BACHELOR OF COMPUTER APPLICATION LAB MANUAL

3rd Semester

MIDNAPORE CITY COLLEGE

ESTD. - 2017

Prepared By

Pure and Applied Science Dept.

Computer Application

MIDNAPORE CITY COLLEGE

INSTRUCTIONS TO STUDENTS

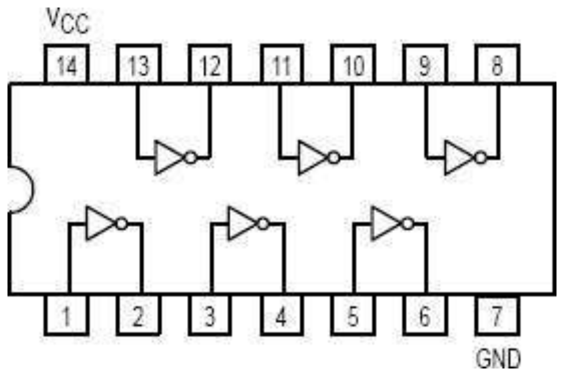
- Before entering the lab, the student should carry the following things (MANDATORY)
    1. Identity card issued by the college.
    2. Class notes
    3. Lab observation book
    4. Lab Manual
    5. Lab Record
- Student must sign in and sign out in the register provided when attending the lab sessionwithout fail.
- Come to the laboratory in time. Students, who are late more than 10 min., will not beallowed to attend the lab.
- Students need to maintain 80% attendance in lab if not a strict action will be taken.
- All students must follow a Dress Code while in the laboratory
- Foods, drinks are NOT allowed.
- All bags must be left at the indicated place.
- Refer to the lab staff if you need any help in using the lab.
- Respect the laboratory and its other users.
- Workspace must be kept clean and tidy after experiment is completed.
- Read the Manual carefully before coming to the laboratory and be sure about what youare supposed to do.
- Do the experiments as per the instructions given in the manual.
- Copy all the programs to observation which are taught in class before attending the labsession.
- Students are not supposed to use floppy disks, pen drives without permission of lab- incharge.
- Lab records need to be submitted on or before the date of submission.

Computer Architecture Laboratory Manual (Paper Code: BCAHMJ03P)

CONTENTS

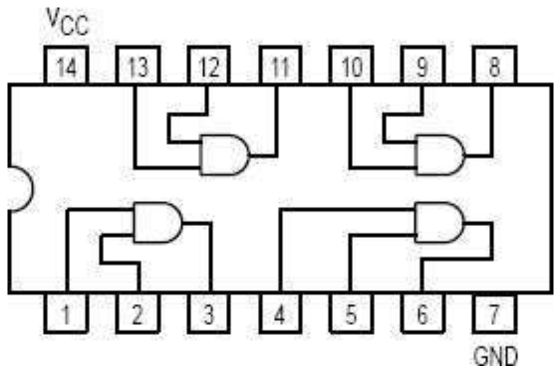
Experiment No

Inverter Gate (NOT Gate) 7404LS



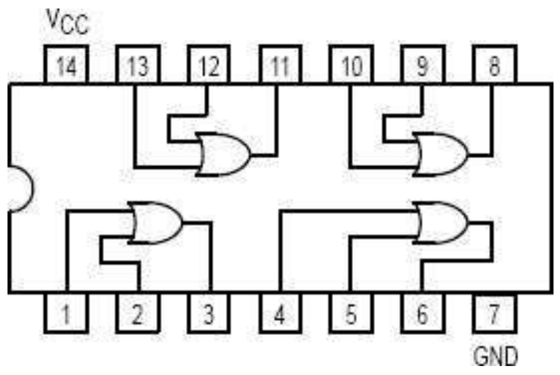
A	O/P	Y1 (V)	Y2 (V)	Y3 (V)	Y4 (V)	Y5 (V)	Y6 (v)
0	1						
1	0						

2-Input AND Gate 7408LS



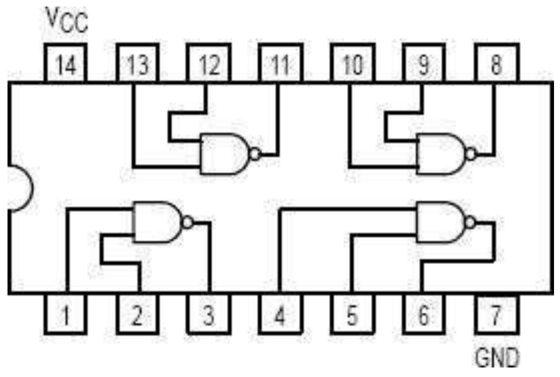
A	B	O/P	Y1 (V)	Y2 (V)	Y3 (V)	Y4 (V)
0	0	0				
0	1	0				
1	0	0				
1	1	1				

2-Input OR Gate 7432LS



A	B	O/P	Y1 (V)	Y2 (V)	Y3 (V)	Y4 (V)
0	0	0				
0	1	0				
1	0	0				
1	1	1				

2-Input NAND Gate 7400LS



A	B	O/P	Y1 (V)	Y2 (V)	Y3 (V)	Y4 (V)
0	0	1				
0	1	0				
1	0	0				
1	1	0				

Experiment No: Date:___/__/

VERIFICATION OF GATES

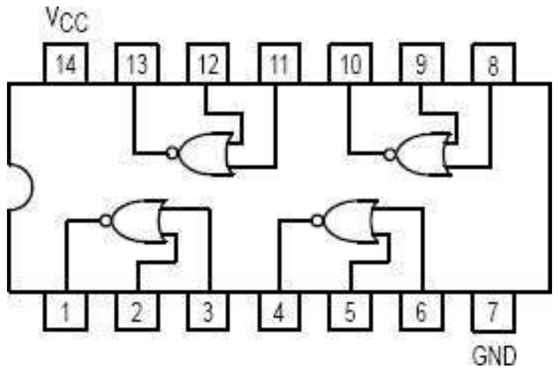
Aim: - To study and verify the truth table of logic gates

Apparatus Required: -
All the basic gates mention in the fig.

Procedure: -

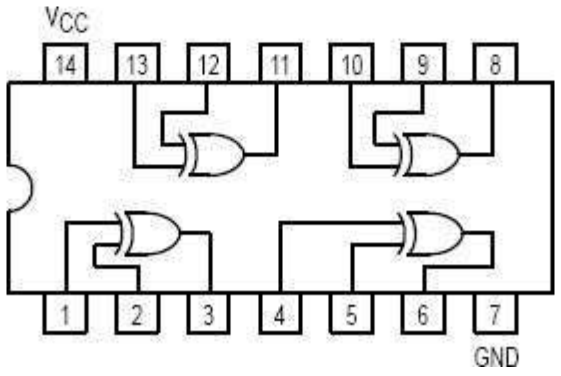
1. **Place the IC-on-IC Trainer Kit.**
2. **Connect V_{CC} and ground to respective pins of IC Trainer Kit.**
3. **Connect the inputs to the input switches provided in the IC Trainer Kit.**
4. **Connect the outputs to the switches of O/P LEDs,**
5. **Apply various combinations of inputs according to the truth table and observe condition of LEDs.**
6. **Disconnect output from the LEDs and note down the corresponding multimeter voltage readings for various combinations of inputs.**

2-Input NOR Gate 7402LS



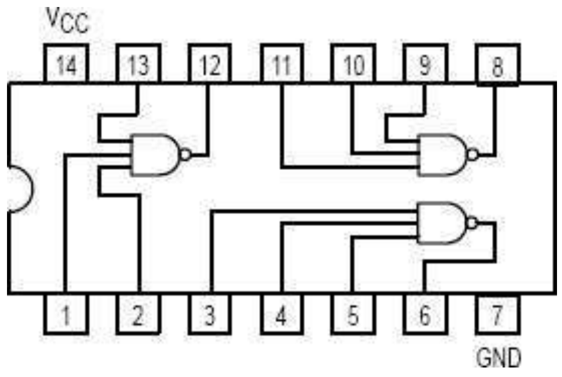
A	B	O/P	Y1 (V)	Y2 (V)	Y3 (V)	Y4 (V)
0	0	1				
0	1	0				
1	0	0				
1	1	0				

2 Input EX-OR Gate 7486LS



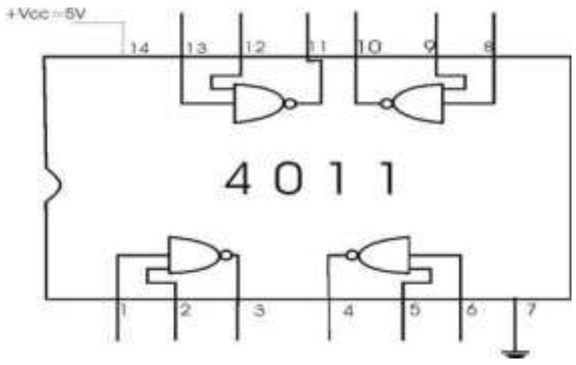
A	B	O/P	Y1 (V)	Y2 (V)	Y3 (V)	Y4 (V)
0	0	0				
0	1	1				
1	0	1				
1	1	0				

3 Input NAND Gate 7410LS



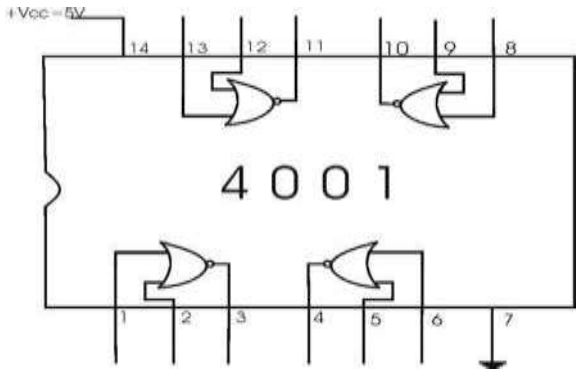
A	B	C	O/P	Y1 (V)	Y2 (V)	Y3 (V)
0	0	0	1			
0	0	1	1			
0	1	0	1			
0	1	1	1			
1	0	0	1			
1	0	1	1			
1	1	0	1			
1	1	1	0			

2-Input NAND Gate CD4011



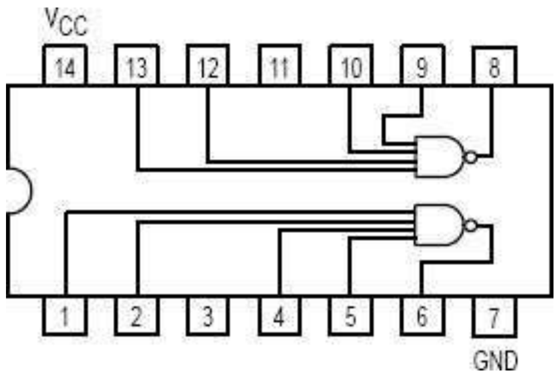
A	B	O/P	Y1 (V)	Y2 (V)	Y3 (V)	Y4 (V)
0	0	1				
0	1	1				
1	0	1				
1	1	0				

2-Input NOR Gate € CD4001



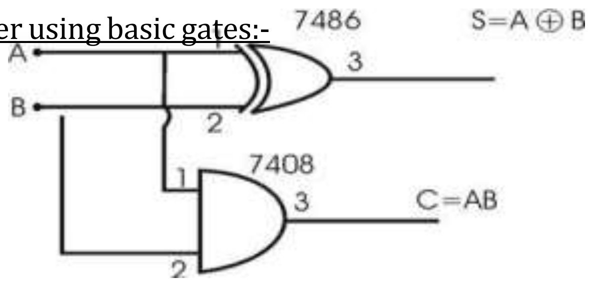
A	B	O/P	Y1 (V)	Y2 (V)	Y3 (V)	Y4 (V)
0	0	1				
0	1	0				
1	0	0				
1	1	0				

4-Input NAND Gate 7420LS

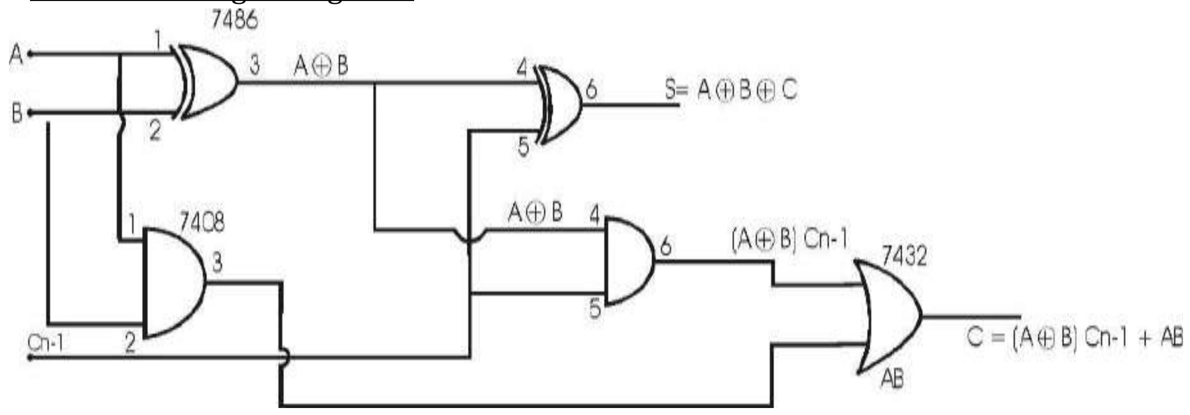


A	B	C	D	O/P	Y1 (V)	Y2 (V)	Y3 (V)
0	0	0	0	1			
0	0	0	1	1			
0	0	1	0	1			
0	0	1	1	1			
0	1	0	0	1			
0	1	0	1	1			
0	1	1	0	1			
0	1	1	1	1			
1	0	0	0	1			
1	0	0	1	1			
1	0	1	0	1			
1	0	1	1	1			
1	1	0	0	1			
1	1	0	1	1			
1	1	1	0	1			
1	1	1	1	0			

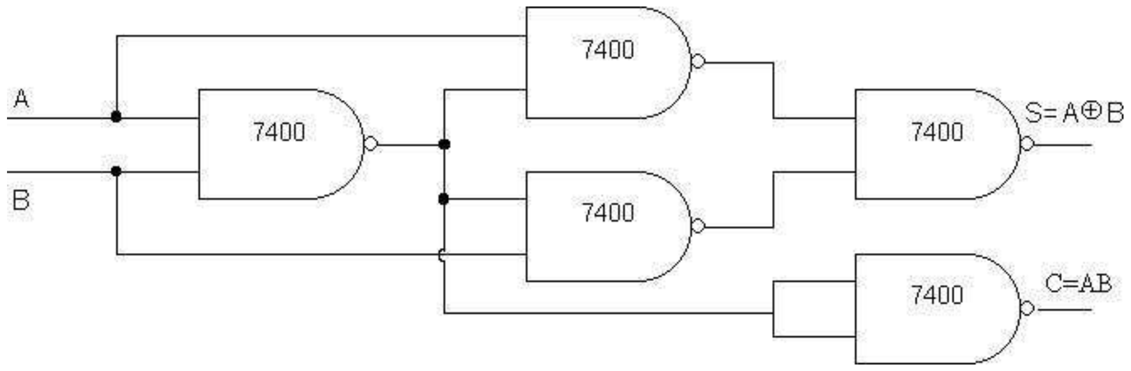
Half Adder using basic gates:-



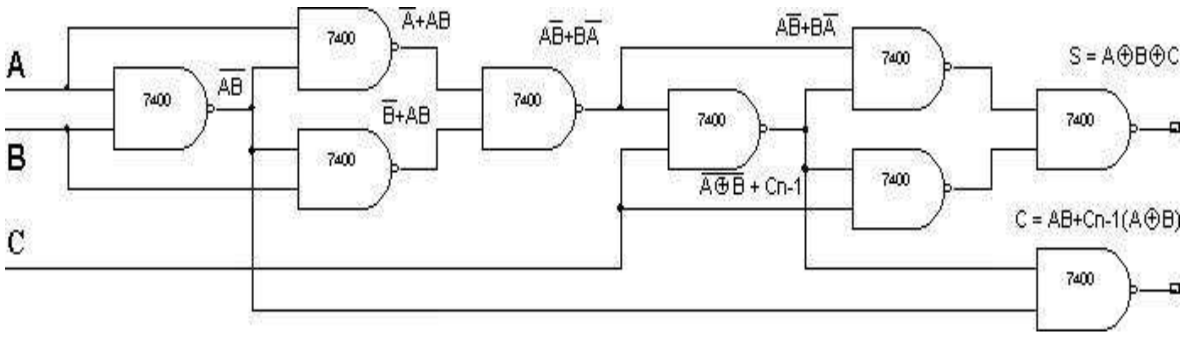
Full Adder using basic gates:-



Half Adder using NAND gates only:-



Full Adder using NAND gates only:-



Experiment No: Date:___/__/

HALF/FULL ADDER & HALF/FULL SUBTRACTOR

Aim: - To realize half/full adder and half/full subtractor.

- i. **Using X-OR and basic gates**
- i. **Using only nand gates.**

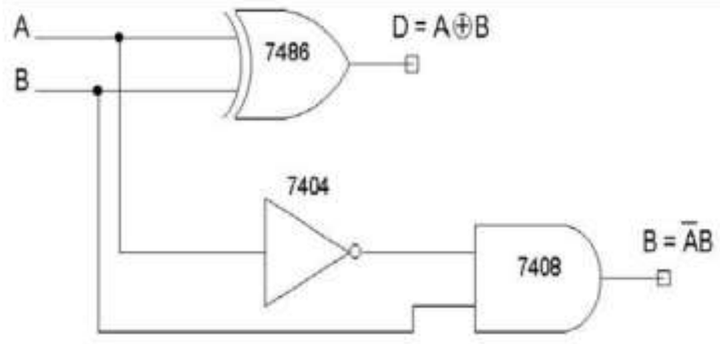
Apparatus Required: -

IC 7486, IC 7432, IC 74 08, IC 7400, etc.

Procedure: -

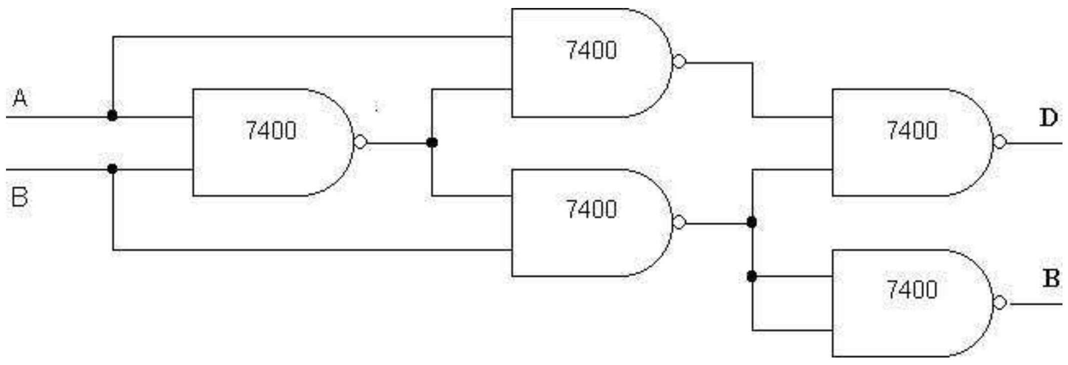
1. **Verify the gates.**
2. **Make the connections as per the circuit diagram.**
3. **Switch on V_{CC} and apply various combinations of input according to the truth table.**
4. **Note down the output readings for half/full adder and half/full subtractor sum/difference and the carry/borrow bit for different combinations of inputs.**

Using X – OR and Basic Gates (a)Half Subtractor

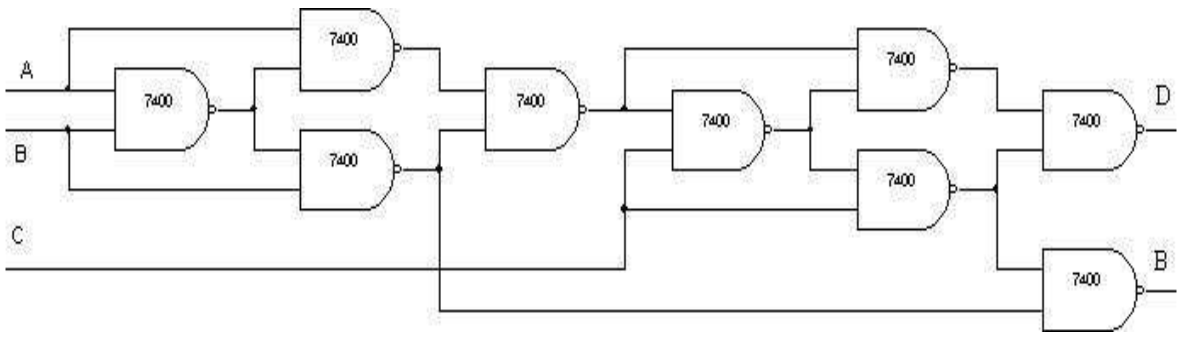


Full Subtractor

(i) Using only NAND gates (a) Half subtractor



(b) Full Subtractor



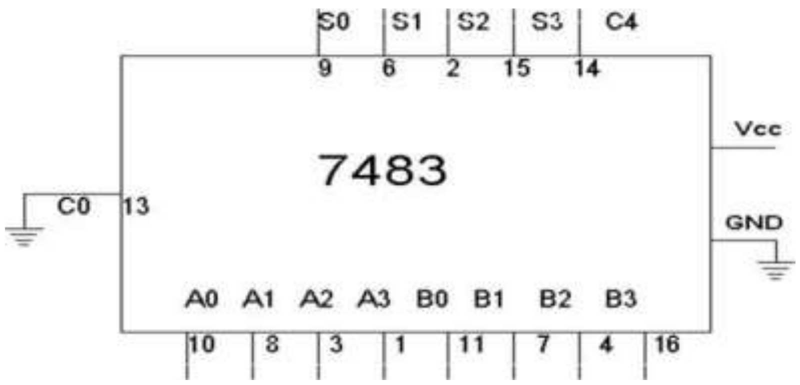
Half Adder					
A	B	S	C	S(V)	C(V)
	0	0	0		
0	1	1	0		
1	0	1	0		
1	1	0	1		

Half Subtractor					
A	B	D	B	D(V)	B(V)
0	0	0	0		
0	1	1	1		
1	0	1	0		
1	1	0	0		

Full Adder						
A	B	Cn-1	S	C	S(V)	C(V)
0	0	0	0	0		
0	0	1	1	0		
0	1	0	1	0		
0	1	1	0	1		
1	0	0	1	0		
1	0	1	0	1		
1	1	0	0	1		
1	1	1	1	1		

Full Subtractor						
A	B	Cn-1	D	B	D(v)	B(v)
0	0	0	0	0		
0	0	1	1	1		
0	1	0	1	1		
0	1	1	0	1		
1	0	0	1	0		
1	0	1	0	0		
1	1	0	0	0		
1	1	1	1	1		

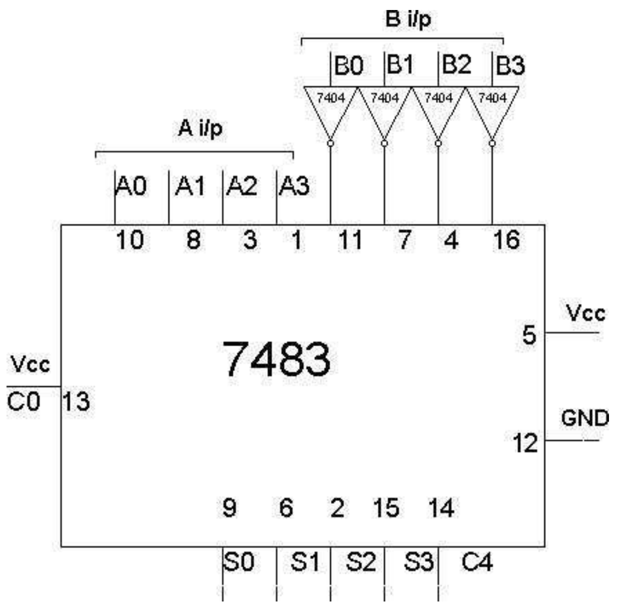
Adder : -



Truth Table: -

A3	A2	A1	A0	B3	B2	B1	B0	C4 (V)	S3(V)	S2(V)	S1(V)	S0(V)
0	0	0	1	0	0	1	0	0	0	0	1	1
0	1	0	1	1	0	1	1	1	1	0	0	0
1	0	1	0	1	0	1	0	1	0	1	0	0
1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	0	0	1	1	0	1	0	1	0

Subtractor:-



Experiment No: Date:___/__/

PARALLEL ADDER AND SUBTRACTOR USING 7483

Aim: - To realize IC7483 as parallel adder / Subtractor.

Apparatus Required: -

IC 7483, IC 7404, etc.

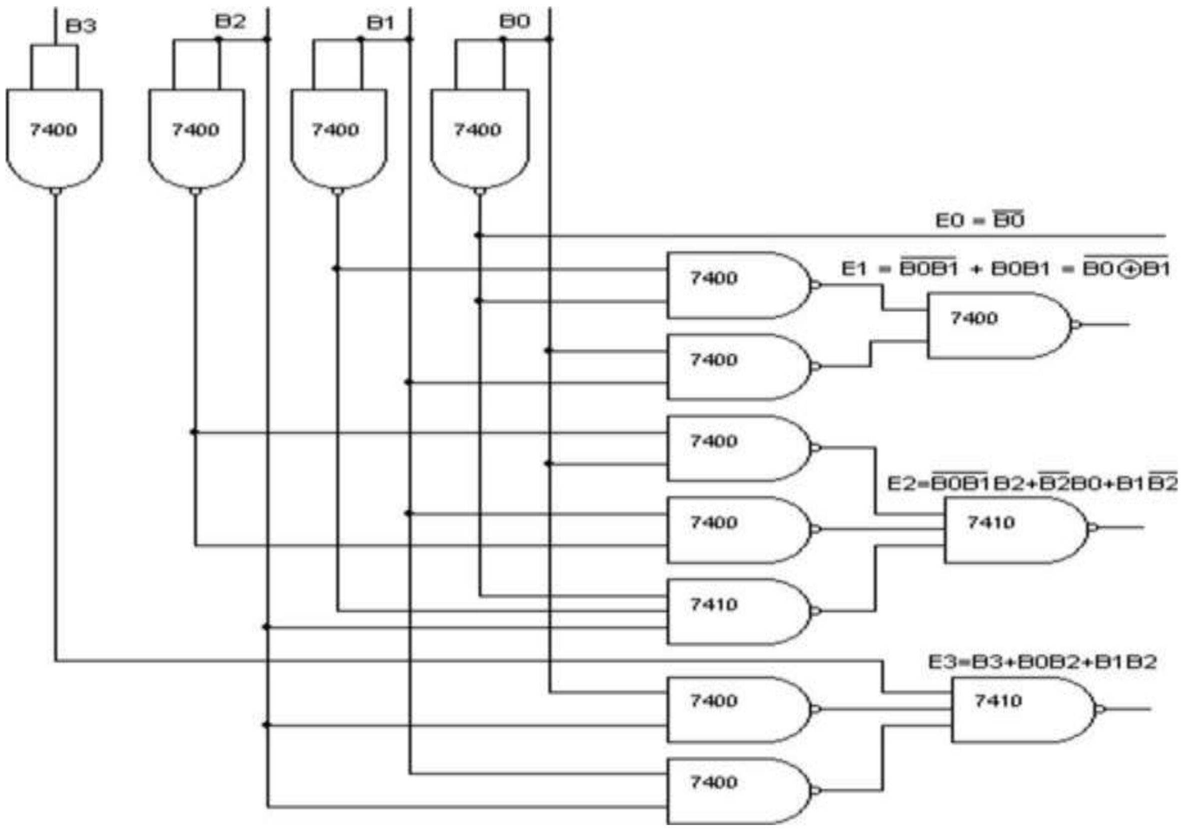
Procedure: -

1. **Apply the inputs to A0 to A3 and B0 to B3.**
2. **Connect C0 to the Ground.**
3. **Check the output sum on the S0 to S3 and also C4.**
4. **For subtraction connect C0 to Vcc, Apply the B input through NOT gate, which gives the complement of B.**
5. **The truth table of adder and Subtractor are noted down.**

Truth Table for Subtractor

A3	A2	A1	A0	B3	B2	B1	B0	C4(V)	S3(V)	S2(V)	S1(V)	S0(V)
0	0	1	0	0	0	0	1	1	0	0	0	1
0	1	0	1	0	0	1	1	1	0	0	1	0
0	0	1	1	0	1	0	1	0	1	1	1	0
1	0	1	0	0	1	1	0	1	0	1	0	0
1	0	0	0	1	1	1	1	0	1	0	0	1
1	0	1	0	0	1	1	0	1	0	1	0	0
1	0	0	0	1	1	1	1	0	1	0	0	1

BCD To Excess-3


B3
B2
B1
B0
7400
7410
E0 = $\overline{B0}$
E1 = $\overline{B0}\,\overline{B1}$ + B0B1 = $\overline{B0 \oplus B1}$
E2=$\overline{B0}\,\overline{B1}$B2+$\overline{B2}$B0+B1$\overline{B2}$
E3=B3+B0B2+B1B2

Truth Table For Code Conversion: -

Inputs				Outputs			
B3	B2	B1	B0	E3 (v)	E2 (v)	E1 (v)	E0 (v)
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

Experiment No: Date:___/__/_____

BCD to Excess 3 AND Excess 3 to BCD

Aim: - To verify BCD to excess –3 code conversion using NAND gates. To study and verify the truth table of excess-3 to BCD code converter

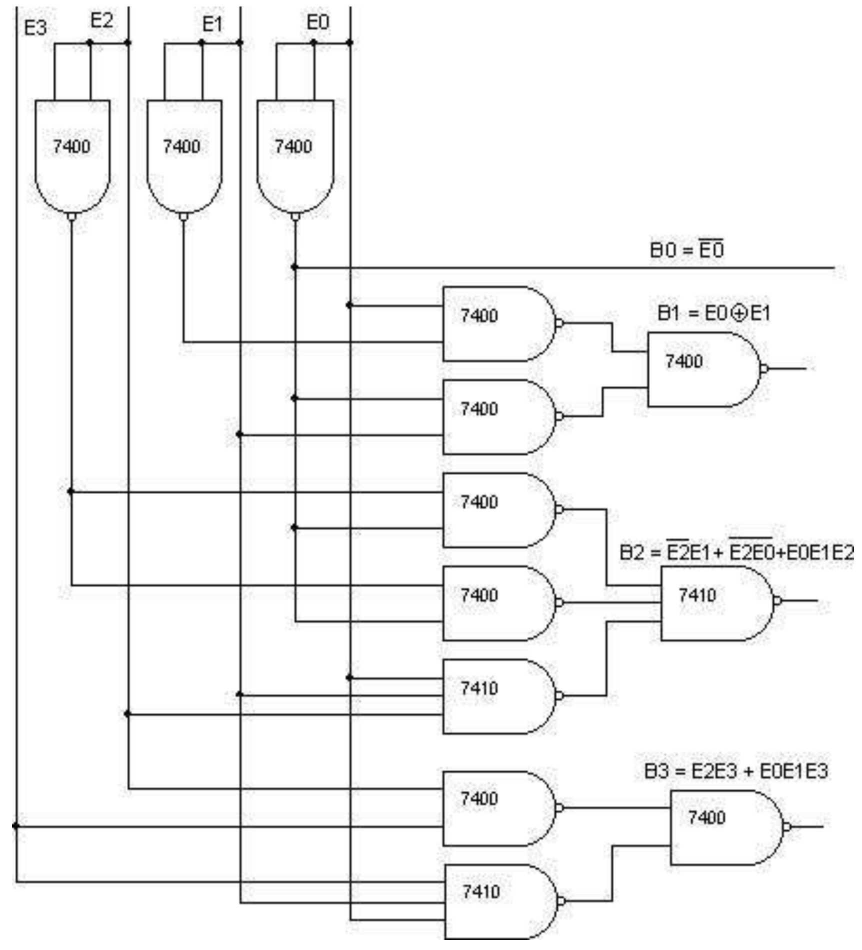
Apparatus Required: -

IC 7400, IC 7404, etc.

Procedure: - (BCD Excess 3 and Vice Versa)

1. **Make the connections as shown in the fig.**
2. **Pin [14] of all IC'S are connected to +5V and pin [7] to the ground.**
3. **The inputs are applied at E3, E2, E1, and E0 and the corresponding outputs at B3, B2, B1, and B0 are taken for excess – 3 to BCD.**
4. **B3, B2, B1, and B0 are the inputs, and the corresponding outputs are E3, E2, E1 and E0 for BCD to excess – 3.**
5. **Repeat the same procedure for other combinations of inputs.**
6. **Truth table is written.**

Excess-3 To BCD :-



Truth Table For Code Conversion: -

Inputs				Outputs			
E3	**E2**	**E1**	**E0**	**B3 (v)**	**B2 (v)**	**B1 (v)**	**B0(v)**
0	**0**	**1**	**1**	**0**	**0**	**0**	**0**
0	**1**	**0**	**0**	**0**	**0**	**0**	**1**
0	**1**	**0**	**1**	**0**	**0**	**1**	**0**
0	**1**	**1**	**0**	**0**	**0**	**1**	**1**
0	**1**	**1**	**1**	**0**	**1**	**0**	**0**
1	**0**	**0**	**0**	**0**	**1**	**0**	**1**
1	**0**	**0**	**1**	**0**	**1**	**1**	**0**
1	**0**	**1**	**0**	**0**	**1**	**1**	**1**
1	**0**	**1**	**1**	**1**	**0**	**0**	**0**
1	**1**	**0**	**0**	**1**	**0**	**0**	**1**

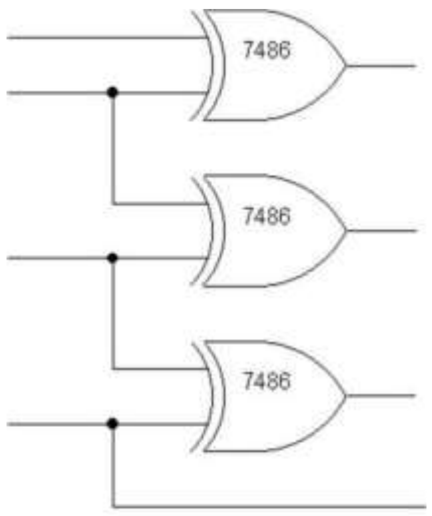
Exercise: -

1. **Obtain the expression for E3, E2, E1 and E0**

2. **Obtain the expression for B3, B2, B1 and B0**

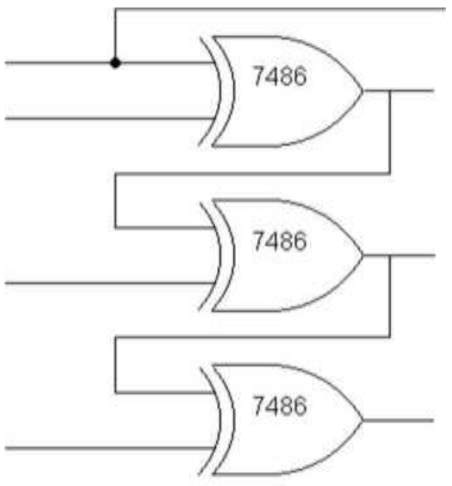
Circuit Diagram: -

Binary To Gray



Using EX-OR gates

Gray To Binary



Using EX-OR gates

Truth Table For Both: -

Inputs				Outputs			
B3	B2	B1	B0	G3 (V)	G2 (V)	G1 (V)	G0 (V)
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

Experiment No: Date:____/__/______

BINARY TO GRAY AND GRAY TO BINARY CONVERSION

Aim: - To convert given binary numbers to gray codes.

Apparatus Required: -

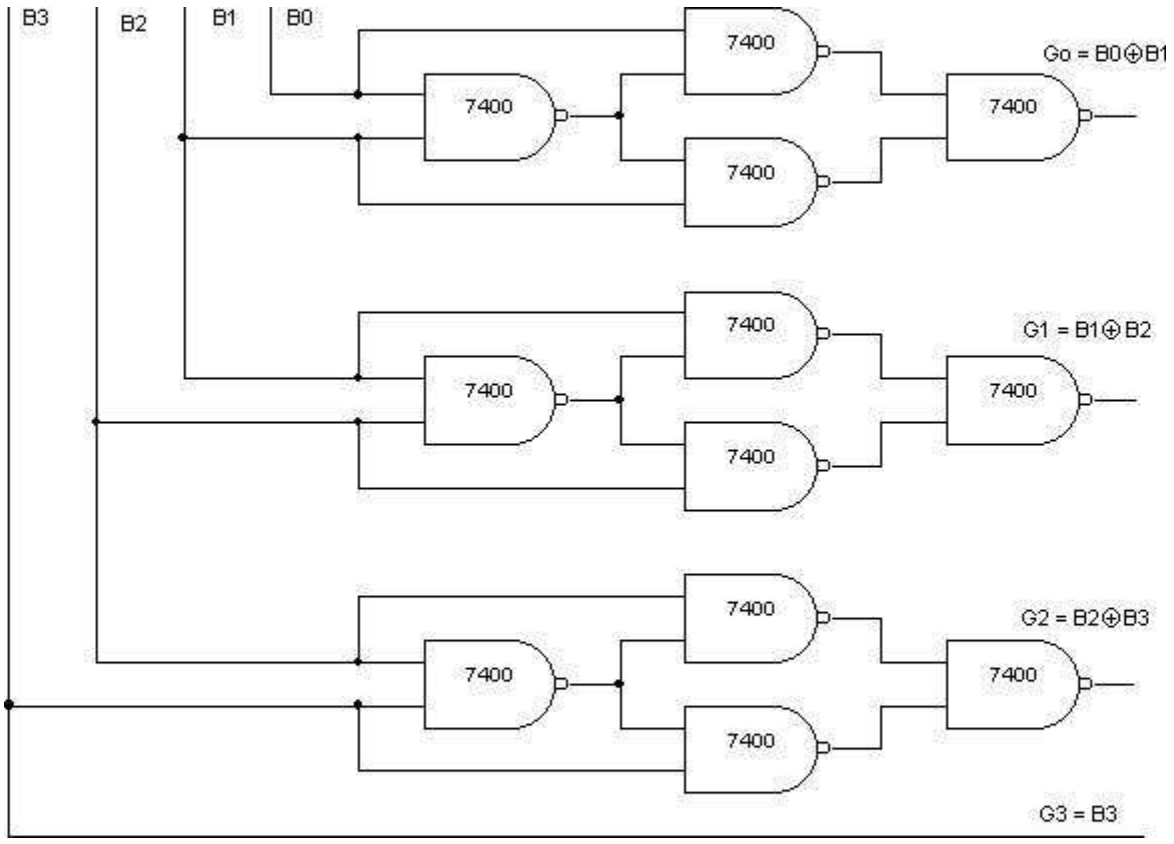
IC 7486, etc

Procedure: -

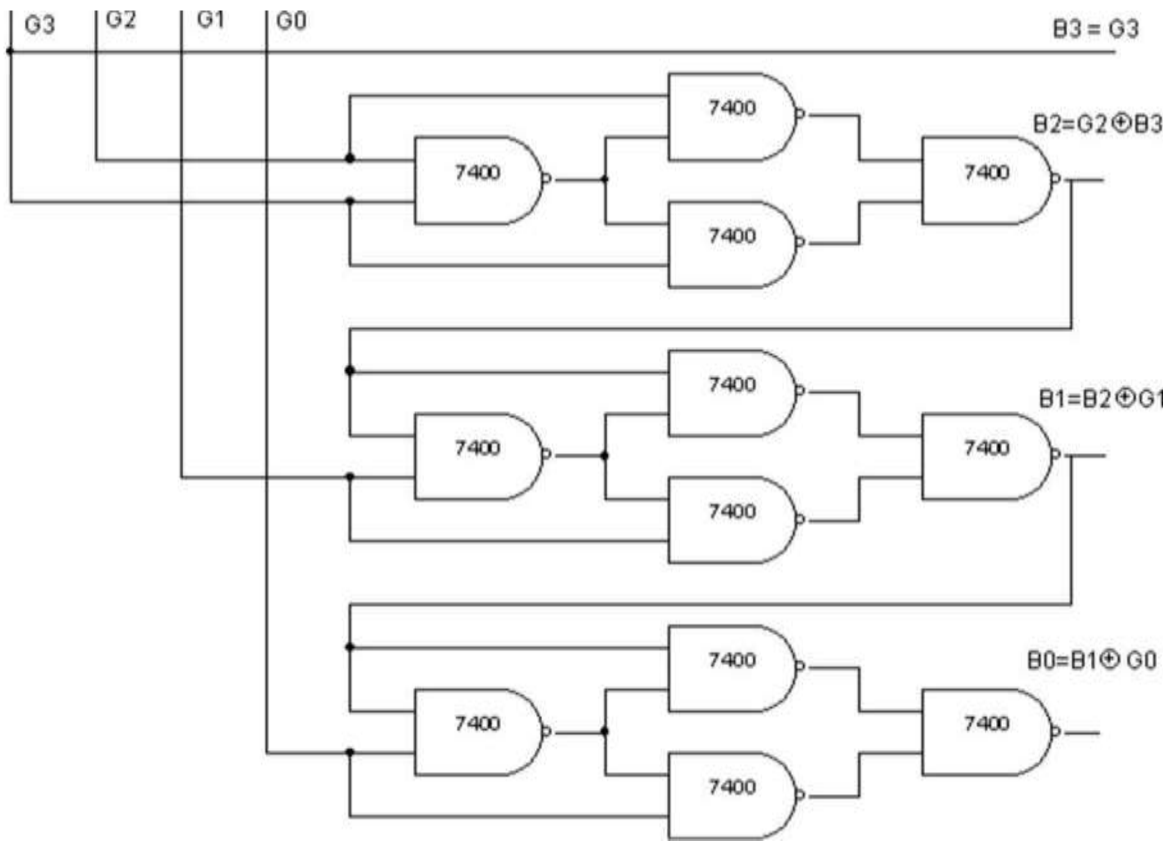
1. **The circuit connections are made as shown in fig.**
2. **Pin (14) is connected to +Vcc and Pin (7) to ground.**
3. **In the case of binary to gray conver sion, the inputs B0, B1, B2 and B3 are given at respective pins and outputs G0, G1, G2, G3 are taken for all the 16 combinations of the input.**
4. **In the case of gray to binary conver sion, the inputs G0, G1, G2 and G3 are given at respective pins and outputs B0, B1, B2, and B3 are taken for all**

   the 16 combinations of inputs.
5. **The values of the outputs are tabulated.**

Using Nand Gates
Only: - Binary To Gra



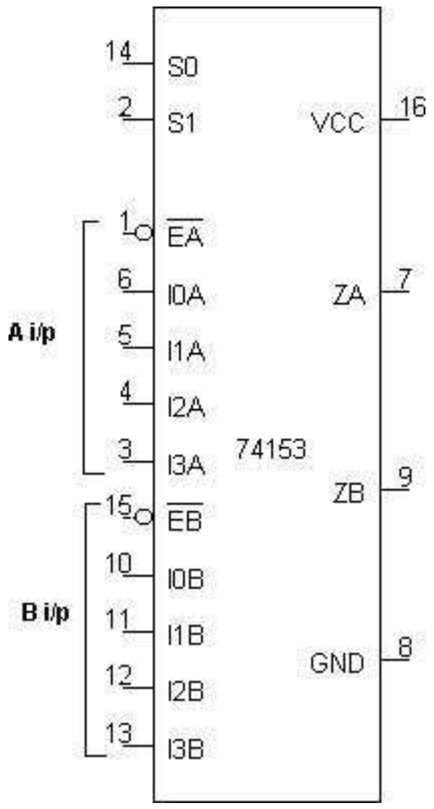
Gray Code



Truth Table For Both: -

Inputs				Outputs			
B3	**B2**	**B1**	**B0**	**G3 (V)**	**G2 (V)**	**G1 (V)**	**G0 (V)**
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

Pin Details: -



Truth Table: -

CHANNEL – A							
INPUTS					SELECT LINES		O/P
Ēa	Ioa	I1a	I2a	I3a	S1	S2	Za(v)
1	X	X	X	X	X	X	0
0	0	X	X	X	0	0	0
0	1	X	X	X	0	0	1
0	X	0	X	X	0	1	0
0	X	1	X	X	0	1	1
0	X	X	0	X	1	0	0
0	X	X	1	X	1	0	1
0	X	X	X	0	1	1	0
0	X	X	X	1	1	1	1

CHANNEL – B							
INPUTS					SELECT LINES		O/P
Ēa	Iob	I1b	I2b	I3b	S1	S2	Za(v)
1	X	X	X	X	X	X	0
0	0	X	X	X	0	0	0
0	1	X	X	X	0	0	1
0	X	0	X	X	0	1	0
0	X	1	X	X	0	1	1
0	X	X	0	X	1	0	0
0	X	X	1	X	1	0	1
0	X	X	X	0	1	1	0
0	X	X	X	1	1	1	1

Experiment No: Date: __/__/

MUX/DEMUX USING 74153 & 74139

Aim: - To verify the truth table of multiplexer using 74153 & to verify a demultiplexer using 74139. To study the arithmetic circuits half-adder half Subtractor, full adder and full Subtractor using multiplexer.

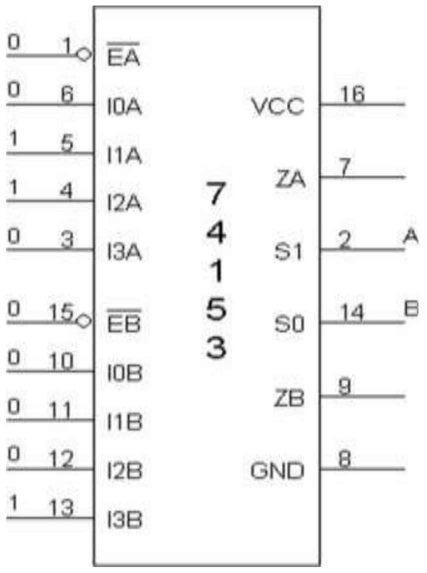
Apparatus Required: -

IC 74153, IC 74139, IC 7404, etc.

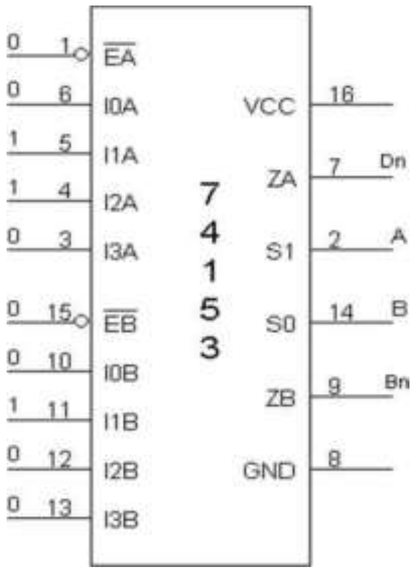
Procedure: - (IC 74153)

1. **The Pin [16] is connected to + Vcc.**
2. **Pin [8] is connected to ground.**
3. **The inputs are applied either to 'A' input or 'B' input.**
4. **If MUX 'A' has to be initialized, Ea is made low and if MUX 'B' has to be initialized, E_b is made low.**
5. **Based on the selection lines one of the inputs will be selected at the output and thus the truth table is verified.**
6. **In case of half adder using MUX, sum and carry is obtained by applying a constant inputs at I_{0a}, I_{1a}, I_{2a}, I_{3a} and I_{0b}, I_{1b}, I_{2b} and I_{3b} and the corresponding values of select lines are changed as per table and the output is taken at Z0a as sum and Z0b as carry.**
7. **In this case, the channels A and B are kept at constant inputs according to the table and the inputs A and B are varied. Making Ea and Eb zero and the output is taken at Za, and Zb.**
8. **In full adder using MUX, the input is applied at Cn-1, An and Bn. According to the table corresponding outputs are taken at Cn and Dn.**

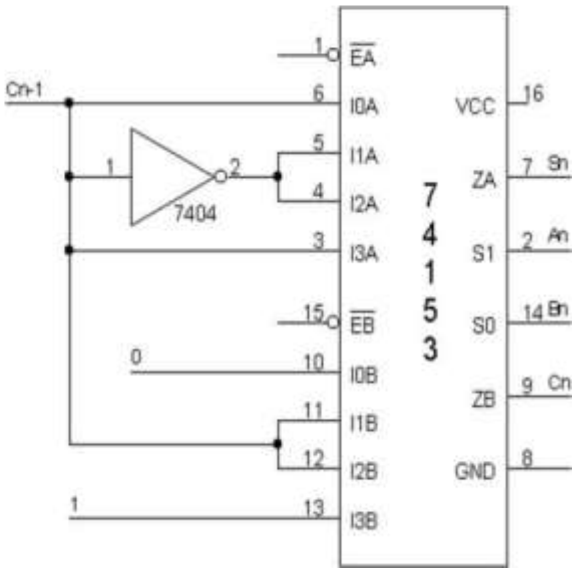
Half Adder Using 74153 –



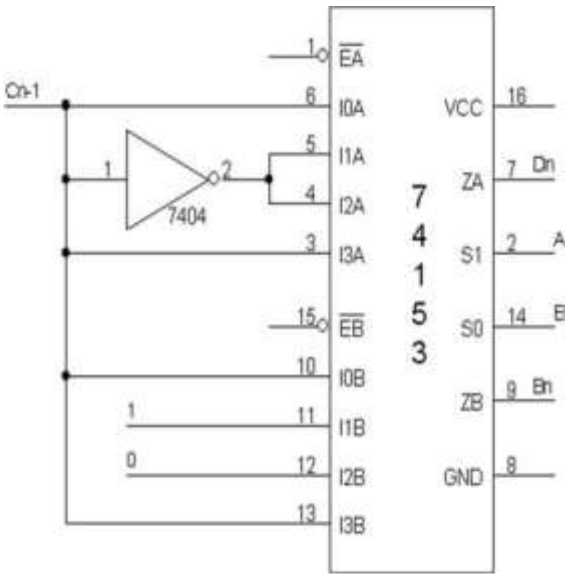
Half Subtractor: -



Full Adder Using 74153: -



Full Subtractor Using 74153: -

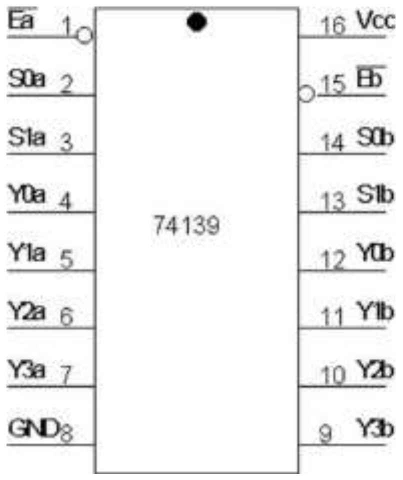


Truth Tables: - Same for both Subtractor and adder

Half adder/subtractor			
A	B	Sn/Dn (V)	Cn/Bn (V)
0	0		
0	1		
1	0		
1	1		

Full Adder/subtractro				
An	Bn	Cn-1	Sn/Dn (V)	Cn/Bn (V)
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Pin Details: -



Truth Table For Demux: -

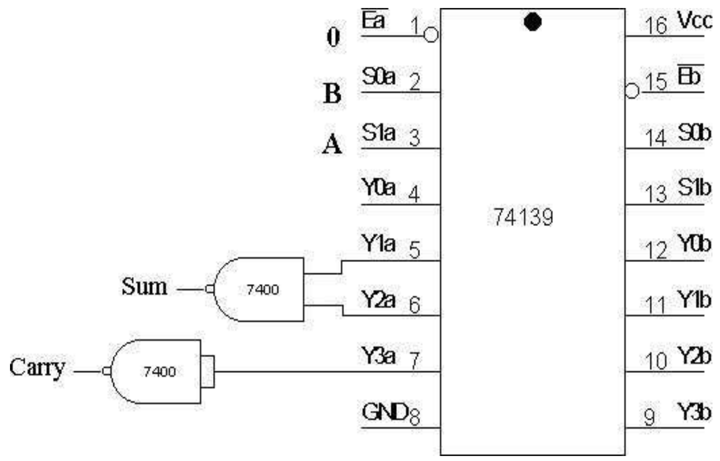
CHANNEL – A						
Inputs			Outputs			
Ēa	S1a	S0a	Y0a	Y1a	Y2a	Y3a
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

CHANNEL – B						
Inputs			Outputs			
Ēb	S1b	S0b	Y0b	Y1b	Y2b	Y3b
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

Procedure: - (IC 74139)

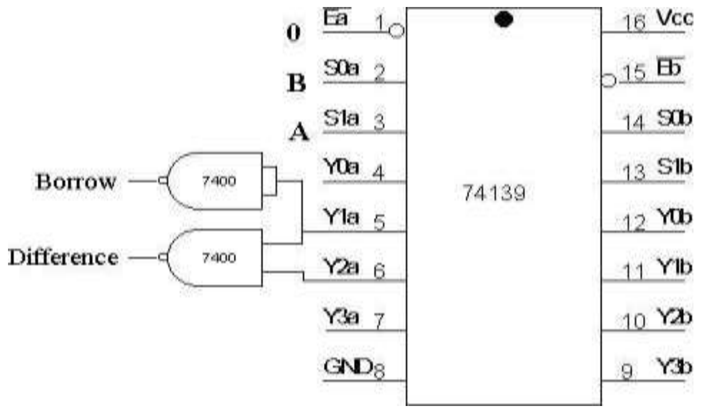
1. **The inputs are applied to either ‘a’ input or ‘b’ input**
2. **The demux is activated by making Ea low and Eb low.**
3. **The truth table is verified.**

Half adder



Half Adder			
A	B	Sn (V)	Cn (V)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Half subtractor:-

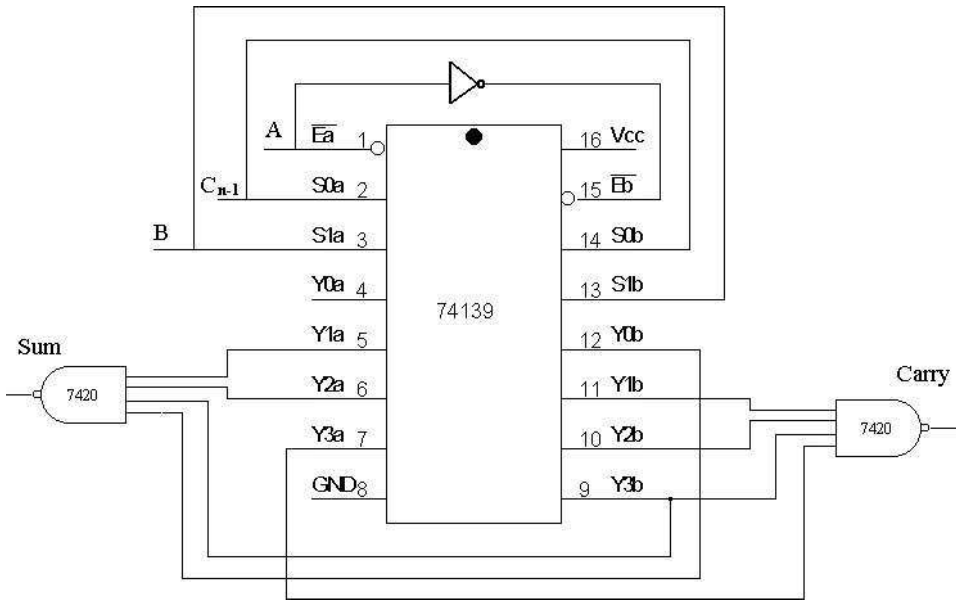


Half Subtractor			
A	B	Dn (V)	Bn (V)
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

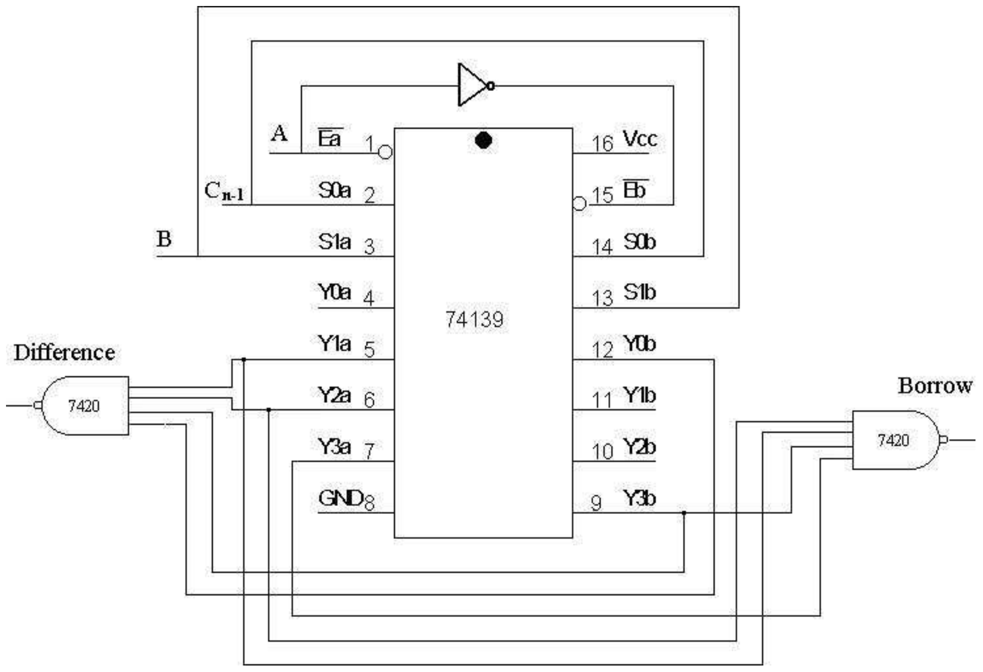
Exercise:-

- **Repeat the experiment to verify**

ChannelB. <u>Full Adder using IC 74139:-</u>



Full subtractor using IC 74139:-

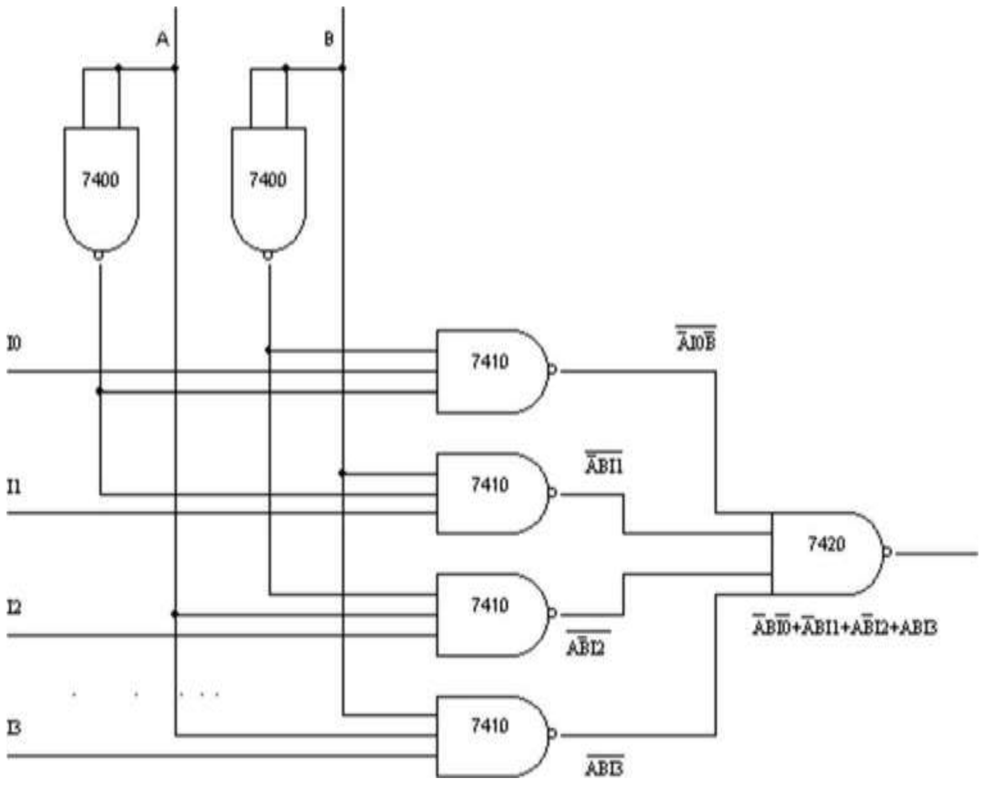


Truth Tables:-

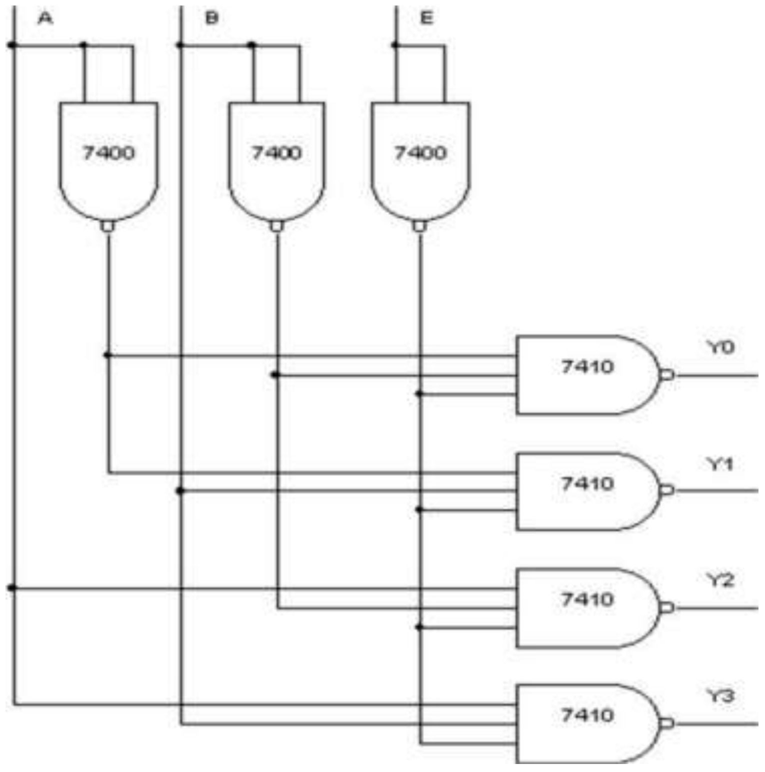
Full Adder				
An	**Bn**	**Cn-1**	**Sn (V)**	**Cn (V)**
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Full Subtractor				
An	**Bn**	**Cn-1**	**Dn (V)**	**Bn (V)**
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

MUX USING NAND GATES ONLY: -



DEMUX USING NAND GATES ONLY: -



Experiment No: DATE: __/__/

MUX AND DEMUX USING NAND GATES

AIM: - To verify the truth table of MUX and DEMUX using NAND.

APPARATUS REQUIRED: -

IC 7400, IC 7410, IC 7420, etc.

PROCEDURE: -

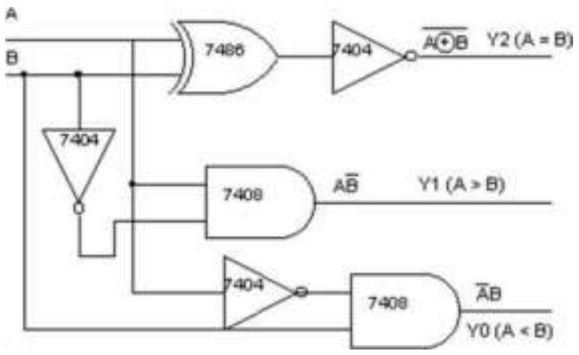
1. **Connections are made as shown in the Circuit diagram.**
2. **Change the values of the inputs as per the truth table and note down the outputs readings using multimeter.**

TRUTH TABLES: -

INPUT						OUPUT
A	B	I0	I1	I2	I3	Y (V)
0	0	0	X	X	X	0
0	0	1	X	X	X	1
0	1	X	0	X	X	0
0	1	X	1	X	X	1
1	0	X	X	0	X	0
1	0	X	X	1	X	1
1	1	X	X	X	0	0
1	1	X	X	X	1	1

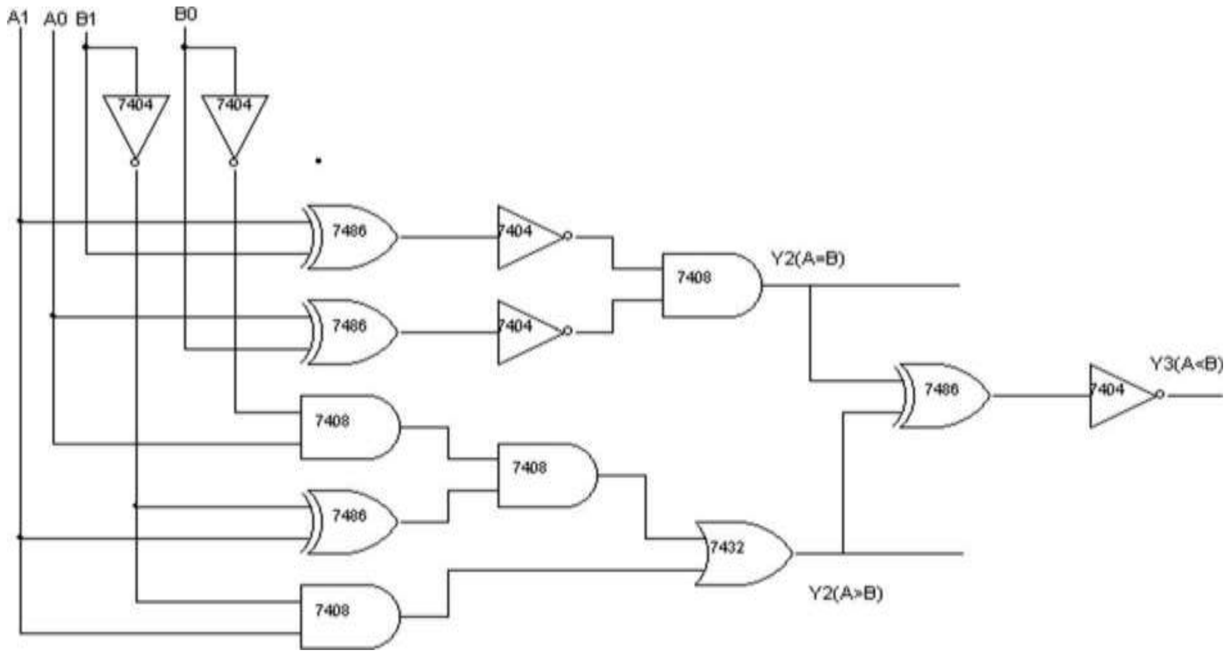
INPUT			OUPUT			
Ē	A	B	Y0 (V)	Y1 (V)	Y2 ()	Y3 (V)
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

One Bit Comparator: -



A	B	Y1 (A>B)	Y2 (A = B)	Y3 (A < B)
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

Two Bit Comparator: -



Two-Bit Comparator: -

A1	A0	B1	B0	Y1 (A > B)	Y2 (A = B)	Y3 (A < B)
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

Experiment No: Date:___/__/_____

COMPARATORS

Aim: - To verify the truth t able of one bit and two bit comparators using logic gates.

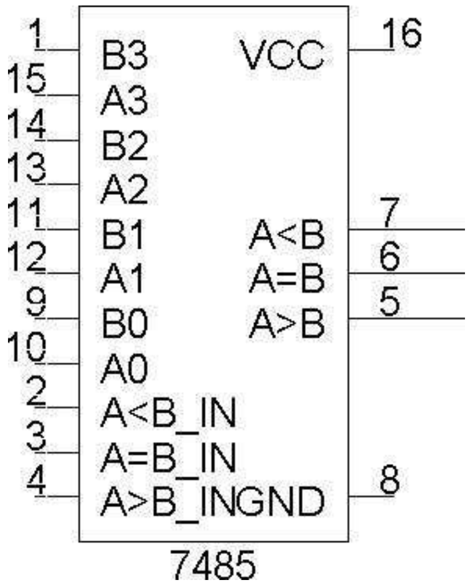
Apparatus Required: -

IC 7486, IC 7404, IC 7408, et c.

Procedure: -

1. **Verify the gates.**
2. **Make the connections as per the circuit diagram.**
3. **Switch on Vcc.**
4. **Applying i/p and Check for the outputs.**
5. **The voltameter readings ofoutputs are taken and tabulated in tabular column.**
6. **The o/p are verified.**

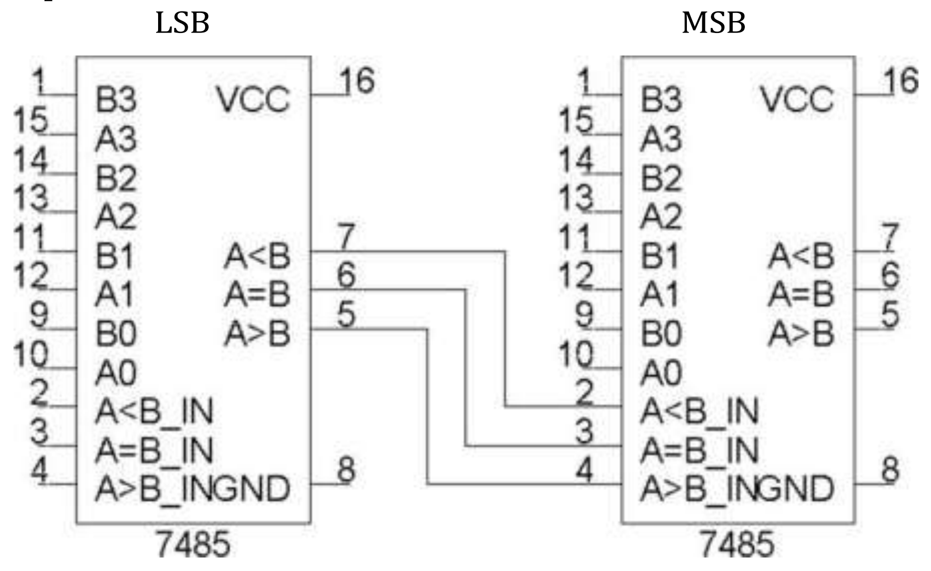
2- **bit Comparator**



Tabular Coloumn For 8-Bit Comparator: -

A_3 B_3	A_2 B_2	A_1 B_1	A_0 B_0	A>B	A=B	A<B	A>B	A=B	A<B
$A_3>B_3$	X	X	X	X	X	X			
$A_3<B_3$	X	X	X	X	X	X			
$A_3=B_3$	$A_2>B_2$	X	X	X	X	X			
$A_3=B_3$	$A_2<B_2$	X	X	X	X	X			
$A_3=B_3$	$A_2=B_2$	$A_1>B_1$	X	X	X	X			
$A_3=B_3$	$A_2=B_2$	$A_1<B_1$	X	X	X	X			
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0>B_0$	X	X	X			
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0<B_0$	X	X	X			
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	1	0	0			
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	0	1	0			
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	0	0	1			

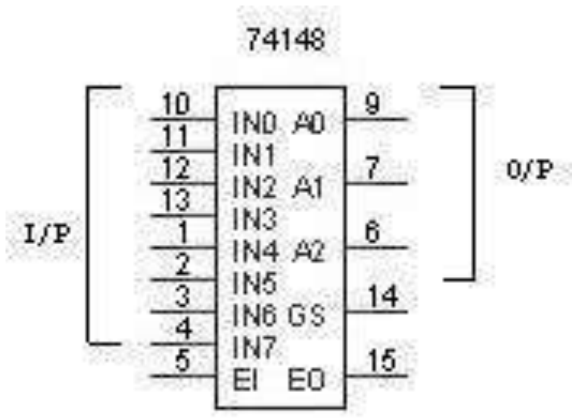
8-Bit Comparator: -



Exercise:-

- **Write the truth table for 8-bit comparator and verify the same for the above circuit.**

PIN DETAILS:-



TRUTH TABLE:-

E_n	A	B	C	D	E	F	G	H	Q_2(V)		Q_1(V)		Q_0(V)		E_S(V)		E_O(V)	
1	X	X	X	X	X	X	X	X	1		1		1		1		1	
0	0	1	1	1	1	1	1	1	1		1		1		0		1	
0	X	0	1	1	1	1	1	1	1		1		0		0		1	
0	0	X	0	1	1	1	1	1	1		0		1		0		1	
0	0	0	X	0	1	1	1	1	1		0		0		0		1	
0	0	0	0	X	0	1	1	1	0		1		1		0		1	
0	0	0	0	0	X	0	1	1	0		1		0		0		1	
0	0	0	0	0	0	X	0	1	0		0		1		0		1	
0	0	0	0	0	0	0	0	0	0		0		0		0		1	
0	1	1	1	1	1	1	1	1	1		1		1		1		0	

Experiment No: DATE:____/__/_____

ENCODER & DECODER

AIM:-To convert a given octal input to the binary output and to study the LED display using 7447 7-segment decoder/ driver.

APPARATUS REQUIRED: -

IC 74148, IC 7447, 7-segment display, etc.

PROCEDURE: - (Encoder)

1. **Connections are made as per circuit diagram.**
2. **The octal inputs are given at the corresponding pins.**
3. **The outputs are verified at the corresponding output pins.**

PROCEDURE: - (Decoder)

1. **Connections are made as per the circuit diagram.**
2. **Connect the pins of IC 7447 to the respective pins of the LED display board.**
3. **Give different combinations of the inputs and observe the decimal numbers displayed on the board.**

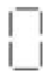
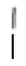
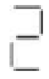
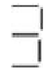
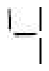
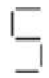
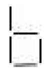
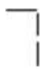
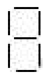
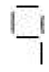
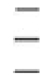
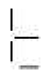
RESULT: -

The given octal numbers are converted into binary

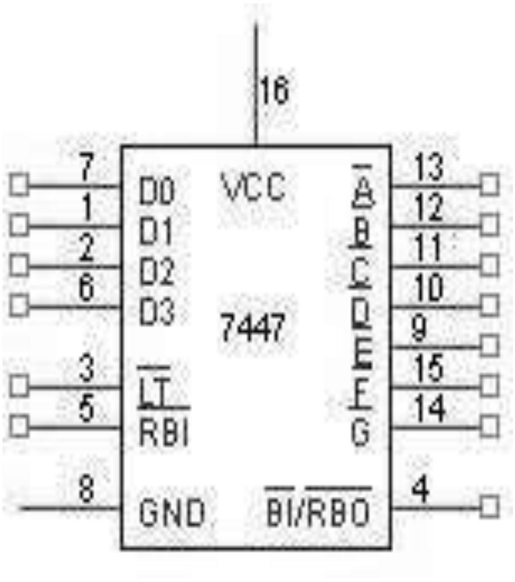
numbers. The given data is displayed using &-segment

LED decoder.

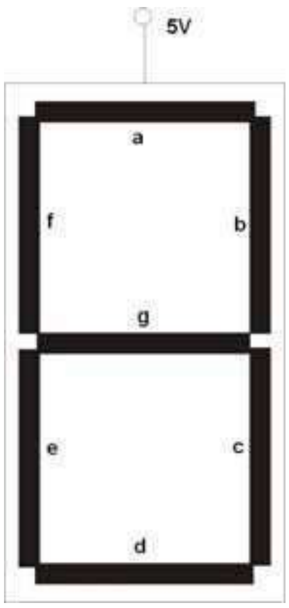
TABULAR COLUMN:-

Q4	Q3	Q2	Q1	O/P	Display	Glowing LEDs
0	0	0	0	0		a,b,c,d,e,f
0	0	0	1	1		b,c
0	0	1	0	2		a,b,d,e,g
0	0	1	1	3		a,b,c,d,g
0	1	0	0	4		b,c,f,g
0	1	0	1	5		a,c,d,f,g
0	1	1	0	6		a.c.d.e.f.g
0	1	1	1	7		a.b.c
1	0	0	0	8		a,b,c,d,e,f,g
1	0	0	1	9		a,b,c,d,f,g
1	0	1	0	10		d,e,g
1	0	1	1	11		c,d,g
1	1	0	0	12		c,d,e
1	1	0	1	13		a,g,d
1	1	1	0	14		d,e,f,g
1	1	1	1	15		blank

PIN DETAILS:-

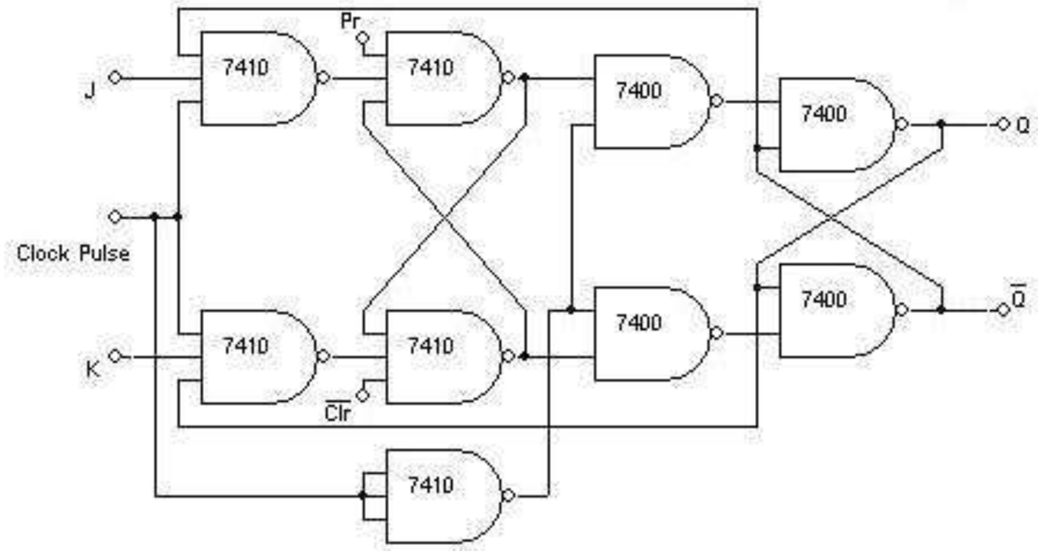


DISPLAY:-

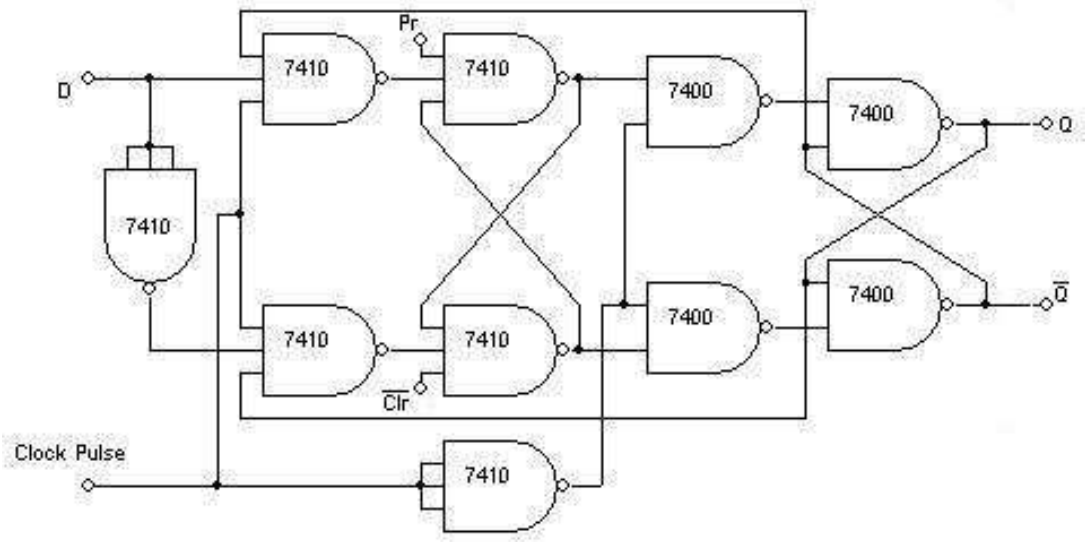


Conclusion:-

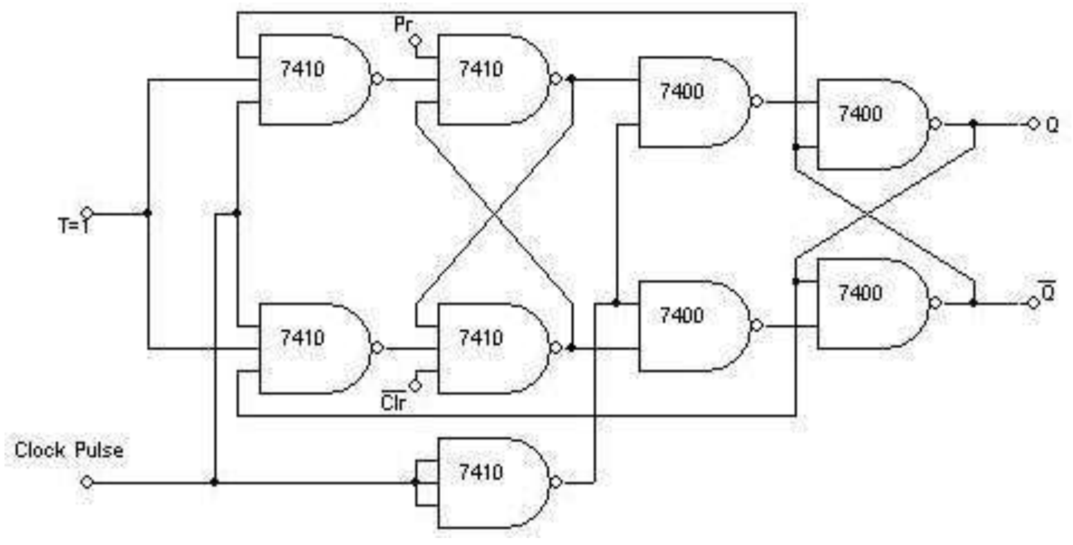
Circuit Diagram: - (Mas ter Slave JK Flip-Flop)



D Flip-Flop:-



T-Flip Flop


Pr
7410
7410
7410
7410
7400
7400
7400
7400
T=1
Q
$\overline{Q}$
$\overline{Clr}$
Clock Pulse
7410

Experiment No: Date:___/__/

FLIP-FLOP

Aim:- Truth table verification of Flip-Flops : (i) JK Master Slave
(ii) D- Type
(iii) T- Type.

Apparatus Required: -
IC 7410, IC 7400, etc.

Procedure: -

1. **Connections are made as per circuit diagram.**
2. **The truth table is verified for various combinations of inputs.**

Truth Table:- (Master Slave JK Flip-Flop)

Preset	Clear	J	K	Clock	Qn+1	$\overline{\text{Qn}\square 1}$	
0	1	X	X	X	1	0	Set
1	0	X	X	X	0	1	Reset
1	1	0	0	⎍	Qn	$\overline{\text{Qn}}$	No Change
1	1	0	1	⎍	0	1	Reset
1	1	1	0	⎍	1	0	Set
1	1	1	1	⎍	$\overline{\text{Qn}}$	Qn	Toggle

D Flip-Flop:-

Preset	Clear	D	Clock	Qn+1	$\overline{\text{Qn}\square 1}$
1	1	0	⎍	0	1
1	1	1	⎍	1	0

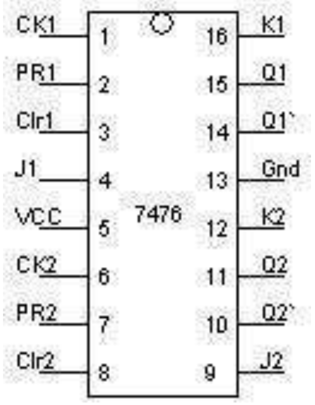
T Flip-Flop:-

Preset	Clear	T	Clock	Qn+1	$\overline{\text{Qn}\square 1}$
1	1	0	⎍	Qn	$\overline{\text{Qn}}$
1	1	1	⎍	$\overline{\text{Qn}}$	Qn

Exercise:-

- **Write the timing diagrams for all the above Flip-Flops**

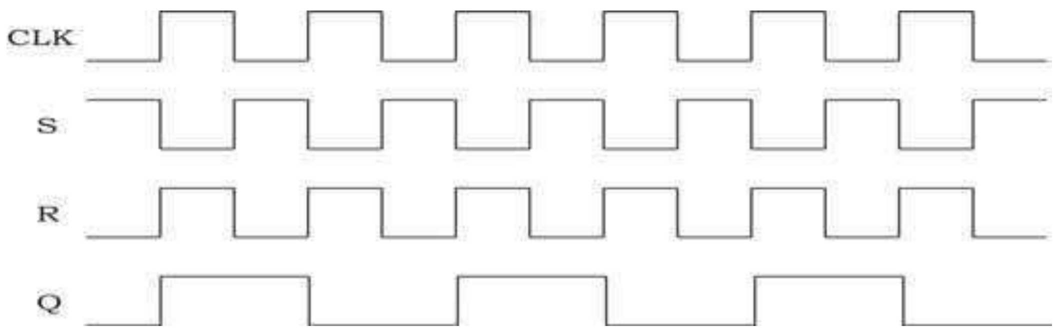
Pin Details: -



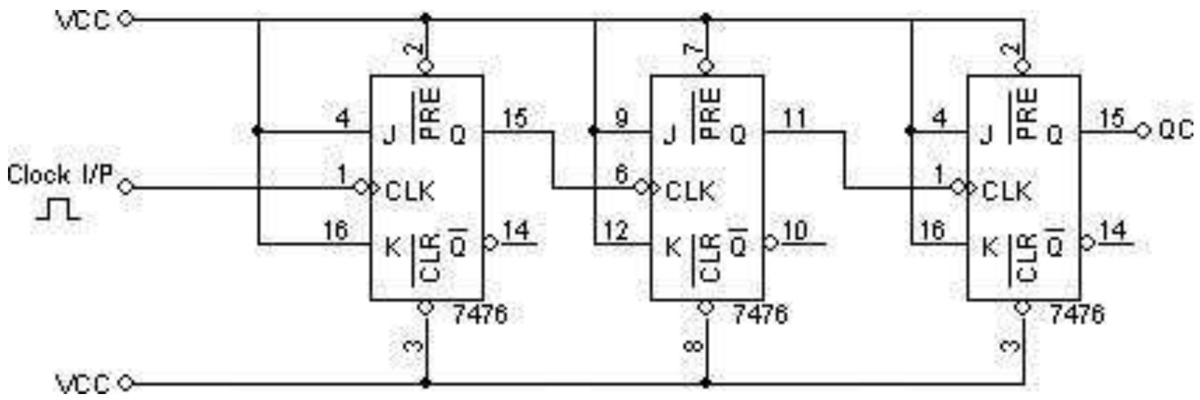
Truth Table:-

Clock	QC	QB	QA
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

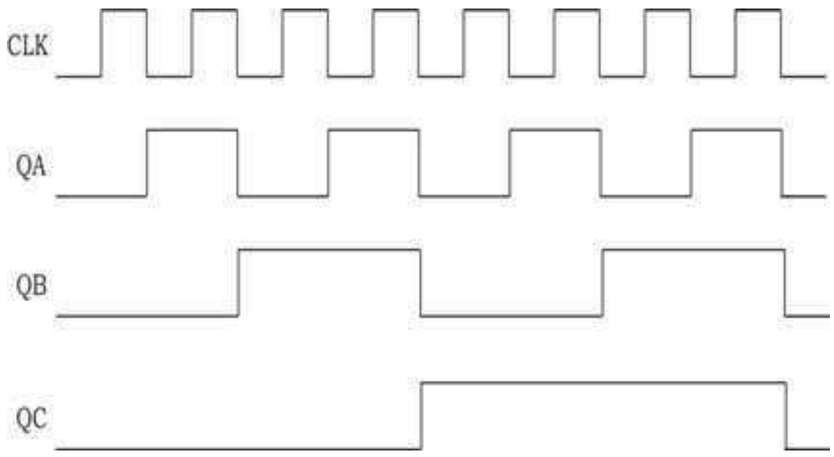
Timing Diagram:-



Circuit Diagram: - 3-Bit Asynchronous Up Counter



3-bit Asynchronous up counter			
Clock	**QC**	**QB**	**QA**
0	**0**	**0**	**0**
1	**0**	**0**	**1**
2	**0**	**1**	**0**
3	**0**	**1**	**1**
4	**1**	**0**	**0**
5	**1**	**0**	**1**
6	**1**	**1**	**0**
7	**1**	**1**	**1**
8	**0**	**0**	**0**



Experiment No: Date:___/__/

COUNTERS

Aim:- Realization of 3-bit counters as a sequential circuit and Mod-N counter design (7476, 7490, 74192, 74193).

Apparatus Required: -

IC 7408, IC 7476, IC 7 490, IC 74192, IC 74193, IC 7400, IC 7416, IC 7432 etc.

Procedure: -

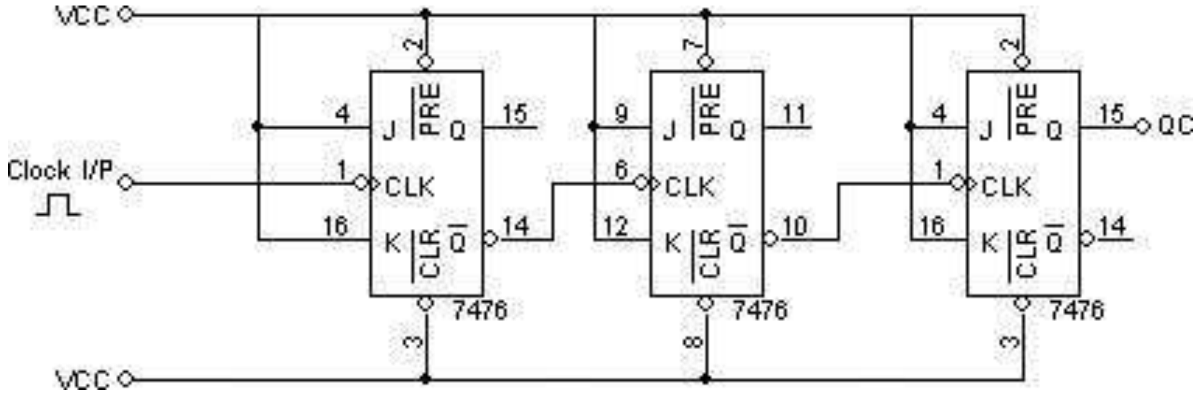
1. **Connections are made as per circuit diagram.**
2. **Clock pulses are applied one by one at the clock I/P and the O/P is observed at QA, QB & QC for IC 7476.**
3. **Truth table is verified.**

Procedure (IC 741 92, IC 74193):-

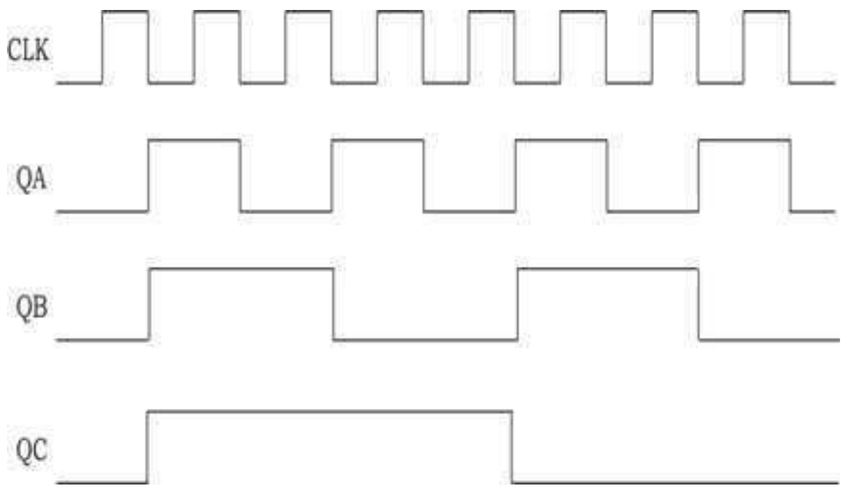
1. **Connections are made as per the circuit diagram except the connection from output of NAND gate to the load input.**
2. **The data (0011) = 3 is made available at the da ta i/ps A, B, C & D respectively.**
3. **The load pin made low so that the data 0011 appears at QD, QC, QB & QA respectively.**
4. **Now connect the output of the NAND gate to the load input.**
5. **Clock pulses are applied to “count up” pin and the truth table is verified.**
6. **Now apply (1100) = 12 for 12 to 5 count er and remaining is same as for 3 to 8 counter.**

7. **The pin diagram of IC 7419 2 is same as that of 74193. 74192 can be configured to count between 0 and 9 in either direction. The starting value can be any number between 0 and 9.**

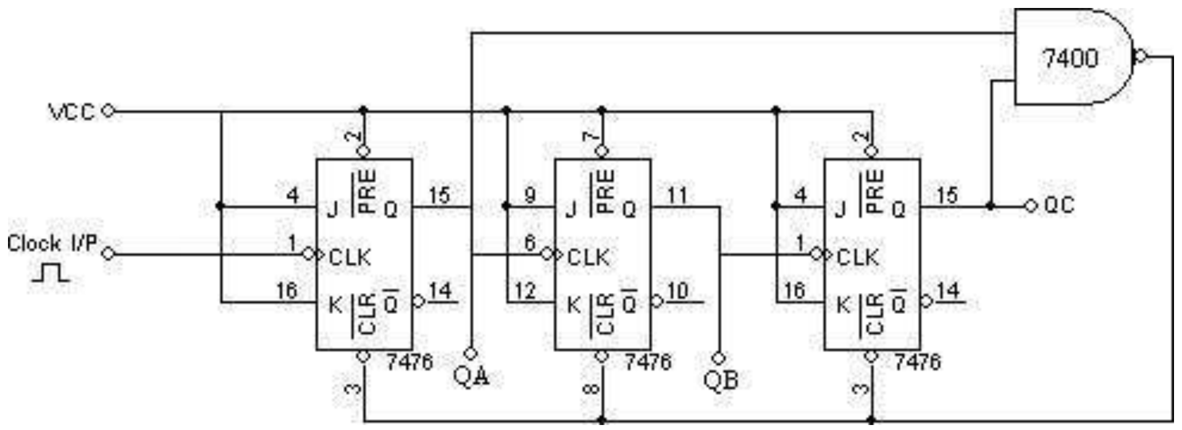
Circuit Diagram: - 3-Bit Asynchronous Down Counter



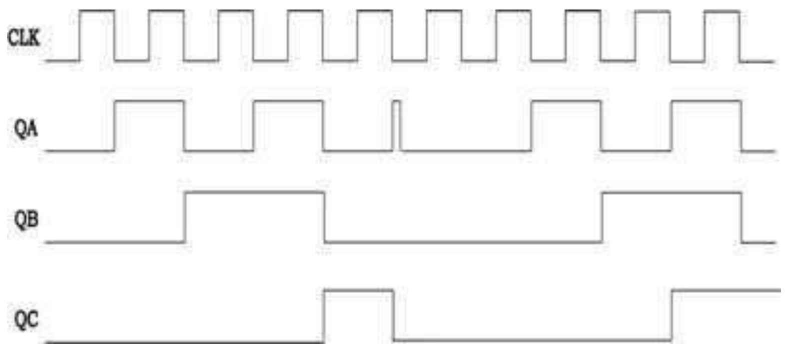
3-bit Asynchronous down counter			
Clock	**QC**	**QB**	**QA**
0	1	1	1
1	1	1	0
2	1	0	1
3	1	0	0
4	0	1	1
5	0	1	0
6	0	0	1
7	0	0	0
8	1	1	1
9	1	1	0



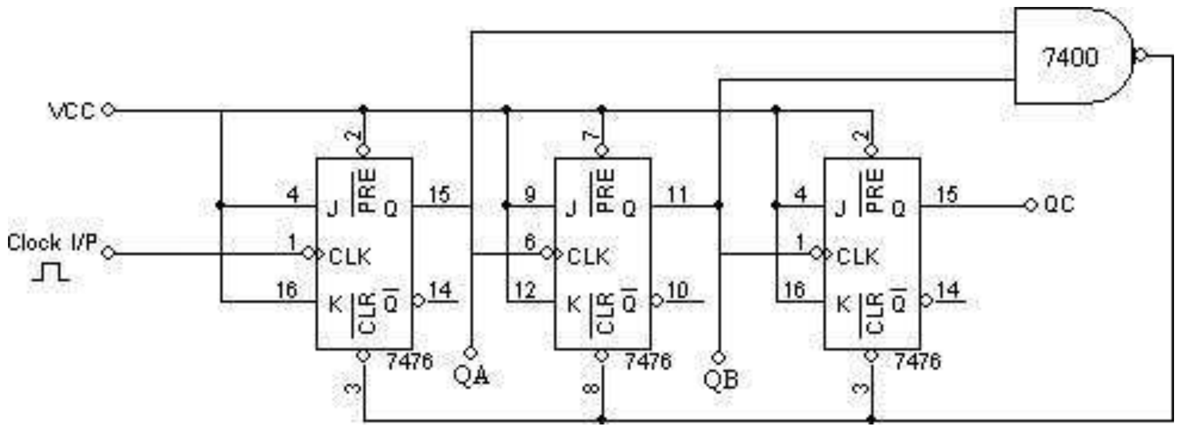
Mod 5 Asynchronous Counter: -



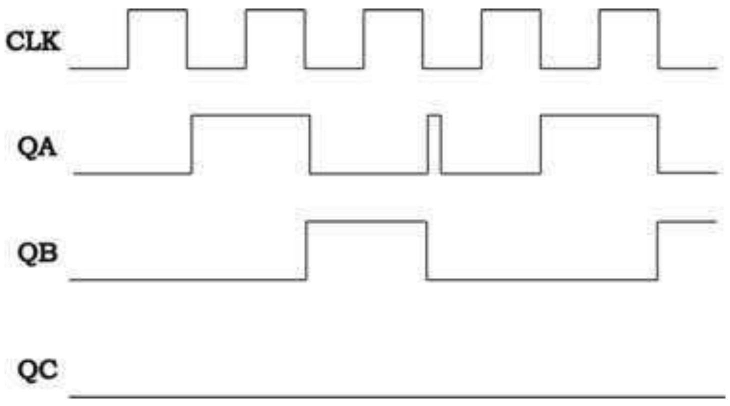
MOD 5 Asynchronous counter			
Clock	**QC**	**QB**	**QA**
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	0	0	0



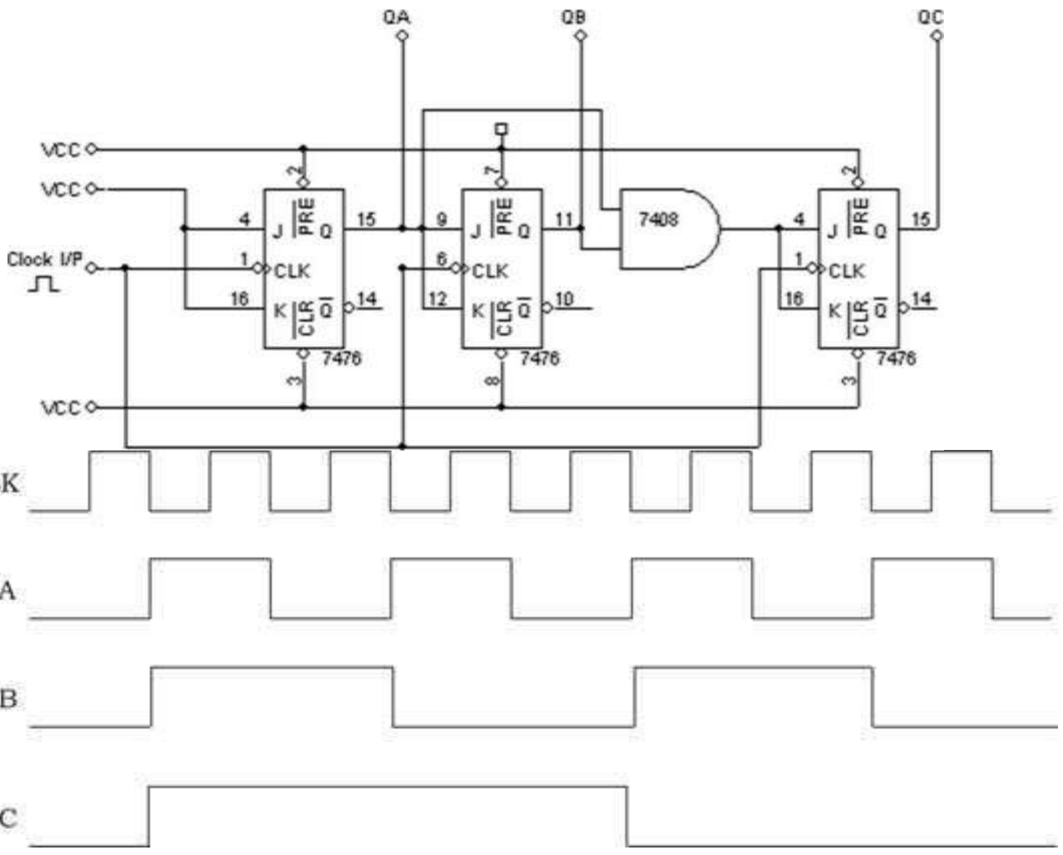
MOD 3 Asynchronous Counter:-



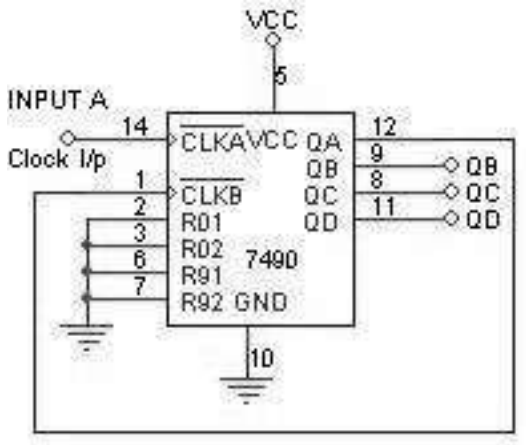
Mod 3 Asynchronous counter			
Clock	**QC**	**QB**	**QA**
0	0	0	0
1	0	0	1
2	0	1	0
3	0	0	0
4	0	0	1
5	0	1	0



3- bit Synchronous Counter:-

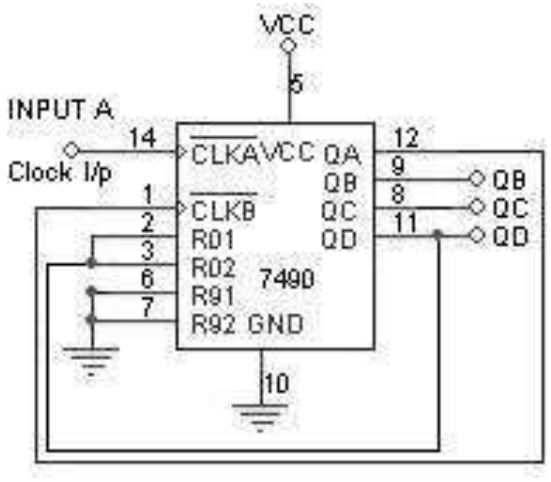


IC 7490 (Decade Counter):-



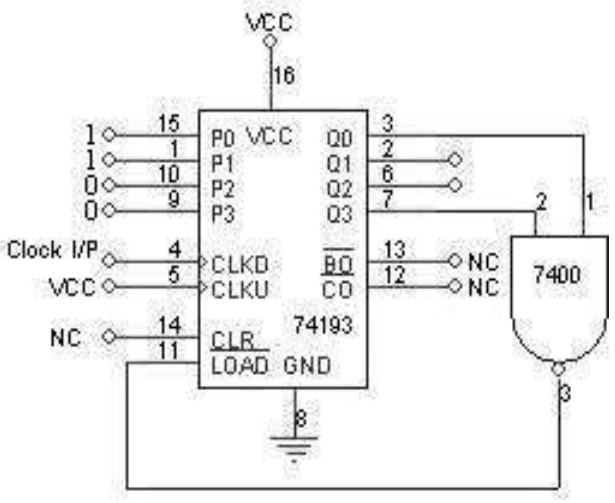
Clock	QD	QC	QB	QA
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	0	0	0	0

IC 7490 (MOD-8 Counter):-



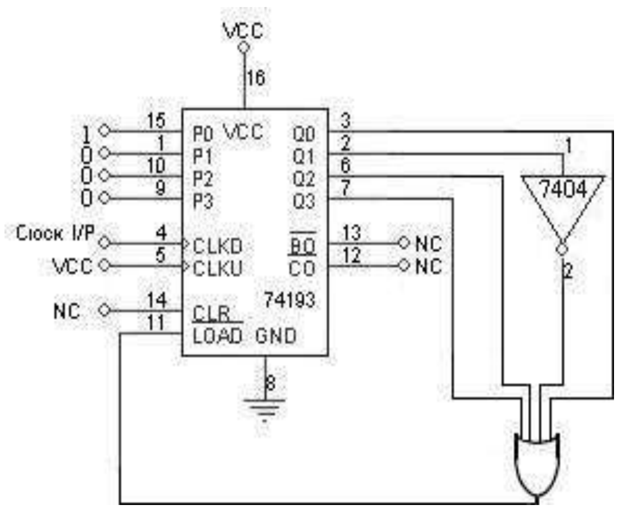
Clock	QD	QC	QB	QA
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	0	0	0	0
9	0	0	0	1

Circuit Diagram (IC 74193) To Count from 3 to 8:-



Clock	QD	QC	QB	QA	Count in Decimal
0	0	0	1	1	3
1	0	1	0	0	4
2	0	1	0	1	5
3	0	1	1	0	6
4	0	1	1	1	7
5	1	0	0	0	8
6	0	0	1	1	3
7	repeats				4

Circuit Diagram (IC 74193) To Count from 8 to 3:-

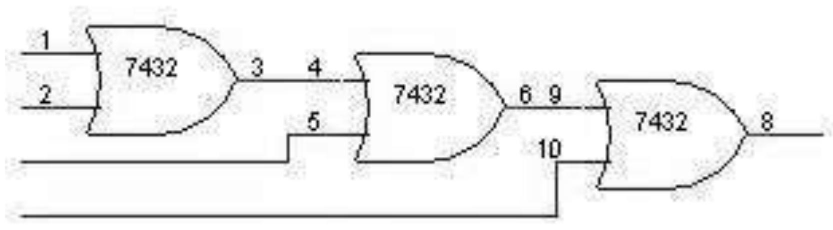


Clock	QD	QC	QB	QA	Count in Decimal
0	0	1	0	1	5
1	0	1	1	0	6
2	0	1	1	1	7
3	1	0	0	0	8
4	1	0	0	1	9
5	1	0	1	0	10
6	1	0	1	1	11
7	1	1	0	0	12
8	0	1	0	1	5
9	repeats				6

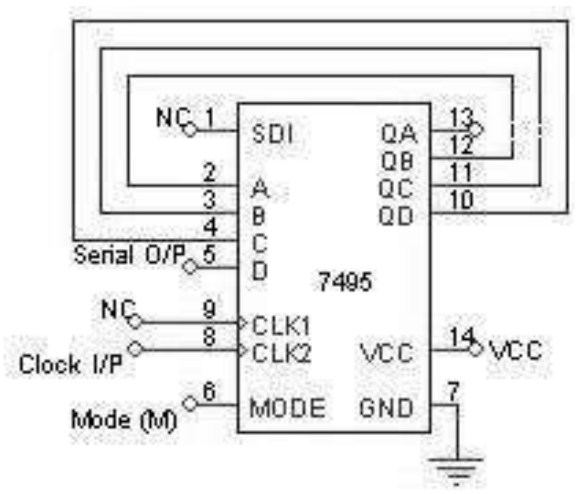
Function Table for 7490:-

Clock	R1	R2	S1	S2	QD	QC	QB	QA	
X	H	H	L	X	L	L	L	L	RESET
X	H	H	X	L	L	L	L	L	RESET
X	X	X	H	H	H	L	L	H	SET TO 9
⎍	X	L	X	L	COUNT				
⎍	L	X	L	X	COUNT				
⎍	L	X	X	L	COUNT				
⎍	X	L	L	X	COUNT				

4 I/P OR Gate can be realized as follows:-

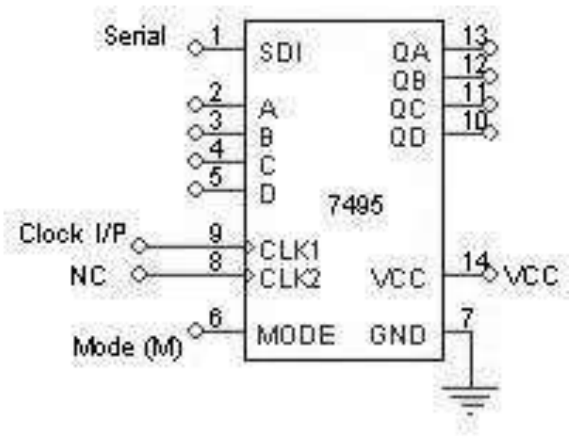


Circuit Diagram: - Shift Left



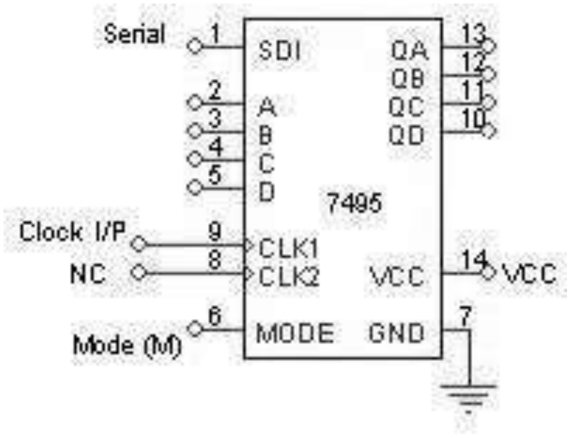
Clock	Serial i/p	QA	QB	QC	QD
1	1	X	X	X	1
2	0	X	X	1	0
3	1	X	1	0	1
4	1	1	0	1	1

SIPO (Right Shift):-



Clock	Serial i/p	QA	QB	QC	QD
1	0	0	X	X	X
2	1	1	0	X	X
3	1	1	1	0	X
4	1	1	1	1	0

SISO:-



Clock	Serial i/p	QA	QB	QC	QD
1	do=0	0	X	X	X
2	d1=1	1	0	X	X
3	d2=1	1	1	0	X
4	d3=1	1	1	1	0=do
5	X	X	1	1	1=d1
6	X	X	X	1	1=d2
7	X	X	X	X	1=d3

SHIFT REGISTERS

Aim: - Realization of 3-bit counters as a sequential circuit and Mod-N counter design (7476, 7490, 74192, 74193).

Apparatus Required: -

IC 7495, etc.

Procedure: -

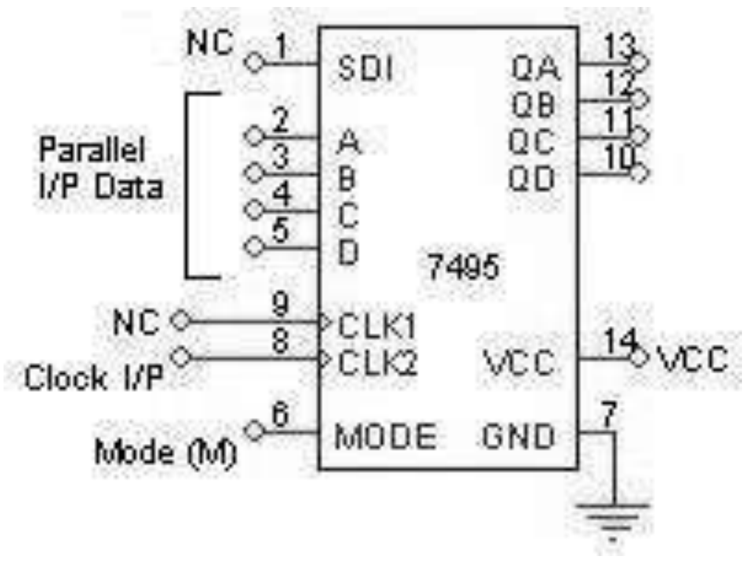
Serial In Parallel Out:-

1. **Connections are made as per circuit diagram.**
2. **Apply the data at serial i/p**
3. **Apply one clock pulse at clock 1 (Right Shift) observe this data at QA.**
4. **Apply the next data at serial i/p.**
5. **Apply one clock pulse at clock 2, observ e that the data on QA will shift to QB and the new data applied will appear at QA.**
6. **Repeat steps 2 and 3 till all the 4 bits data are entered on e by one into the shift register.**

Serial In Serial Out:-

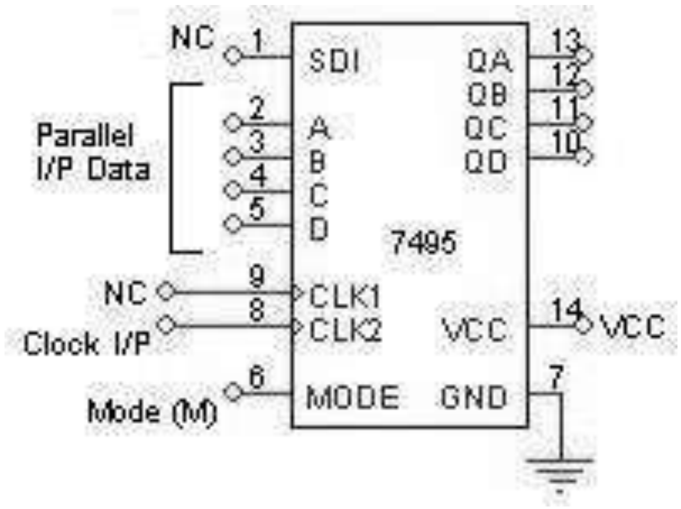
1. **Connections are made as per circuit diagram.**
2. **Load the shift register with 4 bits of data one by one serially.**
3. **At the end of 4th clock pulse the first data 'd0' appears at QD.**
4. **Apply another clock pulse; the second data 'd1' appears at QD.**
5. **Apply another clock pulse; the third data appears at QD.**
6. **Application of next clock pulse will enable the 4 th data 'd3' to appear at QD. Thus the data applied serially at the input comes out serially at QD**

PISO:-



Mode	Clock	Parallel i/p				Parallel o/p			
		A	B	C	D	QA	QB	QC	QD
1	1	1	0	1	1	1	0	1	1
0	2	X	X	X	X	X	1	0	1
0	3	X	X	X	X	X	X	1	0
0	4	X	X	X	X	X	X	X	1

PIPO:-



Clock	Parallel i/p				Parallel o/p			
	A	B	C	D	QA	QB	QC	QD
1	1	0	1	1	1	0	1	1

Parallel In Parallel Out:-

1. **Connections are made as per circuit diagram.**
2. **Apply the 4 bit data at A, B, C and D.**
3. **Apply one clock pulse at Clock 2 (Note: Mode control M=1).**
4. **The 4 bit data at A, B, C and D appears at QA, QB, QC and QD respectively.**

Parallel In Serial Out:-

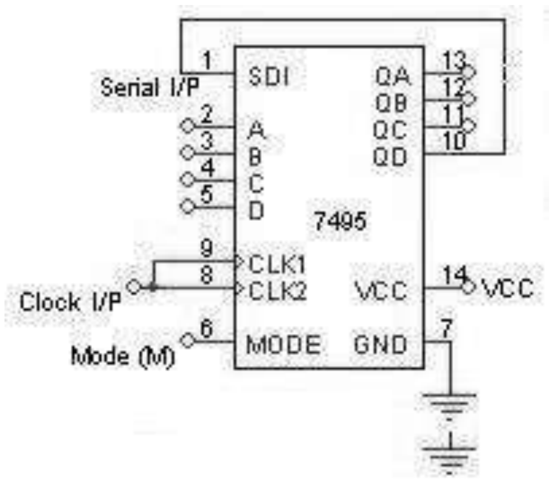
1. **Connections are made as per circuit diagram.**
2. **Apply the desired 4 bit data at A, B, C and D.**
3. **Keeping the mode control M=1 apply one clock pulse. The data applied at A, B, C and D will appear at QA, QB, QC and QD respectively.**
4. **Now mode control M=0. Apply clock pulses one by one and observe the data coming out serially at QD.**

Left Shift:-

1. **Connections are made as per circuit diagram.**
2. **Apply the first data at D and apply one clock pulse. This data appears at QD.**
3. **Now the second data is made available at D and one clock pulse applied. The data appears at QD to QC and the new data appears at QD.**
4. **Step 3 is repeated until all the 4 bits are entered one by one.**
5. **At the end 4th clock pulse, the 4 bits are available at QA, QB, QC**

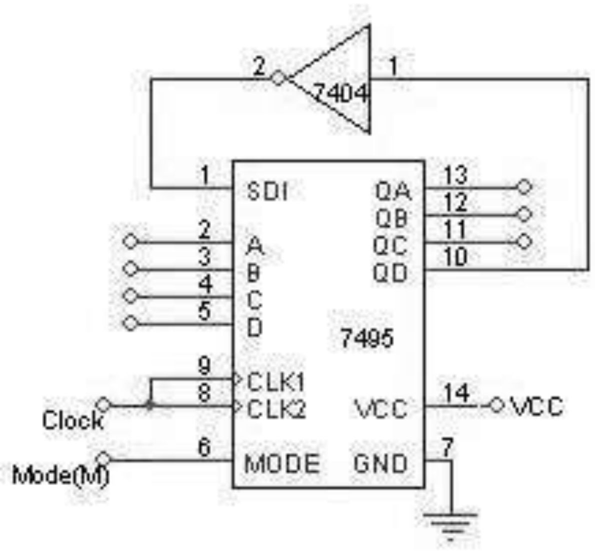
and QD. Conclusion: -

Circuit Diagram: - Ring Counter



Mode	Clock	QA	QB	QC	QD
1	1	1	0	0	0
0	2	0	1	0	0
0	3	0	0	1	0
0	4	0	0	0	1
0	5	1	0	0	0
0	6	repeats			

Johnson Counter:-



Mode	Clock	QA	QB	QC	QD
1	1	1	0	0	0
0	2	1	1	0	0
0	3	1	1	1	0
0	4	1	1	1	1
0	5	0	1	1	1
0	6	0	0	1	1
0	7	0	0	0	1
0	8	0	0	0	0
0	9	1	0	0	0
0	10	repeats			

Experiment No: Date:___/__/

JOHNSON COUNTERS / RING COUNTER

Aim:- Design and testing of Ring counter/ Johnson counter.

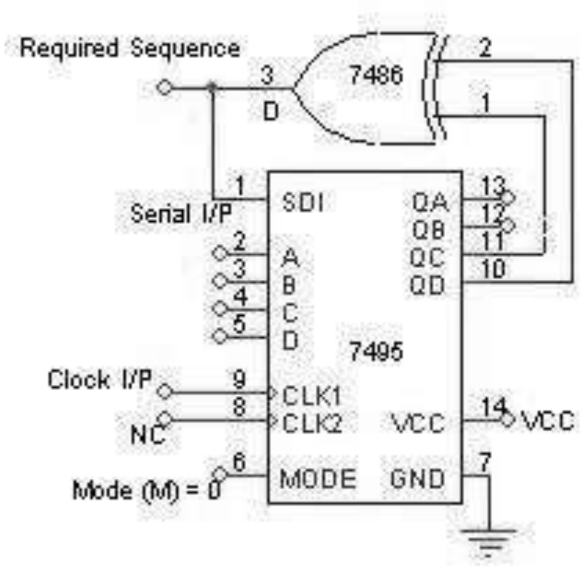
Apparatus Required: -

IC 7495, IC 7404, etc.

Procedure: -

1. **Connections are made as per the circuit diagram.**
2. **Apply the data 1000 at A, B, C and D respectively.**
3. **Keeping the mode M = 1, apply one clock pulse.**
4. **Now the mode M is made 0 and clock pulses are applied one by one, and the truth table is verified.**
5. **Above procedure is repeated for Johnson counter also.**

Circuit Diagram: - Sequence Generator



Truth Table:-

Map Value	Clock	QA	QB	QC	QD	o/p D
15	1	1	1	1	1	0
7	2	0	1	1	1	0
3	3	0	0	1	1	0
1	4	0	0	0	1	1
8	5	1	0	0	0	0
4	6	0	1	0	0	0
2	7	0	0	1	0	1
9	8	1	0	0	1	1
12	9	1	1	0	0	0
6	10	0	1	1	0	1
11	11	1	0	1	1	0
5	12	0	1	0	1	1
10	13	1	0	1	0	1
13	14	1	1	0	1	1
14	15	1	1	1	0	1

Karnaugh Map for D:-

QCQD \ QA QB	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	0	0
10	1	1	1	1

Experiment No: Date: __/__/

SEQUENCE GENERATOR

Aim:- Design of Sequence Generator.

Apparatus Required: -

IC 7495, IC 7486, etc.

Design:-

To generate a sequence of length S it is necessary to use at least N number of Flip-Flops, which satisfies the condition $S \leq 2^N - 1$.

The given sequence length S = 15.

Therefore N = 4.

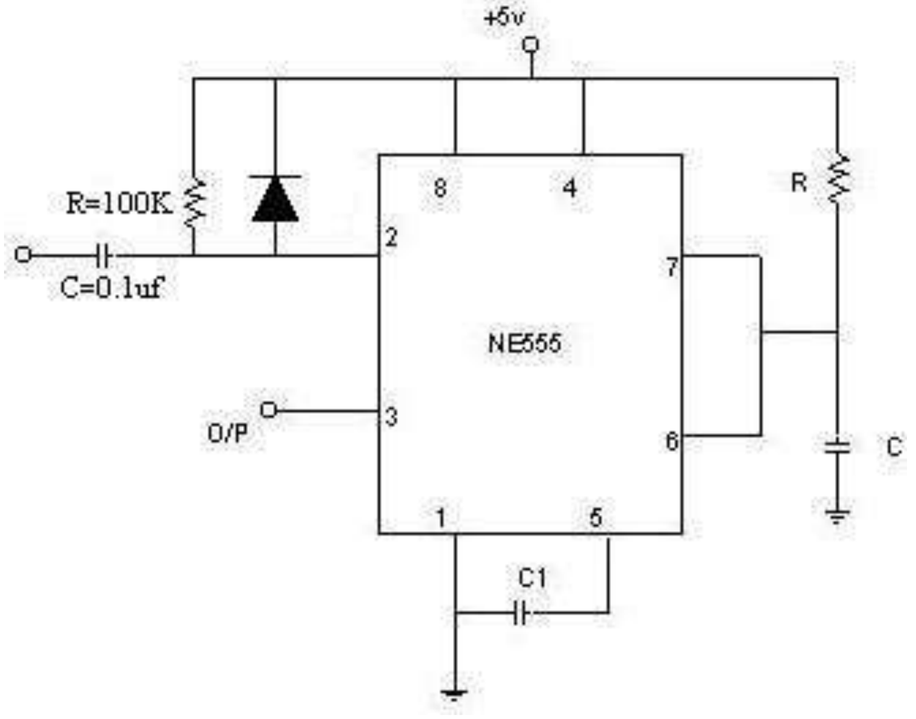
Note: - There is no guarantee that the given sequence can be generated by 4 f/fs. If the sequence is not realizable by 4 f/fs then 5 f/fs must be used and so on.

Procedure: -

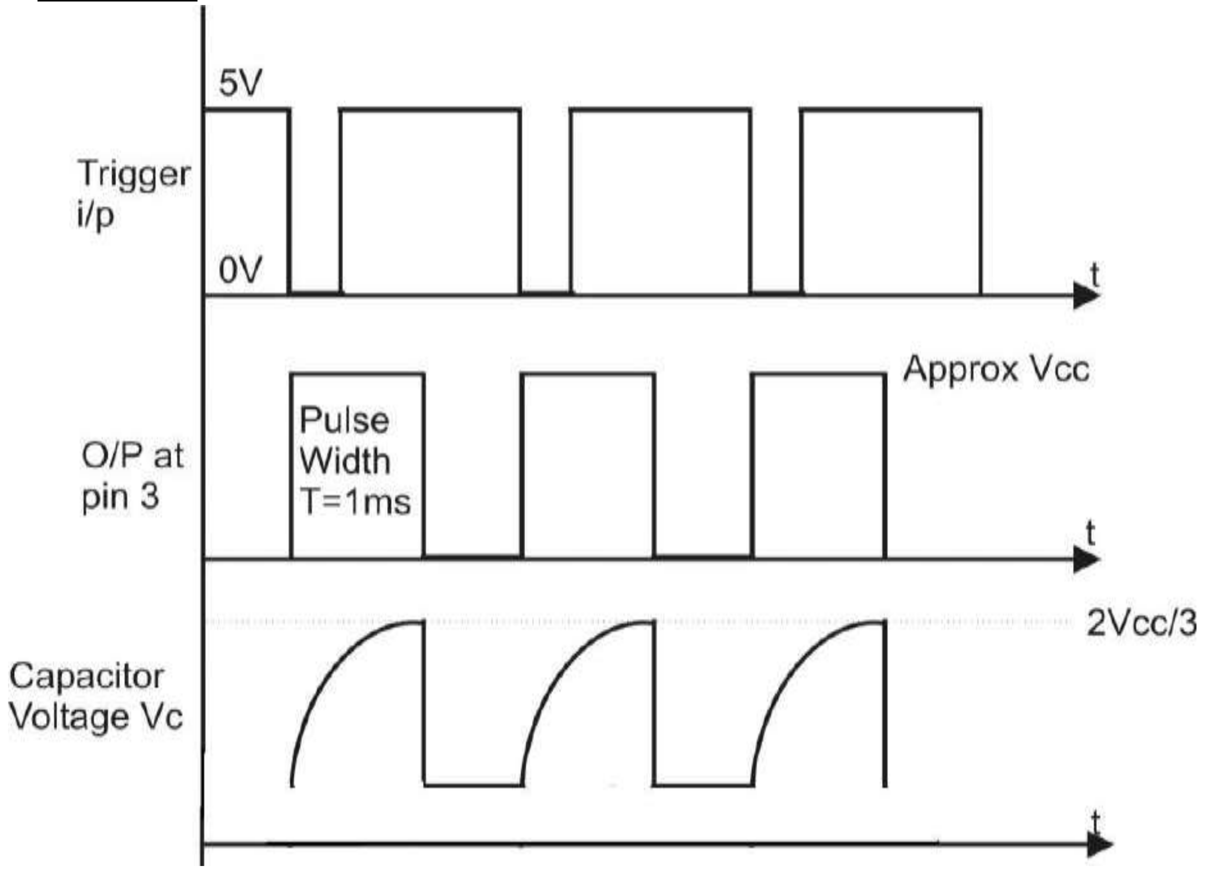
1. **Connections are made as per the circuit diagram.**
2. **Clock pulses are applied one by one and truth table is verified.**

Conclusion:-

Circuit Diagram: - Monostable Multivibrator



Waveform:-



BCAMIN03P: DATA STRUCTURE LABORATORY MANUAL

SL. No.	**Experiments**
1.	Write a program to search an element from a list. Give user the option to perform Linear or Binary search. Use Template functions.
2.	WAP using templates to sort a list of elements. Give user the option to perform sorting using Insertion sort, Bubble sort or Selection sort.
3.	Implement Linked List include functions for insertion, deletion and search of a number, reverse the list and concatenate two linked lists.
4.	Implement Circular Linked List include functions for insertion and deletion.
5.	Perform Stack operations using Linked List implementation.
6.	Perform Stack operations using Array implementation.
7.	Perform Queues operations using Circular Array implementation.
8.	WAP to scan a polynomial using linked list and add two polynomials.
9.	WAP to calculate factorial of a given no. (i) using recursion, (ii) using iteration.
10.	WAP to display Fibonacci series (i)using recursion, (ii) using iteration.
11.	WAP to calculate GCD of 2 number (i) with recursion (ii) without recursion.
12.	WAP to create a Binary Search Tree and include following operations in tree: (a) Insertion (Recursive and Iterative Implementation) (b) Deletion.
13.	WAP to create Binary Tree and display its pre-order, post-order and in-order traversals Recursively.
14.	WAP to create Binary Tree and display its pre-order, post-order and in-order traversals Iteratively.
15.	WAP to implement Diagonal Matrix using one-dimensional array.
16.	WAP to implement Lower Triangular Matrix using one-dimensional array.
17.	WAP to implement Upper Triangular Matrix using one-dimensional array.

1. ***Write a program to search an element from a list. Give user the option to perform Linear or Binary search. Use Template functions.***

Program:

```
#include<iostream>
#include<conio.h>
using namespace std;
//Linear Search Method using Template class
template <class T>
 T Linear_Search(T Iarr[],T key,int n){
     for(int i=0;i<n;i++){
         if(Iarr[i]==key){
             return i;
         }
     }
     return -1;
 }
 //Binary Search Method using Tempalte class
template <class T>
T Binary_Search(T *Iarr,T key,int n)
{
 int l,mid,h;
 l=0;
 h=n-1;
 while(l<=h)
 {
 mid=(l+h)/2;
 if(key==Iarr[mid])
     return mid;
 else if(key<Iarr[mid])
    h=mid-1;
 else
    l=mid+1;
 }
    return -1;
}
int main()
{
//Integer elements
 int Iarr[20],Ielement,i,no;
 cout<<"Enter the number of elements of Integer array: "<<endl;
 cin>>no;
 for(i=0;i<no;i++){
     cin>>Iarr[i];
 }
 cout<<"Elements of Integer Array "<<endl;
 for(i=0;i<no;i++)
 {
  cout<<Iarr[i]<<" ";
 }
 cout<<"\nEnter an item to be search using Linear_Search Method: ";
 cin>>Ielement;
 //Linear Search technique is used for Searching the Integer element
 int result=Linear_Search(Iarr,Ielement,no);
 if(result==-1){
     cout<<"Linear_Search Method Element not found in the list ";
 }
 else{
     cout<<"Linear_Search Method Element is found at position: "<<result<<endl;
 }
//Binary Search technique is used for Searching the Integer element
cout<<"\nEnter an item to be search using Binary_Search Method: ";
 cin>>Ielement;
result=Binary_Search(Iarr,Ielement,no);
if(result==-1){
     cout<<"Binary_Search Method Element not found in the list "<<endl;
 }
 else{
     cout<<"Binary_Search Method Element is found at position: "<<result<<endl;
 }
 //Float elements
 float  F_arr[10],F_element;
 cout<<"Enter the "<<no<<" elements of Float array: "<<endl;
 for(i=0;i<no;i++){
     cin>>F_arr[i];
 }
 cout<<"\nElements of Float Array \n";
 for(int i=0;i<no;i++)
 {
  cout<<F_arr[i]<<" ";
 }
 //Linear Search technique is used for Searching the Floating element
 cout<<"\nEnter an item to be search using Linear_Search Method: ";
 cin>>F_element;
 result=Linear_Search(F_arr,F_element,no);
 if(result==-1){
     cout<<"Linear_Search Method Element not found in the list ";
 }
 else{
     cout<<"Linear_Search Method Element is found at position: "<<result;
 }
 //Binary Search technique is used for Searching the Floating element
 cout<<"\nEnter an item to be search using Binary_Search Method: ";
 cin>>F_element;
```

```
result=Binary_Search(F_arr,F_element,no);
if(result==-1){
    cout<<"Binary_Search Method Element not found in the list "<<endl;
 }
 else{
    cout<<"Binary_Search Method Element is found at position: "<<result<<endl;
 }
//Character elements
char Carr[10]={'a','b','c','d','e','f','g','h','i','j'};
char Celement;
cout<<"Elements of Character Array "<<endl;
 for(i=0;i<10;i++)
 {
  cout<<Carr[i]<<" ";
 }
 //Linear Search technique is used for Searching the Character element
 cout<<"\nEnter an item to be search using Linear_Search Method: ";
 cin>>Celement;
 result=Linear_Search(Carr,Celement,10);
 if(result==-1){
    cout<<"Linear_Search Method Element not found in the list ";
 }
 else{
    cout<<"Linear_Search Method Element is found at position: "<<result<<endl;
 }
 //Binary search technique is used for Searching the Character element
 cout<<"\nEnter an item to be search using Binary_Search Method: ";
 cin>>Celement;
result=Binary_Search(Carr,Celement,10);
if(result==-1){
    cout<<"Binary_Search Method Element not found in the list "<<endl;
 }
 else{
    cout<<"Binary_Search Method Element is found at position: "<<result<<endl;
 }
 getch();
 return 0;
}
```

Input and Output Section:

```
Enter the number of elements of Integer array:
8
10 20 30 40 50 60 70 80
Elements of Integer Array
10 20 30 40 50 60 70 80
Enter an item to be search using Linear_Search Method: 20
Linear_Search Method Element is found at position: 1
```

```
Enter an item to be search using Binary_Search Method: 70
Binary_Search Method Element is found at position: 6
Enter the 8 elements of Float array:
1.1 2.3 4.5 6.4 6.55 7.1 7.11 2.1
Elements of Float Array
1.1 2.3 4.5 6.4 6.55 7.1 7.11 2.1
Enter an item to be search using Linear_Search Method: 2.1
Linear_Search Method Element is found at position: 7
Enter an item to be search using Binary_Search Method: 3.7
Binary_Search Method Element not found in the list
Elements of Character Array
a b c d e f g h i j
Enter an item to be search using Linear_Search Method: f
Linear_Search Method Element is found at position: 5
Enter an item to be search using Binary_Search Method: k
Binary_Search Method Element not found in the list
```

2. ***WAP using templates to sort a list of elements. Give user the option to perform sorting using Insertion sort, Bubble sort or Selection sort.***

Program:

```
#include <iostream>
#include<conio.h>
using namespace std;
template <class T>
void swap(T *x,T *y)
{
   T temp=*x;
   *x=*y;
   *y=temp;
}
//Bubble sort Method using template class
template <class T>
void Bubble(T A[],int n)
{
 int i,j;
 for(i=0;i<n-1;i++)
 {
   for(j=0;j<n-i-1;j++)
   {
      if(A[j]>A[j+1]){
          swap(&A[j],&A[j+1]);
         }
    }
 }
}
//Insersion sort Method using Template class
```

```
template <class T>
void Insersion(T A[],int n)
{
 int i,j;
 T x;
 for(i=1;i<n;i++)
 {
    j=i-1;
    x=A[i];
        while(j>-1 && A[j]>x)
    {
        A[j+1]=A[j];
        j--;
    }
    A[j+1]=x;
 }
}
//Selection Sort Method using Template Class
template <class T>
void SelectionSort(T A[],int n)
{
 int i,j,k;
 for(i=0;i<n-1;i++)
 {
    for(j=k=i;j<n;j++)
    {
        if(A[j]<A[k])
         k=j;
    }
    swap(&A[i],&A[k]);
 }
}

int main()
{

    //Bubble sort technique is used for Integer type array
    int A[100],i,n;
    cout<<"Enter the how many integer elements to be sort using Bubble
sorting:"<<endl;
    cin>>n;
    cout<<"Elements of integer array are: "<<endl;
    for(i=0;i<n;i++){
        cin>>A[i];
    }
    cout<<"Before Bubble sorting the elements are: "<<endl;
    for(i=0;i<n;i++){
        cout<<A[i]<<" ";
    }
    Bubble(A,n);
    cout<<"\nAfter Bubble sorting the elements are: "<<endl;
    for(i=0;i<n;i++)
    printf("%d ",A[i]);
    printf("\n");
    //Bubble sort technique is used for Floating type array
    float B[100];
    cout<<"Enter the how many float elements to be sort using Bubble
sorting:"<<endl;
    cin>>n;
    cout<<"Elements of floating array are: "<<endl;
    for(i=0;i<n;i++){
        cin>>B[i];
    }
    cout<<"Before Bubble sorting the elements are: "<<endl;
    for(i=0;i<n;i++){
        cout<<B[i]<<" ";
    }
    Bubble(B,n);
    cout<<"\nAfter Bubble sorting the elements are: "<<endl;
    for(i=0;i<n;i++)
    printf("%f ",B[i]);
    printf("\n");

//Insersion sort technique is used for integer type array
    //int A[100],i,n;
    cout<<"Enter the how many integer elements to be sort using Insersion
sorting:"<<endl;
    cin>>n;
    cout<<"Elements of integer array are: "<<endl;
    for(i=0;i<n;i++){
        cin>>A[i];
    }
    cout<<"Before Insersion sorting the elements are: "<<endl;
    for(i=0;i<n;i++){
        cout<<A[i]<<" ";
    }
    Insersion(A,n);
    cout<<"\nAfter Insersion sorting the elements are: "<<endl;
    for(i=0;i<n;i++)
    printf("%d ",A[i]);
    printf("\n");
    //Insersion sort technique is used for Floating type array
    //float B[100];
    cout<<"Enter the how many float elements to be sort using Insersion
```

```
sorting:"<<endl;
    cin>>n;
    cout<<"Elements of floating array are: "<<endl;
    for(i=0;i<n;i++){
        cin>>B[i];
    }
    cout<<"Before Insersion sorting the elements are: "<<endl;
    for(i=0;i<n;i++){
        cout<<B[i]<<" ";
    }
    Insersion(B,n);
    cout<<"\nAfter Insersion sorting the elements are: "<<endl;
    for(i=0;i<n;i++)
    printf("%f ",B[i]);
    printf("\n");

//Selection sort technique is used for integer type array
    //int A[100],i,n;
    cout<<"Enter the how many integer elements to be sort using Selection
sorting:"<<endl;
    cin>>n;
    cout<<"Elements of integer array are: "<<endl;
    for(i=0;i<n;i++){
        cin>>A[i];
    }
    cout<<"Before Selection sorting the elements are: "<<endl;
    for(i=0;i<n;i++){
        cout<<A[i]<<" ";
    }
    SelectionSort(A,n);
    cout<<"\nAfter Selection sorting the elements are: "<<endl;
    for(i=0;i<n;i++)
    printf("%d ",A[i]);
    printf("\n");
    //Insersion sort technique is used for Floating type array
   // float B[100];
    cout<<"Enter the how many float elements to be sort using Selection
sorting:"<<endl;
    cin>>n;
    cout<<"Elements of floating array are: "<<endl;
    for(i=0;i<n;i++){
        cin>>B[i];
    }
    cout<<"Before Selection sorting the elements are: "<<endl;
    for(i=0;i<n;i++){
        cout<<B[i]<<" ";
    }
    SelectionSort(B,n);
    cout<<"\nAfter Selection sorting the elements are: "<<endl;
    for(i=0;i<n;i++)
    printf("%f ",B[i]);
    printf("\n");
    getch();
    return 0;
}
```

Input and Output Section:

```
Enter the how many integer elements to be sort using Bubble sorting:
5
Elements of integer array are:
10 23 09 11 15
Before Bubble sorting the elements are:
10 23 9 11 15
After Bubble sorting the elements are:
9 10 11 15 23
Enter the how many float elements to be sort using Bubble sorting:
5
Elements of floating array are:
9.1 2.3 4.5 1.1 2.9
Before Bubble sorting the elements are:
9.1 2.3 4.5 1.1 2.9
After Bubble sorting the elements are:
1.100000 2.300000 2.900000 4.500000 9.100000
Enter the how many integer elements to be sort using Insersion sorting:
6
Elements of integer array are:
19 24 10 11 14
23
Before Insersion sorting the elements are:
19 24 10 11 14 23
After Insersion sorting the elements are:
10 11 14 19 23 24
Enter the how many float elements to be sort using Insersion sorting:
3
Elements of floating array are:
2.3 1.2 .9
Before Insersion sorting the elements are:
2.3 1.2 0.9
After Insersion sorting the elements are:
0.900000 1.200000 2.300000
Enter the how many integer elements to be sort using Selection sorting:
10
Elements of integer array are:
10 20 50 30 21 35 44 9 99 102
```

```
Before Selection sorting the elements are:
10 20 50 30 21 35 44 9 99 102
After Selection sorting the elements are:
9 10 20 21 30 35 44 50 99 102
Enter the how many float elements to be sort using Selection sorting:
5
Elements of floating array are:
1.1 .9 .2 .99 1.234
Before Selection sorting the elements are:
1.1 0.9 0.2 0.99 1.234
After Selection sorting the elements are:
0.200000 0.900000 0.990000 1.100000 1.234000
```

3. ***Implement Linked List include functions for insertion, deletion and search of a number, reverse the list and concatenate two linked lists.***

Insertion Operation using linked list:

Program:

```
        /* linked List Insertion */
#include <stdio.h>
#include <stdlib.h>
struct Node
{
 int data;
 struct Node *next;
}*first=NULL;

void create(int A[],int n)
{
 int i;
 struct Node *t,*last;
 first=(struct Node *)malloc(sizeof(struct Node));
 first->data=A[0];
 first->next=NULL;
 last=first;
 for(i=1;i<n;i++)
 {
   t=(struct Node*)malloc(sizeof(struct Node));
   t->data=A[i];
   t->next=NULL;
   last->next=t;
   last=t;
 }
}
int count(struct Node *p)
{
 int l=0;
 while(p)
 {
   l++;
   p=p->next;
 }
 return l;
}
void Display(struct Node *p)
{
 while(p!=NULL)
 {
   printf("%d ",p->data);
   p=p->next;
 }
}
void Insert(struct Node *p,int index,int x)
{
 struct Node *t;
 int i;

 if(index < 0 || index > count(p))
   return;
 t=(struct Node *)malloc(sizeof(struct Node));
 t->data=x;
 if(index == 0)
 {
   t->next=first;
   first=t;
 }
 else
 {
   for(i=0;i<index-1;i++)
   p=p->next;
   t->next=p->next;
   p->next=t;
 }
}
int main()
{
```

```
 //int A[]={10,20,30,40,50};
 //create(A,5);
```

```
//Insert Node
 printf("Node Insert: \n");
 Insert(first,0,5);
 Display(first);
 printf("\nNode Insert: \n");
 Insert(first,1,10);
 Display(first);
 //First Node Insert
 printf("\nFirst Node Insert: \n");
 Insert(first,0,15);
 Display(first);
 //Last Node Insert
 printf("\nLast Node Insert: \n");
 Insert(first,3,20);
 Display(first);
 //Any position Node Insert
 printf("\nGiven position Node Insert: \n");
 Insert(first,2,39);
 Display(first);
 return 0;
}
```

Input and Output Section:

```
Node Insert:
5
Node Insert:
5 10
First Node Insert:
15 5 10
Last Node Insert:
15 5 10 20
Given position Node Insert:
15 5 39 10 20
```

Deletion Operation using linked list

Program:

```
/* Linked List Delete element */
#include <stdio.h>
#include <stdlib.h>
struct Node
{
 int data;
 struct Node *next;
}*first=NULL;
void create(int A[],int n)
```

```
{
 int i;
 struct Node *t,*last;
 first=(struct Node *)malloc(sizeof(struct Node));
 first->data=A[0];
 first->next=NULL;
 last=first;
 for(i=1;i<n;i++)
 {
    t=(struct Node*)malloc(sizeof(struct Node));
    t->data=A[i];
    t->next=NULL;
    last->next=t;
    last=t;
 }
}
void Display(struct Node *p)
{
 while(p!=NULL)
 {
    printf("%d ",p->data);
    p=p->next;
 }
}
int count(struct Node *p)
{
 int l=0;
 while(p)
 {
    l++;
    p=p->next;
 }
 return l;
}
int Delete(struct Node *p,int index)
{
 struct Node *q=NULL;
 int x=-1,i;
 if(index < 1 || index > count(p))
    return -1;
 if(index==1)
 {
    q=first;
    x=first->data;
    first=first->next;
    free(q);
    return x;
 }
 else
 {
    for(i=0;i<index-1;i++)
    {
    q=p;
    p=p->next;
    }
    q->next=p->next;
    x=p->data;
    free(p);
    return x;
 }
}
int main()
{
 int i;
 int A[]={10,20,30,40,50};
 create(A,5);
 printf("Lined List element are: \n");
 Display(first);
 printf("\nDelete element %d\n",Delete(first,2));
 Display(first);
 printf("\nDelete element %d\n",Delete(first,1));
 Display(first);
 printf("\nDelete element %d\n",Delete(first,0));
 Display(first);
         return 0;
}
```

Input and Output Section:

```
Lined List element are:
10 20 30 40 50
Delete element 20
10 30 40 50
Delete element 10
30 40 50
Delete element -1
30 40 50
```

Search of a number using linked list

Program:

```
/*Searching of a number using Lined List */
#include <stdio.h>
#include <stdlib.h>
struct Node
{
 int data;
 struct Node *next;
}*first=NULL;
void create(int A[],int n)
{
 int i;
 struct Node *t,*last;
 first=(struct Node *)malloc(sizeof(struct Node));
 first->data=A[0];
 first->next=NULL;
 last=first;
 for(i=1;i<n;i++)
 {
    t=(struct Node*)malloc(sizeof(struct Node));
    t->data=A[i];
    t->next=NULL;
    last->next=t;
    last=t;
 }
}
struct Node * LSearch(struct Node *p,int key)
{
 while(p!=NULL)
 {
     if(key==p->data){
            return p;
     }
     else{
        p=p->next;
     }
 }
    return NULL;
}
struct Node * RSearch(struct Node *p,int key)
{
 if(p==NULL)
    return NULL;
 if(key==p->data)
    return p;
 return RSearch(p->next,key);

}
int main()
{
 struct Node *temp;
 int A[]={3,5,7,10,25,8,32,2},i;
 printf("Linked list elements are: \n");
 for(i=0;i<8;i++){
    printf(" %d ",A[i]);
 }
 //Recursive LinearSearch
 create(A,8);

 temp=RSearch(first,8);
 if(temp)
    printf("\nKey is Found* %d \n",temp->data);
else
    printf("Key is not Found: \n");
//Iterative LinearSearch
temp=LSearch(first,27);
if(temp)
    printf("\nKey is Found* %d \n",temp->data);
else
    printf("Key is not found: ");

 return 0;
}
```

Input and Output Section:

```
Linked list elements are:
 3  5  7  10  25  8  32  2
Key is Found* 8
Key is not found:
```

Reverse a linked list:

Program:

```
/* Reverse a Linked List */
#include <stdio.h>
#include <stdlib.h>
struct Node
{
 int data;
 struct Node *next;
}*first=NULL,*second=NULL,*third=NULL;
void Display(struct Node *p)
{
 while(p!=NULL)
 {
 printf("%d ",p->data);
 p=p->next;
 }
}
void create(int A[],int n)
```

```
{
 int i;
 struct Node *t,*last;
 first=(struct Node *)malloc(sizeof(struct Node));
 first->data=A[0];
 first->next=NULL;
 last=first;

 for(i=1;i<n;i++)
 {
 t=(struct Node*)malloc(sizeof(struct Node));
 t->data=A[i];
 t->next=NULL;
 last->next=t;
 last=t;
 }
}
int count(struct Node *p)
{
 int l=0;
 while(p)
 {
    l++;
    p=p->next;
 }
 return l;
}
void Reverse1(struct Node *p)
{
 int *A,i=0;
 struct Node *q=p;

 A=(int *)malloc(sizeof(int)*count(p));

 while(q!=NULL)
 {
 A[i]=q->data;
 q=q->next;
 i++;
 }
 q=p;
 i--;
 while(q!=NULL)
 {
 q->data=A[i];
 q=q->next;
 i--;
 }
}
void Reverse2(struct Node *p)
{
 struct Node *q=NULL,*r=NULL;

 while(p!=NULL)
 {
 r=q;
 q=p;
 p=p->next;
 q->next=r;
 }
 first=q;
}
void Reverse3(struct Node *q,struct Node *p)
{
 if(p)
 {
 Reverse3(p,p->next);
 p->next=q;
 }
 else
 first=q;
}

int main()
{

 int A[]={10,20,40,50,60};
 create(A,5);
 printf("Linked list elements are: \n");
 Display(first);

 Reverse3(NULL,first);
 printf("\nReverse Linked list elements are: \n");
 Display(first);

 return 0;
}
```

Input and Output Section:

```
Linked list elements are:
10 20 40 50 60
Reverse Linked list elements are:
60 50 40 20 10
```

Concatenate of two linked list:

Program:

```
/*Concatenate two Linked List*/
#include <stdio.h>
#include <stdlib.h>
struct Node
{
 int data;
 struct Node *next;
}*first=NULL,*second=NULL,*third=NULL;
void Display(struct Node *p)
{
 while(p!=NULL)
 {
 printf("%d ",p->data);
 p=p->next;
 }
}
void create(int A[],int n)
{
 int i;
 struct Node *t,*last;
 first=(struct Node *)malloc(sizeof(struct Node));
 first->data=A[0];
 first->next=NULL;
 last=first;

 for(i=1;i<n;i++)
 {
 t=(struct Node*)malloc(sizeof(struct Node));
 t->data=A[i];
 t->next=NULL;
 last->next=t;
 last=t;
 }
}
void create2(int A[],int n)
{
 int i;
 struct Node *t,*last;
 second=(struct Node *)malloc(sizeof(struct Node));
 second->data=A[0];
 second->next=NULL;
 last=second;

 for(i=1;i<n;i++)
 {
 t=(struct Node*)malloc(sizeof(struct Node));
 t->data=A[i];
 t->next=NULL;
 last->next=t;
 last=t;
 }
}
/*void Merge(struct Node *p,struct Node *q)
{
 struct Node *last;
 if(p->data < q->data)
 {
 third=last=p;
 p=p->next;
 third->next=NULL;
 }
 else
 {
 third=last=q;
 q=q->next;
 third->next=NULL;
 }
 while(p && q)
 {
 if(p->data < q->data)
 {
 last->next=p;
 last=p;
 p=p->next;
 last->next=NULL;

 }
 else
 {
 last->next=q;
 last=q;
 q=q->next;
 last->next=NULL;

 }
 }
 if(p)last->next=p;
 if(q)last->next=q;
```

```
}
*/
```

```
void Concat(struct Node *p,struct Node *q){
    third=p;
    while(p->next!=NULL){
        p=p->next;
    }
    p->next=q;

}
int main()
{

 int A[]={10,20,40,50,60};
 int B[]={15,18,25,30,55};
 create(A,5);
 create2(B,5);


 //Merge(frist,second);
 //Display(third);
 printf("First Linked List \n");
 Display(first);
 printf("\n");
 printf("Second Linked List \n");
 Display(second);
 printf("\n");
 Concat(first,second);
 printf("Concatenate two Linked List: \n");
 Display(third);
 printf("\n\n");
 return 0;
}
```

Input and Output Section:

```
First Linked List
10 20 40 50 60
Second Linked List
15 18 25 30 55
Concatenate two Linked List:
10 20 40 50 60 15 18 25 30 55
```

4. ***Implement Circular Linked List include functions for insertion and deletion.***

Circular Linked List for Insertion Operation:

Program:

```
/*Circular Linked List*/
#include <stdio.h>
```

```
#include <stdlib.h>
#include<conio.h>
struct Node
{
 int data;
 struct Node *next;
}*Head;
void create(int A[],int n)
{
 int i;
 struct Node *t,*last;
 Head=(struct Node*)malloc(sizeof(struct Node));
 Head->data=A[0];
 Head->next=Head;
 last=Head;

 for(i=1;i<n;i++)
 {
    t=(struct Node*)malloc(sizeof(struct Node));
    t->data=A[i];
    t->next=last->next;
    last->next=t;
    last=t;
 }
}
void Display(struct Node *h)
{
 do
 {
    printf("%d ",h->data);
    h=h->next;
 }while(h!=Head);
    printf("\n");
}
/*void RDisplay(struct Node *h)
{
 static int flag=0;
 if(h!=Head || flag==0)
 {
 flag=1;
 printf("%d ",h->data);
 RDisplay(h->next);
 }
 flag=0;
}
*/
int Length(struct Node *p)
```

```
{
   int len=0;
 do
 {
   len++;
   p=p->next;

 }while(p!=Head);
   return len;
}
void Insert(struct Node *p,int index, int x)
{
 struct Node *t;
 int i;
 if(index<0 || index > Length(p))
 return;

 if(index==0)
 {
   t=(struct Node *)malloc(sizeof(struct Node));
   t->data=x;
   if(Head==NULL)
   {
      Head=t;
      Head->next=Head;
   }
 else
 {
   while(p->next!=Head)p=p->next;
   p->next=t;
   t->next=Head;
   Head=t;
 }
 }
 else
 {
   for(i=0;i<index-1;i++)p=p->next;
   t=(struct Node *)malloc(sizeof(struct Node));
   t->data=x;
   t->next=p->next;
   p->next=t;
 }
}

int main()
{
 int A[]={2,3,4,5,6};
 create(A,5);
 printf("Linked List are: \n");
 Display(Head);

//Insert Node
 printf("\nNode Insert: \n");
 Insert(Head,2,10);
 Display(Head);
 //First Node Insert
 printf("\nFirst Node Insert: \n");
 Insert(Head,0,15);
 Display(Head);
 //Last Node Insert
 printf("\nLast Node Insert: \n");
 Insert(Head,7,20);
 Display(Head);
 //Any position Node Insert
 printf("\nGiven position Node Insert: \n");
 Insert(Head,2,39);
 Display(Head);
getch();
return 0;
}
```

Input and Output Section:

```
Linked List are:
2 3 4 5 6

Node Insert:
2 3 10 4 5 6

First Node Insert:
15 2 3 10 4 5 6

Last Node Insert:
15 2 3 10 4 5 6 20

Given position Node Insert:
15 2 39 3 10 4 5 6 20
```

Circular Linked List for Deletion Operation:

Program:

```
/*Circular Linked List Deletion*/
#include <stdio.h>
#include <stdlib.h>
#include<conio.h>
```

```
struct Node
{
 int data;
 struct Node *next;
}*Head;
void create(int A[],int n)
{
 int i;
 struct Node *t,*last;
 Head=(struct Node*)malloc(sizeof(struct Node));
 Head->data=A[0];
 Head->next=Head;
 last=Head;
 for(i=1;i<n;i++)
 {
    t=(struct Node*)malloc(sizeof(struct Node));
    t->data=A[i];
    t->next=last->next;
    last->next=t;
    last=t;
 }
}
void Display(struct Node *h)
{
 do
 {
    printf("%d ",h->data);
    h=h->next;
 }while(h!=Head);
    printf("\n");
}
int Length(struct Node *p)
{
    int len=0;
 do
 {
    len++;
    p=p->next;

 }while(p!=Head);
    return len;
}

int Delete(struct Node *p,int index)
{
    struct Node *q;
    int i,x;
 if(index <0 || index >Length(Head))
    return -1;
 if(index==1)
 {
    while(p->next!=Head)p=p->next;
    x=Head->data;
 if(Head==p)
 {
    free(Head);
    Head=NULL;
 }
 else
 {
    p->next=Head->next;
    free(Head);
    Head=p->next;
    }
 }
 else
 {
    for(i=0;i<index-2;i++)
    p=p->next;
    q=p->next;
    p->next=q->next;
    x=q->data;
    free(q);
 }
 return x;
}
int main()
{
 int A[]={2,3,4,5,6,7,8,9,10};
 create(A,9);
 printf("Linked List are: \n");
 Display(Head);

 //First Node Delete
 printf("\nFirst Node Delete: \n");
 Delete(Head,1);
 Display(Head);

 //Last Node Delete
 printf("\nLast Node Delete: \n");
 Delete(Head,8);
 Display(Head);
 //Any position Node Insert
 printf("\nGiven position Node Delete: \n");
 Delete(Head,5);
 Display(Head);
 getch();
return 0;
}
```

Input and Output Section:

```
Linked List are:
2 3 4 5 6 7 8 9 10

First Node Delete:
3 4 5 6 7 8 9 10

Last Node Delete:
3 4 5 6 7 8 9

Given position Node Delete:
3 4 5 6 8 9
```

5. ***Perform Stack operations using Linked List implementation.***

Program:

```
/*Stack using Linked List*/
#include <stdio.h>
#include <stdlib.h>
struct Node
{
 int data;
 struct Node *next;
}*top=NULL;

void push(int x)
{
 struct Node *t;
 t=(struct Node*)malloc(sizeof(struct Node));
 if(t==NULL)
    printf("stack is full\n");
 else
 {
    t->data=x;
    t->next=top;
    top=t;
 }
}
```

```
int pop()
{
 struct Node *t;
 int x=-1;
 if(top==NULL)
    printf("Stack is Empty\n");
 else
 {
    t=top;
    top=top->next;
    x=t->data;
    free(t);
 }
 return x;
}

void Display()
{
 struct Node *p;
 p=top;
 while(p!=NULL)
 {
    printf("%d ",p->data);
    p=p->next;
 }
 printf("\n");
}
int main()
{
// push the elements
printf("Push the elements are: \n");
//last elements top of the stack
 push(10);
 push(20);
 push(30);
 push(40);
 push(50);
 Display();

 printf("pop element %d \n",pop());
 printf("pop element %d %d ",pop(),pop()); //right to left operation
 return 0;
}
```

Input and Output Section:

```
Push the elements are:
50 40 30 20 10
```

```
pop element 50
pop element 30 40
```

6. Perform Stack operations using Array implementation.

Program:

```
/* Stack using Array */
#include <stdio.h>
#include <stdlib.h>
struct Stack
{
 int size;
 int top;
 int *S;
};
void create(struct Stack *st)
{
 printf("Enter Size ");
 scanf("%d",&st->size);
 st->top=-1;
 st->S=(int *)malloc(st->size*sizeof(int));
}
void Display(struct Stack st)
{
 int i;
 for(i=st.top;i>=0;i--)
    printf("%d ",st.S[i]);
 printf("\n");

}
void push(struct Stack *st,int x)
{
 if(st->top==st->size-1)
    printf("Stack overflow\n");
 else
 {
    st->top++;
    st->S[st->top]=x;
 }
}
int pop(struct Stack *st)
{
    int x=-1;
    if(st->top==-1)
       printf("Stack Underflow\n");
    else
    {
        x=st->S[st->top--];
     }
 return x;
}

int peek(struct Stack st,int index)
{
 int x=-1;
 if(st.top-index+1<0)
    printf("Invalid Index \n");
    x=st.S[st.top-index+1];
 return x;
}
int isEmpty(struct Stack st)
{
 if(st.top==-1)
    return 1;
 return 0;
}
int isFull(struct Stack st)
{
    return st.top==st.size-1;
}
int stackTop(struct Stack st)
{
    if(!isEmpty(st))
         return st.S[st.top];
 return -1;
}
int main()
{
 struct Stack st;
 create(&st);
 //push the elements into the Stack
 printf("Stack elements are: \n");
 push(&st,10);
 push(&st,20);
 push(&st,30);
 push(&st,40);
 push(&st,50);
 push(&st,60);
 //Stack overflow when Enter the size 5
 Display(st);
 //pop the elements from a Stack
 printf("pop element: %d \n",pop(&st));
 printf("After the pop operation Stack elements are: \n");
 Display(st);
 //Stack Underflow when more then 6 elements pop out
 //printf("pop element: %d %d %d %d %d
     %d\n",pop(&st),pop(&st),pop(&st),pop(&st),pop(&st),pop(&st));

 //Peek the element into the Stack
 printf("peek element: %d \n",peek(st,1));
 printf("After the peek operation Stack elements are: \n");
 Display(st);
 return 0;
}
```

Input and Output Section:

```
Enter Size 6
Stack elements are:
60 50 40 30 20 10
pop element: 60
After the pop operation Stack elements are:
50 40 30 20 10
peek element: 50
After the peek operation Stack elements are:
50 40 30 20 10

Enter Size 5
Stack elements are:
Stack overflow
50 40 30 20 10
pop element: 50
After the pop operation Stack elements are:
40 30 20 10
peek element: 40
After the peek operation Stack elements are:
40 30 20 10

Enter Size 6
Stack elements are:
60 50 40 30 20 10
pop element: 60
After the pop operation Stack elements are:
50 40 30 20 10
Stack Underflow
pop element: -1 10 20 30 40 50
Invalid Index
peek element: 0
After the peek operation Stack elements are:
```

7. Perform Queues operations using Circular Array implementation.

Program:

```
/* Queue Operation Using Circular Array Implementation*/
#include <stdio.h>
#include <stdlib.h>
struct Queue
{
 int size;
 int front;
 int rear;
 int *Q;
};
void create(struct Queue *q,int size)
{
 q->size=size;
 q->front=q->rear=0;
 q->Q=(int *)malloc(q->size*sizeof(int));
}
void enqueue(struct Queue *q,int x)
{
 if((q->rear+1)%q->size==q->front)
 printf("Queue is Full\n");
 else
 {
 q->rear=(q->rear+1)%q->size;
 q->Q[q->rear]=x;
 }
}
int dequeue(struct Queue *q)
{
 int x=-1;

 if(q->front==q->rear)
 printf("Queue is Empty\n");
 else
 {
 q->front=(q->front+1)%q->size;
 x=q->Q[q->front];
 }
 return x;
}
void Display(struct Queue q)
{
 int i=q.front+1;

 do
 {
```

```
 printf("%d ",q.Q[i]);
 i=(i+1)%q.size;
 }while(i!=(q.rear+1)%q.size);

 printf("\n");
}
int main()
{
 struct Queue q;
 create(&q,6);
 printf("\nQueue elements are: \n");
 //enqueue Operation
 enqueue(&q,10);
 enqueue(&q,20);
 enqueue(&q,30);
 enqueue(&q,40);
 enqueue(&q,50);
 //enqueue(&q,60);
 Display(q);
 //dequeue Operation
 printf("dequeue element: %d \n",dequeue(&q));
 printf("After dequeue Queue elements are: \n");
 Display(q);
 return 0;
}
```

Input and Output Section:

```
Queue elements are:
10 20 30 40 50
dequeue element: 10
After dequeue Queue elements are:
20 30 40 50
```

8. WAP to scan a polynomial using linked list and add two polynomials.

Program:

```
//Polynomial addition using linked list
# include <stdio.h>
# include <stdlib.h>
struct node
{
     float coef;
     int expo;
     struct node *link;
};

 int display(struct node *ptr)
```

```
{
    if(ptr==NULL)
    {
        printf("Empty\n");
        return 0;
    }
    while(ptr!=NULL)
    {
        printf("(%.1fx^%d) + ", ptr->coef,ptr->expo);
        ptr=ptr->link;
    }
    printf("\b\b \n"); // \b\b to erase the last + sign
}
struct node *insert(struct node *start,float co,int ex)
{
    struct node *ptr,*tmp;
    tmp= (struct node*)malloc(sizeof(struct node));
    tmp->coef=co;
    tmp->expo=ex;

    //list empty or exp greater than first one
    if(start==NULL || ex>start->expo)
    {
        tmp->link=start;
        start=tmp;
    }
    else
    {
        ptr=start;
        while(ptr->link!=NULL && ptr->link->expo>ex)
                ptr=ptr->link;
        tmp->link=ptr->link;
        ptr->link=tmp;
        if(ptr->link==NULL)  //item to be added in the end
                tmp->link=NULL;
    }
    return start;
}
struct node *enter(struct node *start)
{
    int i,n,ex;
    float co;
    printf("How many terms you want to enter : ");
    scanf("%d",&n);
    printf("Enter each term with coeff and exp\n");
    for(i=1;i<=n;i++)
    {
        scanf("%f %d",&co,&ex);
        start=insert(start,co,ex);
    }
    return start;
}
struct node *poly_add(struct node *p1,struct node *p2)
{
    struct node *p3_start,*p3,*tmp;
    p3_start=NULL;
    if(p1==NULL && p2==NULL)
        return p3_start;

    while(p1!=NULL && p2!=NULL )
    {
        tmp= (struct node*)malloc(sizeof(struct node));
        if(p3_start==NULL)
        {
                p3_start=tmp;
                p3=p3_start;
        }
        else
        {
                p3->link=tmp;
                p3=p3->link;
        }
        if(p1->expo > p2->expo)
        {
                tmp->coef=p1->coef;
                tmp->expo=p1->expo;
                p1=p1->link;
        }
        else
                if(p2->expo > p1->expo)
                {
                        tmp->coef=p2->coef;
                        tmp->expo=p2->expo;
                        p2=p2->link;
                }
                else
                        if(p1->expo == p2->expo)
                        {
                                tmp->coef=p1->coef + p2->coef;
                                tmp->expo=p1->expo;
                                p1=p1->link;
                                p2=p2->link;
                        }
    }
```

```
    while(p1!=NULL)
    {
        tmp= (struct node*)malloc(sizeof(struct node));
        tmp->coef=p1->coef;
        tmp->expo=p1->expo;
        if (p3_start==NULL) /*poly 2 is empty*/
        {
                p3_start=tmp;
                p3=p3_start;
        }
        else
        {
                p3->link=tmp;
                p3=p3->link;
        }
        p1=p1->link;
    }
    while(p2!=NULL)
    {
        tmp= (struct node*)malloc(sizeof(struct node));
        tmp->coef=p2->coef;
        tmp->expo=p2->expo;
        if (p3_start==NULL) /*poly 1 is empty*/
        {
                p3_start=tmp;
                p3=p3_start;
        }
        else
        {
                p3->link=tmp;
                p3=p3->link;
        }
        p2=p2->link;
}
    p3->link=NULL;
    return p3_start;
}
int main( )
{
    struct node *p1_start,*p2_start,*p3_start;
    p1_start=NULL;
    p2_start=NULL;
    p3_start=NULL;
    printf("Polynomial 1 :\n");
    p1_start=enter(p1_start);
    printf("Polynomial 2 :\n");
    p2_start=enter(p2_start);
```

```
     p3_start=poly_add(p1_start,p2_start);
     printf("Polynomial 1 is :  ");
     display(p1_start);
     printf("Polynomial 2 is :  ");
     display(p2_start);
     printf("Added polynomial is :  ");
     display(p3_start);
     return 0;
}
```

Input and Output Section:

```
Polynomial 1 :
How many terms you want to enter : 3
Enter each term with coeff and exp
1.2 5
2.5 4
4 3
Polynomial 2 :
How many terms you want to enter : 3
Enter each term with coeff and exp
2 6
1 2
1 0
Polynomial 1 is :  (1.2x^5) + (2.5x^4) + (4.0x^3)
Polynomial 2 is :  (2.0x^6) + (1.0x^2) + (1.0x^0)
Added polynomial is :  (2.0x^6) + (1.2x^5) + (2.5x^4) + (4.0x^3) + (1.0x^2) +
(1.0x^0)
```

9. ***WAP to calculate factorial of a given no. (i) using recursion, (ii) using iteration.***

Program:

```
#include <stdio.h>
#include<conio.h>
//Iterative function definition
long ifact(int n){
 long fact=1,i;
 for(i=1;i<=n;i++){
     fact=fact*i;
 }
 return fact;
}
//Recursion function definition
long rfact(int n){
    //base condition
    if(n<=1){
        return n;
    }
    // Recrsive Procedure
    else{
        return n*rfact(n-1);
    }
}
int main()
{
    int n;
    printf("Enter the number \n");
    scanf("%d",&n);
    //iterative function call
    printf("Iteration Method: %d Factorial is %ld",n,ifact(n));
    //recursion function call
    printf("\nRecursion Method: %d Factorial is %ld",n,rfact(n));
    getch();
    return 0;
}
```

Input and Output Section:

```
Enter the number
5
Iteration Method: 5 Factorial is 120
Recursion Method: 5 Factorial is 120
Enter the number
9
Iteration Method: 9 Factorial is 362880
Recursion Method: 9 Factorial is 362880
```

10. WAP to display Fibonacci series (i)using recursion, (ii) using iteration.

Program:

```
#include <stdio.h>
#include<conio.h>
//Iterative function definition
void ifib(int n){
    int a=0,b=1,c,i;
    if(n<1){
        printf("Fibonacci series : %d ",a);
    }
    else if(n==1){
        printf("Fibonacci series : %d %d ",a,b);
            }
        else{
            printf("Fibonacci series are: %d %d",a,b);
            for(i=2;i<=n;i++){
                c=a+b;
                a=b;
                b=c;

            printf(" %d",c);
            }
        }
    }
//Recursion function definition
int rfib(int n){
    int a=0,b=1;
//base condition
    if(n<=1){
        return n;
    }
//Recursive Procedure
    else{
        return rfib(n-2)+rfib(n-1);
    }
}
int main()
{
    int n,result;
    printf("Enter the number \n");
    scanf("%d",&n);
    //iterative function call
    ifib(n);
    //recursion function call
    result=rfib(n);
    printf("\nLast value of Fibonacci series %d",result);
    getch();
    return 0;
}
```

Input and Output Section:

```
Enter the number
13
Fibonacci series are: 0 1 1 2 3 5 8 13 21 34 55 89 144 233
Last value of Fibonacci series 233

Enter the number
10
Fibonacci series are: 0 1 1 2 3 5 8 13 21 34 55
Last value of Fibonacci series 55
```

11.WAP to calculate GCD of 2 number (i) with recursion (ii) without recursion.

Program:

```
#include <stdio.h>
#include<conio.h>
//Iterative function definition
int igcd(int n,int m){
    while(m!=n){
        if(m>n){
            m=m-n;
        }
        else{
            n=n-m;
        }
            }
return m;
    }
//Recursion function definition
int rgcd(int n,int m){
    //base condition
    if(n==m){
        return n;
    }
    // Recrsive Procedure
    if(m>n){
        return rgcd(m-n,n);
    }
        rgcd(m,n-m);
}
int main()
{
    int a,b;
    printf("Enter the two number \n");
    scanf("%d%d",&a,&b);
    //iterative function call
    printf("Iterative Method %d and %d GCD is %d ",a,b,igcd(a,b));
    //recursion function call
    printf("\nRecursion Method %d and %d GCD is %d",a,b,rgcd(a,b));
    getch();
    return 0;
}
```

Input and Output Section:

```
Enter the two number
13 117
Iterative Method 13 and 117 GCD is 13
Recursion Method 13 and 117 GCD is 13
Enter the two number
93 131
```

```
Iterative Method 93 and 131 GCD is 1
Recursion Method 93 and 131 GCD is 1
```

12. ***WAP to create a Binary Search Tree and include following operations in tree: (a) Insertion (Recursive and Iterative Implementation) (b) Deletion.***

    (a) Binary Search Tree Insertion Operation (Recursive and Iterative Implementation):

Program:

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
struct Node{
    struct Node *lchild;
    int data;
    struct Node *rchild;
}*root=NULL;
void Insert(int key)
{
    struct Node *t=root;
    struct Node *r,*p;
    if(root==NULL){
        p=(struct Node *)malloc(sizeof(struct Node));
        p->data=key;
        p->lchild=p->rchild=NULL;
        root=p;
        return ;
    }
    while(t!=NULL){
        r=t;
        if(key<t->data)
            t=t->lchild;
        else if(key>t->data)
            t=t->rchild;
        else
            return;
    }
        p=(struct Node *)malloc(sizeof(struct Node));
        p->data=key;
        p->lchild=p->rchild=NULL;
        if(key<r->data) r->lchild=p;
        else r->rchild=p;
}
    void Inorder(struct Node *p){
        if(p){
            Inorder(p->lchild);
            printf("%d ",p->data);
            Inorder(p->rchild);
        }
    }
struct Node * Search(int key){
    struct Node *t=root;
    while(t!=NULL){
        if(key==t->data)
            return t;
        else if(key<t->data)
            t=t->lchild;
        else
            t=t->rchild;
    }
    return NULL;
}

struct Node *RInsert(struct Node *p,int key){
    struct Node *t=NULL;
    if(p==NULL){
        t=(struct Node *)malloc(sizeof(struct Node));
        t->data=key;
        t->lchild=t->rchild=NULL;
        return t;
    }
        if(key<p->data)
            p->lchild=RInsert(p->lchild,key);
        else if(key>p->data)
            p->rchild=RInsert(p->rchild,key);

        return p;
}

int main(){
    struct Node *temp;
    printf("Iterative BST: \n");
    Insert(10);
    Insert(5);
    Insert(20);
    Insert(8);
    Insert(30);
    Inorder(root);
    printf("\n");
```

```
     temp=Search(30);
    if(temp!=NULL){
        printf("Element %d is Found \n",temp->data);
    }
    else
        printf("Element is not found \n");

    printf("Recursive BST: \n");
    root=RInsert(root,10);
    RInsert(root,5);
    RInsert(root,20);
    RInsert(root,8);
    RInsert(root,30);
    Inorder(root);
    printf("\n");
    temp=Search(20);
    if(temp!=NULL){
        printf("Element %d is Found \n",temp->data);
    }
    else
        printf("Element is not found \n");
    getch();
    return 0;
}
```

Input and Output Section:

```
Iterative BST:
5 8 10 20 30
Element 30 is Found
Recursive BST:
5 8 10 20 30
Element 20 is Found
```

(b) Binary Search Tree Deletion Operation (Recursive and Iterative Implementation):

Program:

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
struct Node{
    struct Node *lchild;
    int data;
    struct Node *rchild;
}*root=NULL;
void Insert(int key)
{
    struct Node *t=root;
    struct Node *r,*p;
    if(root==NULL){
        p=(struct Node *)malloc(sizeof(struct Node));
        p->data=key;
        p->lchild=p->rchild=NULL;
        root=p;
        return ;
    }
    while(t!=NULL){
        r=t;
        if(key<t->data)
            t=t->lchild;
        else if(key>t->data)
            t=t->rchild;
        else
            return;
    }
        p=(struct Node *)malloc(sizeof(struct Node));
        p->data=key;
        p->lchild=p->rchild=NULL;
        if(key<r->data) r->lchild=p;
        else r->rchild=p;
}
    void Inorder(struct Node *p){
        if(p){
            Inorder(p->lchild);
            printf("%d ",p->data);
            Inorder(p->rchild);
        }
    }
/*struct Node * Search(int key){
    struct Node *t=root;
    while(t!=NULL){
        if(key==t->data)
            return t;
        else if(key<t->data)
            t=t->lchild;
        else
            t=t->rchild;
    }
    return NULL;
}
*/
struct Node *RInsert(struct Node *p,int key){
    struct Node *t=NULL;
    if(p==NULL){
        t=(struct Node *)malloc(sizeof(struct Node));
        t->data=key;
        t->lchild=t->rchild=NULL;
        return t;
    }
        if(key<p->data)
            p->lchild=RInsert(p->lchild,key);
        else if(key>p->data)
            p->rchild=RInsert(p->rchild,key);

        return p;
}
int Height(struct Node *p){
    int x,y;
    if(p==NULL) return 0;
    x=Height(p->lchild);
    y=Height(p->rchild);
    return x>y?x+1:y+1;
}
struct Node *InSucc(struct Node *p){
    while(p && p->lchild!=NULL)
        p=p->lchild;
    return p;
}
struct Node *InPre(struct Node *p){
    while(p && p->rchild!=NULL)
        p=p->rchild;
    return p;
}
//BST Delete from a Element
struct Node *Delete(struct Node *p,int key){
    struct Node *q;
    if(p==NULL)
        return NULL;
    if(p->lchild==NULL && p->rchild==NULL){
        if(p==root)
            root=NULL;
        free(p);
            return NULL;
    }
    if(key<p->data)
        p->lchild=Delete(p->lchild,key);
    else if(key>p->data)
        p->rchild=Delete(p->rchild,key);
    else
        {
            if(Height(p->lchild)>Height(p->rchild)){
                q=InPre(p->lchild);
                p->data=q->data;
                p->lchild=Delete(p->lchild,q->data);
            }
            else{
                q=InSucc(p->rchild);
                p->data=q->data;
                p->rchild=Delete(p->rchild,q->data);
            }
        }
        return p;
}

int main(){

    printf("Iterative BST: \n");
    Insert(10);
    Insert(5);
    Insert(20);
    Insert(8);
    Insert(30);
    Inorder(root);
    printf("\nAfter Iterative BST Delete Element:  ");
    Delete(root,5);
    Inorder(root);
    printf("\n");


    printf("Recursive BST: \n");
    root=RInsert(root,10);
    RInsert(root,5);
    RInsert(root,20);
    RInsert(root,8);
    RInsert(root,30);
    Inorder(root);
    Delete(root,20);
    printf("\nAfter Recursive BST Delete Element:  ");
    Inorder(root);
    printf("\n");
    getch();
    return 0;
}
```

Input and Output Section:

```
Iterative BST:
5 8 10 20 30
After Iterative BST Delete Element:  8 10 20 30
```

```
Recursive BST:
5 8 10 20 30
After Recursive BST Delete Element:  5 8 10 30
```

13. ***WAP to create Binary Tree and display its pre-order, post-order and in-order traversals Recursively.***

Program:

```
#include <stdio.h>
#include <stdlib.h>
#include "Queue.h"
#include<conio.h>
struct Node *root=NULL;
void Treecreate()
{
 struct Node *p,*t;
 int x;
 struct Queue q;
 create(&q,100);

 printf("Enter root value ");
 scanf("%d",&x);
 root=(struct Node *)malloc(sizeof(struct Node));
 root->data=x;
 root->lchild=root->rchild=NULL;
 enqueue(&q,root);

 while(!isEmpty(q))
 {
 p=dequeue(&q);
 printf("enter left child of %d ",p->data);
 scanf("%d",&x);
 if(x!=-1)
 {
 t=(struct Node *)malloc(sizeof(struct Node));
 t->data=x;
 t->lchild=t->rchild=NULL;
 p->lchild=t;
 enqueue(&q,t);
 }
 printf("enter right child of %d ",p->data);
 scanf("%d",&x);
 if(x!=-1)
 {
 t=(struct Node *)malloc(sizeof(struct
Node));
 t->data=x;
 t->lchild=t->rchild=NULL;
 p->rchild=t;
 enqueue(&q,t);
 }
 }
}
void Preorder(struct Node *p)
{
 if(p)
 {
 printf("%d ",p->data);
 Preorder(p->lchild);
 Preorder(p->rchild);
 }
}
void Inorder(struct Node *p)
{
 if(p)
 {
 Inorder(p->lchild);
 printf("%d ",p->data);
 Inorder(p->rchild);
 }
}
void Postorder(struct Node *p)
{
 if(p)
 {
 Postorder(p->lchild);
 Postorder(p->rchild);
 printf("%d ",p->data);
 }
}
int main()
{
 Treecreate();
 printf(" \nIn order ");
 Inorder(root);
 printf(" \nPre order ");
 Preorder(root);
 printf("\nPost Order ");
 Postorder(root);
 getch();
 return 0;
}
```

Queue Header File

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
 struct Node *lchild;
 int data;
 struct Node *rchild;
};
struct Queue
{
 int size;
 int front;
 int rear;
 struct Node **Q;
};
void create(struct Queue *q,int size)
{
 q->size=size;
 q->front=q->rear=0;
 q->Q=(struct Node **)malloc(q->size*sizeof(struct
Node *));
}
void enqueue(struct Queue *q,struct Node *x)
{
 if((q->rear+1)%q->size==q->front)
 printf("Queue is Full");
 else
 {
 q->rear=(q->rear+1)%q->size;
 q->Q[q->rear]=x;
 }
}
struct Node * dequeue(struct Queue *q)
{
 struct Node* x=NULL;

 if(q->front==q->rear)
 printf("Queue is Empty\n");
 else
 {
 q->front=(q->front+1)%q->size;
 x=q->Q[q->front];
 }
 return x;
}
int isEmpty(struct Queue q)
{
 return q.front==q.rear;
}
```

Input and Output Section:

```
Enter root value 10
enter left child of 10 20
enter right child of 10 30
enter left child of 20 40
enter right child of 20 50
enter left child of 30 60
enter right child of 30 70
enter left child of 40 80
enter right child of 40 90
enter left child of 50 -1
enter right child of 50 -1
enter left child of 60 -1
enter right child of 60 -1
enter left child of 70 -1
enter right child of 70 -1
enter left child of 80 -1
enter right child of 80 -1
enter left child of 90 -1
enter right child of 90 -1

In order 80 40 90 20 50 10 60 30 70
Pre order 10 20 40 80 90 50 30 60 70
Post Order 80 90 40 50 20 60 70 30 10
```

14. ***WAP to create Binary Tree and display its pre-order, post-order and in-order traversals Iteratively.***

Program:

```
#include <stdio.h>
#include <stdlib.h>
#include "Stack.h"
#include "Queue.h"
#include<conio.h>
struct Node *root=NULL;
void Treecreate()
{
 struct Node *p,*t;
 int x;
 struct Queue q;
 create(&q,100);

 printf("Enter root value ");
 scanf("%d",&x);
 root=(struct Node *)malloc(sizeof(struct Node));
 root->data=x;
 root->lchild=root->rchild=NULL;
 enqueue(&q,root);

 while(!isEmpty(q))
 {
 p=dequeue(&q);
 printf("enter left child of %d ",p->data);
 scanf("%d",&x);
 if(x!=-1)
 {
 t=(struct Node *)malloc(sizeof(struct Node));
 t->data=x;
 t->lchild=t->rchild=NULL;
 p->lchild=t;
 enqueue(&q,t);
 }
 printf("enter right child of %d ",p->data);
 scanf("%d",&x);
 if(x!=-1)
 {
 t=(struct Node *)malloc(sizeof(struct
Node));
 t->data=x;
 t->lchild=t->rchild=NULL;
 p->rchild=t;
 enqueue(&q,t);
 }
 }
}
void IPreorder(struct Node *p){
        struct Stack stk;
        Stackcreate(&stk,100);
        while(p || !isEmpty(stk))
        {
                if(p){
                        printf("%d ",p->data);
                        push(&stk,p);
                        p=p->lchild;
                }
                else{
                        p=pop(&stk);
                        p=p->rchild;
                }
        }
```

```
}
void IInorder(struct Node *p){
        struct Stack stk;
        Stackcreate(&stk,100);
        while(p || !isEmpty(stk))
        {
                if(p){

                        push(&stk,p);
                        p=p->lchild;
                }
                else{
                        p=pop(&stk);
                        printf("%d ",p->data);
                        p=p->rchild;
                }
        }
}

int main()
{
 Treecreate();
 printf(" \nIn order ");
 IInorder(root);
 printf(" \nPre order ");
 IPreorder(root);

 getch();
 return 0;
}
```

Stack Header File

```
#include <stdio.h>
#include <stdlib.h>
//#include "Queue.h"
struct Stack
{
 int size;
 int top;
 struct Node **S;
};
void Stackcreate(struct Stack *st,int size)
{
 st->size=size;
 st->top=-1;
 st->S=(struct Node **)malloc(st->size*sizeof(struct Node *));
}
```

```
void push(struct Stack *st,struct Node *x)
{
 if(st->top==st->size-1)
 printf("Stack overflow\n");
 else
 {
 st->top++;
 st->S[st->top]=x;
 }

}
struct Node *pop(struct Stack *st)
{
 struct Node *x=NULL;
 if(st->top==-1)
 printf("Stack Underflow\n");
 else
 {
 x=st->S[st->top--];
 }
 return x;
}

int isEmpty(struct Stack st)
{
 if(st.top==-1)
          return 1;
 return 0;
}
int isFull(struct Stack st)
{
 return st.top==st.size-1;
}
```

Input and Output Section:

```
Enter root value 10
enter left child of 10 20
enter right child of 10 30
enter left child of 20 40
enter right child of 20 50
enter left child of 30 60
enter right child of 30 70
enter left child of 40 -1
enter right child of 40 -1
enter left child of 50 -1
enter right child of 50 -1
enter left child of 60 -1
enter right child of 60 -1
enter left child of 70 -1
enter right child of 70 -1

In order 40 20 50 10 60 30 70
Pre order 10 20 40 50 30 60 70
```

15.WAP to implement Diagonal Matrix using one-dimensional array.

Program:

```
#include<stdio.h>
#include<conio.h>
struct Matrix
{
 int A[10];
 int n;
};
void Set(struct Matrix *m,int i,int j,int x)
{
 if(i==j)
 m->A[i-1]=x;

}
int Get(struct Matrix m,int i,int j)
{
 if(i==j)
 return m.A[i-1];
 else
 return 0;
}
void Display(struct Matrix m)
{
 int i,j;
 for(i=0;i<m.n;i++)
 {
    for(j=0;j<m.n;j++)
    {
    if(i==j)
        printf("%d ",m.A[i]);
    else
        printf("0 ");
 }
    printf("\n");
 }
}
int main()
{
```

```
//clrscr();
 struct Matrix m;
 m.n=4;

 Set(&m,1,1,5);
 Set(&m,2,2,8);
 Set(&m,3,3,9);
 Set(&m,4,4,12);
// printf("%d \n",Get(m,3,3));
 Display(m);
 getch();
 return 0;
}
```

Input and Output section:

```
5 0 0 0
0 8 0 0
0 0 9 0
0 0 0 12
```

16.WAP to implement Lower Triangular Matrix using one-dimensional array.

Program:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct Matrix
{
 int *A;
 int n;
};
void Set(struct Matrix *m,int i,int j,int x)
{
 if(i>=j)
 m->A[m->n*(j-1)+(j-2)*(j-1)/2+i-j]=x;
}
int Get(struct Matrix m,int i,int j)
{
 if(i>=j)
 return m.A[m.n*(j-1)+(j-2)*(j-1)/2+i-j];
 else
 return 0;
}
void Display(struct Matrix m)
{
 int i,j;
 for(i=1;i<=m.n;i++)
 {
 for(j=1;j<=m.n;j++)
 {
 if(i>=j)
 printf("%d  ",m.A[m.n*(j-1)+(j-2)*(j-1)/2+i-j]);
 else
 printf("0   ");
 }
 printf("\n");
 }
}
int main()
{
 struct Matrix m;
 int i,j,x;
//clrscr();
 printf("Enter Dimension");
 scanf("%d",&m.n);
 m.A=(int *)malloc(m.n*(m.n+1)/2*sizeof(int));
 printf("enter all elements");
 for(i=1;i<=m.n;i++)
 {
 for(j=1;j<=m.n;j++)
 {
 scanf("%d",&x);
 Set(&m,i,j,x);
 }
 }
 printf("\n\n");
 Display(m);

getch();
 return 0;
}
```

Input and Output Section:

```
Enter Dimension
4
enter all elements
10 20 30 40
11 22 33 44
```

```
12 24 36 48
13 26 39 52

Lower Triangular Matrix are:
10   0    0   0
11  22   0    0
12  24  36   0
13  26  39  52
```

17.WAP to implement Upper Triangular Matrix using one-dimensional array.

Program:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct Matrix
{
 int *A;
 int n;
};
void Set(struct Matrix *m,int i,int j,int x)
{
 if(i>=j)
 m->A[m->n*(j-1)+(j-2)*(j-1)/2+i-j]=x;
}
int Get(struct Matrix m,int i,int j)
{
 if(i>=j)
 return m.A[m.n*(j-1)+(j-2)*(j-1)/2+i-j];
 else
 return 0;
}
void Display(struct Matrix m)
{
 int i,j;
 for(i=1;i<=m.n;i++)
 {
 for(j=1;j<=m.n;j++)
 {
 if(i<=j)
 printf("%d  ",m.A[m.n*(i-1)+(i-2)*(i-1)/2+j-i]);
 else
 printf("0   ");
 }
 printf("\n");
 }
}
```

```
int main()
{
 struct Matrix m;
 int i,j,x;
//clrscr();
 printf("Enter Dimension");
 scanf("%d",&m.n);
 m.A=(int *)malloc(m.n*(m.n+1)/2*sizeof(int));
 printf("enter all elements");
 for(i=1;i<=m.n;i++)
 {
 for(j=1;j<=m.n;j++)
 {
 scanf("%d",&x);
 Set(&m,i,j,x);
 }
 }
 printf("\n");
 printf("Upper Triangular Matrix are: \n");
 Display(m);

getch();
 return 0;
}
```

Input and Output Section:

```
10 20 33 44
12 11 23 45
12 34 56 78
13 23 45 23

Upper Triangular Matrix are:
10  12  12  13
0   11  34  23
0   0   56  45
0   0   0   23
```

PYTHON PROGRAMMING
LAB MANUAL
(PAPER CODE: BCASEC03P)

Introduction to Python:

What does it mean by language?

Human beings use languages for communication. We use the languages like English, Hindi, French, Bengali. These are the natural languages. These are called as natural languages. So, we use these languages for communication.

Likewise, machines also have their languages. This is our computer system that is also having its own language. And that language we call it as machine language. And this machine works in number system that is binary number system that is in the form of zeros and ones only.

What is programming?

Programming means a step-by-step procedure for solving any problem.

So a language that is used for writing the procedures which machines can understand and execute them and machines do the processing so that languages are nothing but programming languages.

Types of Programming Languages

- Low-Level Language
  - Machine Language
    1. The machine language that is in the form of only zeros and ones.
    2. Humans cannot understand this language that is zeros and ones easily.
  - Assembly Language
- High-Level Language
  - General Purpose
  - Special Purpose
  - Scripting

  Then later people thought that let us use English like languages for writing the programs such languages are called as high-level languages.

  We write the programs using C language, then we compile it and the compiler will convert this C language program into machine language code.

  - General Purpose

    The language can be used for writing various types of programs like desktop application, web application, cloud application, enterprise application, mobile application, telecommunication application system programming so on.

But a language which is used for writing any type of application is called as general-purpose language.

he languages as C, C++, Java, C#, python. These are all high-level languages which are useful for various purpose of programming

- Special Purpose

  There are some special purpose languages, like there are some languages only for web development, there are some languages that are only for mobile development, like Swift is the language or a Android language. There are some languages only for gaming purpose, so such languages are called as special purpose languages.

  Example: JS, Android

- Scripting

  They are very easy to write means they will not have very strict rules of programming. It is easy for the programmer to write down the code, so those are usually called as scripting languages.

  Python is a general-purpose language, but it is more like scripting language, so it's easy to learn and develop the application.

High-Level Language

- ***Compiler Based (C, C++)***
- ***Interpreter Based (JS, PHP)***
- ***Hybrid (Java, C#, Python, .Net)***

The high-level languages are either compiler based or interpreter based or hybrid.

We write the programs using high level language. Machine cannot understand that language, so this high-level language program code should be translated into machine language code.

Who does the translation?

So, translation is done either by compiler or interpreter. So, it's done either by compiler or interpreter. There are two methods of translation.

So, some programming languages uses only compiler some programming languages, uses only interpreter, and some uses both. Some are hybrid languages, which uses both compiler and interpreter.

Compiler VS Interpreter

Compiler:

i) Compiler takes the complete source program and generates complete machine code program. It takes a file and it gives a new file that contains a complete machine code.
ii) If there are no errors in your program, then only this translation is done into machine code.
iii) Only one time translation is done.

Interpreter:

If these are the lines of JavaScript code, the interpreter will translate one line into machine code. It will generate the machine code just for one line and also get it executed at that time. Only then second line translate into machine code, get it executed next line, convert it into machine code executed each line it will take and convert into machine code and also it will get executed. So this is how interpreter works.

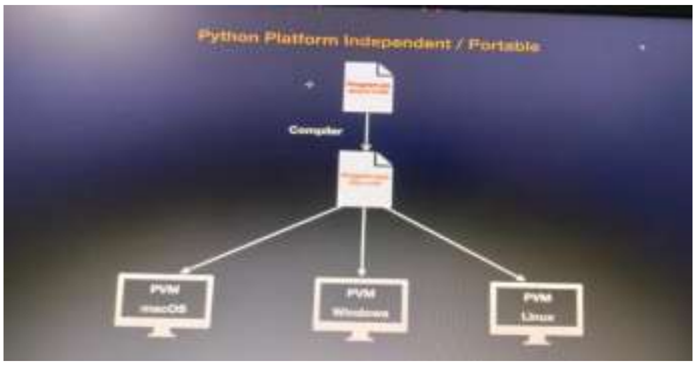
Difference between Compiler and Interpreter

i) See, the compiler was not executing it. Compiler was giving complete machine code. You execute whenever you want, but interpreter will directly execute it on the machine.
ii) Interpreter will convert each line into machine code and getting executed and separate file is not generated but compiler will generate two separate files.
iii) Compiler based languages are faster than interpreter-based languages because in interpreter-based languages, every time translation is done, program is translated into machine code and executed. So, we say it is done line by line. Sometime the entire code is converted also, but always the translation is done.
iv) Interpreter languages are easier to write and run because they run inside the runtime environment, which is more comfortable for the programmer to develop such programs by taking the support of runtime environment.

How a Python Program Runs?

It will compile this source code into intermediate code called bytecode. it converts into bytecode and it generates one hidden file. Actually, it's cannot be seen. So, the file name is program (.pyc) so it was (.pyc) file but now it is compiled file.

When the compilation is done. Compiler will check if there are any errors in the program. If there are no errors, then it will convert into machine code. Interpreter will not check for the errors because already the bytecode is error free. So, what it has to do just it has to translate this bytecode into machine code that is machine language. So, Python's interpreter will translate bytecode into machine code and execute. When you run a program, it gets compiled and then interpreted and executed by Python Virtual Machine that is PVM.



Python Features

Python provides many useful features which make it popular and valuable from the other programming languages. It supports object-oriented programming, procedural programming approaches and provides dynamic memory allocation. We have listed below a few essential features.

1) Easy to Learn and Use

Python is easy to learn as compared to other programming languages. Its syntax is straightforward and much the same as the English language. There is no use of the semicolon or curly-bracket, the indentation defines the code block. It is the recommended programming language for beginners.

2) Expressive Language

Python can perform complex tasks using a few lines of code. A simple example, the hello world program you simply type **print("Hello World")**. It will take only one line to execute, while Java or C takes multiple lines.

3) Interpreted Language

Python is an interpreted language; it means the Python program is executed one line at a time. The advantage of being interpreted language, it makes debugging easy and portable.

4) Cross-platform Language

Python can run equally on different platforms such as Windows, Linux, UNIX, and Macintosh, etc. So, we can say that Python is a portable language. It enables programmers to develop the software for several competing platforms by writing a program only once.

5) Free and Open Source

Python is freely available for everyone. It is freely available on its official website www.python.org. It has a large community across the world that is dedicatedly working towards make new python modules and functions. Anyone can contribute to the Python community. The open-source means, "Anyone can download its source code without paying any penny."

6) Object-Oriented Language

Python supports object-oriented language and concepts of classes and objects come into existence. It supports inheritance, polymorphism, and encapsulation, etc. The object-oriented procedure helps to programmer to write reusable code and develop applications in less code.

7) Extensible

It implies that other languages such as C/C++ can be used to compile the code and thus it can be used further in our Python code. It converts the program into byte code, and any platform can use that byte code.

8) Large Standard Library

It provides a vast range of libraries for the various fields such as machine learning, web developer, and also for the scripting. There are various machine learning libraries, such as Tensor flow, Pandas, Numpy, Keras, and Pytorch, etc. Django, flask, pyramids are the popular framework for Python web development.

9) GUI Programming Support

Graphical User Interface is used for the developing Desktop application. PyQT5, Tkinter, Kivy are the libraries which are used for developing the web application.

10) Integrated

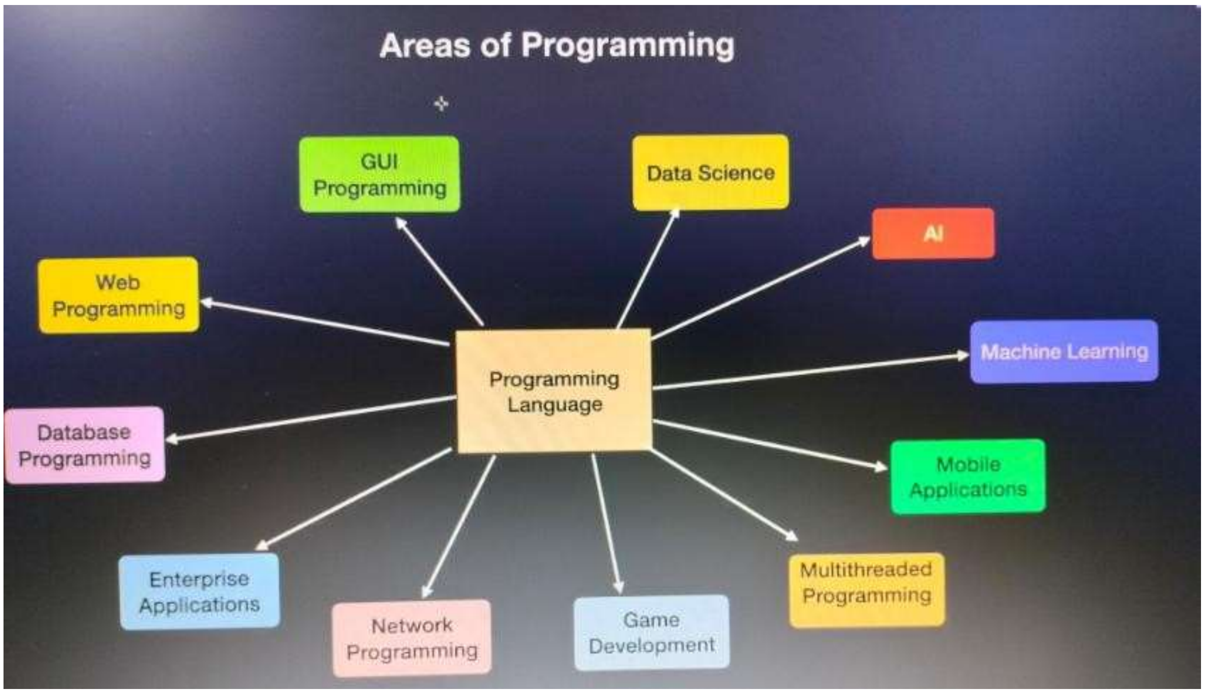
It can be easily integrated with languages like C, C++, and JAVA, etc. Python runs code line by line like C, C++ Java. It makes easy to debug the code.

11. Embeddable

The code of the other programming language can use in the Python source code. We can use Python source code in another programming language as well. It can embed other language into our code.

12. Dynamic Memory Allocation

In Python, we don't need to specify the data-type of the variable. When we assign some value to the variable, it automatically allocates the memory to the variable at run time. Suppose we are assigned integer value 15 to **x,** then we don't need to write **int x = 15.** Just write x = 15.



What is the Python Shell?

Python is an interpreted language. This means that the Python interpreter reads a line of code, executes that line, then repeats this process if there are no errors.

The Python Shell gives you a command line interface you can use to specify commands directly to the Python interpreter in an interactive manner.

Variables

• ***Program*** is a set of instructions written in a programming language to perform tasks.

• A program has two important elements.

1. Data: It is an information such as facts and numbers upon which operations are performed.

2. Instructions: A set of steps to perform operations upon data.

• Every program needs data and instructions irrespective of the programming language used.

• Instructions without data is useless.

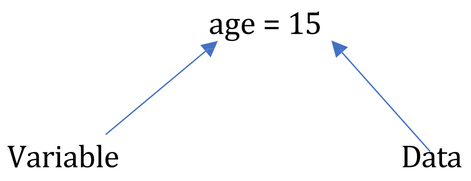
What are Variables?

• Data in a program is handled by variables.

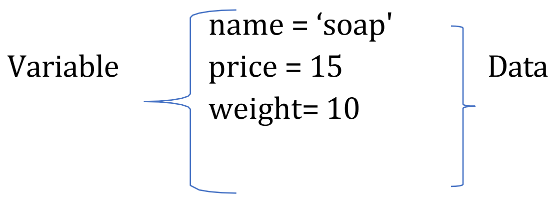
• Variables are the names given to the data stored in a memory location.

• Variable is the identifier, reference to the data.

Example.

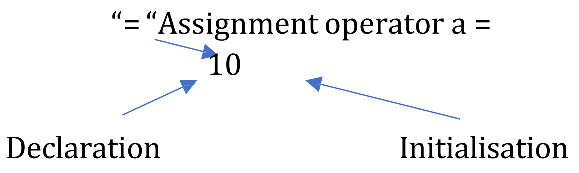


Another example



• ***Python variable declaration:***

- Unlike other programming languages, in python variable is declared and initialised at the same time.



- Declaring a variable is giving the name to the data.
- Initialisation is storing the value in a variable.

• In python declaring a variable ***without initialising or say without the***

data is not allowed.

b= 12.5 (declaration and initialisation) c
(only declaration, is not allowed)

• Unlike other programming languages like c, c++, java, in python declaration of data type for a variable is not required.

• The value assigned to the variable determines the data type of the variable.

• ***Declaration and initialisation of multiple variables:***

- In python multiple variables can be declared and initialised in a single statement.

Example:

a,b,c = 5,10,15

Here:

a=5,b=10,c=15 , values will be stored in the respective variables in the order they are declared.

name, price,weight='soap',10,15 Here:

name='soap', price=10, weight=15

Example: Assigning same value to multiple variables: x, y, z=1,1,1 or

x=y=z=1

Python- Dynamically Typed Language

• When a variable is declared, it must be initialised as well

• Variables do not have a specific data type; its type depends on the value assigned to it.

Example:

X = 25 - integer type X =
13.75 - float type X = 'A'
- string type

• For the above example we can say that, Python is a dynamically typed language meaning while defining a variable we don't have to give its data type explicitly like in any other languages such as c++ , Java

• To know what type of data is provided to the variable we use function type(x) Example:

```
>>>X=25
>>>type(X)
<class 'int'>
```

• The above result is <class ‘int’> meaning the given data type belongs to a class of integer
• As we know everything in python is an object and every object will have its own class
• The class is decided on the type of value assigned to the variable.

Rules for Declaring a variable

• Although we can use any name for declaring a variable but we must follow certain rules, so that it is easy to understand by us and others too

• The variable names can be taken as:

```
a= 10
b = 12.5
c= ‘ John ‘
```

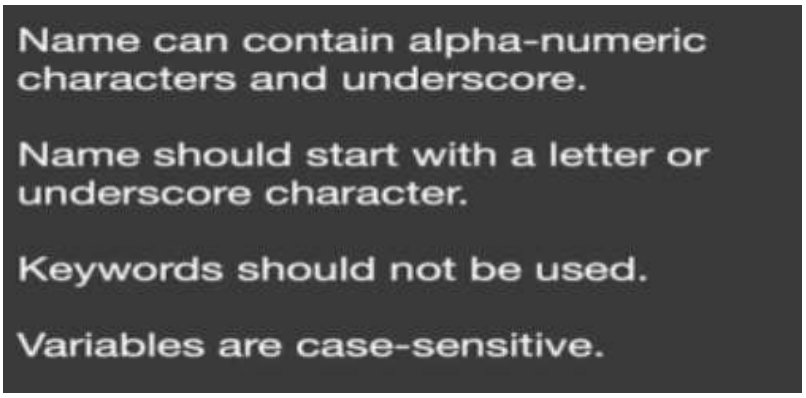
• Even though the above declaration is correct but we cannot extract the exact meaning of the variables.

• To make the above variables more meaningful we can declare them as:

```
roll_no = 10
price = 12.5
cust_name= ‘ John ‘
```

• Now the above variables are more descriptive and understandable.

• The rules for declaring a variable are as follows


Name can contain alpha-numeric characters and underscore.

Name should start with a letter or underscore character.

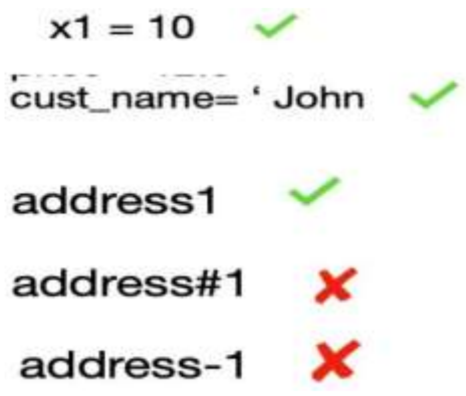
Keywords should not be used.

Variables are case-sensitive.

• ***The first rule says that we can mix alphabet and numbers while declaring the variables, we can even use an underscore.***

• However, we cannot use any special symbol like $, &, @, #, - etc…

• ***Example:***

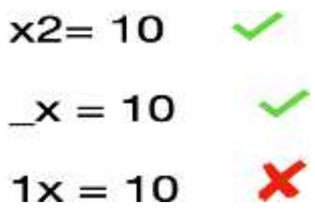


```
x1 = 10          ✓
cust_name= ' John   ✓
address1         ✓
address#1        ✗
address-1        ✗
```

• ***The second rule says that the variable must start with a letter or underscore character only.***

• Although we can use alphabet and numbers in a variable but numbers cannot be used at the beginning of the variable name.

• Example:



```
x2= 10    ✓
_x = 10   ✓
1x = 10   ✗
```

• ***The third rule states we cannot use keywords to declare a variable.***

• In python program we use words, numbers or symbols.

• Here the words can be categorised into two, that is identifiers and keywords.

Identifiers: These are the words given by the programmer, It is used for identifying something that is define exclusively by the programmer.

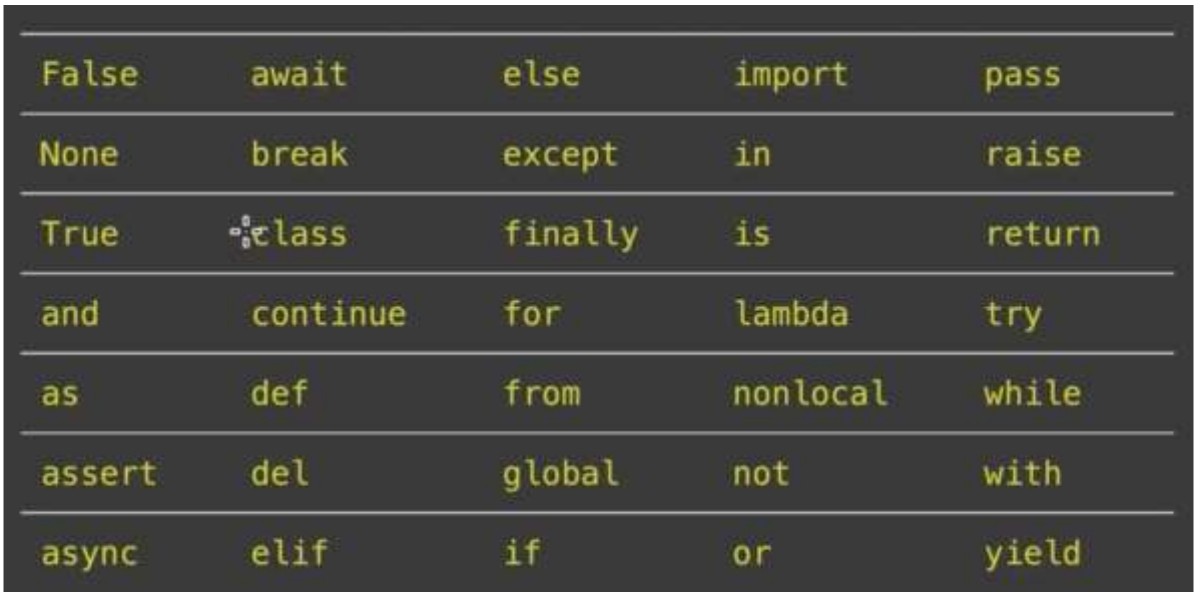
price = 12.5

• Variable name, function name, class name and even module name are all identifiers.

Keywords: The words which are predefined in the language are called keywords or reserved words.

• The list of the keyword is given below.

• These word cannot be used while declaring the variable.



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

• ***The last rule states that variables are case sensitive.***

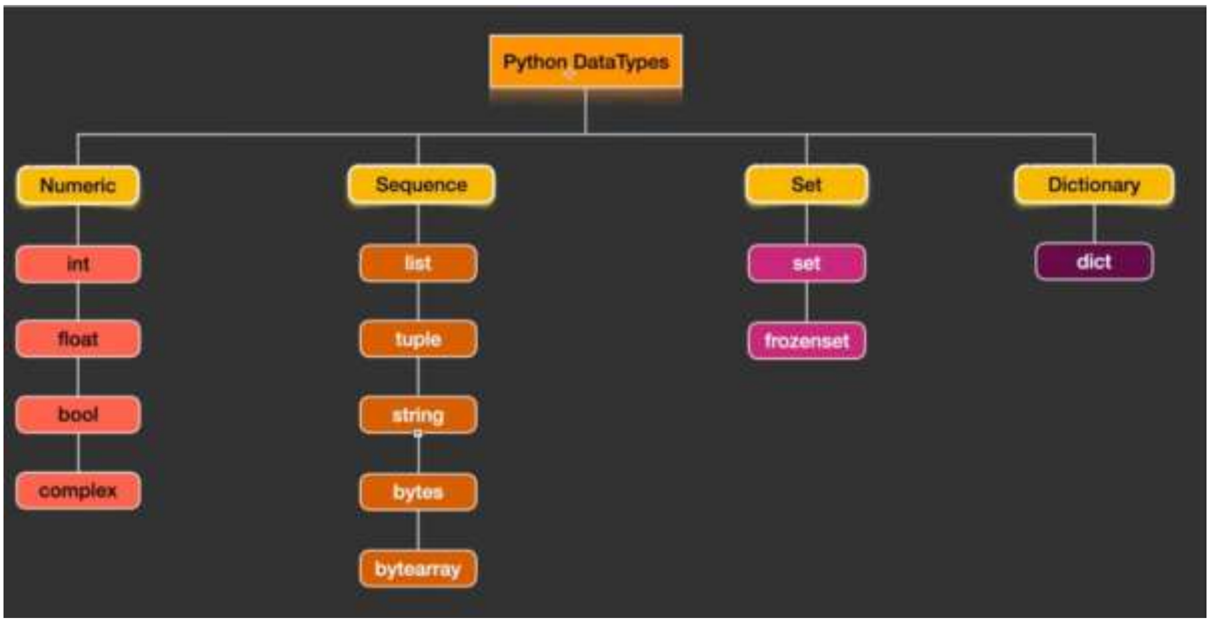
A=10

a=10

• The above variables are not same, 'A' is not same as 'a'.

Python Data Types

• Python is a very rich language in terms of data types as it provides various data types as shown below



• The example of numeric data types are

Int:

x = 10

Float:

y = 19.5

bool:

z = TRUE

Complex:

a = c + ib

• The ***sequence*** data type holds the collection of values.

• The set type of data is also collection of values, however the difference between set and sequence is, In Sequence data type, all data item have its own index where as in set there is no index value for data elements.

Example of Sequence data type is:

List: It is a collection of items/values in a single variable. Its mutable. The values of a list can be modified.

[2,3,4,75,7]

Tuple: It is a collection of values. Tuples are immutable, they cannot be modified.

(4,6,8,2,4,6)

String: It is a collection of alphabets, numbers, characters or symbols. "python", 'hello'

Byte:

- It contains only 1 byte of data
- It allows value from 0 -255

Bytearray:

It contains array of byte data

Example of set data type is:

Set: The set type of data is also collection of values. Set there is no index value for data elements.

{ 2,4,8,0,12 }

Frozen set: It is also like set but frozen mean the data cannot be modified.

Dictionary: The Dictionary data type is the collection of key-value pair.

• It is useful for storing and retrieving the data much

faster Example:

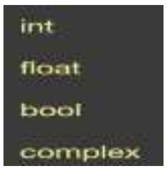
name : "John"

rollno : 23 dept

: 'cs'

Numeric Data Type

- The numeric data type that are available in python are:



```
int
float
bool
complex
```

- **Integer (int)** is the numeric data without any decimal point.
- It can contain, both positive and negative numbers.
- **Example:**

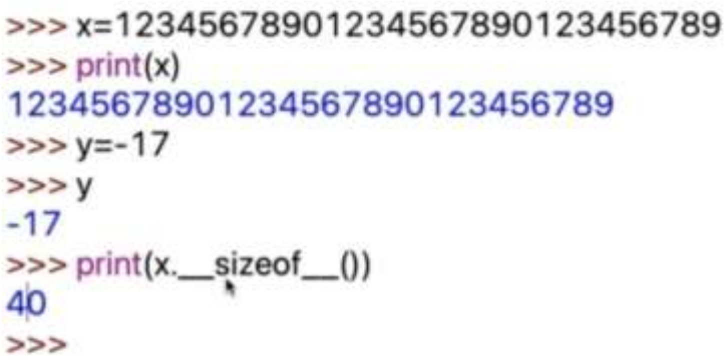
  a = 125
  b = 2164
  c = -17
  x = 129734864

- In integers there is no size limit or range to an integer

  value. x =

  12345678901234567890123456789

- Not only in integers but there is no fixed size memory taken by any datatype in python.
- Suppose you want to know the size of any value then we use 'sizeof' method, see the example below:
- x._sizeof_()



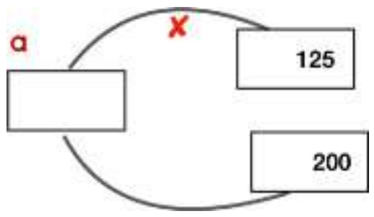
```
>>> x=12345678901234567890123456789
>>> print(x)
12345678901234567890123456789
>>> y=-17
>>> y
-17
>>> print(x.__sizeof__())
40
>>>
```

✓ Let's consider an example and see what happen when we assign a variable two different values.

✓ a = 125 and a = 200.

- ✓ Here the first **a** is the reference to the value 125 and it is available in the memory.
- ✓ when we change the value of **a** from 125 to 200, then **a** will point/refer to the new object 200.
- ✓ Here the actual value that is 125 is not changing but a new value is created and now **a** point to this new value, therefore we can say that values are immutable or it cannot be changed.

Immutable



- Now the question is where is 125 in memory? The value 125 will be garbage collected by PVM (python virtual machine), and it will be deleted from the memory.

Floating point:

- Any numeric value with a decimal point is called a floating-point number.
- They can either be positive or negative numbers.

Example

a = 13.25

b = -17

- Floating point number can also be in written scientific format as well,

  12.59 0 . 1259 * 100

  0 . 1259 * 10^2

  0 . 1259E2 (floating point representation)

- Float datatype is immutable.

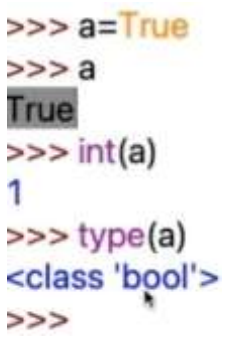
Numeric Data Type (bool & complex)

- Boolean and complex are numeric data type.
- **Boolean** data is logical data.
- Boolean data types are used in writing conditions using relational and logical operators.
- The result of a Boolean is either True or False.

  True — 1
  False — 0

  The value of True is 1and False is 0.

- Example:

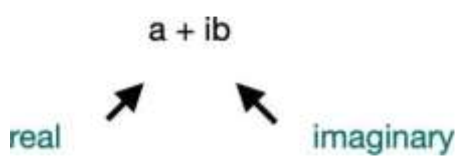


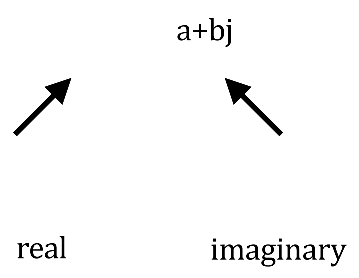
```
>>> a=True
>>> a
True
>>> int(a)
1
>>> type(a)
<class 'bool'>
>>>
```

Complex Numbers

- Complex numbers have real part and an imaginary part.
- Complex numbers are mostly used in mathematical operations.

Mathematical representation: Python representation:





- Here **i** and **j** are the pre- defined constants.
- where i= √-1, or j= √-1
- In mathematics we know that the square root of negative numbers is undefined,

- let's take an example to understand this:

$$25+\sqrt{-9}$$
$$25+\sqrt{-1}*9$$
$$25+\sqrt{-1}*\sqrt{9}$$
$$25+\sqrt{-1}\ \ 3$$
$$25+i3$$

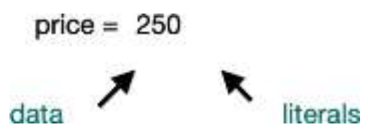
- Complex Data types can be used when an application is being developed in Pythoninvolving complex numbers.
- We can create complex numbers using integer, float value and even using functions.

```
 X = 3 + 5j                  // Integer
x = 3.5 + 5.9j               // float
X = complex( 3.5, 5.9)            // function
```

- Operators like + , - , * , / can be used to perform operations on complex numbers.

Literals or Constants

- Literals are the direct values / data written in the program



- They are 2 methods a variable can have value
  - By directly writing the values in the program
  - By taking input from the user

- We can store any type of data in the literals

- Let's understand this with an

example Example For taking ***integer literals***:

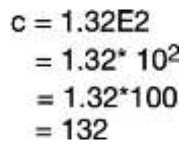
a = 125 // *basic literal*
a = 12520 // *this is a big number and to store it as literal we need to give an underscore to it as shown below*
a = 12_520

- In python we use _ (under course) so we can easily understand number system

Example For taking ***Float literals:***

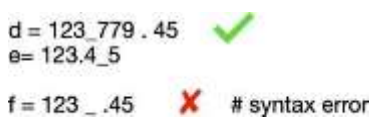
a= 12.59 // ***basic float literal***
b=13 // in float we need to have decimal value so, we take this value as b = 13. 0

- Float number can also be written in scientific notation



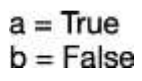
```
c = 1.32E2
  = 1.32* 10^2
  = 1.32*100
  = 132
```

- There are some specifications while giving underscore in float value that are the _ cannot begiven before and after the decimal point



```
d = 123_779 . 45   ✓
e= 123.4_5

f = 123 _ .45    ✗   # syntax error
```

- When you are directly writing True / False values in your program we can use ***Bool Literals***



```
a = True
b = False
```

- The T and F of True and False must be capital otherwise it will given syntax error

complex literal:
The complex number literals are as follows; it can have an _ as well a =
5+4j
a = 5_1 + 4_3j

string literal:
The string literals are as follows

a = ' John '
b = " John"

Literals

Decimal: 0 - 9 its also said as base 10
Binary: its of 2 digits is said as base 2
Octal: 0 - 7 is also said as base 8
Hexadecimal: 16 digits 0 - 9 . A - F base
16

int: Both positive and negative numbers including 0

```
a = 10
b = 0b1010          # bin
```

Both a and b are same

```
a = 0o12        # octal
a = 0xA         # hexa
```

Both are 10

float: number with both integer and fractional parts. a =
0b125
f = 0b111 . 0b11 ✗

complex:

```
c = 5 + 6 j
c = 0b101+6j          ✓
```

We can give a real part but not imaginary part c =
b101+b101 j ✗

str
: price = input(" enter price ")
Output : enter price 0b101 '
0b101,

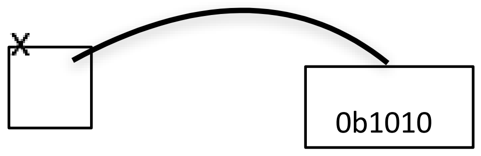
- But if we want in int type so we use type

casting price = int (input (" enter price ") , 2)
# base 2 cause we are entering the binary form. enter
price : 0b101
Output : 5

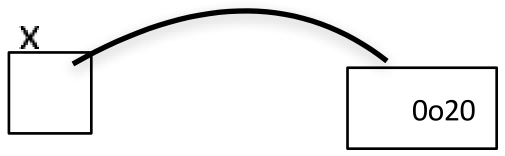
Base Conversions

a = 10	bin ()
b = 16	oct()
c = 235	hex()

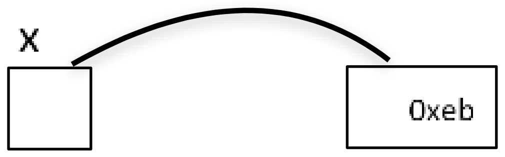
x= bin(a)



x = oct (b)



X = hex(c)



Suppose we want to know the binary equivalent of any value we can call the function
y = bin (b)
output: ‘ 0b10000’

Type conversion

- It will convert one datatype to another datatype

1) ***Implicit***
2) ***Explicit***

Implicit: In Implicit type conversion of data types, the Python interpreter automatically converts one data type to another without any user involvement.

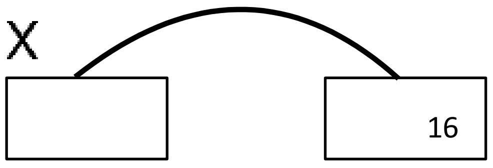
Explicit: The programmer has to convert and programmer have to mention int ()

float() complex() bool() str()

Int: It will convert float to int type

    f=16.59 ✓

    x = int (f)



The output of x will be 16

- Lets try with a

    string s1 =

    'John' ✗

x = int (s1) **# error**

Lets give a try with int given in string s2

    = '123' ✓

    x = int (s2)

Its possible and the output will be 123 Lets

take in binary form

    s3 = '0b1111' ✓

    x = int (s3, 2)

    ↑ ↑

    String base value

Output: 15

float: In float we can convert int to float and bool to float

x = float(i) ✓ x =
float (b) ✓ x = float
(c) ✓

Lets try with string

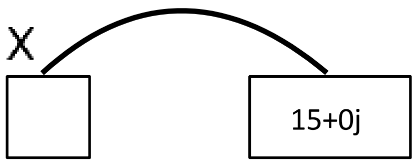
s1 = 'John'
x =float(s1) ✗

s2 = '125'
x = float (s2)✓

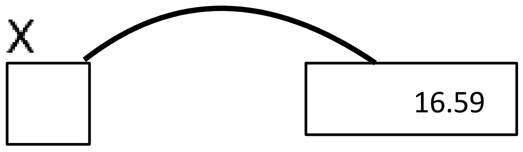
The output: 125.0

complex: It has real part and imaginary part

i = 15
x = complex (i)



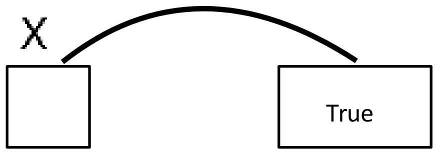
x = complex (f)



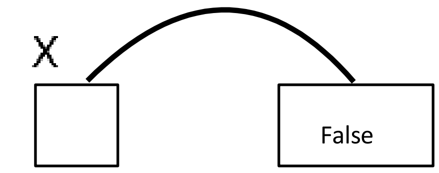
- Same goes for str

s1 = ' John' ✗
x = complex (s1) s2
= '125'
x = complex (s2) ✓

bool: It will convert all datatype to bool type x = bool (i)



x = bool (0 + 0j)



- Same goes for string

x = bool (‘ hi ‘)

Output: True

Operators and Expression

Arithmetic Operator:

- Arithmetic operator is use to perform arithmetic operation like addition, sub, div etc.
- There are 7 arithmetic operators

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division(float)	x / y
%	Modulus	x % y
**	Exponentiation (Power)	x ** y
//	Division(floor)	x // y

a = 15 b = 4
c = a+b output: 19
c = a*b output: 60
c =a-b output: 11

c=a**b

- Now the question is what is ** It is said as

power c= a**b
=(15)^4
= 50,625

Float division (//) :

- If we don't want in float form, we want in int form then we will use float

division C = a // b
= 3

Mod (%) :

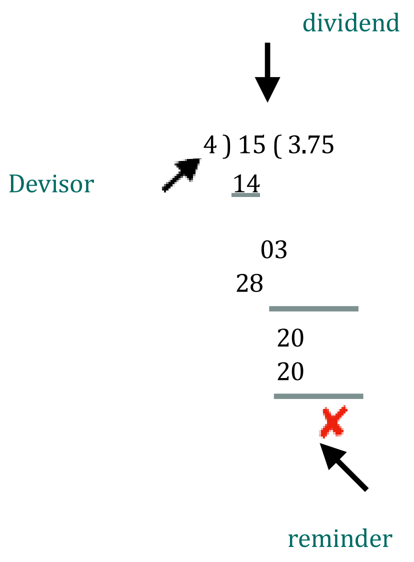
- This will divide. It will not take quotient it will take

remainder a = 15 b = 4
Z = x//y = 3 Z
= x % y = 0

Division (/) :

C = a/b

=3.5



Expressions

- The instruction that we write using operators is called expressions

- c = a + b

# c , a , b are operands #
= , + are operators Lets
take example:

d = 2+3*5 #3*5=15
d = 2+15
d=17

- This happened because of precedence.
- First multiplication takes place because it has higher precedence than addition
- They will execute based on their
  precedence d = 2*3 + 8/2
  = 6 + 4 = 10
- To increase the precedence we use ( )

2+3*5 # first multiplication take place because it has higher precedence than addition 2+15 = 17

By using parentheses

(2+3) * 5 # here addition will take place because parentheses have higher precedence than all

5 * 5 = 25

- What if we are having

d=3*4*5/4 # * , / have same precedence

$\longrightarrow$

So, it will do left to right #first
multiplication than division
d=2**3**2
$\longleftarrow$

It will execute from right to left

Python expressions:

Square:

Area of a square = lb

- In python we write using expressions

The area of the square can be written l * b

Triangle
Area of a triangle = 1/2 bh

In python = 1/2*b*h
Trapezium
Area of a trapezium = a+b/2 h

In python we write using python expressions area= (a+b) / 2*h

Displacement

$(v^2-u^2/2a)$

In python using expressions

(v**2-u**2)/(2*a)

Equal roots

(-b/2a)

In python using expressions (-b/(2*a))

Program Using Expressions.

• The input function gives string type of result.

• So, when other type of values is taken, we should typecast it [i.e., convert str to int etc]

Q) Write a program for finding area of a rectangle.

#The formula for finding area of the rectangle is a = l * b where l is length and b is breath.

```
length = int(input('Enter length of rectangle ')) breadth =
int(input('Enter breadth of a rectangle ')) area = length *
breadth
print('Area is ', area)
```

Q) Write a program for finding area of a Triangle.

```
base = int(input('Enter base ')) height
= int(input('Enter height ')) area =
1/2 * base * height print('area is ',
area)
```

Q) Write a program for finding area of a trapezium.

```
a = float(input('Enter side a '))
b = float(input('Enter side b ')) height =
float(input('Enter height ')) area = 1/2
* (a + b) * height print('area is', area)
```

Q) Write a program for finding a displacement

```
u = float(input('Enter initial velocity ')) v =
float(input('Enter  Final  Velocity  '))  a  =
float(input('Enter Acceleration '))
d  =  (v**2  -  u**2)  /  (2  *  a)
print("Displacement is", d)
```

Q) Write a program to convert km to miles

```
# 1km = 0.621371 miles
    km = float(input('Enter Km'))
    miles = km * 0.621371
    print('Miles =', miles)
```

Q) Write a program to calculate area of a circle

```
import math
radius = int(input("Enter the radius "))
#area = math.pi*radius*radius
area = math.pi*radius**2
print("Area of a circle is: ",area)
```

Q) Write a program to calculate area of a

cuboid Total surface Area of a cuboid
# it has 6 sides
Front and back = 2lh Bottom and top = 2lb Left and right = 2bh Total = 2(lh+lb+bh)

```
length = float(input('Enter Length '))
breadth = float(input('Enter Breadth '))
height = float(input('Enter Height '))
area = 2 * (length * breadth + length * height + breadth * height) print('Total Surface Area is', area)
```

Q) Write a program to finding roots of quadratic equations

eg: ax2 + bx + c = 0
Roots:
r1 = -b + √ b2-4ac / 2a
r2 = -b - √ b2-4ac / 2a
If √ b2-4ac will give negative value than it will become complex number

```
import math

a = int(input('Enter a value '))
b = int(input('Enter b value '))
c = int(input('Enter c value '))
root1 = (-b + math.sqrt(b**2 - 4 * a * c))/(2 * a) root2
= (-b - math.sqrt(b**2 - 4 * a * c))/(2 * a) print('Roots
are  ', root1, root2)
```

Assignment Arithmetic Operator

Operator	***Example***	***Same As***
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

:=	print(x := 3)	x = 3
		print(x)

Arithmetic operator joined with assignment operator.

- suppose we have a variable

a=5

a=a+1

a=5+1

stores 6 in a

- whatever the value is stored results into it.
- It means we want to modify the value of a and increaseit by 1
- The a is assigned two times the value of a is taken andincreased

by 1 So, this type of statements usually used for counting Count= 0

Count= count+1 $\longrightarrow$ this means

1= 0+1

- Count = count+1. This type of statements can be written inshort like count+=1
- a=a+1 instead of writing this elaborated statement we canwrite it in short like a+=1.
- This statement was for addition Now, in same way if we have anything to subtract, we can use subtraction.

n=10

n=n-1

n-=1

this means the same thing n- assigns 1

- if we have a variable p=10 and we want to multiply it witha variable x=5 then we may be assigning p=p*x

p*=x to make the statements short these operations are given

- bitwise operators can also be used with these assignmentoperators.
- If we have two variables a=10, b=14 and want to perform& operation a=a & b

a& =b

if we want to do something in other way
b=a&b this is the statement we can change to b&=a.

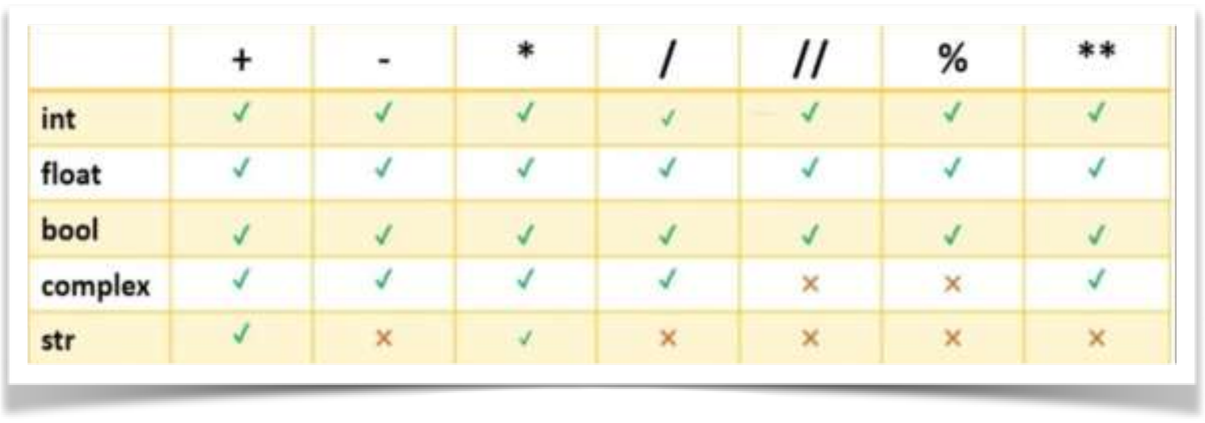
- when want to store the result in a then
- say a=&b this type of statement can be converted to a&=b
- if we want b to store result in b then say b=a & b this typeof statement can be converted to b&=a assignment as wellas bitwise operator can be joined together to make our statement shorter most of the time counting statement is used

```
>>>> a=5         # assigning value 5 to variable
>>>> a=a+1# incrementing by 1
>>>> a
        6        # a becomes 6 after incrementing
>>>>count=0
>>>>count+1              #incremneted by1
         1                # becomes 1
>>>>count+=1
>>>>count
        1
>>>>count+=1
 >>>>count 2
```

Instead of writing this complex statement we can write it inshort Like this

```
  >>>>count+=1
  >>>> count
            3
```

Arithmetic With All Type



	+	-	*	/	//	%	**
int	✓	✓	✓	✓	✓	✓	✓
float	✓	✓	✓	✓	✓	✓	✓
bool	✓	✓	✓	✓	✓	✓	✓
complex	✓	✓	✓	✓	×	×	✓
str	✓	×	✓	×	×	×	×

- This table shows how operators are compatible with other Datatypes.
- ***If you are performing float division you'll get float result.***
- ***If you are performing floor division you'll get integer result.***
- True = 1, False = 0

Boolean

- Bool v/s Bool the result will always be integer.
- Bool v/s complex the result will always be complex.

Complex

- ***Complex with integer the result is always complex.***
- ***Complex works with all except // and % as // is used to convert float type to int type but can a complex number do all this, no so it doesn't work same goes for %.***

String

- Adding 2 strings is called 'concatenation'. ***If one type is integer and the other is string concatenation doesn't work.*** The 2 types to be added should be string only.
- ***In multiplication i.e., * one type should be string and the other should be an integer only.***
- Float doesn't work for strings.

Conditional Statements

- Control statements are the statements that control the flow of execution of statements so that they can be executed repeatedly and randomly
- Whenever we write a program it will execute linearly but in set of statement, we add conditions it will break the linear execution and will execute the conditional statement (if else) then go further

if

- First the condition is tested If condition is true it will execute. if block is false than it will execute else block.
- We can write one or more statement after colon (:)
- if is false, then the statement mentioned after : are not executed
- Let's see the syntax:

```
if condition :
        statements
```

- We can see in syntax there is indentation. Indentation is important in python. It is used in thebeginning of the statement

if else

- The if ....else executes statement evaluates test expression and will execute the block of if only when the test condition is True.
- If the condition is False, the body of else is executed. Indentation is used to separate the block.
- Let's see syntax for if else statement

```
if condition :
        statement
else :
        statement
```

Relational Operator:

Operator	*Name*	*Example*
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

- For writing the conditions we use relational and logical operators and the output will be always " True " or " False "
- This is also called as comparison operators
- Let's take an example for better understanding a = 10 b = 20

```
if a < b :  T a
  <= b :T a >
  b :  F a >= b
  :  F  a == b :
  F a != b  :  T
```

- Lets take an example by using if ... else

```
a = int ( input ( " enter a number" ) ) if a

< 0 :
            print ( " Negative " )
    else:
             print ( " Positive " )
```

Output: Negative

Logical Operator:

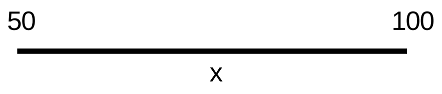
Operator	*Description*	*Example*
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

Logical operators use to write compound conditions

AND

- Lets take example - cond1 and cond2
  - If the cond1 and cond2 both are true than it will return true value
    - True - 1   False - 0

. AND is like multiplication and it is useful to check the range of values



If x >= 50 and x <= 100

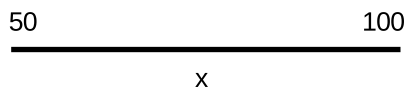
- Lets take an example about subjects (maths , phy , and chem) where above 45 will be the passing marks

```
if math>=45 and phy >=45 and chem >=45 : print
           ("pass")
 else :
           print (" fail ")
```

- AND is use to check this type of conditions where everything must be true (both conditions should be true)

OR

- If the cond1 or cond2 both are false returns false
- OR is like addition



- OR is useful to check outside the range

```
if x <= 50 or x >=100 :
```

NOT

- It will negate it
- Returns the opposite value (If its true returns false if its false return true)
- It is not commonly used

Find difference between 2 numbers.

# If 5 - 15 = -10 we don't want in negative one we want to know only differences

Solution:

```
no1 = int(input('Enter first number')) no2 =
int(input('Enter second number')) if no1 -
no2 >=0:
    print(no1-no2) else:
    print(no2-no1)
```

Check if a number is odd or even.

# When we divide it by 2 the remainder must be zero then it is said as even

Solution:

```
n = int(input('Enter a Number')) if n
% 2 == 0:
    print('Even')
else:
    print('Odd')
```

Check for Age eligibility for casting a vote.

#Only 18+ can vote

Solution:

```
age = int(input('Enter your age')) if
age >= 18:
    print('Eligible') else:
    print('Not Eligible')
```

Check if marks of a subject are within range 0 - 100.

#if - valid else - invalid

Solution:

```
marks = int(input('Enter Marks')) if
marks >= 0 and marks <= 100:
    print('Valid')
else:
    print('Invalid')
```

Check if a person is ‘ Male ‘ or ‘ Female’

# to check person is Male or female by taking input from the user

Solution:

```
gender = input('Enter Gender') if
gender=='m' or gender=='M':
    print('Male')
else:
    print('Female')
```

Check if the person is eligible to work.

Solution:

```
age = int(input('Enter Age')) if
age >= 18 and age <= 60:
    print('Eligible') else:
    print('Not Eligible')
```

Check if a statement has passed or failed, by taking marks in 3 subjects.

# lets take marks of subjects (maths, physics, chemistry) marks are between 0 - 100 and passing marks should be above 45

Solution:

```
math = int(input('Enter Maths Marks')) phy
= int(input('Enter Physics Marks'))
chem = int(input('Enter Chemestry Marks'))
if math >= 45 and phy >= 45 and chem >= 45:
    print('Passed')
else:
    print('Failed')
```

Check if a person is authorise for admin access

# only admin can access the data by taking the user input admins are ‘ John ‘ and ‘ smith’

Solution:

```
username = input('Enter Username')
if username=='john' or username=='smith':
    print('Authorised')
else:
    print('Not Authorised')
```

Check if a given lowercase character is vowel and consonant

#vowels - a , e , i , o , u consonants - other than vowels are said as consonants

Solution:

```
ch = input('Enter a lower case letter')
if ch == 'a' or ch == 'e' or ch == 'i' or ch == 'o' or ch == 'u': print('Vowel')
else:
    print('Consonant')
```

Nested if and elif

- elif is also a keyword in python like if.
- An if inside another if or another else statement is called nested if.
- Instead of using nested if python provides elif because using if will make the code more deeper and take much lines of code when compare to elif.

Q) Write a program taking input from user and check which one is elder among them?

# Taking input from users and typecasting it into float value, setting if and else condition for comparison, then again applying NESTED if and else for further comparisons and printing the results based on the input given by the user.

- Nested if else without elif.

```
john =
float(input("Enter the John's age: ")) smith =
float(input("Enter the Smith's age: ")) ajay =
float(input("Enter the Ajay's age: "))
if john > smith and john > ajay: print("John is
    elder ")
else:
    if smith > ajay: print("Smith is
        elder ")
    else:
        print("Ajay is elder ")
```

- Nested if else with elif.

# Replacing else and if with elif hence making the code more readable and understandable.

```
john = float(input("Enter the John's age: ")) smith =
float(input("Enter the Smith's age: ")) ajay =
float(input("Enter the Ajay's age: "))
if john > smith and john > ajay:
    print("John is elder ")
elif smith > ajay:
    print("Smith is elder ")
else:
    print("Ajay is elder ")
```

Calculate discounted amount

amount <= 1000 —— 10%
1000 < amount <=5000 ——20%
5000 <=10000 ——30%
10000 < amount ——50%

Solution:

```
amount = float(input('Enter Bill Amount')) if
amount <= 1000:
    discount = amount * 10 / 100
elif amount > 1000 and amount <= 5000: discount =
    amount * 20 / 100
elif amount > 5000 and amount <= 10000: discount =
    amount * 30 / 100
else:
    discount = amount * 50 / 100 payAmount
= amount - discount print('Pay ', payAmount)
```

Take a day number and display day name

# Take any number and give it a day name. It’s like switch case in other languages

```
day = int(input('Enter Day Number')) if
day == 1:
    print('Sunday')
elif day == 2:
    print('Monday')
elif day == 3:
    print('Tuesday')
elif day == 4:
    print('Wednesday')
elif day == 5:
    print('Thursday') elif
day == 6:
    print('Friday') elif
day == 7:
    print('Saturday')
else:
    print('Invalid Day Number')
```

Q) Check whether year is the leap year or not

#A year have 365 days and 1/4 day, 4 quarter day makes one day and that will be added to the February month. Usually, February will have 28 days but in leap year (year % 400 == 0) it will have 29 days. Every fourth year will be the leap year. If the year is century (year %100 == 0) then it is not leap year.

Nested if-elif Statement

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

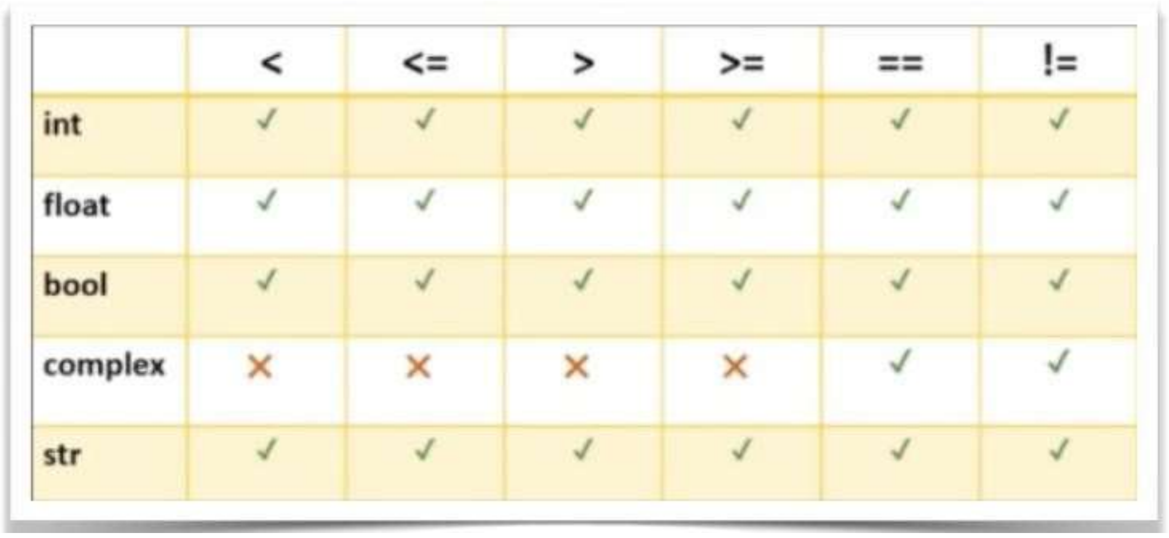
Or

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

Or(*if-else*)

```
year = int(input('Enter year: '))
if year % 4 == 0 and year % 100!=0 or year % 400==0: print(year,"is Leap
    Year")
else:
    print(year,"is not a Leap Year")
```

Detail Relational Operator



	<	<=	>	>=	==	!=
int	✓	✓	✓	✓	✓	✓
float	✓	✓	✓	✓	✓	✓
bool	✓	✓	✓	✓	✓	✓
complex	✗	✗	✗	✗	✓	✓
str	✓	✓	✓	✓	✓	✓

Complex

• Relational operator can perform using int float bool and string but in complex it will not let's see why.

```
a = 5+6j                b = 9+8j
        a < b #error
```

• Because it will have 2 elements that are real and imaginary. Which one should be compared a real part or imaginary part. It can't be compared. So, it is not possible.

String

• Strings are compared based on their dictionary
order s1= " America" s2= " Brazil "
s1<s2 ———> ***True***

What if a string have same alphabet
s1= "Brazil " s2= " Bracket "
s1<s2 ———> ***False***

Because in dictionary Bracket will come first then Brazil so s1 Is less than s2 What if we have
s1 = " brazil" s2="Brazil "

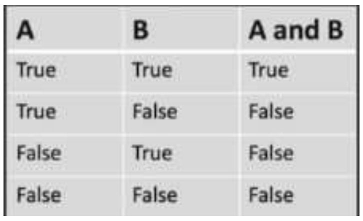
Both are not same cause python is case sensitive s1<s2 —
——> ***False***

Because uppercase letter is lower then the lowercase. So, s1 is not less than s2.

Logical Operator

• and, or, not are logical operators.
• Logical operators work upon compound conditional statement by using and, or, not

***Truth Table for* and**



A	B	A and B
True	True	True
True	False	False
False	True	False
False	False	False

- They will work upon non bool type????? Let's see
  a = 5 b = 10 c = 15
  if a<b and a<c 5<10
  and 5<15

• And will return true if all are true. The conditions given in if are
true 0 - False other than 0 is - True

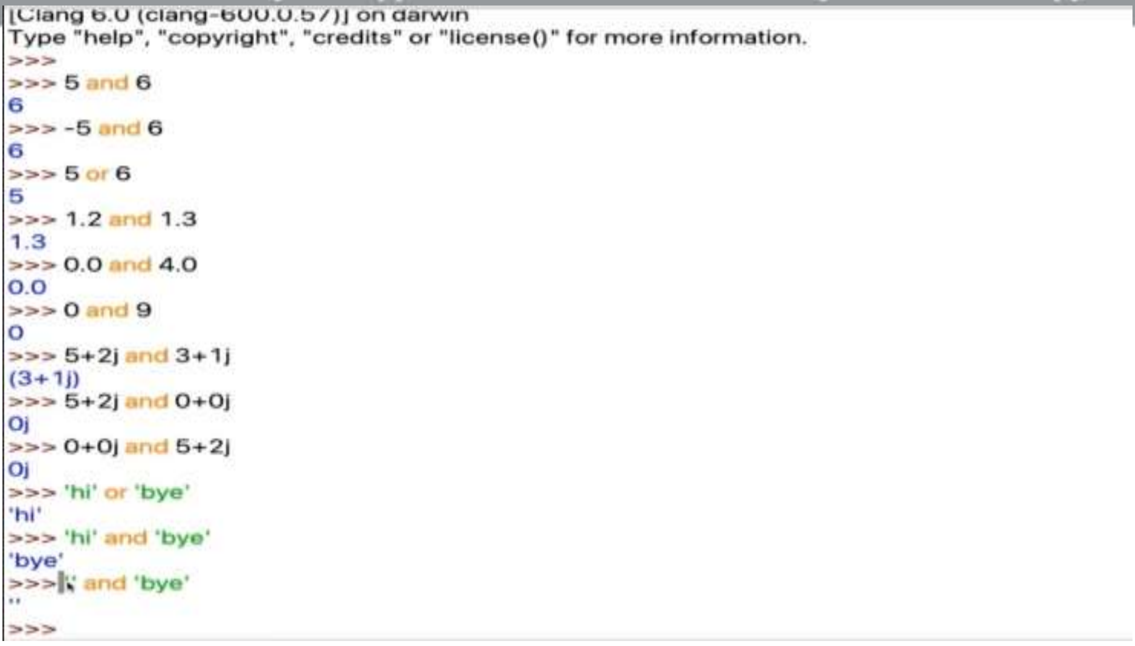
if 5 and 6 returns True T
T

if -5 and 7 returns True # anything other than is True

if 0 and 6 returns False # 0 – False

• They will work on non bool type. It will not give the result T but it will give the second number (Anything other than 0 is True)

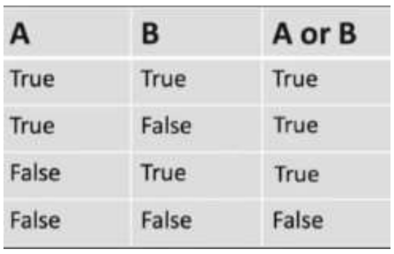
•Empty string is treated as false



```
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>>
>>> 5 and 6
6
>>> -5 and 6
6
>>> 5 or 6
5
>>> 1.2 and 1.3
1.3
>>> 0.0 and 4.0
0.0
>>> 0 and 9
0
>>> 5+2j and 3+1j
(3+1j)
>>> 5+2j and 0+0j
0j
>>> 0+0j and 5+2j
0j
>>> 'hi' or 'bye'
'hi'
>>> 'hi' and 'bye'
'bye'
>>>'' and 'bye'
''
>>>
```

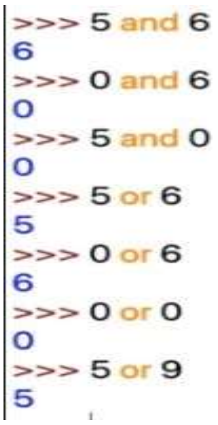
Short Circuit: In ***and*** when the first one is false, we don't have to check the second value cause it will be false cause and will return true if both the conditions are true otherwise false. If the first is True then it will check the second statement.

Truth Table for or



A	B	A or B
True	True	True
True	False	True
False	True	True
False	False	False

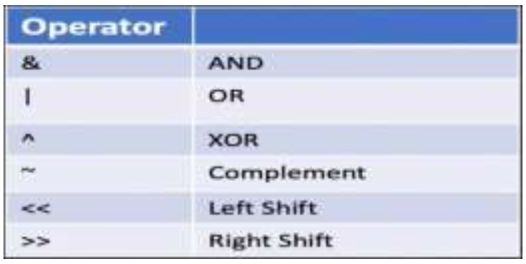
In ***or*** it will check the first statement If it true than it will not check the second statement cause it will return true.



```
>>> 5 and 6
6
>>> 0 and 6
0
>>> 5 and 0
0
>>> 5 or 6
5
>>> 0 or 6
6
>>> 0 or 0
0
>>> 5 or 9
5
```

Bitwise Operator

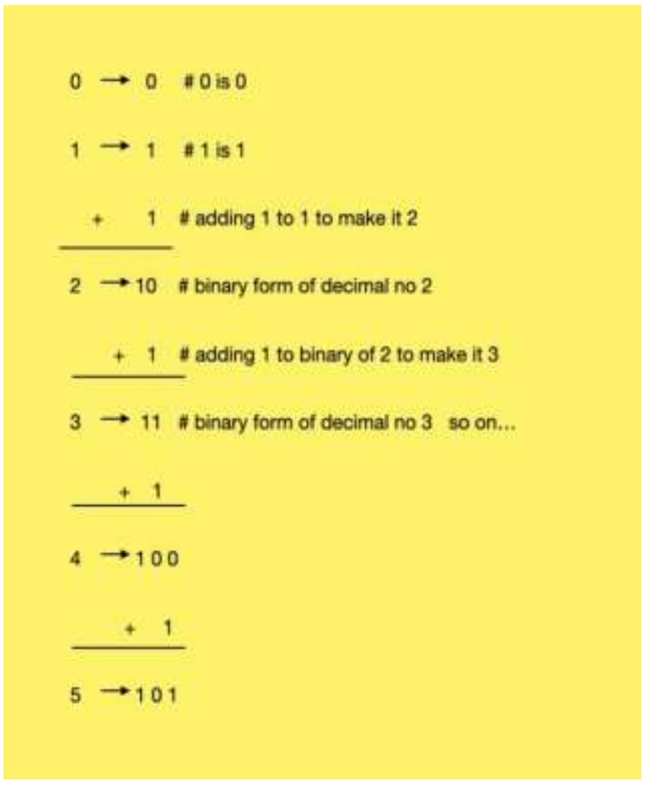
• The Bitwise operator are AND, OR, XOR, Complement, Left shift , Right Shift.
• These operators work on integral type of data and they perform operation on Binary representation of data i.e., 0's and 1's.
• Bitwise operators are used in applications of networking, Encryption and Decryption etc.
• Bitwise calculations start from right hand side.



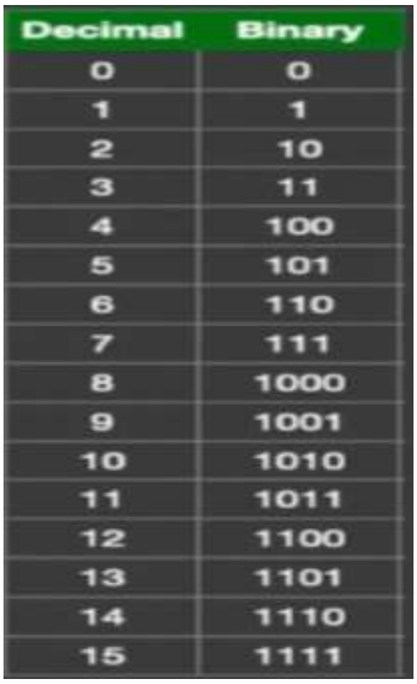
Operator	
&	AND
|	OR
^	XOR
~	Complement
<<	Left Shift
>>	Right Shift

Understanding Binary numbers system:

• To understand bitwise, we first need to understand binary number system. Binary number system uses only 0's and 1's
• Decimal number system uses numbers from 0 - 9
• To convert decimal to binary we can do it in either 2 ways

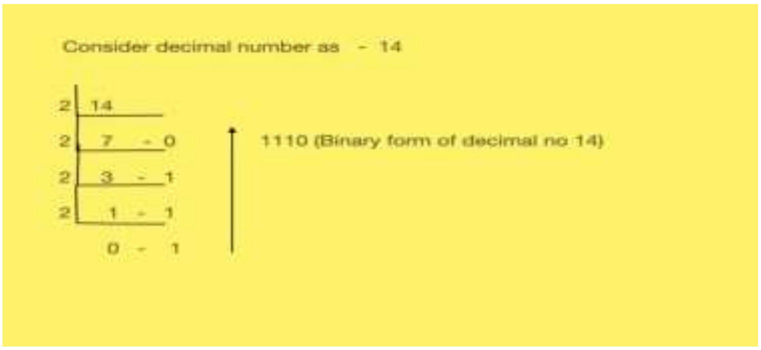


```
0 → 0    # 0 is 0

1 → 1    # 1 is 1

  +  1   # adding 1 to 1 to make it 2
-------
2 → 10   # binary form of decimal no 2

   + 1   # adding 1 to binary of 2 to make it 3
-------
3 → 11   # binary form of decimal no 3   so on...

   + 1
-------
4 →100

   + 1
-------
5 →101
```

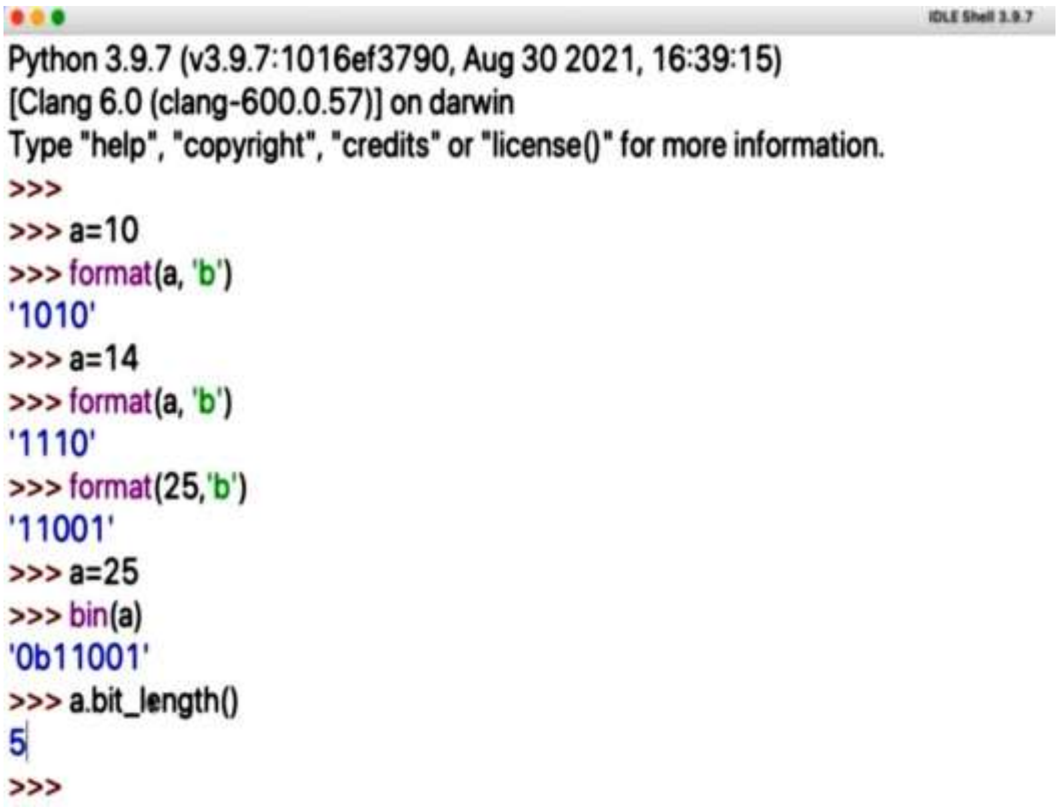


Decimal	Binary
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Table for decimal to binary


Consider decimal number as - 14
2 | 14
2 | 7 - 0
2 | 3 - 1
2 | 1 - 1
0 - 1
1110 (Binary form of decimal no 14)

Let us check this in IDLE



```
Python 3.9.7 (v3.9.7:1016ef3790, Aug 30 2021, 16:39:15)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>>
>>> a=10
>>> format(a, 'b')
'1010'
>>> a=14
>>> format(a, 'b')
'1110'
>>> format(25,'b')
'11001'
>>> a=25
>>> bin(a)
'0b11001'
>>> a.bit_length()
5
>>>
```

Bitwise Operations

- Let us understand all the bitwise operators with an example
- Consider a = 10 (binary of 10 is 1010) b = 13 (binary of 13 is 1101)

AND Operations (&)

- AND works on multiplication
- Working of AND –

1 * 1 = 1
1 * 0 = 0
0 * 1 = 0
0 * 0 = 0

Example: a - 1010
b - 1101
a & b - 1000 = 8 (Decimal form of binary number)

OR Operations (|)

- OR works on addition
- Working of OR –

1 + 1 = 1
1 + 0 = 1
0 + 1 = 1
0 + 0 = 1

Example: a - 1010
b - 1101
a | b - 1111 = 15 (Decimal form of binary number)

XOR operations (^)

- Working of XOR –

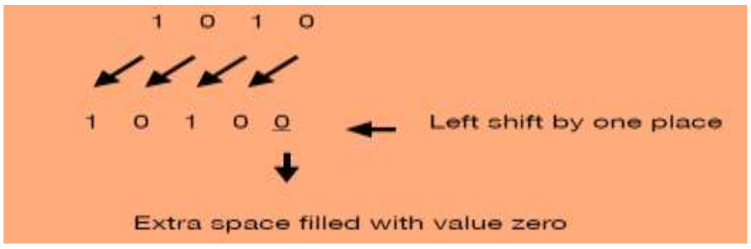
1 ^ 1 = 0
1 ^ 0 = 1
1 ^ 1 = 1
1 ^ 0 = 0

Example: a - 1010
b - 1101
a ^ b - 0111 = 7 (Decimal form of binary number)

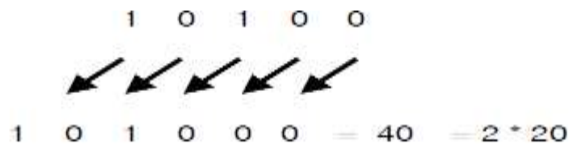
- Leading zero doesn't make sense
- Hence, we consider 111 whose decimal form is 7

Left shift (<<) and right shift (>>):

- a = 10 (binary of 10 = 1010)
- If we left shift 'a' by one place i.e., a << 1 then

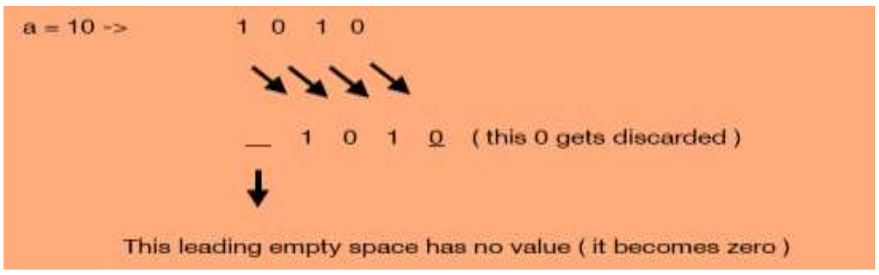


- After left shift by one place the bit that is freed will be taken as zero.
- Now 10100 = 20 = 2* 10
- If we left shift 20 again then,



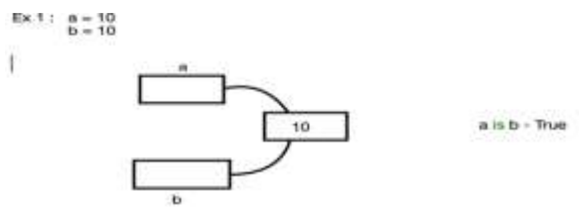
- We see that when we left shift the number will double for one place i.e; a << n , a * 2 (power n)
- If we double the left shift i.e., a. << 2 then a << 2 = 2(power n) * a = 4*10 = 40
- similarly, if we do a << 5, then the value gets double for 5 times a << 5 = 2(power 5)

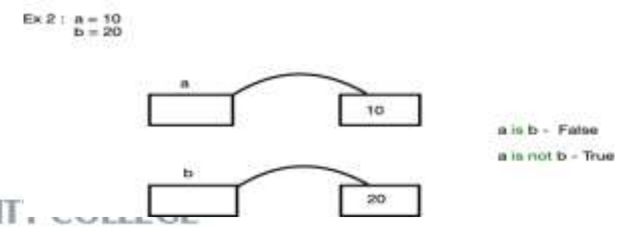
* a = 32 * 10 = 320
- If we RIGHT SHIFT the number will become half.
- Right shift operator divides by 2 i.e.; a >> n a / 2 (power n)


a = 10 -> 1 0 1 0
__ 1 0 1 0 (this 0 gets discarded)
This leading empty space has no value (it becomes zero)

is - is not operator.

- is and is not will check whether two variables are referring to the same memory or not.
- They return Boolean value (Ture, False) as result.
- They are checked based on variable and not on value in the memory.
- For each input values (user define) python creates new memory even when the given valuesare same.
- When we are assigning values in the program python will refer it to the same memory as it isdynamically typed Programming language.





Precedence level	***Operator***	***Meaning***
1 (Highest)	()	Parenthesis
2	**	Exponent
3	+x, -x ,~x	Unary plus, Unary Minus, Bitwise negation
4	*, /, //, %	Multiplication, Division, Floor division, Modulus
5	+, -	Addition, Subtraction
6	<<, >>	Bitwise shift operator
7	&	Bitwise AND
8	^	Bitwise XOR
9	|	Bitwise OR
10	==, !=, >, >=, <, <=	Comparison
11	is, is not, in, not in	Identity, Membership
12	not	Logical NOT
13	and	Logical AND
14 (Lowest)	or	Logical OR

Python Operators Precedence

Python Interpreter, Using Python as calculator, Python shell, Indentation. Atoms, Identifiers and keywords, Literals, Strings, Operators (Arithmetic operator, Relational operator, Logical or Boolean operator, Assignment, Operator, Ternary operator, Bit wise operator, Increment or Decrement operator)

Creating Python Programs:

Input and Output Statements, Control statements (Branching, Looping, Conditional Statement, exit function, Difference between break, continue and pass.), Defining Functions, default arguments, Errors and Exceptions.

Iteration and Recursion: Conditional execution, Alternative execution, Nested conditionals, The return statement, Recursion, Stack diagrams for recursive functions, Multiple assignment, The while statement, Tables, Two-dimensional tables Strings and Lists: String as a compound data type, Length, Traversal and the for loop, String slices, String comparison, A find function, Looping and counting, List values, Accessing elements, List length, List membership, Lists and for loops, List operations, List deletion. Cloning lists, Nested lists

Object Oriented Programming: Introduction to Classes, Objects and Methods, Standard Libraries.

Data Structures: Arrays, list, set, stacks and queues.

Searching and Sorting: Linear and Binary Search, Bubble, Selection and Insertion sorting.

- Loops are also called as repeating statements. The loops are of 2 types in python

  I. ***while loop***
  II. ***for loop***

- If you want a set of statements to repeat again and again in the program then we use loops. The statements can repeat either ***'for number of times'*** or ***'As long as the condition is true'.***
- In ***while condition,*** If the condition is true, the while block will get executed after the execution it will go back again and check the condition, if again the condition is true, it'll get executed, once the condition becomes false it'll stop the execution.

Syntax: while condition :

```
        Statements
```

- Whatever condition you are using in while you should modify it in the statements else your loop will not work properly.

Example 1: (Statements repeating number of times)

Q) Print hello 10 times using while loop.

#in while loop use the counter as condition and set its value from 1 -10, modify the counter in the statements and get the result.

```
count = 0
while count < 10:
    print(count + 1, 'Hello')
    count = count + 1
```

Example 2: (As long as the condition is true)

Q) Write a program to print output of the given number from last to first.

# Take input from users in integer form, the condition should be a number greater than zero and divide the number and print its result in integer form only.

```
n=int(input("Enter the number: "))
while n>0:
    r=n%10
    n=n//10
    print(r)
```

While Loop: Student Challenge

1 Q) Display multiplication table for a given number.

#We have a number and for that we have to print a multiplication table, as the pattern for multiplying is same, we are using while loop here.

```
n = int(input('Enter a number for Multiplication Table')) counter =
1
while counter <= 10:
    print(n, 'X', counter, '=', n*counter)
    counter = counter + 1
```

2 Q) Counting the number of digits in a number.

#Taking input from user in integrant write a counter condition in while loop for just counting the numbers given as input from user.

```
n = int(input('Enter a Number '))
counter = 0
while n > 0: n
    = n // 10
    counter += 1
print('Number of Digits are', counter)
```

3 Q) Finding sum of digits in a number

#If the number is given then find its sum by using while loop

```
n = int(input('Enter a Number '))
sum = 0
while n > 0:  r
    = n % 10 n
    = n // 10
    sum = sum + r
print('Sum of Digits is', sum)
```

4 Q) Reversing a number

#Take input from user and make that given number into reverse order using while loop

```
n = int(input('Enter a Number')) rev
= 0
while n > 0:  r
    = n % 10 n
    = n // 10
    rev = rev * 10 + r print('Reverse
number is', rev) print(n)
```

5 Q) Check if a number is a palindrome.

# If the reverse of a number is equal to the original number then we say its a palindrome

```
n = int(input('Enter a Number')) m =
n
rev = 0 while
n > 0:
    r = n % 10 n
    = n // 10
    rev = rev * 10 + r if
m == rev:
    print('Number is a Palindrome')
else:
    print('Number is not a Palindrome')
```

6 Q) find sum of given numbers as input

#you have to ask number of numbers by taking user input and find the sum of that inputs. And we have to count so we use counter for it.

```
num_of_nos = int(input('Enter number of number')) sum = 0
count = 0
while count < num_of_nos:
    n = int(input('Enter a number'))
    sum=sum + n
    count = count + 1
print('Sum is ', sum)
```

7 Q) Find the sum of +ve and -ve number

# user may enter positive or negative number.

```
num_of_nos = int(input('Enter number of number')) Psum =
0
Nsum = 0
count = 0
while count < num_of_nos:
    n = int(input('Enter a number')) if
    n > 0:
        Psum = Psum + n else:
        Nsum = Nsum + n count
    = count + 1
print('Positive Sum is ', Psum) print('Negative
Sum is ', Nsum)
```

8 Q) find the maximum numbers from the given number.

```
num_of_nos = int(input('Enter number of number')) count =
0
max = int(input('Enter a num'))
while count < num_of_nos - 1:
    n = int(input('Enter a number')) if
    n > max:
        max = n
    count = count + 1 print('Max
number is', max)
```

9 Q) Convert decimal to binary.

```
n = int(input('Enter a number')) bin
= 0
while n > 0: r
    = n % 2 n =
    n // 2
    bin  =  bin  *  10  +  r
brev = 0
while bin > 0:  r =
    bin % 10
    bin = bin //10
    brev = brev *10 +r
```

Solution
:

```
n  =  int(input('Enter  a  number'))
bin = ''
while n > 0: r
    = n % 2 n =
    n // 2
    bin   =   str(r)+bin
print(bin)
```

10 Q) Factorial of a number

```
n = int(input('Enter a number: '))
m=n
fact = 1 if
n<0:
    print("Negative number factrorial can,t possible. ") else:
    while n>=1:
        fact=fact*n
        n=n-1
    print(m,"factrorial is",fact)
```

Infinite Loop - break - continue – pass

• Infinite loop is a loop that goes on, which never stops such type of loop is called infinite loop.
• The following example is of infinite loop.
//This while loop never stops, you need to keep on giving input and it generated results it never stops

```
while True:
    no = int(input("Enter a number: ")) if
    no > 0:
        print("Positive Number ")
    else:
        print("Negative Number")
```

Break Statement

- To stop such infinite loop, we can use break statement.
- A break statement will stop the loop.
- A break statement can be used in other loop situations as well apart from infinite loop
- The below example shows the use of break statement

```
while True:
    no = int(input("Enter a number: ")) if
    no > 0:
        print("Positive Number ")
    elif no < 0:
        print("Negative Number")
    else:
        break
```

Continue Statement

- Continue means it will not execute the rest of the loop it will simply go to beginning of the loop and continue its execution.
- Continue is used for logical design.
- Let's understand this with an

```
example count = 0
while count < 3:
    no = int(input("Enter a number: ")) if
    no % 3 == 0:
        continue
    print(no)
    count = count+1
```

Pass Statement

- Pass - In some cases when you don't have to do anything use them just pass the it.
- Pass means nothing to do.
- In python when you write a block of code you must write something if there's nothing to write, then simply write pass statement.
- Let's understand this with an example

```
while count < 3:
    no = int(input("Enter a number: ")) if
    no % 3 == 0:
        pass
    else:
        print(no) count
    = count+1
```

Else suite

```
        if condition:
        - - - - - -
        -----------        {If condition is true then this if}
        - - - - - - -
    else:
    - - - -
    -----  {if condition is false then this else block}
    - - -
- - - -
- - -
- - - -
- - - -
```

True-> *if block is executed*
False->*else block is executed*

What is else suite?

• The else suite will be always executed irrespective of the statements in the loop are executed or not.

• If block can be written with while as well as it can also be written with for loop and also along with try block.

How else suite is useful with while?

Sample:

```
    while condition:
    --------- as long as this condition is true
    --------- its get executed repeatedly
    - - - - - - -
    - - - - - - -
```

And once the condition becomes false it will come out of the loop and executes next statement.

We can also write else block.

```
    while condition:
    - - - - - - - - -
    - - - - - - - - -true
    - - - - - - - - -
    else:
    - - - - - -
    - - - - - - false
    - - - - -
```

• If false it will come to else block.

• *Suppose if this loop is stopped using* ***break statement without becoming false****. Then this* ***else block will not be executed.***

• This else block will execute if only condition becomes false.

• So, it is like if else statement means true if while is executed and if it is false else will be executed.

• if while never becomes false then else block will never be executed.so, this else block can be used to confirm that while loop has successfully executed without any break.

Working of else suite

```
count = 1
while count <= 10:
    print(count) count
    = count+1
else:
    print("Printed all 10 Number ")
print("End of a Program ")
```

- It has printed all the 10 numbers and also shows that it jumped to else on becoming false
- Now we will use break to stop the loop

```
count = 1
while count <= 10:
    print(count) count
    = count+1 if count
    > 5:
        break
else:
    print("Printed all 10 Number ")
print("End of a Program ")
```

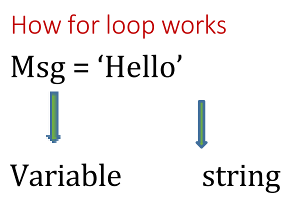
- So, else block is used to confirm that while loop has executed successfully. And it stops when it becomes false.
- If it stops abruptly using break then else will not execute.

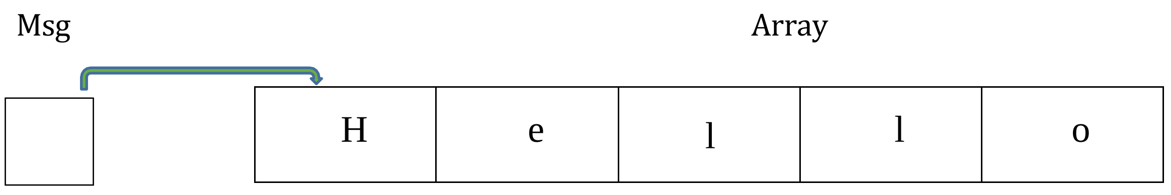
11 Q) GCD and LCM of two number

```
m = int(input("Enter the first number: "))
n = int(input("Enter the second number: "))
a = m
b = n
while m != n: if
    m > n:
        m = m-n
    else:
        n = n-m
print(a, "and", b, "GCD is ", m) lcm =
a*b // m
print(a, "and", b, "LCM is ", lcm)
```

Introduction: For Loop

- Loops are useful for repeatedly executable statements.





So, the string is arranged in the form of an array.

Suppose if we write for x

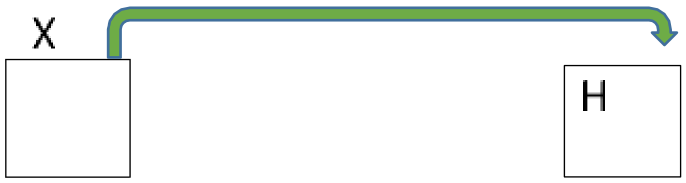
in msg:

    print(x)

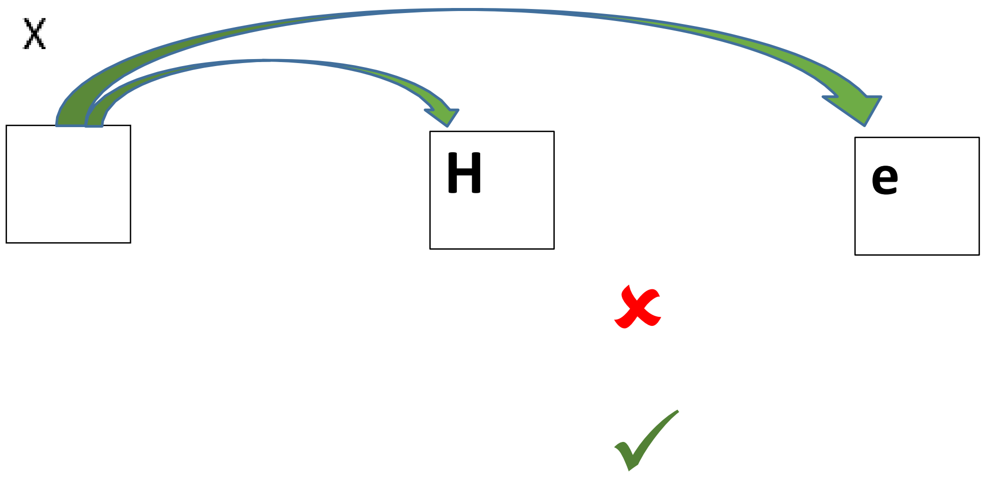
It will print like this

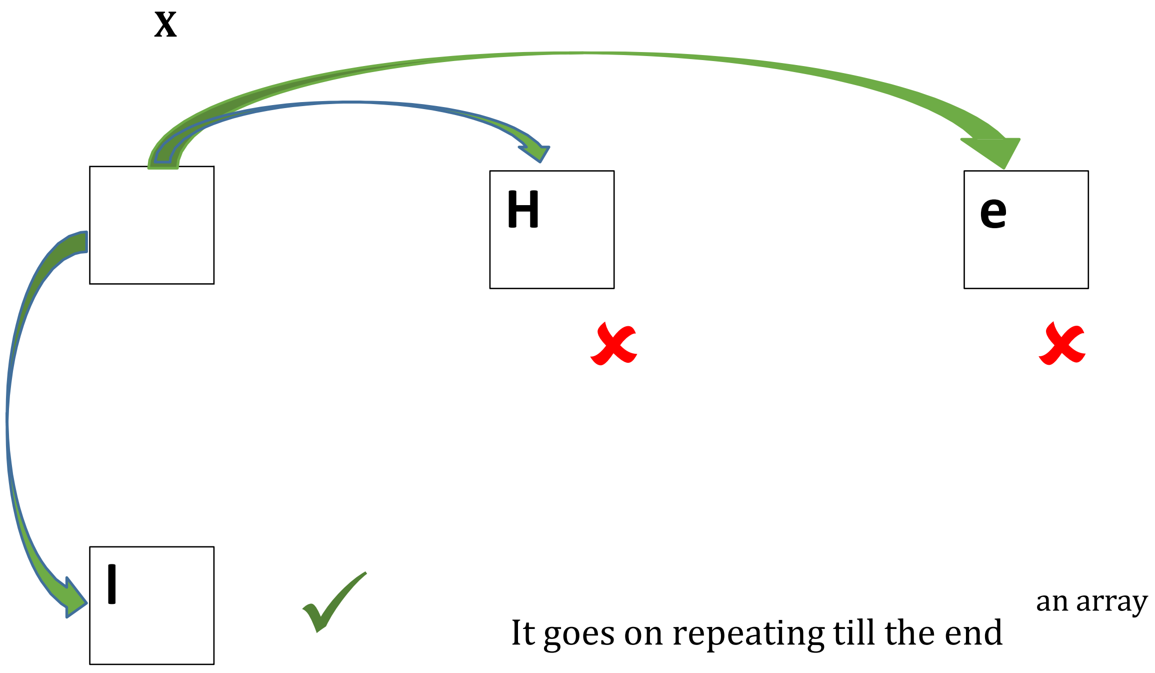
```
H
e
ll
o
```

first it prints H



Then again it repeats using for and prints the next letter





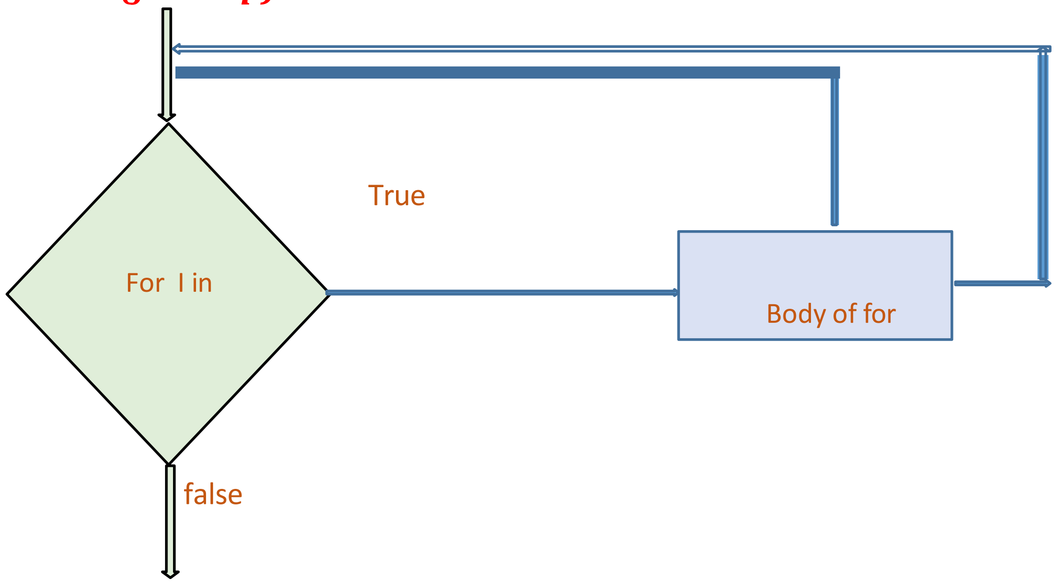
It goes on repeating till the end an array

This loop is also called as for each loop in languages like java and c++ Msg='hello'

For x in msg: #for loop runs for each statement in the sequence

(print x)

Flow Chart (for loop)



- So, if the value condition is true so loop of the body is executed continuously and then at one point it stops.
- Basically, for loop depends upon the elements in a sequence.
- Elements has finished means false and elements has not finished means true.

What is Range?

- Range is a function
- Suppose range (10)
- 0,1,2,3,4,5,6,7,8,9 it will stop at just one number beforethe last number.
- If you give range (5,10) 5,6,7,8,9

  Start End
- Can also give step range (1,10,2) 1,3,5,7,9

  Start End Step
- Range (10, 0, -1) 10,9,8,7,6,5,4,3,2,1

  Start End Step
- Range (-10, -1, 2) –10, -8, -6, -5, -4,-2

  Start End Step
- this is how range function works

Coding

Input:

```
msg='Hello'
for x in msg:

     print(x)
```

It's printing as H e l l o one by one. So, for loop executes for each element inside given statement.

Usually, these types of things which are having elements called as sequence.

Input:

```
for i in range(10): #it will take in from 0 to 10
    print(i)
```

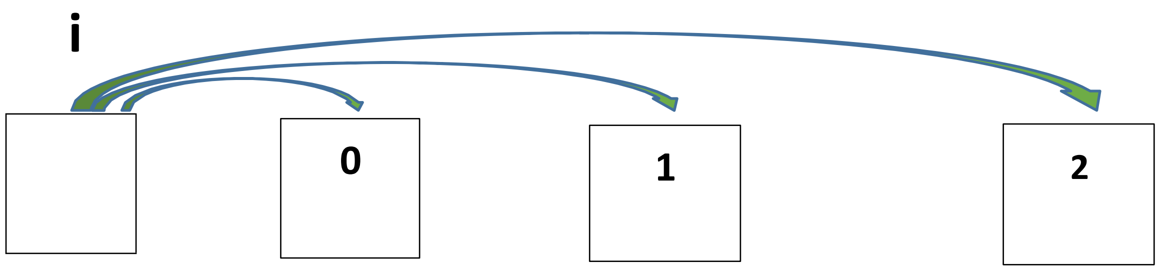
```
for i in range(5,10): #it will print 5 to
    10print(i)
```

```
for i in range(0,10,2): #it will print on
everytwo steps from 0 to 10

    print(i)
```

```
for i in range(10,0,-1): #it prints the numbers
inreverse order

    print(i)
```

How i is working?

```
for i in range (0,10):
    print(i)
```



It works upto 10.

For each i in this range(0,10)

print(i) will be executed Each i is

0,1,2---upto 9

For Loop: Student Challenge

# 1 Q) Display multiplication table for a given number.

```
no = int(input('Enter a Number ')) for
i in range(1, 11):
    print(no,'X', i,'=',i*no)
```

2 Q) Find the factorial of a given number.

```
no = int(input('Enter a Number ')) fact
= 1
if no < 0:
    print("Negative number factorial is not possible.") else:
    for i in range(1, no+1): fact =
        fact * i
    print('Factorial of',no,'is', fact)
```

3 Q) Print n terms of AP series.

```
a = int(input("Enter initial term: "))
d = int(input("Enter common Difference ")) n =
int(input("Enter Number of Terms ")) for t in
range(a, a + n * d, d):
    print(t)
```

4 Q) Print n terms of Fibonacci series.

```
n = int(input('Enter Number of Terms '))
a = 0
b = 1
print(n," terms Fibonacci series are: ") for i
in range(n):
    print(a)
    c = a + b
    a = b
    b = c
```

5 Q) Find the Factors of a number.

```
n = int(input('Enter a Number ')) for i
in range(1, n+1):
    if n % i == 0:
        print(i)
```

6 Q) Check if a number is prime or not.

```
n = int(input('Enter a Number '))
count = 0
for i in range(1, n+1): if n
    % i == 0:
        count += 1 if
count == 2:
    print(n,'is a Prime')
else:
    print(n,'is Not a Prime')
```

7 Q) Break_Continue_Pass with for

loop Break example with for loop:

```
for i in range(0,10): if i
    > 5:
        break
    else:
        print(i)
```

else suit example with for loop

```
for i in range(0,10):
    print(i)
else:
    print("For completed properly")
```

or

```
for i in range(0,10): if i
    > 5:
        break
    else:
        print(i)
else:
    print("For completed properly")
```

Continue example with for loop:

```
for i in range(0,10): if i
    % 5==0:
        continue
    print(i)
```

Pass example with for loop:

```
for i in range(0,10): pass
print("Program ended")
```

Nested Loops

We know that how for loops works

• For example,

```
for i in range( 0,5):
          print ( i )
```

Output:

```
0
1
2
3
4
```

• ***Loop inside loop is said as nested loop***

```
for i in range( 0,5) : for j
    in range( 0,5) :
              print ( i , j )
```

- The outer loop can contain any amount of inner loop.
- In each iteration of the outer loop inner loop execute all its iteration. For each iteration of an outer loop the inner loop re-start and completes its execution before the outer loop can continue to its next iteration.
- Working of nested loop

```
for i in range( 0,5) : for j
    in range( 0,5) :
              print ( '(',i , j,')', end= " " )
    print("")
```

Q 1) To Print prime numbers 1-100.

```
for n in range( 1, 100+1):
    count=0
    for i in range(1, n+1): if n
        % i ==0:
            count+=1 if
    count == 2:
        print(n)
```

Q 2) Draw Patterns

```
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
```

```
for i in range( 0,5): for j
    in range(0,5):
        print('* ',end=' ')
    print('')
```

or

```
for i in range(0,5): print('*
    '*5)
```

```
*
* *
* * *
* * * *
* * * * *
```

```
for i in range( 0,5): for j
    in range(0,5):
        if i>=j:
            print('* ',end=' ') print('')
```

or

```
for i in range(0,5): print('*
    '* (i+1))
```

```
* * * * *
* * * *
* * *
* *
*

for i in range(5,0,-1):
    print('* '* i)
```

Match Case

- Match case is used instead of switch case in languages like c++, java we use switch case it is like if else.
- It is similar to if else but syntax written is similar to switch case . But works same as if and elif.
    - ***Example:***

```
day = int(input('Enter Day Number'))
match day:
    case 1:
        print('Sunday')
    case 2:
        print('Monday')
    case 3:
        print('Tuesday')
    case 4:
        print('Wednesday') case
    5:
        print('Thursday')
    case 6:
        print('Friday')
    case 7:
        print('Saturday')
    case _:
        print('Invalid Day Number')
```

Python range () function:

The Python range () function returns a sequence of numbers, in a given range. The most common use of it is to iterate sequences on a sequence of numbers using Python loops.

Python xrange () function:

The xrange () function in Python is used to generate a sequence of numbers, similar to the Python range () function. The Python xrange () is used only in Python 2.x whereas the range () function in Python is used in Python 3.x.

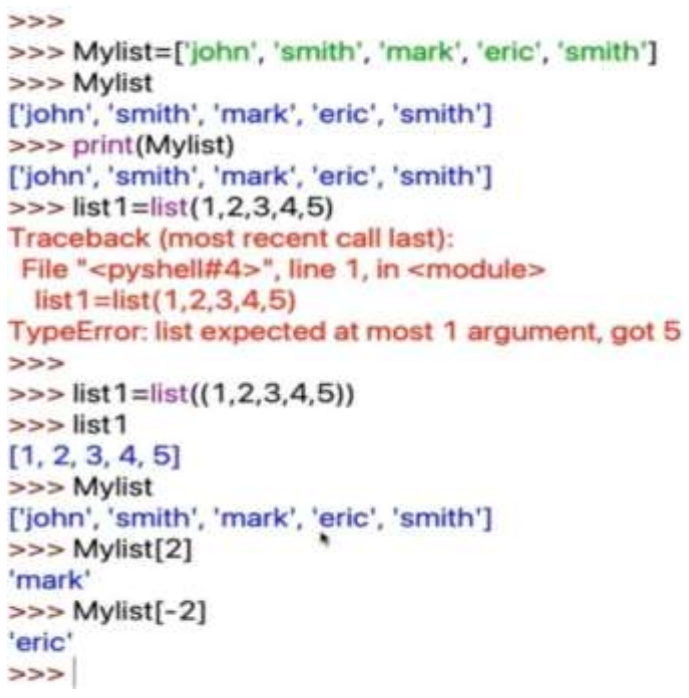
Difference between range() and xrange() in Python:

range()	***xrange()***
Returns a list of integers.	Returns a generator object.
Execution speed is slower	Execution speed is faster.
Takes more memory as it keeps the entire list of elements in memory.	Takes less memory as it keeps only one element at a time in memory.
All arithmetic operations can be performed as it returns a list.	Such operations cannot be performed on xrange().
In python 3, xrange() is not supported.	In python 2, xrange() is used to iterate in for loops.

Introduction to List

- List is a collection of ***ordered objects*** *and* ***can have duplicates.***
- It is created using **[ ]** and items inside are separated using a (**,**) comma.
- A list have ***+ve and -ve index*** as well.
- A list can be created in 2 ways that is

```
List1 = [ 1,2,3,4,5 ]
List2 = list( ( 1,2,3,4,5 ) )
```

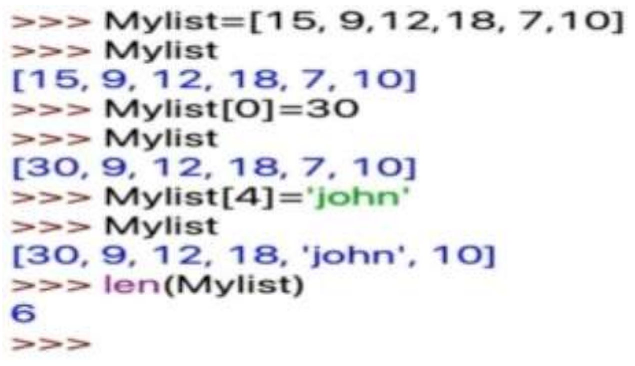


```
>>>
>>> Mylist=['john', 'smith', 'mark', 'eric', 'smith']
>>> Mylist
['john', 'smith', 'mark', 'eric', 'smith']
>>> print(Mylist)
['john', 'smith', 'mark', 'eric', 'smith']
>>> list1=list(1,2,3,4,5)
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    list1=list(1,2,3,4,5)
TypeError: list expected at most 1 argument, got 5
>>>
>>> list1=list((1,2,3,4,5))
>>> list1
[1, 2, 3, 4, 5]
>>> Mylist
['john', 'smith', 'mark', 'eric', 'smith']
>>> Mylist[2]
'mark'
>>> Mylist[-2]
'eric'
>>> |
```

- List is ***heterogeneous*** i.e.; it can contain different type of data in it.

  Example: Mylist = ['John', 15, 14.6, True, 'Steven', 5+7j]

- List is ***mutable*** [changeable], you can change any value in a list.
- ***len()*** given length of a list.



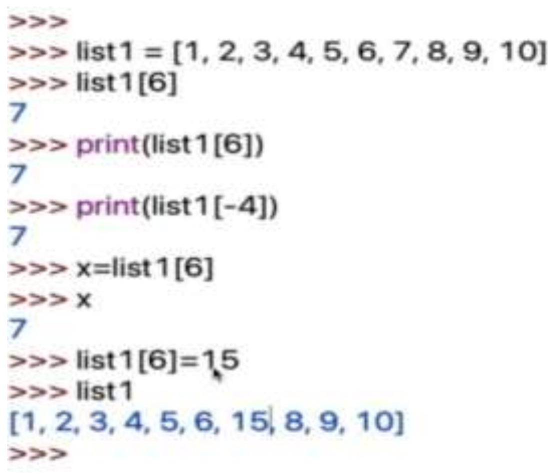
```
>>> Mylist=[15, 9,12,18, 7,10]
>>> Mylist
[15, 9, 12, 18, 7, 10]
>>> Mylist[0]=30
>>> Mylist
[30, 9, 12, 18, 7, 10]
>>> Mylist[4]='john'
>>> Mylist
[30, 9, 12, 18, 'john', 10]
>>> len(Mylist)
6
>>>
```

- ***Append( )*** is used to add more values to a list.

Example: Mylist = [1,2,3,4,5,6]
Mylist.append(50) **O/P:** [1,2,3,4,5,6,50]

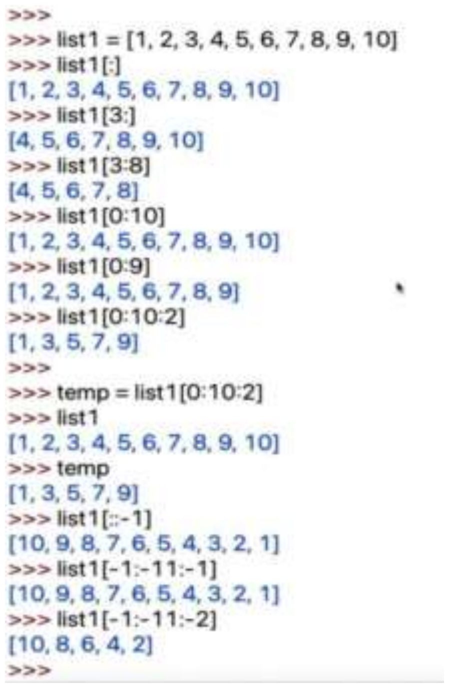
List Operator #1

- The operators on list discussed here are **[ ] , [:].**
- **[ ]** is used for ***indexing*** , Both ***positive and negative indexing*** is possible on list through indexing we can ***read the data of list as well as write / modify*** a list.

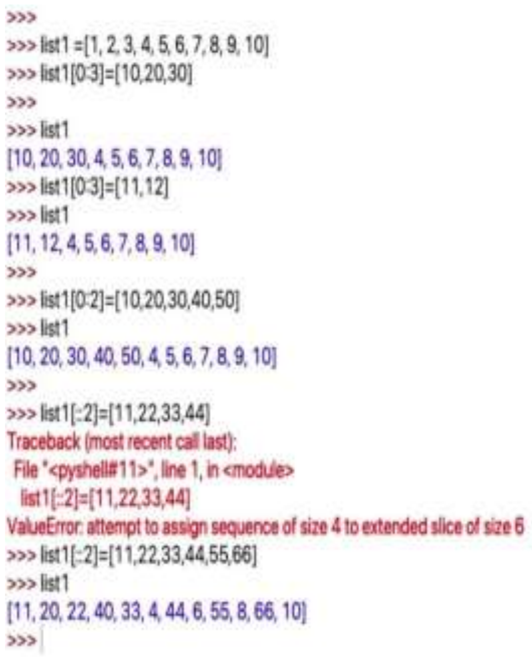


```
>>>
>>> list1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> list1[6]
7
>>> print(list1[6])
7
>>> print(list1[-4])
7
>>> x=list1[6]
>>> x
7
>>> list1[6]=15
>>> list1
[1, 2, 3, 4, 5, 6, 15, 8, 9, 10]
>>>
```

- **[:] -** ***slice / slicing operator*** is similar to index , but we can take a range of values in a list.
- In slicing you can give **[start : end : step size].**
- ***Slicing will give you new list it doesn't modify the existing list***.
- It is used for reading either the complete list or a section of a list.
- It is discussed in detail in the example below



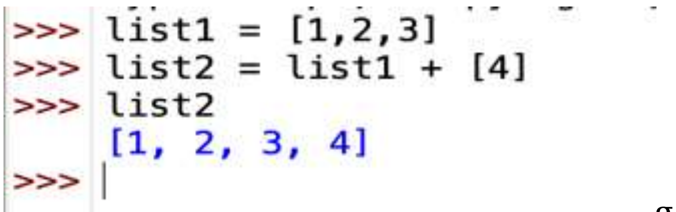
```
>>>
>>> list1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> list1[:]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> list1[3:]
[4, 5, 6, 7, 8, 9, 10]
>>> list1[3:8]
[4, 5, 6, 7, 8]
>>> list1[0:10]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> list1[0:9]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list1[0:10:2]
[1, 3, 5, 7, 9]
>>>
>>> temp = list1[0:10:2]
>>> list1
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> temp
[1, 3, 5, 7, 9]
>>> list1[::-1]
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
>>> list1[-1:-11:-1]
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
>>> list1[-1:-11:-2]
[10, 8, 6, 4, 2]
>>>
```



```
>>>
>>> list1 =[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> list1[0:3]=[10,20,30]
>>>
>>> list1
[10, 20, 30, 4, 5, 6, 7, 8, 9, 10]
>>> list1[0:3]=[11,12]
>>> list1
[11, 12, 4, 5, 6, 7, 8, 9, 10]
>>>
>>> list1[0:2]=[10,20,30,40,50]
>>> list1
[10, 20, 30, 40, 50, 4, 5, 6, 7, 8, 9, 10]
>>>
>>> list1[::2]=[11,22,33,44]
Traceback (most recent call last):
 File "<pyshell#11>", line 1, in <module>
  list1[::2]=[11,22,33,44]
ValueError: attempt to assign sequence of size 4 to extended slice of size 6
>>> list1[::2]=[11,22,33,44,55,66]
>>> list1
[11, 20, 22, 40, 33, 4, 44, 6, 55, 8, 66, 10]
>>>
```

- '+' is also known as ***concatenation operator.***
- In list '+' will join two list to make a new list which contains all the elements of two list.
- The '+' doesn't modify the existing list it just simply generates a separate list I.e., a new list.
- If you want to add an element to a list you need to first make it as a list then concatenate itwith the list in which you want to add it.

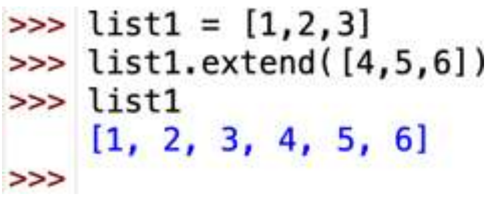
Example:



```
>>> list1 = [1,2,3]
>>> list2 = list1 + [4]
>>> list2
    [1, 2, 3, 4]
>>>
```

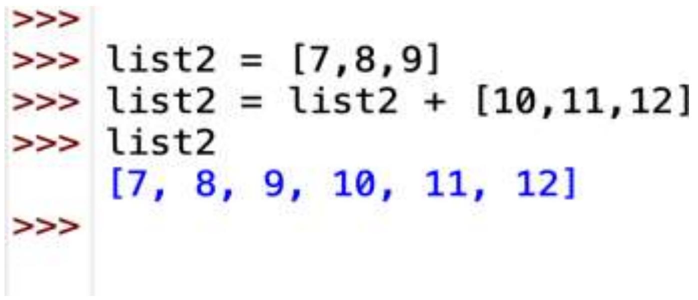
# adding 4 to list1 and storing this in another list i.e. list2.

- If you want to expand/ modify an existing string then you can use a method called **extend( )** to include more elements.
- By using **extend( )** you can change a list.



```
>>> list1 = [1,2,3]
>>> list1.extend([4,5,6])
>>> list1
    [1, 2, 3, 4, 5, 6]
>>>
```

- You can modify the list as follows as well

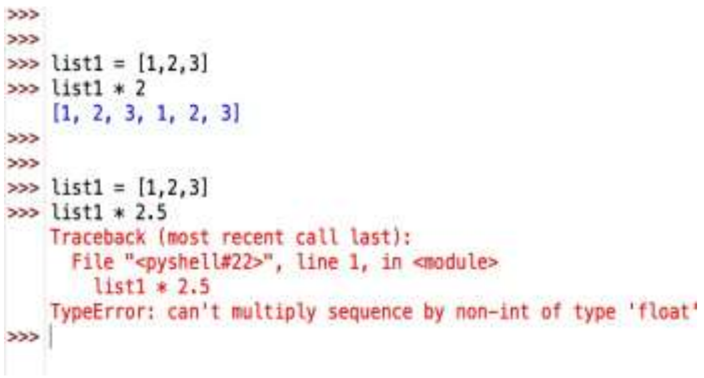


```
>>>
>>> list2 = [7,8,9]
>>> list2 = list2 + [10,11,12]
>>> list2
    [7, 8, 9, 10, 11, 12]
>>>
```

Here, list2 is modified simply, by assigning the old list2 to new list2 and concatenating old list2 with another list hence, modified new list2 is created by concatenation.

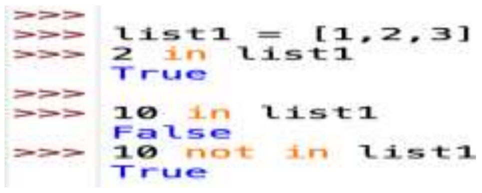
- ‘ * ’ is a ***repetition operator.***
- Float values cannot be used for repetition only integer values should be used.

Example:



```
>>>
>>>
>>> list1 = [1,2,3]
>>> list1 * 2
    [1, 2, 3, 1, 2, 3]
>>>
>>>
>>> list1 = [1,2,3]
>>> list1 * 2.5
    Traceback (most recent call last):
      File "<pyshell#22>", line 1, in <module>
        list1 * 2.5
    TypeError: can't multiply sequence by non-int of type 'float'
>>> |
```

- The other two operators are ***in, not in.***
- They check if an element is present in the list or not and return Boolean type of result.
- For ***in operator***, if it is present then it will return true if not than it will return false.
- For ***not in operator***, if it is present then it will return false if not than it will return true.
- Let us see this with an example.



```
>>>
>>> list1 = [1,2,3]
>>> 2 in list1
    True
>>>
>>> 10 in list1
    False
>>> 10 not in list1
    True
```

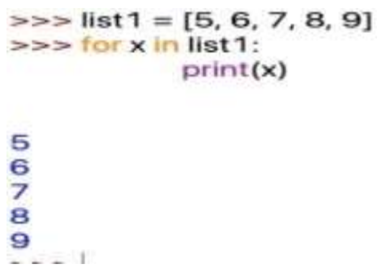
List Iteration

Iterating a list

list1=

0	1	2	3	4
5	6	7	8	9
-5	-4	-3	-2	-1

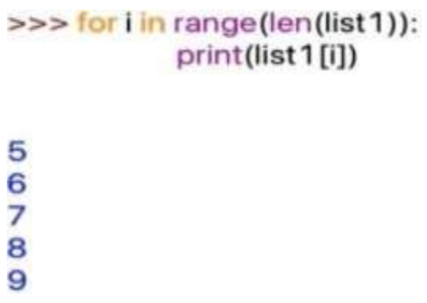
- Iterating a list means visiting a list or accessing all the list elements in a list one by one. We can also call it as traversing.
- We can go reverse also.



```
>>> list1 = [5, 6, 7, 8, 9]
>>> for x in list1:
            print(x)


5
6
7
8
9
```

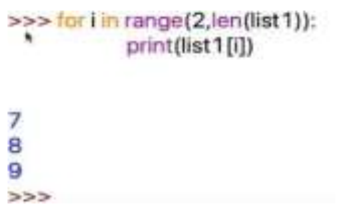
- Iteration is done using for loop also.
- Let's see some examples to make a list

For loop using range



```
>>> for i in range(len(list1)):
            print(list1[i])


5
6
7
8
9
```

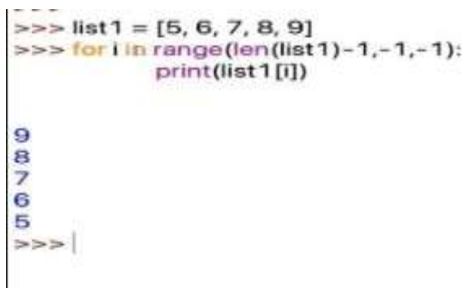
If want to Start from 2



```
>>> for i in range(2,len(list1)):
            print(list1[i])


7
8
9
>>>
```

- It is not printing element at 0 and 1 that is 5 and 6 it is starting printing from third element that is 7.

- If we want to print numbers in reverse so, we can print using positive indices also negative indices also.



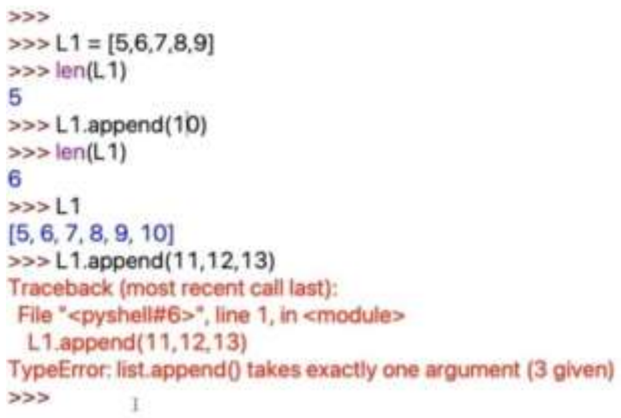
```
>>> list1 = [5, 6, 7, 8, 9]
>>> for i in range(len(list1)-1,-1,-1):
            print(list1[i])


9
8
7
6
5
>>> 
```

List Method #1

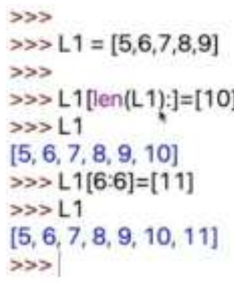
- Methods are member function of a class.
- The methods discussed here are **append(x), extend (iterable), insert (i, x), copy ( ).**

append(x)

- It means adding one element at a time to a list.
- It modifies the same list.
- We can use slicing on append method.



```
>>>
>>> L1 = [5,6,7,8,9]
>>> len(L1)
5
>>> L1.append(10)
>>> len(L1)
6
>>> L1
[5, 6, 7, 8, 9, 10]
>>> L1.append(11,12,13)
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    L1.append(11,12,13)
TypeError: list.append() takes exactly one argument (3 given)
>>>
```

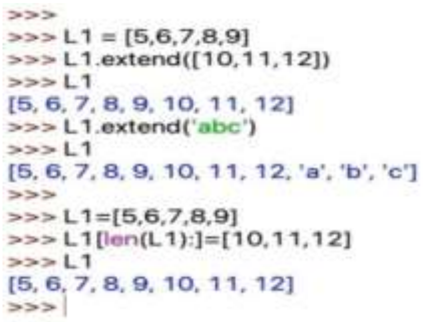


```
>>>
>>> L1 = [5,6,7,8,9]
>>>
>>> L1[len(L1):]=[10]
>>> L1
[5, 6, 7, 8, 9, 10]
>>> L1[6:6]=[11]
>>> L1
[5, 6, 7, 8, 9, 10, 11]
>>> 
```

extend (iterable)

- It is same as append but it can take collection of elements in parameter.
- It will modify the same list.
- Extend can take list, tuple and string as parameter because they are iterable as well.

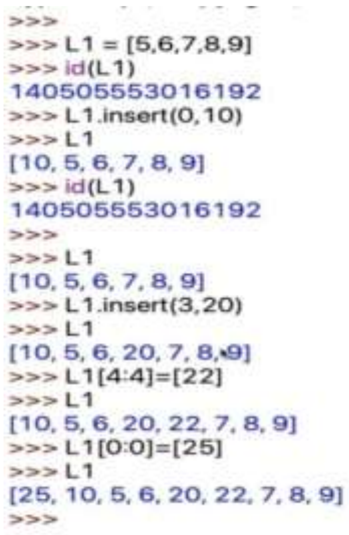
- Slicing is also applicable on extend method.



```
>>>
>>> L1 = [5,6,7,8,9]
>>> L1.extend([10,11,12])
>>> L1
[5, 6, 7, 8, 9, 10, 11, 12]
>>> L1.extend('abc')
>>> L1
[5, 6, 7, 8, 9, 10, 11, 12, 'a', 'b', 'c']
>>>
>>> L1=[5,6,7,8,9]
>>> L1[len(L1):]=[10,11,12]
>>> L1
[5, 6, 7, 8, 9, 10, 11, 12]
>>>
```

insert (i, x)

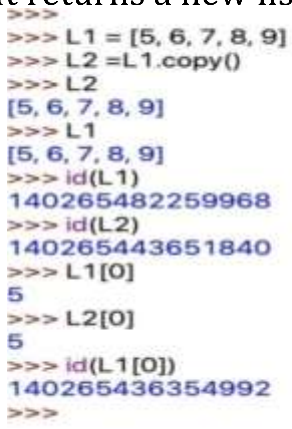
- It is used to insert any element at a given index.
- Same list is modified here.
- Slicing is applicable on insert method.



```
>>>
>>> L1 = [5,6,7,8,9]
>>> id(L1)
140505553016192
>>> L1.insert(0,10)
>>> L1
[10, 5, 6, 7, 8, 9]
>>> id(L1)
140505553016192
>>>
>>> L1
[10, 5, 6, 7, 8, 9]
>>> L1.insert(3,20)
>>> L1
[10, 5, 6, 20, 7, 8, 9]
>>> L1[4:4]=[22]
>>> L1
[10, 5, 6, 20, 22, 7, 8, 9]
>>> L1[0:0]=[25]
>>> L1
[25, 10, 5, 6, 20, 22, 7, 8, 9]
>>>
```

Copy

- It copy( ) will create a shallow copy of the list.
- It returns a new list after copying it.



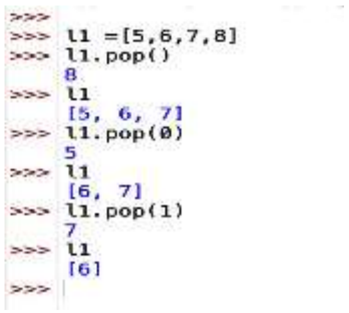
```
>>>
>>> L1 = [5, 6, 7, 8, 9]
>>> L2 =L1.copy()
>>> L2
[5, 6, 7, 8, 9]
>>> L1
[5, 6, 7, 8, 9]
>>> id(L1)
140265482259968
>>> id(L2)
140265443651840
>>> L1[0]
5
>>> L2[0]
5
>>> id(L1[0])
140265436354992
>>>
```

List Methods #2

- The methods used to remove elements from a list are
    - pop ([])
    - remove (x) and
    - clear ()

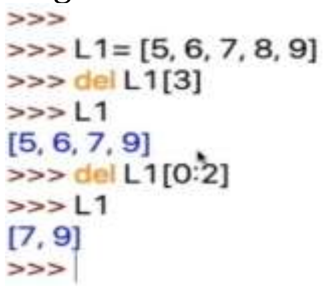
pop ([])

- It will delete the last element from a list if index is not given.
- By mentioning any index of a list in pop() that particular element will be deleted.



```
>>>
>>> l1 =[5,6,7,8]
>>> l1.pop()
8
>>> l1
[5, 6, 7]
>>> l1.pop(0)
5
>>> l1
[6, 7]
>>> l1.pop(1)
7
>>> l1
[6]
>>>
```

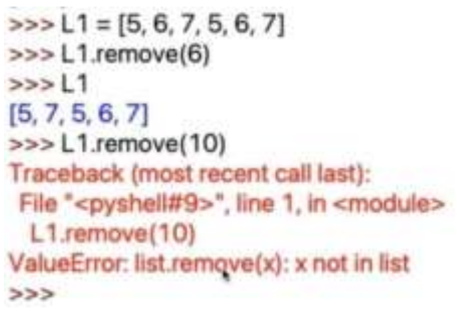
- The **del** keyword is used to delete an element from a list, slicing works for del keyword, we can also delete a particular element by mentioning the index number.



```
>>>
>>> L1= [5, 6, 7, 8, 9]
>>> del L1[3]
>>> L1
[5, 6, 7, 9]
>>> del L1[0:2]
>>> L1
[7, 9]
>>>
```

remove(x)

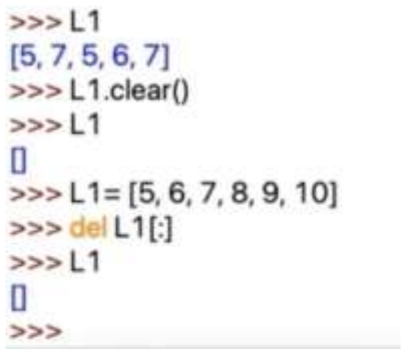
- It searches a particular element from a list and remove it if not found it gives an error.
- If duplicates are present then it removes the 1st occurrence of the duplicate from the list.



```
>>> L1 = [5, 6, 7, 5, 6, 7]
>>> L1.remove(6)
>>> L1
[5, 7, 5, 6, 7]
>>> L1.remove(10)
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    L1.remove(10)
ValueError: list.remove(x): x not in list
>>>
```

clear()

- It will clear the entire list.
- Even using slicing, we can clear a list.
- Delete can also be used to delete a list.

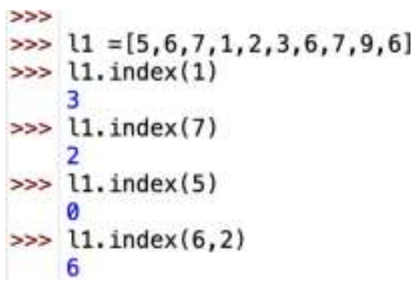


```
>>> L1
[5, 7, 5, 6, 7]
>>> L1.clear()
>>> L1
[]
>>> L1= [5, 6, 7, 8, 9, 10]
>>> del L1[:]
>>> L1
[]
>>>
```

List Methods #3

- The syntax of index method is index (x [, start [, end]])
- The index () is used to search a particular element/item in a list and when found it returns its index.
- By default, index takes starting and ending value as zero if the index is not defined.
- The parameter [] (start, end) are optional in index method.
- The start and end value in index are mostly given when you want to know the index value of the duplicate element because when value is not given it takes default value as 0 and returns the index value of first occurrence of the element.

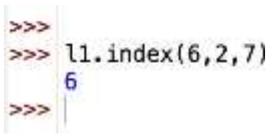
Example:



```
>>>
>>> l1 =[5,6,7,1,2,3,6,7,9,6]
>>> l1.index(1)
    3
>>> l1.index(7)
    2
>>> l1.index(5)
    0
>>> l1.index(6,2)
    6
```

#Return the index of 1 I.e., 3

#Return the index of 6 there indexing is starting from 2

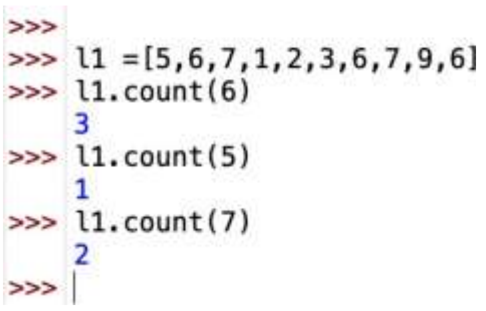


```
>>>
>>> l1.index(6,2,7)
    6
>>> |
```

# Number to be search is 6 in range of index from 2 to 7

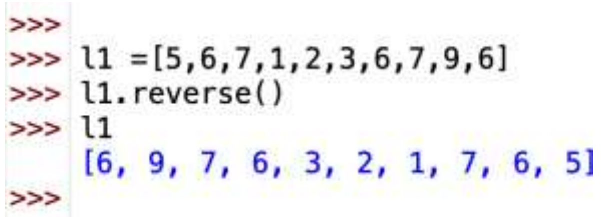
- When you give an ending point in indexing, they will stop one step before that value. If element is not found then we'll get an error.

- The **count(x)** method is used to know how many times a particular value/element is appearing in a list. It will not give the index value it will count the given number for example.



```
>>>
>>> l1 =[5,6,7,1,2,3,6,7,9,6]
>>> l1.count(6)
    3
>>> l1.count(5)
    1
>>> l1.count(7)
    2
>>> |
```

- When the **reverse ()** method is called it simply reverse the contents of a list.
- The syntax for sort method is `sort` (*, key = none, reverse = false)

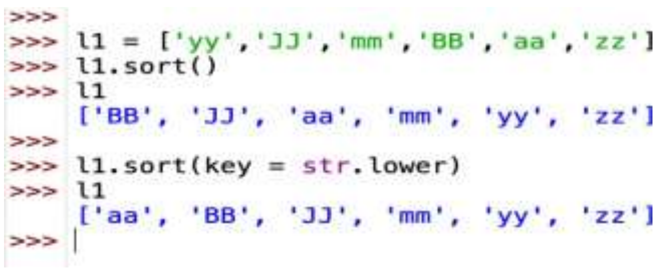


```
>>>
>>> l1 =[5,6,7,1,2,3,6,7,9,6]
>>> l1.reverse()
>>> l1
    [6, 9, 7, 6, 3, 2, 1, 7, 6, 5]
>>>
```

- The sort ( ) sort the elements in the list and you'll get the elements of the list In sorted order.
- understanding the working of sort ( )

Sort (* , key = none , reverse = false)

Key = none - here we are defining our own criteria / function Reverse = false-reverse is key and false is value



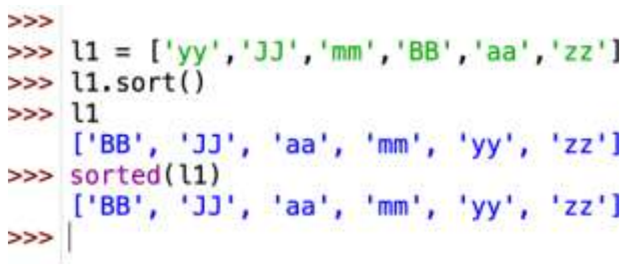
```
>>>
>>> l1 = ['yy','JJ','mm','BB','aa','zz']
>>> l1.sort()
>>> l1
    ['BB', 'JJ', 'aa', 'mm', 'yy', 'zz']
>>>
>>> l1.sort(key = str.lower)
>>> l1
    ['aa', 'BB', 'JJ', 'mm', 'yy', 'zz']
>>> |
```

#Before applying key = none

#after applying key =str.lower

- Upper case letter is always treated as smaller than lower case letter [based on ASCII value it happens].

- There's a global function called **sorted ( )** .This function will modify the original list it gives a new sorted list.



```
>>>
>>> l1 = ['yy','JJ','mm','BB','aa','zz']
>>> l1.sort()
>>> l1
    ['BB', 'JJ', 'aa', 'mm', 'yy', 'zz']
>>> sorted(l1)
    ['BB', 'JJ', 'aa', 'mm', 'yy', 'zz']
>>> |
```

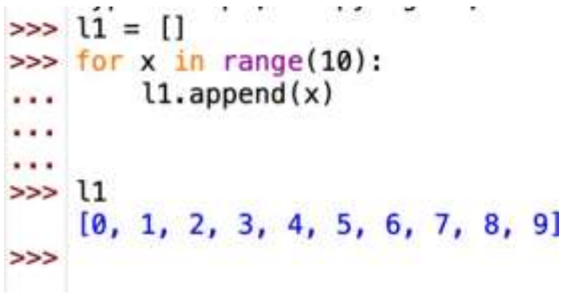
#Modifies the given list I.e., sort it

#Not modify list but create a new sorted list.

List Comprehension

- It is a method to create a list in a simple way from existing list (or) other iterables.
- A list can be generated from a list, tuple, string or a set.
- A list can be empty, it is denoted as [ ]
- ***append( )*** in list adds an element to a list.
- A loop can be used to append values in a list of particular range.

Example 1:



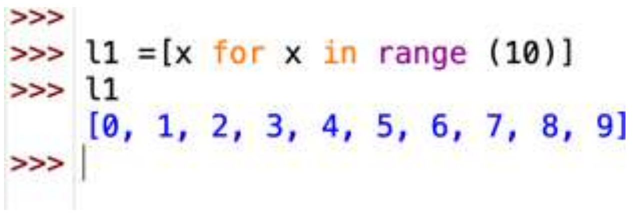
```
>>> l1 = []
>>> for x in range(10):
...     l1.append(x)
...
...
>>> l1
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>
```

- The above process can be done in a simple way using. 'List Comprehension'.

Syntax of List Comprehension:

L1 = [Expression for item in iterable]

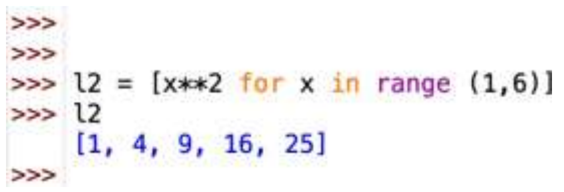
Example 2:



```
>>>
>>> l1 =[x for x in range (10)]
>>> l1
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>
```

- We can observe from this example that how using list comprehension we simplified using example 1.
- Some other examples of list comprehension are

Example 3:

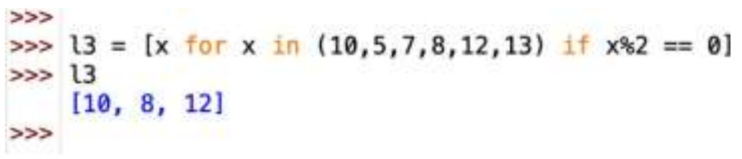


```
>>>
>>>
>>> l2 = [x**2 for x in range (1,6)]
>>> l2
    [1, 4, 9, 16, 25]
>>>
```

#using Expression

# Here we are using an expression I.e for numbers in range 1 to 6 which give its square value.

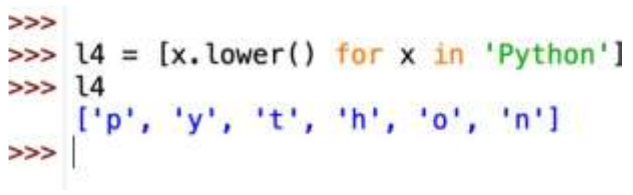
Example 4:



```
>>>
>>> l3 = [x for x in (10,5,7,8,12,13) if x%2 == 0]
>>> l3
    [10, 8, 12]
>>>
```

# We are defining a tuple and writing a condition that says for given numbers in the tuple, divide it by 2 give the remainder of those numbers divisible by 2 as result.

Example 5:

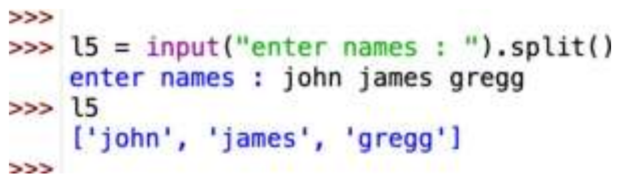


```
>>>
>>> l4 = [x.lower() for x in 'Python']
>>> l4
    ['p', 'y', 't', 'h', 'o', 'n']
>>> |
```

# Here x.lower( ) is an expression which means take all the string letters in lower case.

- You can also read a list from a keyboard
- Suppose you want to take input from keyboard (which is str type) and want to convert it into list then this can also be done in a simplified way using list comprehension.

Example 6:



```
>>>
>>> l5 = input("enter names : ").split()
    enter names : john james gregg
>>> l5
    ['john', 'james', 'gregg']
>>>
```

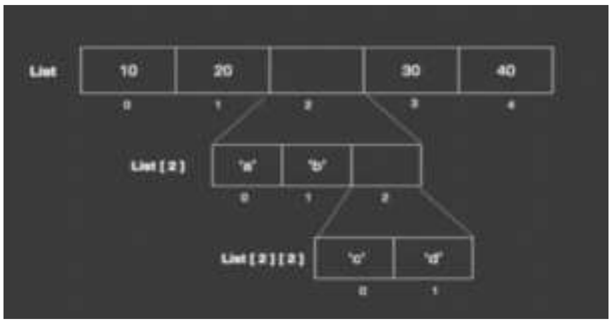
# Input from keyboard and splitting them using space.
#entering values
#output as list.

Nested List

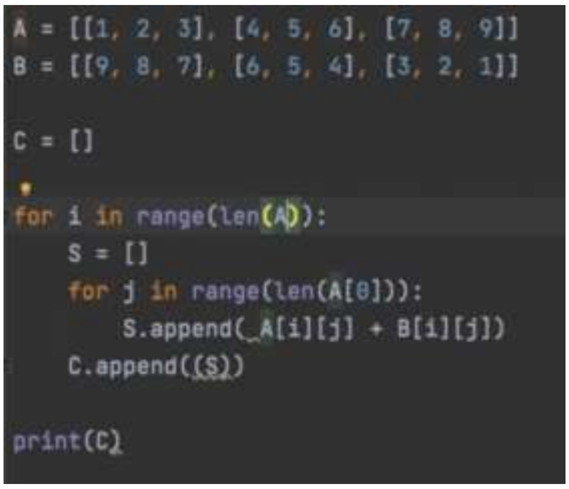
- A list can have heterogeneous elements like int, float etc.
- A list can have a list as an element inside it this is called nested list for example

Example: **list = [10, 20 [‘ a ’, ‘ b ’ [‘ c ’, ‘ d ’] , 30 , 40]]**

- Diagrammatically it can be represented as

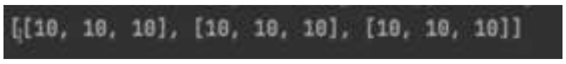


- If you are having nested list, you can prepare a matrix also with same type of values. Let’s, see this in a program.



```
A = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
B = [[9, 8, 7], [6, 5, 4], [3, 2, 1]]

C = []

for i in range(len(A)):
    S = []
    for j in range(len(A[0])):
        S.append( A[i][j] + B[i][j])
    C.append((S))

print(C)
```

# A, B are 2 list in which nested list resides. On these 2 lists we are performing addition and appending the results in C and printing the result.



```
[[10, 10, 10], [10, 10, 10], [10, 10, 10]]
```

- Just like + you can also perform -, * on a matrix

Q 1) To Calculate Salary, weekly working hours given in a list.

Program:

```
work_hours = [int(x) for x in input('Enter hours per day in entire week, separated by
space ').split()]
wage = int(input('Enter hourly wage ')) total
= sum(work_hours)
print(total)
salary = total * wage
print('Salary is',salary)
```

Q 2) Checking and removing if there is any duplicate in a List.

Program:

```
L1 = [3, 5, 7, 9, 3, 6, 5, 2, 3, 7, 10] L2 = []
for x in L1:
    if x not in L2:
        L2.append(x)
print(L2)
```

Q 3) Concatenate all integer from a list to a single number.

Program:

```
L1 = [3, 5, 12, 6, 4] L2=''
for i in L1: L2=L2+str(i)
print(L2)
```

Q 4) To convert a string into a list, as in list (‘abc‘) gives [a, b, c].

Program:

```
str=input("Enter the string: ")
list=str.split()
print(list)
```

Q 5) Addition of two matrix.

Program:

```
L1 = [[1, 2, 3, 4], [1, 2, 3, 4], [1, 2, 3, 4]]
L2 = [[5, 6, 7, 8], [5, 6, 7, 8], [5, 6, 7, 8]] L3 = []
for i in range(3): s
    = []
    for j in range(4):
        r = L1[i][j] + L2[i][j]
        s.append(r)
    L3.append(s)
for x in L3:
    print(x)
```

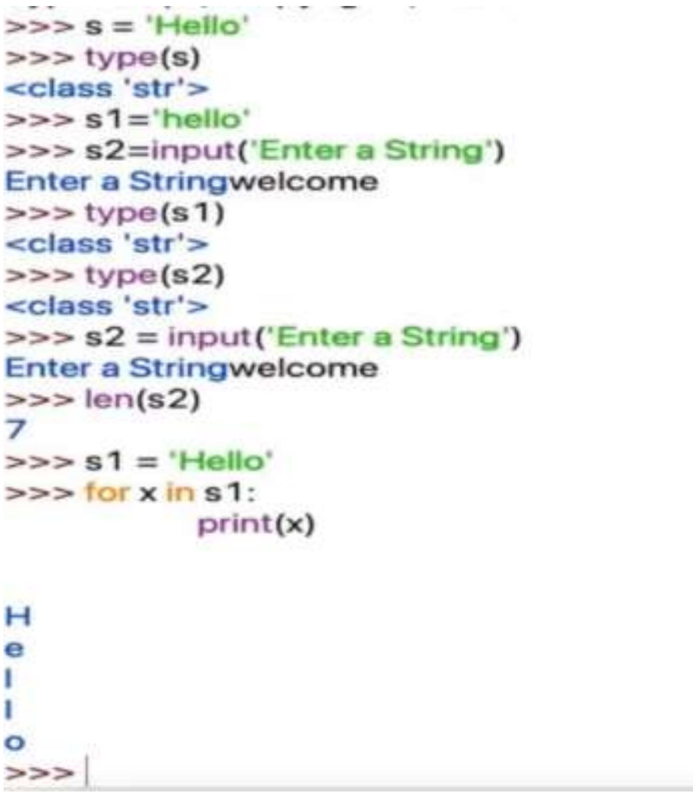
Q 6) Transpose of a matrix.

Program:

```
L1 = [[1, 2, 3, 4], [1, 2, 3, 4], [1, 2, 3, 4]] L2 = []
for i in range(4): s
    = []
    for j in range(3):
        s.append(L1[j][i])
    L2.append(s)
for x in L2:
    print(x)
```

Introduction **to** *String*

- ✓ Anything written inside **‘’, “”** in quotes is called string.
- ✓ String can contain English alphabet, special characters like **@, -, \, /, #, (,),** it can also contain characters from other languages also.
- ✓ In memory strings looks like an ***array of Character*** it has +ve and -ve indexing as well.
- ✓ When you call an input function normally it'll give string by default.
- ✓ ***len ()*** gives you length of a string.
- ✓ We can use ***for loop*** for accessing every element in a string.


```
>>> s = 'Hello'
>>> type(s)
<class 'str'>
>>> s1='hello'
>>> s2=input('Enter a String')
Enter a Stringwelcome
>>> type(s1)
<class 'str'>
>>> type(s2)
<class 'str'>
>>> s2 = input('Enter a String')
Enter a Stringwelcome
>>> len(s2)
7
>>> s1 = 'Hello'
>>> for x in s1:
                print(x)


H
e
l
l
o
>>> 
```

- ✓ When a string value is given directly in the program then it’s called ***string literals.***
- ✓ String literals can be in **‘ ’ , “ ” , ‘’ ‘’ , “’” “’”**
- ✓ When a string already contains a single quote (inner quotes) then you should enclose thestring in double quotes (outer quotes) and vice versa.

Example: s = “ John ’ s ”

or

S = ‘ John ” s ’

If a string is in multiple lines, then you have to use ***triple single quotes (or)*** *triple double quotes*

Example: ' ' ' hello
How are

you '''

or

" " " hello
How are you " " "

Operators on Strings

- The operators that work on strings are
    - ***Concatenation***
    - ***Repetition***
    - ***Indexing***
    - ***Slicing***
    - ***In***
    - ***Not in***

- ***Concatenation:***

- In python we can join / concatenate 2 strings using **+**

- **Example:**
  s1 ='hello'
  s2 = 'world'
  s3 = s1 + s2 →'helloworld'

- Because string is immutable it will not modify it. It will create a new string.

- When concatenation the 2 strings must be of string datatype only, if one is string and the other is integer this will not work.

- ***Example:***

  s1 ='hello' **s2** = 15
  s3=s1+s2 // error

To add a string and a number we must type case the integer value then adds it.

- ***Example:***

    s1 ='hello' **s2** = str(15)

    s3=s1+s2 // (o/p) hello15

- Repetition:

- We can multiply a string with integer numbers

- Example:

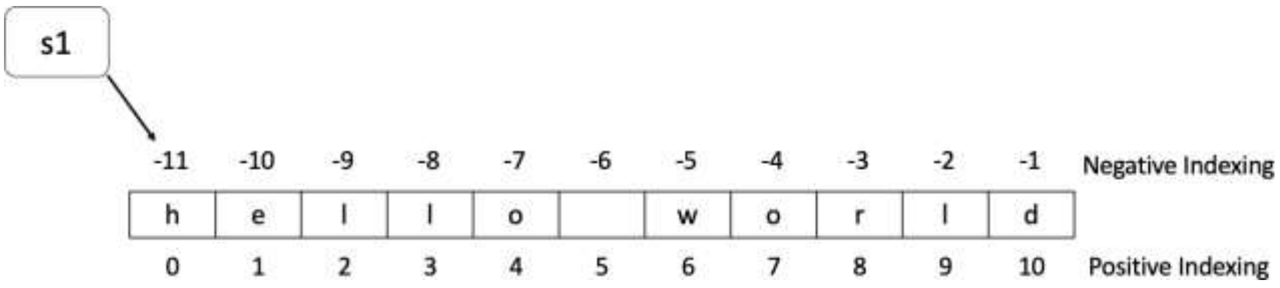
    s1 ='hi' * 3 // (O/P) hihihi

- Repetition only works with a string and integer value only.
- Float numbers cannot be used here.
- ***Example:***

    ' hello ' * 2.5 // error

- Indexing:

- Suppose we are having a string s1 ='hello world ', it looks like this with indexing



s1
-11 -10 -9 -8 -7 -6 -5 -4 -3 -2 -1 Negative Indexing
h e l l o w o r l d
0 1 2 3 4 5 6 7 8 9 10 Positive Indexing

- It has both +ve and -ve indexing values
  - You can access any character of the string using index

  values. s1[0] —— h

  s[-11] —— h

  s1[4] —— o

  s1[6] —— w

  s1[-6] —— blank space

- Slicing:

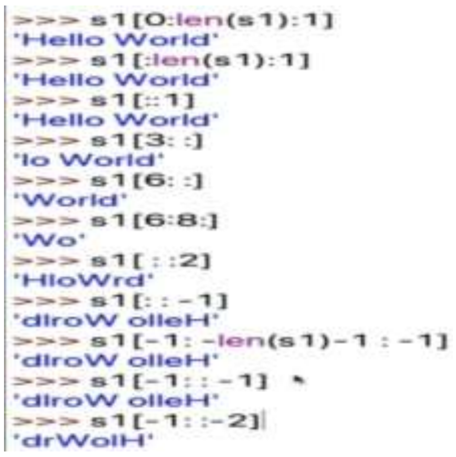
- We can access values of strings using index.
- Another way of accessing elements in a string is through slicing

- The syntax for slicing a string is

s1[start: stop: step]

- Using slicing, we can extract specific potion of a string.

Example:



```
>>> s1[0:len(s1):1]
'Hello World'
>>> s1[:len(s1):1]
'Hello World'
>>> s1[::1]
'Hello World'
>>> s1[3: :]
'lo World'
>>> s1[6: :]
'World'
>>> s1[6:8:]
'Wo'
>>> s1[ : :2]
'HloWrd'
>>> s1[: : -1]
'dlroW olleH'
>>> s1[-1: -len(s1)-1 : -1]
'dlroW olleH'
>>> s1[-1:: -1]
'dlroW olleH'
>>> s1[-1::-2]
'drWolH'
```

- in:

- It will say if a character is present in the string or not.
  - If it is present then it will return True or else False.

Example:

s1 = 'hello world'

h in s1

—True world

in s1

—True me

in s1

— False

- not in:

  - It will say TRUE if the character is not present

Example:

s1 = 'hello world'

me not in s1 —— True world

not in s1 —— False

- ***in, not in*** is also called as Membership operator.
- ***in, not in*** make case sensitive comparison. (It considers lower and upper case differently.

Introduction to string methods

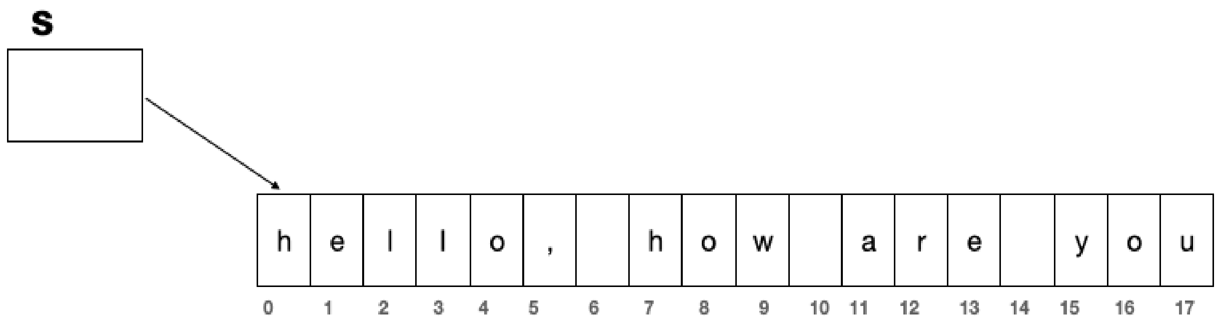
- ✓ The string is a built - in class in python.
- ✓ Classes contains definition / design (there are of 2 types data and operation
  / methods/function)
- ✓ Objects is an instance of that definition design.
- ✓ In string many members function is available these member functions are called ***string methods.***
- ✓ To know all the methods, present in string class we can type a function called dir(str) it shows all the method.
- ✓ You can access all the methods using dot (.) operator (or) just wants to know about particular method.
- ✓ Example:
  s = ' hello'
  help(s.endswith)

String Method #1

• Methods are the member of the class which performs operation upon the data of an object.

S = " hello , how are you "

This is how String is stored.



- It has 18 characters and it is hold by a reference that is s
- ***s.find (sub [, start [,end]])***
- ***sub*** (find the occurrence of the substring)
- Methods are called by using object name that is variable name.

s.find('o')

It will start looking for o from the left hand of the string. The result is 4 cause we found ' o ' there

s.find (' how ')

The result will be 7

s.find(' k ')

- The result will be -1. Why it is -1 because the character will start from 0.

  -1 means outside the range. So, it is invalid position.

s.find (' o ' , 5)

$\swarrow \quad \searrow$

sub start

- If you want to find out after the 5th index, we write 5 in starting index.
- If you want to find out other substring then you should give starting and ending index.

s.find ('o' , 5 , 7)

It will check from 5 to 7

- You can pass the find by single, double or 3 parameters.

s.rfind (sub [, start [,end]])

- It is same as find but in ***find we were searching from left*** but in ***rfind will search substring from right.***
  - s.rfind('o') ——— 16 index

- s.find ('o ' , 0 , 15) ——— 8 index
- In rfind ending index will work but in find starting index will work
- -1 will return if it's out of string.

s.index ( )

- s.index and s.find is same but have minor differences . rindex is same as rfind.

s.count( )

• It will count the number of occurrences. Let's take an example - character ' o '

s.count("o")----------------→it repeated 3 times

• The count gives counting of the string. It will not give all the indexes. It will only count

s.find (' k ')--------------→ -1

s.index (' k ') ....→ substring not found

s.rindex('o ' , 0 , 15)----------------→ 8

s.count ('me')-----------------→ 0

s.count('how') --------------- → 1

String Method #2

s.ljust(width[,fill])

s.rjust(width[,fill])

s.center(width[,fill])

- This methods are useful for text alignment. s = "Python"
- If you want to write in extra spaces like 10

spaces s.ljust ( )

```
P  y  t  h  o  n  _  _  _  _
```

s.rjust ( )

```
_  _  _  _  P  y  t  h  o  n
```

s.center ( )

```
__  __  P  y  t  h  o  n  __  __
```

- If the string is larger than the spaces given by
  using s.ljust(3)
- It will take the entire string it will not just take 3 letters.
- If you want bigger space, you mention the width bigger than the length of the string.
- All this have one more parameter that is fill.

  s.center(10 , ' * ')

- Python have only 6 alphabets but we want 10 spaces. '*' Will be filled in empty spaces.
- 10 space vacant spaces with * otherwise it will fill with
  spaces s = python
- String is immutable so it will not modify it will create a new string.

s.strip ([chars])

s.lstrip ([chars]) ***this is useful for removing the characters from the string***

s.rstrip ([chars])

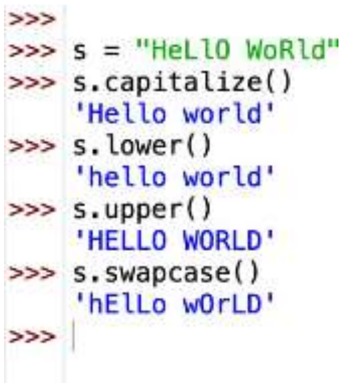
- They remove leading char, tailing characters and characters from both sides ... by default theywill remove spaces.
- s.lstrip - it will remove leading char
- s.rstrip - tailing removes character
- s.strip - removes spaces from both the side

S = '...++aaaapython '

- s. lstrip (' . ') ———> it will remove leading dots and stops when there is no dot
  - O/p - '.........................++aaaapython '
- s.lstrip(' .   + ') ———> it will remove dot spaces and +
  - O/p - 'aaaapython'
- All these methods will return new string they will generate new string after performing theoperations.

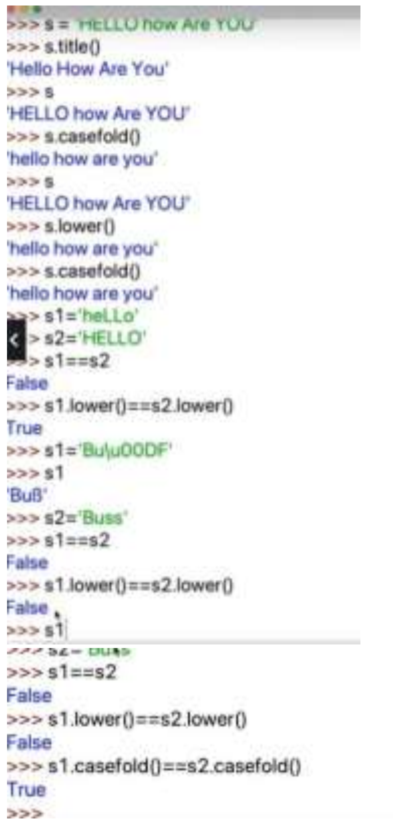
String Methods #3

- The methods we are discussing here are
  - ***s.capitalize( )***
  - ***s.lower( )***
  - ***s.upper( )***
  - ***s.title( )***
  - ***s.swapcase( )***
  - ***s.casefold( )***
- Some methods are discussed below



```
>>>
>>> s = "HeLlO WoRld"
>>> s.capitalize()
    'Hello world'
>>> s.lower()
    'hello world'
>>> s.upper()
    'HELLO WORLD'
>>> s.swapcase()
    'hEllo wOrLD'
>>>
```

- **s.title( )** , here first letter of every word in a string becomes capital.
- **s.casefold( )** , It converts string into lowercase letters as s.lower( ) but there are some difference between them.



```
>>> s = 'HELLO how Are YOU'
>>> s.title()
'Hello How Are You'
>>> s
'HELLO how Are YOU'
>>> s.casefold()
'hello how are you'
>>> s
'HELLO how Are YOU'
>>> s.lower()
'hello how are you'
>>> s.casefold()
'hello how are you'
>>> s1='heLLo'
>>> s2='HELLO'
>>> s1==s2
False
>>> s1.lower()==s2.lower()
True
>>> s1='Bu\u00DF'
>>> s1
'Buß'
>>> s2='Buss'
>>> s1==s2
False
>>> s1.lower()==s2.lower()
False
>>> s1
>>> s2='Buss'
>>> s1==s2
False
>>> s1.lower()==s2.lower()
False
>>> s1.casefold()==s2.casefold()
True
>>>
```

s.isupper()

s.islower( )

s.istitle ( )

s.isalnum ( )

s.isalpha ( )

s.isspace ( )

s.isascii ( )

• It will return in Boolean result (True or False)

s.isupper( )

s = ‘ HELLO ‘

- It will return true if only all the alphabets are in capital letters

s.islower( )

s = ‘ hellO ‘

- It will return false because all should be in lower case but in this O is capital so its false.

s.istitle ( )

s = 'How Are You '

- Every starting letter should be capital so it will return True and empty string will return False.

s.isalnum ( )

- If the string contains alphabets and numbers, it will return true. If it contains special alphabets than it will return false

  s = 'abc-123' — — > False

s.isalpha ( )

- If it is only alphabets then it will return

  true s = 'abcd '—>True

  s = 'abcd123' — — > False

This alpha and album works on any language.

s.isspace ( )

- It will check if there are any spaces present in the string then it will return true.

  s = 'hello world '——> True

s.isascii ( )

- If it contains all the ASCII character, can have lower case, upper case, special character etc., then it will return True

  ' abc12#! ' ——> True

  s ——> True #empty string will return True.

s.isidentifier( )

s = ' length1 ' ——> True s = '

1length ' ——> False

- Variable cannot Start with a number so it will return false.
- This method will check if it gives the valid name of the variable.
- For keywords also it will return

  true s = ' if ' ——> True

s.isprintable( )

- All the ASCII letters, other language letters are printable but there are

escape characters \n \t \r are not printable so it will return false. s = ‘ hello ‘ ——> True
s = ‘ hello \n how are you ‘ ——> False

s.isdecimal( )

- It will say there are decimal present in the string

s = ‘125‘ ——> True
s = ‘1.25‘ ——>False

What are decimal?? Decimal are the numbers between 0 - 9

s.isdigit( )

s = ‘16^2‘——> True

- If you have special numbers, it will return true.
- But in s.isdecimal ( ) it will return False

s.isnumeric( )

s = “$5^{2/3}$” ——> True

- But in decimal and digit it will be false.
- It will return true to any number.

s = ‘ python is very easy ‘

s.startswith(prefix [, start [, end]])

- Start and end are in square brackets so it is optional.
- It will say whether a given string is starts

with. s.startswith (‘ python ‘) ——>

True s.startswith (‘is’ , 7) ——> True

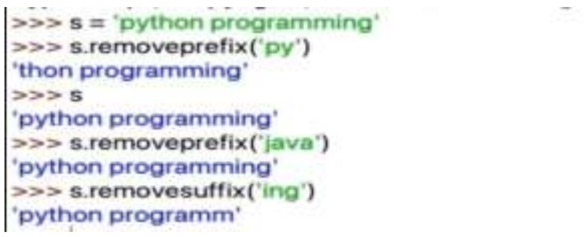
s.endswith(prefix [, start [, end]])

- It will say if the string ends with (certain string) then it will return

True s.endswith(‘easy’) ——> True

s.removesuffix(suffix , /)

- It will remove the substring



```
>>> s = 'python programming'
>>> s.removeprefix('py')
'thon programming'
>>> s
'python programming'
>>> s.removeprefix('java')
'python programming'
>>> s.removesuffix('ing')
'python programm'
```

s.removeprefix(prefix , /)

- It will remove the beginning of the string if it is available and gives the original string
- The two methods (prefix, suffix) will remove and gives back the new string.
  It will not modify the existing string

s.partition(sep)

- It will divide
  s.partition(‘ is ‘)
- It will check where is ‘is’ then it will form a tuple.

```
s.partition( ‘ is ‘ ) ——> ( ‘ python ‘ , ‘is’ , ‘ very easy ‘ )
s.partition ( ‘ s ‘ ) ——> ( ‘ python i’ , ’s’ , ‘ very easy ‘ )
```

s.rpartition(sep)

- It will perform from the right-hand side

```
s.rpartition ( ‘ s ‘ ) ——> ( ‘ python is  very ea ’s’ ’y ‘ )
```

s.replace(old, new,[,count]) s.join(iterable)

s.split([sep[, max split]) s.rspli([sep[, max

split]) s.splitlines([keepends])

s=‘a-b-c-d-e’ want to replace all these hyphens with commas using these methods

s.replace(‘-’ ‘,’ 3)

So, the first one is old and the next one is new

‘a, b, c, d-e’->This is a new string modify the old string and generate new string

We can mention the count but it is optional.

```
s='a-b-c-d-e'
s
 'a-b-c-d-e'
 s.replace('-', ‘,’)#replacing it with comma
 'a,b,c,d,e'
 s
 'a-b-c-d-e'
 s.replace('k','m')
 'a-b-c-d-e'
 s='abcd@gmail.com'
 s.replace(‘gmail’,'yahoo')#replacing it with
```

yahoo 'abcd@yahoo.com'

This is the replace method

s.join(iterable)

The join method:
```
s1='xyz'
s2='abc'
s1.join(s2)      # calling s1 method and joining upon s2
'axyzbxyzc'
s1='/'
s1.join(s2)# letters of s1 and s2 as a separator
'a/b/c'
```

So, the parameter is string so it's taking letters of a string.

The split method:

```
s='john smith ajay'
s.split() # it splits

['john', 'smith', 'ajay']
s.split('h')
['jo', 'n smit', ' ajay']
s.split(',')
['john smith ajay']
s='john-smith-ajay-khan-james'

s.split('-')
['john', 'smith', 'ajay', 'khan', 'james']
```

The s. rsplit method:
The s.rsplit means splitting is done from right hand side

s.splitlines works same as split only

s='aaa\n' bbb\t ccc\t

Q 1) To print sorting letters of a string.

```
str1 = 'klmndgfhbpa' ss
= sorted(str1) print(ss)
str2 = ''.join(ss)
print(str2)
```

Q 2) Display data in given format (25 letters)
#Example
Product Name………price
Chicken Pizza………300

```
item = input('Enter The Item') price =
input('Enter price') total_len =
len(item) + len(price) print(total_len)
dots = '.' * (25 - total_len)
print(item+dots+price)
```

Q 3) Check if the password and confirm password are same.

```
pass1 = input('Enter Password') pass2
= input('Confirm Password') if pass1
== pass2:
    print('Yes They are Matching')
else:
    if pass1.casefold() == pass2.casefold(): print('Please
        check for the cases and try again')
    else:
        print('No They are Not Matching Try them again')
```

Q 4) Display credit Card Number
Card Number- 4456 8897 8978 1234
Display - * **** **** 1234***

```
cardno = input('Enter Card No')
lastDigits = cardno[15::]
four = '*' * 4 + ' '
dispno = four * 3 + lastDigits
print(dispno)
```

Q 4) Find user id and domain name from email address.

```
emailid = input('Enter email id')
atrate = emailid.find('@')
print(atrate)
print('user id:', emailid[:atrate]) print('domain
name:', emailid[atrate+1:])
```

Q 5) Checking a string is a palindrome.

```
s1 = input('Enter a string: ') rev
= s1[::-1]
if s1 == rev: print("Palindrome
    ")
else:
    print("Not a Palindrome")
```

Q 6) Convert a given string to palindrome.

```
s1 = input('Enter a string: ') rev
= s1[::-1]
print(s1+rev)
```

Q 7) Find day month and year from date.

```
mydate = input('Enter Date in dd/mm/yyyy format:') l =
mydate.split('/')
print('Day:',l[0])
print('Month:',l[1])
print('Year:',l[2])
```

Q 8) Checking if two string are Anagram.

```
s1 = input('Enter a String')
s2 = input('Enter second String') if
len(s1) != len(s2):
    print('Not Anagram') else:
    for x in s1:
        if x not in s2:  print('Not
            Anagarm') break;
    else:
        print('Anagram')
```

Q 9) Rearrange: lowercase then upper.

```
str1 = "AbcDEfgHi"
lower=''
upper=''
for x in str1:
    if x.islower():
        lower=lower+x
    else:
        upper=upper+x
str2=lower+upper
print(str2)
```

Q 10) Removing punctuations:

```
punct='''!()-[]{};:'"\/<>?@#$%^~.''' s1
= '[sanjoy.cs007@gmail.com]' s2=''
for x in s1:
    if x not in punct: s2
        = s2+x
print(s2)
```

TUPLES

Tuple is a collection of objects separated by commas. A tuple is similar to a Python list in terms of indexing, nested objects, and repetition but the main difference between both is Python tuple is immutable, unlike the Python list which is mutable.

- We cannot update items to a tuple once it is created.
- Tuples cannot be appended or extended.
- We cannot remove items from a tuple once it is created.

Example:

```
t = (1, 2, 4, 2, 3, True, 10.5, “Python”)

print(t)
```

⇨ (1, 2, 4, 2, 3, True, 10.5, “Python”)

Accessing Values in Python Tuples:

Tuples in Python provide two ways by which we can access the elements of a tuple.

- **Python Access Tuple using a Positive Index**
  Using square brackets we can get the values from tuples in Python.

```
t = (10, 5, 20)
print("Value in t[0] = ", t[0])     #10
print("Value in t[1] = ", t[1])     #5
print("Value in t[2] = ", t[2])     #20
```

- **Access Tuple using Negative Index**
  In the above methods, we use the positive index to access the value in Python, and here we will use the negative index within [].

```
t = (10, 5, 20)
print("Value in t[-1] = ", t[-1])   #20
print("Value in t[-2] = ", t[-2])   #5
print("Value in t[-3] = ", t[-3])   #10
```

Different Operations:

1. **Concatenation**

```
# Code for concatenating 2 tuples
t1 = (0, 1, 2, 3)
t2 = ('python', programming)
# Concatenating above two
print(t1 + t2)
⇨ (0, 1, 2, 3, 'python', 'programming')
```

2. **Nesting**

```
# Code for creating nested tuples
t1 = (0, 1, 2, 3)
t2 = ('python', 'programming')
t3 = (t1, t2)
print(t3)
⇨ ((0, 1, 2, 3), ('python', 'programming'))
```

3. **Repetition**

```
# Code to create a tuple with repetition
t = ('python',)*3
print(t)
⇨ ('python', 'python', 'python')
```

4. **Slicing**

```
# code to test slicing
t = (0 ,1, 2, 3)
print(t[1:])
print(t[::-1])
print(t[2:4])
⇨ (1, 2, 3)
⇨ (3, 2, 1, 0)
⇨ (2, 3)
```

5. **Deleting**

```
# Code for deleting a tuple
t = ( 0, 1)
del t
print(t)
```

6. **Finding the Length**

```
# Code for printing the length of a tuple
t = ('python', 'programming')
print(len(t))
⇨ 2
```

7. Converting a List to a Tuple

```
# Code for converting a list and a string into a tuple
a = [0, 1, 2]
t = tuple(a)
print(t)
⇨ (0, 1, 2)
```

Different Ways of Creating a Tuple

- Using round brackets
- Without Brackets
- Tuple Constructor
- Empty Tuple
- Single Element Tuple
- Using Tuple Packing

➢ **Using Round Brackets**

```
t = ("C", "Python")
print(t)
    ⇨ ('C', 'Python')
```

➢ **Using Comma Separated**

```
# Creating a tuple without brackets
t = 4, 5, 6
print(t)
    ⇨ (4, 5, 6)
```

➢ **Using Tuple Constructor**

```
# Creating a tuple using the tuple() constructor
t = tuple([7, 8, 9])
print(t)
    ⇨ (7, 8, 9)
```

➢ **Creating an Empty Tuple**

```
# Creating an empty tuple
t = ()
print(t)
    ⇨ ()
```

➢ **Single Element Tuple**

# Creating a single-element tuple

```
t = (10, ) # Comma is important here
print(t)
print(type(t))
```

# What if we do not use comma

```
t = (10) # This an integer (not a tuple)
print(t)
print(type(t))
```

Output

```
(10,)
<class 'tuple'>
10
<class 'int'>
```

Tuple Packing

```
# Tuple packing
a, b, c = 11, 12, 13
t = (a, b, c)
print(t)
```

⇨ (11, 12, 13)

SET in PYTHON

- ❖ Sets are used to store multiple items in a single variable.
- ❖ Set items are unordered, unchangeable, and do not allow duplicate values.
- ❖ It is written with curly brackets.

Syntax:

set_name = {item1, item2, item3}

Ex:

```
set = {"apple", "banana", "cherry", "apple"}
```

Print Set

```
print(set)        =>  {"apple",  "banana",  "cherry"}
```

Length of Set

```
print(len(set))
```

Type checking

```
print(type(set))
```

Various Types of Sets:

```
set1 = {"apple", "banana", "cherry"}
set2 = {1, 5, 7, 9, 3}
set3 = {True, False, False}
set4 = {"abc", 34, True, 40, "male"}
print(set1)
print(set2)
print(set3)
print(set4)
```

Methods of Sets:

```
#Add one element
set.add("Orange")
print(set)
#Add more element
myset = {"apple", "banana", "cherry"}
yourset = {"pineapple", "mango", "papaya", "banana"}
myset.update(yourset)
print(myset)
#Remove & Discard (Delete the particular element)
myset.remove("banana")      #remove
yourset.discard("mango")    #discard
print(myset)
print(yourset)
#Pop (Delete first element)
set1 = {1, 2, 3, 4, 5}
set1.pop()
print(set1)
x=set1.pop()
print(x)
print(set1)
#Clear (Clear the set)
set1.clear()
print(set1)
```

#Delete set (Delete the set permanently)

```
n = {1,2,3}
print(n)
del n
```

#Copy (Copy a set to another)

```
x={10,20,30}
y = x.copy()
print(x,y)
```

Check Subset and Disjoint in Sets:

#Check the set is Subset or not (<=)

```
x = {"a", "b", "c"}
y = {"f", "e", "d", "c", "b", "a"}
z = x.issubset(y)    #z=x<=y
print(z)
```

#Disjoint (Check common elements are present or not)

(If present then False, otherwise True)

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft"}
z = x.isdisjoint(y)
print(z)
```

Operations of Sets:

```
#Union (|) (Join Sets)
a={1,2,3,4,5}
b={1,2,6,7}
c=a.union(b)     #c=a|b
print(c)
#intersection (&) (Return elements which are present
in both sets)
a={1,2,3,4,5}
b={1,2,6,7}
c=a.intersection(b)     #c=a&b
print(c)
#Difference (-) (Remove the elements of b from a)
a={1,2,3,4,5}
b={1,2,6,7}
c=a.difference(b)     #c=a-b
print(cD
#Symmetric Difference (^) (Return elements except
intersection)
a={1,2,3,4,5}
b={1,2,6,7}
c=a.symmetric_difference(b)     #c=a^b
print(c)
```

Python Dictionaries

A dictionary is a collection which is ***ordered, changeable*** and do not allow duplicates. It is denoted by ***dict*** keyword.

It is a data structure that stores the value in ***key: value*** pairs. Values in a dictionary can be of any data type and can be duplicated, whereas keys can't be repeated and must be ***immutable***.

***Syntax*:**
dictionary_name = {key : value, key : value}

***Example*:**
Here, The data is stored in key:value pairs in dictionaries, which makes it easier to find values**.**

```
d = {1: 'Geeks', 2: 'For', 3: 'Geeks'}
print(d)
```

✵ ***Access Items:***

#print the dictionary

```
mydict = {
  "name": "Tarun",
  "roll": 100,
  "marks": 90.99
}
print(mydict)
```

#access the items of a dictionary by referring to its key

```
mydict = {
  "name": "Tarun",
  "roll": 100,
  "marks": 90.99
}
x = mydict["model"]
print(x)
```

#get keys

```
x=mydict.keys()
print(x)
```

#get values
```
x=mydict.values()
print(x)
```
#get items
```
x=mydict.items()
print(x)
```

❈ *Change Items:*

#change the item
```
mydict = {
 "name": "Tarun",
 "roll": 100,
  "marks": 90.99
}
mydict ["roll"] = 200
print(mydict)
```

#update the dictionary with the items from the given argument
```
mydict = {
  "name": "Tarun",
  "roll": 100,
  "marks": 90.99
}
mydict.update({“roll”: 200})
print(mydict)
```

❈ *Add Items:*

#add new item in dictionary
```
mydict = {
  "name": "Tarun",
  "roll": 100,
  "marks": 90.99
}
mydict ["age"] = 20
print(mydict)
```
#update dictionary
```
mydict = {
  "name": "Tarun",
  "roll": 100,
  "marks": 90.99
}
mydict.update({“age”: 20})
print(mydict)
```

❈ Remove Items

# The pop() method removes the item

```
mydict = {
  "name": "Tarun",
  "roll": 100,
  "marks": 90.99
}
mydict.pop("marks")
print(mydict)
```

# The popitem() method removes the last item

```
mydict = {
  "name": "Tarun",
  "roll": 100,
  "marks": 90.99
}
mydict.pop()
print(mydict)
```

#The del keyword removes the item

```
mydict = {
  "name": "Tarun",
  "roll": 100,
  "marks": 90.99
}
del mydict["marks"]
print(mydict)
```

#del keyword to delete dictionary permanently

```
mydict = {
  "name": "Tarun",
  "roll": 100,
  "marks": 90.99
}
del mydict
print(mydict)
```

# clear() method empties the dictionary

```
mydict = {
  "name": "Tarun",
  "roll": 100,
  "marks": 90.99
}
mydict.clear()
print(mydict)
```

✵ Copy Dictionary

You cannot copy a dictionary simply by typing **dict2 = dict1**, because: dict2 will only be a reference to dict1, and changes made in dict1 will automatically also be made in dict2.

There are ways to make a copy, one way is to use the built-in Dictionary method ***copy().***

#Make a copy of a dictionary with the copy() method

```
dict1 = {
  "name": "Tarun",
  "roll": 100,
  "marks": 90.99
}
dict2=dict1.copy()
print(dict1)
print(dict2)
```

# Another way to make a copy is to use the built-in function dict().

```
dict1 = {
  "name": "Tarun",
  "roll": 100,
  "marks": 90.99
}
dict2=dict(dict1)
print(dict2)
```

✵ Nested Dictionaries

A dictionary can contain dictionaries, this is called nested dictionaries.

#Create a dictionary that contain three dictionaries

```
family = {
  "child1" : {
    "name" : " Atanu",
    "year" : 2000
  },
  "child2"  :  {
    "name" : " Deepak",
    "year" : 2003
  },
  "child3"  :  {
    "name" : " Lokesh",
    "year" : 2006
  }
}

print(family)
```

#Another way of creating nested dictionaries

```
child1  =  {
  "name" : "Atanu",
    "year" : 2000
},
"child2"  =  {
    "name" : "Deepak",
    "year" : 2003
},
"child3"  =  {
    "name" : "Lokesh",
    "year" : 2006
}
family = {
  "child1": child1,
  "child2": child2,
  "child3": child3
}
print(family)
```

#Access any item of nested dictionaries

```
print(family["child3"]["name"])
```

✵ Dictionary Methods

formkeys()

The fromkeys() method returns a dictionary with the specified keys and the specified value.

Create a dictionary with 3 keys, all with the value 1:

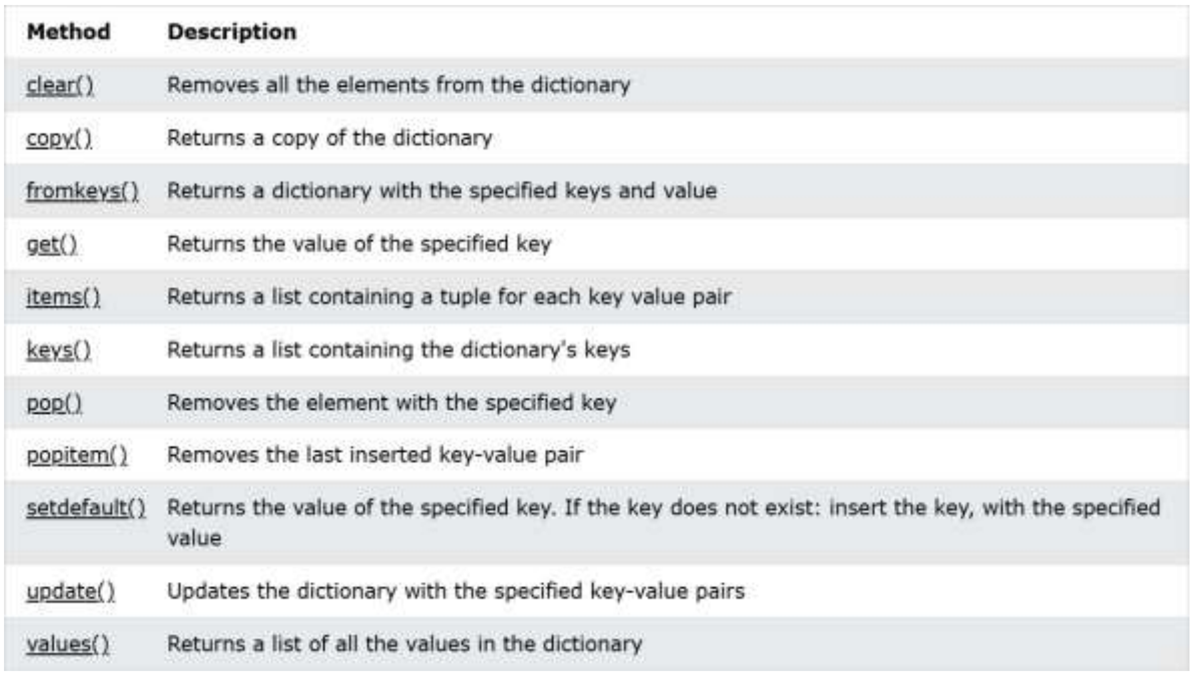
```
x = ('key1', 'key2', 'key3')
y = 1
mydict = dict.fromkeys(x, y)
print(mydict)
```

setdefault()

Get the value of the "color" item, if the "color" item does not exist, insert "color" with the value "black":

```
car = {
  "brand": "Maruti",
  "model": "Suzuki",
  "year": 1999
}
x = car.setdefault("color", "black")
print(x)
print(car)
```

Follow these methods:



Method	Description
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and value
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

Functions

What are Functions?

- A function is a ***piece of code*** which ***performs a specific task***.
- It is used to ***reduce the repetition*** of the code hence ***reducing the size*** of the code, less chances of making mistakes in the code.
- Functions make the task easy for the programmers as each programmer in a team can be given a specific task (function) and at the end we can combine these tasks together to make single application.

- ***Some points to remember are:***

  - Program is a general term.
  - A very big program is called application.
  - An application broke into pieces are called modules that performs a specifictask.
  - Modules broken into pieces are called Functions / Procedure.

Advantages of a function:

- Easy development
- Error free
- Can be developed by a team of programmers
- Reuse modules for function in other projects

Creating a Function:

Syntax: def Name_of_the_function (<parameter List >)

```
          ……….
          ……….
          ………. return

          result
```

- A function must return a result and its python it returns result.
- If you don't write return inside a function then it'll automatically returns ***None.***

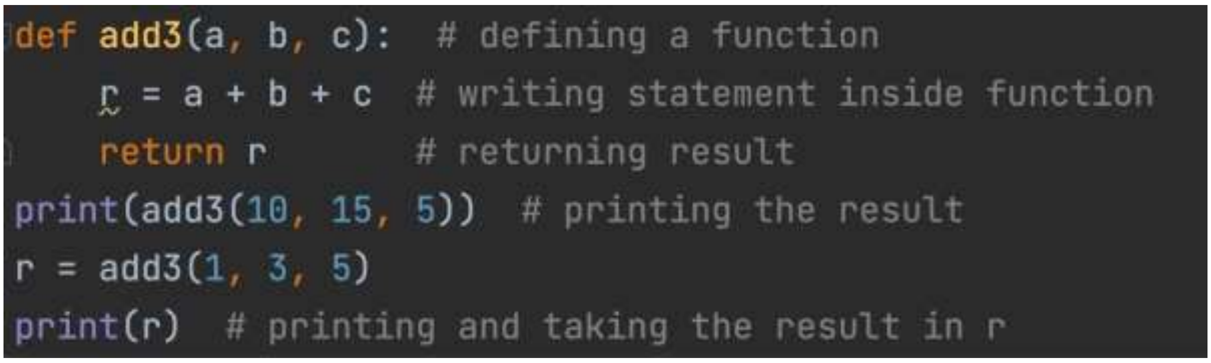
How to write a function?

- ✓ We can write a function by using the keyword ***def*** followed by a function name.
- ✓ The function name is user define and it is suggested to take a meaningful name while defining a function.
- ✓ The rules for defining a function is same as giving variable names.
- ✓ Within the ( ) you can pass parameters to a function these Parameter are called ***"Formal parameter".***
- ✓ Parameters are called input to a function; a function can take multiple Parameter.
- ✓ Parameters can be of any datatype.
- ✓ Returning values of the statements is called ***"output".***
- ✓ You can call a function by using the function name and pass parameter in ( ) , these Parameterare called ***"Actual parameter".***
- ✓ The actual parameter values are copied into formal parameter which acts as input to afunction they are copied in the same position / order.
- ✓ When you call a functioning, a result is returned you should place that result into another variable (or) print it directly.
- ✓ If you don't write return in function, it'll return ***NONE.***
- ✓ So, every function returns whether you write it or not.

Syntax:

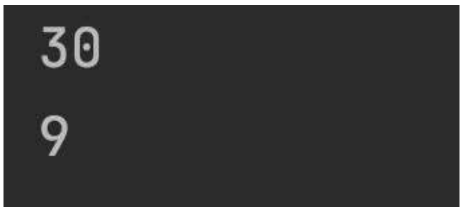
```
def fun_name (para1, para2, para3):         #Formal parameter
        statement1
        statement2
        …………                                #Statements of function
        …………
        return result                       #Returning result
fun_name(para1, para2, para3)               #Calling a function
```

A simple example to understand function.



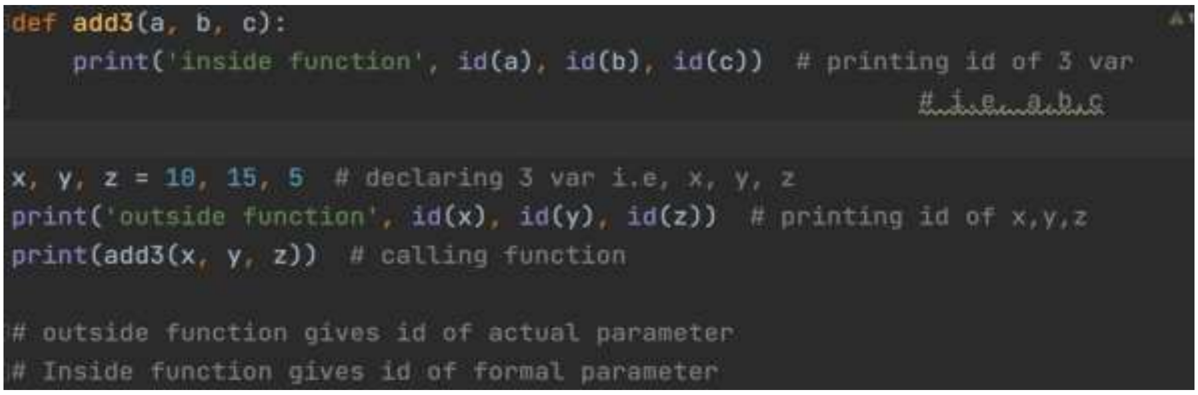
```python
def add3(a, b, c):  # defining a function
    r = a + b + c  # writing statement inside function
    return r       # returning result
print(add3(10, 15, 5))  # printing the result
r = add3(1, 3, 5)
print(r)  # printing and taking the result in r
```

Output:



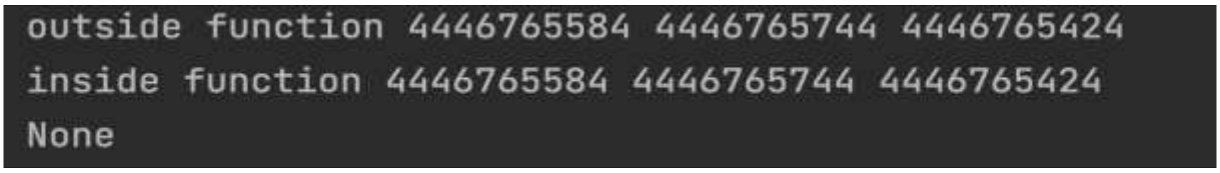
```
30
9
```

- The values in formal parameter acts as a pointer to actual parameter.
- Therefore, they'll be referring to the same thing.
- Let's understand this with an



```python
def add3(a, b, c):
    print('inside function', id(a), id(b), id(c))  # printing id of 3 var
                                                    # i.e. a,b,c

x, y, z = 10, 15, 5  # declaring 3 var i.e, x, y, z
print('outside function', id(x), id(y), id(z))  # printing id of x,y,z
print(add3(x, y, z))  # calling function

# outside function gives id of actual parameter
# Inside function gives id of formal parameter
```

Output:



```
outside function 4446765584 4446765744 4446765424
inside function 4446765584 4446765744 4446765424
None
```

- In python object are always pass just like reference only, copy of an object will never be pass.

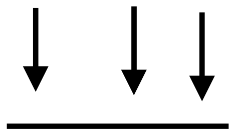
Positional vs keyword Arguments

def net_sal(basic, allowance, deduction): # for calculating net sal of an employee.

net=basic + allowance - deduction

return net

n= net_sal(8000, 6000,2000) # way to call function for net sal



Actual parameters

$$\begin{tikzcd}
\text{Basic} \arrow[d] & \text{allowance} \arrow[d] & \text{deduction} \arrow[d] \\
8000 & 6000 & 2000
\end{tikzcd}$$

Actual parameters can be copied.

def net_sal(basic, allowance, deduction): #function for net sal net=

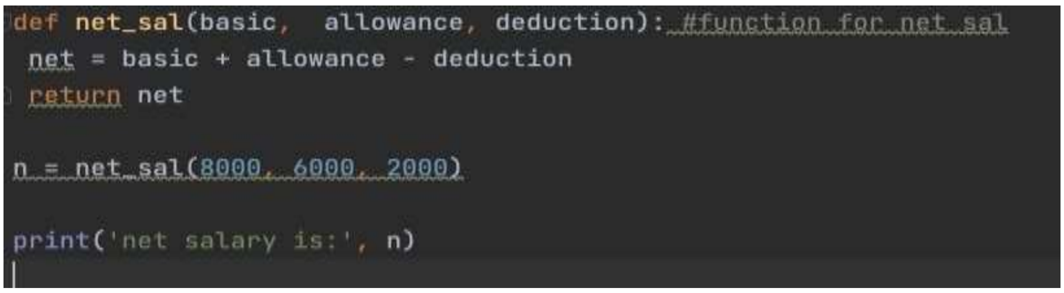
basic+allowance-deduction

return net

n=net_sal(basic, allowance, deduction)

print('Net salary is:, n')

To know which variable has print which value.



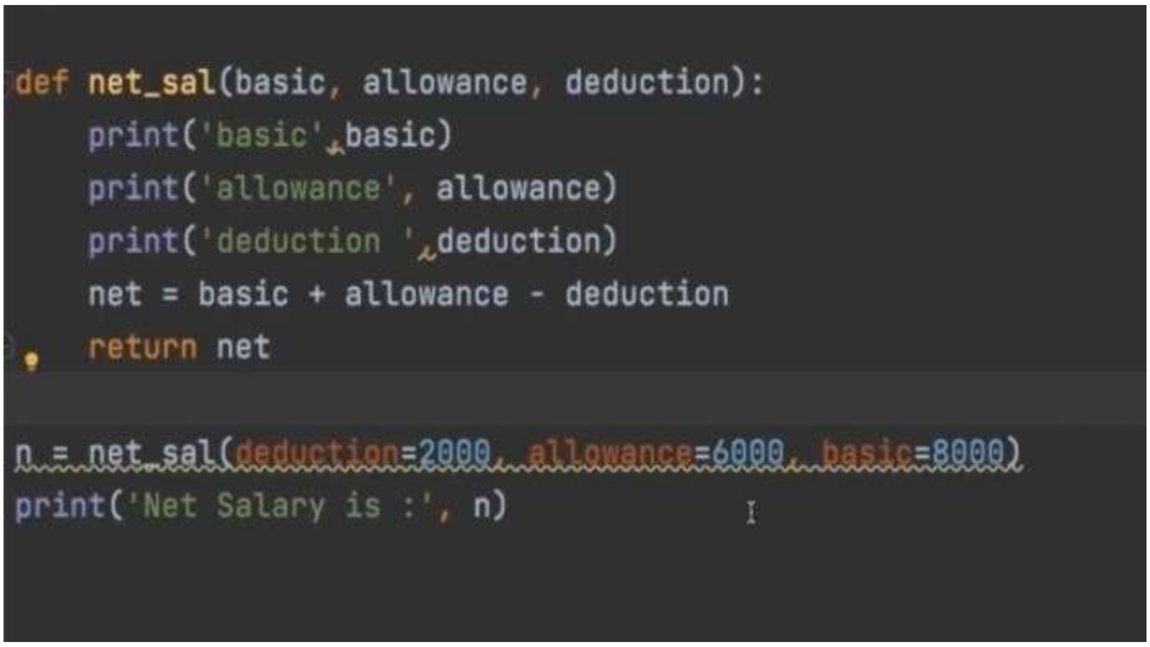
```
def net_sal(basic,  allowance, deduction): #function for net sal
  net = basic + allowance - deduction
  return net

n = net_sal(8000, 6000, 2000)

print('net salary is:', n)
```

Output:

```
net salary is : 12000
```

# So, this is called as positional arguments.

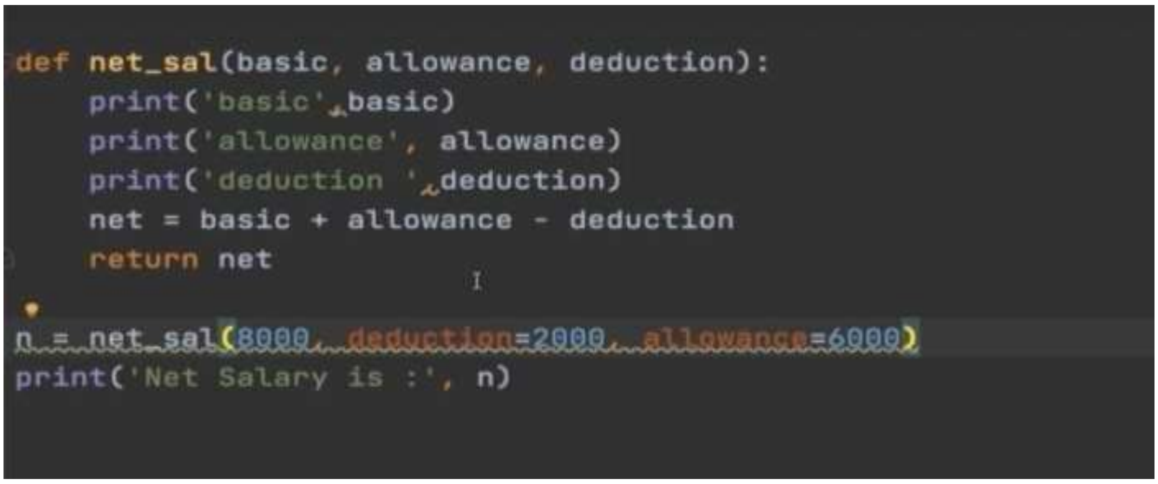
Program:



```python
def net_sal(basic, allowance, deduction):
    print('basic',basic)
    print('allowance', allowance)
    print('deduction ',deduction)
    net = basic + allowance - deduction
    return net

n = net_sal(deduction=2000, allowance=6000, basic=8000)
print('Net Salary is :', n)
```

By writing the names of parameters also we can call the function.

These are called as keyword argument. Without that it is a positional argument.



```python
def net_sal(basic, allowance, deduction):
    print('basic',basic)
    print('allowance', allowance)
    print('deduction ',deduction)
    net = basic + allowance - deduction
    return net

n = net_sal(8000, deduction=2000, allowance=6000)
print('Net Salary is :', n)
```

# Both positional and keyword argument.

Default Argument

Default Arguments:

Python has a different way of representing syntax and default values for function arguments. Default values indicate that the function argument will take that value if no argument value is passed during the function call. The default value is assigned by using the assignment(=) operator of the form *keywordname*=value.

Syntax:

def fun_name(para1, para2=default_value2,para3=default_value3)

We can make argument default or optional

Example:

def add (a,b,c):

        return a+b+c print(

add(10,5,2)) ✓

print(add(10,5) ✗

print(add(10)) ✗

Can we pass different number of parameters? Yes, we can pass def add

a , b=0 , c=0):

        return a+b+c add

(5,7)

It will take a as 5 and b=7 and c as 0

Here two cases are default add(a=10,

b=5, c=2) ✓

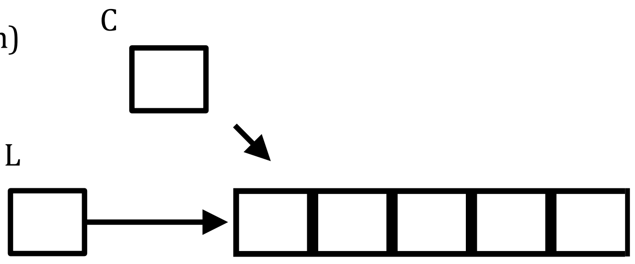
add(b=5, c=2, 10) ✗

Default are created only once

def addition (item, L =[ ])

L.append(item)

return L



Mixed positional keyword

If we want a function to take only positional arguments or only keyword can enforce and make it mandatory.

Let's see how some arguments can have positional only or keywords only. def

add(a, b, c, d, e, f): # taking 6 arguments

return a+ b+ c+ d+ e+ f # adding and returning result

if we have integer type of all so, it will add all and return the result of all

add(2, 5, 9, 7, 3, 8) and these arguments will be copied to the corresponding arguments.

In the same position to this formal parameter get copied in actual parameters. 2 goes

in a

5 goes in b

……………..

Add(f=8, c=9, b=5, c=3, d=7, a=2)

So we can use keyword argument also Here 8

goes to f

………

def add(a, b, /,c, d, e, f): return

a+b+c+d+e+f

Taking 6 arguments adding and returning result but here the slash means we can only give positional arguments before slash and after we can give both positional and keyword arguments both are allowed.

add(2,5,6,7,9,4) # all positional

add(2, 5, d=7, f=4, e=9, c=6)

$\searrow$

This two must be positional

Where keyword passed order must not be important.

```
def add(a, b, c, d, e, f):
    return a+b+c+d+e+f
r=add(2, 5, 7, 4, 8, 9) # updating this to check whether it is working or not
```

def add(a, b, / ,c, d, e, f):

print(a, b, c, d, e, f) return

a+b+c+d+e+f

r=add(f=9,e=8, a=2, c=7, b=5, d=4) print(r)

If given slash it will give an error as we have given a and b as positional argument.

def add(a, b, / , c, d, * ,e, f):

$\downarrow$ $\downarrow$ $\downarrow$

Positional any keyword

return a+b+c+d+e+f

add(2,5,d=8, c=4, 9, 3)

By writing / we say that only positional arguments.

By writing * at the beginning of the statement we say keywords are mandatory.

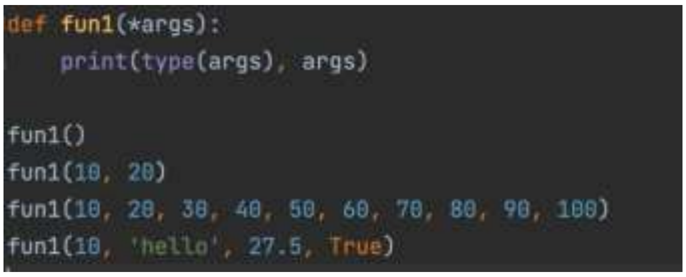
Program

```
def add(a, b, /, c, d, *, e, f):
    print(a, b, c, d, e, f)
    return a+b+c+d+e+f
r = add(2, 5, 6, 7, f=9, e=8)
print(r)
```

Variable Length Arguments

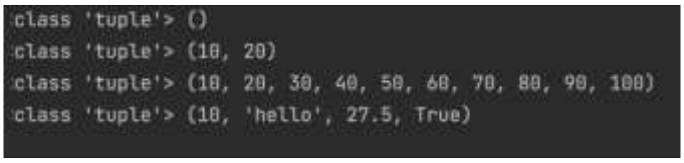
- The number of arguments a function takes is pre-defined.
- When default arguments are given, we have options to pass parameter
- In python we can take as many parameters we want in a function by using variable length arguments, this is taken in the form of a tuple but the parameter name should start with *.

syntax: def fun (* args) :
print (args)



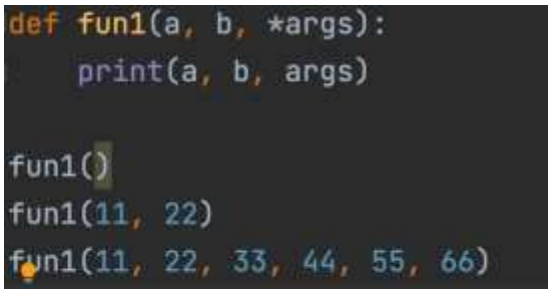
```
def fun1(*args):
    print(type(args), args)

fun1()
fun1(10, 20)
fun1(10, 20, 30, 40, 50, 60, 70, 80, 90, 100)
fun1(10, 'hello', 27.5, True)
```

Output:



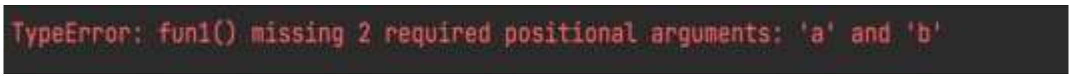
```
class 'tuple'> ()
class 'tuple'> (10, 20)
class 'tuple'> (10, 20, 30, 40, 50, 60, 70, 80, 90, 100)
class 'tuple'> (10, 'hello', 27.5, True)
```

- If there are more parameter given before *args then it is compulsory to give its value during function call for *args its optional.



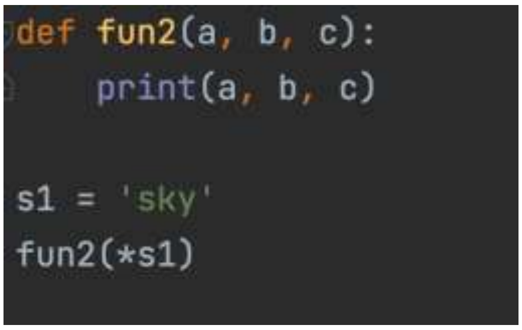
```
def fun1(a, b, *args):
    print(a, b, args)

fun1()
fun1(11, 22)
fun1(11, 22, 33, 44, 55, 66)
```

Output:



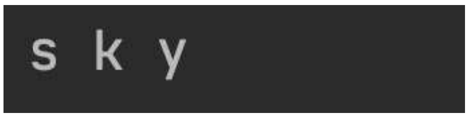
```
TypeError: fun1() missing 2 required positional arguments: 'a' and 'b'
```

If you have a sequence like list, tuple, string then you can unpack that in a function call using *



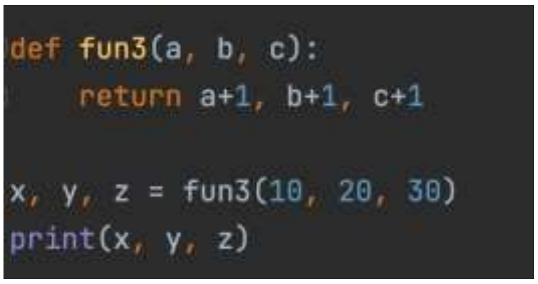
```
def fun2(a, b, c):
    print(a, b, c)

s1 = 'sky'
fun2(*s1)
```

Output:



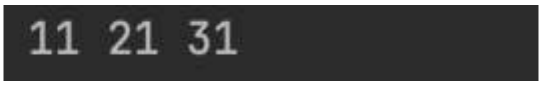
```
s k y
```

- If a function is returning multiple values, it is possible in python, then all values can be unpacked in different variable or take them in a tuple.



```
def fun3(a, b, c):
    return a+1, b+1, c+1

x, y, z = fun3(10, 20, 30)
print(x, y, z)
```

Output:

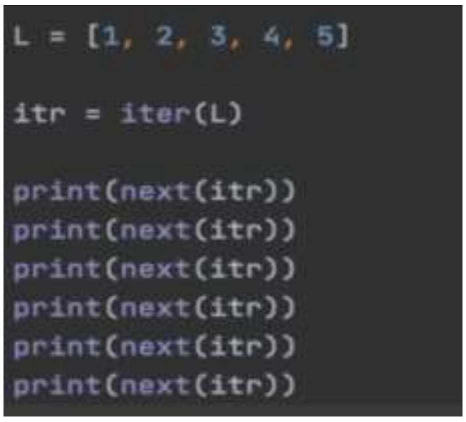


```
11 21 31
```

Iterators and Generators

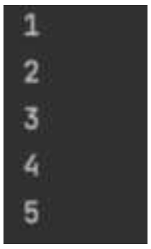
- Iterators allow us to iterate through a sequence of element I.e., visiting eachelement once in a sequence.
- We can pass any sequence to the iterator like list, string, tuple, set anddictionary.

- In python there is a built-in function called iter for iteration.



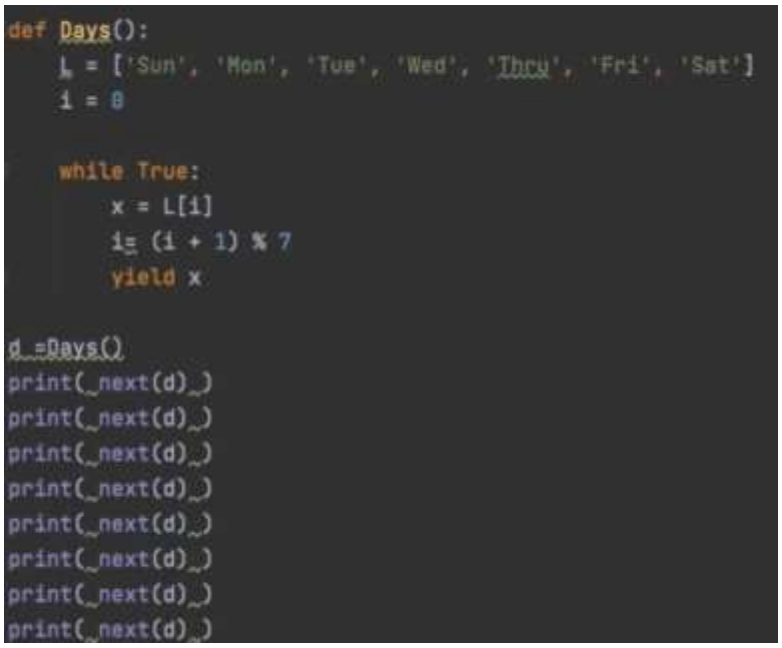
```
L = [1, 2, 3, 4, 5]

itr = iter(L)

print(next(itr))
print(next(itr))
print(next(itr))
print(next(itr))
print(next(itr))
print(next(itr))
```

#next(itr) give next element in sequence.



```
1
2
3
4
5
```

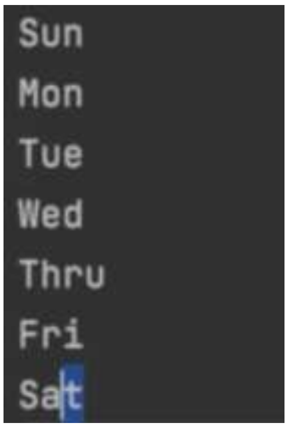
- We can write our own iterators which are called “GENERATORS” they work just like iterators.
- Let’s see this with an example which displays days of a week and once the sequence is completed and called again it should return the first value and soon…



```
def Days():
    L = ['Sun', 'Mon', 'Tue', 'Wed', 'Thru', 'Fri', 'Sat']
    i = 0

    while True:
        x = L[i]
        i= (i + 1) % 7
        yield x

d =Days()
print( next(d) )
print( next(d) )
print( next(d) )
print( next(d) )
print( next(d) )
print( next(d) )
print( next(d) )
print( next(d) )
```

-
-
-
-
-
-

# For a loop to remain in same state we use **yield** it’ll not stop the function but keep function on hold and return value.



```
Sun
Mon
Tue
Wed
Thru
Fri
Sat
```

Local vs global variable

```
g=10                    #global variable
def fun1(a, b):
    c=a+b               #local variable
    print('local', c)
    print('global',g)
fun1(4,8)
```

- ✓ Access the g value.
- ✓ Variables of the python file are the global variables.
- ✓ The variables declared outside the function is global variable.
- ✓ Global variable can be declared anywhere inside any function.
- ✓ Global variable cannot be modified in any function for modification.

The Local Variable:

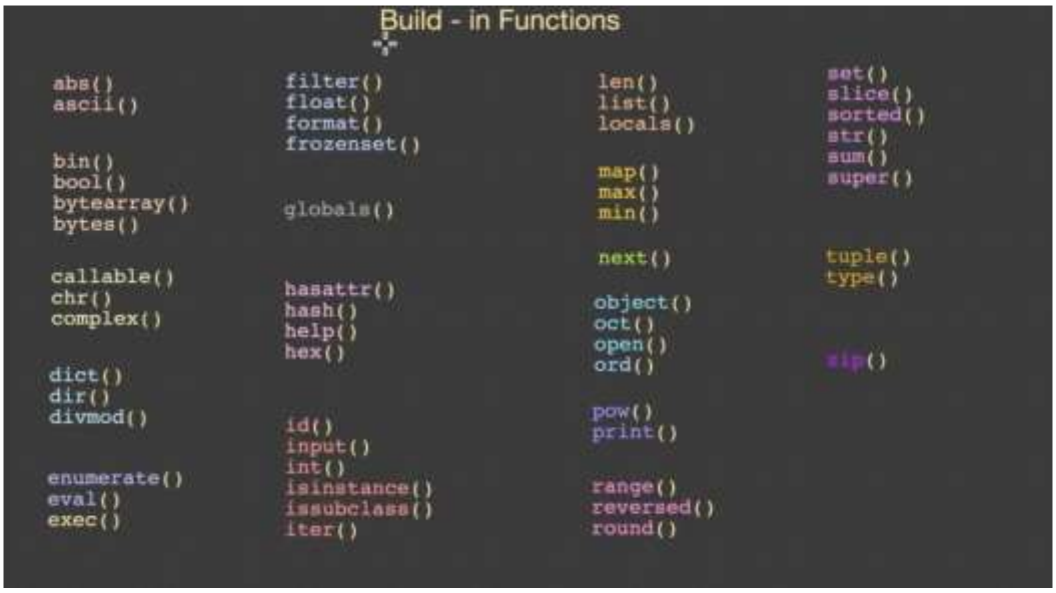
- ✓ When we declare a variable inside a function it becomes a local variable.
- ✓ A local variable is variable whose scope is limited to only that function and where it is created.
- ✓ That means the local variable value is available only in that function where it is created.

The Global Variable:

- ✓ Sometimes, the global variable and the local variable has the same name. In that case, thefunction, by default, refers to the local variable and ignores the global variable.
- ✓ So, the global variable is not accessible.

Build-in Function #1

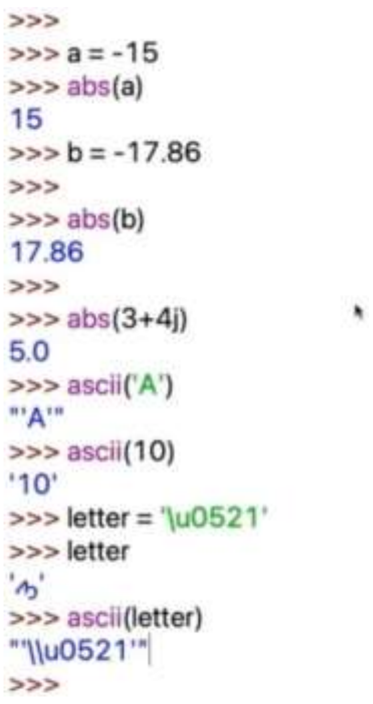
- They are many Build-in Functions / global functions available in python
- The Build-in Functions are



Build - in Functions

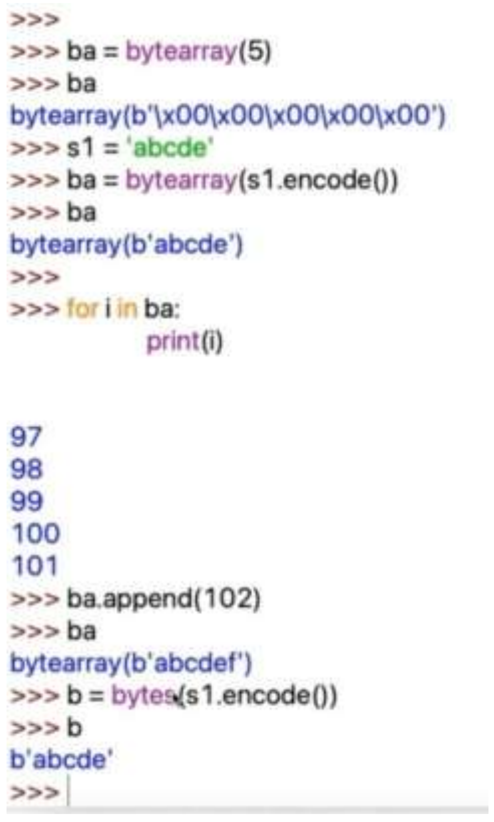
```
abs()          filter()       len()          set()
ascii()        float()        list()         slice()
               format()       locals()       sorted()
bin()          frozenset()                   str()
bool()                        map()          sum()
bytearray()    globals()      max()          super()
bytes()                       min()
                                             tuple()
callable()                    next()         type()
chr()          hasattr()
complex()      hash()         object()
               help()         oct()          zip()
dict()         hex()          open()
dir()                         ord()
divmod()       id()
               input()        pow()
enumerate()    int()          print()
eval()         isinstance()
exec()         issubclass()   range()
               iter()         reversed()
                              round()
```

- Lets see the working of these functions
- ***abs( ) , ascii( )***



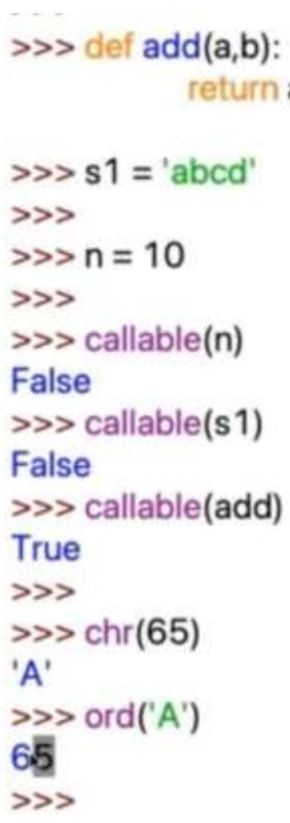
```
>>>
>>> a = -15
>>> abs(a)
15
>>> b = -17.86
>>>
>>> abs(b)
17.86
>>>
>>> abs(3+4j)
5.0
>>> ascii('A')
"'A'"
>>> ascii(10)
'10'
>>> letter = '\u0521'
>>> letter
'ԡ'
>>> ascii(letter)
"'\\u0521'"
>>>
```

- ***bin( )*** it takes number and convert it into binary form
- ***bool( )*** convert anything into bool type
- ***bytearray( )*** and ***bytes( )*** they both are similar the difference is byte array is mutable whereas bytes is immutable



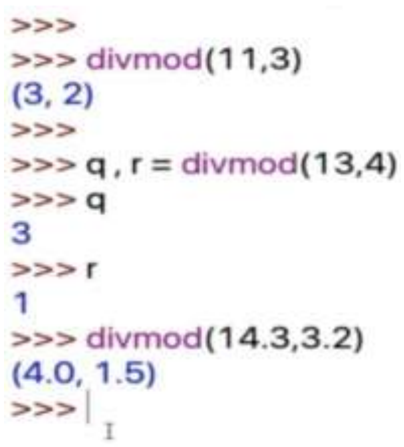
```
>>>
>>> ba = bytearray(5)
>>> ba
bytearray(b'\x00\x00\x00\x00\x00')
>>> s1 = 'abcde'
>>> ba = bytearray(s1.encode())
>>> ba
bytearray(b'abcde')
>>>
>>> for i in ba:
            print(i)


97
98
99
100
101
>>> ba.append(102)
>>> ba
bytearray(b'abcdef')
>>> b = bytes(s1.encode())
>>> b
b'abcde'
>>>
```

- ***callable( )*** we can know if the given identifier is a function or not.
- ***chr( )*** gives you the character for any given ascii code.
- ***complex( )*** used for creating complex datatype.



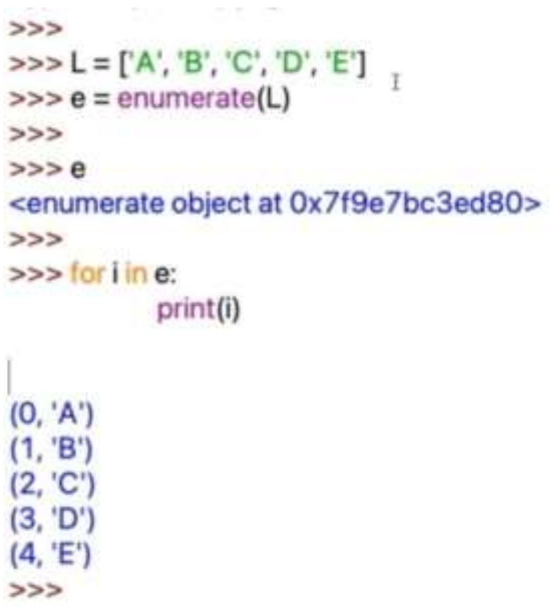
```
>>> def add(a,b):
            return a+b

>>> s1 = 'abcd'
>>>
>>> n = 10
>>>
>>> callable(n)
False
>>> callable(s1)
False
>>> callable(add)
True
>>>
>>> chr(65)
'A'
>>> ord('A')
65
>>>
```

- ***dict( )*** used for creating a Dictionary.
- ***dir( )*** give details of particular class.
- ***divmod( )*** takes 2 parameters and gives division as well as modulus as result.



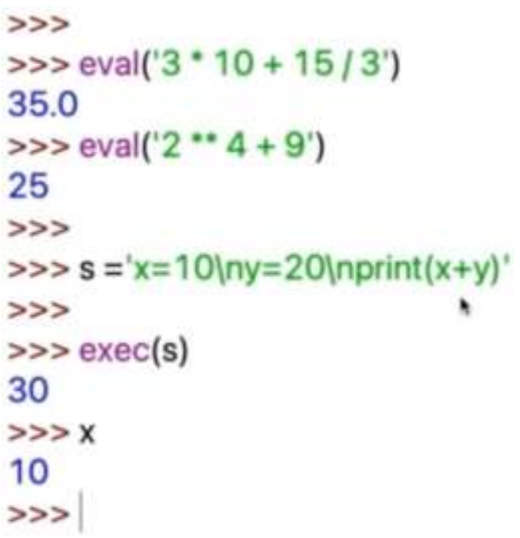
```
>>>
>>> divmod(11,3)
(3, 2)
>>>
>>> q , r = divmod(13,4)
>>> q
3
>>> r
1
>>> divmod(14.3,3.2)
(4.0, 1.5)
>>>
```

- ***enumerate( )*** gives indexing for all items in given sequence



```
>>>
>>> L = ['A', 'B', 'C', 'D', 'E']
>>> e = enumerate(L)
>>>
>>> e
<enumerate object at 0x7f9e7bc3ed80>
>>>
>>> for i in e:
            print(i)


(0, 'A')
(1, 'B')
(2, 'C')
(3, 'D')
(4, 'E')
>>>
```

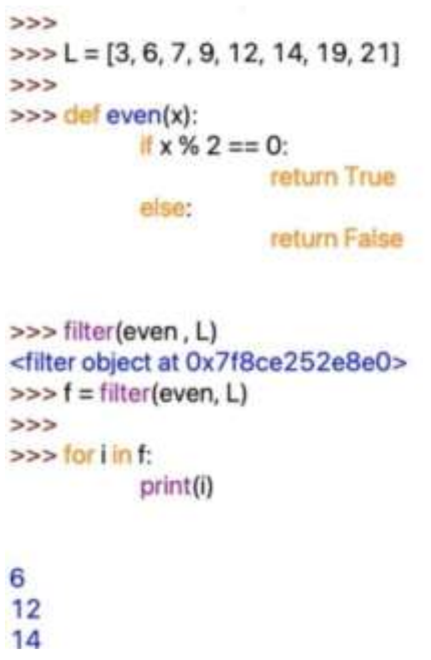
- ***eval( )*** evaluates an expression.
- ***exec( )*** execute python statements.



```
>>>
>>> eval('3 * 10 + 15 / 3')
35.0
>>> eval('2 ** 4 + 9')
25
>>>
>>> s ='x=10\ny=20\nprint(x+y)'
>>>
>>> exec(s)
30
>>> x
10
>>>
```

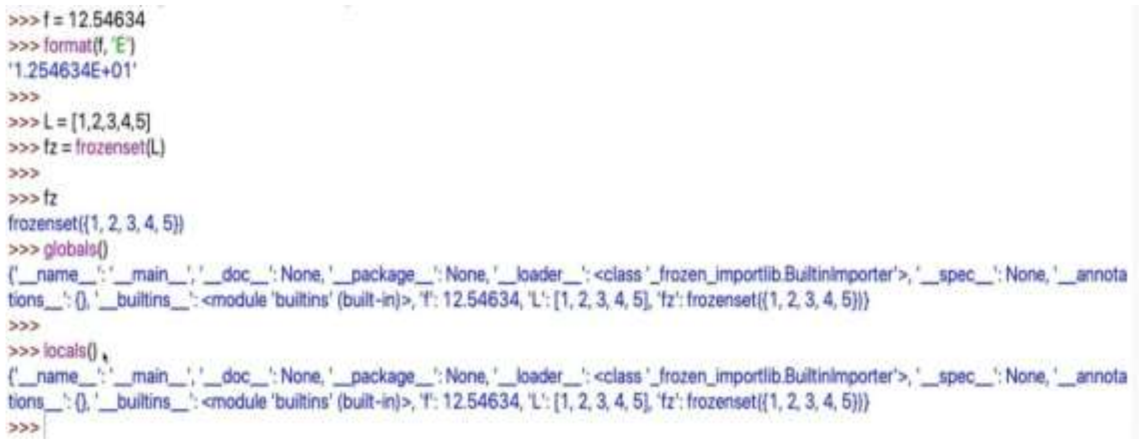
Build-in Functions #2

- ***filter( )*** - filter objects from any iterable based on function you give



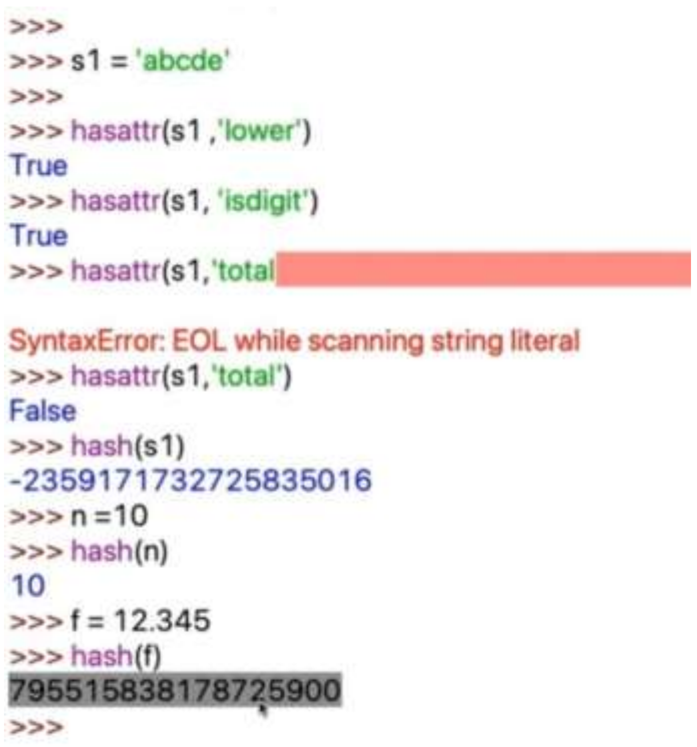
```
>>>
>>> L = [3, 6, 7, 9, 12, 14, 19, 21]
>>>
>>> def even(x):
        if x % 2 == 0:
                return True
        else:
                return False


>>> filter(even , L)
<filter object at 0x7f8ce252e8e0>
>>> f = filter(even, L)
>>>
>>> for i in f:
        print(i)


6
12
14
```

- ***float( )*** - Convert datatype into float type
- ***format( )*** - same as string formatting
- ***frozenset( )*** - Takes any literal and convert it into immutable set
- ***globals( )*** - gives all the global variables declared inside a python program

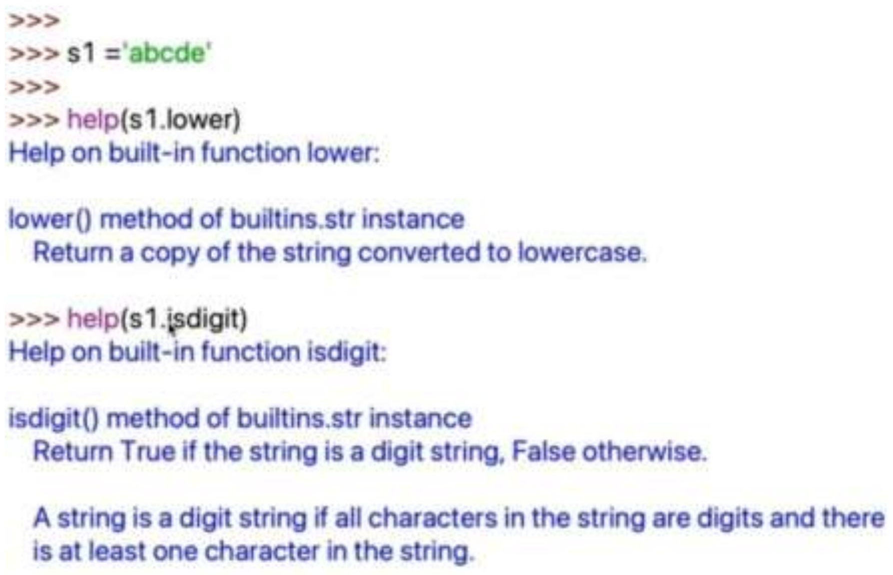


```
>>> f = 12.54634
>>> format(f, 'E')
'1.254634E+01'
>>>
>>> L = [1,2,3,4,5]
>>> fz = frozenset(L)
>>>
>>> fz
frozenset({1, 2, 3, 4, 5})
>>> globals()
{'__name__': '__main__', '__doc__': None, '__package__': None, '__loader__': <class '_frozen_importlib.BuiltinImporter'>, '__spec__': None, '__annota
tions__': {}, '__builtins__': <module 'builtins' (built-in)>, 'f': 12.54634, 'L': [1, 2, 3, 4, 5], 'fz': frozenset({1, 2, 3, 4, 5})}
>>>
>>> locals()
{'__name__': '__main__', '__doc__': None, '__package__': None, '__loader__': <class '_frozen_importlib.BuiltinImporter'>, '__spec__': None, '__annota
tions__': {}, '__builtins__': <module 'builtins' (built-in)>, 'f': 12.54634, 'L': [1, 2, 3, 4, 5], 'fz': frozenset({1, 2, 3, 4, 5})}
>>>
```

- ***hasattr( )*** - verifying if you are having so and so attribute or not
- ***hash( )*** - it'll show the hash value of the particular object you called
- ***help( )*** - gives details about any method or class
- ***hex( )*** - base conversion

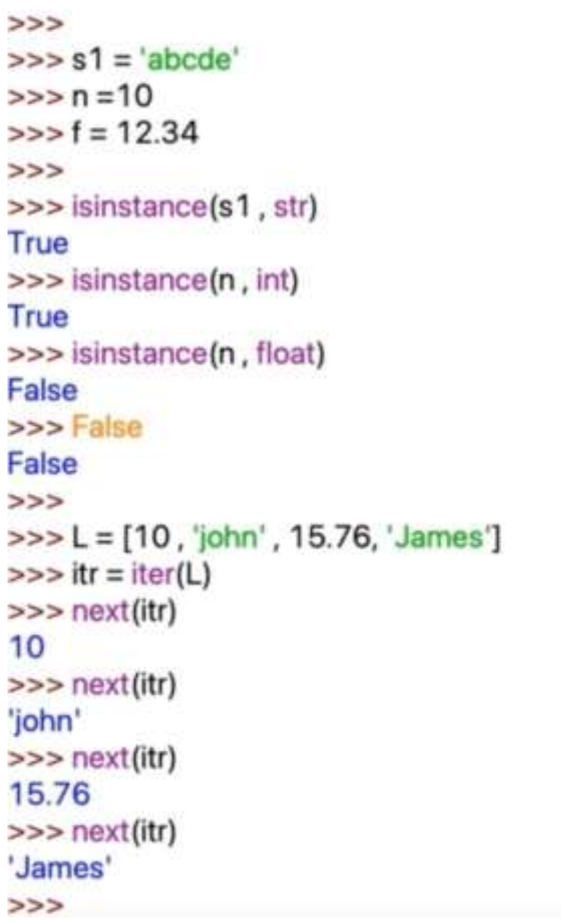


```
>>>
>>> s1 = 'abcde'
>>>
>>> hasattr(s1 ,'lower')
True
>>> hasattr(s1, 'isdigit')
True
>>> hasattr(s1,'total

SyntaxError: EOL while scanning string literal
>>> hasattr(s1,'total')
False
>>> hash(s1)
-2359171732725835016
>>> n =10
>>> hash(n)
10
>>> f = 12.345
>>> hash(f)
795515838178725900
>>>
```



```
>>>
>>> s1 ='abcde'
>>>
>>> help(s1.lower)
Help on built-in function lower:

lower() method of builtins.str instance
    Return a copy of the string converted to lowercase.

>>> help(s1.isdigit)
Help on built-in function isdigit:

isdigit() method of builtins.str instance
    Return True if the string is a digit string, False otherwise.

    A string is a digit string if all characters in the string are digits and there
    is at least one character in the string.
```

- ***isinstance( )*** - check if an object is an instance of particular class
- ***issubclass( )*** - tells if a class is a subclass or not
- ***iter( )*** - helps us iterate over every element in a sequence used along with next( )



```
>>>
>>> s1 = 'abcde'
>>> n =10
>>> f = 12.34
>>>
>>> isinstance(s1 , str)
True
>>> isinstance(n , int)
True
>>> isinstance(n , float)
False
>>> False
False
>>>
>>> L = [10 , 'john' , 15.76, 'James']
>>> itr = iter(L)
>>> next(itr)
10
>>> next(itr)
'john'
>>> next(itr)
15.76
>>> next(itr)
'James'
>>>
```

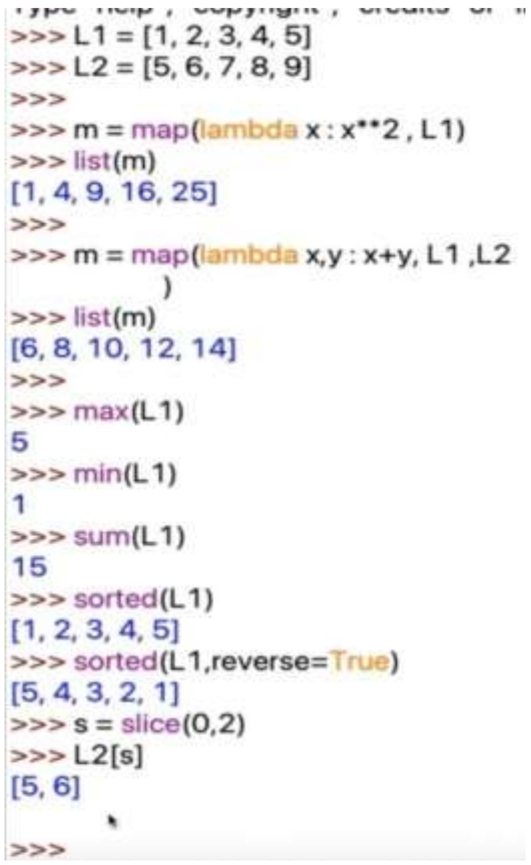
Build-in Function #3

- Functions we already know

- ***len( )*** - find length of any object
- ***list( )*** - creating items of list
- ***local( )*** - give local variables of given function
- ***object( )*** - base class for all python object
- ***oct( )*** - base conversion function
- ***open( )*** - opening a file
- ***ord( )*** - gives ascii code for given class
- ***print( )*** - prints the result
- ***range( )*** - gives value in specific range
- ***set( )*** - create object of set class
- ***str( )*** - create object of str class

- ***super( )*** - refers object of super class
- ***tuple( )*** - create object of tuple class
- ***type( )*** - gives type if datatype

- ***New Functions***
- ***map( )*** - map element of one sequence into another
- ***max( )*** - gives maximum value in a sequence
- ***min( )*** - gives minimum value in a sequence
- ***sum( )*** - gives sum of all elements in a sequence
- ***sorted( )*** - sort a sequence
- ***slice( )*** - gives slice object of a sequence



```
>>> L1 = [1, 2, 3, 4, 5]
>>> L2 = [5, 6, 7, 8, 9]
>>>
>>> m = map(lambda x : x**2 , L1)
>>> list(m)
[1, 4, 9, 16, 25]
>>>
>>> m = map(lambda x,y : x+y, L1 ,L2
             )
>>> list(m)
[6, 8, 10, 12, 14]
>>>
>>> max(L1)
5
>>> min(L1)
1
>>> sum(L1)
15
>>> sorted(L1)
[1, 2, 3, 4, 5]
>>> sorted(L1,reverse=True)
[5, 4, 3, 2, 1]
>>> s = slice(0,2)
>>> L2[s]
[5, 6]

>>>
```

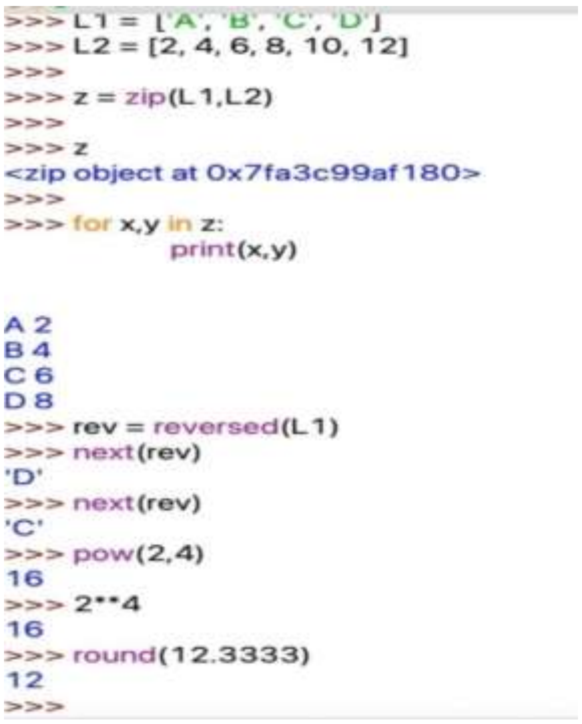
Random Numbers

Python defines a set of functions that are used to generate or manipulate random numbers through the random module.
Functions in the random module rely on a pseudo-random number generator ***function random (),*** which generates a random float number between 0.0 and
1.0. This particular type of functions is used in a lot of games, lotteries, or any application requiring a random number generation.

Example:

```
import random
num = random.random()
print(num)
```

- ***zip( )*** - join elements from corresponding 2 sequence
- ***reversed( )*** - same as iterator but perform reverse iteration
- ***pow( )*** - gives power value of a given number
- ***round( )*** - round up the integer value



```
>>> L1 = ['A', 'B', 'C', 'D']
>>> L2 = [2, 4, 6, 8, 10, 12]
>>>
>>> z = zip(L1,L2)
>>>
>>> z
<zip object at 0x7fa3c99af180>
>>>
>>> for x,y in z:
            print(x,y)


A 2
B 4
C 6
D 8
>>> rev = reversed(L1)
>>> next(rev)
'D'
>>> next(rev)
'C'
>>> pow(2,4)
16
>>> 2**4
16
>>> round(12.3333)
12
>>>
```

Recursive Function

- A function calling itself is called recursive function.
- It is same as the recursive function in mathematics.
- A factorial is defined in terms of factorial only; therefore, it is called recursive function.

Q 1)Factorial of a number using Recursion

Example: n ! = 1 * 2 * 3 * 4 * 5 ….. * n

5! = 5 * 4 * 3 * 2 * 1

5! = 5 * 4!

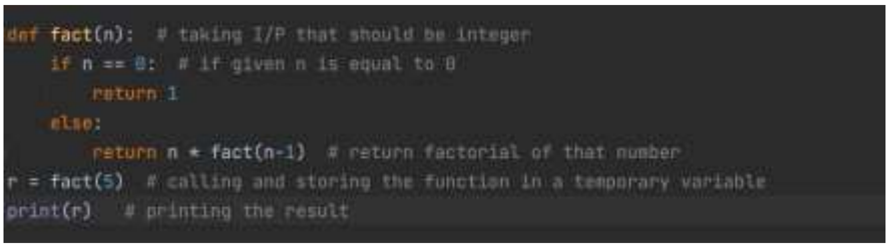
n! = n * (n -1)

- In maths recursive function is also define as

$$\text{fact}(n) = \begin{cases} 1 & \text{if } n=0 \\ n*\text{fact}(n-1) & \text{if } n>0 \end{cases}$$

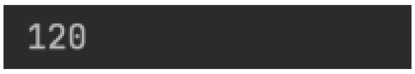
In python we can write the same thing as
Example:

```
def fact ( n ) :
        if n == 0 :
                return 1
        else :
                return n * fact ( n-1)
```



```
def fact(n):  # taking I/P that should be integer
    if n == 0:  # if given n is equal to 0
        return 1
    else:
        return n * fact(n-1)  # return factorial of that number
r = fact(5)  # calling and storing the function in a temporary variable
print(r)  # printing the result
```

Output:



```
120
```

Q 2) Sum of Natural number using Recursion:

```
def sum ( n ) : if
    n == 0 :
        return 0
    else :
        return n + sum ( n-1)
r=sum(5)
print(r)            O/P=15
```

Q 3) Power using Recursion (m^n)

```
def pow( m,n ) : if
    n == 0 :
        return 1
    else :
        return pow(m,n-1)*m
r=pow(5,3)
print(r)                O/P=125
```

Q 4) Fibonacci Series using Recursion:

```
def rfib( n ) : if
          n <= 1 :
        return n
    else :
        return rfib(n-2)+rfib(n-1)
r=rfib(6)
print(r)                O/P=8
```

Q 5) GCD of two number using Recursion:

```
def rgcd( m,n ) : if
    m==n :
        return n elif
    m>n :
        return rgcd(m-n,n)
    else:
        return rgcd(m,n-m) r=rgcd(21,7)
print(r)                O/P=7
```

Searching and Sorting:

Linear Search and Binary Search

Q 1) Write a program in python to search an element using Linear Search method:

Program:

```
def LSearch(list,key):
    for i in
        range(len(list)): if
        list[i]==key:
            return i
    return -1
list=[]
no=int(input("Enter number of elements in list:
")) for i in range(0,no):
    data=input()
    list.append(dat
    a)
print("Elements are: ")
print(list)
print("Enter the search
elements: ") key=input()
result=LSearch(list,key)
if result==-1:
    print("Element is not found. ")
else:
    print("Element is found at position ",result)
```

Output:

```
Enter number of elements in list:
10 10
Sanjo
y 90
78.90
76
34
29
56
Ram
Mada
m
Elements are:
['10', 'Sanjoy', '90', '78.90', '76', '34', '29', '56', 'Ram',
'Madam']
Enter the search elements:
Madam
Element is found at position  9
```

Q 1) Write a program in python to search an element using Binary Search method:

Program:

```
def
   BinarySearch(list,key):
   low=0
   high=len(list)-1
   mid=0
   while low<=high:
      mid=(low+high)//
      2 if
      list[mid]==key:
         return mid
      elif list[mid]<key:
         low=mid+1
      else:
         high=mid-1
```

```
    return -1

list=[]
no=int(input("Enter number of elements in list:
")) for i in range(0,no):
    data=input()
    list.append(data)
print("Elements are: ")
print(list)
list1=sorted(list)
print("Sorted elements are:
") print(list1)
print("Enter the search elements:
") key=input()
result=BinarySearch(list1,key)
if result==-1:
    print("Element is not found. ")
else:
    print("Element is found at position ",result)
```

Output:

```
Enter number of elements in list:
5 12
34
11
23
10
Elements are:
['12', '34', '11', '23', '10']
Sorted elements are:
['10', '11', '12', '23',
'34']
Enter the search
elements: 23
```

```
Element is found at position  3
```

Bubble, Selection and Insertion sorting.

Q 1) Write a program to sort an element using Bubble sort technique:

Program:

```
def BubbleSort(list,no):
    for i in range(0,no-
    1,1):
        for j in range(0,no-i-1,1):
            if list[j]>list[j+1]:
                temp=list[j]
                list[j]=list[j+1
                ]
                list[j+1]=tem
                p

list=[]
no=int(input("Enter the number of elements in list:
")) for i in range(0,no):
    data=int(input())
    list.append(data)
print("Before Bubble sort elements are: ")
print(list)
BubbleSort(list,no)
print("After Bubble sort elements are:
") print(list)
```

Output:

```
Enter the number of elements in list:
10 10
45
11
34
23
87
```

90

```
23
45
78
Before Bubble sort elements are:
[10, 45, 11, 34, 23, 87, 90, 23, 45, 78]
After Bubble sort elements are:
[10, 11, 23, 23, 34, 45, 45, 78, 87, 90]
```

Q 2) Write a program to sort an element using Insertion sort technique:

Program:

```
def InsertionSort(list,no):
    for i in range(0,no,1):
        j=i-1
        x=list[i]
        while j>-1 and list[j]>x:
            list[j+1]=list[j]
            j=j-1
        list[j+1]=x
list=[]
no=int(input("Enter the number of elements in list:
")) for i in range(0,no):
    data=int(input())
    list.append(data)
print("Before Insertion sort elements are: ")
print(list)
InsertionSort(list,no)
print("After Insertion sort elements are: ")
print(list)
```

Output:

```
Enter the number of elements in list:
7 10
4
23
```

```
1
22
78
56
Before Insertion sort elements are:
[10, 4, 23, 1, 22, 78, 56]
After Insertion sort elements are:
[1, 4, 10, 22, 23, 56, 78]
```

Q 3) Write a program to sort an element using Selection sort technique:

Program:

```
def SelectionSort(list,no):
    for i in range(0,no-1,1):
        j=k=i
        for j in range(j,no,1):
            if list[j]<list[k]:
                k=j
        temp=list[i]
        list[i]=list[k]
        list[k]=temp
list=[]
no=int(input("Enter the number of elements in list:
")) for i in range(0,no):
    data=int(input())
    list.append(data)
print("Before Selection sort elements are: ")
print(list)
SelectionSort(list,no)
print("After Selection sort elements are: ")
print(list)
```

Output:

```
Enter the number of elements in list:
10 1
```

```
20
3
23
56
89
32
12
45
30
Before Selection sort elements
are: [1, 20, 3, 23, 56, 89, 32, 12,
45, 30]
After Selection sort elements are:
[1, 3, 12, 20, 23, 30, 32, 45, 56, 89]
```

Errors and Exceptions

What are Exceptions and types of error

- Exceptions are runtime error
- There are three types of error:
  - (a) Syntax Error
  - (b) Logical Error
  - (c) Runtime Error

Syntax Error: -

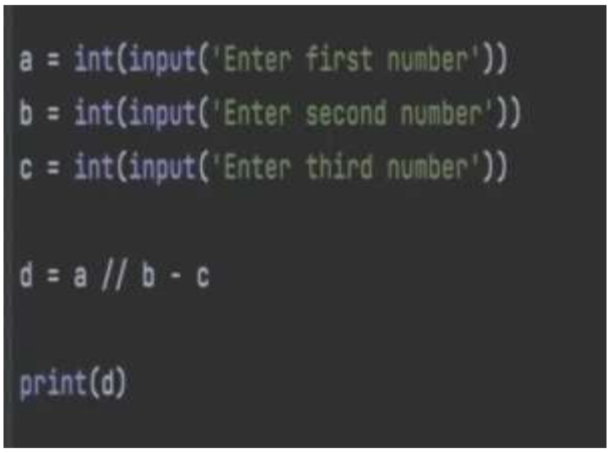
If there is the typing mistake in the program then it will show syntax error if x>y

print(“something”)

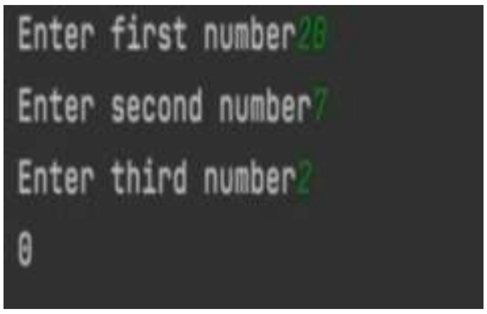
In if statement :(colon) is missing and in print indentation is missing. This type of errors is said as syntax error. If there is syntax error then the program will not run.

Logical Error: -

a = 20
b = 7
c = 2
d = a / b-c
d = 20/7-2 = 4



```
a = int(input('Enter first number'))
b = int(input('Enter second number'))
c = int(input('Enter third number'))

d = a // b - c

print(d)
```



```
Enter first number20
Enter second number7
Enter third number2
0
```

How can we correct this logical error. By adding parentheses to it d = a // (b-c) By doing this the answer will be 4.

Programmer user exception

Developer develops software or app and client uses it

Types of errors

1. ***Syntax Error***
2. ***Logical Error***
3. ***Runtime Error***

A developer has to give error free software to user

1. *Syntax error:* Developer is responsible for syntax error.

2. *Logical error:* These errors depict flaws in the logic of program The program might be using wrong formula or the design of the program is wrong itself. Logical errors are not detected either by python compiler or Pam. The programmer is solely responsible for them.

3. *Runtime error:* The reason could be giving invalid input File not available, no connection, etc.

- Runtime errors are caused due to mishandling a software at client side
- Due to runtime error program crashes stops suddenly
- User is responsible for runtime error
- But when developer gives robust idea, it can be handled means Exception handling given in program.

How to handle the runtime error?

Example:

A printer:

When the manufacturer gives any property on which the printer will blinks if paper is not inserted. It blinks because it needs some input means it is raising an exception. Likewise, also a developer can raise an exception asking user to check if there is anything missing.

Examples of Exception

1. ***Index error***
2. ***Key error***
3. ***Value error and type error***
4. ***Zero division error***

When these errors occur L
```
= [10, 20, 30, 40, 50]
index=int (input ('enter an index')                    #user has to enter value here
print(L[index])
```

Program:

```
L=[10, 20, 30, 40, 50]                  # LIST OF ELEMENT
index= int(input('Enter index'))        # ASKING USER TO ENTER INDEX
print(L[index])
```

- If we print valid index, it will print else
- If not, valid index it will print error
- So, user itself is responsible for giving wrong index

What our program can do?

- It can give a message "please enter an index which is from 0 to 4 only".
- Now there in output the error will be given as traceback, means it is showing in which line theerror is appearing.

Index error*:* List index out of range

↓ ↓

Name of the error. This is the description of the error.

Value Error and Type error:

```
val= int('abc')        # here we are doing type conversion
res='2'+3              # this type is called as value error
```

↓

We are saying here '2'+3 2 to
be concatenate with 3
But it is a string how can a type int be concatenated with string.

```
a=10
s='number'
print(s+a)
```

$\longrightarrow$ *THIS IS TYPE ERROR*

Key Error:

```
    D=[1:'a', 2:'b', 3:'c']              # dictionary list
    key=int(input('Enter a key'))
    print(D[key])

    D={1:'a', 2:'b', 3:'c'}
    key= int(input('Enter a key'))
    print(D[key])                        # it will show key error
```

Two possibilities of getting an error

1. Not entering a proper key and
2. Entering string instead of integers

Zero Division Error:

```
a= int(input('Enter first number')
b=int(input('Enter second number')
res=a//b
print(res)              # it will show zero division error
```

Exception Handling Construct

We will learn how to handle exception in python.

Python skeleton for try-except.

Syntax:

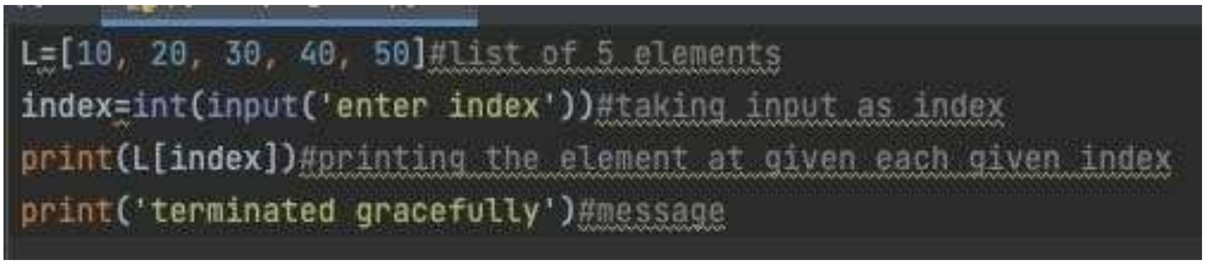
```
1)———
2)———
3)———
Try 4)—
—-
5)——-
6)——-
Except:
7)——-
8)——-
9)———
10)——-
```

- This may cause Exception. So, the lines which may cause Exception are written inside the block.
- Inside except we may print the message about the exception.
- The program starts running from the first line then continue 2,3,4,5,6 then it will not go to line 7 as there is no error.

What happens if there is no error?

- First line, second line gets executed then third fourth. Suppose there is an error in 5th line. The exception raised in 5th line then it will not execute in 6th line.
- It will directly jump to 7th line I.e. except block.
- If there is any exception in try block then it will execute except block. If condition gets in 4th, 5th and ,6th will not execute and remaining 8 9 10 execute properly.
- The programme executes successfully it will not get terminated or crash abruptly it gets terminated gracefully.
- If suppose try and accept has not been used.
- If there is an exception. In the 4th line. The program will crash Program.

Program1:



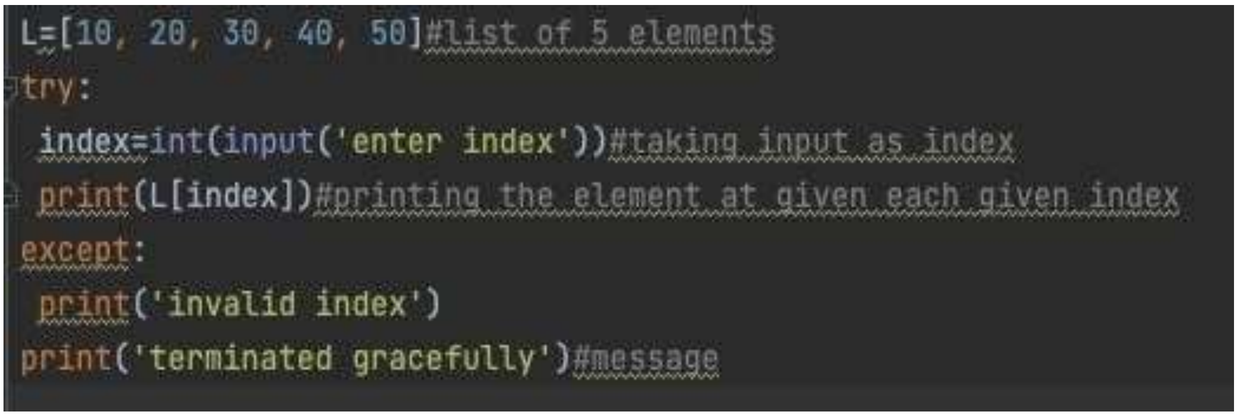
```
L=[10, 20, 30, 40, 50]#list of 5 elements
index=int(input('enter index'))#taking input as index
print(L[index])#printing the element at given each given index
print('terminated gracefully')#message
```

Output:

```
enter index3
40
terminated gracefully enter
index9
Traceback (most recent call last):
File "/Users/sanjuvai/PycharmProjects/pythonProject/python programs.py", line 3, in <module> print(L[index])  #printing the element at given each given index
IndexError: list index out of range
```

If the index is out of range, it will get an exception error i.e index error If we give an invalid index, it will raise an exception

Program2:



```
L=[10, 20, 30, 40, 50]#list of 5 elements
try:
 index=int(input('enter index'))#taking input as index
 print(L[index])#printing the element at given each given index
except:
 print('invalid index')
print('terminated gracefully')#message
```

Output 1:

```
enter index3
 40
terminated gracefully
```

Output2:

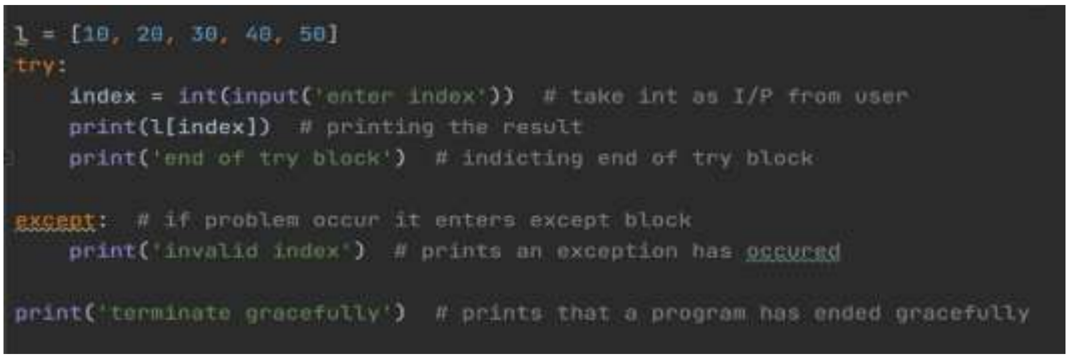
enter index9 invalid index

terminated gracefully

Now the program does not crash it executes safely.

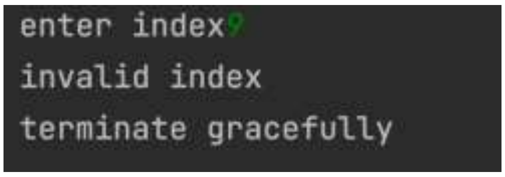
Multiple Exceptions

- If the Index is not given properly, it'll give **index error.**
- You can handle a particular exception as well by simply defining except with that error, then this except block will not handle any other except block other than what is defined.
- If exception block is not defined it'll handle all type of exceptions.

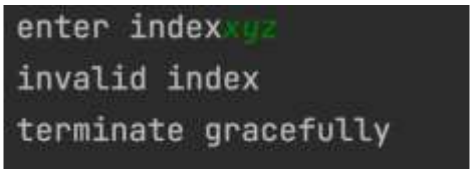


```
l = [10, 20, 30, 40, 50]
try:
    index = int(input('enter index'))  # take int as I/P from user
    print(l[index])  # printing the result
    print('end of try block')  # indicting end of try block

except:  # if problem occur it enters except block
    print('invalid index')  # prints an exception has occured

print('terminate gracefully')  # prints that a program has ended gracefully
```

output 1 ***output 2***

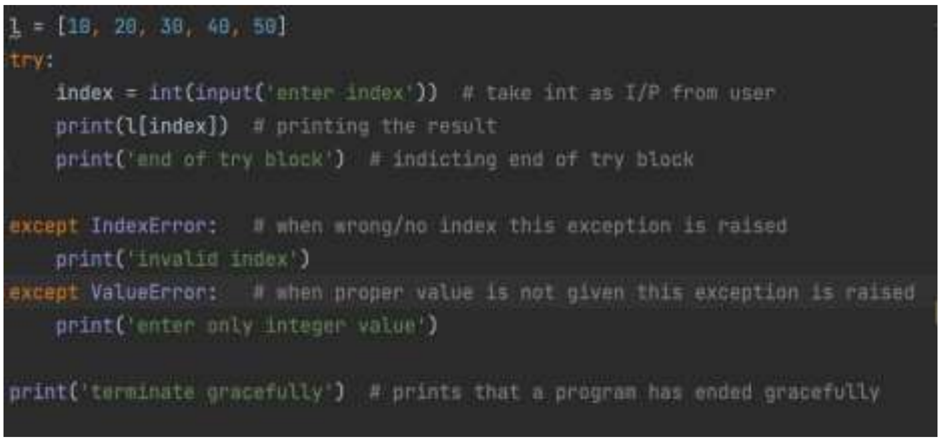


```
enter index9
invalid index
terminate gracefully
```



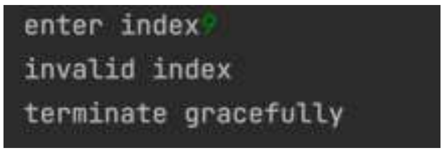
```
enter indexxyz
invalid index
terminate gracefully
```

- You can write multiple except block to handle different type of exception.

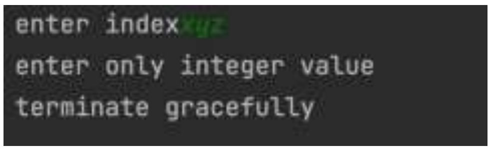


```
l = [10, 20, 30, 40, 50]
try:
    index = int(input('enter index'))  # take int as I/P from user
    print(l[index])  # printing the result
    print('end of try block')  # indicting end of try block

except IndexError:   # when wrong/no index this exception is raised
    print('invalid index')
except ValueError:   # when proper value is not given this exception is raised
    print('enter only integer value')

print('terminate gracefully')  # prints that a program has ended gracefully
```

Output 1 ***output 2***

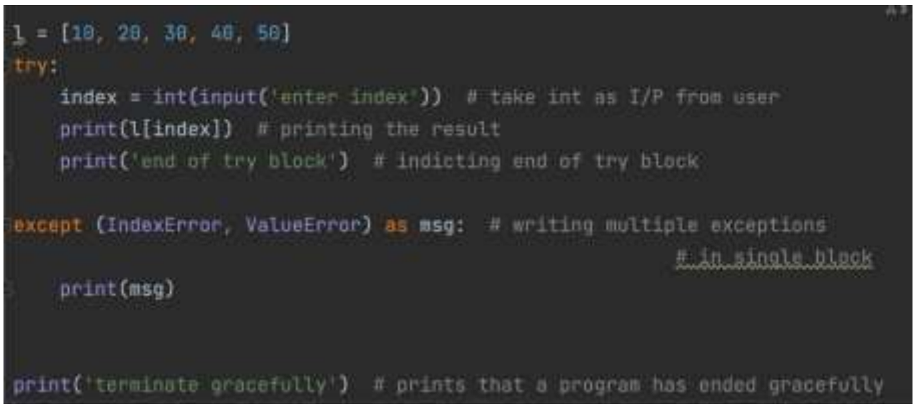


```
enter index9
invalid index
terminate gracefully
```



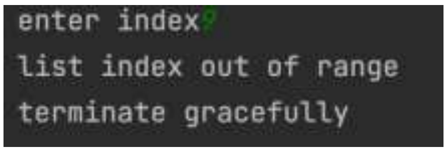
```
enter indexxyz
enter only integer value
terminate gracefully
```

- Instead of writing two except block we can write a single except block that can handle both the exception.

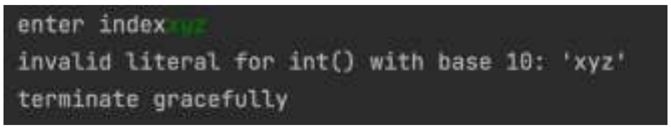


```
l = [10, 20, 30, 40, 50]
try:
    index = int(input('enter index'))  # take int as I/P from user
    print(l[index])  # printing the result
    print('end of try block')  # indicting end of try block

except (IndexError, ValueError) as msg:  # writing multiple exceptions
                                                              # in single block
    print(msg)


print('terminate gracefully')  # prints that a program has ended gracefully
```

Output 1 ***output 2***



```
enter index9
list index out of range
terminate gracefully
```



```
enter indexxyz
invalid literal for int() with base 10: 'xyz'
terminate gracefully
```

Why Try Except

Can we handle our program without using try and except?

Yes

Why try except is required

ZeroDivisionError

a=int(input('enter first number')) #taking integer input and storing it into a
b=int(input('enter second number')

res=a//b # floor div but this div may cause error if a number is divided by zero print(res)

#This program may cause zero division error

a=int(input('enter first number')) #taking integer input and storing it into a
b=int(input('enter second number')

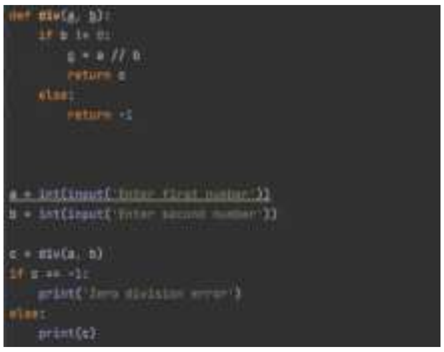
c=div(a, b)

If c==-1

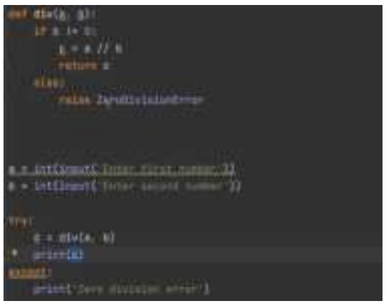
print('zero division error') else:

print(c)

It's working without Try-Except also. But problem arises when we use codes If a function able to divide it will. If it's not able to divide then it raises the exception. So, there are two possibilities so, here div function is also having two possibilities.



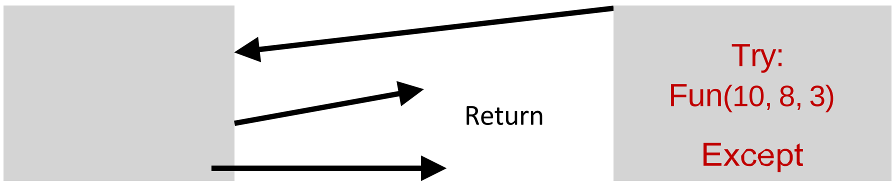
```
def div(a, b):
    if b != 0:
        c = a // b
        return c
    else:
        return -1


a = int(input('Enter first number'))
b = int(input('Enter second number'))

c = div(a, b)
if c == -1:
    print('Zero division error')
else:
    print(c)
```



```
def div(a, b):
    if b != 0:
        c = a // b
        return c
    else:
        raise ZeroDivisionError


a = int(input('Enter first number'))
b = int(input('Enter second number'))

try:
    c = div(a, b)
    print(c)
except:
    print('Zero division error')
```

The output will be same but the internal working program will chang

fun(a,b,c) call main



Raise

When you call a function there are two things possible: Values return or Exception raised

Main block should try and except

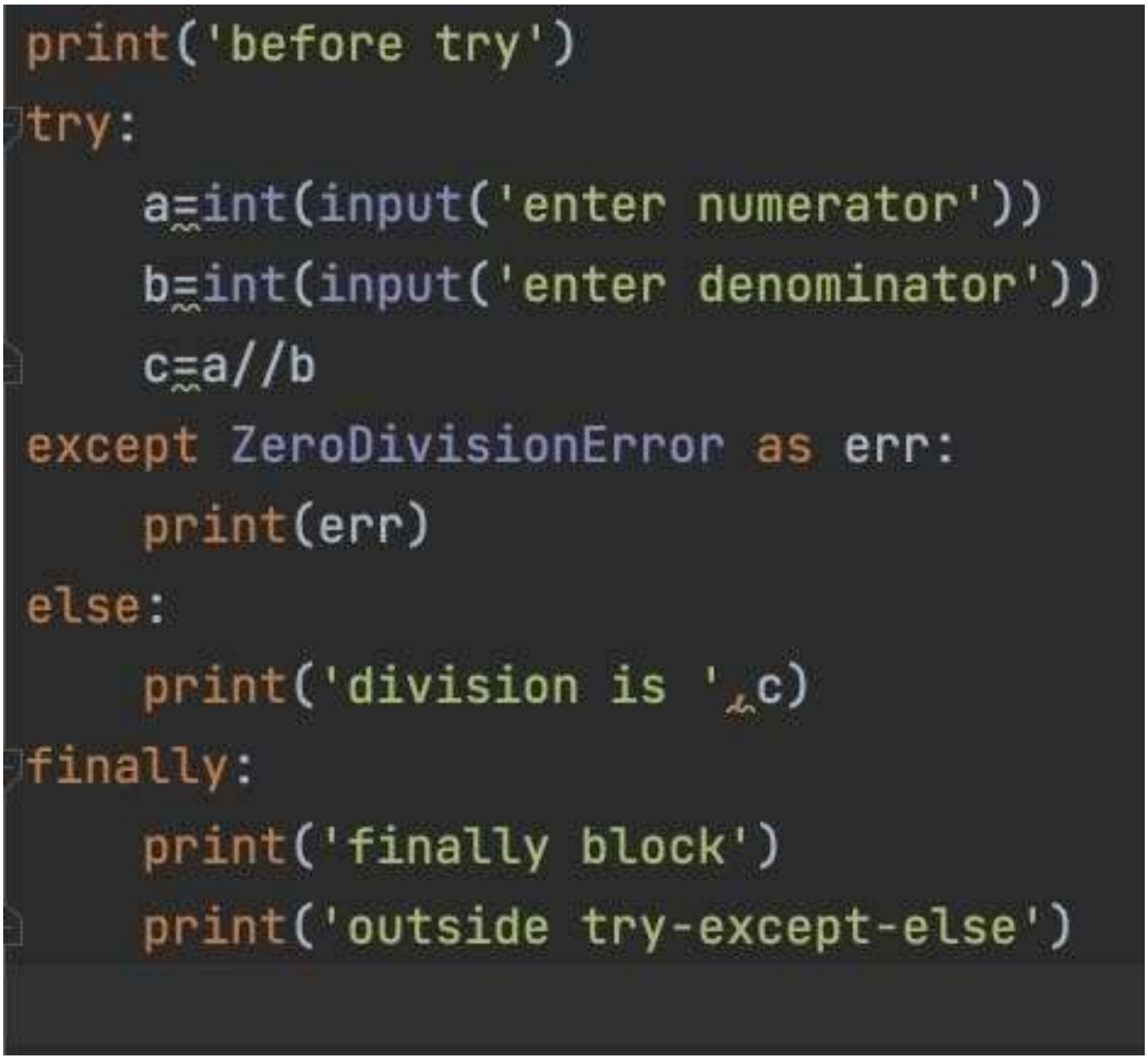
So, this is the reason why try and except is introduced.

So, when we communicate between two functions it's difficult to work using if and so, try and except is suitable.

Finally block

As we have already seen try except else, in previous lectures. Now we will see finally. The difference between else and finally is that else block is executed if there is no exception and finally block will execute if there is an exception. So, it is independent of the exception and it guarantees that it will always execute.

Program: Input



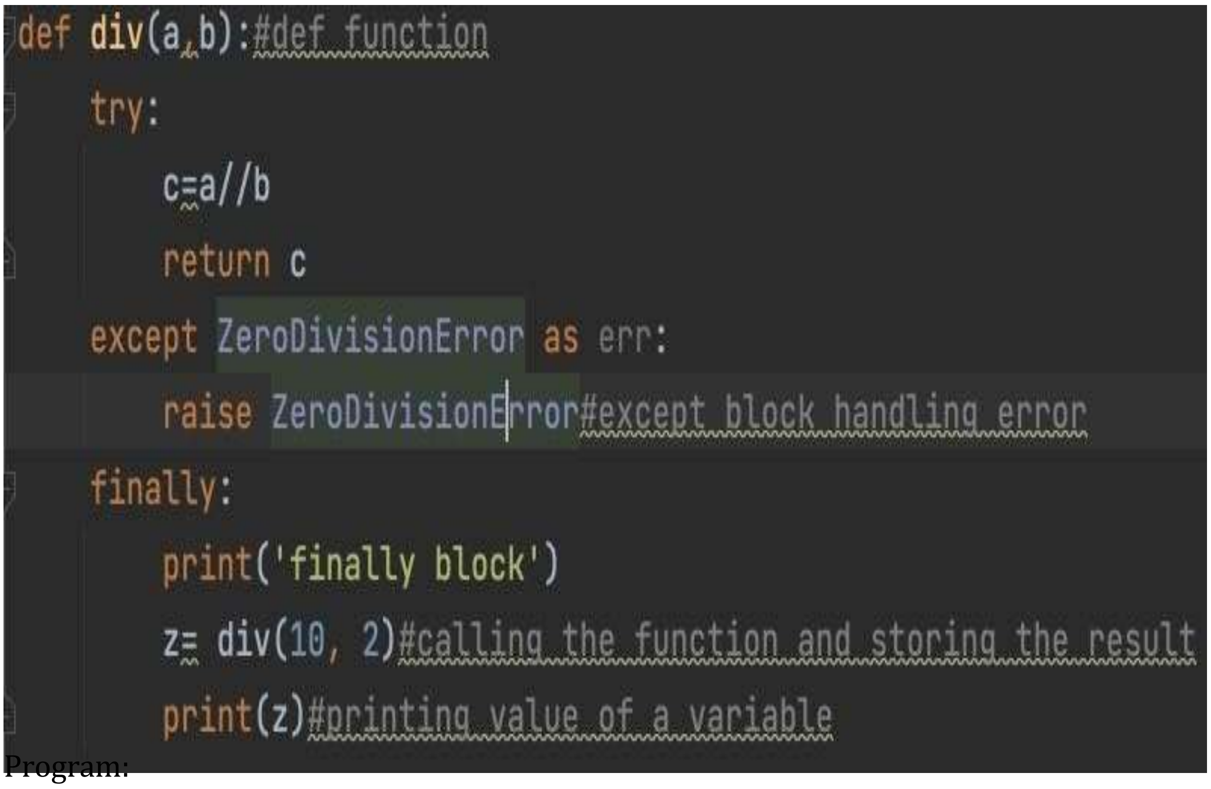
```
print('before try')
try:
    a=int(input('enter numerator'))
    b=int(input('enter denominator'))
    c=a//b
except ZeroDivisionError as err:
    print(err)
else:
    print('division is ',c)
finally:
    print('finally block')
    print('outside try-except-else')
```

Output

before try

enter numerator2 enter denominator3 ('division is ', 0) finally block

outside try-except-else.



```
def div(a,b):#def function
    try:
        c=a//b
        return c
    except ZeroDivisionError as err:
        raise ZeroDivisionError#except block handling error
    finally:
        print('finally block')
        z= div(10, 2)#calling the function and storing the result
        print(z)#printing value of a variable
```

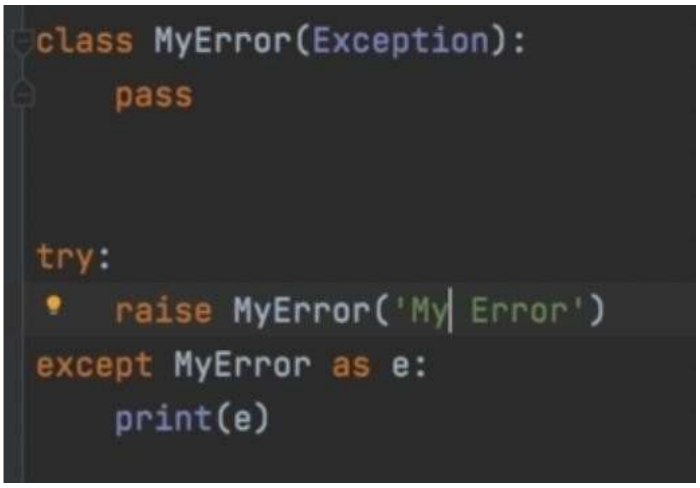
Program:

Finally, is useful inside the function

User define Exception

- When we use python on real life application. Let's take an example for age. Age can't be entered in negative value.
- What if the user gave negative age as a input. But these types of constraints cannot be applied in the python programs automatically
- To handle this type of exception we use user define exception
- A program automatically terminates after showing which inbuilt exception has occurred while executing the program when it reaches into an undesired state. We can stop it by using user define exception.

- User defined exceptions can be implemented by raising an exception explicitly,



```python
class MyError(Exception):
    pass


try:
    raise MyError('My Error')
except MyError as e:
    print(e)
```

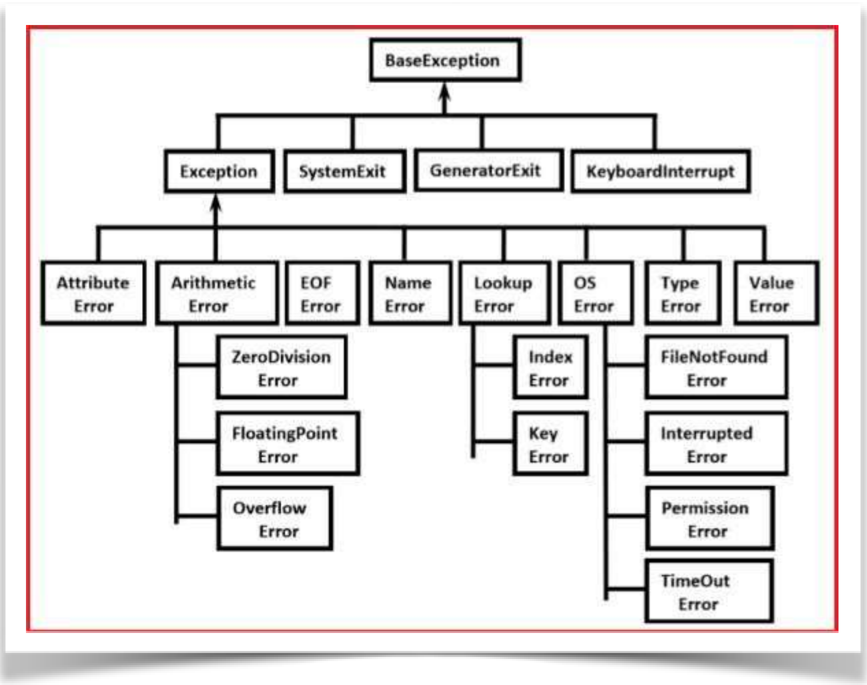
.

- The syntax for raise statement is raise ExceptionName. When an error is raised, the code in the except block should handle the exception otherwise it will cause error in the program.
- So, we can see that after using raise statement in try except block, the program gives correct output in both the cases. The diagrams show the built in exception.

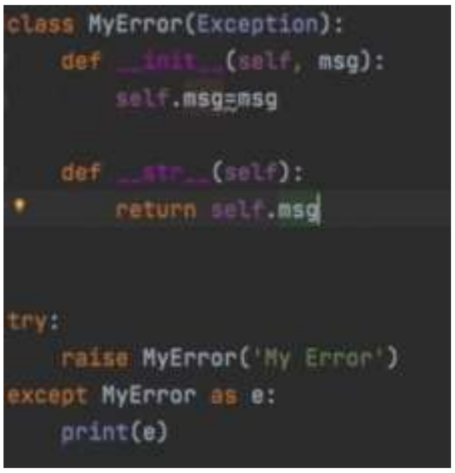
We have discussed about, exception , athematic error , zero division etc …

The built-in exceptions listed below can be generated by the interpreter or built- in functions

User code can raise built-in exceptions. This can be used to test an exception handler



A single try statement can have multiple except statements. This is useful when the try block contains statements that may throw different types of exceptions.



```
class MyError(Exception):
    def __init__(self, msg):
        self.msg=msg

    def __str__(self):
        return self.msg


try:
    raise MyError('My Error')
except MyError as e:
    print(e)
```

Principles Of Object-Oriented Programming (OOP'S):

It is a paradigm (oop's is a paradigm)

What is object?

- Everything in the world is tangible object or intangible object.
- Example: student, car, washing machine, movie, order etc.
- So, we believe that anything in the world can be object.
- By defining in terms of its properties and methods.

Encapsulation:

- Means combining all the related things together.
- Example: car, Tv, Washing machine.

Abstraction:

- Means showing required features and hiding internal details.
- To make an object first we design the classes.
- Class: a Class is a blueprint of an object.

Inheritance:

- Inheriting the features of existing class.
- Example: parent->child.

Polymorphism:

- One name and different actions
- By using a single item, we can refer multiple things together.

Class v/s Object

- In OOP's we consider everything as an object and for every object there is a class.
- So, class is a definition of an object. It is also known as blueprint, design, plan etc...
- Every object will have 2 things

1) Properties / Attributes
2) Methods / Operations / Function

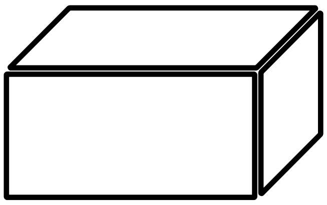
Example:

- The property of a cuboid are l , b , h
- The methods of a cuboid are lidarea( ), volume ( ), totalarea( )
- A sudo code for cuboid

is Class cuboid

```
properties :
        length
        breathe
        height

Methods :
        lidarea( )
        totalarea( ) volume
        ( )
```

How to Write a class

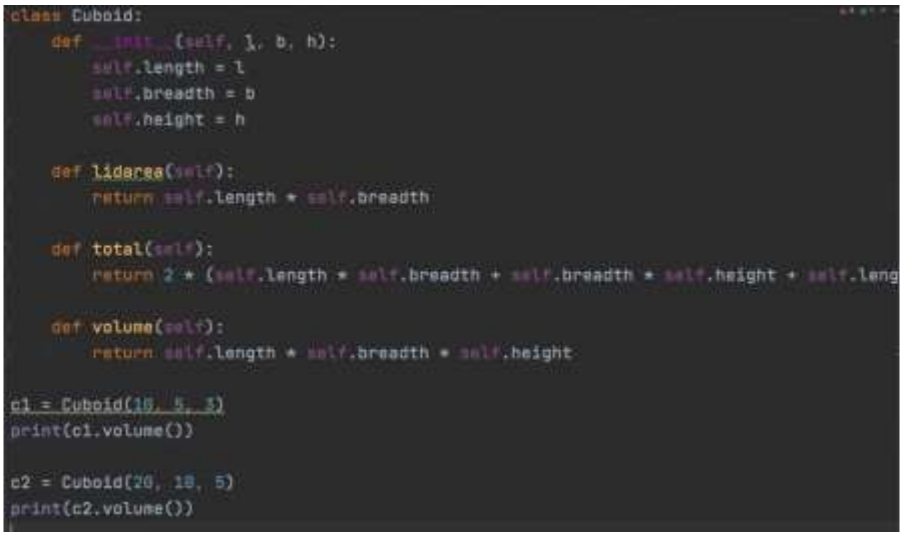


Methods:

lidarea( ) = l *b
total( ) = l * b * b * h * l * h
volume = l*b*h

Writing a class in python:

- When we write a variable, we also need to initialize it. Initialization is done in constructor (java) but in python we can write constructor method.
- Initializing and declaring both should be done in python.



```
class Cuboid:
    def __init__(self, l, b, h):
        self.length = l
        self.breadth = b
        self.height = h

    def lidarea(self):
        return self.length * self.breadth

    def total(self):
        return 2 * (self.length * self.breadth + self.breadth * self.height + self.leng

    def volume(self):
        return self.length * self.breadth * self.height

c1 = Cuboid(10, 5, 3)
print(c1.volume())

c2 = Cuboid(20, 10, 5)
print(c2.volume())
```

- First parameter should be self, it is defined to declare and initialize it and properties are defined inside the function __init__and this is standard function that acts as the constructor in class.

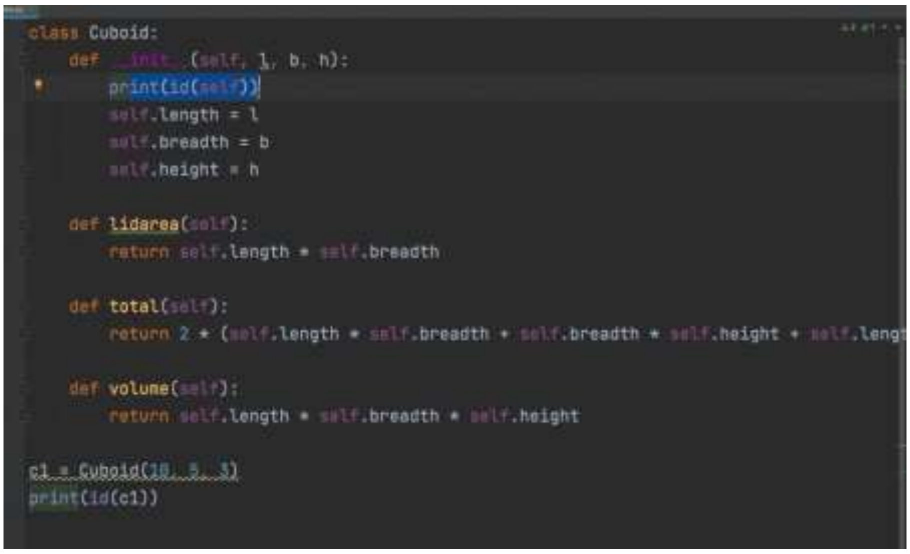
Create object in python

c1=cuboid(10,5,3)

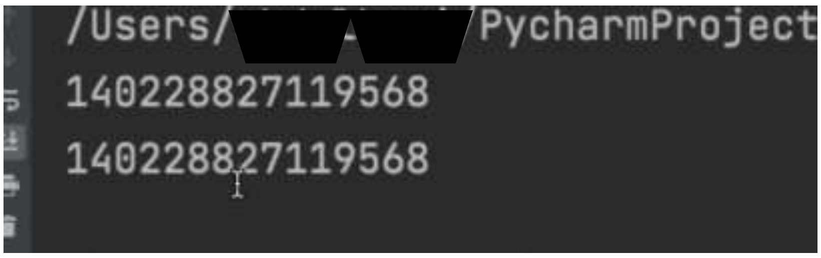
Self and Construct

Self

- Self is the **reference** to the current value.
- It is the first argument of the constructor.
- If self is not given PVM will take the 1st parameter value as self by default.
- You can write different names for self but self is recommended.
- When you create 2 object the self will refer to the current object that is being called I.e., the self-value will be same for these 2 objects but it will show values of the method being called (i.e., 1st or 2nd).
- You can verify this by checking the ID of self in both method call.



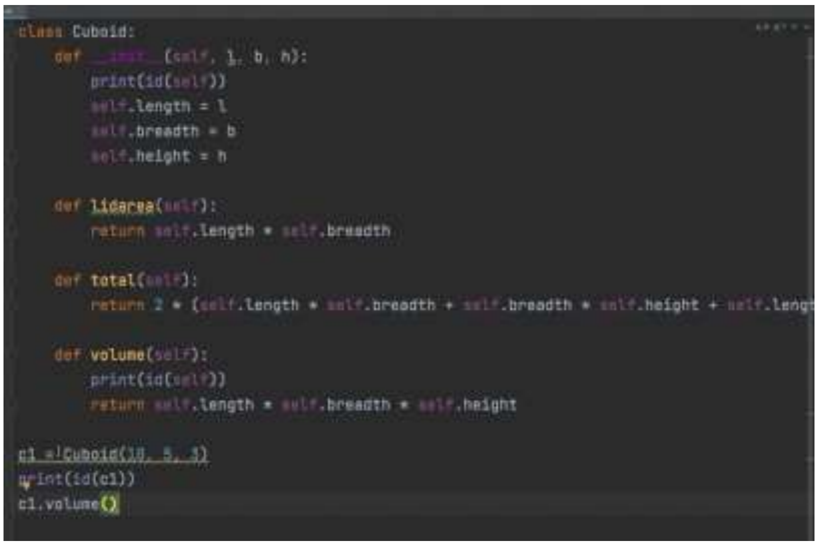
```
class Cuboid:
    def __init__(self, l, b, h):
        print(id(self))
        self.length = l
        self.breadth = b
        self.height = h

    def lidarea(self):
        return self.length * self.breadth

    def total(self):
        return 2 * (self.length * self.breadth + self.breadth * self.height + self.leng

    def volume(self):
        return self.length * self.breadth * self.height

c1 = Cuboid(10, 5, 3)
print(id(c1))
```



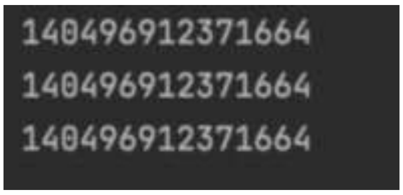
```
/Users/[redacted]/PycharmProject
140228827119568
140228827119568
```

Constructor

- In a method if _ _init_ _ is used then that method becomes constructor method as _ _init_ _ is a constructor.
- When you create an object of this class then automatically constructor method is called and initialized.
- If a class having attributes, then its mandatory to have _ _init_ _ method. Therefore, every class must have _ _init_ _
- When _ _init_ _ is not given in a class then PVM will provide default constructor which will not have any property.
- You can write more than 1 constructor but only 1 will execute but it's not recommended to do so.
- You can give default argument in parameter and when you call the method you can either give values or it will consider the default values i.e. you can make constructor using different Parameters.
- Constructors are not overloaded.



```
class Cuboid:
    def __init__(self, l, b, h):
        print(id(self))
        self.length = l
        self.breadth = b
        self.height = h

    def lidarea(self):
        return self.length * self.breadth

    def total(self):
        return 2 * (self.length * self.breadth + self.breadth * self.height + self.leng

    def volume(self):
        print(id(self))
        return self.length * self.breadth * self.height

c1 = Cuboid(10, 5, 3)
print(id(c1))
c1.volume()
```



```
140496912371664
140496912371664
140496912371664
```

Program explanation (1 and 2):

- A class of cuboid is created, in method we are writing a constructor it takes 1st parameter as self, and we are initialization it by writing its properties
  i.e. length, breath and height, we are printing the ID of self as well to show that only one self is used for multiple methods call.
- Then we are defining a method for lidarea, total area and volume and initialization it with self.
- When printing the results for lidarea, total area and volume we can see that the ID of self is constant while answer for theses 3 methods are different.

Types of Variables and Methods

In python there are different types of variables and methods

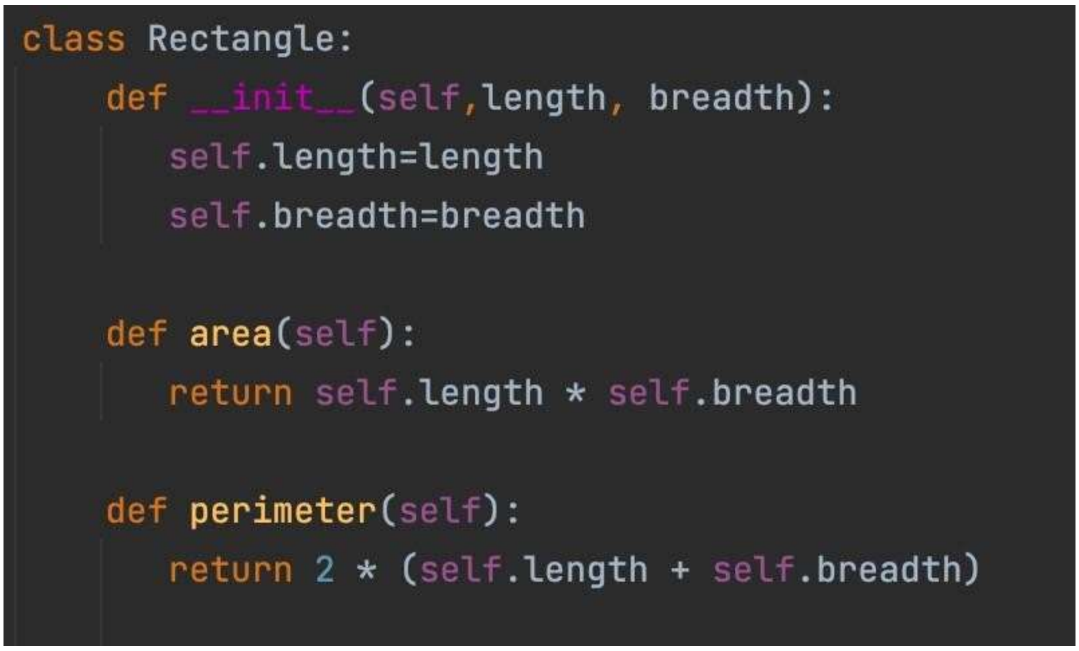
Variables:

- Instance variable
- Static variable

Methods:

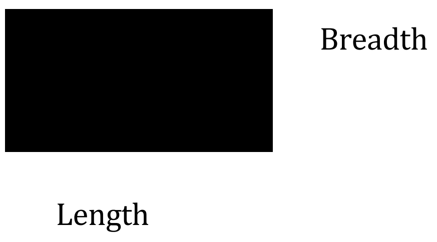
- Instance method
- Class method
- Static method

Instance:



```
class Rectangle:
    def __init__(self,length, breadth):
        self.length=length
        self.breadth=breadth

    def area(self):
        return self.length * self.breadth

    def perimeter(self):
        return 2 * (self.length + self.breadth)
```

Example: Rectangle

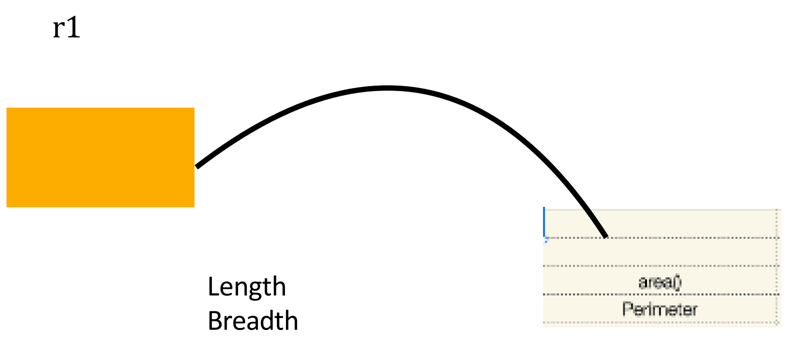


area= length*breadth

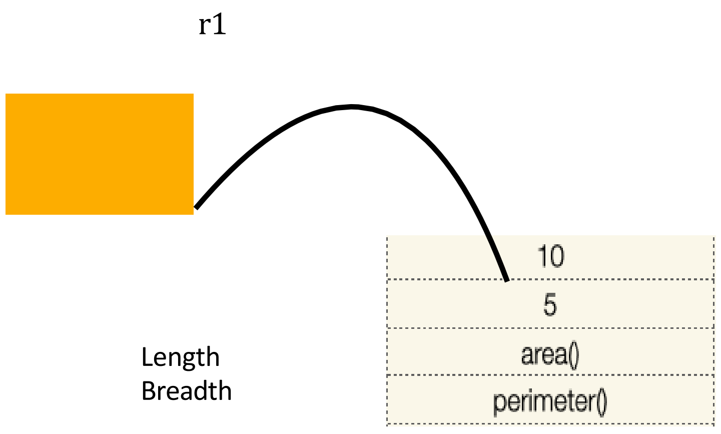
Creating a variable for rectangle

r1= rectangle()

# Automatically calls the constructor of the class. In the memory it looks like



It will form like this



r2= rectangle (15,7) # another object

So again, one more object screamed

in memory These are called instance of

class

Each instance will have its own members

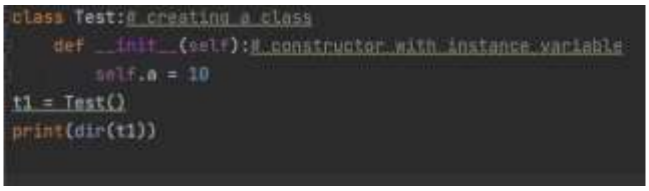
The parameter that is always taken by instance method is self and using self they access instance variable and the methods which are accessing them are called as instance methods.

Ways of creating instance variables.

So, there are multiple ways we can declare instance variable.

Program:

Input:

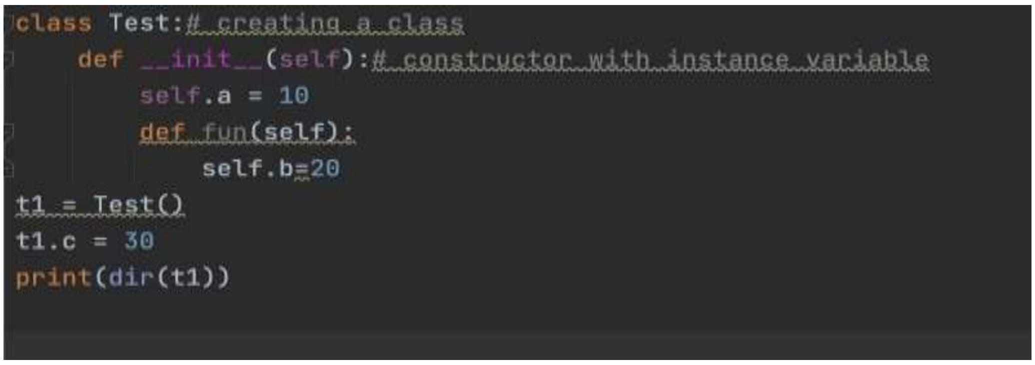


```
class Test:# creating a class
    def __init__(self):# constructor with instance variable
        self.a = 10
t1 = Test()
print(dir(t1))
```

Output:
['__doc__', '__init__', '__module__', 'a'] So, it is having one variable that is a.

Program

Input:



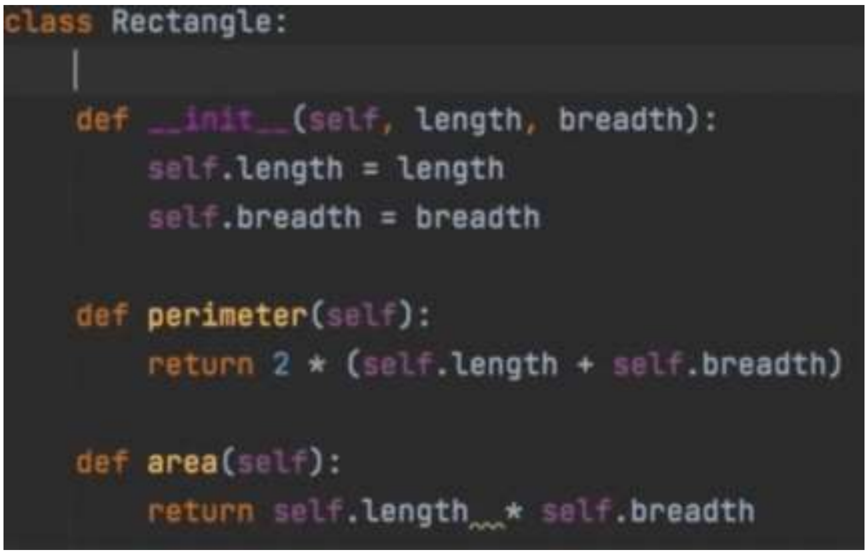
```
class Test:# creating a class
    def __init__(self):# constructor with instance variable
        self.a = 10
        def fun(self):
            self.b=20
t1 = Test()
t1.c = 30
print(dir(t1))
```

Output:
['_doc_', '_init_', '_module_', 'a', 'c']

- It is still having a and function but b is not there for that function should be called We have added one more variable to it so, it means we can declare and initialise Instance variable in init method also and other method also.
- Instead of self we can use the variable name and.
- Self is not a keyword it can be anything. The first we are writing is constructor is called as (instance variable)

- We can give anytime instead of self
- If a method is accessing instance variable, then that method becomes instance method.

Class and Static Variable


```
class Rectangle:

    def __init__(self, length, breadth):
        self.length = length
        self.breadth = breadth

    def perimeter(self):
        return 2 * (self.length + self.breadth)

    def area(self):
        return self.length * self.breadth
```

When we declare a variable inside a class but outside any method, it is called as class or static variable in python.

- self.length and breath are
  instance variable.def
  perimeter(self)

- Class variables can be accessed from within the class as well as from outside the class.

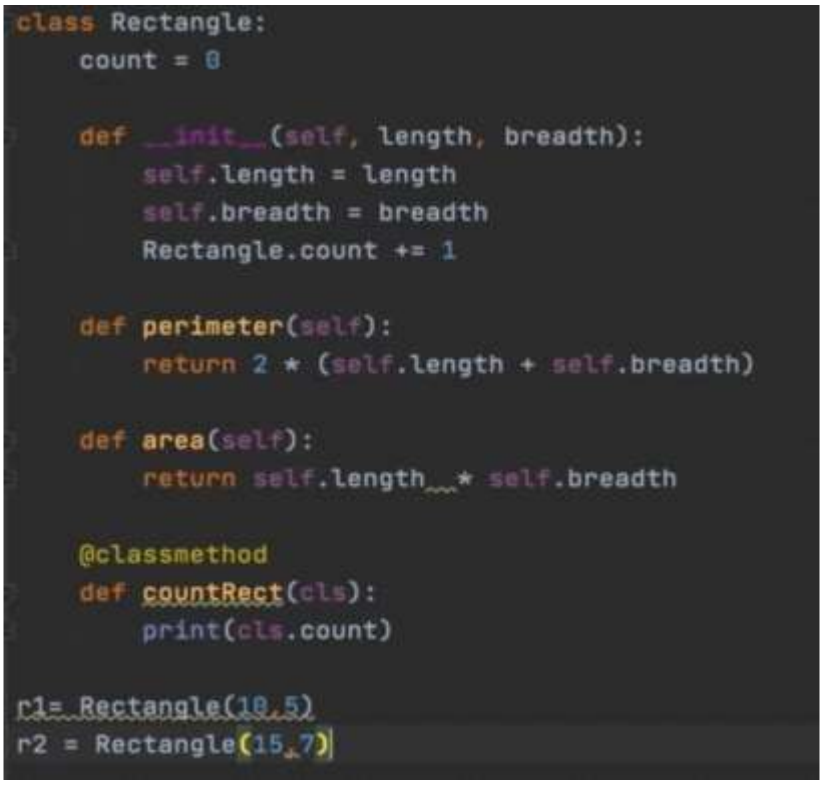
class and static method can access class variables. Each method has its own way of accessing static variables.

There are two ways of accessing class variables from inside class method-

- cls.ClassVariable
- ClassName.ClassVaraiable

There is only one way to access class variables from static method-

- ClassName.ClassVariable

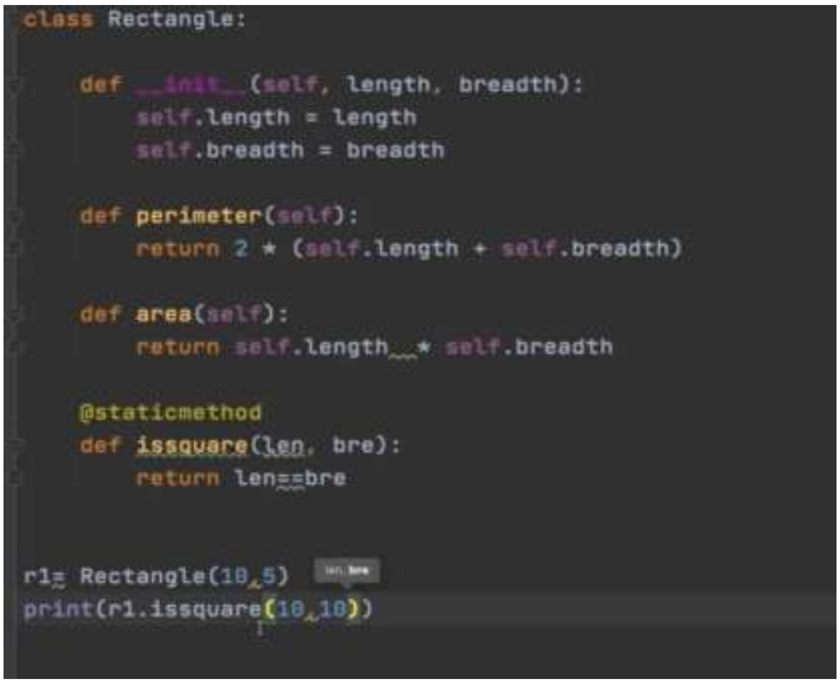


```
class Rectangle:
    count = 0

    def __init__(self, length, breadth):
        self.length = length
        self.breadth = breadth
        Rectangle.count += 1

    def perimeter(self):
        return 2 * (self.length + self.breadth)

    def area(self):
        return self.length * self.breadth

    @classmethod
    def countRect(cls):
        print(cls.count)

r1= Rectangle(10,5)
r2 = Rectangle(15,7)
```

- Here count= 0
- It is known as class or static cause its not inside the method
- The counter belongs to rectangle r1 and r2 that means count variable is commonto both objects (r1 and r2)
- Count will belong to class (Rectangle) so it is also static(fixed, single copy to bothr1,r2)
- @classmethod (decorator)

Static Methods

- Instance method access instance variable,class method access class variable.
- This methods are like global function.if there is anyrelation in the class we can put inside the class.
- The staticmethod() returns a static method for a function passed asthe parameter.



```
class Rectangle:

    def __init__(self, length, breadth):
        self.length = length
        self.breadth = breadth

    def perimeter(self):
        return 2 * (self.length + self.breadth)

    def area(self):
        return self.length * self.breadth

    @staticmethod
    def issquare(len, bre):
        return len==bre


r1= Rectangle(10,5)
print(r1.issquare(10,10))
```

- ***Example:*** Given rectangle is square or not.
- If the length and breath both are equal then it is said assquare.
- This program will return true or false.
- It is not accessing instance variable or class variable suchmethods are said as static method (its doesn't access).
- We can use the decorator @staticmethod. It is not compulsory. It will just help to read (readable).

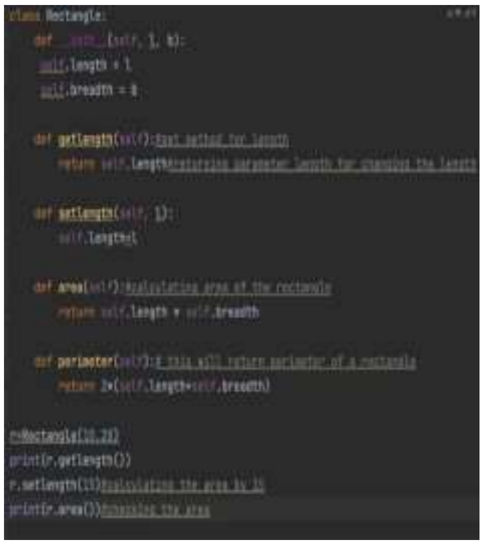
Accessors and Mutators

- These are the types of methods that can be written inside any class.
- Accessor methods are useful for reading the property of a class of an object.
- Mutator is used for writing and updating properties of class and its objects.
- So, these methods can be called as reading and writing methods
- These methods are followed by object-oriented programming

Example: class Rectangle

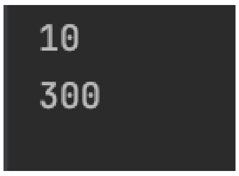
Program:

Input:



```
class Rectangle:
    def __init__(self, l, b):
        self.length = l
        self.breadth = b

    def getLength(self):#get method for length
        return self.Length#returning parameter length for changing the length

    def setLength(self, l):
        self.Length=l

    def area(self):#calculating area of the rectangle
        return self.length * self.breadth

    def perimeter(self):# this will return perimeter of a rectangle
        return 2*(self.length+self.breadth)

r=Rectangle(10,20)
print(r.getLength())
r.setLength(15)#calculating the area by 15
print(r.area())#printing the area
```

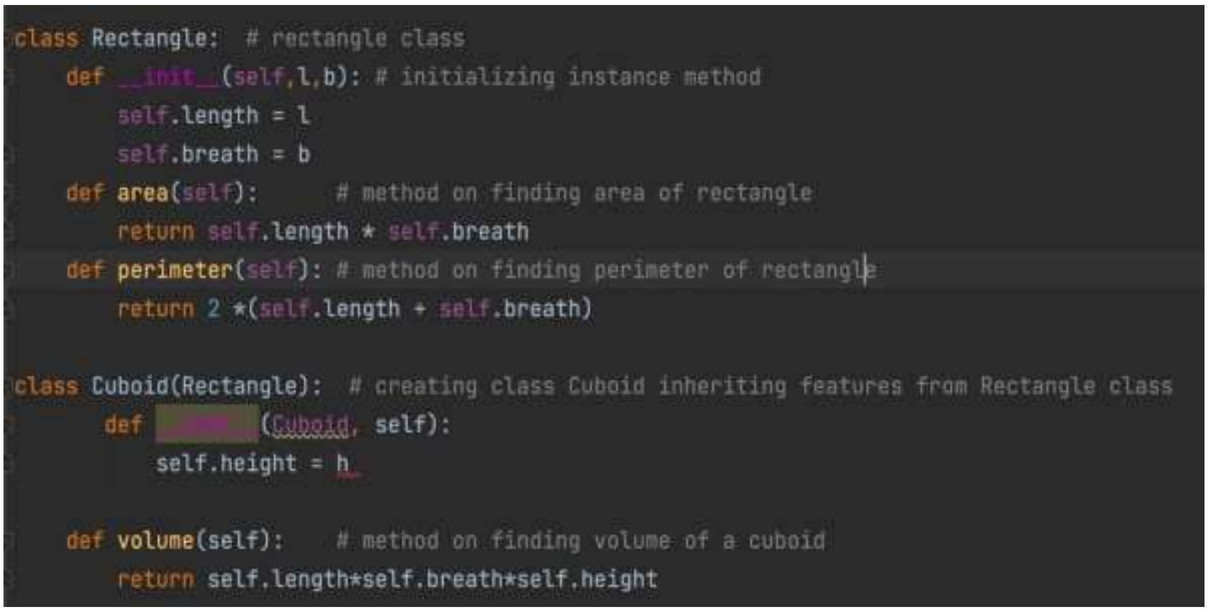
Output:



```
10
300
```

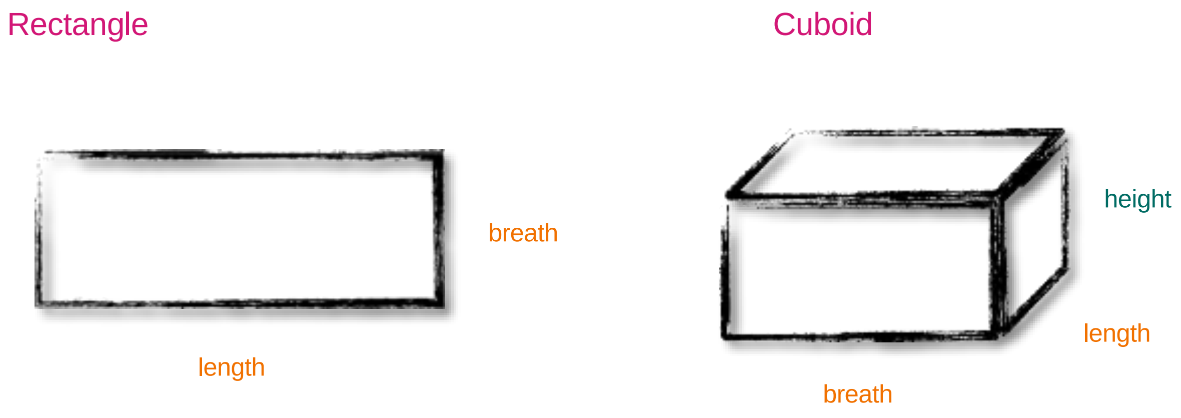
Introduction to Inheritance

- Inheritance is acquiring features in a class from an existing class.
- Inheritance is like ***parent - child relation***, a child class can have all of its parentfeatures plus his own features.
- The main advantage of Inheritance is ***reusability*** of code.
- Every class in python directly or indirectly is inheriting from **object class.**



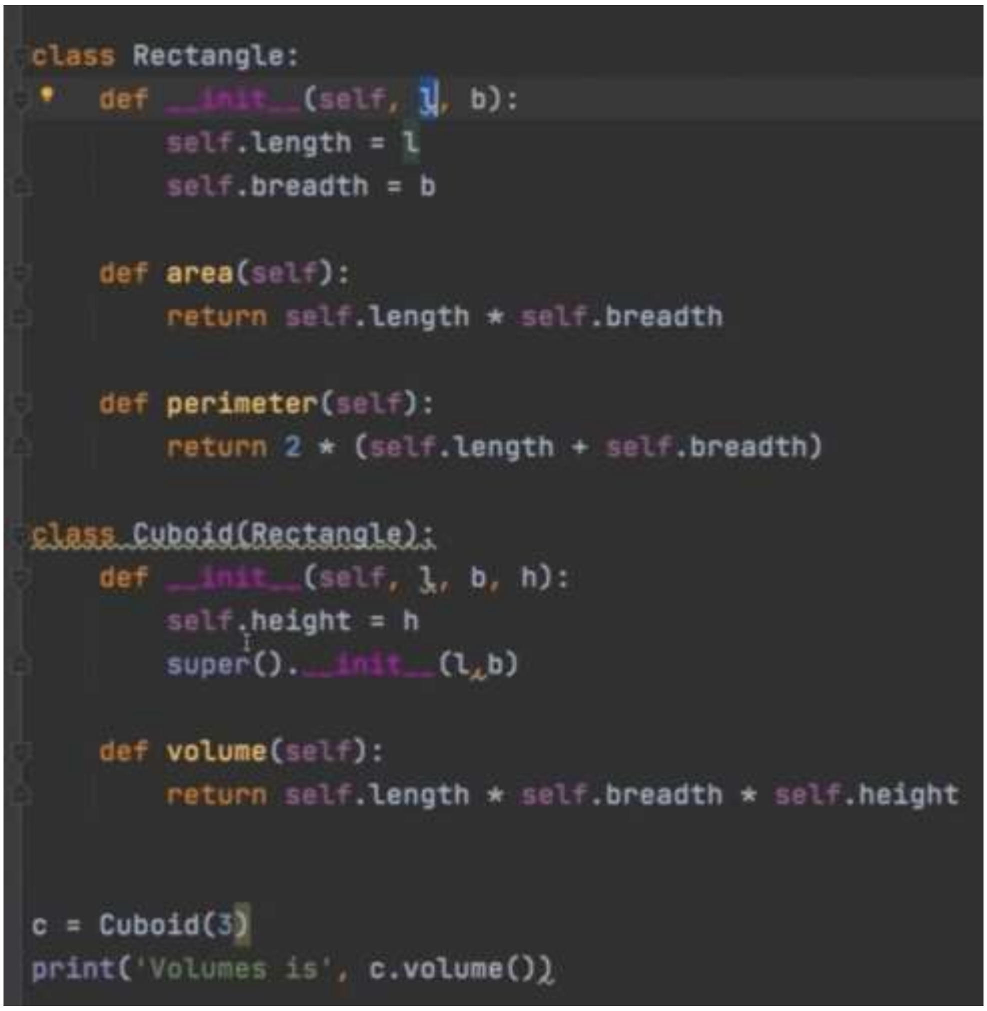
```
class Rectangle:  # rectangle class
    def __init__(self,l,b): # initializing instance method
        self.length = l
        self.breath = b
    def area(self):      # method on finding area of rectangle
        return self.length * self.breath
    def perimeter(self): # method on finding perimeter of rectangle
        return 2 *(self.length + self.breath)

class Cuboid(Rectangle):  # creating class Cuboid inheriting features from Rectangle class
       def __init__(Cuboid, self):
           self.height = h

    def volume(self):     # method on finding volume of a cuboid
        return self.length*self.breath*self.height
```

- Suppose you want to access features of rectangle in cuboid as they are almostsame then you can do the following:

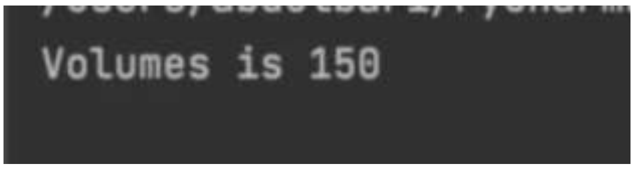


Writing Construct Inheritance

- Let's understand the concept of construct with an example



```
class Rectangle:
    def __init__(self, l, b):
        self.length = l
        self.breadth = b

    def area(self):
        return self.length * self.breadth

    def perimeter(self):
        return 2 * (self.length + self.breadth)

class Cuboid(Rectangle):
    def __init__(self, l, b, h):
        self.height = h
        super().__init__(l,b)

    def volume(self):
        return self.length * self.breadth * self.height


c = Cuboid(3)
print('Volumes is', c.volume())
```



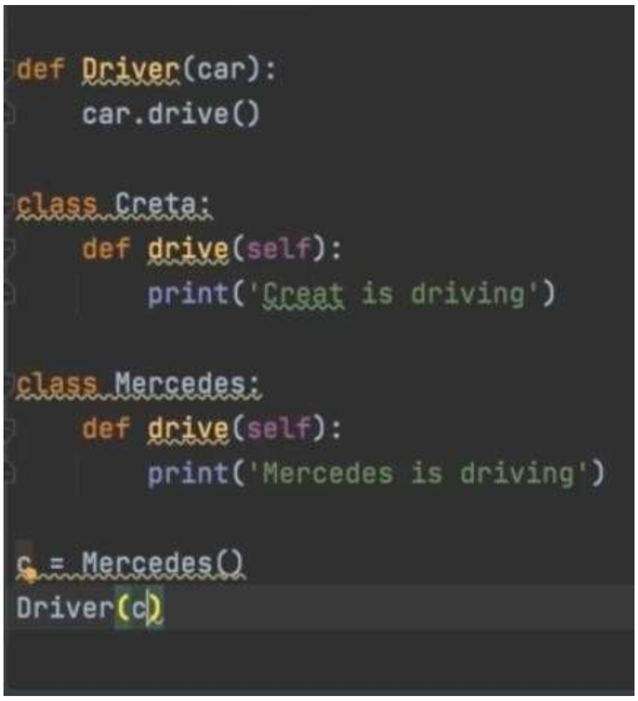
```
Volumes is 150
```

- Our parent class is rectangle, which is having constructor and two methods
  i.e. to find area and perimeter of a rectangle.
- we are creating another class (child class) called cuboid which is inheriting all properties of a rectangle class but the cuboid class is also have its own constructor, we are unable to use rectangle method because
- When a child class is having its own construct parent class construct cannot be called in child class
- To overcome the constructor of child class we should use super( ) method , so, that we can access parent class method in child class
- Therefore, this process is called constructor overriding in Inheritance.

Polymorphism

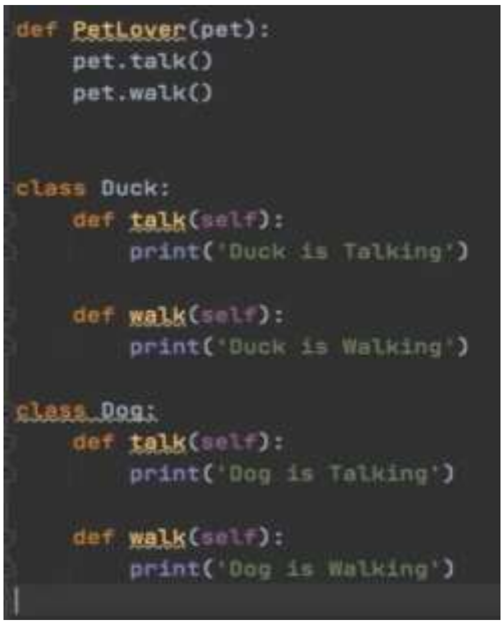
Polymorphism: many functions

Polymorphism means One name different action.



```
def Driver(car):
    car.drive()

class Creta:
    def drive(self):
        print('Creat is driving')

class Mercedes:
    def drive(self):
        print('Mercedes is driving')

c = Mercedes()
Driver(c)
```

Driver is a function who is driving the car and running cars. So, it is the example of polymorphism a driver can use or run multiple cars means different actions and forms but same function.

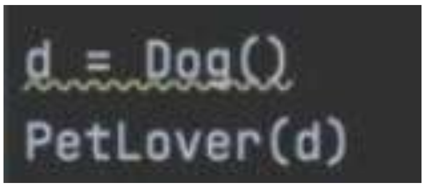


```
def PetLover(pet):
    pet.talk()
    pet.walk()


class Duck:
    def talk(self):
        print('Duck is Talking')

    def walk(self):
        print('Duck is Walking')

class Dog:
    def talk(self):
        print('Dog is Talking')

    def walk(self):
        print('Dog is Walking')
```

# taking method as pet lover and passing parameter as pet

#defining a class duck

#defining class dog



```
d = Dog()
PetLover(d)
```