Infix Conversions

1. You are given an infix expression.
2. You are required to convert it to postfix and print it.
3. You are required to convert it to prefix and print it.
Constraints

1. Expression is balanced
2. The only operators used are +, -, , /
3. Opening and closing brackets - () - are used to impact precedence of operations
4. + and - have equal precedence which is less than * and /. * and / also have equal precedence.
5. In two operators of equal precedence give preference to the one on left.
6. All operands are single digit numbers.

Sample Input

*a(b-c+d)/e
**

Sample Output

*abc-d+e/
*/a+-bcde

```
def infixEvaluation(string):
    pre = []
    post = []
    signs = []
    for i in range(len(string)):
        ch = string[i]
        if ch == "(":
            signs.append(ch)
        elif ch in {'o', 'g', 'q', 'k', 'b', 'u', 'x', 'n', 'j', 'r', 'l',
'p', 'v', 'e', 'f', 'd', 's', 'y', 'a', 'i',
                    't', 'h', 'w', 'z', 'm', 'c'}:
            pre.append(ch)
            post.append(ch)
        elif ch == ')':
            while signs[-1] != '(':
                op = signs.pop()
                prev2 = pre.pop()
                prev1 = pre.pop()
```

```
                temp1 = op+prev1+prev2
                pre.append(temp1)

                postv2 = post.pop()
                postv1 = post.pop()
                temp2 = postv1+postv2+op
                post.append(temp2)
            signs.pop()
        elif ch in {'+', '-', '*', '/'}:
            while len(signs) > 0 and signs[-1] != '(' and
precedence(signs[-1]) >= precedence(ch):
                op = signs.pop()
                prev2 = pre.pop()
                prev1 = pre.pop()
                temp1 = op + prev1 + prev2
                pre.append(temp1)

                postv2 = post.pop()
                postv1 = post.pop()
                temp2 = postv1 + postv2 + op
                post.append(temp2)
            signs.append(ch)

    while len(signs) > 0:
        op = signs.pop()
        prev2 = pre.pop()
        prev1 = pre.pop()
        temp1 = op + prev1 + prev2
        pre.append(temp1)

        postv2 = post.pop()
        postv1 = post.pop()
        temp2 = postv1 + postv2 + op
        post.append(temp2)
    return pre[-1],post[-1]


# Checks Precedence of operator.    "/"="*" > "+" = "-"
def precedence(operator):
    if operator == "+" or operator == "-":
        return 1
    else:
        return 2
```

```
# For evaluating the value of 2 operand and 1 operator
def eval(v1, v2, operator):
    if operator == '+':
        return v1 + v2
    elif operator == "-":
        return v1 - v2
    elif operator == "*":
        return v1 * v2
    else:
        return v1 // v2


string = "a*(b-c+d)/e"
print(infixEvaluation(string))

# print(set('abcdefghijklmnopqrstuvwxyz'))
```

This doesn't follow bodmass rule. Be aware.

```
def convertInfix(s):
    prefix = []
    postfix = []
    symbols = []
    nums = {'1', '2', '3', '4', '5', '6', '7', '8', '9'}

    for i in range(len(s)):
        char = s[i]
        if char == '(':
            symbols.append(char)
        elif char.isalnum():
            prefix.append(char)
            postfix.append(char)
        elif char == ')':
            while len(symbols) > 0 and symbols[-1] != '(':
                v2Pre = prefix.pop()
                v1Pre = prefix.pop()
                v2Pos = postfix.pop()
                v1Pos = postfix.pop()
                op = symbols.pop()
                res1 = preEval(v1Pre, v2Pre, op)
                res2 = postEval(v1Pos, v2Pos, op)
                prefix.append(res1)
```

```
                postfix.append(res2)
            symbols.pop()
        elif char in {'+', '-', '*', '/'}:
            while len(symbols) and symbols[-1] != '(' and
getPrecedence(char) <= getPrecedence(symbols[-1]):
                v2Pre = prefix.pop()
                v1Pre = prefix.pop()
                v2Pos = postfix.pop()
                v1Pos = postfix.pop()
                op = symbols.pop()
                res1 = preEval(v1Pre, v2Pre, op)
                res2 = postEval(v1Pos, v2Pos, op)
                prefix.append(res1)
                postfix.append(res2)
            symbols.append(char)

    while len(symbols) > 0:
        v2Pre = prefix.pop()
        v1Pre = prefix.pop()
        v2Pos = postfix.pop()
        v1Pos = postfix.pop()
        op = symbols.pop()
        res1 = preEval(v1Pre, v2Pre, op)
        res2 = postEval(v1Pos, v2Pos, op)
        prefix.append(res1)
        postfix.append(res2)
    return prefix[0], postfix[0]


def getPrecedence(char):
    if char in {'+', '-'}:
        return 1
    else:
        return 2


def preEval(v1, v2, char):
    return char + v1 + v2


def postEval(v1, v2, char):
    return v1 + v2 + char
```

```
s = 'a*(b-c)/d+e'
print(convertInfix(s))
```