Kalyani Government Engineering College
Department of Computer Application

Kalyani Government Engineering College Department of Computer Application Data Structure through Python – MCAN203, Year: 2020-2021 Assignment: 2

2a. Write a Python program to implement the basic operations of Stack. Write the corresponding algorithm.

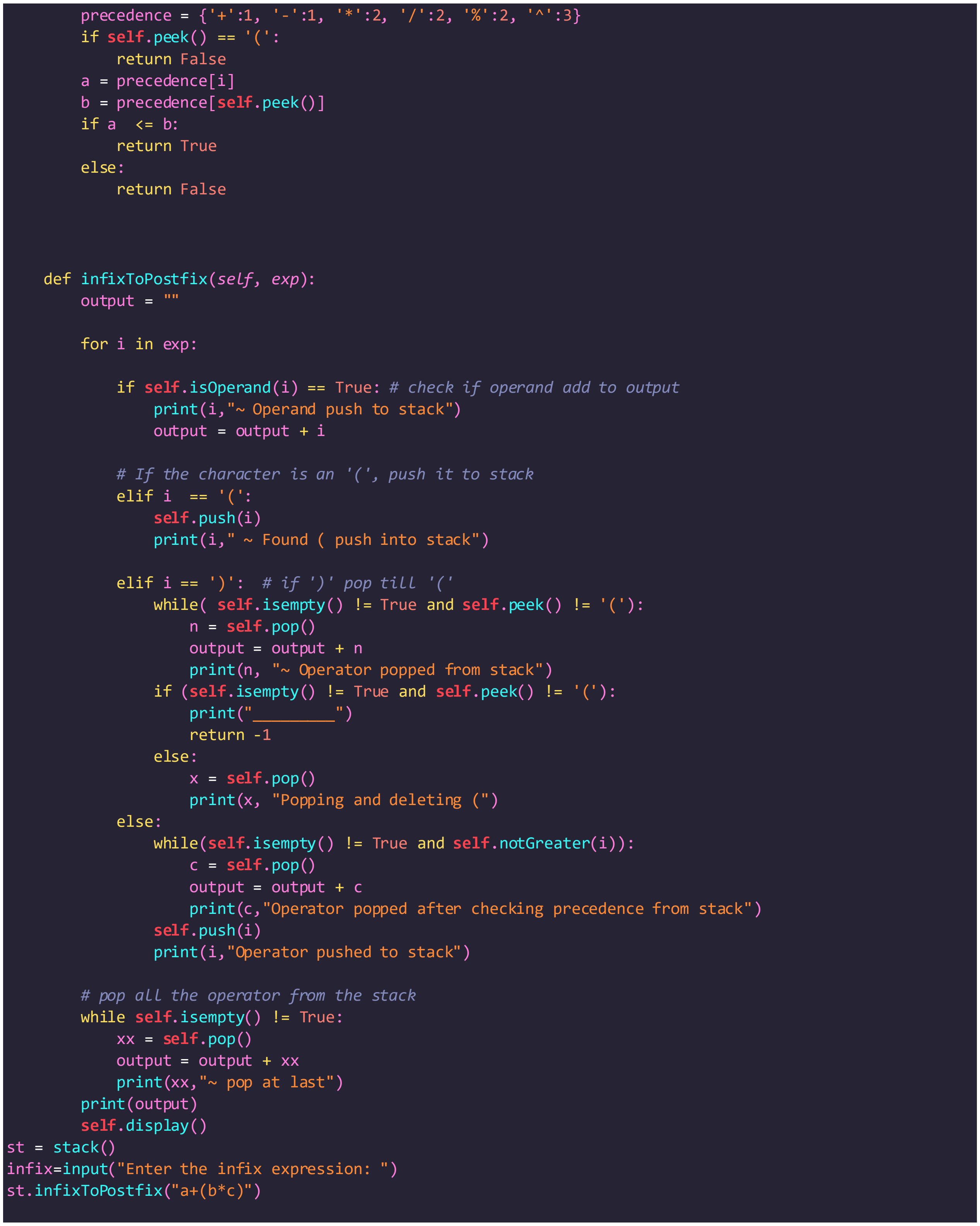
Sol:

```python
class Stack:

    top = -1

    size = 0

    s = []

    def __init__(self, size):

        self.size = size -1

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

    def isfull(self):
        if self.top == size:
            return True
        return False

    def push(self, data):

        if self.top >= self.size:

            print("Stack Overflow!")

        else:

            self.top += 1

            self.s.append(data)


    def pop(self):

        if self.top < 0:

            print("Stack Underflow!")

            return False

        else:

            data = self.s[self.top]

            del self.s[self.top]

            self.top -= 1

            return data
    def display(self):
        if self.top < 0:
            print("Stack Underflow!")
            return False
        else:
            for i in range(self.top , -1, -1):
                print(f"|_{self.s[i]}_|")



n = int(input("Enter the size of stack: "))

s = Stack(n)

flag = True
print("Menu")
print("1.Push")
print("2.Pop")
print("3.Display")
print("4.Exit")

while flag:
    opt = int(input("Enter your option: "))
    if opt == 1:
        data = int(input("Enter the element: "))
        s.push(data)
    elif opt == 2:
        data = s.pop()
        if data != False:
            print(f"Popped element is {data}")
    elif opt == 3:

        s.display()



    elif opt == 4:

        flag = False
```

2b. Write a Python program which takes a postfix expression as argument and evaluate it using Stack. Write the corresponding algorithm.

Sol:

```
class stack:
    def __init__(self):
        self.item = []

    def push(self,it):
        self.item.append(it)
    def peek(self):
        if self.isempty() == True:
            return 0
        return self.item[-1]

    def pop(self):
        if self.isempty() == True:
            return 0
        return(self.item.pop())

    def length(self):
        return (len(self.item))


    def isempty(self):
        if self.item == []:
            return True
        else:
            return False

    def display(self):
        if self.isempty()== True:
            return
        temps = stack()
        while(self.isempty() != True):
            x = self.peek()
            print("~",x)
            temps.push(x)
            self.pop()
        while(temps.isempty() != True):
            x = temps.peek()
            self.push(x)
            temps.pop()


    def isOperand(self, ch):
        return ch.isalpha()
    def notGreater(self, i):
```



```
        precedence = {'+':1, '-':1, '*':2, '/':2, '%':2, '^':3}
        if self.peek() == '(':
            return False
        a = precedence[i]
        b = precedence[self.peek()]
        if a  <= b:
            return True
        else:
            return False



    def infixToPostfix(self, exp):
        output = ""

        for i in exp:

            if self.isOperand(i) == True: # check if operand add to output
                print(i,"~ Operand push to stack")
                output = output + i

            # If the character is an '(', push it to stack
            elif i  == '(':
                self.push(i)
                print(i," ~ Found ( push into stack")

            elif i == ')':  # if ')' pop till '('
                while( self.isempty() != True and self.peek() != '('):
                    n = self.pop()
                    output = output + n
                    print(n, "~ Operator popped from stack")
                if (self.isempty() != True and self.peek() != '('):
                    print("_________")
                    return -1
                else:
                    x = self.pop()
                    print(x, "Popping and deleting (")
            else:
                while(self.isempty() != True and self.notGreater(i)):
                    c = self.pop()
                    output = output + c
                    print(c,"Operator popped after checking precedence from stack")
                self.push(i)
                print(i,"Operator pushed to stack")

        # pop all the operator from the stack
        while self.isempty() != True:
            xx = self.pop()
            output = output + xx
            print(xx,"~ pop at last")
        print(output)
        self.display()
st = stack()
infix=input("Enter the infix expression: ")
st.infixToPostfix("a+(b*c)")
```