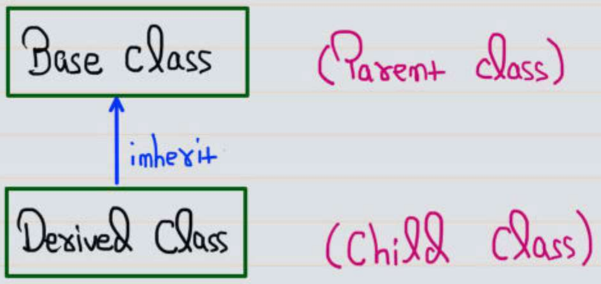
iii) Multilevel inheritance

iv) Hierarchical inheritance

v) Hybrid inheritance

i) Single Inheritance :-

→ When a child class inherits from only one parent class, it is known as single inheritance.



Syntax

```
class  derived_class (base_class):
    ____________
    ____________
    ____________
```

example

Parent Class

```
class Animal :
    def speak (self):
        print ('Animal Speaking')
```

inherits

Child Class

```
class Dog (Animal):
    def bark (self):
        print ('Dog Barking')
```

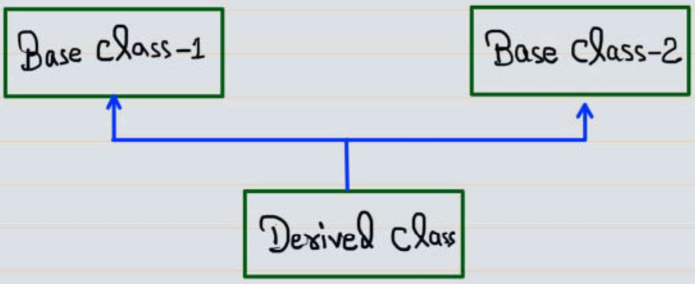
```
obj = Dog()
obj.bark()
obj.speak()
```

O/P:- Dog Barking
Animal Speaking

ii) Multiple Inheritance :-

ii) Multiple Inheritance :-

→ When a child class inherits from multiple parent classes, it is called as multiple inheritance.



Syntax

```
class derived_class (base_class1, base_class2, ----):
```

Example:

```
class Class1:
    def eat(self):
        print('Eating')
```

```
class Class2:
    def drink(self):
        print('Drinking')
```

```
class Sample(Class1, Class2):
    Def sleep(self):
        print('Sleeping')
```

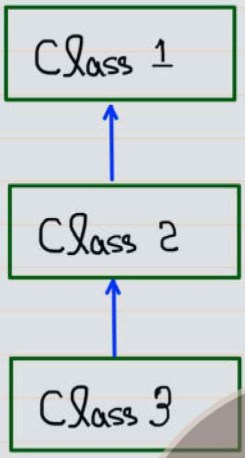
```
s = sample()
s.eat()
s.drink()
s.sleep()
```

O/p: Eating
Drinking

Sleeping

iii) Multilevel Inheritance :-

→ In python, multi-level inheritance is achieved when a derived class inherits another derived class.



→ There is no any limit on the number of levels, upto which, the multi-level inheritance is achieved in python.

Example :

```
class Electronics:
    def f1():
        print('Electronic Items')

class Speaker(Electronics):
    def f2():
        print('Speaker Items')

class Headphone(Speaker):
    def f3():
        print('Headphone Items')
```

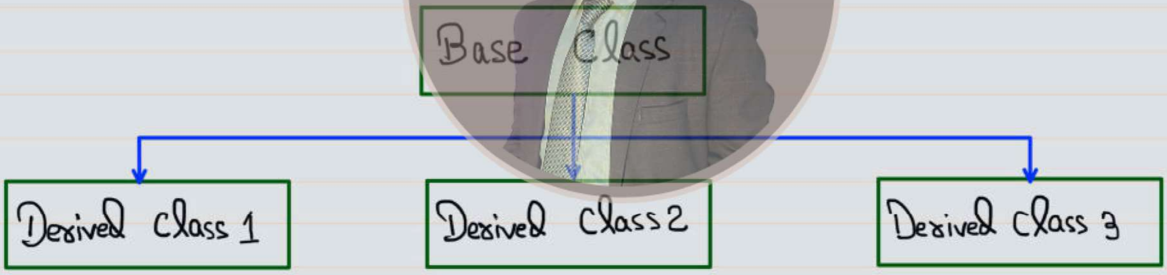
```
obj = Headphone()
obj.f1()
obj.f2()
obj.f3()
```

o/p :-

```
Electronic Items
Speaker Items
Headphone Items
```

iv) Hierarchical Inheritance :-

→ It is a type of inheritance in which a single base class is inherited by multiple derived class.



Example :

```
class Animal :
    def speak (self):
        print ('Animal Speaking')

class Dog ( Animal ):
    def bark (self):
        print ('Dog Barking')
```

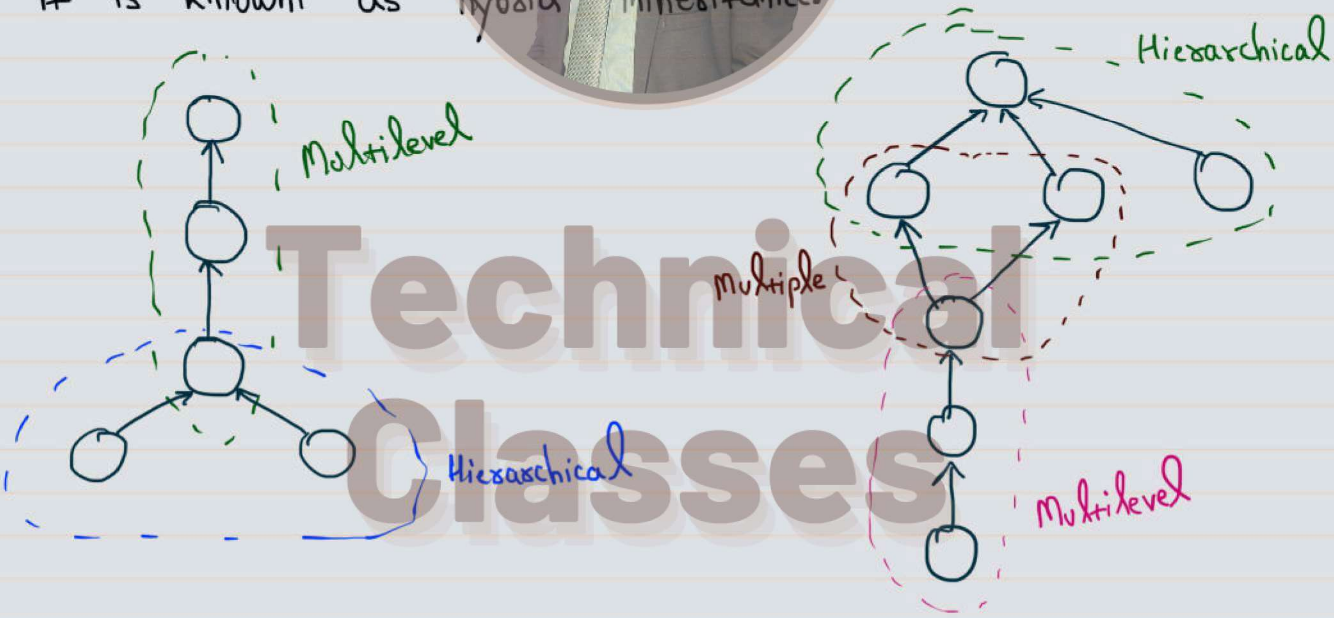
```
class Cat (Animal):
    def meow (self):
        print ('meow meow')

d = dog()
d.speak()
d.bark()
C = Cat()
c.speak()
C.meow()
```

O/P:

```
Animal Speaking
Dog Barking
Animal Speaking
meow meow
```

V.) Hybrid Inheritance :-

→ When we combine two or more types of inheritance then it is known as hybrid inheritance.



Polymorphism :-

→ It contains two words "Poly" and "morphs".

→ 'Poly' means many and 'morph' means shape.

→ By polymorphism, we understand that one task can be performed in different ways.

→ For example - we have a class 'Animal' and all animals speak but they speak differently.

Polymorphism

Compile-time (static) — e.g. method overloading

Run time (dynamic) — e.g. method overriding

i) Compile-time polymorphism :-

→ Compile-time polymorphism performs at the time of compilation of the program.

→ Since python is an interpreted programming language it doesn't support compile-time polymorphism and hence there is no any concept of method overloading in python.

ii) Run-time Polymorphism :-

→ As the name suggested, run-time polymorphism performs at the time of execution of the program.

→ It is also known as Dynamic polymorphism.

time of execution of the program.

→ It is also known as dynamic polymorphism.

→ Method overriding is an example of run-time polymorphism.

Method Overriding :-

Method → function

Overriding → overwrite

→ Method overriding is a process in which child class over-write some specific method of parent class.

→ It is used to change the behaviour of parent methods in child class.

e.g.

```
class Parent :
    def property():
        print('5 Acres Land')
```

```
class Child (Parent):
    def property ():
        print ('3 Acres Land')
        print ('100g Gold')
```

```
p = Parent()
p.property()
c = Child()
c.property()
```

o/p:-

```
5 Acres Land
3 Acres Land
100g Gold
```

Unit - 5 : The End !!

Unit - 6

Exception Handling & File Handling

Syntax Error / Error :-

→ Errors are problems in program due to which the program will stop the execution.

→ Errors are generally caused due to the mistake of the programmers.

→ We can remove the error from our code by modifying the syntax.

e.g.

```
a = 5
if a < 5
    print('Hi')
print('Hello')
```

O/p: Syntax Error

modified code

```
a = 5
if a < 5:
    print('Hi')
print('Hello')
```

O/p: Hello

Exception :-

→ An exception in python is an incident that happens while executing a program that halts the regular execution of the program.

→ When a python code comes across a condition it can't handle, it raises an exception.

→ When a python code throws an exception, it has two options: handle the exception immediately or stop and quit.

e.g.

```
a = 50
```

```
b = a/0
print(b)
```

→ Exception : ZeroDivisionError

→ We can handle exceptions using try-except block in python.

Try-Except Block :-

→ In python, we catch exceptions and handle them using try and except code blocks.

→ The try block contains the code that can raise an exception, while the except block contains the code that handles the exception.

e.g.

```
a = int(input('Enter a number :'))
b = int(input('Enter another number :'))
try:
    c = a/b
except:
    print ('Exception Occurred')
    C='undefined'
print ('Result =', c)
```

Dry Run 1:

Enter a number : 5

Enter another number : 0

Dry Run 2:

Enter a number : 6

Enter another number : 2

```
Exception Occurred                Result = 3
Result = undefined
```

→ If any exception occurred by the code written in try block then except block will be executed and program will flow normally.

→ If no any exception occurred then the except block will not be executed.

→ In a try block there may arise different type of exception and we can handle them differently by using multiple exception blocks.

e.g.

```
try:
    a = int(input('Enter a number :'))
    b = int(input('Enter another number :'))
    c = a/b

except ValueError:
    print('Enter a valid number')
    C = 'undefined'
except ZeroDivisionError:
    print('Exception Occurred')
    C='undefined'
print('Result =', c)
```

Dry Run 1:

```
Enter a number: 6
Enter another number: 0
Exception Occured
undefined
```

Dry Run 2:

```
Enter a number: XI
Enter Valid Number
undefined
```

Dry Run 3:

```
Enter a number: 8
Enter another number: 2
4.0
```

else block :-

→ It is optional in exception handling.

→ It must be written after all except blocks.

→ The else block runs only if no exceptions are raised in the try block.

→ This is useful for code that should execute if the try block succeeds.

finally block :-

→ It is also optional in exception handling.

→ The finally block is used at the end of try-except block.

→ It is always executed irrespective of whether the exception has occurred or not.

e.g.

```
try:
    a = int(input('Enter a number :'))
    b = int(input('Enter another number :'))
    c = a/b
except ValueError :
    print ('Please enter a valid number.')
except ZeroDivisionError:
    print ('Can't divide by zero' )
else :
    print ('Result =', c)
```

```
finally:
        print ('Execution Complete')
```

Dry Run 1: No any exception occurred

```
Enter a number: 9
Enter another number: 3
Result =  3.0           → else block
Execution Complete      → finally block
```

Dry Run 2: ZeroDivisionError Exception occurred

```
Enter a number: 5
Enter another number: 0
Cannot divide by zero   → corresponding except block
Execution Complete      → finally block
```

Dry Run 3: ValueError Exception Occurred

```
Enter a number: amir
Please enter a valid number  → corresponding except block
Execution Complete           → finally block
```

Raising Exceptions :-

→ We can raise an exception in python using raise keyword followed by an instance of the exception class that we want to trigger.

Syntax:

```
raise ExceptionType ("Error Message")
```

example:

```
age = int (input ('Enter age :'))
if (age<0):
    raise ValueError ('Age cannot be negative')
print (age)
```

```
    print (age)
```

Dry Run 1:

```
Enter your age: 15
15
```

Dry Run 2:

```
Enter your age: -5
ValueError: Age cannot be negative
```

File Handling in Python :-

→ Python provides built-in functions for creating, writing and reading files.

→ Two types of files can be handeled in python :-

i) Binary file – Binary files written in binary language i.e 0's and 1's. In this type of file, there is no terminator for a line, and the data is stored after converting it into machine understandable binary language.

ii) Text file – It is a simple plain text file in which each line of text is terminated with a special character called EOL (End of Line), which is the new line character ('\n') in python by default.

File Access Modes :-

→ Access modes govern the type of operations possible in the opened file.

→ It refers to how the file will be used once its opened.

→ These modes also define the location of File Handle in the file.

→ The file handle is like a cursor, which defines from where the data has to be read or written in the file and we can get python output in text file.

There are 6 access modes in python :-

There are 6 access modes in python :-

i) Read only ('r')

ii) Read and write ('r+')

iii) write only ('w')

iv) write and read ('w+')

v) Append only ('a')

vi) Append and read ('a+')

Opening a text file in python :-

→ It is done using the open() function. No module is required to be imported for this function.

Syntax

file_object = open($\overset{\text{raw text}}{r}$"File_name", "Access_Mode")

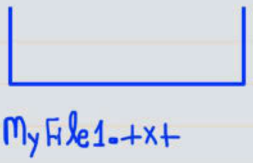
→ The file should exists in the same directory as the python program file, otherwise the full address of the file should be written in place of the file name.

Example: Let's suppose we've a text file named MyFile1.txt in the current working directory.

Shubham Kumar

file1 = open("MyFile1.txt", 'a')

→ The above Statement open 'MyFile1.txt' in append mode and stores its reference



append mode and stores its reference in variable 'file1'.

Closing a text file :-

→ close() function closes the file and frees the memory space acquired by that file.

→ It is used at the time when the file is no longer needed or if it is to be opened in a different file mode.

Syntax :

```
file_object.close()
```

example :

```
file1 = open("MyFile1.txt", "a")
file1.close()
```

Writing a File :-

→ There are two ways to write in a text file in python :-

i) write() function

ii) writelines() function

i) write() :-

→ It inserts the given string in a single line in a text file.

Syntax :

```
file_object.write(string)
```

ii) writelines() :-

→ It is used to inserts multiple strings at a single time.

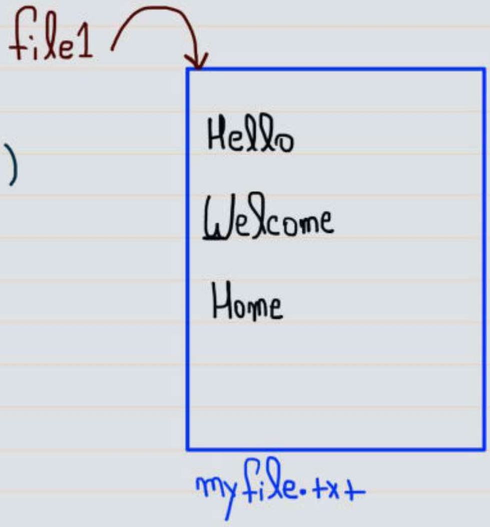
→ For a list of string elements, each string is inserted in the

- It is used to inserts multiple strings at a single t
- For a list of string elements, each string is inserted i he text file.

Syntax: `file_object.writelines([str1, str2, str3, ---])`

Example:

```
file1 = open("myfile.txt", "w")
file1.write("Hello\n")
L = ["Welcome\n", "Home\n"]
file1.writelines(L)
file1.close()
```



Reading from a file :-

- There are three ways to read data from a text file in python:-
  i) read()
  ii) readline()
  iii) readlines()

i) read() :-

1byte ← 1 character

- It returns the read byte in form of a string.
- It reads n bytes, if no n specified, reads the entire file.

Syntax: `file_object.read(n)`

ii) readline( ) :-

→ It reads a line of the file and returns in form of a string.

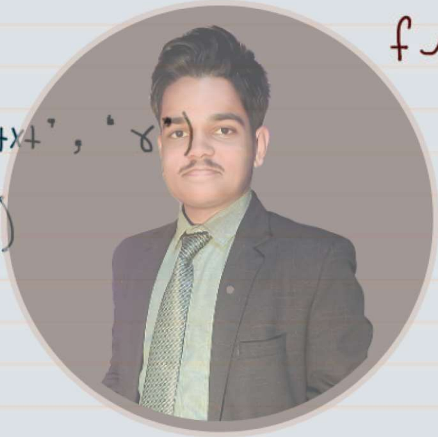
→ For specified n, reads n bytes but atmost a single line.

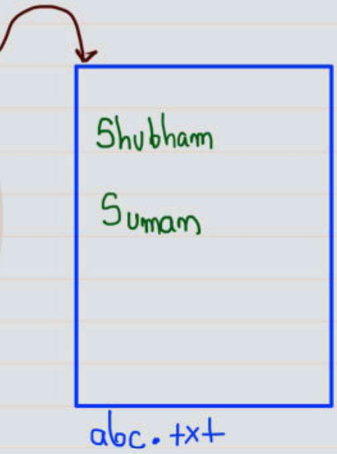
Syntax : file_object . readline (n)

iii) readlines( ) :-

→ It reads all the lines and return them as each line a string element in a list.

Syntax : file_object . readlines ( )





e.g.(i)

```
f = open ('abc.txt', 'r')
print(f. read( ))
```

O/P:-

```
Shubham
Suman
```

e.g.(ii)

```
f = open ('abc.txt', 'r')
print(f.read (5))
```

O/P:- Shubh

e.g.(iii)

```
f = open ('abc.txt', 'r')
print(f. readline (14))
```

O/P :- Shubham

e.g.(iv)

```
f = open('abc.txt', 'r')
print(f.readlines())
```

O/p :- ['Shubham\n' , 'Suman']

Appending to a file in python :-

Let's take a file named 'abc.txt' having 2 lines of content.

```
f = open('abc.txt', 'a')
f.write('\nAll are good')
f.close()
```

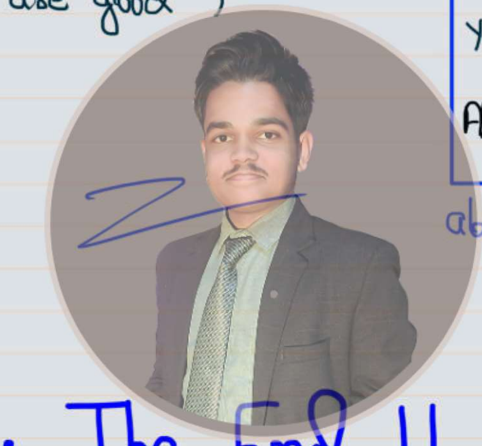
f

I am good.

You are good.

All are good.

abc.txt



Unit - 6 : The End !!

Python : The End !!