计算器的功能实现是栈的典型应用，因为在得到一个运算符之后要得到前一个相邻的字符，和栈的先进后出性质相符。

普通计算器需要满足的功能是：

1. 输入一个字符串包括+-\*/、数字、空格和括号，返回运算结果
2. 符合运算法则
3. 除号是整数除法
4. 假定输入算式合法

**首先**处理加减法。以1-2+3为例，一个很简单的思路是：

1. 先给第一个数字加一个默认符号+，变成+1-2+3
2. 把一个运算符和一个数字组合成一对，变成+1，-2，+3转为数字后放入栈中
3. 对栈中数字求和得到结果

算法的核心思想是使用一个字符记录上一个出现的字符，在遇到下一个字符时，根据上一个字符的信息记录当前得到的结果放入到栈中

**然后**处理乘除法：在上述加法的基础上，如果是乘除法，则取出栈顶数字与当前数字做运算，并将结果放回到栈顶。

最后处理括号，括号表示内部优先运算，对应的子字符串依然可以使用上述的方法计算，所以是一个递归的过程。在遇到左括号时，递归处理直到遇到右括号结束。

代码如下;

def calculate(s: str) -> int:

def helper(s: List) -> int:

stack = []

sign = '+'

num = 0

while len(s) > 0:

c = s.pop(0)

if c.isdigit():

num = 10 \* num + int(c)

if (not c.isdigit() and c != ' ') or len(s) == 0:

if sign == '+':

stack.append(num)

elif sign == '-':

stack.append(-num)

elif sign == '\*':

stack[-1] = stack[-1] \* num

elif sign == '/':

# python 除法向 0 取整的写法

stack[-1] = int(stack[-1] / float(num))

num = 0

sign = c

return sum(stack)

# 需要把字符串转成列表方便操作

return helper(list(s))

带括号版本

def calculate(s: str) -> int:

def helper(s: List) -> int:

stack = []

sign = '+'

num = 0

while len(s) > 0:

c = s.pop(0)

if c.isdigit():

num = 10 \* num + int(c)

# 遇到左括号开始递归计算 num

if c == '(':

num = helper(s)

if (not c.isdigit() and c != ' ') or len(s) == 0:

if sign == '+': ...

elif sign == '-': ...

elif sign == '\*': ...

elif sign == '/': ...

num = 0

sign = c

# 遇到右括号返回递归结果

if c == ')': break

return sum(stack)

return helper(list(s))