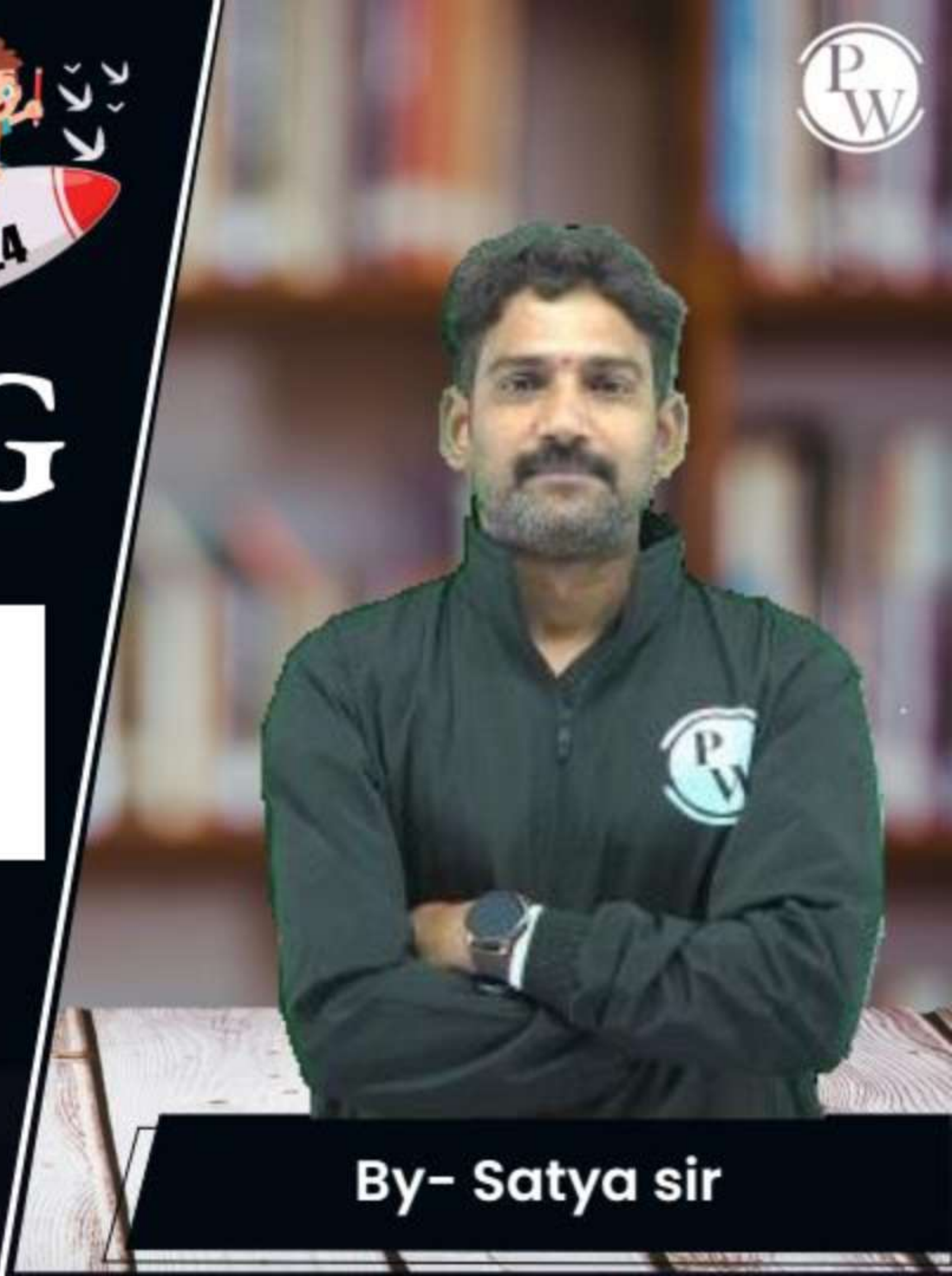
CS & IT ENGINEERING

Data Structures Through Python

STACK

Lecture No.- 03



By- Satya sir

Recap of Previous Lecture

- Applications of Stack (LIFO)
- Expression Conversion
    - Arithmetic Expression
        - infix Notation $a+b$
        - Prefix Notation $+ab$
        - Postfix Notation $ab+$
    - Infix to Postfix Notation

Topics to be Covered

Infix to Prefix conversion

- Procedure
- Examples
- Python Code

Topic : Homework Problem

Convert The given Infix Expression to Postfix Expression

$$X = (a + b - (c / d * (e \wedge f \wedge g) / h) \wedge (i * j + k))$$

Stack

$\cancel{(}$	$+$	$-$	$\cancel{(}$	$/$	$*$	$($	$\wedge$	$\wedge$	$\cancel{(}$	$/$	$\cancel{)}$	$\wedge$	$\cancel{(}$	$*$	$+$	$\cancel{)}$	$\cancel{)}$
	Pop	Pop		Pop	Pop		Pop	Pop		Pop		Pop		Pop	Pop		

$$Y : ab+cd/ef\wedge g\wedge *h/ij*k+\wedge -$$

Topic : Expression Conversion - 2

① ∧ ② *, / ③ +, −

Infix to Prefix conversion

PROCEDURE :

1) Let 'X' be infix Expression, 'Y' be intermediate Expression, 'Z' be Resultant Prefix expression, 'S' be Empty stack.

② Scan 'X' from Right to Left, ')' == '(' and '(' == ')'

③ Perform Infix to Postfix Conversion Procedure, Except the following:
If Top operator (OP_T) > scanned operator (OP_S) : Pop OP_T, append to Y, Push OP_S
If OP_T <= OP_S, Push OP_S onto stack, S.

④ Reverse Y (Intermediate Exp) == Resultant Prefix Expression, Z.

Topic : Expression Conversion - 2

Example - 1

Convert Infix Exp to Prefix Exp.

X : A + B * (C − D ∧ E ∧ F) / G * (H − I) / J

Stack :

/	$\cancel{(}$	−	)	*	/	$\cancel{(}$	∧	∧	−	)	*	+
Pop		Pop		Pop	Pop		Pop	Pop	Pop		Pop	Pop

Intermediate Expression (Y) : JIH−GFED∧∧C−B*/*/A+

$$\boxed{Z : +A/*/*B-C\wedge\wedge DEFG-HIJ}$$

Topic : Expression Conversion - 2

Example-2 : Convert infix Exp to Prefix Exp.

X : $p+(q*r)*(s-t)\wedge(u/v)\wedge w\wedge(x/y*z)$

Stack	(	*	/	)	^	^	(	/	)	^	(	-	)	*	(	*	)	+
		Pop	Pop		Pop	Pop		Pop		Pop		Pop		Pop		Pop		Pop

Y : $zyx/*wvu/ts-\wedge\wedge\wedge rq**p+$

$$\boxed{Z : +p**qr\wedge\wedge\wedge-st/uvw*/xyz}$$

Topic : Expression Conversion - 2

Prefix to infix
Prefix to Postfix
Postfix to infix
Postfix to Prefix

uses Operand stack

Example-3 Infix Expression : 3+5^2-(4^1+9)*7/6

Postfix Expression

Stack	+	^	-	(	^	+	)	*	/
	Pop	Pop	Pop		Pop	Pop		Pop	Pop

Y: 3 5 2 ^ + 4 1 ^ 9 + 7 * 6 / -

** NOTE: Infix to Prefix | Postfix Conversion make use of Operator Stack

Prefix Expression

Stack :	/	*	(	+	^	)	-	^	+
	Pop	Pop		Pop	Pop		Pop	Pop	Pop

Y: 6 7 9 1 4 ^ + * / 2 5 ^ 3 + -

Z: - + 3 ^ 5 2 / * + ^ 4 1 9 7 6

Topic : Expression Conversion - 2

```
#Python Code to convert infix to Prefix Expression

def isOperator(c):          Not a small letter
   return (not (c >= 'a' and c <= 'z') and
            not(c >= '0' and c <= '9') and not(c >= 'A' and c <= 'Z'))
                Not a digit                         Not a Big letter

def getPriority(C):
   if (C == '-' or C == '+'):
      return 1
   elif (C == '*' or C == '/'):
      return 2
   elif (C == '^'):
      return 3
   return 0

         3==High Priority, 1==Low Priority
```

```
def infixToPrefix(infix):
   operators = []
   operands = []

   for i in range(len(infix)):

      if (infix[i] == '(' ):
         operators.append(infix[i])

      elif (infix[i] == ')'):
         while (len(operators)!=0 and (operators[-1] != '(' )):
            op1 = operands[-1]   Top operator
            operands.pop()
            op2 = operands[-1]   Next Top
            operands.pop()
            op = operators[-1]
            operators.pop()
            tmp = op + op2 + op1
            operands.append(tmp)
         operators.pop()
      elif (not isOperator(infix[i])):
         operands.append(infix[i] + "")
```

Topic : Expression Conversion - 2

```
else:
   while (len(operators)!=0 and getPriority(infix[i]) <= getPriority(operators[-1])):

op1 = operands[-1]
            operands.pop()

            op2 = operands[-1]
            operands.pop()

            op = operators[-1]
            operators.pop()

            tmp = op + op2 + op1
            operands.append(tmp)
         operators.append(infix[i])
```

```
while (len(operators)!=0):
      op1 = operands[-1]
      operands.pop()

      op2 = operands[-1]
      operands.pop()

      op = operators[-1]
      operators.pop()

      tmp = op + op2 + op1
      operands.append(tmp)
   return operands[-1]

while(1):
   s = input("Infix Expression : ")
   print("Prefix Expression : ", infixToPrefix(s))
```

2 mins Summary

H/w Problem

Convert infix Expresion to Prefix and Postfix Exp.

Infix Exp : $a \wedge b \wedge c \wedge d * e / f * g + h * i \wedge j$

THANK - YOU