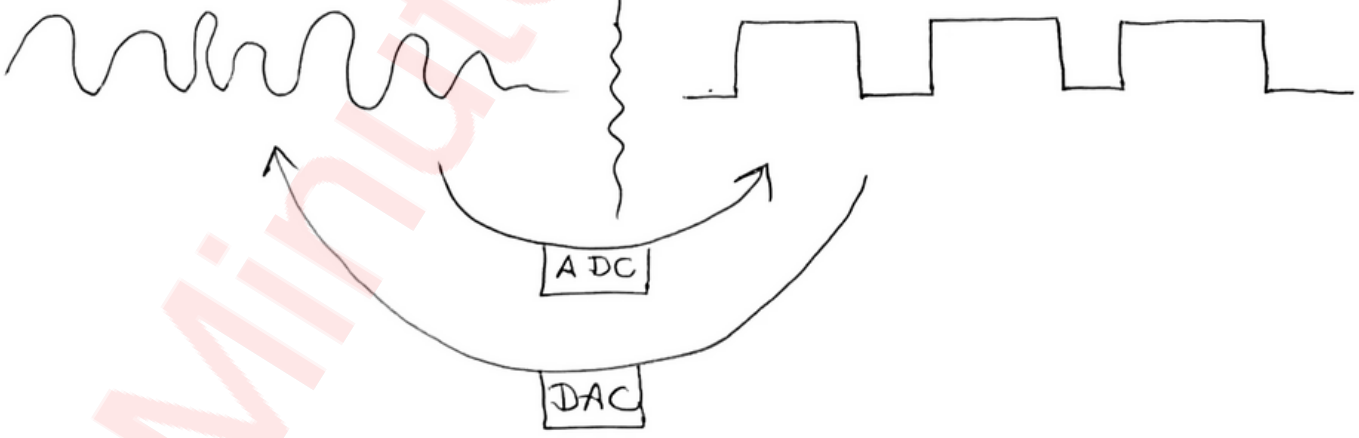
Introduction

Digital **Electronics**

Signal/ Discrete value — Branch

→ Analog Vs Digital

- Continuous Nature
- Discrete Nature [0 & 1]
- Sine wave
- Square wave
- Analog circuits [Resistors, capacitors, inductors]
- Digital circuits [Transistors, logic Gates, ICs]



- Number System → count
  → Representation

→ Binary (2), Octal (8), Decimal (10), Hexadecimal (16)

Base
[No. of symbols]

⇒ MSB & LSB

eg 0 1 1 0

Bit Bit Bit Bit Bit Bit Bit Bit

Byte

Binary	Octal	Decimal
0 0 0	0	0
0 0 1	1	1
0 1 0	2	2
0 1 1	3	3
1 0 0	4	4
1 0 1	5	5
1 1 0	6	6
1 1 1	7	7

Hexadecimal

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

A, B, C, D, E, F

↓ ↓ ↓ ↓ ↓ ↓

(10) (11) (12) (13) (14) (15) Total = (16)

Binary Arithmetic

↳ addition, substraction, division, multiplication.

⇒ Addition:

0 + 0 = 0

0 + 1 = 1

1 + 0 = 1

1 + 1 = 0 [Carry = 1]

⇒ Substraction:

0 - 0 = 0

0 - 1 = 1 [Borrow = 1]

1 - 0 = 1

1 - 1 = 0

⇒ Multiplication

0 × 0 = 0 × 1 = 1 × 0 = 0

1 × 1 = 1

- Division

$0 \div 0 = ND$

$0 \div 1 = 0$

$1 \div 0 = ND$

$1 \div 1 = 1$

- 1's Complement

eg: 1010101 $\Rightarrow$ 0101010

- 2's complement = 1 + 1's complement

1010101

↓ 1's complement

0101010

+ 1

0101011 ← 2's complement.

Shortcut: from LSB complement bits as we see first '1'

$\Rightarrow$ 1010101

↓↓↓↓↓↓↓ ← first '1'

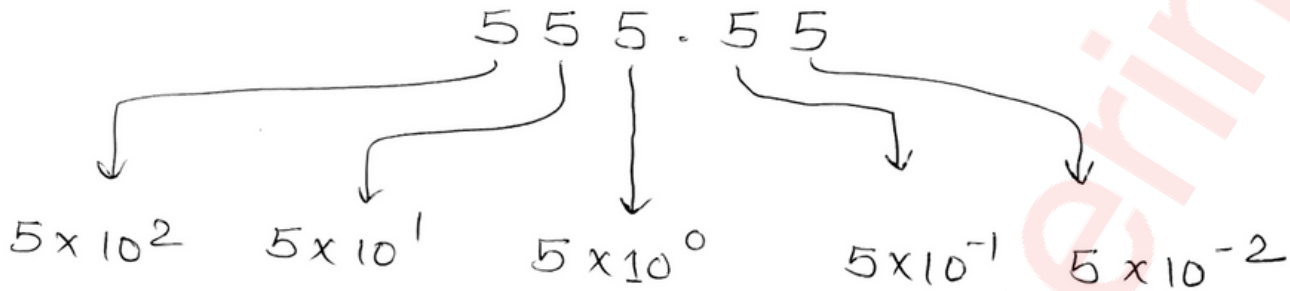
0101011

$\Rightarrow$ 1010000

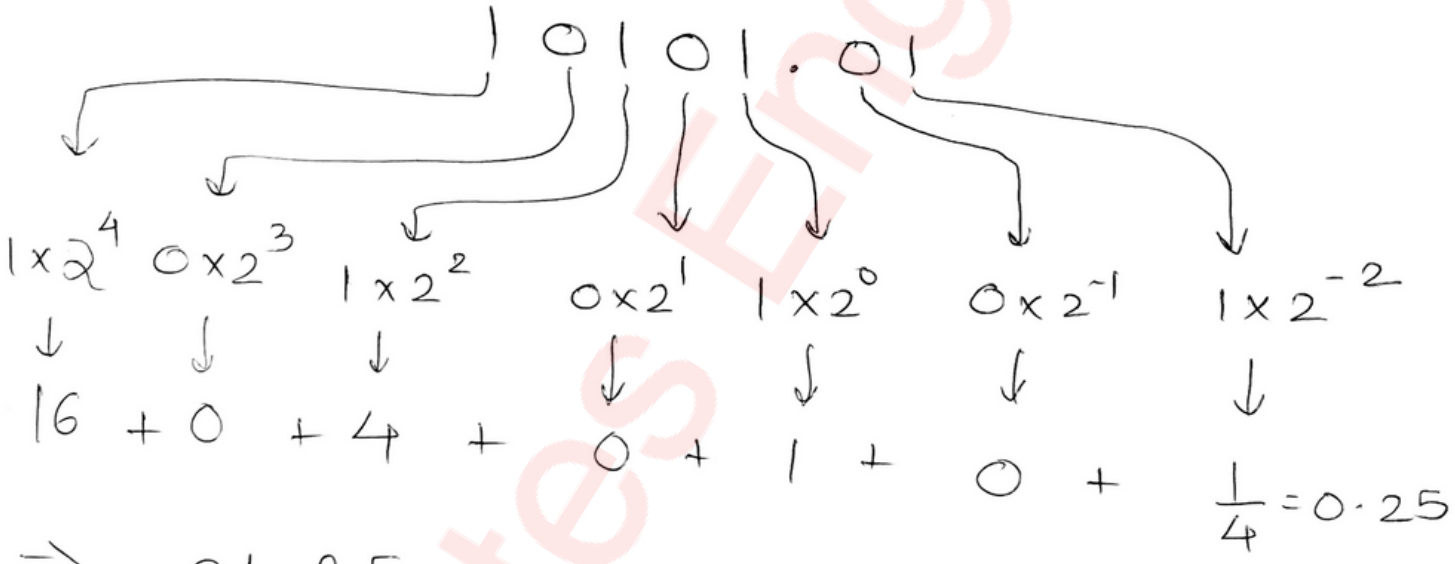
↓↓↓ ← first '1'

$\Rightarrow$ 011

Eg:- Decimal Number System

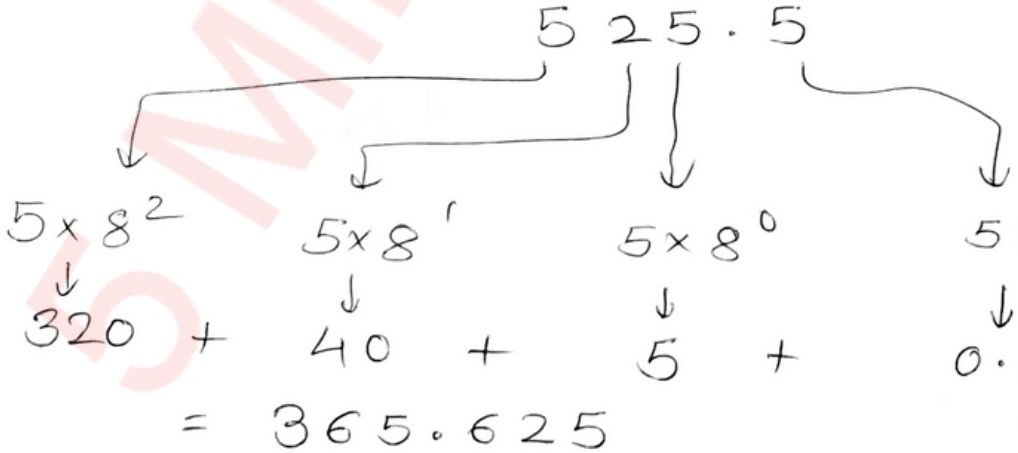


- Binary Number System



$\Rightarrow$ 21.25

- Octal Number System



= 365.625

$(1\ 0\ 1\ 0\ 1)_2$

$1\times2^4 \quad 0\times2^3 \quad 1\times2^2 \quad 0\times2^1 \quad 1\times2^0$

$16 + 0 + 4 + 0 + 1$

$(21)_{10}$

Convert to Octal.
[Divide by 8]

$$8\overline{)21}\quad 2,\ 16,\ 5$$

$(25)_8$

$= 2\times8^1 + 5\times8^0$

$= 16 + 5 = (21)_{10}$ ✓

$\Rightarrow (555.44)_{10}$

Convert to Octal.

$$8\overline{)555}\quad 69,\ 48,\ 75,\ 72,\ 3$$

$$8\overline{)69}\quad 8,\ 64,\ 5$$

$\Rightarrow 853.341$

$0.44\times8 = 3.52$

$0.52\times8 = 4.16$

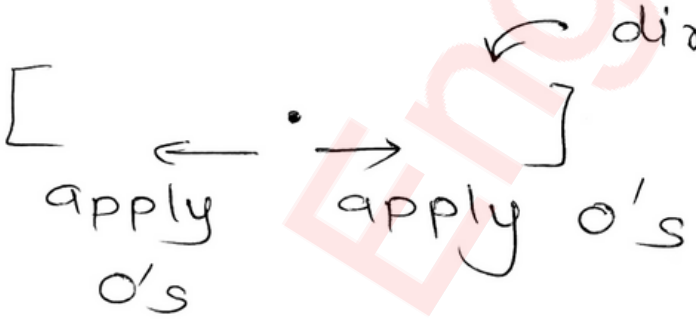
$0.16\times8 = 1.28$

- $(101010.10)_2$

  → $(222.2)_4$

  → $(52.4)_8$

  → $(2A.8)_{16}$



But to go from $B_4 \rightarrow B_8$, $B_8 \rightarrow B_{16}$

⇒ Try to first get it in B_2 then next.

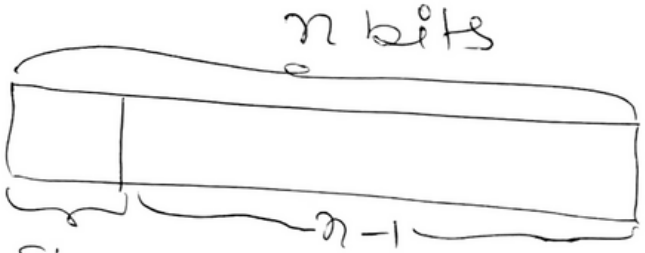
$B_4 \rightarrow B_2 \rightarrow B_8 / B_{16}$

Signed Numbers

↓

Magnitude + Sign

(+5, -5)



0 → +ve
1 → -ve

$[-(2^{n-1}-1)$ to $+(2^{n-1}-1)]$ (SM)

↓

Unsigned Numbers

↓

Only Magnitude

No sign (5, 55)

⇒ 5 bit

↳ $2^n - 1$

$= 2^5 - 1$

$= 31$ total/possible.

Decimal values	2's C	Signed Magnitude (MSB ↓)
+5	0 1 0 1	0 1 0 1
+3	0 0 1 1	0 0 1 1
+1	0 0 0 1	0 0 0 1
+0	0 0 0 0	0 0 0 0
-0	0 0 0 0	1 0 0 0
-1	1 1 1 1	1 0 0 1
-3	1 1 0 1	1 0 1 1
-5	1 0 1 1	1 1 0 1

Same

⇒ Eg Given 1 0 1 1 0

SB | mag.

1 → -ve
0 → +ve

1×2^4 0×2^3 1×2^2 1×2^1 0×2^0

$(-16) + 0 + 4 + 2 + 0$

$-16 + 6 = -10$

$(2^{1}s\,C)$ 1 0 1 1 0

(-ve) (10)

⇒ Substraction by Addition (2's C)

eg: 10 − 5

↓

10 + (−5)

0 1 0 1 0 + 1 1 0 1 1

(+ve) 10 −ve 5

↓

−16 + 8 + 0 + 2 + 1

= −5

0 1 0 1

↓ 2's C

1 0 1 1

1 1 1 1

1 0 1 0 1 0

\+ 1 1 0 1 1

0 0 1 0 1

↓ +ve 5

⇒ 10 − 11

↓

10 + (−11)

↓ ↓ 2's com

01010 1 0101

+ve 10 −ve

1011

0101

01010

10101

11111 → −ve no.

1×2^5 1×2^4 1×2^3 1×2^2 1×2^1 1×2^0

↓ ↓ ↓ ↓ ↓ ↓

$-32+16+8+4+2+1$

$-32+31 = -1$ Ans.

⇒ BCD code

→ 8421

→ 2421

} weighted code.

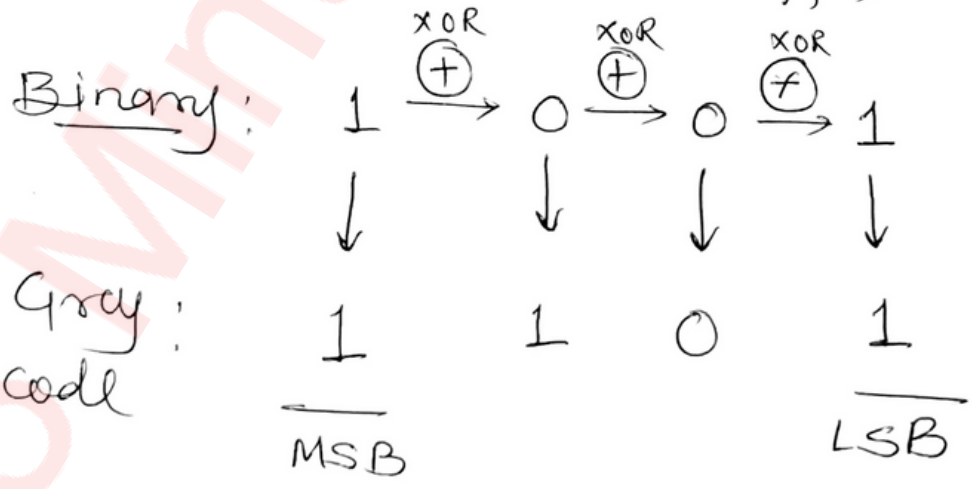
Decimal	8421	2421
0	0000	0000
1	0001	0001
2	0010	0010
3	0011	0011
4	0100	0100
5	0101	1011
6	0110	1100
7	0111	1101
8	1000	1110

- Non weighted codes
  - Excess-3
  - Gray code

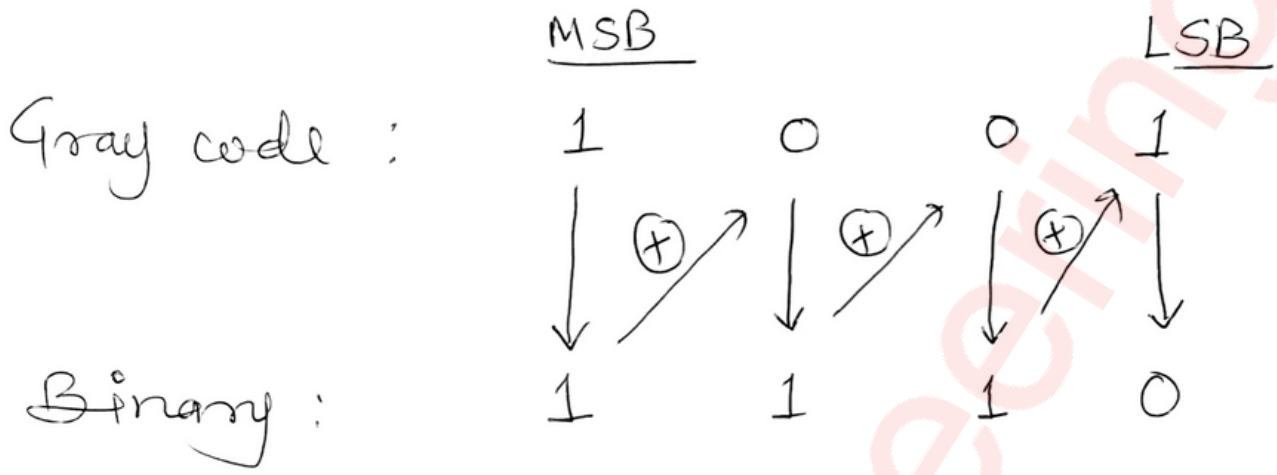
Decimal	BCD 8421	Excess-3 (+0011)
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

Complement

- Gray Code : (A bit diff)



- Gray code → Binary



- Boolean Algebra:
  - ↓ Branch of Algebra
  - → True/false
  - → 1/0

: Laws

① Idempotent law:

$$x \cdot x = x \quad \& \quad x + x = x$$

② Associative law

$$x \cdot (y \cdot z) = (x \cdot y) \cdot z$$
$$x + (y + z) = (x + y) + z$$

③ Distributive law

$$x \cdot (y + z) = (x \cdot y) + (x \cdot z)$$
$$x + (y \cdot z) = (x + y) + (x + z)$$

④ Commutative law

$$x \cdot y = y \cdot x$$
$$x + y = y + x$$

⑤ De-Morgan law

$$\overline{(x + y)} = \overline{x} \cdot \overline{y}$$
$$\overline{(x \cdot y)} = \overline{x} + \overline{y}$$

⑥ Identity law

$$x + 0 = x$$
$$x \cdot 0 = 0$$
$$x + 1 = 1$$
$$x \cdot 1 = x$$

⑦ Complementation law

$$\overline{T} = F \qquad \overline{0} = 1$$
$$\overline{F} = T \qquad \overline{1} = 0$$

$$\boxed{x \cdot \overline{x} = 0} \qquad \boxed{x + \overline{x} = 1}$$

⑧ Involution law

$$\overline{(\overline{x})} = x$$

Logic Gates:

↳ 1 or more I/p

But

only 1 O/p.

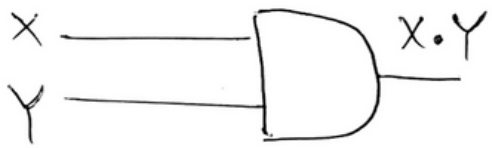
Basic Gates: AND, OR & NOT

Universal Gates: NAND, NOR

Special Gates: XOR, xNOR

- AND Gate [O/p is T/High if all I/p are True]

X	Y	X.Y
0	0	0
0	1	0
1	0	0
1	1	1

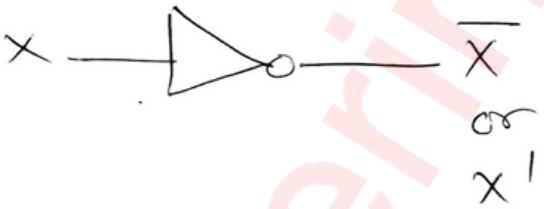


- OR Gate [O/p is 'T' if any one I/p is 'T']

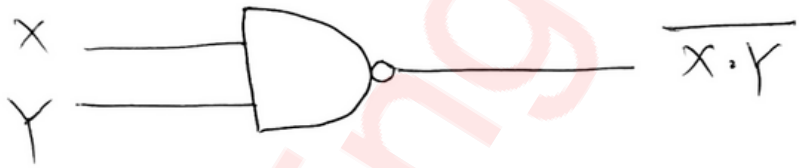
X	Y	X+Y
0	0	0
0	1	1
1	0	1
1	1	1

- **NOT Gate** [0 → 1 & 1 → 0]

X	$\overline{X}$ or X'
0	1
1	0

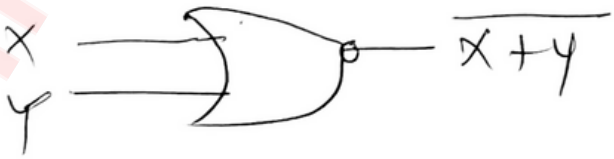


- **NAND Gate** [Complement of 'AND']



X	Y	$\overline{X \cdot Y}$
0	0	1
0	1	1
1	0	1
1	1	0

- **NOR Gate** [Complement of 'OR']



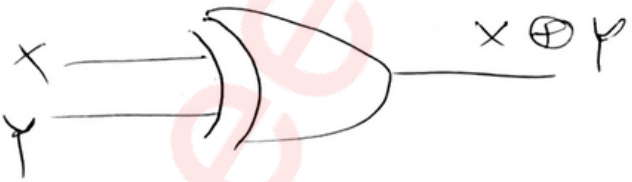
X	Y	$\overline{X+Y}$
0	0	1
0	1	0
1	0	0
1	1	0

° XOR Gate [O/p is True if Different]

" 0 1 "

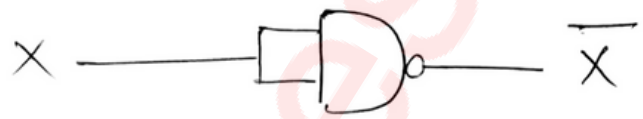
" 1 0 "

X	Y	X ⊕ Y
0	0	0
0	1	1
1	0	1
1	1	0

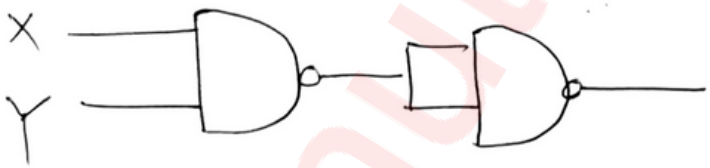


$$X \oplus Y = X.\overline{Y} + \overline{X}.Y$$

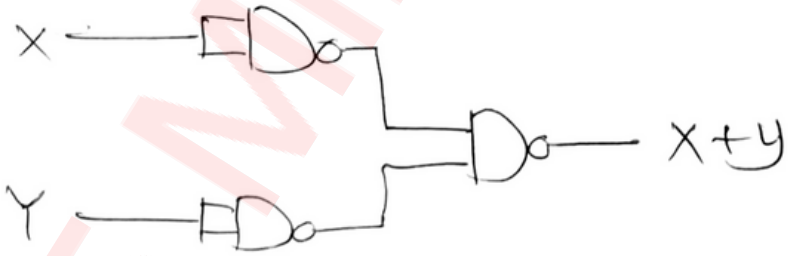
⇒ Realization of NOT, OR, AND & XOR using NAND gate.

⇒ 

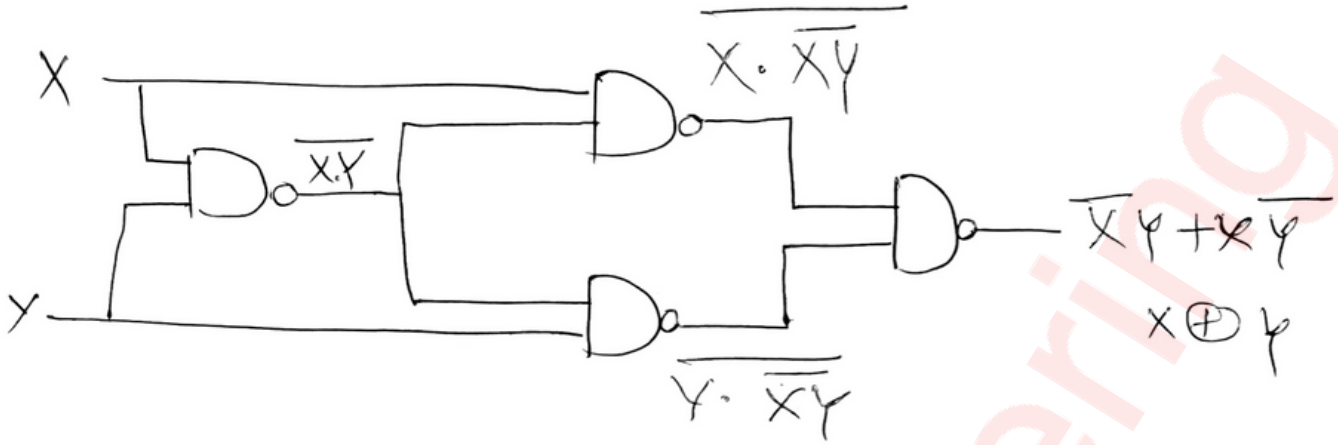
(NOT)



(AND)

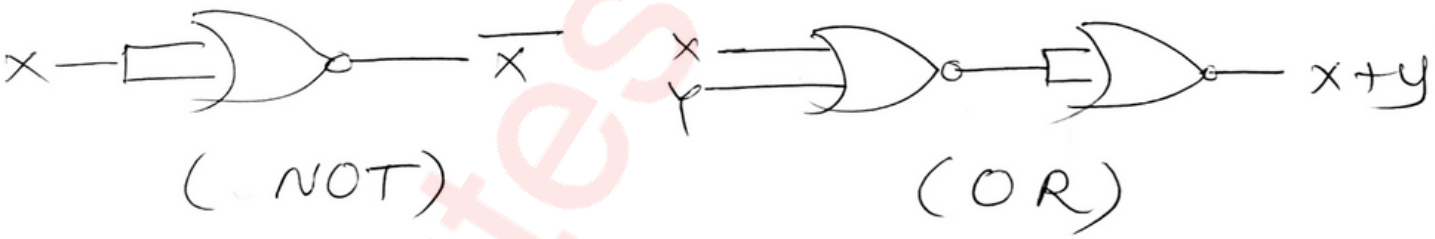


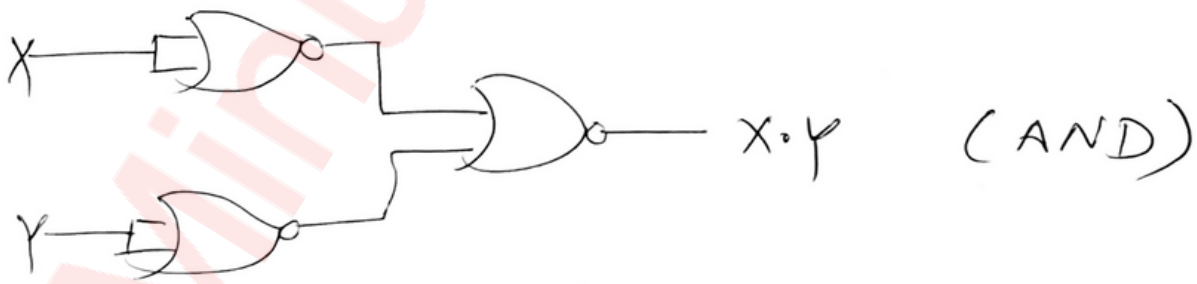
(OR)

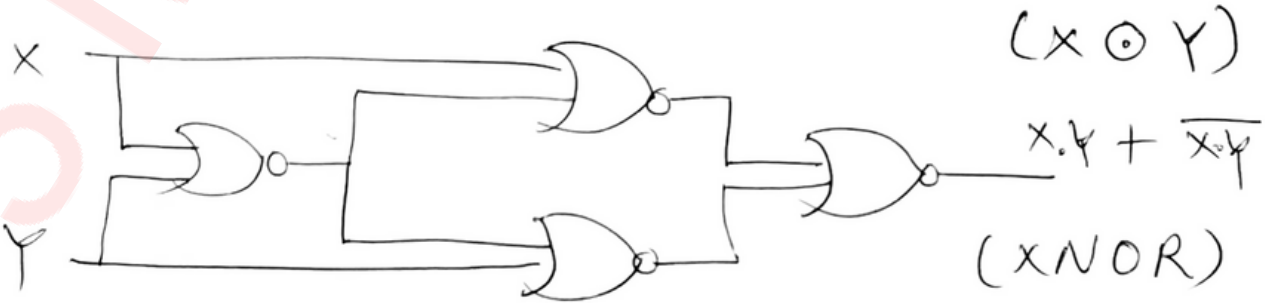


X	Y	X.Y	$\overline{X.Y}$	$\overline{X.\overline{XY}}$	$\overline{Y.\overline{XY}}$	$(X \oplus Y)$ O/P
0	0	0	1	1	1	0
0	1	0	1	1	0	1
1	0	0	1	0	1	1
1	1	1	0	1	1	0

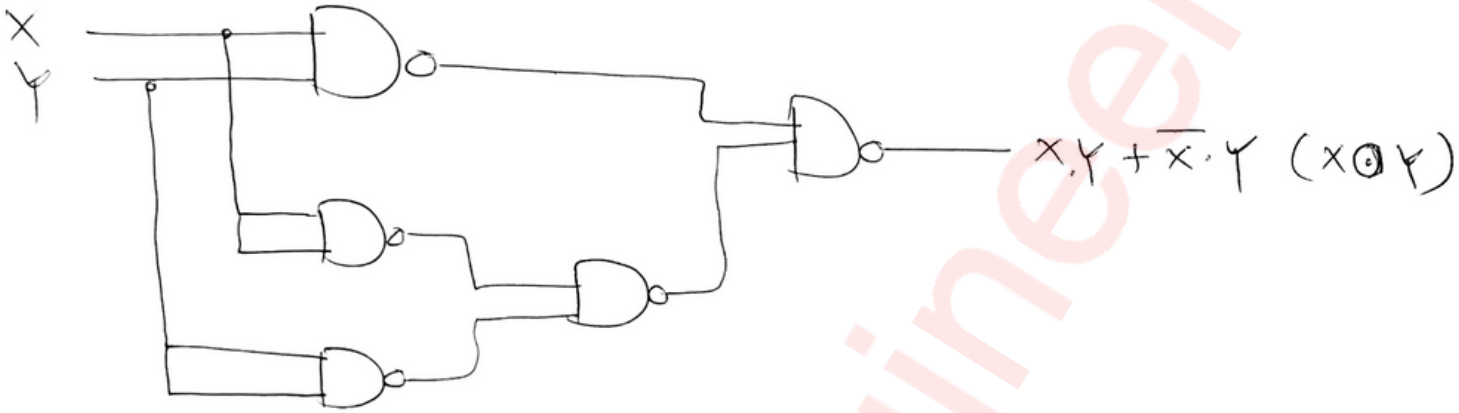
Similarly using NOR Gate.



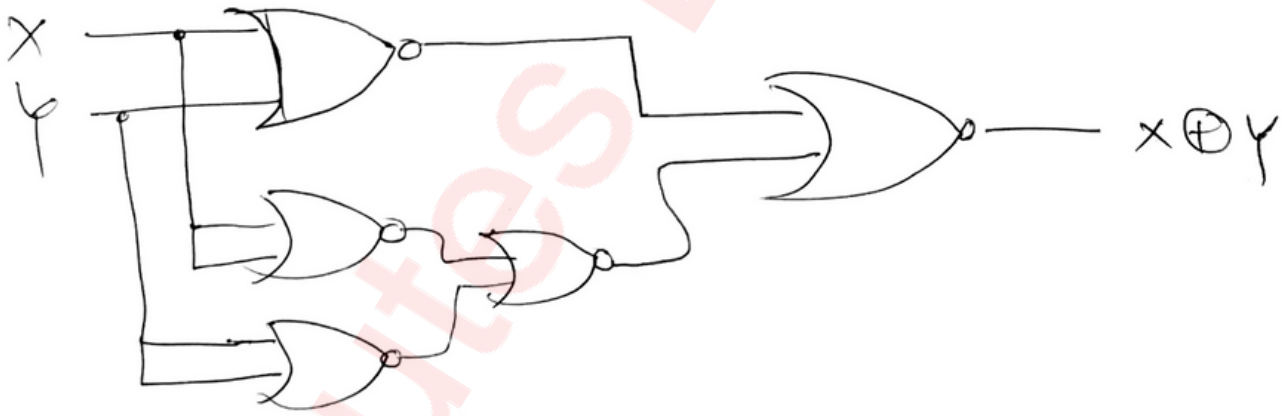




(XNOR using NAND)



(XOR using NOR)



Boolean Expression

↳ Expression that can tell when o/p → 0 or 1

→ SOP [Sum of Product] "1"

→ POS [Product of sum] "0"

X	Y	O/P
0	0	1
0	1	1
1	0	0
1	1	0

$O(X,Y) = \sum_m (0, 1)$ (m_0 m_1)

$O(X,Y) = \prod_M (2, 3)$ (M_2 M_3)

Eg:

X	Y	O/P = X+Y (POS)
0	0	0
0	1	1
1	0	1
1	1	1

$SOP = XY + \bar{X}y + x\bar{y}$

↑ Sum of Product

- Sum of Product

→ Expression have:

Product terms (AND) summed (OR)

Eg: $XY + YZ + \bar{X}Y\bar{Z}$

↑ 2 literals ↑ 3 literals

[complement or Non-complement allowed ✓]

⇒ minterm = $[XY, YZ, \bar{X}Y\bar{Z}]$

$f(x,y,z)$ ✗ ✗ ✓

Binary

0	0	0	$\bar{a}\bar{b}\bar{c} \rightarrow m_0$
0	0	1	$\bar{a}\bar{b}c \rightarrow m_1$
0	1	0	$\bar{a}b\bar{c} \rightarrow m_2$
0	1	1	$\bar{a}bc \rightarrow m_3$
1	0	0	$a\bar{b}\bar{c} \rightarrow m_4$
1	0	1	$a\bar{b}c \rightarrow m_5$
1	1	0	$ab\bar{c} \rightarrow m_6$
1	1	1	$abc \rightarrow m_7$

minterm

- Canonical logic forms

↳ CSOP: $xyz + x\bar{y}z + \bar{x}\bar{y}z$ (✓)

$xy + \bar{x}yz$ (x)

$y + x + z$ (x)

$xy(\bar{z}+z) + \bar{x}y\bar{z}$

↓ ↓

$[xy\bar{z} + xyz + \bar{x}\bar{y}\bar{z}]$ ✓ CSOP

- POS [Product of Sum]

⇒ OR terms → product

$(x+y)(y+z)$

$(x+y)$, $(y+z)$ — OR

$(x+y)\cdot(y+z)$ — AND

⇒ Maxterm = $[\bar{x}, \bar{z}, \bar{x}\bar{z}]$

$f(x,z)$ (x) (x) (✓)

$0 \rightarrow x$

$1 \rightarrow \bar{x}$

(POS)

$0 \rightarrow \bar{x}$

$1 \rightarrow x$

(SOP)

- CPOS :- $(\bar{x}+y+z)\cdot(x+\bar{y}+\bar{z})$ (✓)

$(\bar{x}+\bar{y})\cdot(x+\bar{y}+\bar{z})$ (x)

$(\bar{x})\cdot(\bar{y})\cdot(y+\bar{z})$ (x)

$(\bar{x}+\bar{y}+\bar{z}\cdot z)\cdot(x+\bar{y}+\bar{z})$

$(\bar{x}+\bar{y}+\bar{z})(\bar{x}+\bar{y}+z)\cdot(x+\bar{y}+\bar{z})$

(✓)

Q → Given $n = 3$ ↑ (x, y, z) Boolean variables

① Total combinations $= 2^n$

i.e $n = 2\ (x, y)$

00, 01, 10, 11

$n = 3\ (x, y, z)$

000 --- 111

② Total Boolean fns $= 2^{2^n}$

$n = 2\ (x, y) \rightarrow 2^{2^2} = 16$

$n = 3\ (x, y, z) \rightarrow 2^{2^3} = 2^8 = 256$

Q → Complement of a fn.

$\Rightarrow f(x, y, z, 0, 1, \cdot, +)$

$\downarrow$

$f'(\bar{x}, \bar{y}, \bar{z}, 1, 0, +, \cdot)$

eg:- $\bar{x} + y + z \xrightarrow{C} x \cdot \bar{y} \cdot \bar{z}$

eg: $xy + \bar{x}\,\bar{y} \leftarrow$ [EXNOR]

$\downarrow$

$(\bar{x} + \bar{y}) \cdot (x + y)$

$(\bar{x} \cdot x) + (\bar{x} \cdot y) + (\bar{y}\, x) + (\bar{y} \cdot y)$

$\downarrow \qquad\qquad \downarrow$

$(\bar{x} \cdot y) + (x \cdot \bar{y}) \leftarrow$ [EXOR]

- Duality Theorem

$\cdot \longleftrightarrow +$ (✓)

$\oplus \longleftrightarrow \odot$ (✓)

$0 \longleftrightarrow 1$ (✓)

$x \longleftrightarrow \bar{x}$ (x)

$\sim \longleftrightarrow \sim$ (✓)

eg: ① $XY\bar{Z} + \bar{X}YZ + XYZ$

$(X+Y+\bar{Z}) \cdot (\bar{X}+Y+Z) \cdot (X+Y+Z)$

② $XYZ + \bar{X}Y\bar{Z} = 1$

$(X+Y+Z) \cdot (\bar{X}+Y+\bar{Z}) = 0$

- Self dual fn:

→ If dual is equal to I/p fn.

eg: $(AB + BC + CA)$

$(A+B) \cdot (B+C) \cdot (C+A)$

$AB + AC + B + BC$

$B(A+1+C) + AC$

$(B(1) + AC)(C+A)$

$(BC + BA + AC + AC)$

$\Rightarrow AB + BC + AC$

Q → No. of self dual f^ns -

→ $n = 2$

① Total combinations $(0, 1) = 2^n$

② Total f^ns possible $= 2^{2^n}$

③ Total self dual f^ns possible $= \left[2^{2^{n-1}}\right]$

Eg:

X	Y
0	0
0	1
1	0
1	1

$2^n = 2^2 = 4$

② Total $f^n = 2^{2^2} = 2^4 = \underline{16}$

- 0
- 1
- ✓ A
- ✓ B
- ✓ $\bar{A}$
- ✓ $\bar{B}$
- A B
- $\bar{A}$ B
- A $\bar{B}$
- $\bar{A}$ $\bar{B}$

- A + B
- $\bar{A}$ + B
- A + $\bar{B}$
- $\bar{A}$ + $\bar{B}$
- AB + $\bar{A}\bar{B}$
- $\bar{A}$B + A$\bar{B}$

③ Total self dual $f^n = 2^{2^{1-1}}$

$= 2^2$

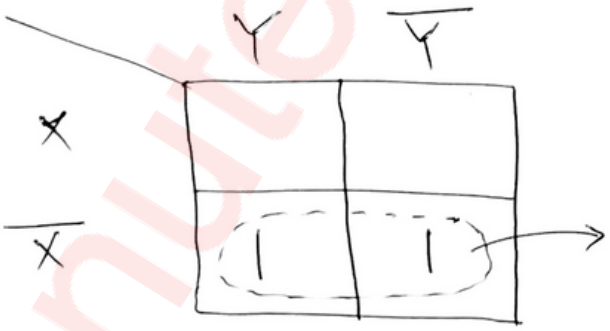
$\Rightarrow = 4$

- **Minimization** [Simplification]
  - → Boolean Laws
  - → Karnaugh map (kmap)

- **kmap** :- $f(X,Y) = \Sigma m(0,1)$

X	Y	f
0	0	1
0	1	1
1	0	0
1	1	0

$$\rightarrow \bar{X}\bar{Y} + \bar{X}Y$$
$$\Rightarrow \bar{X}(Y + \bar{Y})$$
$$\Rightarrow \bar{X}$$

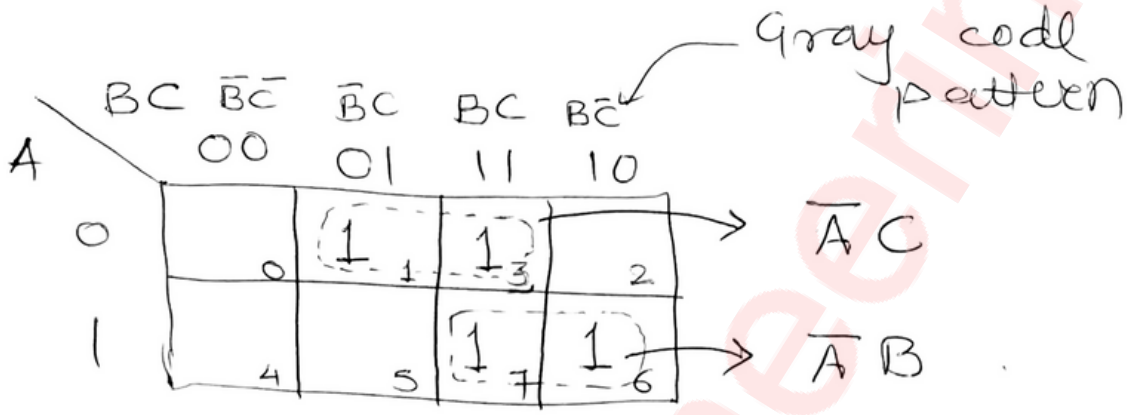


Y is changing $(Y, \bar{Y})$

But X is constant as $(\bar{X})$

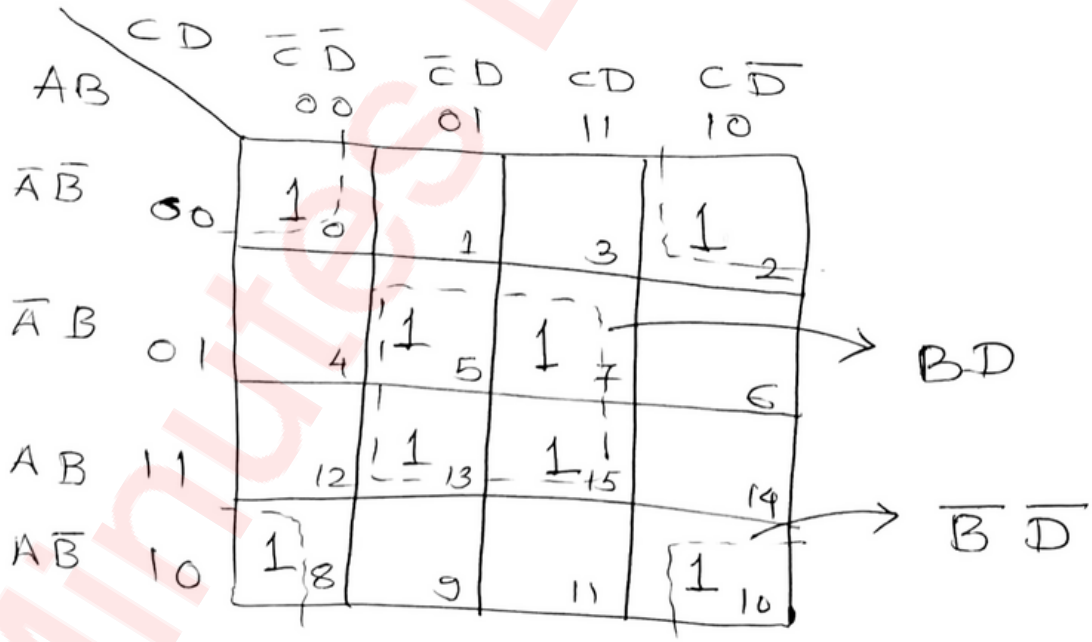
Eg: $n = 3$ $f(x, y, z)$

↳ $\Sigma m(1, 3, 6, 7)$



Eg: $n = 4$ $f(A, B, C, D)$

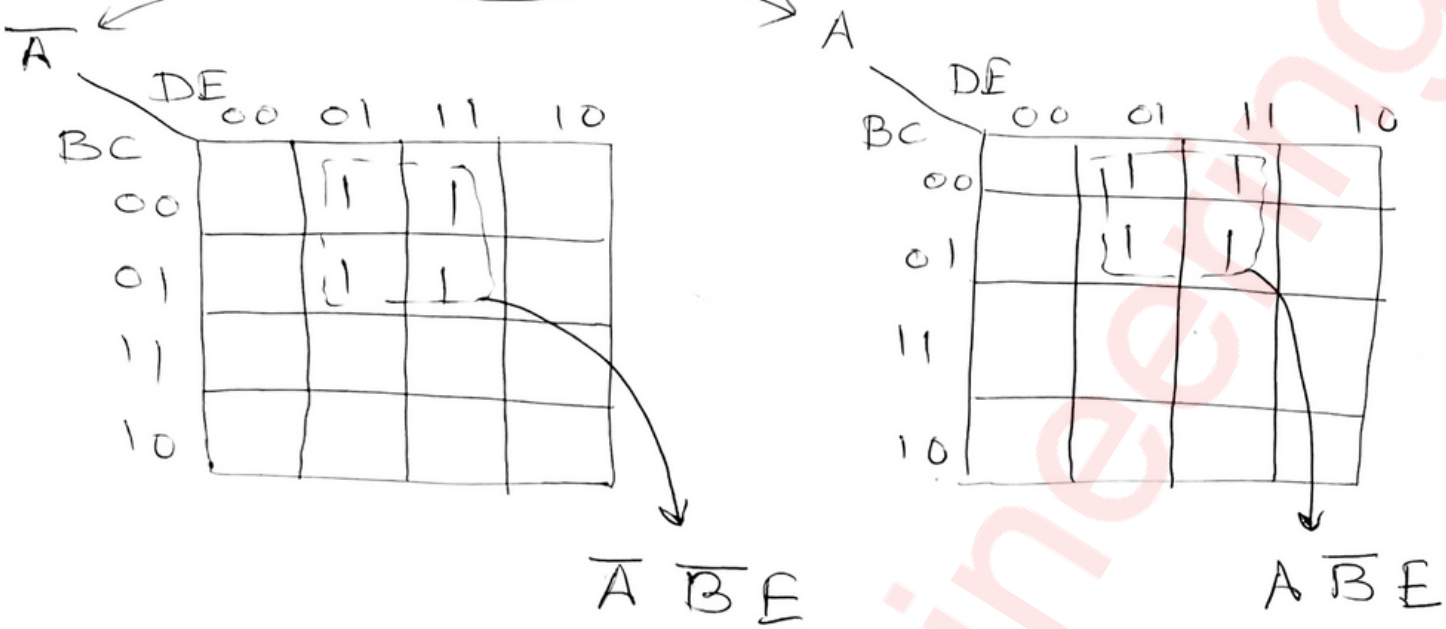
↳ $\Sigma m(0, 2, 5, 7, 8, 10, 13, 15)$

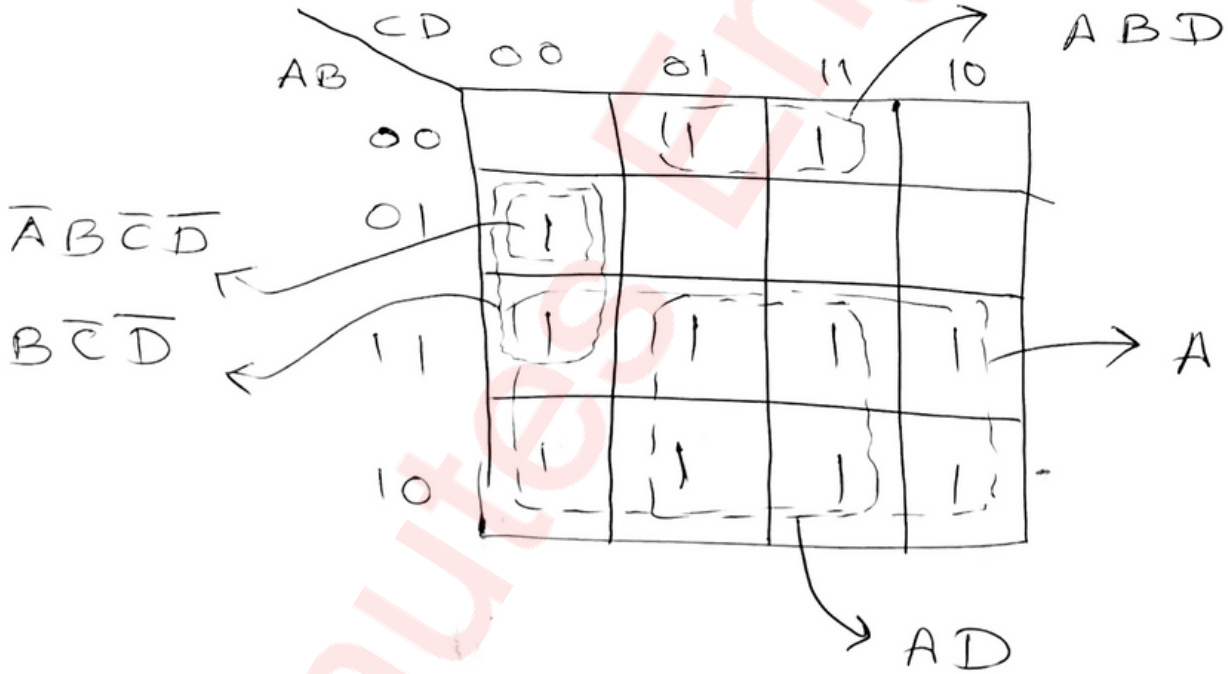


Don't care (x) → can be used to simplify

But we focus mostly on 1 | No separate group of (x) is formed.

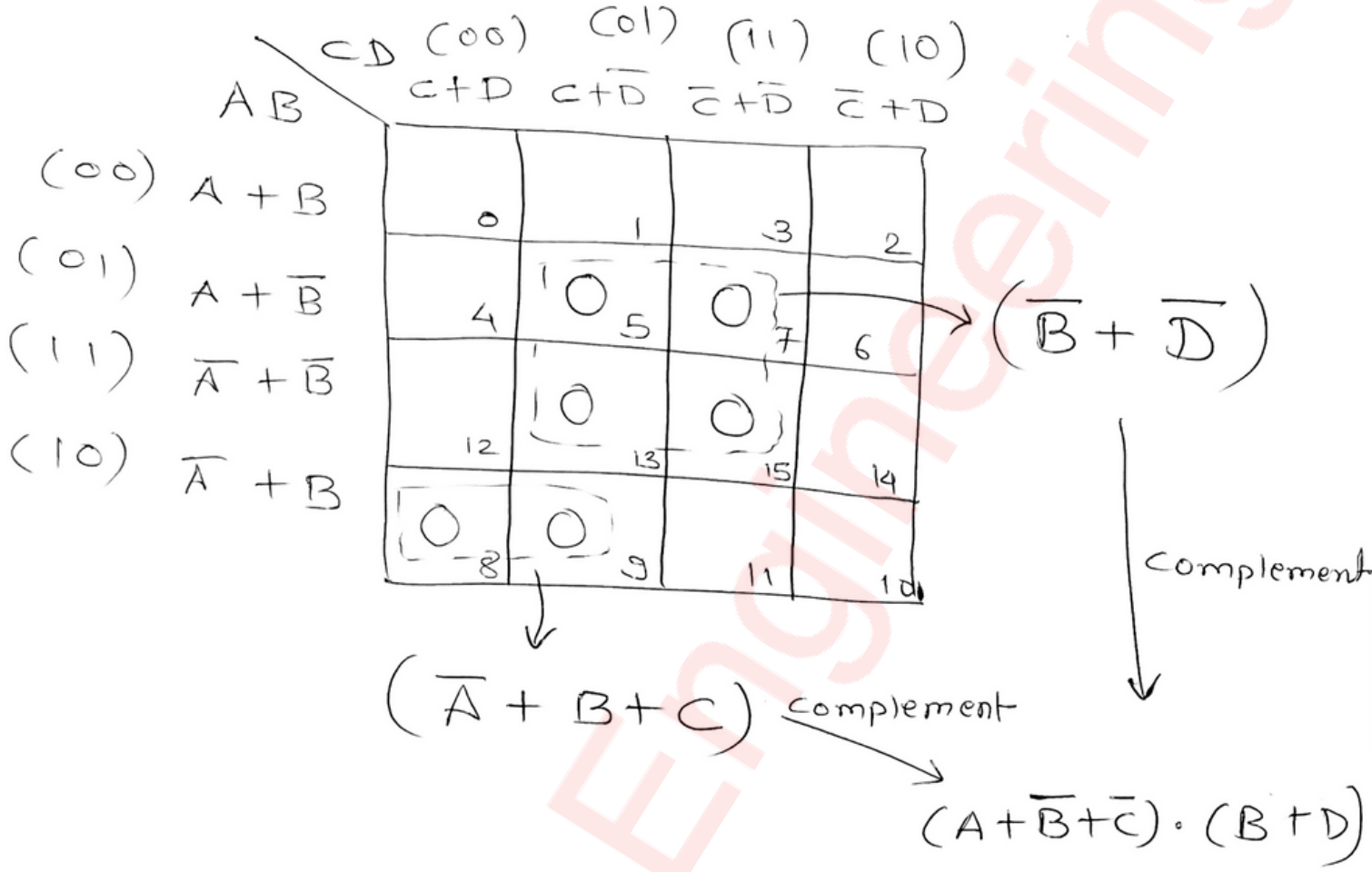
$n = 5$ (A, B, C, D, E)





Kmap for POS (A B C D)

n = 4



- Prime Implicants

(4) → BC̄D, ABD, ACD, ABC

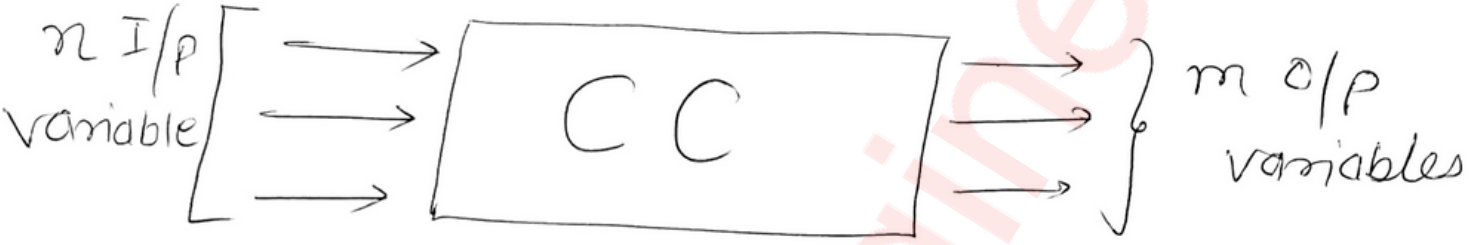
- Essential Prime Implicants

(3) → BC̄D, ACD, ABC

Σm(5, 13, 14, 15)

Combinational Circuit

→ (AND, OR, NOT, NAND, NOR)
→ (I/p & o/p variables)
→ Current combination of I/p
→ NO memory unit



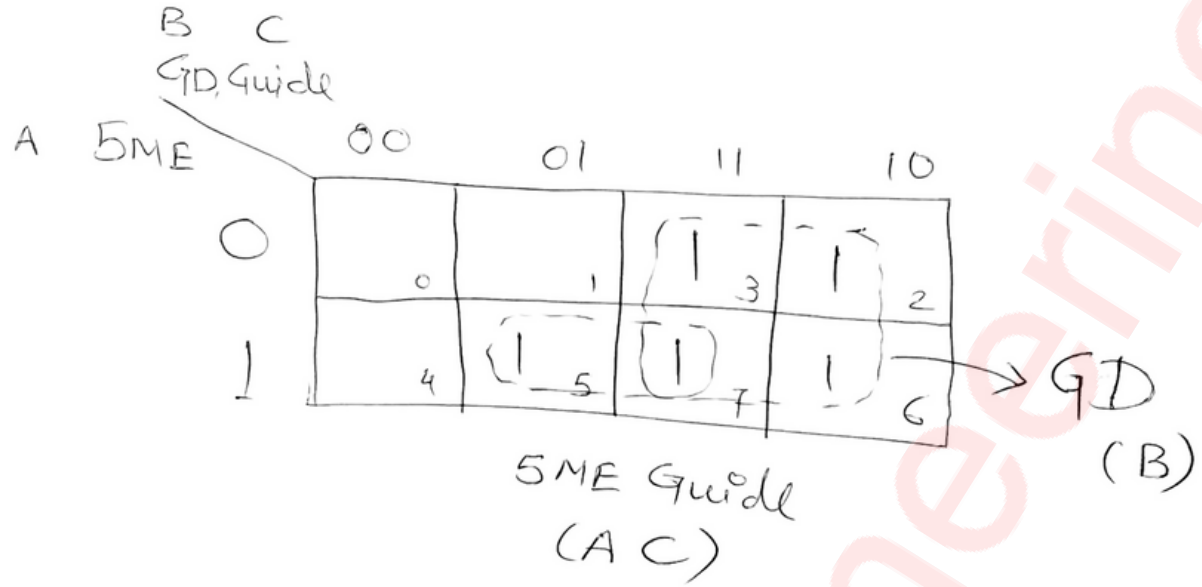
$$o/p = F(I/p)$$

Eg:- I/p: 5ME & Guide & GD
o/p: Result

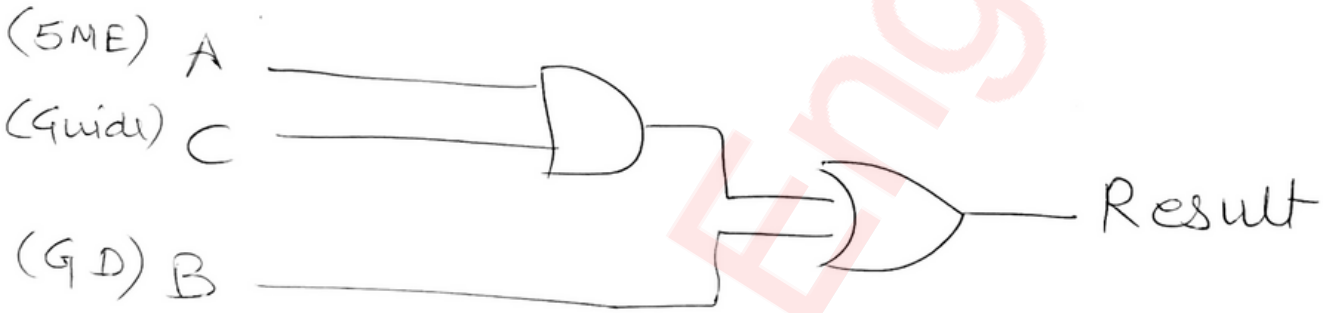
① Truth table

GD	5ME	Guide	Result
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

(2) Use k-map

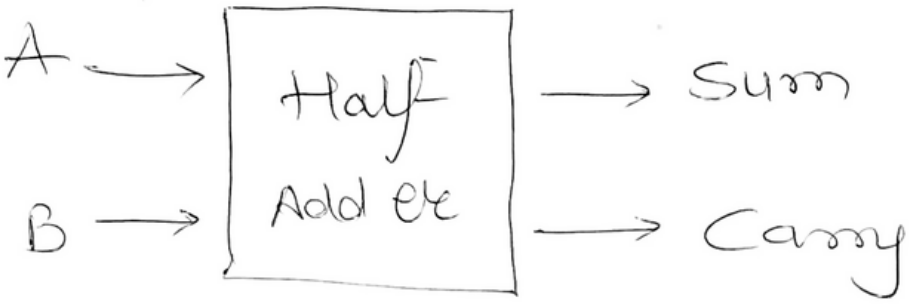


$R \Rightarrow AC + B$



- Adder : CC to perform addition of nos.
  - → Half adder (works on 2 bits addition)
  - → full adder (works on 3 bits additions)

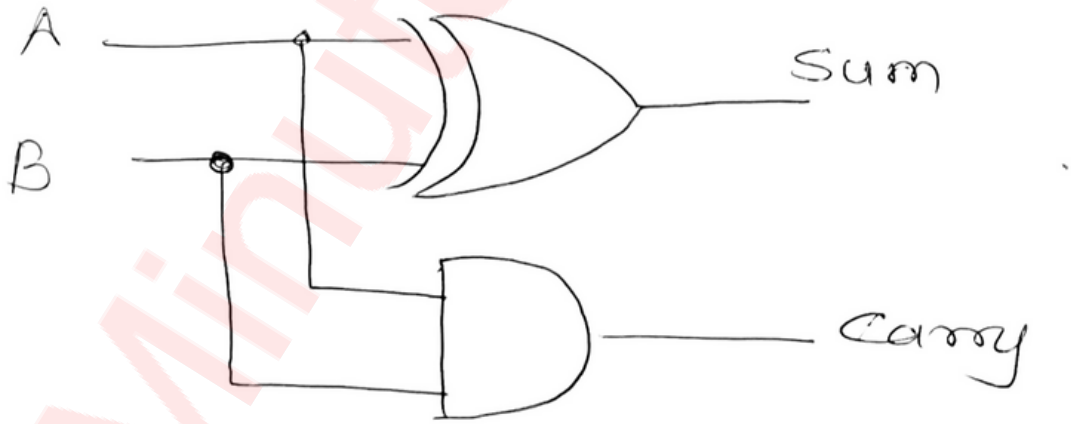
```
  0        0        1        1
+ 0      + 1      + 0      + 1
-----    -----    -----    -----
  0 0      0 1      0 1      1 0
  ↑ ↑      ↑ ↑      ↑ ↑      ↑ ↑
  C R      C R      C R      C R
```



A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

XOR

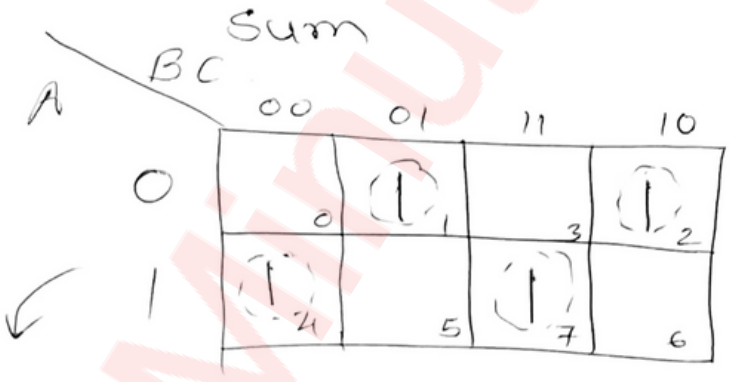
AND



∘full adder

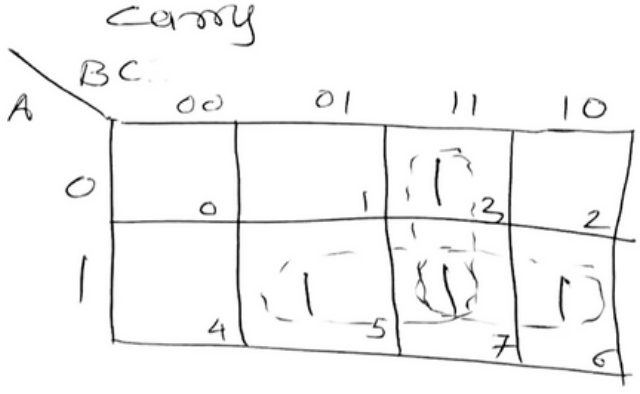
A	B	CI	Sum	CO
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Complement



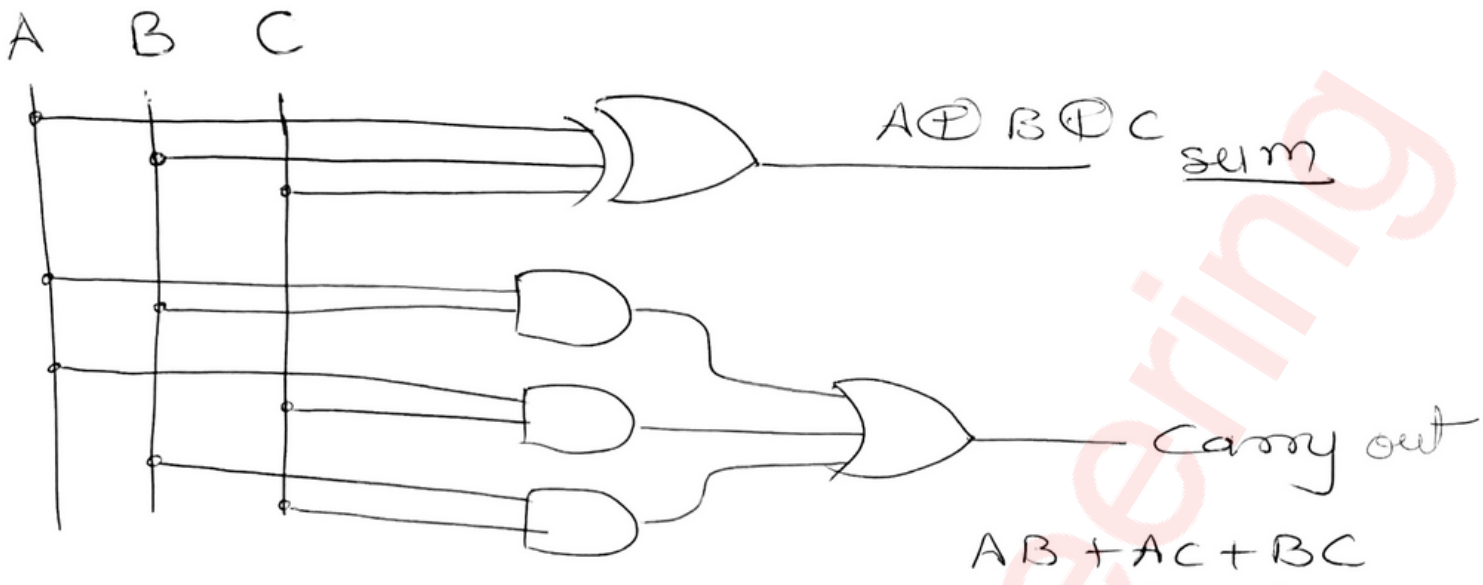
$A\bar{B}\bar{C} + \bar{A}\bar{B}C + ABC + \bar{A}B\bar{C}$

$Sum = A \oplus B \oplus C$

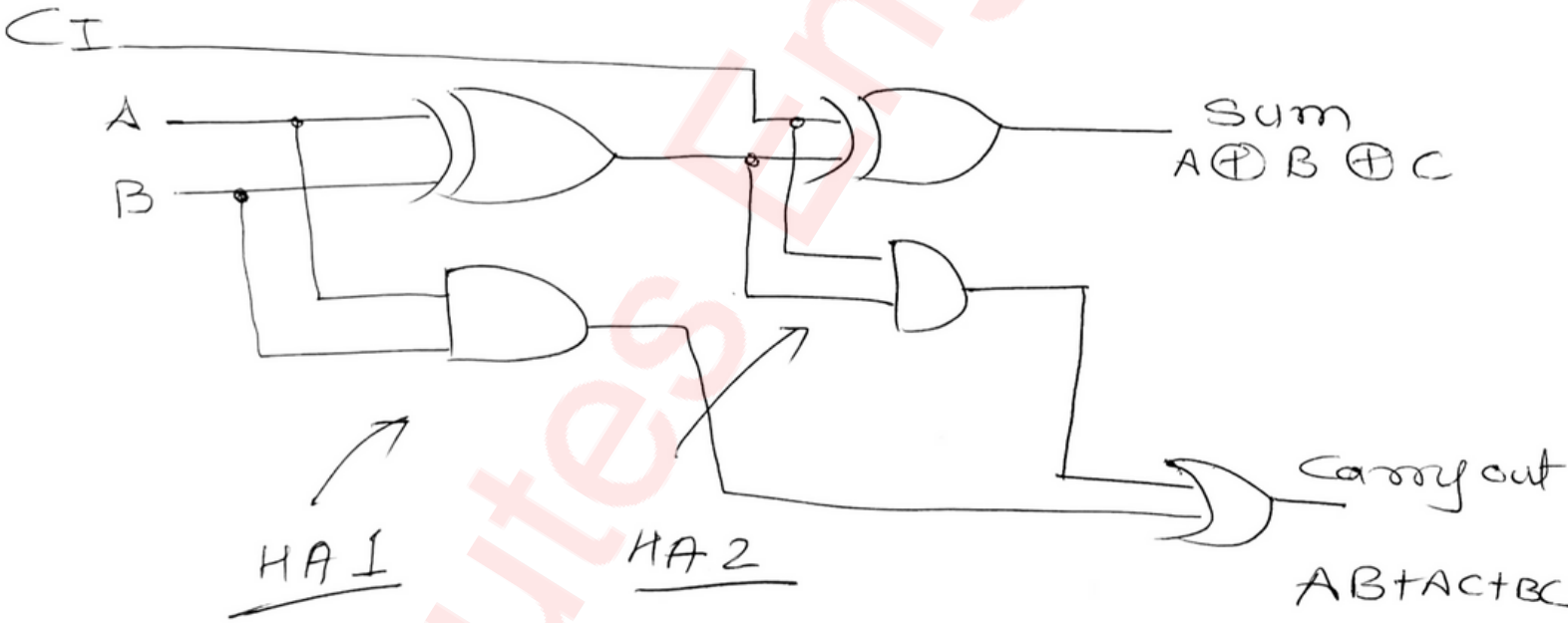


$AC + AB + BC$

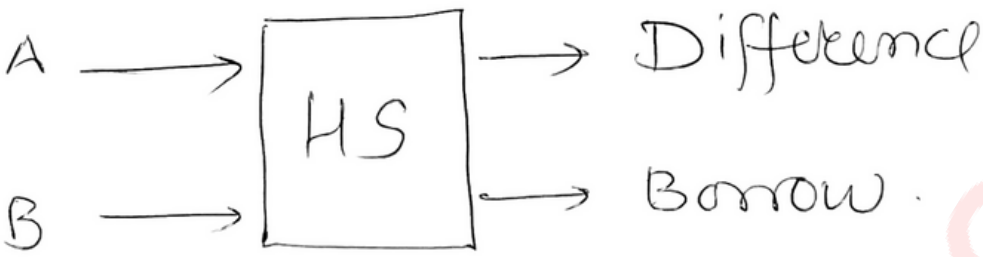
carry =



- NOW implementing the behaviour of Full adder with 2 half adder



Half Subtractor

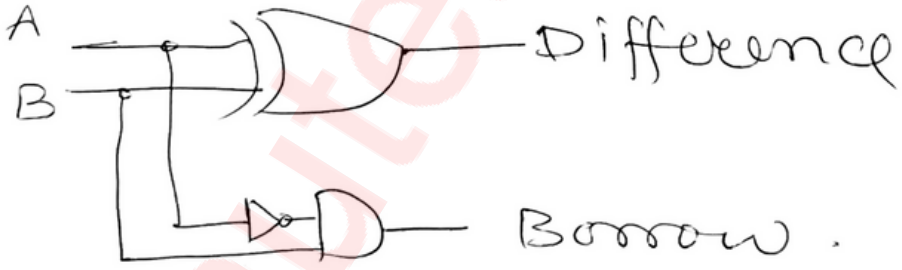


A	B	D	BO
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

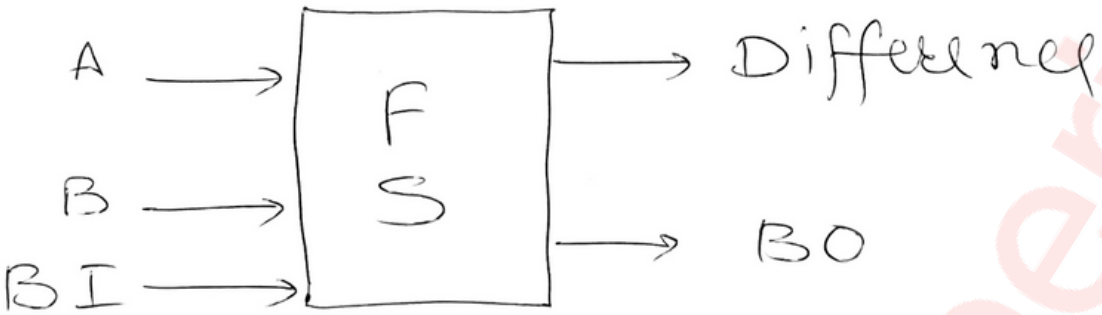
XOR

$\overline{A}$ B

0 1

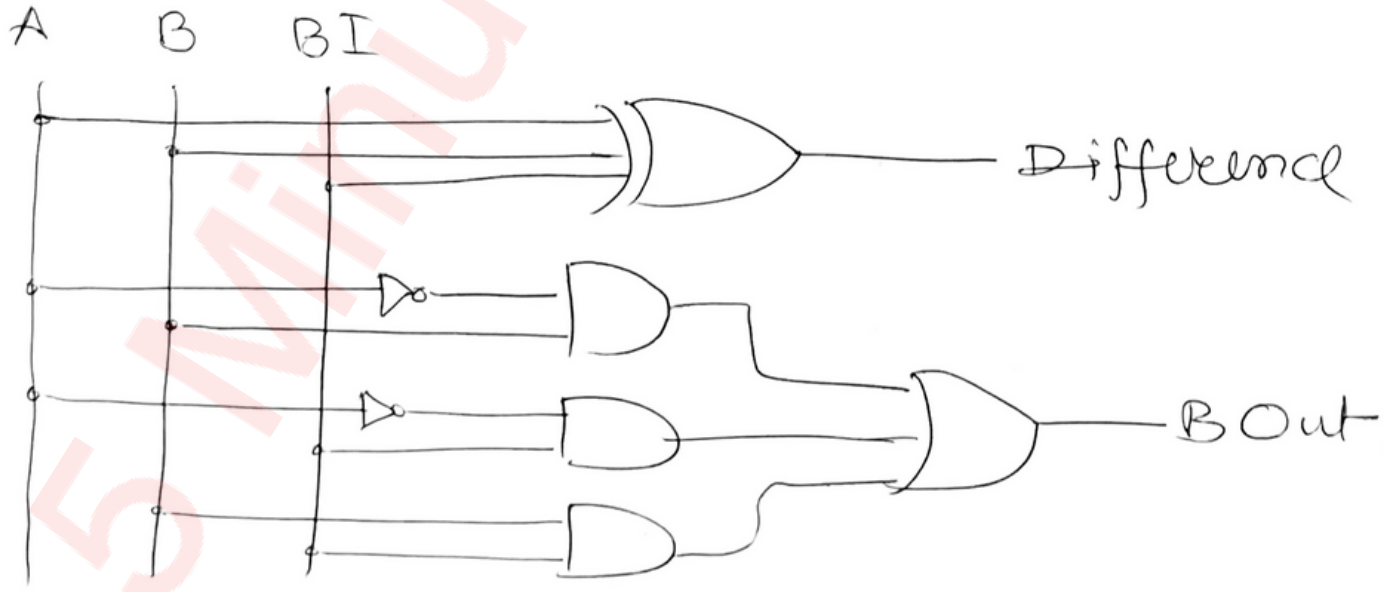


° full Subtractor

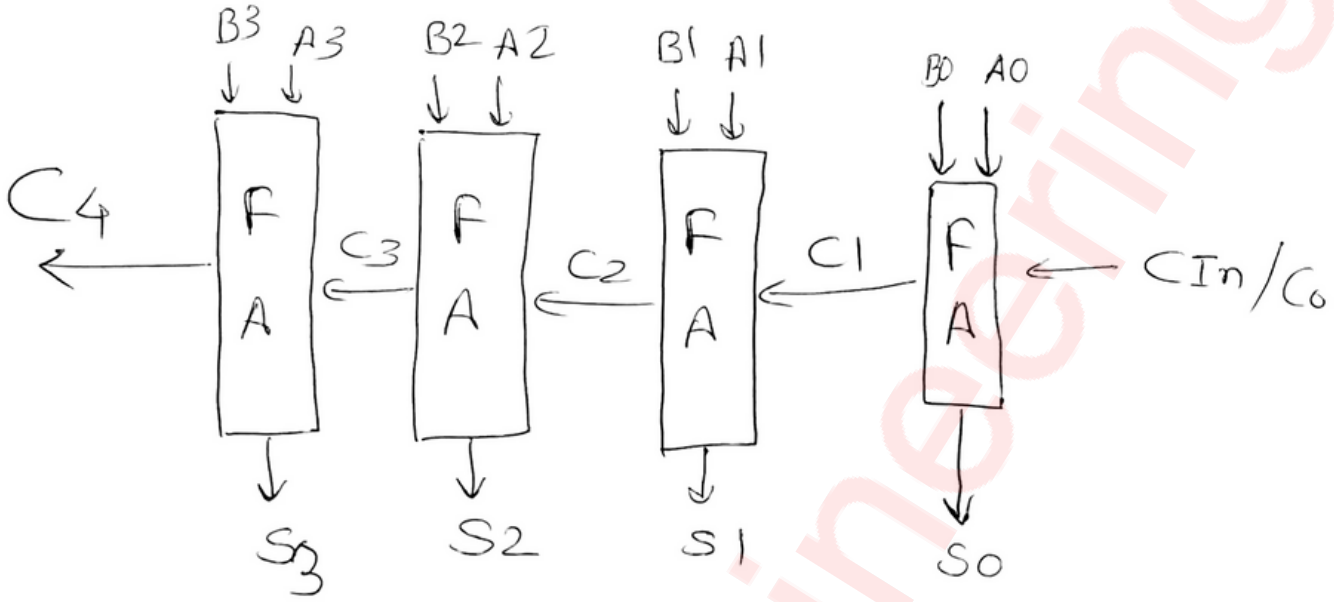


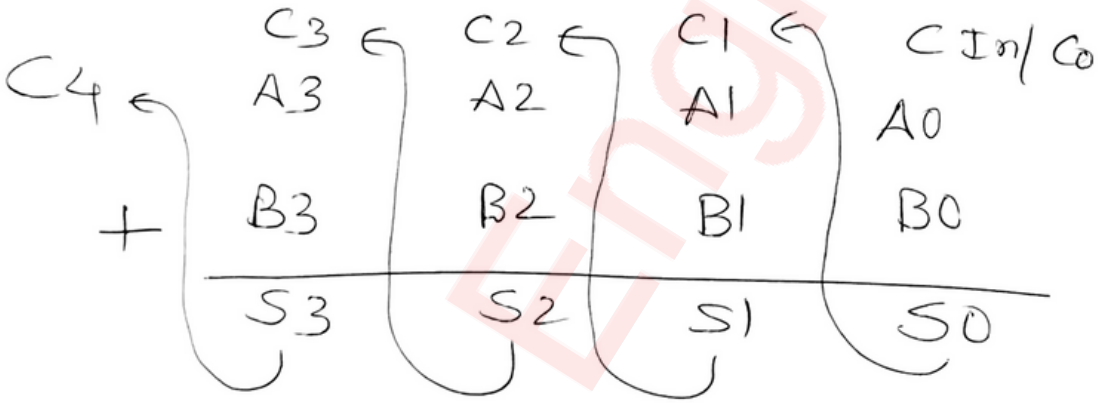
A	B	BI	D	BO
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

compliment



◦ Ripple Adder (4 bit adder)

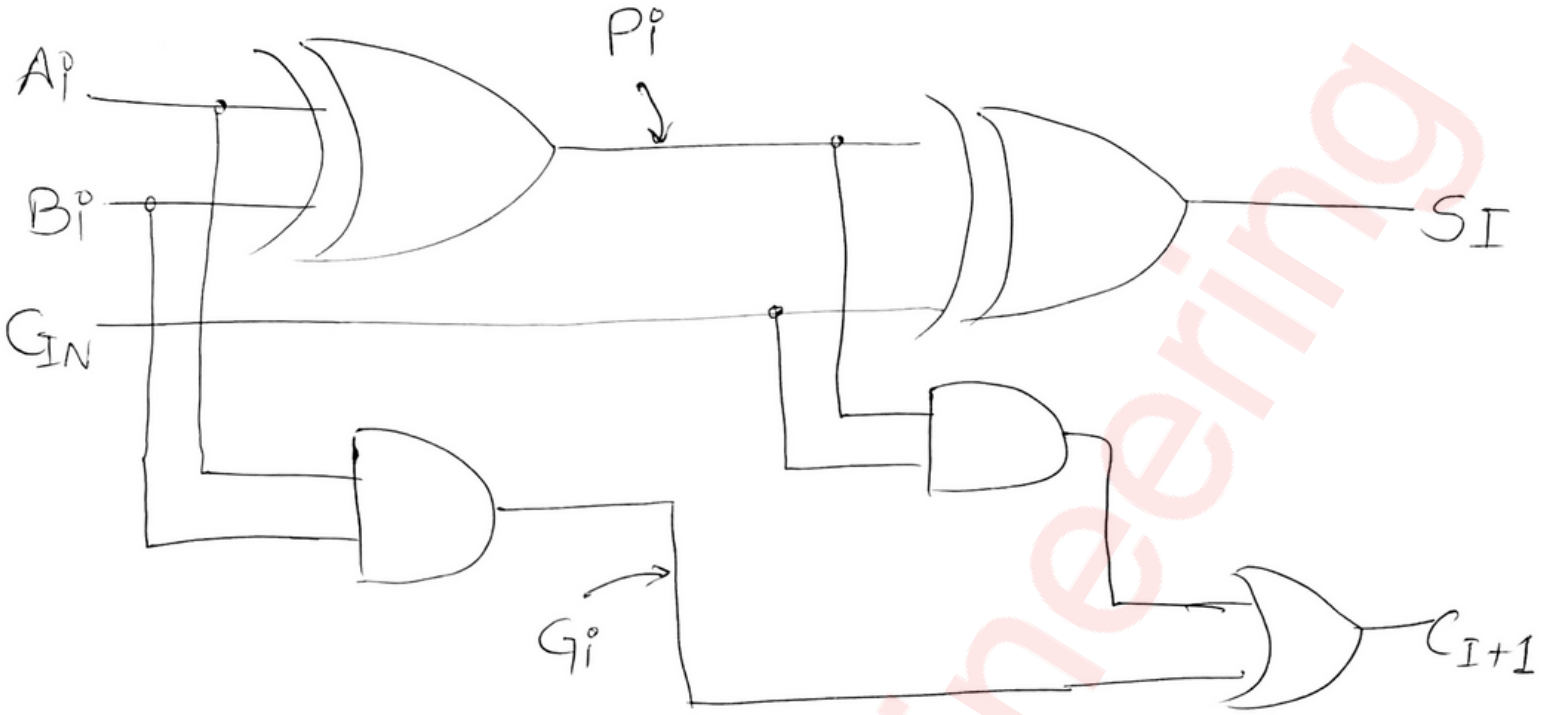




◦ Carry propagation Lag / Delay problem

So we use Look ahead carry adder

Idea is to get C1, C2, C3 from C0 itself.



$$P_i = A_i \oplus B_i$$

$$G_i = A_i B_i$$

$$S_i = P_i \oplus C_i \;/\; A_i \oplus B_i \oplus C_i$$

$$C_{i+1} = G_i + P_i C_i$$

$$C_0 = 0$$

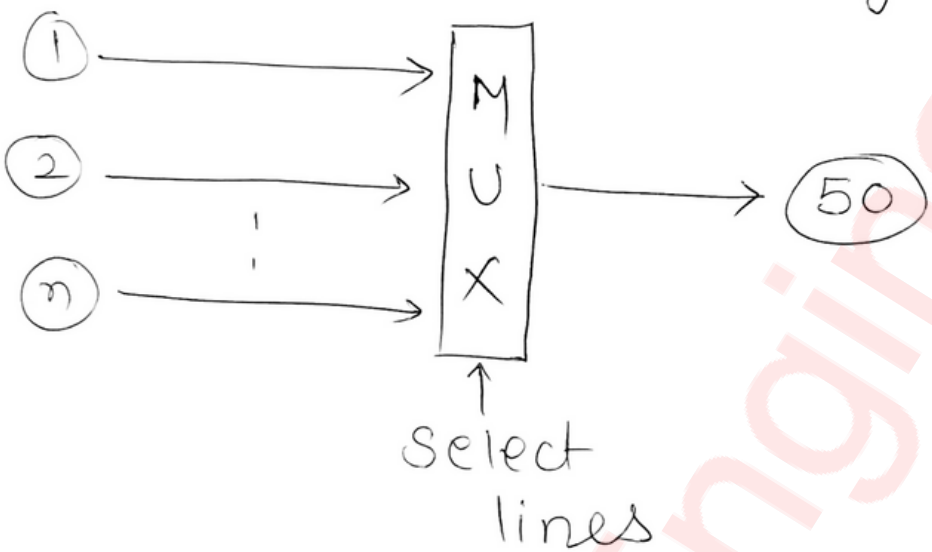
$$C_1 = G_0 + P_0 C_0$$

$$C_2 = G_1 + P_1 G_0 + P_1 P_0 C_0$$

$$C_3 = G_2 + P_2 G_1 + P_2 P_1 P_0 C_0$$

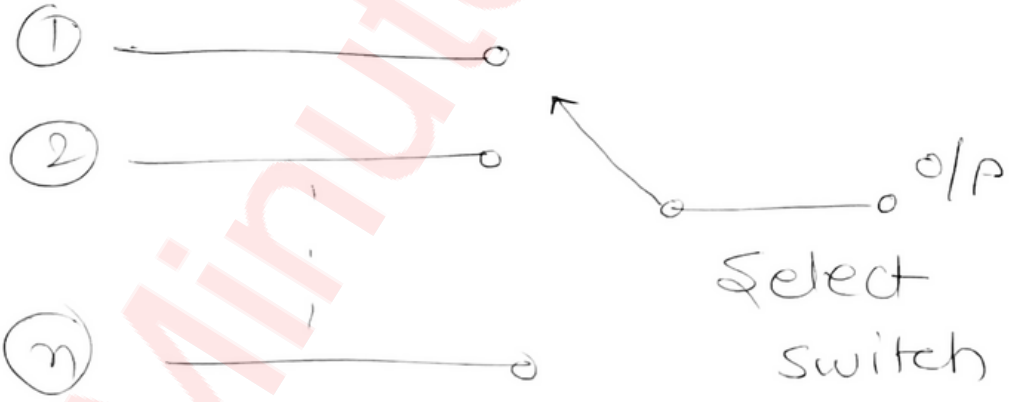
$$C_4 = G_3 + P_3 G_1 + P_3 P_2 G_0 + P_3 P_2 P_1 P_0 C_0$$

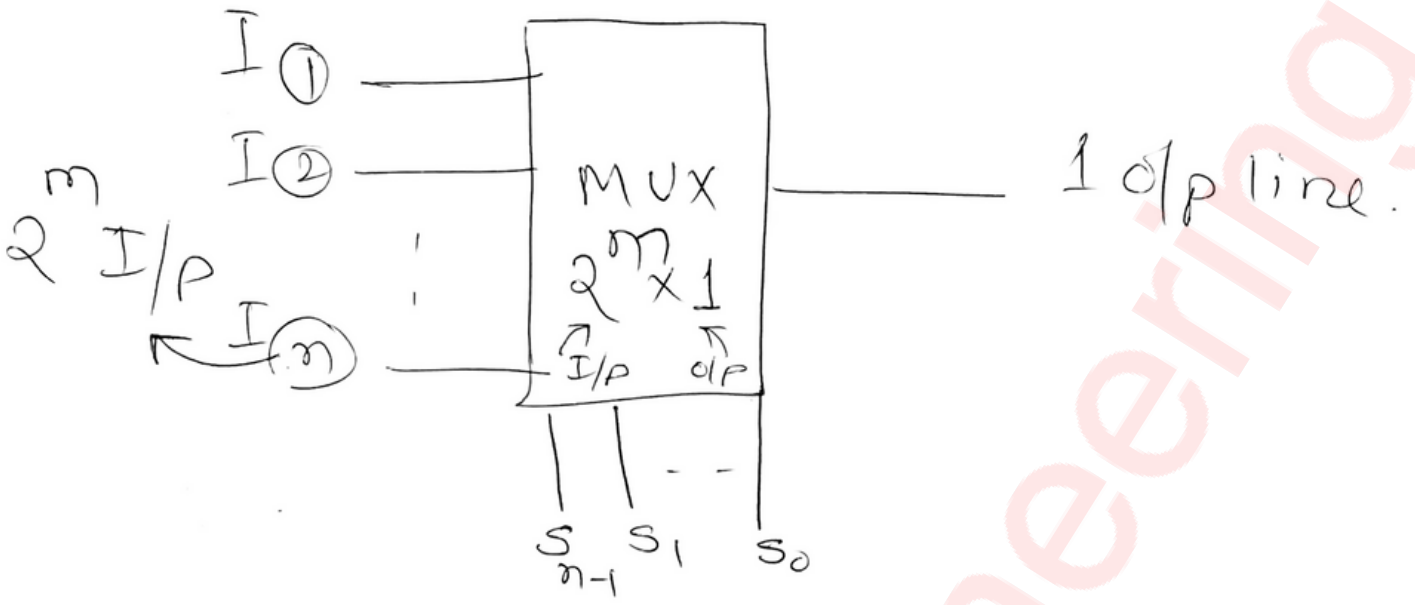
Multiplexer [Trending CC]



Multiple I/p lines — Single o/p Line

[Multiple option select one]

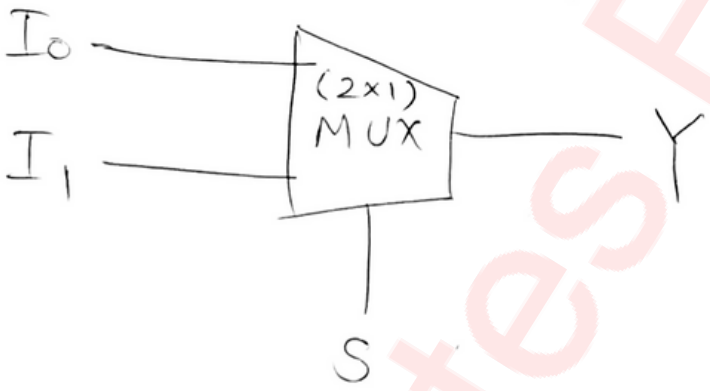




n select lines

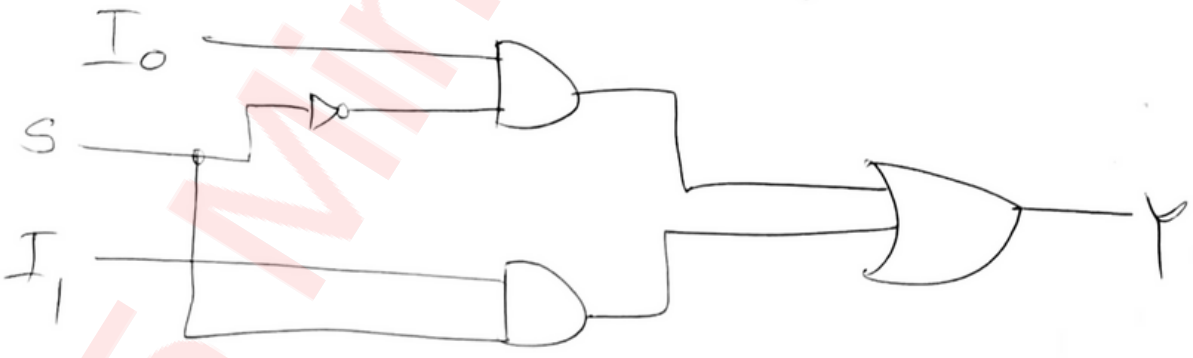
- Two-to-One Multiplexer

I/P → (2×1) → O/P

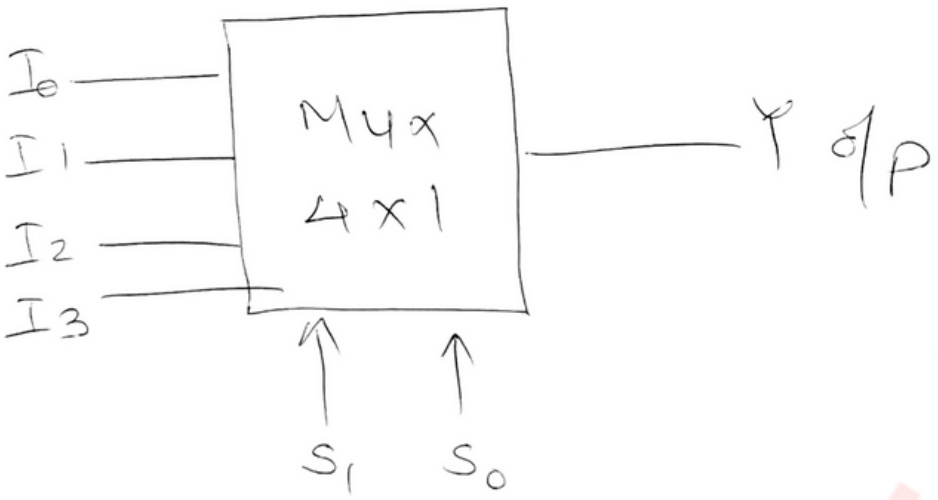


I/P	O/P
S	Y
0 →	I0
1 →	I1

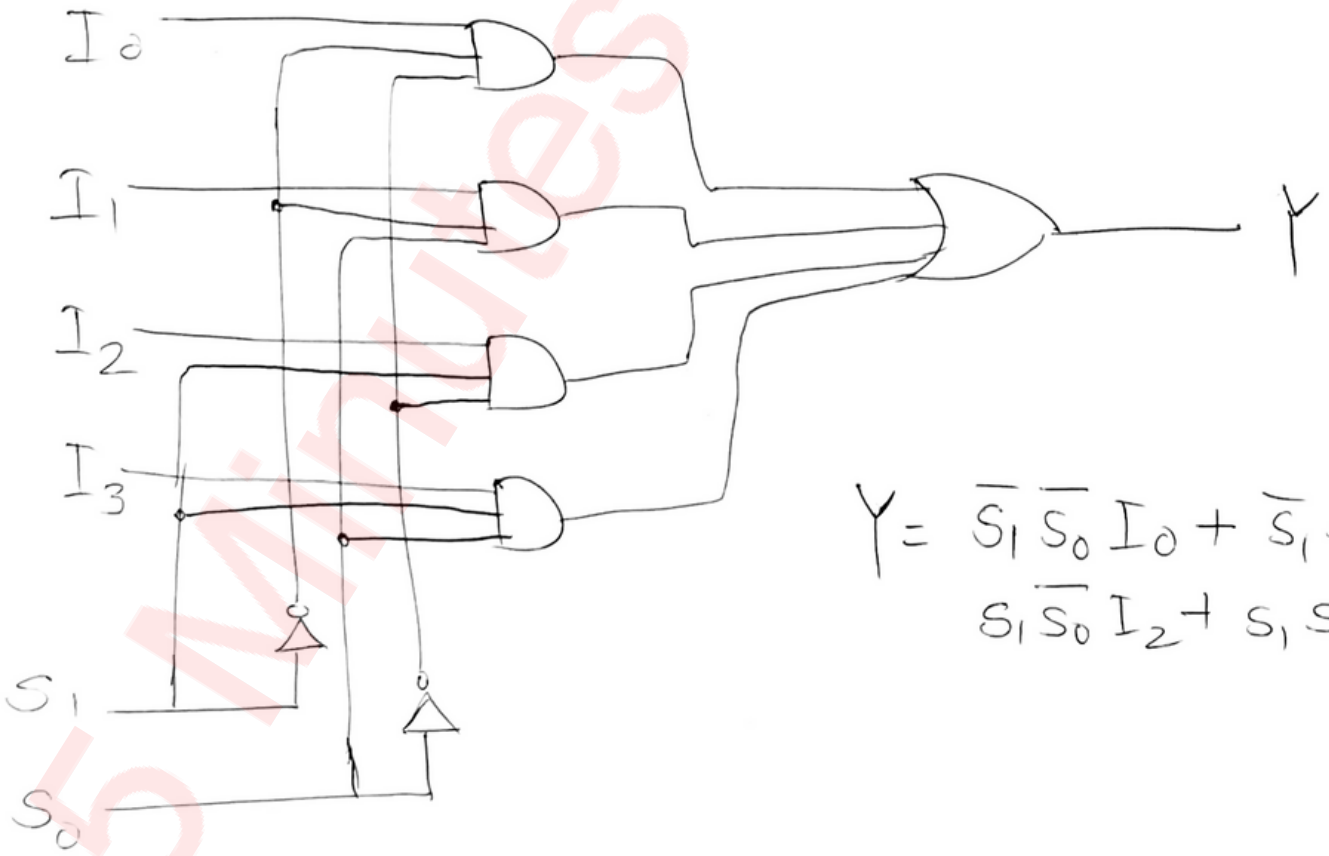
eqn = $Y = \overline{S_0} I_0 + S_0 I_1$



⇒ 4 x 1 Mux

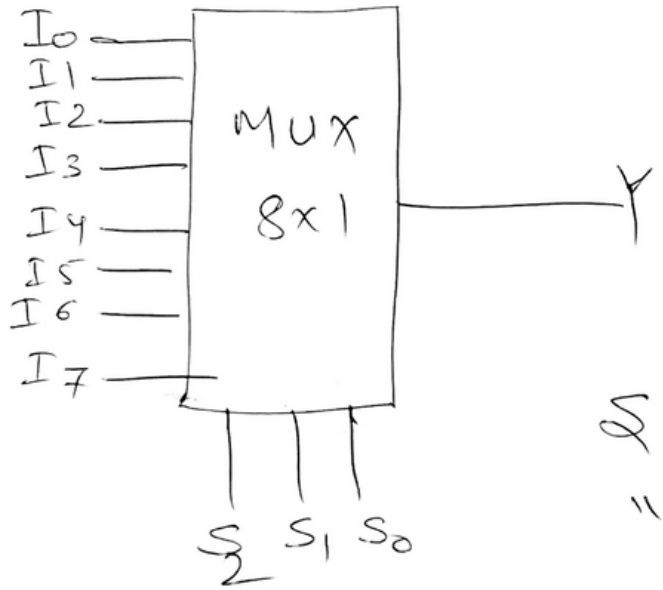


S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3



$$Y = \bar{S_1}\bar{S_0}I_0 + \bar{S_1}S_0I_1 + S_1\bar{S_0}I_2 + S_1S_0I_3$$

Eg: 8x1 Mux



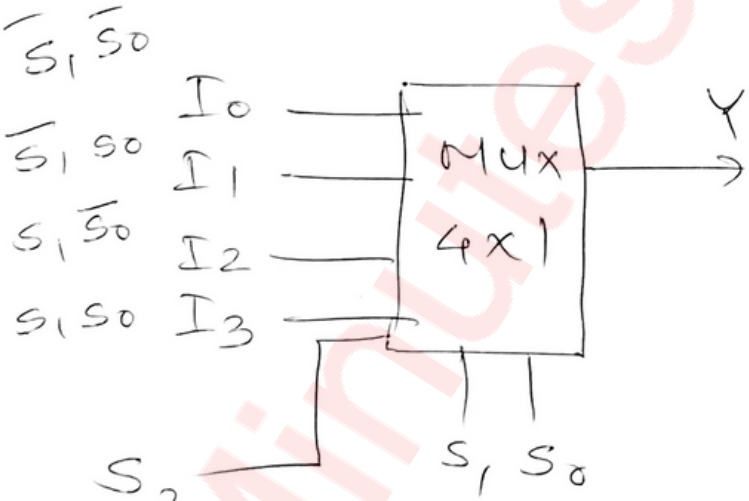
$\Sigma_m(0,1,2,3)$

Simply make.

" $I_0, I_1, I_2 I_3$ " o/p

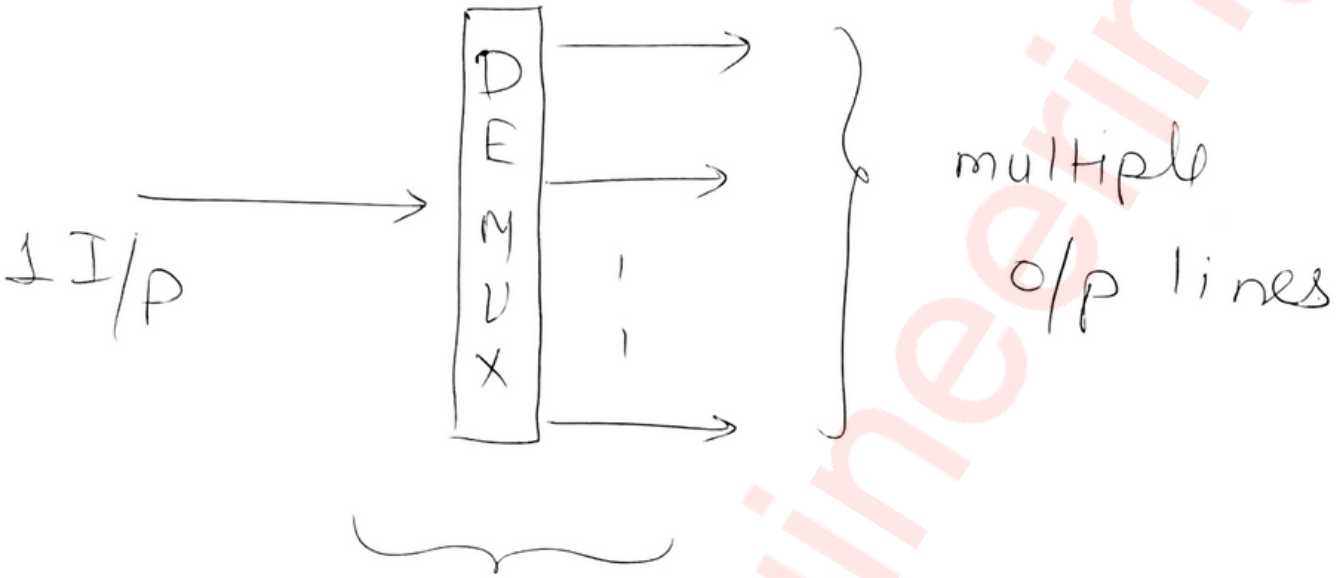
S_0	S_2	S_1	S_0
✓	0	0	0
✓	0	0	1
✓	0	1	0
✓	0	1	1

'OR' we can Implement the same thing using (4x1) Mux.

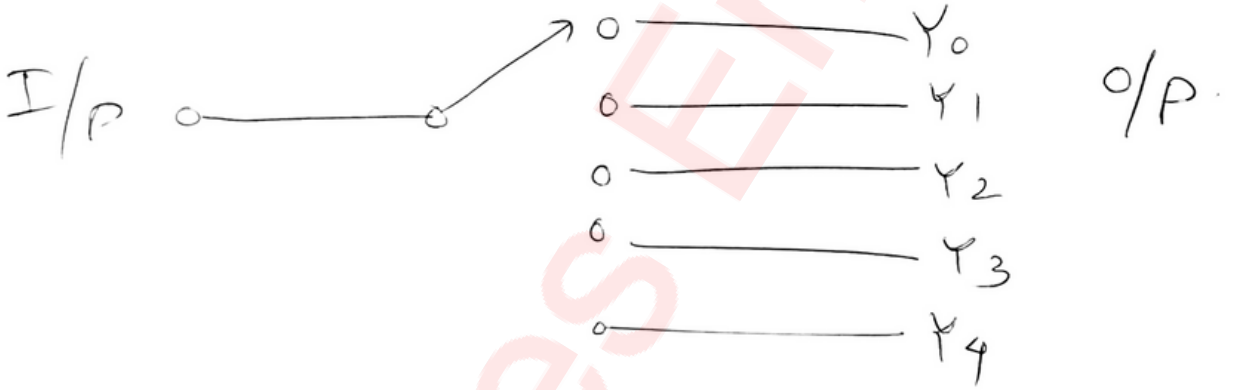


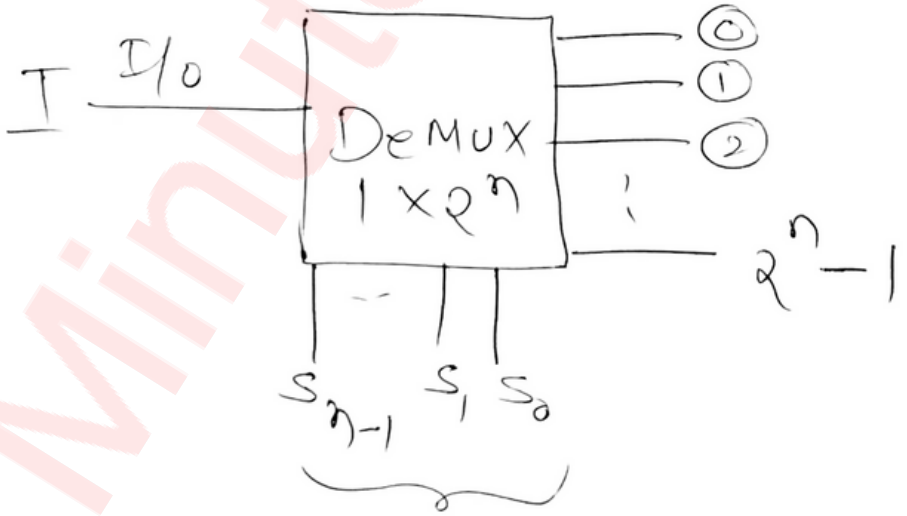
" 0/1 "

Demultiplexer [opposite of Mux]



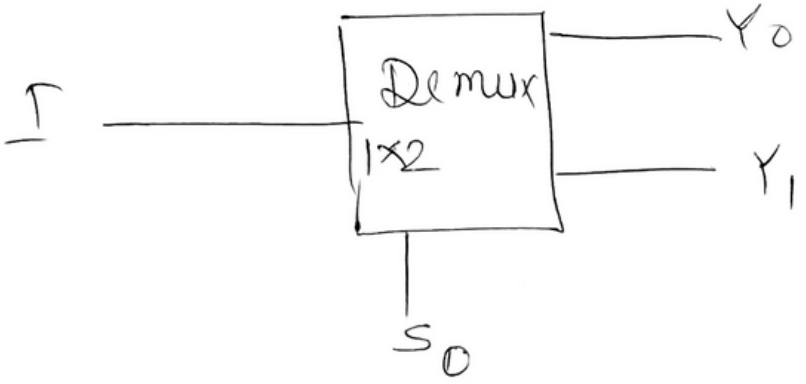
data distributor



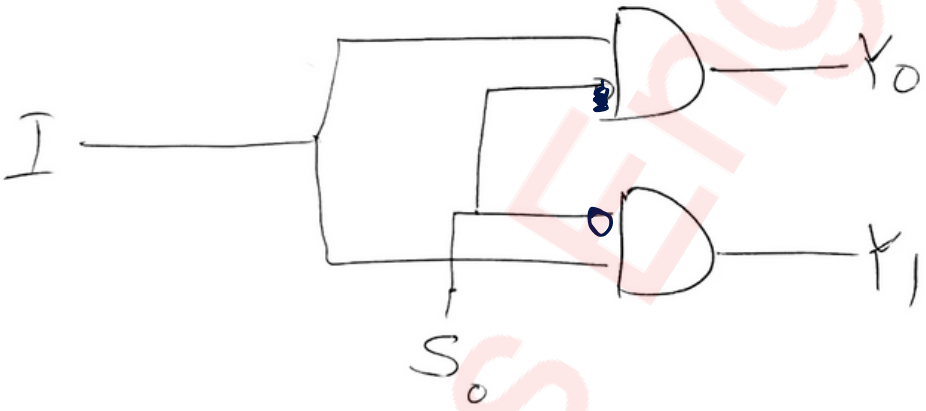


n select lines.

⇒ 1 x 2 Demultiplexer

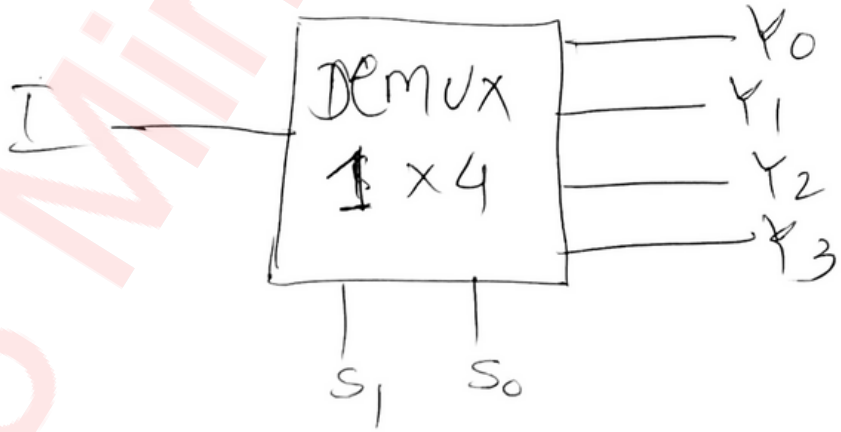


S_0	Y_0	Y_1
0	0	I
1	I	0

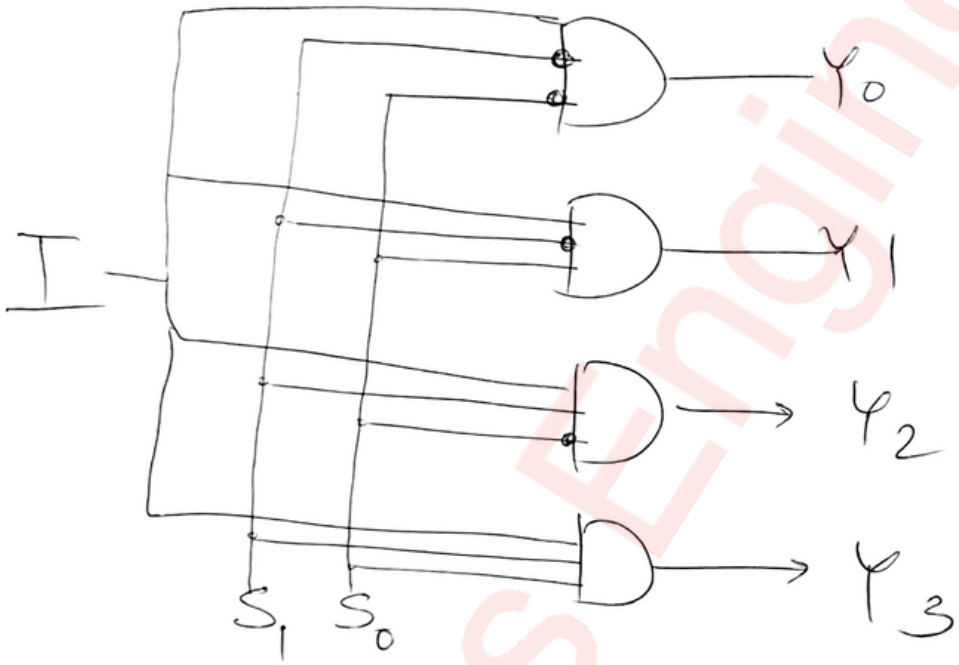


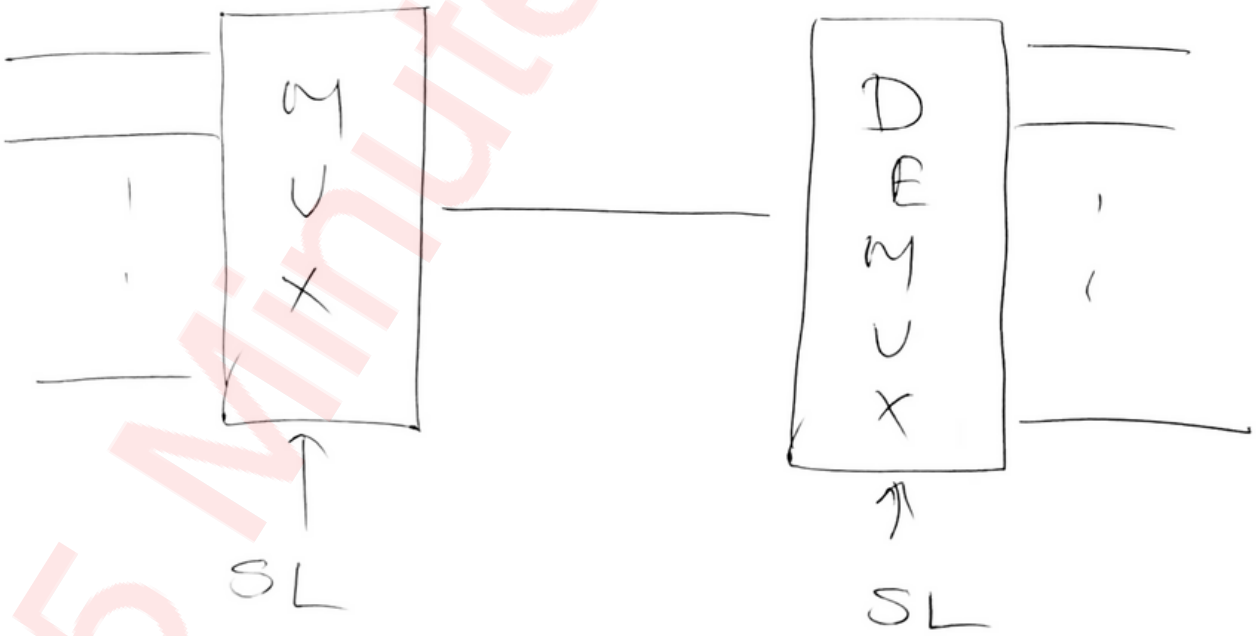
$Y_0 = S_0 \quad Y_1 = \overline{S_0}$

⇒ 1 x 4 Demultiplexer

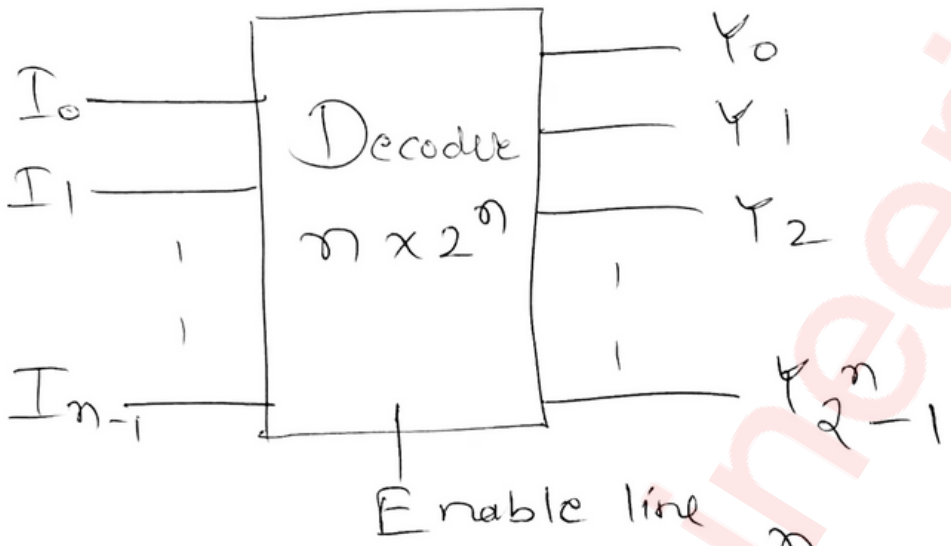


S_0	S_1	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	I
0	1	0	0	I	0
1	0	0	I	0	0
1	1	I	0	0	0





Decoder



n bits $\longrightarrow$ 2^n options

Hence $n \times 2^n$

Eg: $n = 2$

I_0	I_1	
0	0	Y_0
0	1	Y_1
1	0	Y_2
1	1	Y_3

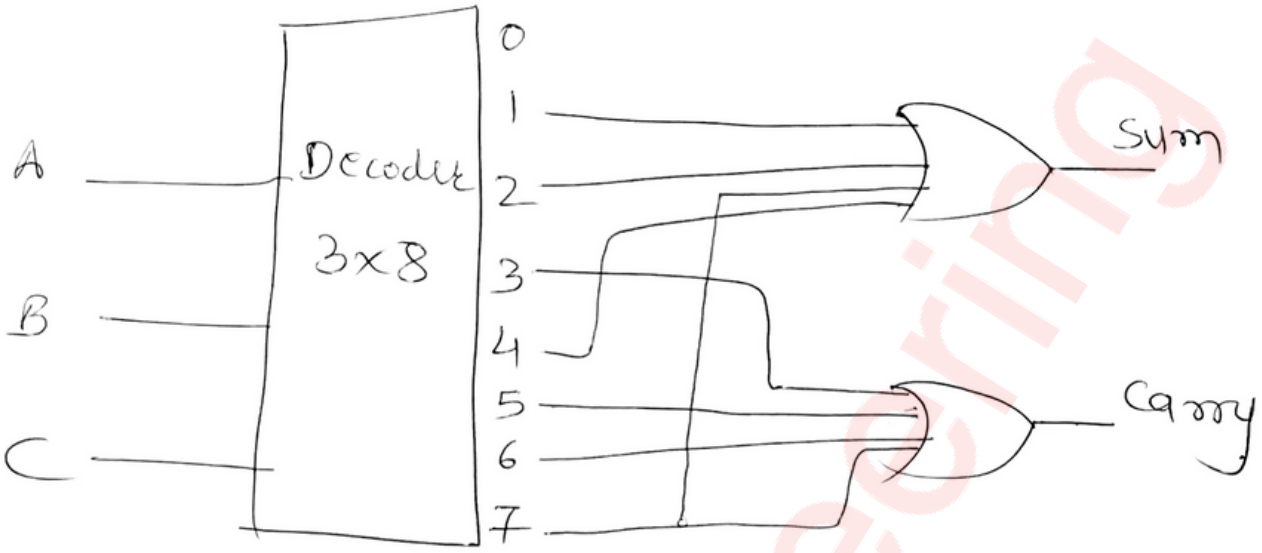
$2^n = 2^2 = 4$

(Decoder) " SL $\leftrightarrow$ I " (DeMux)

I $\rightarrow$ EL
SL $\rightarrow$ I

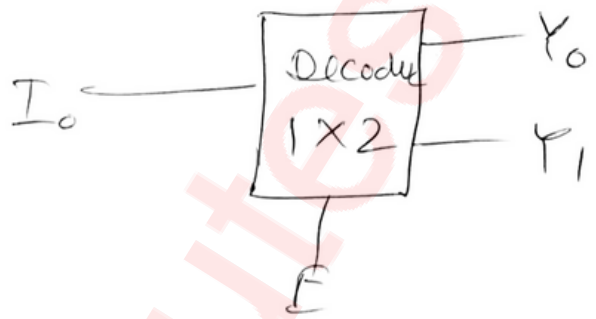
EL $\rightarrow$ I
I $\rightarrow$ SL

Eg

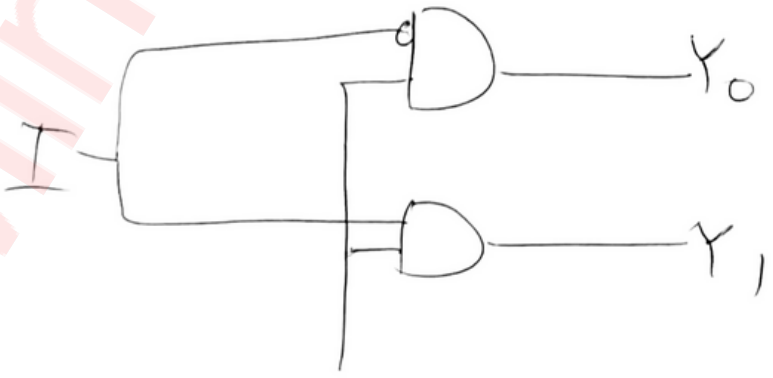


0	1	2	3	4	5	6	7
$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}C$	$\bar{A}B\bar{C}$	$\bar{A}BC$	$A\bar{B}\bar{C}$	$A\bar{B}C$	$AB\bar{C}$	ABC
0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1

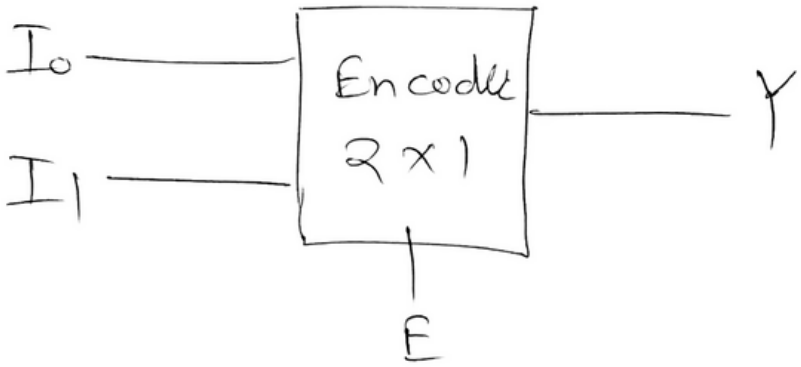
Eg :- 1 x 2 Decoder

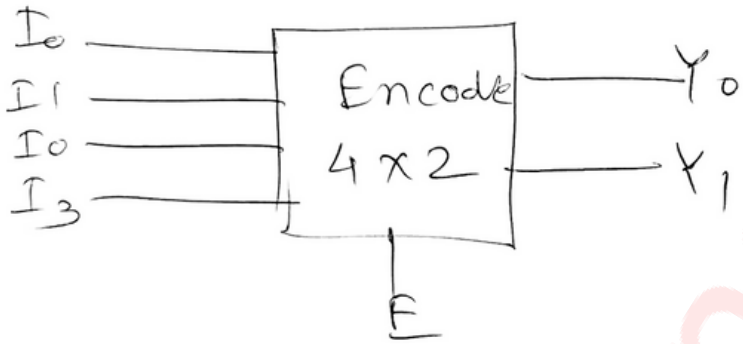


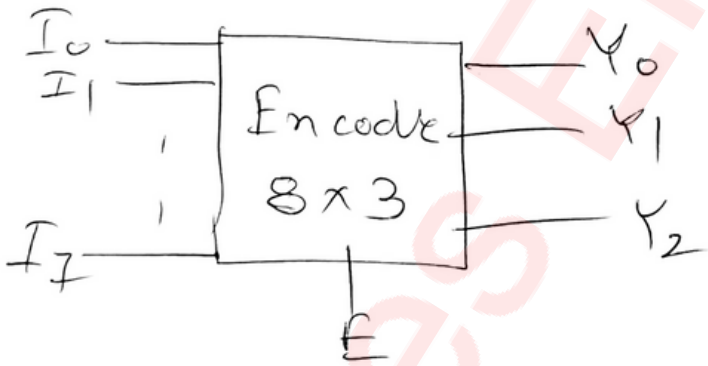
I	Y_0	Y_1
0	0	1
1	1	0



◦ Encoder







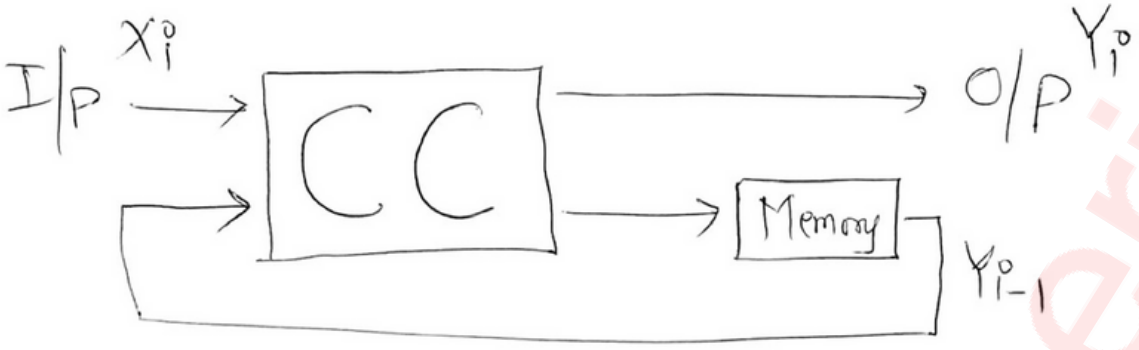
2^n I/P

n O/P

◦ Priority Encoder [Assign the priority to multiple I]

I_3	I_2	I_1	I_0	Y_1	Y_0
0	0	0	1	0	0
0	0	1	DC	0	1
0	1	DC	DC	1	0
1	DC	DC	DC	1	1

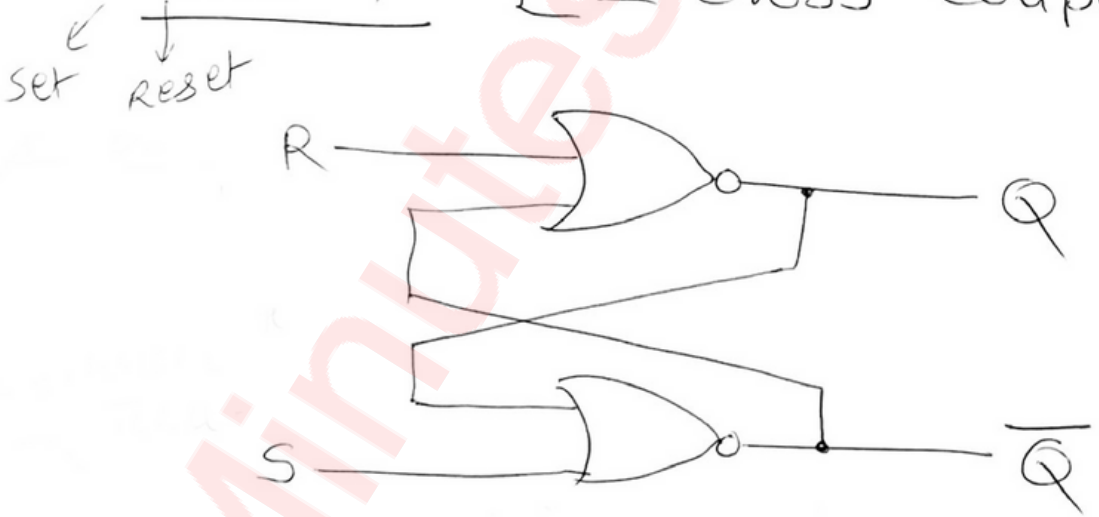
Sequential Circuits = CC + Memory Element



$$F(x_i, Y_{i-1}) = Y_i$$

→ Latch (used to hold a bit)
↳ used in making of FFs.

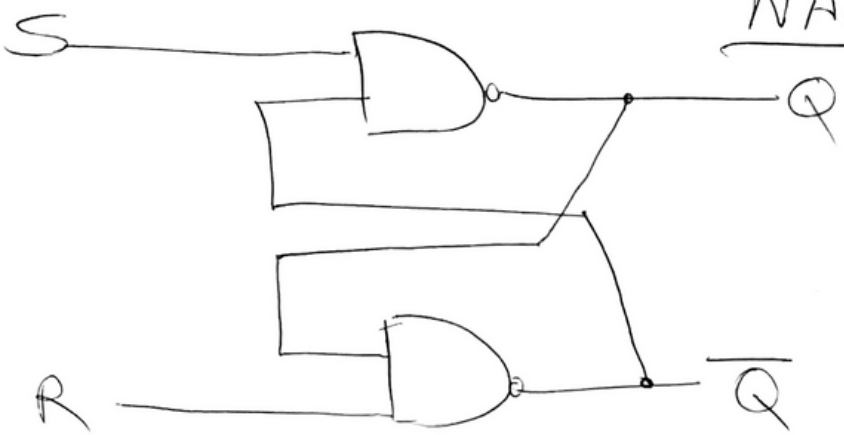
• SR Latch [2 cross coupled NOR gates, NAND]



SR Latch → SR FF → SR TT → SR CT → SR ET

S	R	Q_n	Q_{n+1}	
0	0	0	0	same/NO change.
0	0	1	1	
0	1	0	0	Reset
0	1	1	0	
1	0	0	1	set
1	0	1	1	
1	1	0	X	Random
1	1	1	X	

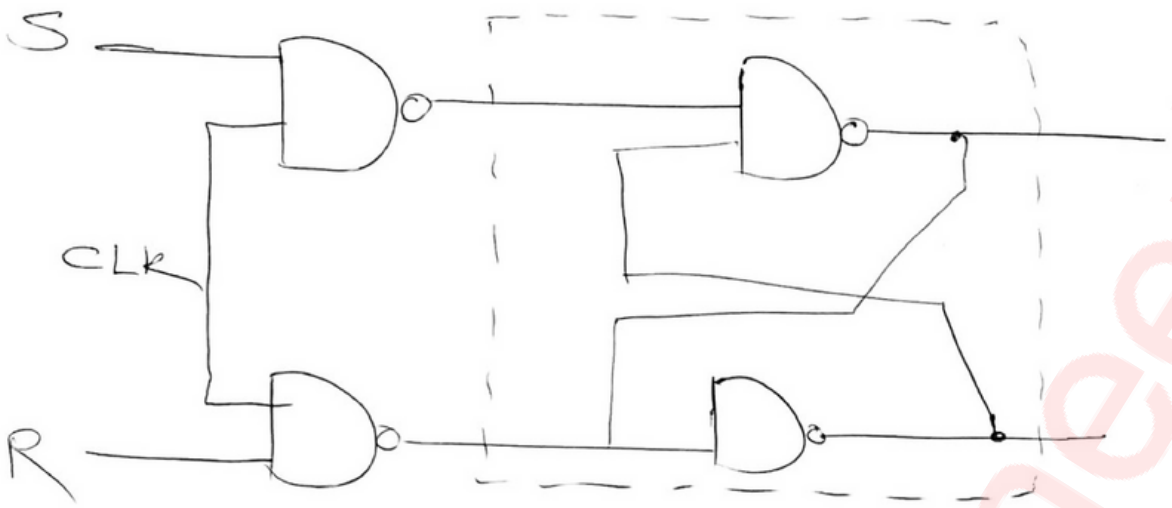
NAND (S ⟷ R)
NOR



S	R	Q_n	Q_{n+1}	
0	0	0	X	Invalid / Random
0	0	1	X	
0	1	0	1	set
0	1	1	1	
1	0	0	0	Reset
1	0	1	0	
1	1	0	0	Same / NO change
1	1	1	1	

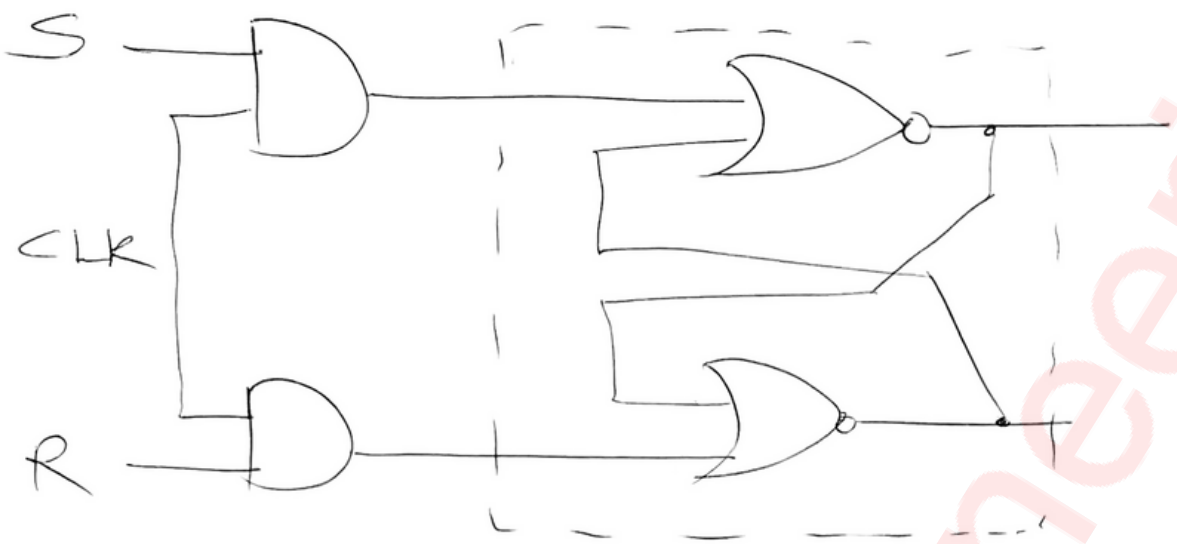
S	R	Q_{n+1}
0	0	Invalid (X)
0	1	set
1	0	Reset
1	1	NO change

SR Flip-Flop (NAND)



S	R	CLK	Q_{n+1}
0	0	T/1	Q_n [Same / Hold]
0	1	T/1	0 [Reset]
1	0	T/1	1 [set]
1	1	T/1	X [Invalid]
X	X	NT (0)	Q_n

(NOR Gate) SR FF



S	R	CLK	Q_{n+1}
0	0	T/↓	[Same/Hold] Q_n
0	1	T/↓	0 [Reset]
1	0	T/↓	1 [set]
1	1	T/↓	x [Invalid]
x	x	NT	Q_n

Same as NOR latch situation.

SR FF Excitation Table

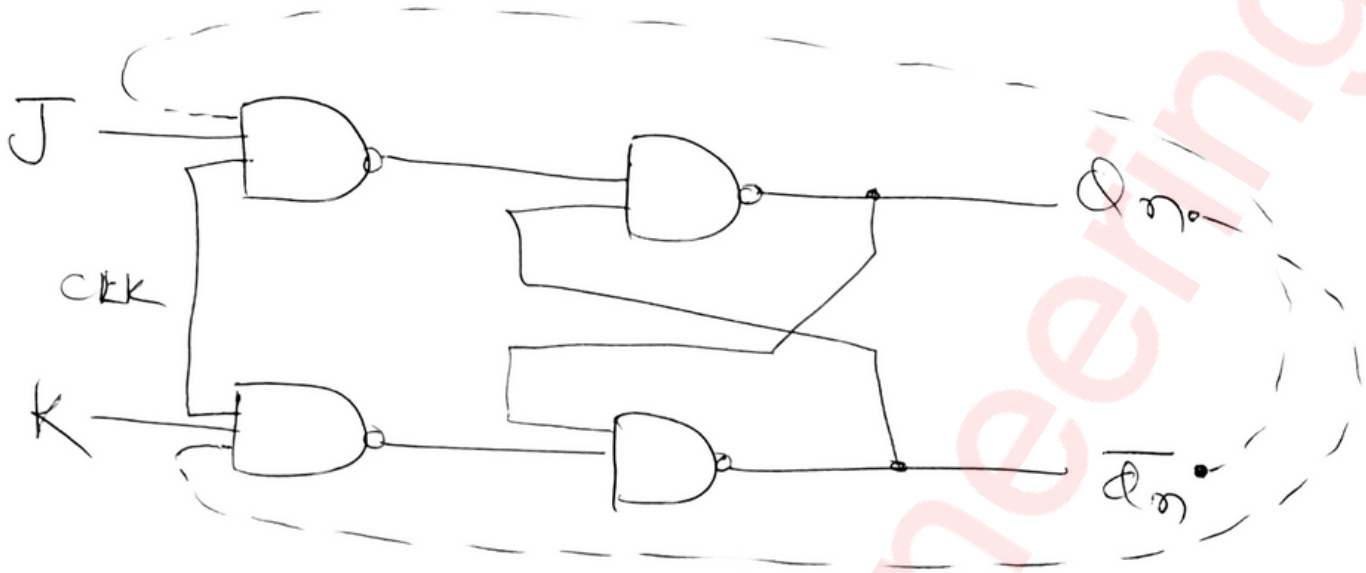
S	R	Qn	Qn+1	
0	0	0	0	Same/hold
0	0	1	1	
0	1	0	0	Reset
0	1	1	0	
1	0	0	1	set
1	0	1	1	
1	1	0	x	Invalid.
1	1	1	x	

$Q_{n+1} = S + \bar{R}Q_n$ → characteristic eqn.

Qn	Qn+1
0	0
0	1
1	0
1	1

S	R
0	x
1	0
0	1
x	0

J K flip flop



S	R	Q_{n+1}
0	0	Hold
0	1	Reset
1	0	set
1	1	Invalid ← focuses / fix this.

Now when

J = 1, K = 1

→ $Q_n = 1$; [1, 0] → Q_{n+1} Reset (0)

→ $Q_n = 0$; [0, 1] → Q_{n+1} set (1)

finally ↓ $\overline{Q_n}$

Toggle

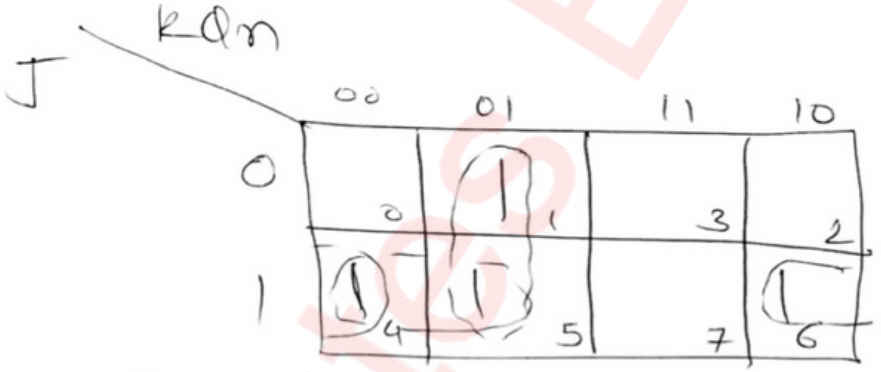
Q_n : 0 → Q_{n+1} : 1

1 → 0

JK flip flop Characteristic Table

⇒

J	k	Q_n	Q_{n+1}	
0	0	0	0	Hold
0	0	1	1	
0	1	0	0	Reset
0	1	1	0	
1	0	0	1	set
1	0	1	1	
1	1	0	1	$\bar{Q}$
1	1	1	0	

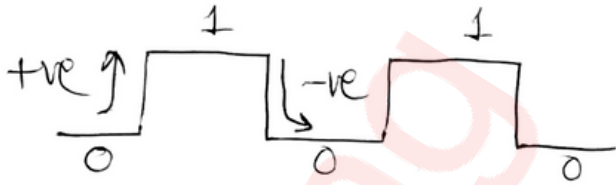


⇒ $Q_{n+1} = J\bar{Q}_n + \bar{K}Q_n$ → char. eqn

- Excitment table

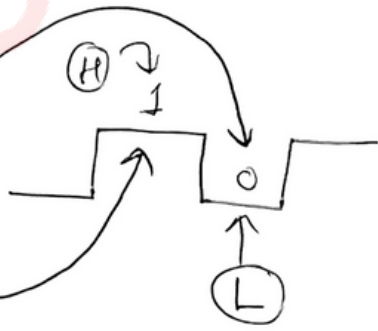
Q_n	Q_{n+1}	J	k
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

→ Triggering

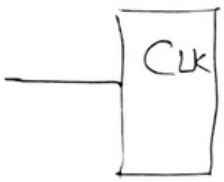


→ Edge → Negative
→ Positive

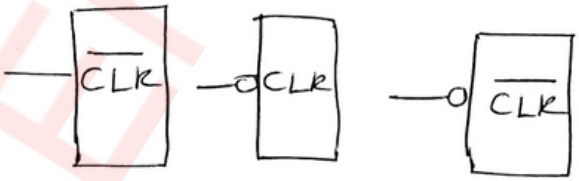
→ Level → Low
→ High



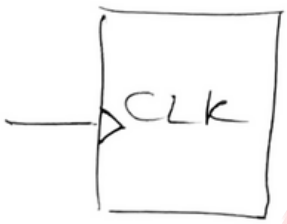
+ve level



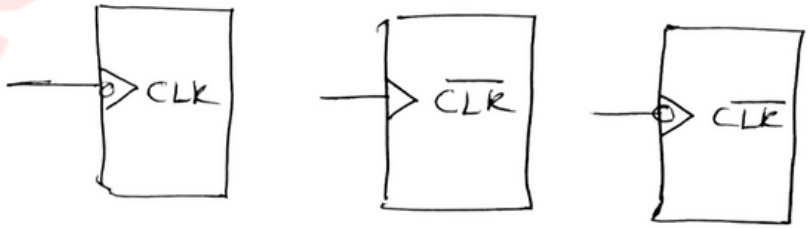
-ve level



+ve edge



-ve Edge



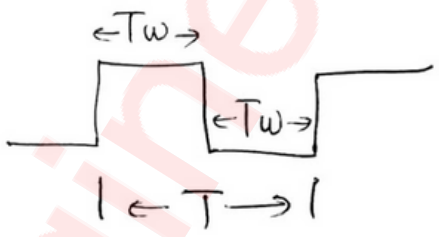
◦ Race condition In JK ff

① K = J = 1 [Toggle]

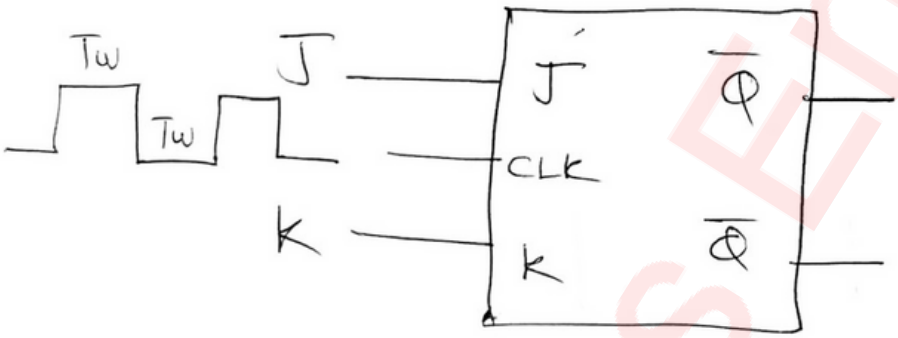
② Level Triggered

③ Td < Tw

↓ delay



$T_w = \frac{1}{2}(T)$



0 → 1 → 0

1 → 0 → 1

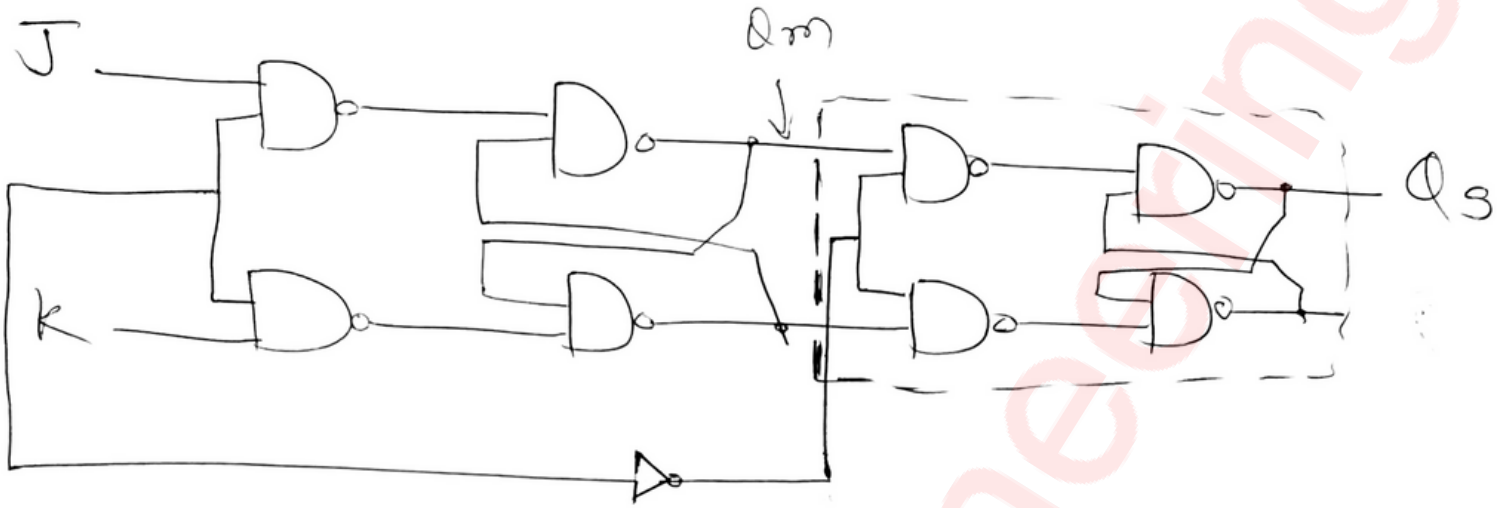
↑ Initially

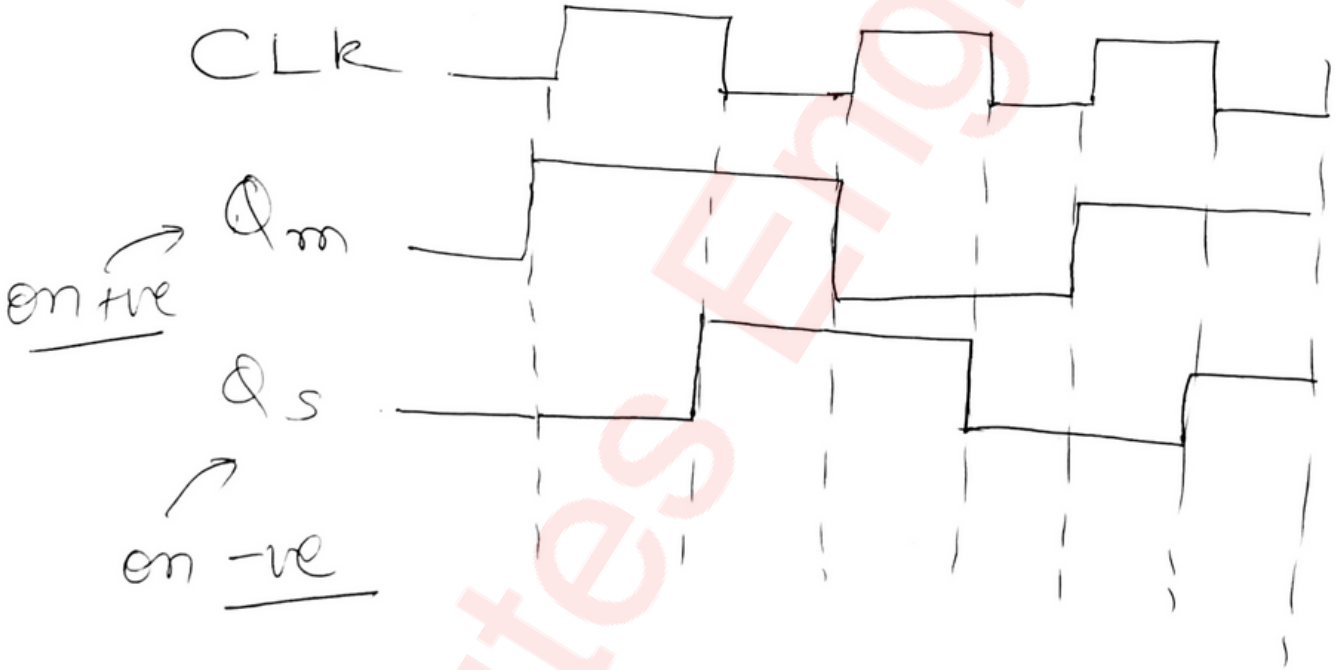
↑ Toggle

↑ Again Toggle

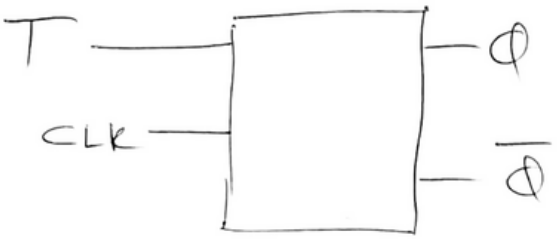
Due to Tw > Td

◦ Master slave JK FF





T flip flop



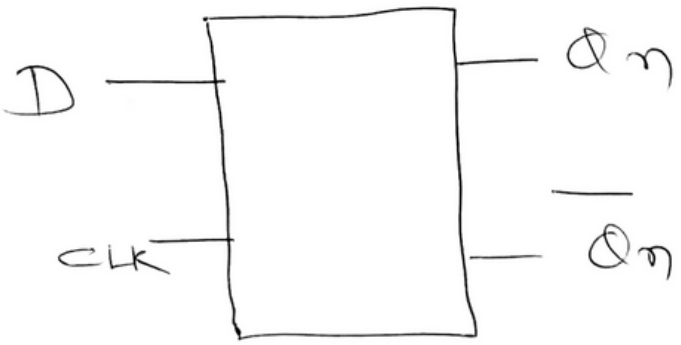
T	Q_{n+1}	
0 →	Q_n	} Toggle behaviour
1 →	$\overline{Q_n}$	

T	Q_n	Q_{n+1}	
0	0	0	] as T = 0 [NO Toggle]
0	1	1	
1	0	1	} as T = 1 [Toggle]
1	1	0	

$$Q_{n+1} = T \oplus Q_n$$

Q_n	Q_{n+1}	T	
0	0	0	
0	1	1	} mirror
1	0	1	
1	1	0	

D flip flop



D	Q_{n+1}	
0 →	0	Same as D
1 →	1	

D	Q_n	Q_{n+1}	
0	0	0	Same as D
0	1	0	
1	0	1	
1	1	1	

$$Q_{n+1} = D$$ characteristic eqn.

Q	Q_{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

SR → D ff conversion

O/P → Dff characteristic Table

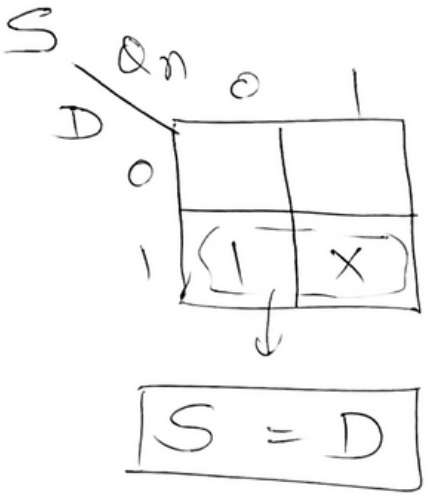
I/P → SR Excitation Table

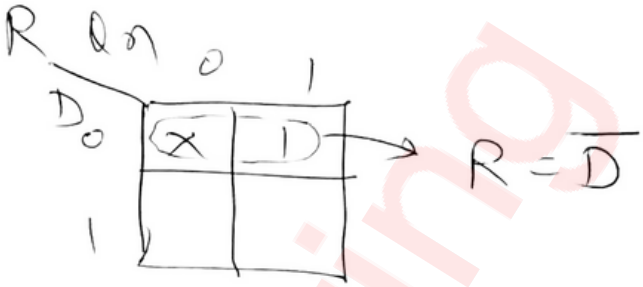
Q_n	Q_{n+1}	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

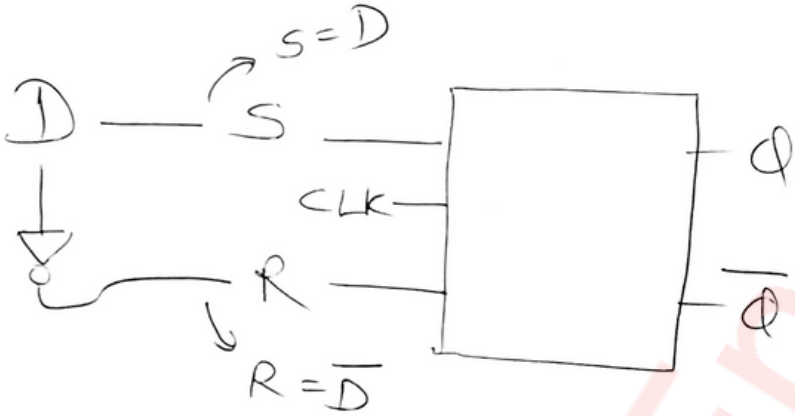
D	Q_n	Q_{n+1}
0	0	0
0	1	0
1	0	1
1	1	1

D	Q_n	Q_{n+1}	S	R
0	0	0	0	X
0	1	0	0	1
1	0	1	1	0
1	1	1	X	0

Refer to ET (SR)







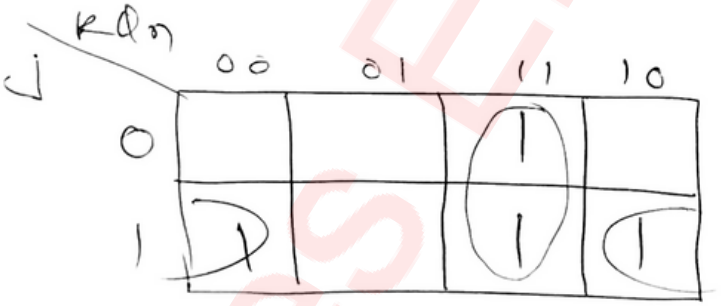
⇒ T → JK ff conversion

↓(ET) (CT)

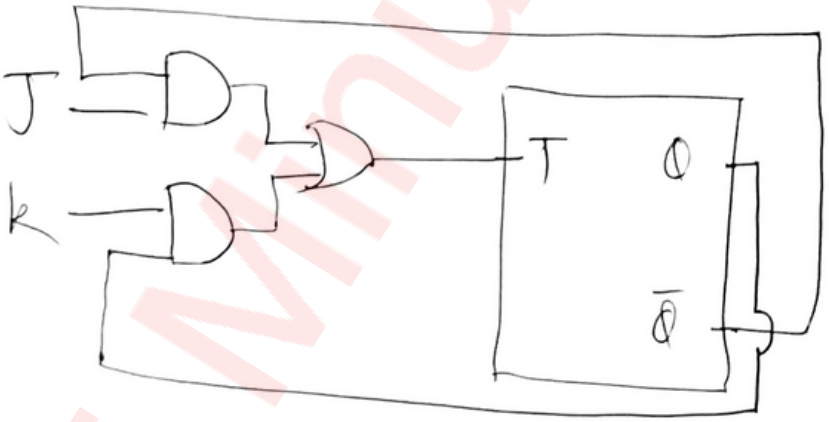
Qn	Qn+1	T
0	0	0
0	1	1
1	0	1
1	1	0

J	k	Qn	Qn+1	
0	0	0	0	] samp.
0	0	1	1	
0	1	0	0	] Reset
0	1	1	0	
1	0	0	1	] Set
1	0	1	1	
1	1	0	1	] Toggle
1	1	1	0	

J	k	Q_n	Q_{n+1}	T
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	1
1	0	1	1	0
1	1	0	1	1
1	1	1	0	1



$$T = J\bar{Q}_n + kQ_n$$

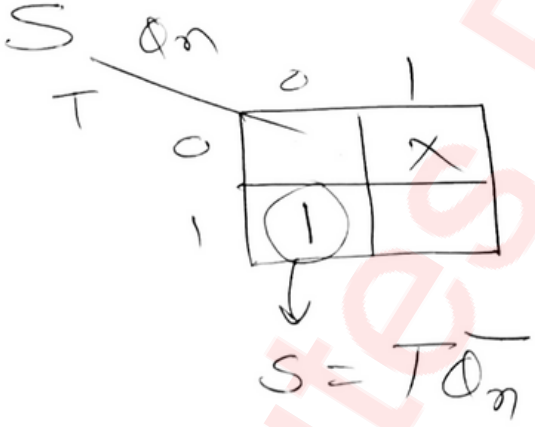


SR — T ff Conversion

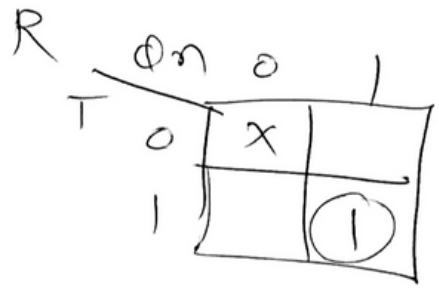
(ET) (CT)

T	Q_n	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

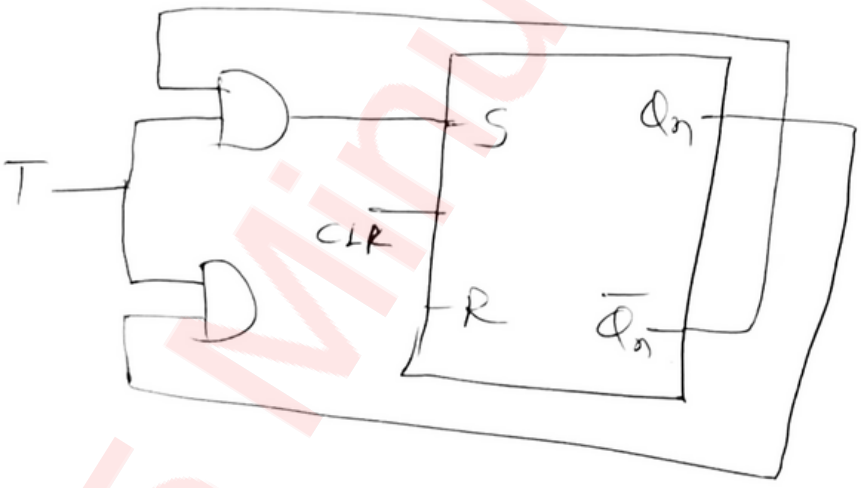
S	R
0	X
X	0
1	0
0	1



$S = T\bar{Q}_n$



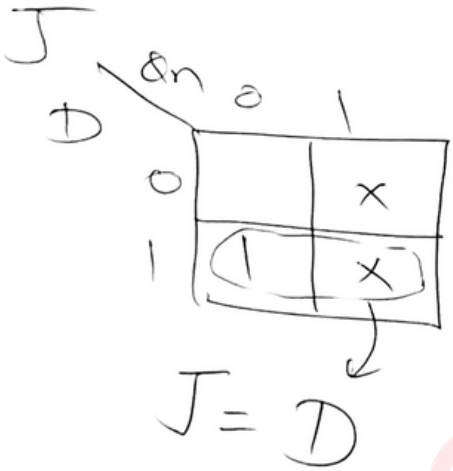
$R = TQ_n$

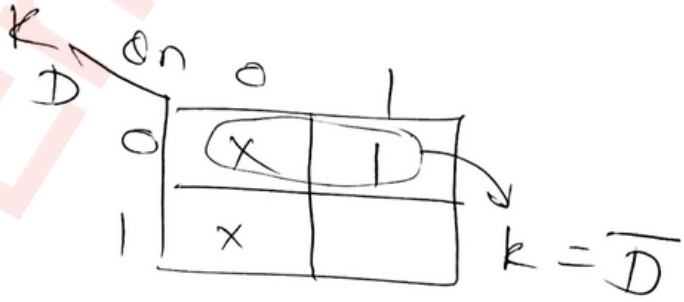


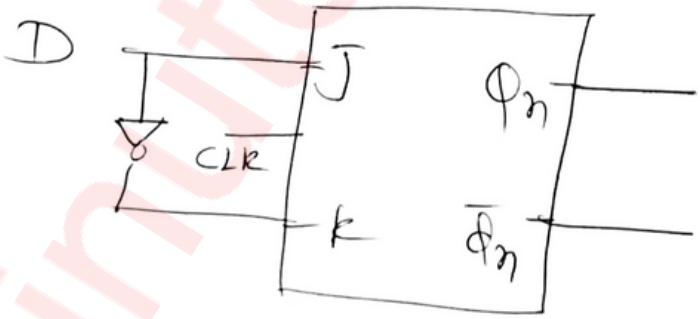
Jk — D ff conversion

(ET) (CT)

D	Q_n	Q_{n+1}	J	k
0	0	0	0	X
0	1	0	X	1
1	0	1	1	X
1	1	1	X	0







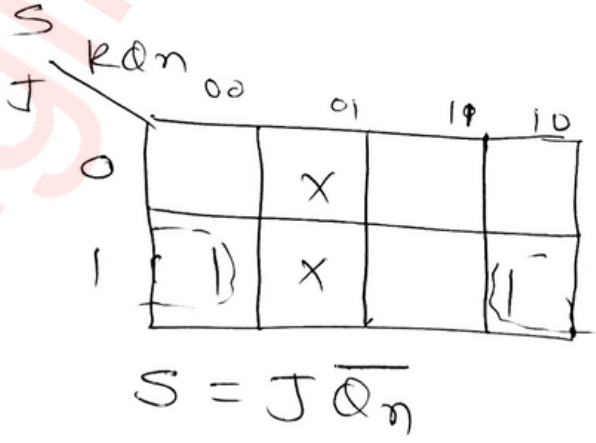
SR → JK Conversion

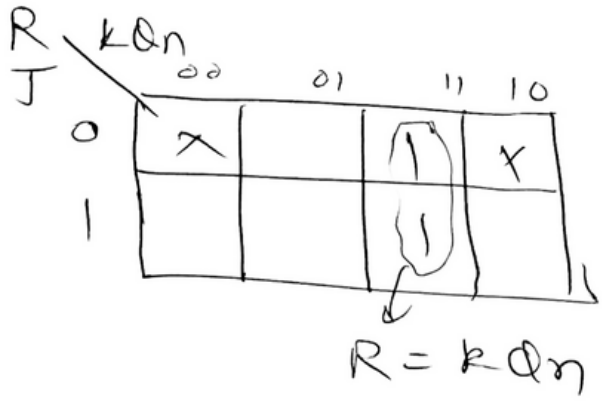
(ET) (CT)

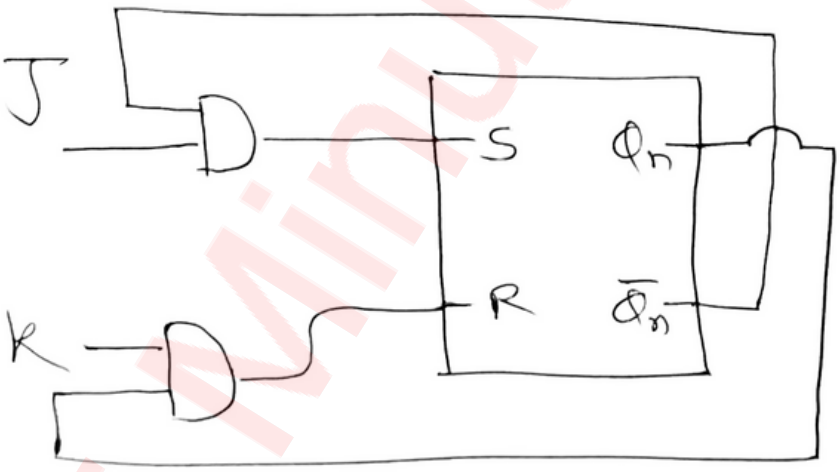
SR (ET)

Q_n	Q_{n+1}	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

J	K	Q_n	Q_{n+1}	S	R
0	0	0	0	0	X
0	0	1	1	X	0
0	1	0	0	0	X
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	1	X	0
1	1	0	1	1	0
1	1	1	0	0	1







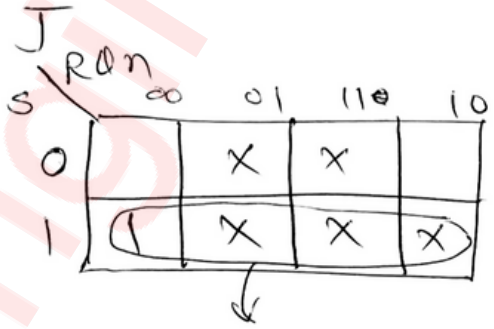
JK → SR Conversion

(ET) (CT)

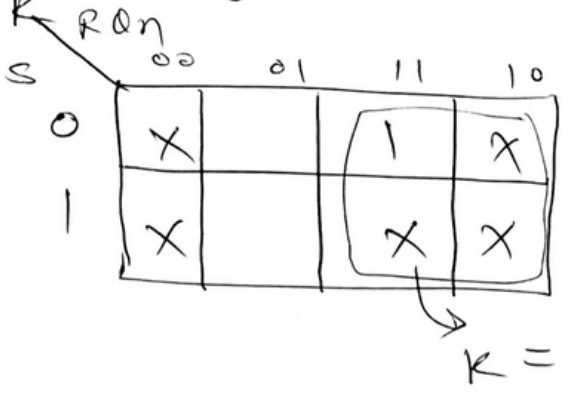
JK (ET)

Q_n	Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

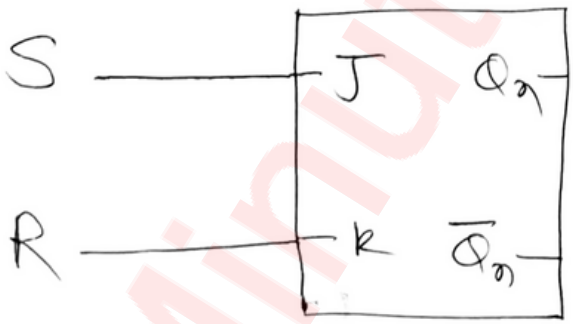
S	R	Q_n	Q_{n+1}	J	K
0	0	0	0	0	X
0	0	1	1	X	0
0	1	0	0	0	X
0	1	1	0	X	1
1	0	0	1	1	X
1	0	1	1	X	0
1	1	0	X	X	X
1	1	1	X	X	X



$J = S$



$K = R$



- **Counters**: count / frequency / occurrence no. of times.
  [store & Display]

Eg: ◦ Washing maching
◦ Microwave oven
◦ Traffic lights

→ n-bit counter
↓ have
n flip flops
[0 to $2^n - 1$]

- Types
  - → Synchronous (parallel counter)
  - → Asynchronous (Ripple counter)

Eg: 0 → 1 → 2 → 3 → 4 → 5 (↺)

↓
Mod 6 Counter
↑
unique states

Q: at 10 clk pulse whats the state.

0 → 1 → 2 → 3 → 4 → 5 → 0 → 1 → 2 → 3
↓
4 ✓

Simply multiple of '6' will take you to state '0' or value 0

Like: 12, 18, 24, 30, 36. etc CLK pulses

for '1' value 13, 19, 25, 31, 37 CLK pulses

for '2' value 14, 20, 26, 32, 38 CLK pulses

Short cut

m CLK pulse for n mod counter

Simply: "'m' mod 'n'" [If it starts from '0']

eg: 12 mod 6 = 0

13 mod 6 = 1

14 mod 6 = 2

Q: for mod n counter [no. of ffs]

use: $2^k \geq n$ mod 6 ← n

$2^k \geq 6$

$2^1 \geq 6$ x $2^2 \geq 6$ x

$2^3 \geq 6$ ✓

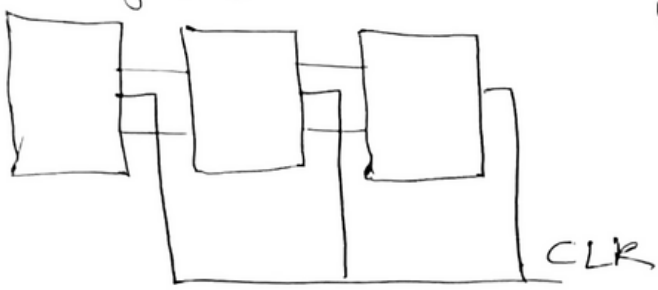
k = 3

eg:- 0 → 2 → 9 → 4 (so, consider the max bits for no. of ffs)

0, 2, 4: 3 bits but 9: 1001 (4 bits)

0, 10, 100

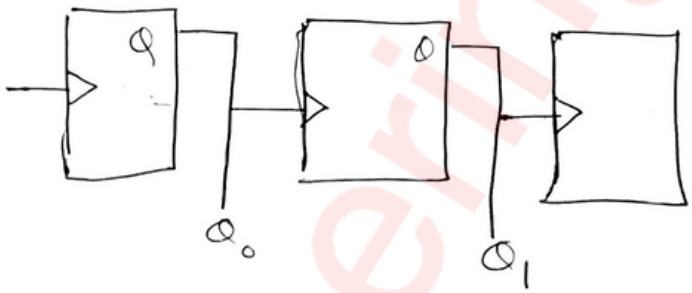
- Synchronous Design



- Parallel nature
- Random sequence ($0 \to 2 \to 4 \to 7$)
- faster
- But complex
- High cost
- delay factor ↓ (PD)
- Eg: Ring & Johnson counters

Vs

- Asynchronous Design

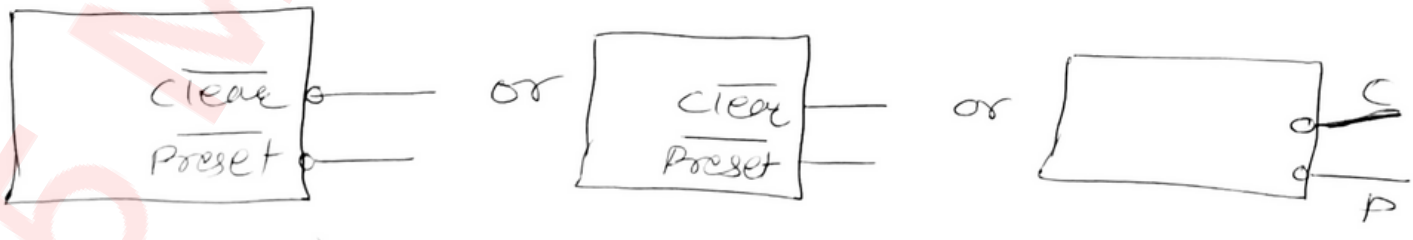


- Ripple nature / serial nature
- Up or down sequence ($0 \to 1 \to 2 \to 3$) or ($3 \to 2 \to 1 \to 0$)
- slower
- But simple & easy
- low cost
- delay factor ↑ (PD)
- Ripple counter.

⇒ Q (o/p of ff)

→ Present (set → 1)

→ clear (Reset → 0)

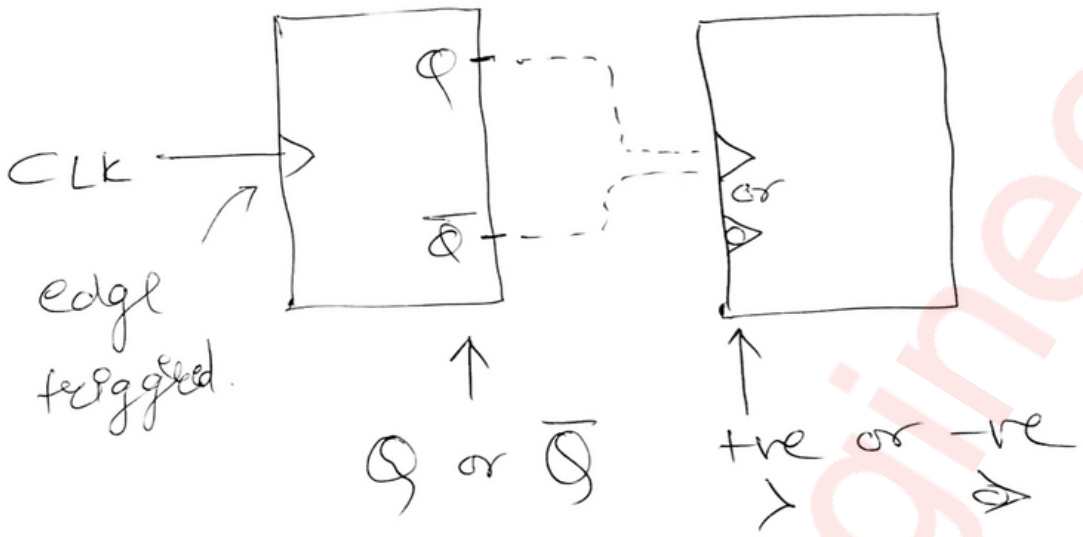


Low enabled.

- Up & down counters

  → (0 → 1 → 2 → 3)

  (3 → 2 → 1 → 0)



	+ve	-ve
+ve Q	Down	Up
-ve Q̄	Up	down

- Steps to design synchronous counter

(1) Let's say Given :- 0 → 3 → 2 → 1 (seq.)

(2) no. of ff's : max no = 3 (bits required = 2)
hence 2 ff's [can use any ff]

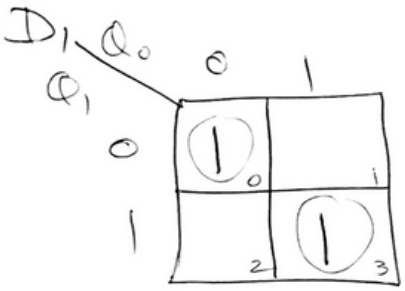
(3) Let's say we use 'D' ff.

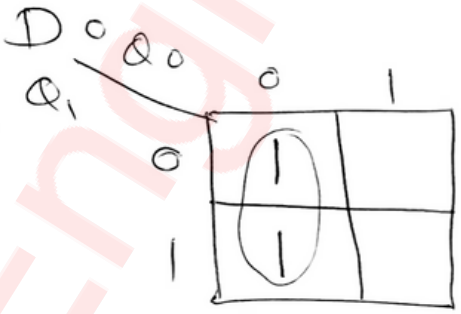
(4) will require excitation table of D ff

Q_n	Q_{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

⑤ Sequence (Present state & next state)

	PS		NS		↓ for D values refer (ET)	
	Q_1	Q_0	Q_1	Q_0	D_1	D_0
0)	0	0	3) 1	1	1	1
1)	0	1	0) 0	0	0	0
2)	1	0	1) 0	1	0	1
3)	1	1	2) 1	0	1	0

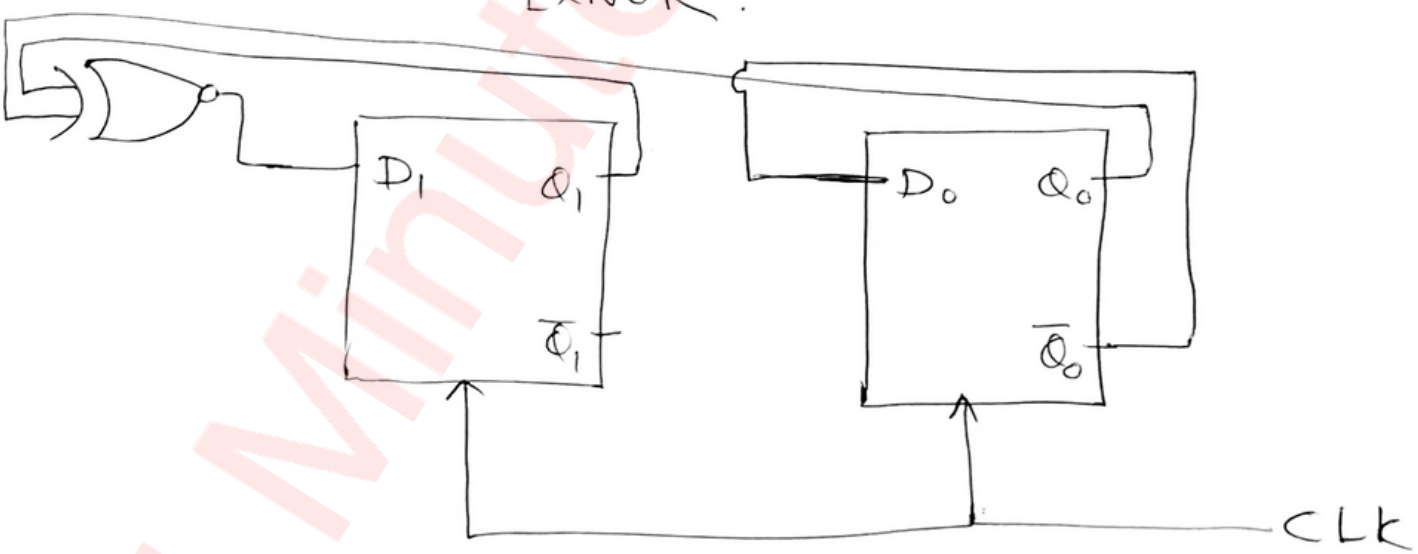




$D_1 = \overline{Q_1}\,\overline{Q_0} + Q_1 Q_0$ $\quad D_0 = \overline{Q_0}$

$= Q_1 \odot Q_0$

↑ EXNOR



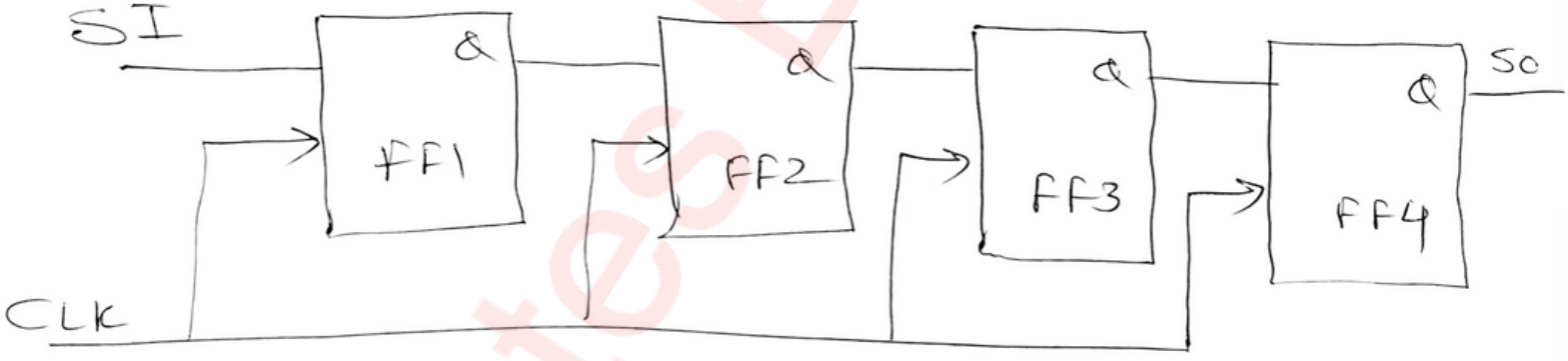
- **Registers**: Storage primary unit using ff.

(D - ff) acts just as buffer memory.

Modes [Shift Registers]

- Serial In Serial Out
- Serial In parallel out
- Parallel In parallel out
- Parallel In serial out

① **SISO**



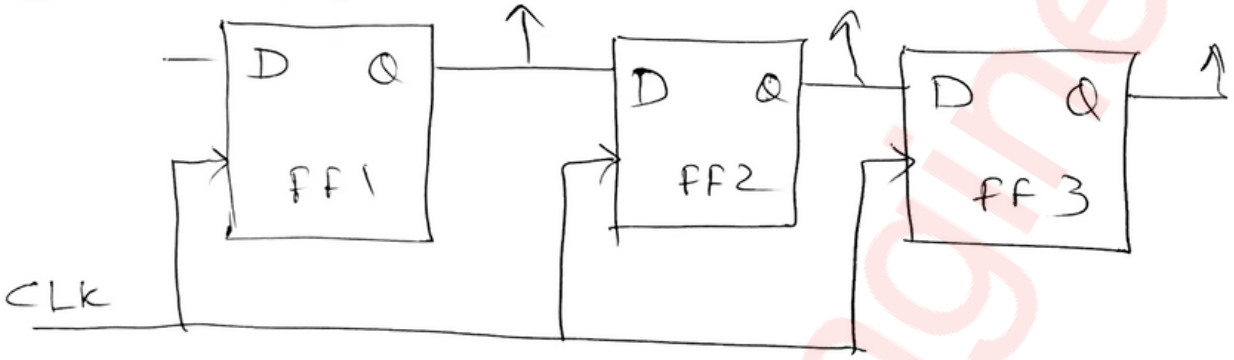
I/P	Q_3	Q_2	Q_1	Q_0	O/P
0	x	x	x	x	x
1	0	x	x	x	x
0	1	0	x	x	x
1	0	1	0	x	x
0	1	0	1	0	0
1	0	1	0	1	1

↑ clock

- No. of clocks to write : 4 for 4 ff
  i.e 'n' for n ff
- No. of clock to Read : 'n-1'

Total = n + n-1
= 2n-1

② SIPO

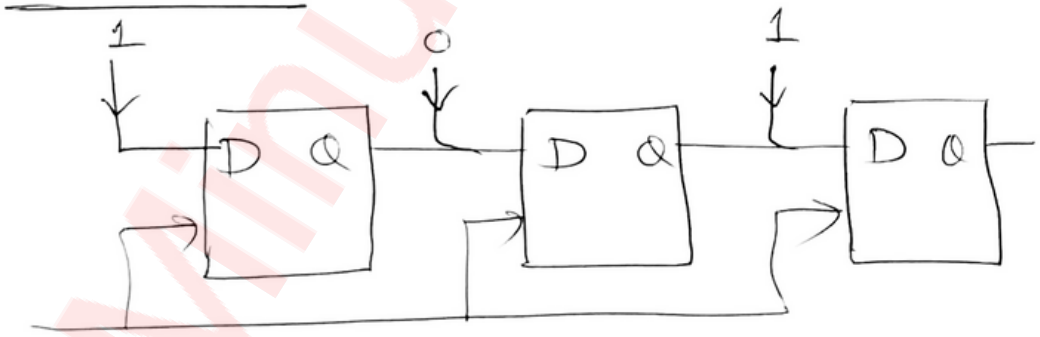


here I/p behaviour is same
so 'n' clock cycle.

But O/p is parallel so '0' CLK

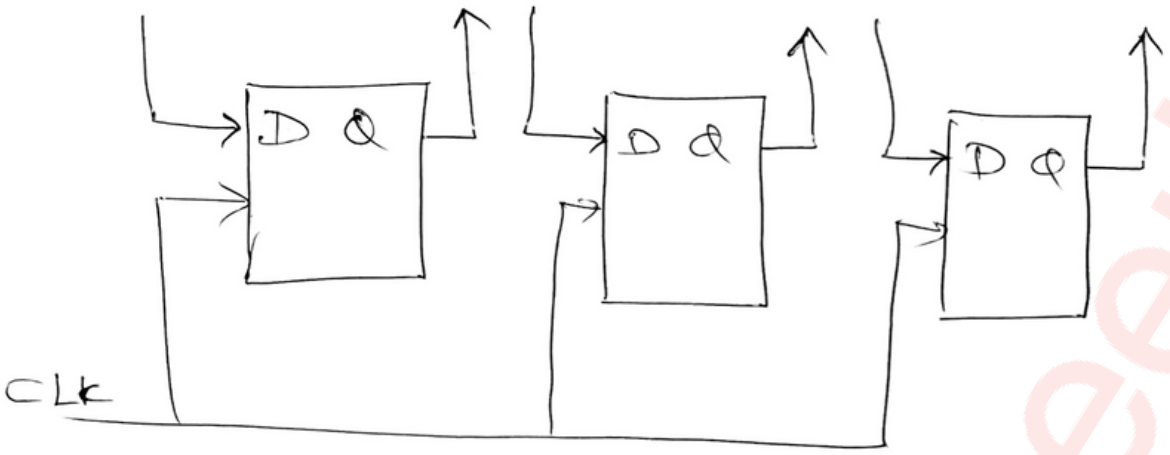
∴ Total = 'n'

③ PISO



I/p behaviour parallel → 1 clk
O/p behaviour serial → n-1 clk
Total = 'n'

④ PIPO



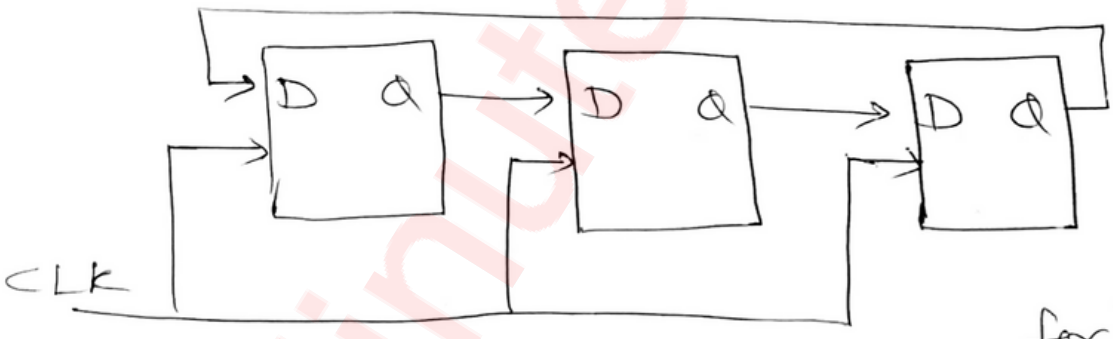
I/p & o/p both parallel behaviour

20 for write → 1 clk

— " — Read → 0 clk

Total 1 ✓

- Ring counter [circular shift Reg]



CLK	Q_2	Q_1	Q_0
0	1	0	0
1	0	1	0
2	0	0	1
0	1	0	0

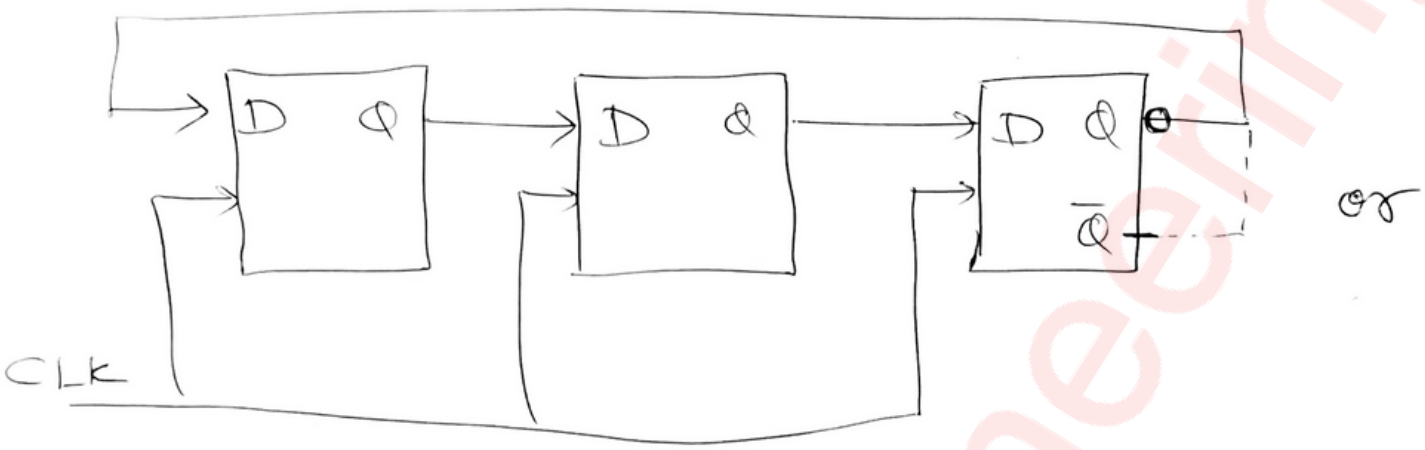
for 'n' no. of ff

→ 2^n (0 to $2^n - 1$)

→ But here we used only 'n' i.e 3 state

→ So $2^n - n$ unused state.

◦ Johnson Counter



CLK	Q_2	Q_1	Q_0
0	0	0	0
1	1	0	0
2	1	1	0
3	1	1	1
4	0	1	1
5	0	0	1
0	0	0	0

so here we get '6' states previously it was just '3' i.e 2(n)

So for n bits →
- Ideally (2^n count)
- Ring (n state count)
- Johnson (2n state count)

Google Cloud

All Readings:
Introduction to Generative AI

Kindly note that the 30 minutes indicated on the platform considers the time that it may take you to browse through the reading resources provided. The total time required depends on the readings you decide to explore further.

Assembled readings on generative AI:

- Ask a Techspert: What is generative AI?
  https://blog.google/inside-google/googlers/ask-a-techspert/what-is-generative-ai/
- What is generative AI?
  https://www.mckinsey.com/featured-insights/mckinsey-explainers/what-is-generative-ai
- Google Research, 2022 & beyond: Generative models:
  https://ai.googleblog.com/2023/01/google-research-2022-beyond-language.html#GenerativeModels
- Building the most open and innovative AI ecosystem:
  https://cloud.google.com/blog/products/ai-machine-learning/building-an-open-generative-ai-partner-ecosystem
- Generative AI is here. Who Should Control It?
  https://www.nytimes.com/2022/10/21/podcasts/hard-fork-generative-artificial-intelligence.html
- Stanford U & Google's Generative Agents Produce Believable Proxies of Human Behaviors:
  https://syncedreview.com/2023/04/12/stanford-u-googles-generative-agents-produce-believable-proxies-of-human-behaviours/
- Generative AI: Perspectives from Stanford HAI:
  https://hai.stanford.edu/sites/default/files/2023-03/Generative_AI_HAI_Perspectives
- Generative AI at Work:
  https://www.nber.org/system/files/working_papers/w31161/w31161.pdf
- The future of generative AI is niche, not generalized:
  https://www.technologyreview.com/2023/04/27/1072102/the-future-of-generative-ai-is-niche-not-generalized/

- The implications of Generative AI for businesses: https://www2.deloitte.com/us/en/pages/consulting/articles/generative-artificial-intelligence.html
- Proactive Risk Management in Generative AI: https://www2.deloitte.com/us/en/pages/consulting/articles/responsible-use-of-generative-ai.html
- How Generative AI Is Changing Creative Work: https://hbr.org/2022/11/how-generative-ai-is-changing-creative-work

Assembled readings on large language models:

- NLP's ImageNet moment has arrived: https://thegradient.pub/nlp-imagenet/
- LaMDA: our breakthrough conversation technology: https://blog.google/technology/ai/lamda/
- Language Models are Few-Shot Learners: https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf
- Introducing Gemini: our largest and most capable AI model: https://blog.google/technology/ai/google-gemini-ai/#sundar-note
- The Power of Scale for Parameter-Efficient Prompt Tuning: https://arxiv.org/pdf/2104.08691.pdf
- Google Research, 2022 & beyond: Language models: https://ai.googleblog.com/2023/01/google-research-2022-beyond-language.html#LanguageModels
- Solving a machine-learning mystery: https://news.mit.edu/2023/large-language-models-in-context-learning-0207

Additional Resources:

- Attention is All You Need: https://research.google/pubs/pub46201/
- Transformer: A Novel Neural Network Architecture for Language Understanding: https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html
- Transformer on Wikipedia:

https://en.wikipedia.org/wiki/Transformer_(machine_learning_model)#:~:text=Transformers%20were%20introduced%20in%202017,allowing%20training%20on%20larger%20datasets.

- What is Temperature in NLP? https://lukesalamone.github.io/posts/what-is-temperature/
- Model Garden: https://cloud.google.com/model-garden
- Auto-generated Summaries in Google Docs: https://ai.googleblog.com/2022/03/auto-generated-summaries-in-google-docs.html

URBAN EDGE

Python programming notes by Harry

What is Programming?

Just like we use Hindi or English to communicate with each other, we use a programming language like Python to communicate with the computer.
Programming is a way to instruct the computer to perform various tasks.

What is Python?

Python is a simple and easy to understand language which feels like reading simple english. This Pseudo Code nature of python makes it easy to learn and understandable by beginners

Features of Python

Easy to understand = Less development time
Free and open source
High level language
Portable → Works on Linux / Windows / Mac.
+ Fun to work with

Installation

Python can be easily installed from python.org
When you click on the download button, python can be installed right after you complete the setup by executing the file for your platform

URBAN EDGE

Chapter 1 : Modules, Comments & pip

Lets write our very first python program. Create a file called hello.py and paste the below code in it

```
print ("Hello World")
```
→ print is a function (More later)

Execute this file (.py file) by typing python hello.py and you will see Hello World printed on the screen.

Modules

A module is a file containing code written by somebody else (usually) which can be imported and used in our programs

Pip

Pip is the package manager for python
You can use pip to install a module on your system

↳ pip install flask installs flask module.

Types of modules

There are two types of modules in Python

1) Built in modules → Pre installed in python
2) External modules → Need to install using pip.

Some examples of built in modules are os, abc, etc.
Some examples of external modules are tensorflow, flask etc.

Using Python as a Calculator

We can use python as a calculator by typing "python" + ↵ on the terminal

↳ This opens REPL or Read Evaluate Print Loop

Comments

Comments are used to write something which the programmer does not want to execute.

↳ Can be used to mark author name, date etc.

Types of Comments

There are two types of comments in Python

1. Single line Comments → Written using #
2. Multi line Comments → Written using ''' Comment '''

URBAN EDGE

Chapter 1 - Practice Set

1. Write a program to print Twinkle twinkle little star poem in python.

2. Use REPL and print the table of 5 using it

3. Install an external module and use it to perform an operation of your interest

4. Write a python program to print the contents of a directory using os module. Search online for the function which does that.

5. Label the program written in Problem 4 with Comments.

URBAN EDGE

Chapter 2 - Variables and Datatypes

A variable is the name given to a memory location in a program. for example

```
a = 30
b = "Harry"
c = 71.22
```

→ Variables = Containers to store a value.
→ Keywords = Reserved words in Python
Identifiers = class/function/variable name

Data Types

Primarily there are following data types in Python

1. Integers
2. Floating point numbers
3. Strings
4. Booleans
5. None

Python is a fantastic language that automatically identifies the type of data for us.

```
a = 71          ⇒ Identifies a as class<int>
b = 88.44       ⇒ Identifies b as class <float>
name = "Harry"  ⇒ Identifies name as class <str>
```

Rules for defining a Variable name → Also applies to other Identifiers

→ A variable name can contain alphabets, digits and underscores.
→ A variable name can only start with an alphabet and underscore.
→ A variable name cant start with a digit
→ No white space is allowed to be used inside a variable name.

URBAN EDGE

Examples of a few variable names are :-
harry, one8, seven_, _seven etc.

Operators in Python
Following are some common operators in Python:
1. Arithmetic operators ⇒ +, -, *, / etc.
2. Assignment operators ⇒ =, +=, -= etc.
3. Comparison operators ⇒ ==, >, >=, <, != etc.
4. Logical operators ⇒ and, or, not

type() function and Typecasting
type function is used to find the data type of a given variable in Python.

```
a = 31
type(a) => class <int>

b = "31"
type(b) => class <str>
```

A number can be converted into a string and vice versa (if possible)
There are many functions to convert one data type into another

```
str(31) => "31"      => Integer to String Conversion
int("32") => 32      => String to Integer Conversion
float(32) => 32.0    => Integer to Float Conversion

... and so on
```

Here "31" is a string literal and 31 a numeric literal.

URBAN EDGE

input() function
This function allows the user to take input from the keyboard as a string

a = input("Enter name") ⇒ If a is "harry", the user entered harry

It is important to note that the output of input is always a string (even if the number is entered) ⇒ If a is "34" user entered 34

URBAN EDGE

Chapter 2 - Practice Set

1. Write a Python program to add two numbers

2. Write a Python program to find remainder when a number is divided by 2.

3. Check the type of the variable assigned using input() function.

4. Use Comparison operators to find out whether a given variable 'a' is greater than 'b' or not.
   Take a = 34 and b = 80

5. Write a python program to find average of two numbers entered by the user.

6. Write a python program to calculate square of a number entered by the user.

URBAN EDGE

Chapter 3 - Strings

String is a data type in Python.
String is a sequence of characters enclosed in quotes.
We can primarily, write a string in these three ways

1. Single quoted strings → a = 'Harry'
2. Double quoted strings → b = "Harry"
3. Triple quoted strings → c = '''Harry'''

String Slicing

A String in Python can be sliced for getting a part of the string.
Consider the following string:

name = "H a r r y" ⇒ length = 5

	H	a	r	r	y
	0	1	2	3	4
	(-5)	(-4)	(-3)	(-2)	(-1)

The index in a string starts from 0 to (length-1) in Python. In order to slice a string, we use the following syntax:

sl = name [ind_start : ind_end]
first index included — last index is not included

sl [0:3] returns "Har" → characters from 0 to 3
sl [1:3] returns "ar" → characters from 1 to 3

Negative Indices: Negative Indices can also be used as shown in the figure above. -1 corresponds to the (length-1) index, -2 to (length-2)

URBAN EDGE

Slicing with skip value
We can provide a skip value as a part of our slice like this :

```
word = "amazing"
word [1:6:2] → 'mzn'
```

Other advanced slicing techniques

```
word = "amazing"
word [:7] → word [0:7] → 'amazing'
word [0:] → word [0:7] → 'amazing'
```

String functions
Some of the mostly used functions to perform operations on or manipulate strings are :

1> len() function → This function returns the length of the string

len ("Harry") → returns 5

2> string.endswith ("xxy") → This function tells whether the variable string ends with the string "xxy" or not. if string is "Harry", it returns true for "xxy" since Harry ends with xxy

3> string.count ("c") → Counts the total number of occurence of any character

4> string.capitalize() → This function capitalizes the first character of a given string.

URBAN EDGE

5. String.find(word) - This function finds a word and returns the index of first occurence of that word in the string.

6. String.replace(oldword, newword) - This function replaces the oldword with newword in the entire string.

Escape Sequence Characters
Sequence of characters after backslash '\' → Escape Seq characters

Escape sequence character comprises of more than one characters but represents one character when used within the strings.

Examples \n, \t, \', \\ etc.
(\n → newline, \t → Tab, \' → Single quote, \\ → backslash.)

URBAN EDGE

Chapter 3 - Practice Set

1. Write a Python program to display a user entered name followed by Good Afternoon using input() function

2. Write a program to fill in a letter template given below with name and date.

```
letter = ''' Dear <|NAME|>,
                You are selected!
<|DATE|> '''
```

3. Write a program to detect double spaces in a string

4. Replace the double spaces from Problem 3 with single spaces.

5. Write a program to format the following letter using escape sequence characters.

letter = "Dear Harry, This Python course is nice. Thanks!"

Chapter 4 - Lists and Tuples

Python Lists are containers to store a set of values of any data type

```
friends = ["Apple", "Akash", "Rohan", 7, False]
```

"Apple" → str()
"Akash" → can store value of any datatype
7 → int()
False → bool()

List Indexing

A List can be indexed just like a String

```
L1 = [7, 9, "Harry"]
```

```
L1[0]  => 7
L1[1]  => 9
L1[70] => Error
L1[0:2] => [7, 9]   => List Slicing
```

List Methods

Consider the following list:

```
L1 = [1, 8, 7, 2, 21, 15]
```

1. L1.sort(): updates the list to [1, 2, 7, 8, 15, 21]
2. L1.reverse(): updates the list to [15, 21, 2, 7, 8, 1]
3. L1.append(8): adds 8 at the end of the list
4. L1.insert(3,8): This will add 8 at 3 index

URBAN EDGE

5> L1.pop(2): Will delete element at index 2 and return its value

6> L1.remove(21): Will remove 21 from the list.

Tuples in Python

A tuple is an immutable data type in python.
↳ Cannot change

a = ( ) => Empty tuple
a = (1,) => Tuple with only one element needs a comma
a = (1, 7, 2) => Tuple with more than one element

Once defined a tuple's elements cant be altered or manipulated.

Tuple methods

Consider the following tuple
a = (1, 7, 2)

1> a.count(1): a.count(1) will return number of times 1 occurs in a.

2> a.index(1): a.index(1) will return the index of first occurence of 1 in a.

URBAN EDGE

Chapter 4 - Practice Set

1. Write a program to store seven fruits in a list entered by the user.

2. Write a program to accept marks of 6 students and display them in a sorted manner

3. Check that a tuple cannot be changed in python

4. Write a program to sum a list with 4 numbers.

5. Write a program to count the number of zeros in the following tuple:

$$a = (7, 0, 8, 0, 0, 9)$$

Chapter 5 : Dictionary & Sets

Dictionary is a collection of key-value pairs

Syntax :

```
a = { "key" : "value",
      "harry" : "code",
      "marks" : "100",
      "list" : [1,2,9] }
```

(keys → "key", "harry", "marks", "list"; values → "value", "code", "100", [1,2,9])

a["key"] => Prints "value"
a["list"] => Prints [1,2,9]

Properties of a Python Dictionaries

1. It is unordered
2. It is mutable
3. It is indexed
4. Cannot contain duplicate keys

Dictionary Methodes

Consider the following dictionary

```
a = { "name" : "Harry",
      "from" : "India",
      "marks" : [92,98,96] }
```

1. a.items() : returns a list of (key, value) tuples
2. a.keys() : returns a list containing dictionary's keys
3. a.update({"friend" : "Sam"}) : updates the dictionary with supplied key-value pairs

URBAN EDGE

4. a.get("name"): returns the value of the specified keys (and value is returned eg. "Harry" is returned here)

More methods are available on docs.python.org.

Sets in Python

Set is a collection of non repetitive elements

```
S = Set()                    ⇒ No repetition allowed!
S.add(1)
S.add(2)                     ⇒ or Set = {1, 2}
```

If you are a programming beginner without much knowledge of mathematical operations on sets, you can simply look at sets in python as data types containing unique values.

Properties of Sets

1. Sets are unordered ⇒ Element's order doesn't matter
2. Sets are unindexed ⇒ Cannot access elements by index
3. There is no way to change items in sets.
4. Sets cannot contain duplicate values

Operations on Sets

Consider the following Set:

S = {1, 8, 2, 3}

1. len(s): Returns 4, the length of the set
2. S.remove(8): Updates the set S and removes 8 from S.

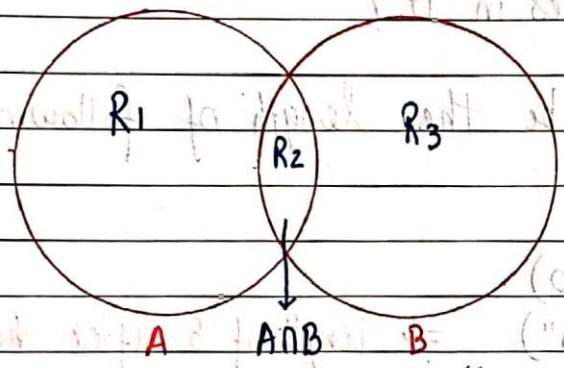
URBAN EDGE

3. S.pop() : Removes an arbitrary element from the set and returns the element removed

4. S.clear() : Empties the set S

5. S.union({8,11}) : Returns a new set with all items from both sets. => {1,8,2,3,11}

6. S.intersection({8,11}) : Returns a set which contains only items in both sets. => {8}



$R_2 \Rightarrow A \cap B$

$R_1 + R_2 + R_3 \Rightarrow A \cup B$

$R_1 + R_3 \Rightarrow A \Delta B$

$R_1 \Rightarrow A - B$

$R_3 \Rightarrow B - A$

URBAN EDGE

Chapter 5: Practice Set

1. Write a program to create a dictionary of Hindi words with values as their english translation. Provide user with an option to look it up!

2. Write a program to input eight numbers from the user and display all the unique numbers (once).

3. Can we have a set with 18 (int) and "18" (str) as a values in it?

4. What will be the length of following set S:
```
S = Set()
S.add(20)
S.add(20.0)
S.add("20")
```
=> Length of S after these operations?

5. S = {}
what is the type of S?

6. Create an empty dictionary. Allow 4 friends to enter their favourite language as values and use keys as their names. Assume that the names are unique

7. If names of 2 friends are same; what will to the program in Problem 6?

8. If languages of two friends are same; what will happen to the program in Problem 6?

9 Can you change the values inside a list which is contained in Set S

$S = \{8, 7, 12, \text{"Harry"}, [1, 2]\}$

Chapter 6 - Conditional Expressions

Sometimes we want to play pubG on our phone if the day is Sunday.
Sometimes we order Icecream online if the day is Sunny.
Sometimes we go hiking if our parents allow.

All these are decisions which depends on a condition being met.
In Python programming too, we must be able to execute instructions on a condition(s) being met. This is what conditionals are for!

If else and elif in Python

If else and elif statements are a multiway decision taken by our program due to certain conditions in our code.

Syntax:

```
if (condition1):                    => if condition 1 is true
    print ("yes")
elif (condition2):                  => if condition 2 is true
    print ("No")
else:                               => otherwise
    print ("May be")
```

Indentation ←

Code example:

```
a = 22
if(a>9):
    print("Greater")
else:
    print("lesser")
```

Quick Quiz: Write a program to print yes when the age entered by the user is greater than or equal to 18.

Relational Operators

Relational operators are used to evaluate conditions inside the if statements. Some examples of relational operators are:

- == → equals
- >= → greater than/equal to
- <=, etc.

Logical operators

In python logical operators operate on conditional statements. Example:

- and → true if both operands are true else false
- or → true if at least one operand is true else false
- not → inverts true to false & false to true

elif clause

elif in python means [else if]. An if statement can be chained together with a lot of these elif statements followed by an else statement

```
if (Condition1):
    # code
elif (condition 2):
    # code
elif (condition 3):
    # code
:
:
else:
    # code
```

⇒ This ladder will stop once a condition in an if or elif is met.

Important notes:

1. There can be any number of elif statements.
2. last else is executed only if all the conditions inside elifs fail.

URBAN EDGE

Chapter 6 - Practice Set

1. Write a program to find greatest of four numbers entered by the user.

2. Write a program to find out whether a student is pass or fail, if it requires total 40% and at least 33% in each subject to pass. Assume 3 subjects and take marks as an input from the user.

3. A spam comment is defined as a text containing following keywords:
"make a lot of money", "buy now", "subscribe this", "click this". Write a program to detect these spams.

4. Write a program to find whether a given username contains less than 10 characters or not.

5. Write a program which finds out whether a given name is present in a list or not.

6. Write a program to calculate the grade of a student from his marks from the following scheme:

90 - 100	→	Ex
80 - 90	→	A
70 - 80	→	B
60 - 70	→	C
50 - 60	-	D
<50	-	F

URBAN EDGE

7. Write a program to find out whether a given post is talking about "Harry" or not.

URBAN EDGE

Chapter 7 - Loops in Python

Sometimes we want to repeat a set of statements in our program. For instance: Print 1 to 1000

Loops make it easy for a programmer to tell the computer, which set of instructions to repeat and how!

Types of loops in Python

Primarily there are two types of loops in Python

1. While loop
2. For loop

We will look into these one by one!

While loop

The syntax of a while loop looks like this:

```
While Condition:
    #Body of the loop
```

⇒ The block keeps executing until the condition is true

In while loops, the condition is checked first. If it evaluates to true, the Body of the loop is executed, otherwise not!
If the loop is entered, the process of [Condition check & execution] is continued until the condition becomes false.

Quick Quiz: Write a program to print 1 to 50 using a while loop.

URBAN EDGE

An Example

```
i = 0
while i < 5:
    print("Harry")
    i = i + 1
```

⇒ Prints "Harry" - 5 times!

Note: If the condition never becomes False, the loop keeps getting executed.

Quick Quiz: Write a program to print the content of a list using while loops.

For loop

A for loop is used to iterate through a sequence like list, tuple or string [Iterables]

The syntax of a for loop looks like this:

```
l = [1, 7, 8]
for item in l:
    print(item)
```

→ print 1, 7 and 8

Range function in Python

The range function in python is used to generate a sequence of numbers.
We can also specify the start, stop and step-size as follows:

```
range(start, stop, step_size)
```

↳ step size is usually not used with range()

An Example demonstrating range() function

```
for i in range(0, 7):
    print(i)
```

→ range(7) can also be used
→ prints 0 to 6

For loop with else

An optional else can be used with a for loop if the code is to be executed when the loop exhausts

Example :

```
l = [1, 7, 8]
for item in l:
    print(item)
else:
    print("Done")
```

→ This is printed when the loop exhausts!

Output :

```
1
7
8
Done
```

The break statement

'break' is used to come out of the loop when encountered
It instructs the program to - Exit the loop now

Example :

```
for i in range(0,80):
    print(i)
    if i==3:
        break
```

→ This will print 0, 1, 2 and 3

URBAN EDGE

The Continue Statement

'Continue' is used to stop the current iteration of the loop and continue with the next one. It instructs the program to "skip this iteration"

Example:

```
for i in range(4):
    print("printing")
    if i == 2:
        continue
    print(i)
```

⇒ if i is 2, the iteration is skipped.

pass statement

pass is a null statement in python. It instructs to "Do nothing"

Example:

```
l = [1, 7, 8]
for item in l:
    pass
```

→ Without pass, the program will throw an error

URBAN EDGE

Chapter 7 - Practice Set

1. Write a program to print multiplication table of a given number using for loop.

2. Write a program to greet all the person names stored in a list l1 and which starts with S

$l_1 = [$"Harry", "Soham", "Sachin", "Rahul"$]$

3. Attempt problem 1 using while loop.

4. Write a program to find whether a given number is prime or not

5. Write a program to find the Sum of first n natural numbers using while loop.

6. Write a program to calculate the factorial of a given number using for loop.

7. Write a program to print the following star pattern

```
    *
  * * *
* * * * *
```
for n = 3

8. Write a program to print the following star pattern:

```
*
* *
* * *
```
for n = 3

URBAN EDGE

9 Write a program to print the following star pattern

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

for $n = 3$

10 Write a program to print multiplication table of n using for loop in reversed order.

URBAN EDGE

Chapter 8 - Functions & Recursions

A function is a group of statements performing a specific task.

When a program gets bigger in size and its complexity grows, it gets difficult for a programmer to keep track on which piece of code is doing what!

A function can be reused by the programmer in a given program any number of

Example and Syntax of a function
The syntax of a function looks as follows:

```
def func1():
    print("Hello")
```

This function can be called any number of times, anywhere in the program.

Function Call
Whenever we want to call a function, we put the name of the function followed by parenthesis as follows:

```
func1()
```

→ This is called function call

function definition
The part containing the exact set of instructions which are executed during the function call.

Quick Quiz: Write a program to greet a user with "Good day" using functions.

Types of functions in Python
There are two types of functions in Python:
1. Built in functions → Already present in Python
2. User defined functions → Defined by the user

Examples of built in function includes len(), print(), range() etc.

The funcI() function we defined is an example of user defined function

Functions with arguments
A function can accept some values it can work with. We can put these values in the parentheses
A function can also return values as shown below:

```
def greet(name):
    gr = "Hello " + name
    return gr

a = greet("Harry")
```

→ "Harry" is passed to greet in name

↳ a will now contain "Hello Harry"

Default Parameter Value
We can have a value as default argument in a function.
If we specify name = 'stranger' in the line containing def, this value is used when no argument is passed.

URBAN EDGE

For example:

```
def greet(name = "Stranger"):
    # function body
```

greet() → Name will be "Stranger" in function body (default)
greet("Harry") → Name will be "Harry" in function body (passed)

Recursion

Recursion is a function which calls itself
It is used to directly use a mathematical formula as a function. For example:

factorial(n) = n × factorial(n-1)

This function can be defined as follows:

```
def factorial(n):
    if i==0 or i==1: → Base condition which doesn't call the function any further
        return 1
    else:
        return n * factorial(n-1) → function calling itself
```

This works as follows:

Factorial(4)
↓ ↳ [Function called]
4 × factorial(3)
4 × [3 ↓ × factorial(2)]
4 × 3 × [2 ↓ × factorial(1)]
4 × 3 × 2 × [1] [Function returned]

The programmer need to be extremely careful while working with recursion to ensure that the function doesnt Infinitely keep calling itself.
Recursion is sometimes the most direct way to code an algorithm

Chapter 8 - Practice Set

1. Write a program using function to find greatest of three numbers

2. Write a python program using function to convert Celsius to fahrenheit.

3. How do you prevent a python print() function to print a new line at the end.

4. Write a recursive function to calculate the sum of first n natural numbers.

5. Write a python function to print first n lines of the following pattern:

```
* * *
* *        → For n = 3
*
```

6. Write a python function which converts inches to cms

7. Write a python function to remove a given word from a list and strip it at the same time.

8. Write a python function to print multiplication table of a given number.

URBAN EDGE

Project 1: Snake, Water, Gun Game

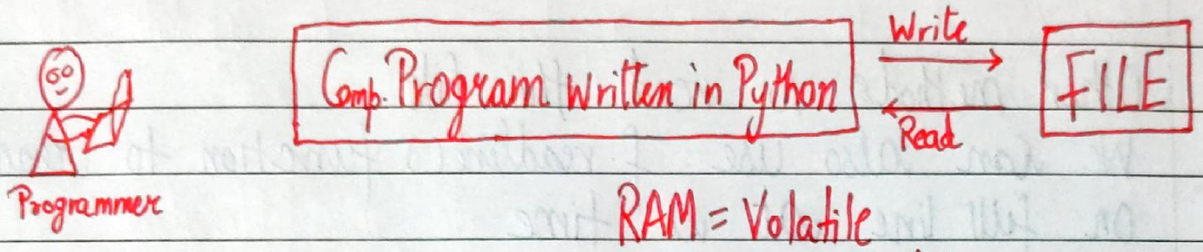
We all have played snake, water gun game in our childhood. If you haven't, google the rules of this game and write a Python program capable of playing this game with the user.

URBAN EDGE

Chapter 9 - File I/O

The random Access memory is volatile and all its contents are lost once a program terminates
In order to persist the data forever, we use files

A file is data stored in a storage device. A Python program can talk to the File by reading content from it and writing content to it



RAM = Volatile
HDD = Non Volatile

Types of Files

There are 2 types of files:

1. Text files (.txt, .c etc)
2. Binary files (.jpg, .dat, etc)

Python has a lot of functions for reading, updating and deleting files.

Opening a file

Python has an open() function for opening files. It takes 2 parameters: filename and mode.

```
open("this.txt", "r")
```

- open is a built-in function
- "this.txt" → Filename
- "r" → mode of opening (read mode)

URBAN EDGE

Reading a file in python

```
f = open("this.txt", "r")  → open the file in r mode
text = f.read()            → Read its contents
print(text)                → Print its contents
f.close()                  → Close the file
```

We can also specify the number of characters in read() function : f.read(2)
↳ Reads first 2 characters

Other methods to read the file
We can also use f.readline() function to read on full line at a time

f.readline() → Reads one line from the file

Modes of opening a file

r → open for reading
w → open for writing
a → open for appending
+ → open for updating

'rb' will open for read in binary mode
'rt' will open for read in text mode

Writing files in Python
In order to write to a file, we first open it in write or append mode after which, we use the python's f.write() method to write to the file!

```
f = open("this.txt", "w")
f.write("This is nice")
f.close()
```

→ Can be called multiple times

With statement

The best way to open and close the file automatically is the with statement

```
with open("this.txt") as f:
    f.read()
```

↳ Dont need to write f.close() as it is done automatically.

URBAN EDGE

Chapter 9 - Practice Set

1. Write a program to read the text from a given file 'poems.txt' and find out whether it contains the word 'twinkle'.

2. The game() function in a program lets a user play a game and returns the score as an integer. You need to read a file 'History.txt' which is either blank or contains the previous Hi-score. You need to write a program to update the Hi-Score whenever game() breaks the Hi-Score.

3. Write a program to generate multiplication tables from 2 to 20 and write it to the different files. Place these files in a folder for a 13-year old.

4. A file contains a word "Donkey" multiple times. You need to write a program which replaces this word with ###### by updating the same file.

5. Repeat program 4 for a list of such words to be censored.

6. Write a program to mine a log file and find out whether it contains 'python'.

7. Write a program to find out the line number where python is present from Ques 6

URBAN EDGE

8 Write a program to make a copy of a text file "this.txt"

9 Write a program to find out whether a file is identical & matches the content of another file.

10 Write a program to wipe out the contents of a file using python

11 Write a python program to rename a file to "renamed_by_python.txt"

URBAN EDGE

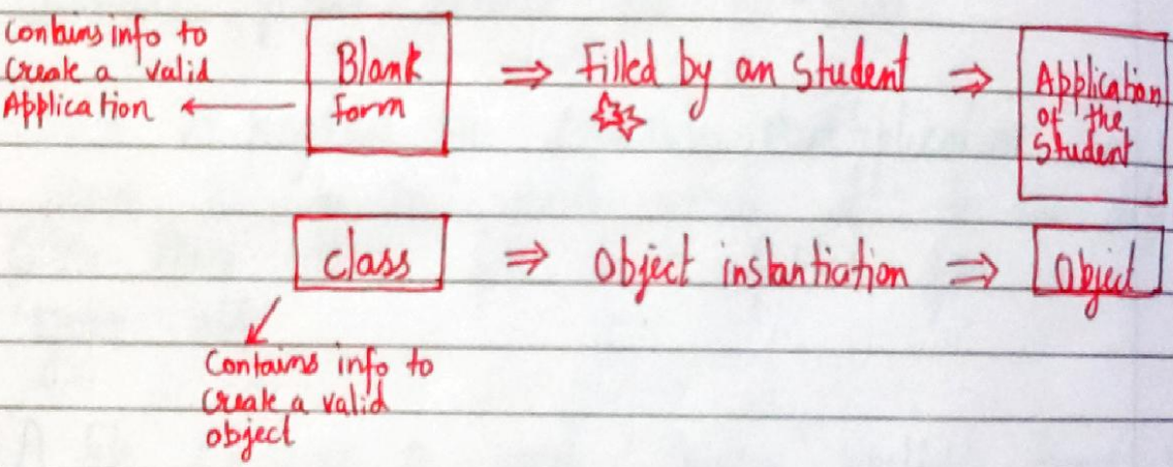
Chapter 10 - Object Oriented Programming

Solving a problem by creating objects is one of the most popular approaches in programming. This is called Object oriented programming.

This concept focuses on using reusable code.
↳ Implements DRY principle

Class

A class is a blueprint for creating objects.



The syntax of a class looks like this:

```
Class Employee:
    # methods & variables
```

[Classname is written in PascalCase]

Object

An Object is an instantiation of a class. When class is defined, a template (info) is defined. Memory is allocated only after object instantiation.

Objects of a given class can invoke the methods available to it without revealing the implementation details to the user. → Abstraction & Encapsulation!

URBAN EDGE

Modelling a problem in OOPs

We identify the following in our problem

Noun → Class → Employee
Adjective → Attributes → name, age, salary
Verbs → Methods → getSalary(), increment()

Class Attributes

An attribute that belongs to the class rather than a particular object.

Example :

```
Class Employee :
    company = "Google"      → [Specific to each class]

harry = Employee()          → object instantiation
harry.company
Employee.company = "YouTube" → changing class attribute
```

Instance Attributes

An attribute that belongs to the Instance (object)
Assuming the class from the previous example :

```
harry.name = "Harry"
harry.salary = "30K"        ⇒ Adding instance attributes
```

Note : Instance attributes take preference over class attributes during assignment & retrieval

harry.attribute1 → ① Is attribute1 present in object ?
② Is attribute1 present in class ?

'self' parameter
self refers to the instance of the class
It is automatically passed with a function call from an object

```
harry.getSalary()
```
→ here self is harry
↳ equivalent to Employee.getSalary(harry)

The function getsalary is defined as:

```
class Employee:
    Company = "Google"
    def getSalary(self):
        print("Salary is not there")
```

Static method
Sometimes we need a function that doesnt use the self parameter. we can define a static method like this:

```
@staticmethod
def greet():
    print("Hello user")
```
→ decorator to mark greet as a static method

__init__() Constructor
__init__() is a special method which is first run as soon as the object is created.
__init__() method is also known as constructor

It takes self argument and can also take further arguments

URBAN EDGE ___/___/___

For Example:

```
Class Employee:
    def __init__(self, name):
        self.name = name

    def getSalary(self):
        ...

harry = Employee("Harry")
```

↳ Object can be instantiated using constructor like this!

URBAN EDGE

Chapter 10 - Practice Set

1. Create a class programmer for storing information of few programmers working at microsoft.

2. Write a class Calculator capable of finding square, cube and squareroot of a number.

3. Create a class with a class attribute a; create an object from it and set a directly using object.a = 0. Does this change the class attribute?

4. Add a static method in problem 2 to greet the user with hello.

5. Write a class Train which has methods to book a ticket, get status (no of seats) and get fare information of trains running under Indian Railways.

6. Can you change the self parameter inside a class to something else (say 'harry'). Try changing self to 'slf' or 'harry' and see the effects.

URBAN EDGE

Chapter 11 - Inheritance & more on OOPs

Inheritance is a way of creating a new class from an existing class

Syntax:

```
Class Employee:                    → Base Class
    # Code
    ...

Class Programmer (Employee):       → Derived or child class
    # Code
```

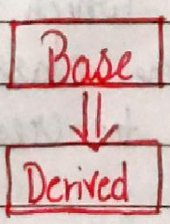
We can use the methods and attributes of Employee in Programmer object.
Also, we can overwrite or add new attributes and methods in Programmer class.

Types of Inheritance

1. Single inheritance
2. Multiple inheritance
3. Multilevel inheritance

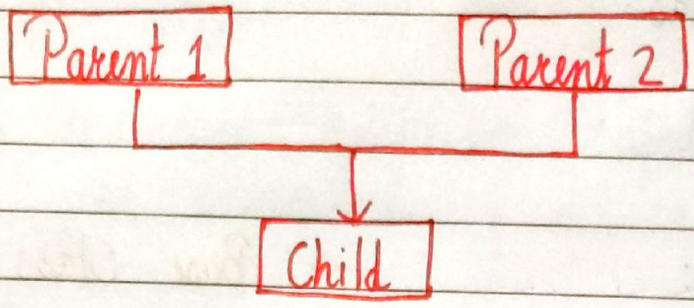
Single Inheritance

Single inheritance occurs when child class inherits only a single parent class



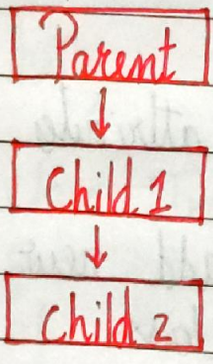
Multiple Inheritance

Multiple inheritance occurs when the child class inherits from more than one parent class.



Multilevel Inheritance

When a child class becomes a parent for another child class



Super() method

Super method is used to access the methods of a super class in the derived class

```
super().__init__()
```

↳ Calls constructor of the base class

Class methods

A class method is a method which is bound to the class and not the object of the class.

@classmethod decorator is used to create a class method

Syntax to create a class method:

```
@classmethod
def (cls, p1, p2):
    ...
```

@property decorators
Consider the following class

```
class Employee:
    @property
    def name(self):
        return self.ename
```

if e = Employee() is an object of class employee, we can print(e.name) to print the ename/call name() function.

@.getters and @.setters
The method name with @property decorator is called getter method
We can define a function + @name.setter decorator like below:

```
@name.setter
def name(self, value):
    self.ename = value
```

Operator overloading in Python
Operators in python can be overloaded using dunder methods.
These methods are called when a given operator is used on the objects.

Operators in python can be overloaded using the following methods:

$p_1 + p_2 \rightarrow p_1.\_\_add\_\_(p_2)$

$p_1 - p_2 \rightarrow p_1.\_\_sub\_\_(p_2)$

$p_1 * p_2 \rightarrow p_1.\_\_mul\_\_(p_2)$

$p_1 / p_2 \rightarrow p_1.\_\_truediv\_\_(p_2)$

$p_1 // p_2 \rightarrow p_1.\_\_floordiv\_\_(p_2)$

Other dunder/magic methods in python

`__str__()` → used to set what gets displayed upon calling `str(obj)`

`__len__()` → used to set what gets displayed upon calling `.__len__()` or `len(obj)`

URBAN EDGE

Chapter 11 - Practice Set

1. Create a class C2dVector and use it to create another class representing a 3-d vector.

2. Create a class pets from a class Animals and further create class Dog from Pets. Add a method bark to class Dog.

3. Create a class Employee and add salary and increment properties to it.
Write a method salaryAfterIncrement method with a @property decorator with a setter which changes the value of increment based on the salary.

4. Write a class Complex to represent complex numbers, along with overloaded operators + and * which adds and multiplies them

5. Write a class vector representing a vector of n dimension. Overload the + and * operator which calculates the sum and the dot product of them.

6. Write __str__() method to print the vector as follows:

$$7\hat{i} + 8\hat{j} + 10\hat{k}$$

Assume vector of dimension 3 for this problem.

URBAN EDGE

7 Override the __len__() method on Vector of problem 5 to display the dimension of the Vector.

URBAN EDGE

Project 2 - The Perfect Guess

We are going to write a program that generates a random number and asks the user to guess it.
If the player's guess is higher than the actual number, the program displays "Lower number please".
Similarly if the user's guess is too low, the program prints "higher number please"
When the user guesses the correct number, the program displays the number of guesses the player used to arrive at the number.

Hint: Use the random module

Chapter 12 - Advanced Python 1

Exception Handling in Python

There are many built-in exceptions which are raised in Python when something goes wrong.
Exceptions in Python can be handled using a try statement. The code that handles the exception is written in the except clause.

```
try:
    # Code                          → Code which might throw Exception
except Exception as e:
    print(e)
```

When the exception is handled, the code flow continues without program interruption.

We can also specify the exceptions to catch like below:

```
try:
    #Code
except ZeroDivisionError:
    # Code
except TypeError:
    # code
except:
    # code          → All other exceptions are handled here.
```

Raising Exceptions

We can raise custom exceptions using the raise keyword in python.

try with else clause
Sometimes we want to run a piece of code when try was successful.

```
try:
    # Some code
except:
    # Some code
else:
    # Code        → This is executed only if the try was successful
```

try with finally
Python offers a finally clause which ensures execution of a piece of code irrespective of the exception.

```
try:
    # Some code
except:
    # Some code
finally:
    # Some code    → executed regardless of error!
```

if __name__ == '__main__' in Python
__name__ evaluates to the name of the module in Python from where the program is ran

If the module is being run directly from the command line, the __name__ is set to string "__main__"
Thus this behaviour is used to check whether the module is run directly or imported to another file.

URBAN EDGE

The global keyword

global keyword is used to modify the variable outside of the current scope.

Enumerate function in Python

The enumerate function adds counter to an iterable and returns it

```
for i, item in list1:
    print(i, item)
```

↳ Prints the items of list 1 with index!

List Comprehensions

list comprehension is an elegant way to create lists based on existing lists

```
list1 = [1, 7, 12, 11, 22]

list2 = [i for item in list1 if item > 8]
```

URBAN EDGE

Chapter 12 - Practice Set

1. Write a program to open three files 1.txt, 2.txt and 3.txt. If any of these files are not present, a message without exiting the program must be printed prompting the same.

2. Write a program to print third, fifth and seventh element from a list using enumerate function

3. Write a list comprehension to print a list which contains the multiplication table of a user entered number.

4. Write a program to display a/b where a and b are integers. If b=0, display Infinite by handling the ZeroDivisionError.

5. Store the multiplication tables generated in Problem 3 in a file named Tables.txt.

URBAN EDGE

Chapter 13 - Advanced Python 2

Virtual Environment

An environment which is same as the System interpretor but is isolated from the other python environments on the system.

Installation

To use virtual environments, we write

```
pip install virtualenv → Install the package
```

We create a new environment using:

```
virtualenv myprojectenv → Creates a new venv
```

The next step after creating the virtual environment is to activate it.
We can now use this virtual environment as a separate python installation.

pip freeze command

pip freeze returns all the packages installed in a given python environment along with the versions

```
" pip freeze > requirements.txt "
```

The above command creats a file named requirements.txt in the same directory containing the output of pip freeze.
We can distribute this file to other users and they can recreate the same environment using:

URBAN EDGE

```
pip install -r requirements.txt
```

Lambda functions

functions created using an expression using lambda keyword

Syntax:

```
lambda arguments : expressions
```

↳ can be used as a normal function

Example:

```
Square = lambda x : x*x
Square(6)   → returns 36

Sum = lambda a,b,c : a+b+c
Sum(1,2,3)  → returns 6
```

.join method (strings)

creates a string from iterable objects

```
l = ["apple", "mango", "banana"]

",and,".join(l)
```

The above line will return "apple,and,mango,and,banana"

format method (strings)

formats the values inside the string into a desired output

```
template.format(p1, p2 ...)
```

↳ arguments

Syntax for format looks like:

"{} is a good {}".format("Harry", "boy") —①

"{1} is a good {0}".format("Harry", "boy") —②

Output for ①
Harry is a good boy

Output for ②
boy is a good Harry

Map, Filter & Reduce
Map applies a function to all the items in an input_list

Syntax:

map(function, input_list)
↳ Can be lambda function

Filter creates a list of items for which the function returns true.

list(filter(function))
↳ Can be a lambda function

Reduce applies a rolling computation to sequential pair of elements

```
from functools import reduce
val = reduce(function, list1)
```
↳ Can be a lambda function

If the function computes sum of two numbers and the

list is [1, 2, 3, 4]

1 2 3 4

3 3 4

6 4

10

=> Sequential Computation

URBAN EDGE

Chapter 13 - Practice Set

1. Create two virtual environments, install few packages in the first one. How do you create a similar environment in the second one?

2. Write a program to input name, marks and phone number of a student and format it using the format function like below:

"The name of the student is Harry, his marks are 72 and phone number is 99999888"

3. A list contains the multiplication table of 7. Write a program to convert it to a vertical string of same numbers (7 / 14 / ⋮)

4. Write a program to filter a list of numbers which are divisible by 5

5. Write a program to find the maximum of the numbers in a list using the reduce function.

6. Run pip freeze for the system interpreter. Take the contents and create a similar virtualenv.

7. Explore the Flask module and create a web server using Flask & Python.

URBAN EDGE

Project 3 - Student Library

Implement a student library system using OOPs where students can borrow a book from the list of books.
Create a separate Library and Student class.
Your program must be menu driven.
You are free to choose methods and attributes of your choice to implement this functionality.

THE ULTIMATE PYTHON HANDBOOK

CodeWithHarry

PREFACE

Welcome to the “Ultimate Python Programming Handbook," your comprehensive guide to mastering Python programming. This handbook is designed for beginners and anyone looking to strengthen their foundational knowledge of Python, a versatile and user-friendly programming language.

PURPOSE AND AUDIENCE

This handbook aims to make programming accessible and enjoyable for everyone. Whether you're a student new to coding, a professional seeking to enhance your skills, or an enthusiast exploring Python, this handbook will definitely be helpful. Python's simplicity and readability make it an ideal starting point for anyone interested in programming.

STRUCTURE AND CONTENT

The handbook is divided into clear, concise chapters, each focused on a specific aspect of Python:

- Fundamental Concepts: Start with the basics, such as installing Python and writing your first program.
- Practical Examples: Illustrative examples and sample code demonstrate the application of concepts.
- Hands-On Exercises: End-of-chapter exercises reinforce learning and build confidence.
- Additional Resources: References to official Python documentation for deeper exploration.

WHY PYTHON?

Python is known for its simplicity and readability, making it perfect for beginners. It is a high-level, interpreted language with a broad range of libraries and frameworks, supporting applications in web development, data analysis, AI, and more. Python's versatility and ease of use make it a valuable tool for both novice and experienced programmers.

ACKNOWLEDGEMENTS

I extend my gratitude to the educators, programmers, and contributors who have shared their knowledge and insights, shaping the content of this handbook. Special thanks to all the students watching my content on YouTube and Python community for maintaining a supportive and inspiring environment for learners worldwide.

CONCLUSION

Learning programming can be both exciting and challenging. The “Ultimate Python Programming Handbook" aims to make your journey smooth and rewarding. Watch my video along with following this handbook for optimal learning. Let this guide be your stepping stone to success in the world of programming.

CONTENTS

PREFACE 1

Purpose and Audience 1

Structure and Content 1

Why Python? 1

Acknowledgements 1

Conclusion 1

Contents 2

Python programming Handbook 6

What is Programming? 6

What is Python? 6

Features of Python 6

Installation 6

Chapter 1 – Modules, Comments & pip 7

Modules 7

pip 7

Types of Modules 7

Using python as a calculator 7

Comments 7

Types of Comments 7

Chapter 1 – Practice Set 9

Chapter 2 – Variables and Datatype 10

Data Types 10

Rules for choosing an identifier 10

Operators in Python 10

type() function and typecasting. 11

input() Function 11

Chapter 2 – Practice Set 12

Chapter 3 – Strings 13

String Slicing 13

Slicing With Skip Value 14

String Functions 14

Escape Sequence Characters 15

Chapter 3 – Practice Set 16

Chapter 4 – Lists And Tuples 17

List Indexing 17

List methods. 17
Tuples in Python 17
Tuple Methods 17
Chapter 4 - Practice Set 19
Chapter 5 – Dictionary & Sets 20
Properties of Python Dictionaries 20
Dictionary Methods 20
Sets in Python. 20
Properties of Sets 21
Operations on sets 21
Chapter 5 – Practice Set 22
Chapter 6 – Conditional Expression 23
If Else and Elif in Python 23
Code example. 23
Relational Operators 24
Logical Operators 24
Elif clause 24
Important notes: 24
Chapter 6 – Practice Set 25
Chapter 7 – Loops in Python 26
Types of Loops in Python 26
While loop 26
For loop 27
range() Function in Python 27
An Example Demonstrating range() function. 27
For Loop with Else 27
The Break Statement 27
The Continue Statement 28
Pass statement 28
Chapter 7 – Practice Set 29
Chapter 8 – Functions & Recursions 30
Example and syntax of a function 30
Function call 30
Function Definition 30
Types of Functions in Python 30
Functions with Arguments 30
Default Parameter Value 31

Recursion 31
Chapter 8 – Practice Set 33
Project 1: Snake, Water, Gun Game 34
Chapter 9 – File I/O 35
Type of Files. 35
Opening a File 35
Reading a File in Python 35
Other methods to read the file. 36
Modes of opening a file 36
Write Files in Python 36
With Statement 36
Chapter 9 – Practice Set 37
Chapter 10 - Object Oriented Programming 38
Class 38
Object 38
Modelling a problem in OOPs 38
Class Attributes 38
Instance attributes 39
self parameter 39
static method 39
__init__() constructor 40
Chapter 10 – Practice Set 41
Chapter 11 - Inheritance & more on OOPs 42
Types of Inheritance 42
Single Inheritance 42
Multiple Inheritance 43
Multilevel Inheritance 43
super() method 43
class method 44
@property Decorators 44
@.getters and @.setters 44
Operator Overloading in Python 44
Chapter 11- Practice set 46
Project 2 – The Perfect Guess 47
Chapter 12 – Advanced Python 1 48
Newly added features in python 48
Walrus Operator 48

Types Definitions in Python 48
Advanced Type Hints 48
Match Case 49
Dictionary Merge & Update Operators 49
Exception handling in Python 50
Raising Exceptions 50
try with else clause 50
try with finally 51
If __name__== '__main__' in python 51
The global keyword 51
enumerate function in python 51
List comprehensions 51
Chapter 12 – Practice set 52
Chapter 13 – Advanced Python 2 53
Virtual envirionment 53
Installation 53
pip freeze command 53
Lambda functions 53
join method (strings) 54
format method (strings) 54
Map, Filter & Reduce 54
Chapter 13 – Practice Set 56
MEGA Project 1: Jarvis 57
Features 57
Workflow 57
Libraries Used 58
Mega Project 2: Auto Reply AI Chatbot 59
Description 59
Features 59
Workflow 59
Libraries Used 60

PYTHON PROGRAMMING HANDBOOK

WHAT IS PROGRAMMING?

Just like we use Hindi or English to communicate with each other, we use a programming language like Python to communicate with the computer.

Programming is a way to instruct the computer to perform various tasks.

WHAT IS PYTHON?

Python is a simple and easy to understand language which feels like reading simple English. This Pseudo code nature is easy to learn and understandable by beginners.

FEATURES OF PYTHON

- Easy to understand = Less development time
- Free and open source
- High level language
- Portable: Works on Linux / Windows / Mac.
- Fun to work with!

INSTALLATION

Python can be easily installed from python.org. When you click on the download button, python can be installed right after you complete the setup by executing the file for your platform.

CHAPTER 1 – MODULES, COMMENTS & PIP

Let's write our very first python program. Create a file called hello.py and paste the below code in it.

```
print("hello world") # print is a function (more later)
```

Execute this file (.py file) by typing python hello.py and you will see Hello World printed on the screen.

MODULES

A module is a file containing code written by somebody else (usually) which can be imported and used in our programs.

PIP

Pip is the package manager for python. You can use pip to install a module on your system.

```
pip install flask # Installs Flask Module
```

TYPES OF MODULES

There are two types of modules in Python.

1. Built in Modules (Preinstalled in Python)
2. External Modules (Need to install using pip)

Some examples of built in modules are os, random etc.

Some examples of external modules are tensorflow, flask etc.

USING PYTHON AS A CALCULATOR

We can use python as a calculator by typing "python" + ↵ on the terminal.

This opens **REPL** or Read Evaluate Print Loop.

COMMENTS

Comments are used to write something which the programmer does not want to execute. This can be used to mark author name, date etc.

TYPES OF COMMENTS

There are two types of comments in python.

1. Single Line Comments: To write a single line comment just add a '#' at the start of the line.

```
# This is a Single-Line Comment
```

2. Multiline Comments: To write multi-line comments you can use '#' at each line or you can use the multiline string (""" """)

```
"""This is an amazing
    example of a Multiline
    comment!"""
```

CHAPTER 1 – PRACTICE SET

1. Write a program to print Twinkle twinkle little star poem in python.
2. Use REPL and print the table of 5 using it.
3. Install an external module and use it to perform an operation of your interest.
4. Write a python program to print the contents of a directory using the os module. Search online for the function which does that.
5. Label the program written in problem 4 with comments.

CHAPTER 2 – VARIABLES AND DATATYPE

A variable is the name given to a memory location in a program. For example.

```
a= 30      # variables = container to store a value.
b= "harry" # keywords = reserved words in python
c= 71.22   # identifiers = class/function/variable name
```

DATA TYPES

Primarily these are the following data types in Python:

1. Integers
2. Floating point numbers
3. Strings
4. Booleans
5. None

Python is a fantastic language that automatically identifies the type of data for us.

```
a= 71          # identifies a as class <int>
b=88.44        # identifies b as class <float>
name= "harry" # identifies name as class <str>
```

RULES FOR CHOOSING AN IDENTIFIER

- A variable name can contain alphabets, digits, and underscores.
- A variable name can only start with an alphabet and underscores.
- A variable name can't start with a digit.
- No while space is allowed to be used inside a variable name.

Examples of a few variable names are: harry, one8, seven, _seven etc.

OPERATORS IN PYTHON

Following are some common operators in python:

1. Arithmetic operators: +, -, *, / etc.
2. Assignment operators: =, +=, -= etc.
3. Comparison operators: ==, >, >=, <, != etc.
4. Logical operators: and, or, not.

TYPE() FUNCTION AND TYPECASTING.

type() function is used to find the data type of a given variable in python.

```
a = 31
type(a) # class <int>

b = "31"
type (b) # class <str>
```

A number can be converted into a string and vice versa (if possible)

There are many functions to convert one data type into another.

```
str(31)    =>"31"   # integer to string conversion
int("32")  => 32    # string to integer conversion
float(32)  => 32.0  # integer to float conversion
```

... and so, on

Here "31" is a string literal and 31 a numeric literal.

INPUT () FUNCTION

This function allows the user to take input from the keyboard as a string.

```
A = input ("enter name")   # if a is "harry", the user entered harry
```

It is important to note that the output of input is always a string (even is a number is entered).

CHAPTER 2 – PRACTICE SET

1. Write a python program to add two numbers.
2. Write a python program to find remainder when a number is divided by z.
3. Check the type of variable assigned using input () function.
4. Use comparison operator to find out whether ‘a’ given variable a is greater than ‘b’ or not. Take a = 34 and b = 80
5. Write a python program to find an average of two numbers entered by the user.
6. Write a python program to calculate the square of a number entered by the user.

CHAPTER 3 – STRINGS

String is a data type in python.

String is a sequence of characters enclosed in quotes.

We can primarily write a string in these three ways.

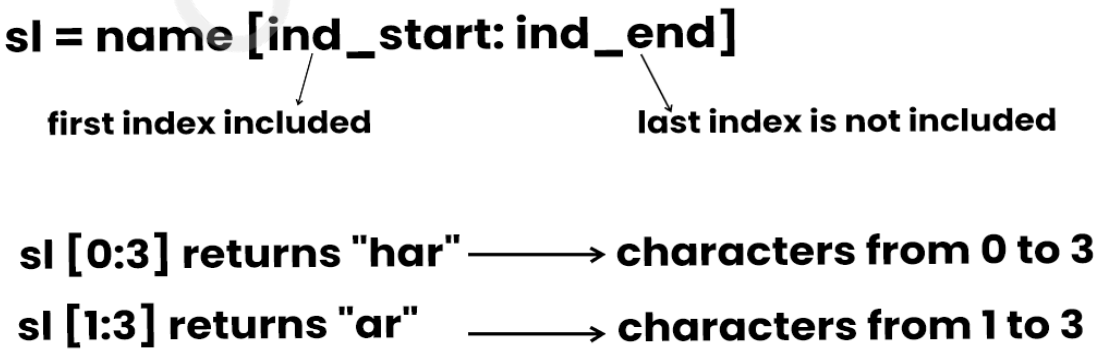
```
a ='harry'        # Single quoted string
b = "harry"       # Double quoted string
c = '''harry'''   # Triple quoted string
```

STRING SLICING

A string in python can be sliced for getting a part of the strings.

Consider the following string:

The index in a sting starts from 0 to (length -1) in Python. In order to slice a string, we use the following syntax:



Negative Indices: Negative indices can also be used as shown in the figure above. -1 corresponds to the (length - 1) index, -2 to (length - 2).

SLICING WITH SKIP VALUE

We can provide a skip value as a part of our slice like this:

```
word = "amazing"

word[1: 6: 2] # "mzn"
```

Other advanced slicing techniques:

```
Word = "amazing"
Word = [:7]  # word [0:7] – 'amazing'
Word = [0:]  # word [0:7] – 'amazing'
```

STRING FUNCTIONS

Some of the commonly used functions to perform operations on or manipulate strings are as follows. Let us assume there is a string ‘str’ as follows:

```
str = 'harry'
```

Now when operated on this string ‘str’, these functions do the following:

1. len () function – This function returns the length of the strings.

```
str = "harry"
print(len(str))  # Output: 5
```

2. String.endswith("rry") – This function_ tells whether the variable string ends with the string "rry" or not. If string is "harry", it returns true for "rry" since Harry ends with rry.

```
str = "harry"
print(str.endswith("rry"))  # Output: True
```

3. string.count("c") – counts the total number of occurrences of any character.

```
str = "harry"
count = str.count("r")
print(count)  # Output: 2
```

4. the first character of a given string.

```
str = "harry"
capitalized_string = str.capitalize()
print(capitalized_string)  # Output: "Harry"
```

5. string.find(word) – This function friends a word and returns the index of first occurrence of that word in the string.

```
str = "harry"
```

```
index = str.find("rr")
print(index)  # Output: 2
```

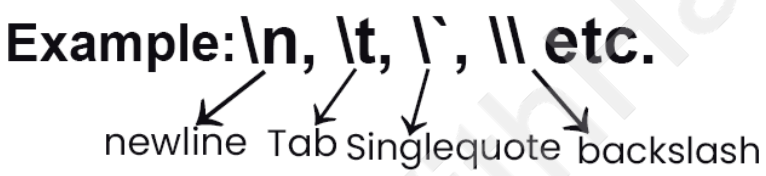
6. string.replace (old word, new word) – This function replace the old word with new word in the entire string.

```
str = "harry"
replaced_string = str.replace("r", "l")
print(replaced_string)  # Output: "hally"
```

ESCAPE SEQUENCE CHARACTERS

Sequence of characters after backslash "\" → Escape Sequence characters

Escape Sequence characters comprise of more than one character but represent one character when used within the strings.



CHAPTER 3 – PRACTICE SET

1. Write a python program to display a user entered name followed by Good Afternoon using input () function.
2. Write a program to fill in a letter template given below with name and date.

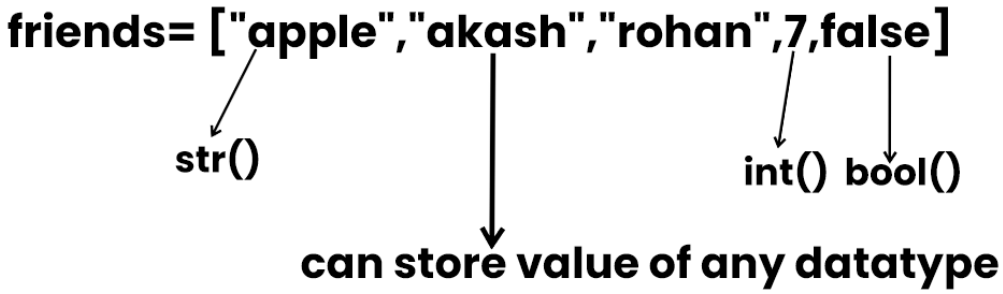
```
letter = '''
       Dear <|Name|>,
       You are selected!
       <|Date|>
        '''
```

3. Write a program to detect double space in a string.
4. Replace the double space from problem 3 with single spaces.
5. Write a program to format the following letter using escape sequence characters.

```
letter = "Dear Harry, this python course is nice. Thanks!"
```

CHAPTER 4 – LISTS AND TUPLES

Python lists are containers to store a set of values of any data type.



LIST INDEXING

A list can be indexed just like a string.

```
l1 = [7,9,"harry"]

l1[0] # 7
l1[1] # 9
l1[70]  # error
l1[0:2] # [7,9]  #list slicing
```

LIST METHODS.

Consider the following list:

```
l1 = [1,8,7,2,21,15]
```

- l1.sort(): updates the list to [1,2,7,8,15,21]
- l1.reverse(): updates the list to [15,21,2,7,8,1]
- l1.append(8): adds 8 at the end of the list
- l1.insert(3,8): This will add 8 at 3 index
- l1.pop(2): Will delete element at index 2 and return its value.
- l1.remove(21): Will remove 21 from the list.

TUPLES IN PYTHON

A tuple is an immutable data type in python.

```
a = () # empty tuple
a = (1,) # tuple with only one element needs a comma
a = (1,7,2) # tuple with more than one element
```

TUPLE METHODS

Consider the following tuple.

```
a = (1, 7, 2)
```

- a.count (1): a count (1) will return number of times 1 occurs in a.
- a.index (1) will return the index of first occurrence of 1 in a.

CHAPTER 4 - PRACTICE SET

1. Write a program to store seven fruits in a list entered by the user.
2. Write a program to accept marks of 6 students and display them in a sorted manner.
3. Check that a tuple type cannot be changed in python.
4. Write a program to sum a list with 4 numbers.
5. Write a program to count the number of zeros in the following tuple:

```
a = (7, 0, 8, 0, 0, 9)
```

CHAPTER 5 – DICTIONARY & SETS

Dictionary is a collection of keys-value pairs.

Syntax:

```
a = {
    "key": "value",
    "harry": "code",
    "marks": "100",
    "list": [1, 2, 9]
}

print(a["key"])  # Output: "value"
print(a["list"])  # Output: [1, 2, 9]
```

PROPERTIES OF PYTHON DICTIONARIES

1. It is unordered.
2. It is mutable.
3. It is indexed.
4. Cannot contain duplicate keys.

DICTIONARY METHODS

Consider the following dictionary.

```
a={"name":"harry"
   "from":"india"
   "marks":[92,98,96]}
```

- a.items(): Returns a list of (key,value)tuples.
- a.keys(): Returns a list containing dictionary's keys.
- a.update({"friends":}): Updates the dictionary with supplied key-value pairs.
- a.get("name"): Returns the value of the specified keys (and value is returned eg."harry" is returned here).

More methods are available on [docs.python.org]

SETS IN PYTHON.

Set is a collection of non-repetitive elements.

```
s = set()           # no repetition allowed!
s.add(1)
s.add(2)           # or set ={1,2}
```

If you are a programming beginner without much knowledge of mathematical operations on sets, you can simply look at sets in python as data types containing unique values.

PROPERTIES OF SETS

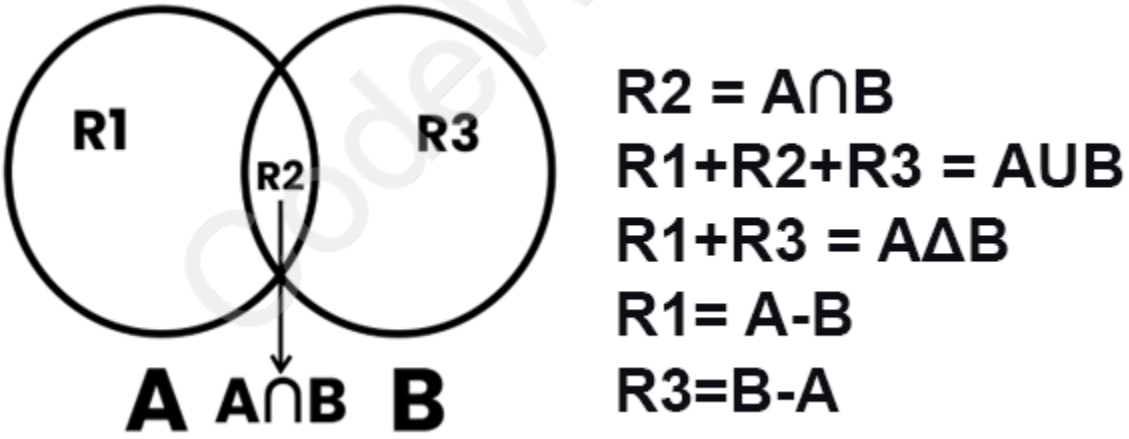
1. Sets are unordered => Element's order doesn't matter
2. Sets are unindexed => Cannot access elements by index
3. There is no way to change items in sets.
4. Sets cannot contain duplicate values.

OPERATIONS ON SETS

Consider the following set:

```
s = {1,8,2,3}
```

- len(s): Returns 4, the length of the set
- s.remove(8): Updates the set s and removes 8 from s.
- s.pop(): Removes an arbitrary element from the set and return the element removed.
- s.clear():empties the set s.
- s.union({8,11}): Returns a new set with all items from both sets. {1,8,2,3,11}.
- s.intersection({8,11}): Return a set which contains only item in both sets {8}.



CHAPTER 5 – PRACTICE SET

1. Write a program to create a dictionary of Hindi words with values as their English translation. Provide user with an option to look it up!
2. Write a program to input eight numbers from the user and display all the unique numbers (once).
3. Can we have a set with 18 (int) and `'18'` (str) as a value in it?
4. What will be the length of following set s:

```
s = set()
s.add(20)
s.add(20.0)
s.add('20') # length of s after these operations?
```

5. s = {}
   What is the type of `'s'`?
6. Create an empty dictionary. Allow 4 friends to enter their favorite language as value and use key as their names. Assume that the names are unique.
7. If the names of 2 friends are same; what will happen to the program in problem 6?
8. If languages of two friends are same; what will happen to the program in problem 6?
9. Can you change the values inside a list which is contained in set S?

```
s = {8, 7, 12, "Harry", [1,2]}
```

CHAPTER 6 – CONDITIONAL EXPRESSION

Sometimes we want to play PUBG on our phone if the day is Sunday.

Sometimes we order Ice Cream online if the day is sunny.

Sometimes we go hiking if our parents allow.

All these are decisions which depend on a condition being met.

In python programming too, we must be able to execute instructions on a condition(s) being met.

This is what conditionals are for!

IF ELSE AND ELIF IN PYTHON

If else and elif statements are a multiway decision taken by our program due to certain conditions in our code.

Syntax:

```
if (condition1): # if condition1 is True
    print ("yes")

elif(condition2): # if condition2 is True
    print("no")

else:             # otherwise
    print("maybe")
```

CODE EXAMPLE.

```
a=22
if(a>9):
    print("greater")
else:
    print("lesser")
```

Quick Quiz: Write a program to print yes when the age entered by the user is greater than or equal to 18.

RELATIONAL OPERATORS

Relational Operators are used to evaluate conditions inside the if statements. Some examples of relational operators are:

==: equals.

> =: greater than/ equal to.

< =: lesser than/ equal to.

LOGICAL OPERATORS

In python logical operators operate on conditional statements. For Example:

- and – true if both operands are true else false.
- or – true if at least one operand is true or else false.
- not – inverts true to false & false to true.

ELIF CLAUSE

elif in python means [else if]. An if statements can be chained together with a lot of these elif statements followed by an else statement.

```
if (condition1):
    #code
elif (condition2):   # this ladder will stop once a condition in an if or
elif is met.
    #code
elif(condition3):
    #code
else:
    #code
```


IMPORTANT NOTES:

1. There can be any number of elif statements.
2. Last else is executed only if all the conditions inside elifs fail.

CHAPTER 6 – PRACTICE SET

1. Write a program to find the greatest of four numbers entered by the user.
2. Write a program to find out whether a student has passed or failed if it requires a total of 40% and at least 33% in each subject to pass. Assume 3 subjects and take marks as an input from the user.
3. A spam comment is defined as a text containing following keywords:
   "Make a lot of money", "buy now", "subscribe this", "click this". Write a program to detect these spams.
4. Write a program to find whether a given username contains less than 10 characters or not.
5. Write a program which finds out whether a given name is present in a list or not.
6. Write a program to calculate the grade of a student from his marks from the following scheme:
   90 – 100 => Ex
   80 – 90 => A
   70 – 80 => B
   60 – 70 =>C
   50 – 60 => D
   <50 => F
7. Write a program to find out whether a given post is talking about "Harry" or not.

CHAPTER 7 – LOOPS IN PYTHON

Sometimes we want to repeat a set of statements in our program. For instance: Print 1 to 1000.

Loops make it easy for a programmer to tell the computer which set of instructions to repeat and how!

TYPES OF LOOPS IN PYTHON

Primarily there are two types of loops in python.

- while loops
- for loops

We will look into these one by one.

WHILE LOOP

Syntax:

```
while (condition): # The block keeps executing until the condition is true
    #Body of the loop
```

In while loops, the condition is checked first. If it evaluates to true, the body of the loop is executed otherwise not!

If the loop is entered, the process of [condition check & execution] is continued until the condition becomes False.

Quick Quiz: Write a program to print 1 to 50 using a while loop.

Example:

```
i = 0
while i < 5: # print "Harry" – 5 times!
    print("Harry")
    i = i + 1
```

Note: If the condition never become false, the loop keeps getting executed.

Quick Quiz: Write a program to print the content of a list using while loops.

FOR LOOP

A for loop is used to iterate through a sequence like list, tuple, or string [iterables]

Syntax:

```
l = [1, 7, 8]
for item in l:
    print(item) # prints 1, 7 and 8
```

RANGE FUNCTION IN PYTHON

The range() function in python is used to generate a sequence of number.

We can also specify the start, stop and step-size as follows:

```
range(start, stop, step_size)
# step_size is usually not used with range()
```

AN EXAMPLE DEMONSTRATING RANGE () FUNCTION.

```
for i in range(0,7): # range(7) can also be used.

    print(i) # prints 0 to 6
```

FOR LOOP WITH ELSE

An optional else can be used with a for loop if the code is to be executed when the loops exhausts.

Example:

```
l= [1,7,8]
for item in l:
    print(item)
else:
    print("done") # this is printed when the loop exhausts!
```

Output:

```
1
7
8
done
```

THE BREAK STATEMENT

'break' is used to come out of the loop when encountered. It instructs the program to – exit the loop now.

Example:

```
for i in range (0,80):
    print(i)     # this will print 0,1,2 and 3
    if i==3
        break
```

THE CONTINUE STATEMENT

'continue' is used to stop the current iteration of the loop and continue with the next one. It instructs the Program to "skip this iteration".

Example:

```
for i in range(4):
    print("printing")
    if i == 2:   # if i is 2, the iteration is skipped
        continue
    print(i)
```

PASS STATEMENT

pass is a null statement in python.

It instructs to "do nothing".

Example:

```
l = [1,7,8]
for item in l:
    pass          # without pass, the program will throw an error
```

CHAPTER 7 – PRACTICE SET

1. Write a program to print multiplication table of a given number using for loop.
2. Write a program to greet all the person names stored in a list ‘l’ and which starts with S.
   l = ["Harry", "Soham", "Sachin", "Rahul"]
3. Attempt problem 1 using while loop.
4. Write a program to find whether a given number is prime or not.
5. Write a program to find the sum of first n natural numbers using while loop.
6. Write a program to calculate the factorial of a given number using for loop.
7. Write a program to print the following star pattern.
   ```
    *
   ***
   ***** for n = 3
   ```
8. Write a program to print the following star pattern:
   ```
   *
   **
   ***   for n = 3
   ```
9. Write a program to print the following star pattern.
   ```
   * * *
   *   *   for n = 3
   * * *
   ```
10. Write a program to print multiplication table of n using for loops in reversed order.

CHAPTER 8 – FUNCTIONS & RECURSIONS

A function is a group of statements performing a specific task.

When a program gets bigger in size and its complexity grows, it gets difficult for a program to keep track on which piece of code is doing what!

A function can be reused by the programmer in a given program any number of

EXAMPLE AND SYNTAX OF A FUNCTION

The syntax of a function looks as follows:

```
def func1():
    print('hello')
```

This function can be called any number of times, anywhere in the program.

FUNCTION CALL

Whenever we want to call a function, we put the name of the function followed by parentheses as follows:

```
func1() # This is called function call.
```

FUNCTION DEFINITION

The part containing the exact set of instructions which are executed during the function call.

Quick Quiz: Write a program to greet a user with “Good day” using functions.

TYPES OF FUNCTIONS IN PYTHON

There are two types of functions in python:

- Built in functions (Already present in python)
- User defined functions (Defined by the user)

Examples of built in functions includes len(), print(), range() etc.

The func1() function we defined is an example of user defined function.

FUNCTIONS WITH ARGUMENTS

A function can accept some value it can work with. We can put these values in the parentheses.

A function can also return value as shown below:

```
def greet(name):
    gr = "hello"+ name
    return gr

a = greet ("harry")
# a will now contain "hello harry"
```

DEFAULT PARAMETER VALUE

We can have a value as default as default argument in a function.

If we specify name = “stranger” in the line containing def, this value is used when no argument is passed.

Example:

```
def greet(name = "stranger"):
    # function body
greet() # name will be "stranger" in function body (default)
greet("harry") # name will be "harry" in function body (passed)
```

RECURSION

Recursion is a function which calls itself.

It is used to directly use a mathematical formula as function.

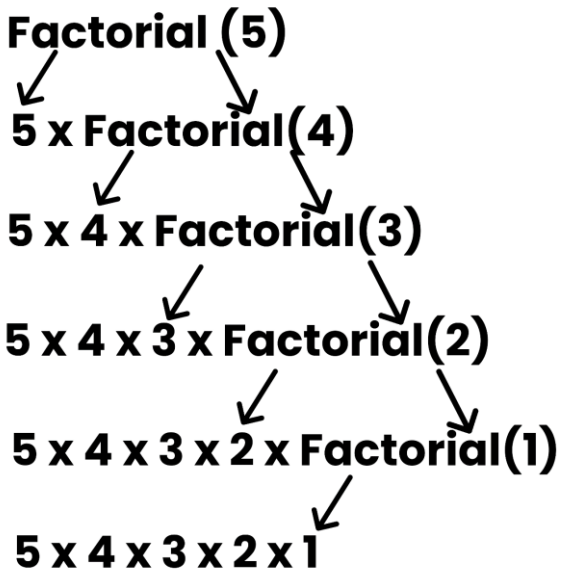
Example:

```
factorial(n) = n x factorial (n-1)
```

This function can be defined as follows:

```
def factorial(n)
    if i == 0 or i==1: # base condition which doesn’t call the function
any further
    return 1
else:
    return n*factorial(n-1) # function calling itself
```

This works as follows:



The programmer needs to be extremely careful while working with recursion to ensure that the function doesn't infinitely keep calling itself. Recursion is sometimes the most direct way to code an algorithm.

CHAPTER 8 – PRACTICE SET

1. Write a program using functions to find greatest of three numbers.
2. Write a python program using function to convert Celsius to Fahrenheit.
3. How do you prevent a python print() function to print a new line at the end.
4. Write a recursive function to calculate the sum of first n natural numbers.
5. Write a python function to print first n lines of the following pattern:

   ***
   ** - for n = 3
   *

6. Write a python function which converts inches to cms.
7. Write a python function to remove a given word from a list ad strip it at the same time.
8. Write a python function to print multiplication table of a given number.

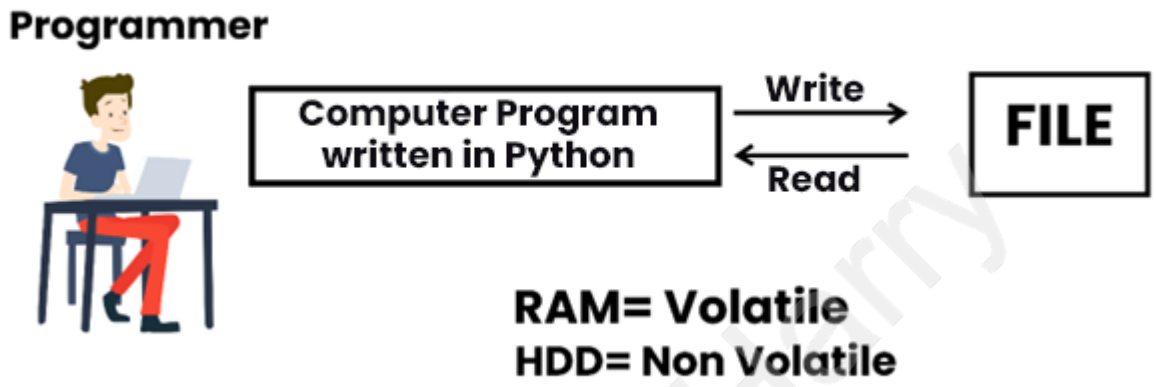
PROJECT 1: SNAKE, WATER, GUN GAME

We all have played snake, water gun game in our childhood. If you haven't, google the rules of this game and write a python program capable of playing this game with the user.

CHAPTER 9 – FILE I/O

The random-access memory is volatile, and all its contents are lost once a program terminates. In order to persist the data forever, we use files.

A file is data stored in a storage device. A python program can talk to the file by reading content from it and writing content to it.



TYPE OF FILES.

There are 2 types of files:

1. Text files (.txt, .c, etc)
2. Binary files (.jpg, .dat, etc)

Python has a lot of functions for reading, updating, and deleting files.

OPENING A FILE

Python has an open() function for opening files. It takes 2 parameters: filename and mode.

```
# open("filename", "mode of opening(read mode by default)")
open("this.txt", "r")
```

READING A FILE IN PYTHON

```
# Open the file in read mode
f = open("this.txt", "r")
# Read its contents
text = f.read()
# Print its contents
print(text)
```

```
# Close the file
f.close()
```

OTHER METHODS TO READ THE FILE.

We can also use f.readline() function to read one full line at a time.

```
f.readline() # Read one line from the file.
```

MODES OF OPENING A FILE

r – open for reading

w - open for writing

a - open for appending

+ - open for updating.

'rb' will open for read in binary mode.

'rt' will open for read in text mode.

WRITE FILES IN PYTHON

In order to write to a file, we first open it in write or append mode after which, we use the python's f.write() method to write to the file!

```
# Open the file in write mode
f = open("this.txt", "w")
# Write a string to the file
f.write("this is nice")
# Close the file
f.close()
```


WITH STATEMENT

The best way to open and close the file automatically is the with statement.

```
# Open the file in read mode using 'with', which automatically closes the
file
with open("this.txt", "r") as f:
    # Read the contents of the file
    text = f.read()

# Print the contents
print(text)
```

CHAPTER 9 – PRACTICE SET

1. Write a program to read the text from a given file ‘poems.txt’ and find out whether it contains the word ‘twinkle’.
2. The game() function in a program lets a user play a game and returns the score as an integer. You need to read a file ‘Hi-score.txt’ which is either blank or contains the previous Hi-score. You need to write a program to update the Hi-score whenever the game() function breaks the Hi-score.
3. Write a program to generate multiplication tables from 2 to 20 and write it to the different files. Place these files in a folder for a 13 – year old.
4. A file contains a word “Donkey” multiple times. You need to write a program which replace this word with ***#####*** by updating the same file.
5. Repeat program 4 for a list of such words to be censored.
6. Write a program to mine a log file and find out whether it contains ‘python’.
7. Write a program to find out the line number where python is present from ques 6.
8. Write a program to make a copy of a text file “this. txt”
9. Write a program to find out whether a file is identical & matches the content of another file.
10. Write a program to wipe out the content of a file using python.
11. Write a python program to rename a file to “renamed_by_ python.txt.

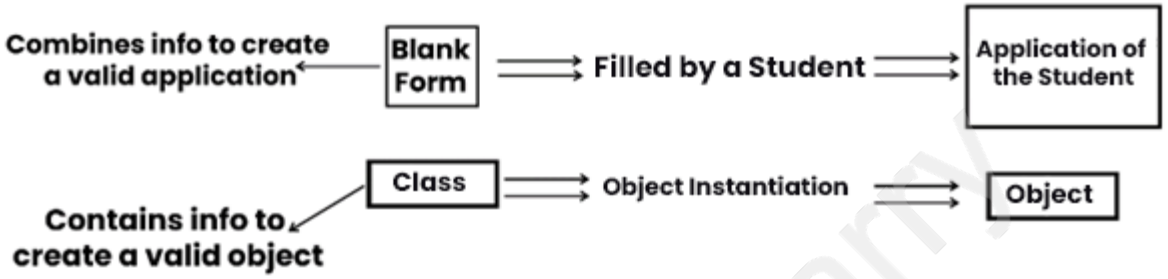
CHAPTER 10 - OBJECT ORIENTED PROGRAMMING

Solving a problem by creating object is one of the most popular approaches in programming. This is called object-oriented programming.

This concept focuses on using reusable code (DRY Principle).

CLASS

A class is a blueprint for creating object.



Syntax:

```
class Employee: # Class name is written in pascal case
    # Methods & Variables
```

OBJECT

An object is an instantiation of a class. When class is defined, a template (info) is defined. Memory is allocated only after object instantiation.

Objects of a given class can invoke the methods available to it without revealing the implementation details to the user. – Abstractions & Encapsulation!

MODELLING A PROBLEM IN OOPS

We identify the following in our problem.

- Noun → Class → Employee
- Adjective → Attributes → name, age, salary
- Verbs → Methods → getSalary(), increment()

CLASS ATTRIBUTES

An attribute that belongs to the class rather than a particular object.

Example:

```
class Employee:
    company = "Google" # Specific to Each Class
harry = Employee() # Object Instatiation
harry.company
Employee.company = "YouTube" # Changing Class Attribute
```

INSTANCE ATTRIBUTES

An attribute that belongs to the Instance (object). Assuming the class from the previous example:

```
harry.name = "harry"
harry.salary = "30k"  # Adding instance attribute
```

Note: Instance attributes, take preference over class attributes during assignment & retrieval.

When looking up for harry.attribute it checks for the following:

1) Is attribute present in object?

2) Is attribute present in class?

SELF PARAMETER

self refers to the instance of the class. It is automatically passed with a function call from an object.

```
harry.getSalary() # here self is harry
# equivalent to Employee.getSalary(harry)
```

The function getSalary() is defined as:

```
class Employee:
    company = "Google"
    def getSalary(self):
        print("Salary is not there")
```

STATIC METHOD

Sometimes we need a function that does not use the self-parameter. We can define a static method like this:

```
@staticmethod   # decorator to mark greet as a static method
def greet():
    print("Hello user")
```

__INIT__() CONSTRUCTOR

__init__() is a special method which is first run as soon as the object is created.

__init__() method is also known as constructor.

It takes 'self' argument and can also take further arguments.

For Example:

```
class Employee:
    def __init__(self, name):
        self.name=name
    def getSalary(self):
        ...

harry = Employee("Harry")
```

CHAPTER 10 – PRACTICE SET

1. Create a class *"Programmer"* for storing information of few programmers working at Microsoft.
2. Write a class *"Calculator"* capable of finding square, cube and square root of a number.
3. Create a class with a class attribute a; create an object from it and set 'a' directly using 'object.a = 0'. Does this change the class attribute?
4. Add a static method in problem 2, to greet the user with hello.
5. Write a Class 'Train' which has methods to book a ticket, get status (no of seats) and get fare information of train running under Indian Railways.
6. Can you change the self-parameter inside a class to something else (say "harry"). Try changing self to "slf" or "harry" and see the effects.

CHAPTER 11 - INHERITANCE & MORE ON OOPS

Inheritance is a way of creating a new class from an existing class.

Syntax:

```
class Employee:  # Base class
    # Code

class Programmer(Employee): # Derived or child class
    # Code
```

We can use the method and attributes of 'Employee' in 'Programmer' object.

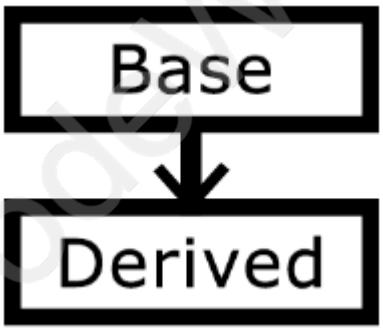
Also, we can overwrite or add new attributes and methods in 'Programmer' class.

TYPES OF INHERITANCE

- Single inheritance
- Multiple inheritance
- Multilevel inheritance

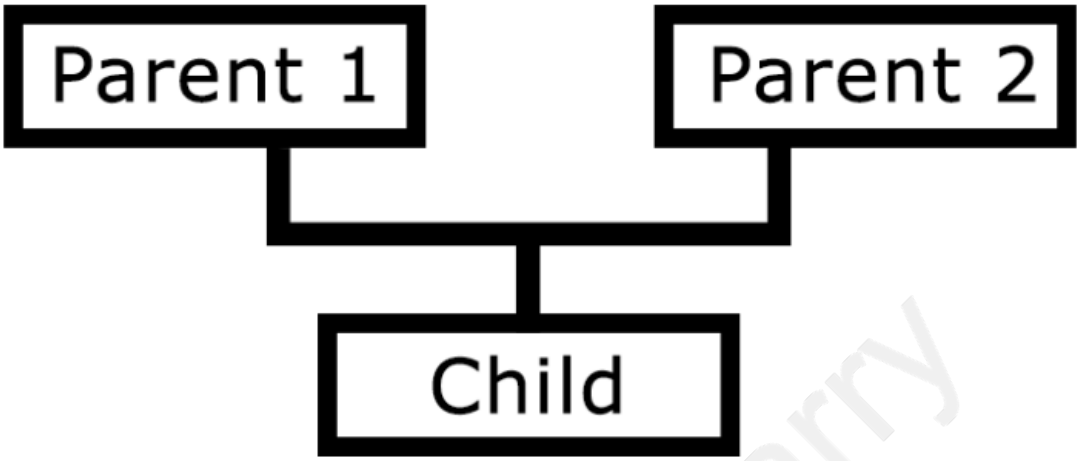
SINGLE INHERITANCE

Single inheritance occurs when child class inherits only a single parent class.



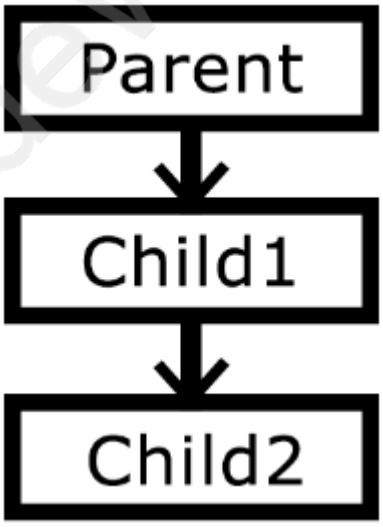
MULTIPLE INHERITANCE

Multiple Inheritance occurs when the child class inherits from more than one parent classes.



MULTILEVEL INHERITANCE

When a child class becomes a parent for another child class.



SUPER() METHOD

super() method is used to access the methods of a super class in the derived class.

```
super().__init__()
# __init__() Calls constructor of the base class
```

CLASS METHOD

A class method is a method which is bound to the class and not the object of the class.

@classmethod decorator is used to create a class method.

Syntax:

```
@classmethod
    def(cls,p1,p2):
```

@PROPERTY DECORATORS

Consider the following class:

```
class Employee:
    @property
    def name(self):
        return self.ename
```

If e = Employee() is an object of class employee, we can print (e.name) to print the ename by internally calling name() function.

@.GETTERS AND @.SETTERS

The method name with '@property' decorator is called getter method.

We can define a function + @ name.setter decorator like below:

```
@name.setter
def name (self,value):
    self.ename = value
```

OPERATOR OVERLOADING IN PYTHON

Operators in Python can be overloaded using dunder methods.

These methods are called when a given operator is used on the objects.

Operators in Python can be overloaded using the following methods:

```
p1+p2 # p1.__add__(p2)
p1-p2 # p1.__sub__(p2)
p1*p2 # p1.__mul__(p2)
p1/p2 # p1.__truediv__(p2)
p1//p2 # p1.__floordiv__(p2)
```

Other dunder/magic methods in Python:

```
str__() # used to set what gets displayed upon calling str(obj)
```

```
__len__() # used to set what gets displayed upon calling.__len__() or
len(obj)
```

CHAPTER 11- PRACTICE SET

1. Create a class (2-D vector) and use it to create another class representing a 3-D vector.

2. Create a class ‘Pets’ from a class ‘Animals’ and further create a class ‘Dog’ from ‘Pets’. Add a method ‘bark’ to class ‘Dog’.

3. Create a class ‘Employee’ and add salary and increment properties to it.

Write a method ‘salaryAfterIncrement’ method with a @property decorator with a setter which changes the value of increment based on the salary.

4. Write a class ‘Complex’ to represent complex numbers, along with overloaded operators ‘+’ and ‘*’ which adds and multiplies them.

5. Write a class vector representing a vector of n dimensions. Overload the + and * operator which calculates the sum and the dot(.) product of them.

6. Write __str__() method to print the vector as follows:

```
7i + 8j +10k
```

Assume vector of dimension 3 for this problem.

7. Override the __len__() method on vector of problem 5 to display the dimension of the vector.

PROJECT 2 – THE PERFECT GUESS

We are going to write a program that generates a random number and asks the user to guess it.

If the player's guess is higher than the actual number, the program displays "Lower number please". Similarly, if the user's guess is too low, the program prints "higher number please" When the user guesses the correct number, the program displays the number of guesses the player used to arrive at the number.

Hint: *Use the random module*.

CHAPTER 12 – ADVANCED PYTHON 1

NEWLY ADDED FEATURES IN PYTHON

Following are some of the newly added features in Python programming language

WALRUS OPERATOR

The walrus operator (`:=`), introduced in Python 3.8, allows you to assign values to variables as part of an expression. This operator, named for its resemblance to the eyes and tusks of a walrus, is officially called the "assignment expression."

```
# Using walrus operator
if (n := len([1, 2, 3, 4, 5])) > 3:
    print(f"List is too long ({n} elements, expected <= 3)")

# Output: List is too long (5 elements, expected <= 3)
```

In this example, `n` is assigned the value of `len([1, 2, 3, 4, 5])` and then used in the comparison within the `if` statement.

TYPES DEFINITIONS IN PYTHON

Type hints are added using the colon (`:`) syntax for variables and the `->` syntax for function return types.

```
# Variable type hint
age: int = 25

# Function type hints
def greeting(name: str) -> str:
    return f"Hello, {name}!"

# Usage
print(greeting("Alice"))  # Output: Hello, Alice!
```

ADVANCED TYPE HINTS

Python's typing module provides more advanced type hints, such as List, Tuple, Dict, and Union.

You can import List, Tuple and Dict types from the typing module like this:

```
from typing import List, Tuple, Dict, Union
```

The syntax of types looks something like this:

```
from typing import List, Tuple, Dict, Union

# List of integers
numbers: List[int] = [1, 2, 3, 4, 5]

# Tuple of a string and an integer
person: Tuple[str, int] = ("Alice", 30)

# Dictionary with string keys and integer values
scores: Dict[str, int] = {"Alice": 90, "Bob": 85}

# Union type for variables that can hold multiple types
identifier: Union[int, str] = "ID123"
identifier = 12345  # Also valid
```

These annotations help in making the code self-documenting and allow developers to understand the data structures used at a glance.

MATCH CASE

Python 3.10 introduced the match statement, which is similar to the switch statement found in other programming languages.

The basic syntax of the match statement involves matching a variable against several cases using the case keyword.

```
def http_status(status):
    match status:
        case 200:
            return "OK"
        case 404:
            return "Not Found"
        case 500:
            return "Internal Server Error"
        case _:
            return "Unknown status"

# Usage
print(http_status(200))  # Output: OK
print(http_status(404))  # Output: Not Found
print(http_status(500))  # Output: Internal Server Error
print(http_status(403))  # Output: Unknown status
```

DICTIONARY MERGE & UPDATE OPERATORS

New operators `|` and `|=` allow for merging and updating dictionaries.

```
dict1 = {'a': 1, 'b': 2}
dict2 = {'b': 3, 'c': 4}
merged = dict1 | dict2
print(merged)  # Output: {'a': 1, 'b': 3, 'c': 4}
```

You can now use multiple context managers in a single `with` statement more cleanly using the parenthesised context manager

```
with (
    open('file1.txt') as f1,
    open('file2.txt') as f2
):
    # Process files
```

EXCEPTION HANDLING IN PYTHON

There are many built-in exceptions which are raised in python when something goes wrong.

Exception in python can be handled using a try statement. The code that handles the exception is written in the except clause.

```
try:
   # Code which might throw exception
except Exception as e:
    print(e)
```

When the exception is handled, the code flow continues without program interruption.

We can also specify the exception to catch like below:

```
try:
    # Code
except ZeroDivisionError:
    # Code
except TypeError:
    # Code
except:
    # Code       # All other exceptions are handled here.
```

RAISING EXCEPTIONS

We can raise custom exceptions using the ‘raise’ keyword in python.

TRY WITH ELSE CLAUSE

Sometimes we want to run a piece of code when try was successful.

```
try:
    # Somecode
except:
    # Somecode
else:
    # Code          # This is executed only if the try was successful
```

TRY WITH FINALLY

Python offers a 'finally' clause which ensures execution of a piece of code inspective of the exception.

```
try:
    # Some Code
except:
    # Some Code
finally:
    # Some Code           # Executed regardless of error!
```

IF __NAME__== '__MAIN__' IN PYTHON

'__name__' evaluates to the name of the module in python from where the program is ran.

If the module is being run directly from the command line, the ' __name__' is set to string "__main__". Thus, this behaviour is used to check whether the module is run directly or imported to another file.

THE GLOBAL KEYWORD

'global' keyword is used to modify the variable outside of the current scope.

ENUMERATE FUNCTION IN PYTHON

The 'enumerate' function adds counter to an iterable and returns it

```
for i,item in list1:
    print(i,item)  # Prints the items of list 1 with index
```

LIST COMPREHENSIONS

List Comprehension is an elegant way to create lists based on existing lists.

```
list1 = [1,7,12,11,22,]
list2 = [i for item in list 1 if item > 8]
```

CHAPTER 12 – PRACTICE SET

1. Write a program to open three files 1.txt, 2.txt and 3.txt if any these files are not present, a message without exiting the program must be printed prompting the same.

2. Write a program to print third, fifth and seventh element from a list using enumerate function.

3. Write a list comprehension to print a list which contains the multiplication table of a user entered number.

4. Write a program to display a/b where a and b are integers. If b=0, display infinite by handling the 'ZeroDivisionError'.

5. Store the multiplication tables generated in problem 3 in a file named Tables.txt.

CHAPTER 13 – ADVANCED PYTHON 2

VIRTUAL ENVIRIONMENT

An environment which is same as the system interpreter but is isolated from the other Python environments on the system.

INSTALLATION

To use virtual environments, we write:

```
pip install virtualenv  # Install the package
```

We create a new environment using:

```
virtualenv myprojectenv # Creates a new venv
```

The next step after creating the virtual environment is to activate it.

We can now use this virtual environment as a separate Python installation.

PIP FREEZE COMMAND

'pip freeze' returns all the package installed in a given python environment along with the versions.

```
pip freeze > requirements .txt
```

The above command creates a file named 'requirements.txt' in the same directory containing the output of 'pip freeze'.

We can distribute this file to other users, and they can recreate the same environment using:

```
pip install -r requirements.txt
```

LAMBDA FUNCTIONS

Function created using an expression using 'lambda' keyword.

Syntax:

```
lambda arguments:expressions

# can be used as a normal function
```

Example:

```
square = lambda x:x*x
square(6)  # returns 36
sum = lambda a,b,c:a+b+c
```

```
sum(1,2,3) # returns 6
```

JOIN METHOD (STRINGS)

Creates a string from iterable objects.

```
l = ["apple", "mango", "banana"]
result = ", and, ".join(l)
print(result)
```

The above line will return “apple,and,mango,and,banana”.

FORMAT METHOD (STRINGS)

Formats the values inside the string into a desired output.

```
template.format(p1,p2...)
```

Syntax:

```
"{} is a good {}".format("harry", "boy")   #1.
"{} is a good {o}".format("harry", "boy")  #2.

# output for 1:
# harry is a good boy

# output for 2:
# boy is a good harry
```

MAP, FILTER & REDUCE

Map applies a function to all the items in an input_list.

Syntax.

```
map(function, input_list)
              # the function can be lambda function
```

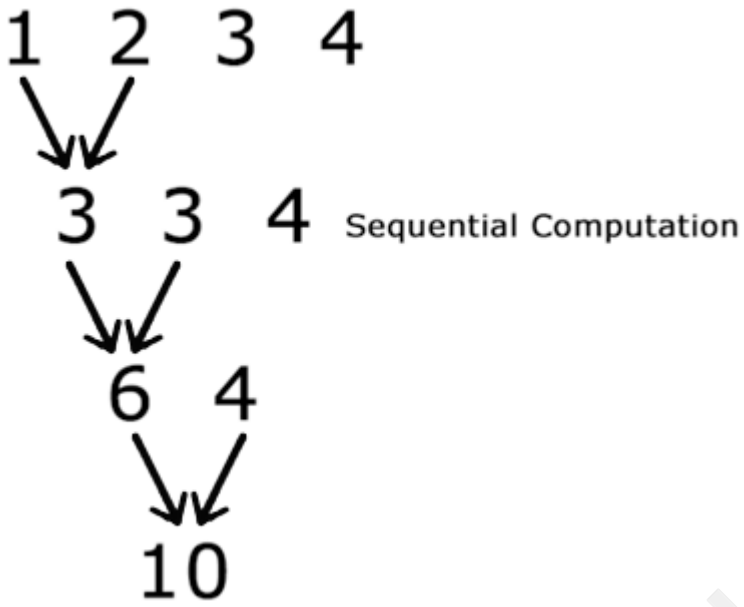
Filter creates a list of items for which the function returns true.

```
list(filter(function))
              # the function can be lambda function
```

Reduce applies a rolling computation to sequential pair of elements.

```
from functools import reduce
val=reduce (function, list1)
              # the function can be lambda function
```

If the function computes sum of two numbers and the list is [1,2,3,4]



CHAPTER 13- PRACTICE SET

1. Create two virtual environments, install few packages in the first one. How do you create a similar environment in the second one?

2. Write a program to input name, marks and phone number of a student and format it using the format function like below:

“The name of the student is Harry, his marks are 72 and phone number is 99999888”

3. A list contains the multiplication table of 7. write a program to convert it to vertical string of same numbers.

4. Write a program to filter a list of numbers which are divisible by 5.

5. Write a program to find the maximum of the numbers in a list using the reduce function.

6. Run pip freeze for the system interpreter. Take the contents and create a similar virtualenv.

7. Explore the ‘Flask’ module and create a web server using Flask & Python.

MEGA PROJECT 1: JARVIS - VOICE-ACTIVATED VIRTUAL ASSISTANT

Jarvis is a voice-activated virtual assistant designed to perform tasks such as web browsing, playing music, fetching news, and responding to user queries using OpenAI's GPT-3.5-turbo model.

FEATURES

- Voice Recognition
- Utilizes the speech_recognition library to listen for and recognize voice commands.
- Activates upon detecting the wake word "Jarvis."
- Text-to-Speech
- Converts text to speech using pyttsx3 for local conversion.
- Uses gTTS (Google Text-to-Speech) and pygame for playback.
- Web Browsing.
- Opens websites like Google, Facebook, YouTube, and LinkedIn based on voice commands.
- Music Playback
- Interfaces with a musicLibrary module to play songs via web links.
- News Fetching
- Fetches and reads the latest news headlines using NewsAPI.
- OpenAI Integration
- Handles complex queries and generates responses using OpenAI's GPT-3.5-turbo.
- Acts as a general virtual assistant similar to Alexa or Google Assistant.
- Activates upon detecting the wake word "Jarvis."
- Text-to-Speech

WORKFLOW

1. Initialization
2. Greets the user with "Initializing Jarvis...."
3. Wake Word Detection
4. Listens for the wake word "Jarvis."
5. Acknowledges activation by saying "Ya."
6. Command Processing.
7. Processes commands to determine actions such as opening a website, playing music, fetching news, or generating a response via OpenAI.
8. Speech Output.
9. Provides responses using speak function with either pyttsx3 or gTTS.
10. Greets the user with "Initializing Jarvis...."
11. Wake Word Detection
12. Acknowledges activation by saying "Ya."

13. Processes commands to determine actions such as opening a website, playing music, fetching news, or generating a response via OpenAI.

LIBRARIES USED

- speech_recognition
- webbrowser
- pyttsx3
- musicLibrary
- requests
- openai
- gTTS
- pygame
- os

MEGA PROJECT 2: AUTO-REPLY AI CHATBOT

DESCRIPTION

This project automates the process of interacting with a chat application, specifically designed to analyze chat history and generate humorous responses using OpenAI's GPT-3.5-turbo model. The virtual assistant, named Naruto, is a character that roasts people in a funny way, based on the chat history.

FEATURES

14. Automated Chat Interaction
15. Uses pyautogui to perform mouse and keyboard operations, interacting with the chat application without manual intervention.
16. Chat History Analysis
17. Copies chat history from the chat application and analyzes it to determine if the last message was sent by a specific user (e.g., "Rohan Das").
18. Humorous Response Generation
19. Integrates with OpenAI's GPT-3.5-turbo model to generate funny, roast-style responses based on the analyzed chat history.
20. Clipboard Operations
21. Utilizes pyperclip to copy and paste text, facilitating the retrieval and insertion of chat messages.
22. Uses pyautogui to perform mouse and keyboard operations, interacting with the chat application without manual intervention.
23. Copies chat history from the chat application and analyzes it to determine if the last message was sent by a specific user (e.g., "Rohan Das").
24. Humorous Response Generation
25. Integrates with OpenAI's GPT-3.5-turbo model to generate funny, roast-style responses based on the analyzed chat history.

WORKFLOW

- Initialization and Setup
- Click on the Chrome icon to open the chat application.
- Wait for a brief period to ensure the application is open and ready for interaction.
- Chat History Retrieval
- Periodically select and copy chat history by dragging the mouse over the chat area and using the copy shortcut.
- Retrieve the copied text from the clipboard.
- Message Analysis

- Analyze the copied chat history to check if the last message is from a specific user (e.g., "Rohan Das").
- If the last message is from the target user, send the chat history to OpenAI's GPT-3.5-turbo to generate a humorous response.
- Copy the generated response to the clipboard.
- Send Response
- Click on the chat input area and paste the generated response.
- Press 'Enter' to send the response.
- Wait for a brief period to ensure the application is open and ready for interaction.
- Chat History Retrieval
- Retrieve the copied text from the clipboard.
- Message Analysis
- Analyze the copied chat history to check if the last message is from a specific user (e.g., "Rohan Das").
- Generate Response
- Copy the generated response to the clipboard.
- Send Response

LIBRARIES USED

1. pyautogui: For automating mouse and keyboard interactions.
2. time: For adding delays between operations.
3. pyperclip: For clipboard operations.
4. openai: For interacting with OpenAI's GPT-3.5-turbo model.