```
def Operator(operand_stack, operator_stack):
    ans = 0
    for i in range(len(operator_stack.stack)):
        a = operand_stack.POP()
        b = operand_stack.POP()
        c = operator_stack.POP()
        if c == "+":
            ans = int(b) + int(a)
            operand_stack.PUSH(ans)
        elif c == "-":
            ans = int(b) - int(a)
            operand_stack.PUSH(ans)
        elif c == "*":
            ans = int(b) * int(a)
            operand_stack.PUSH(ans)
        elif c == "/":
            ans = int(b) / int(a)
            operand_stack.PUSH(ans)
        elif c == "^":
            ans = int(b) ** int(a)
            operand_stack.PUSH(ans)
    return ans


def Evaluating_Infix_Expression(InfixExpression=""):
    operand_stack = STACK()
    operator_stack = STACK()
    for operating in range(len(InfixExpression)):
        if InfixExpression[operating].isnumeric():
            operand_stack.PUSH(InfixExpression[operating])
    for operating in range(len(InfixExpression)):
        if InfixExpression[operating] == "+" or InfixExpression[operating] == "-" or \
                InfixExpression[operating] == "*" or InfixExpression[operating] == "/" or InfixExpression[operating] == "^":
            operator_stack.PUSH(InfixExpression[operating])
    val = Operator(operand_stack, operator_stack)
    return val


inf = "1-2+3-4"
Infix_stack = STACK()
print("Infix expression evaluation : ", Evaluating_Infix_Expression(inf))
```

```
py
Infix expression evaluation :  0
>>>
```