Day 5

Intin and posttin :-

Intin expression :-

The expression of from a op b. when an operator is in-between every pair of operands. (Eg 1+2)

posttin expression :-

The expression of form a b op. when an operator is followed for every pair of operands. (Eg. 1 2 +)

Priorities of operators :-

Rules

^ => highest priority

/, * => next priority

+, - => lowest priority

* No two operators of same priority can stay together in stack column

* lowest priority cannot be placed before highest priority.

* If operator come between left and right prarthesis pop that.

Example :- (A+B/C*(D+E)-F)

Symbol	Stack	postfix
(	(	
A	(	A
+	(+	A
B	(+	AB
/	(+/	AB
C	(+/	ABC
*	(+*	ABC/
(	(+*(	ABC/
D	(+*(	ABC/D
+	(+*(+	ABC/D
E	(+*(+	ABC/DE
)	(+*(+)	ABC/DE+
-	(-	ABC/DE+*+
F	(-	ABC/DE+*+F
)	(	ABC/DE+*+F-

```
class Conversion:
    def __init__(self, capacity):
        self.capacity = capacity
        self.top = -1
        self.stack = []
        self.postfin = []
        self.priority = {'+':1, '-':1, '*':2, '/':2,
                         '^':3}

    def isempty(self):
        return True if self.top == -1 else False

    def pop(self):
        if not self.isempty():
            self.TOP -=1
            return self.Stack.pop()

        else:
            return "$"

    def peek(self):
        return self.Stack[-1]

    def push(self, op):
        self.top +=1
        self.stack.append(op)
```

```
def isoperand(self.ch):
    return ch.isalpha

def notgrater(self, i):
    try:
        a = self.priority[i]
        b = self.priority[self.peek()]
        return True if a<=b else False
    except KeyError:
        return False

def intintopostfin(self, exp):
    for i in exp:
        if self.isoperand(i):
            self.postfin.append(i)
        elif i=='(':
            self.push(i)
        elif i==')':
            while (not self.isempty()) and
                                self.peek()!='('):
                a = self.pop()
                self.postfin.append(a)
```

```
if (not self.isempty() and self.peek() != '('):
        return -1
else:
    self.pop()

elif: else:
        while ((not self.isempty()) and self.notgreate
                                              (i)):
            self.posttfin.append(self.pop())
        self.push(i)
while not self.isempty():
    self.posttfin.append(self.pop())
print("".join(self.posttfin))
```

input :-

```
exp = "a+b*(c^d-e)^(f+g*h)-9"
obj = conversion(len(exp))
obj.infintopostfin(exp)
```

output :-

abcd^e-fgh*+^*+i-

Given a string reverse it using stack:

```
def createstack():
    stack = []
    return stack

def size(stack):
    return len(stack)

def isempty(stack):
    if size(stack) == 0:
        return true

def push(stack, item):
    stack.append(item)

def pop(stack):
    return stack.pop()

def reverse(string):
    n = len(string)
    stack = createstack()
    for i in range(0, n):
        push(stack, string[i])
```

```
String = ""
for i in range(0, n, 1):
    String += pop(stack)
return String
```

~~Str~~i

input :

```
String = "Annamalai"
String = reversed(string)
print(string)
```

output :-

ialamannA