前缀表达式

前缀表达式:又称为波兰表达式

1)前缀表达式又称波兰式，前缀表达式的运算符位于操作数之前

2) 举例说明: (3 + 4)* 5 -6 对应的前缀表达式就是- * + 3 4 5 6

上面内容的 (3 + 4)* 5-6 是中缀表达式，它的前缀表达式表示为: -* + 34 5 6注

意表示直接从中缀表示符号顺序移动过来，即不是 + * 这个原因是:

前缀表达式的计算机求值

- 从右至左扫描表达式，遇到数字时，将数字压入堆栈
- 遇到运算符时，弹出栈顶的两个数，用运算符对它们做相应的计算(栈项元素和次顶元素)，并将结果入栈。
- 重复上述过程直到表达式最左端，最后运算得出的值即为表达式的结果

例如:(3+4)X5-6对应的前缀表达式就是-x +3456,针对前缀表达式求值步骤如下:

1) 从右至左扫描，将6、5、4、3压入堆栈

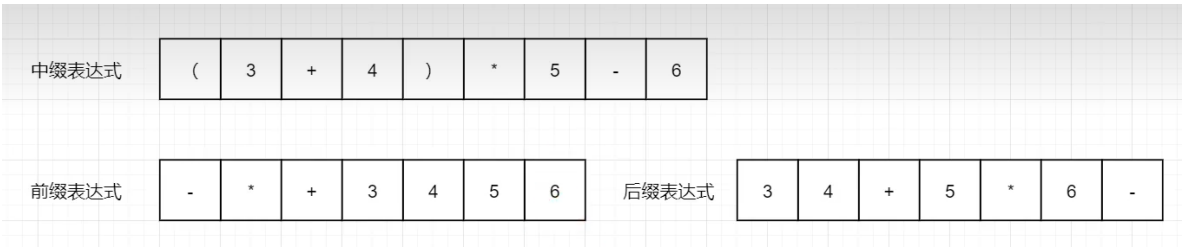
2)遇到+运算符，因此弹出3和4(3为栈项元素，4为次顶元素)，计算出3+4的值，得7再将7入栈

3) 接下来是X运算符，因此弹出7和5，计算出7X5=35，将35入栈

4)最后是-运算符，计算出35-6的值，即29，由此得出最终结果

注意是从右往左扫描，而目不分运算符优先级，只要扫描到运算符，运算符要直接为数栈中弹出的两人元素计算完，然后再把计算结果压入数栈顶部，

数栈计算的顺序永远是 cur num (当前的数)操作 pre num (前一个数)



中缀表达式

中缀表达式:就是我们最熟悉的表达式形式

1)中缀表达式就是常见的运算表达式，如(3+4)X5-6

2)中缀表达式的求值是我们人最熟悉的，但是对计算机来说却不好操作(前面我们进的案例就能看的这个问题)，因此，在计算结果时，往往会将中缀表达式转成其它表达式来操作(一般转成后缀表达式)

后缀表达式

后缀表达式:逆波兰表达式

1)后缀表达式又称逆波兰表达式，与前缀表达式相似，只是运算符位于操作数之后

2)中举例说明: (3+4)x5-6 对应的后缀表达式就是3 4 +5 x 6-

3)再比如:

正常表达式	逆波兰表达式
a+b	a b +
a+(b-c)	a b c - +
a+(b-c)*d	a b c - d * +
a+d*(b-c)	a d b c - * +
a=1+3	a 1 3 + =

后缀表达式的计算过程:

- 从左至右扫描表达式，遇到数字时，将数字压入堆栈
- 遇到运算符时，弹出践顶的两个数，用运算符对它们做相应的计算(次项元素和栈顶元素)，并将结果入栈。
- 重复上述过程直到表达式最右端，最后运算得出的体即为表达式的结果

例如:(3+4)X5-6对应的后缀表达式就是34+5x6号针对后缀表达式求值步骤如下:

1)从左至右扫描，将3和4压入堆栈

2)遇到 +运算符，因此弹出4和3(4为栈顶元素，3为次顶元素)，计算出3 + 4的值，得7，再将7入栈

3)将5入栈

4)接下来是x运算符，因此弹出 5和7，计算出7x 5= 35，将35入栈:

5)将6入栈;

6)最后是-运算符，计算出 35-6的值，即29 由此得出最终结果

注意:是要先求出后缀表达式，然后进行操作的是后缀表达式，因为后缀表达式是从左向右扫描，所以数栈的数值间的操作顺序是pre_num 操作 cur_num 和中缀一样，但和前缀不一样!

中缀表转后缀表达式

(1)具体操作步骤: 后缀表达式适合计算式进行运算，那么获取到它的具体步骤是?

1.初始化两个栈:运算符栈s1和储存中间结果的栈s2:

2.从左至右扫描中缀表达式,

3.遇到操作数时，将其压s2;

4.遇到运算符时，比较其与s1栈顶运算符的优先级:

1、如果 s1 为空，或栈顶运算符为左括号“”，则直接将此运算符入栈

2、否则，若优先级比栈顶运算符的高，也将运算符压入 s1;

3、否则，将 s1 栈顶的运算符弹出并压入到s2中，再次转到(4-1)与1中新的栈顶运算符相比较

5、遇到括号时:

1.如果是左括号“(”，则直接压入 s1

2.如果是右括号""，则依次弹出s1栈顶的运算符，并压入s2，直到遇到左括号为止，此时将这一-对括

号丢弃

6、重复步骤2至5，直到表达式的最右边

7、将s1中剩余的运算符依次弹出并压入s2

8、依次弹出s2中的元素并输出，结果的逆序即为中缀表达式对应的后缀表达式

思路

实现一个逆波兰计算器，主要分两步，第一步是要求出后缀表达式，第二步对后缀表达式进行操作，再者，因为需要栈的功能不像综合计算器那样需要不断的扫描以及注意遇到的运算符后要有处理的先后顺序等等，后缀表达式只要遇到数字就压入，遇到运算符就弹出两位数，进行运算!我们可以用Python中的列表来模拟一个栈即可，**列表的append就是不断的在尾部添加元素，非常像栈的压栈push过程**，因为要后进先出，所以取得时候，我们用列表的pop（）方法，不断的从列表的尾部取元素，**直接用list可以大大降低实现一个栈的复杂度**，因为我们不需要向之前一样用到自定义栈的多个方法，只需要用到栈的“后进先出”的性质!

ASCII 字符代码表 一

高四位 / 低四位		ASCII非打印控制字符											ASCII 打印字符															
		0000					0001						0010		0011		0100		0101		0110		0111					
		0					1						2		3		4		5		6		7					
		+进制	字符	ctrl	代码	字符解释	+进制	字符	ctrl	代码	字符解释		+进制	字符	+进制	字符	+进制	字符	+进制	字符	+进制	字符	+进制	字符	ctrl			
0000	0	0	BLANK NULL	^@	NUL	空	16	▶	^P	DLE	数据链路转意		32		48	0	64	@	80	P	96	`	112	p				
0001	1	1	☺	^A	SOH	头标开始	17	◀	^Q	DC1	设备控制 1		33	!	49	1	65	A	81	Q	97	a	113	q				
0010	2	2	☻	^B	STX	正文开始	18	↕	^R	DC2	设备控制 2		34	"	50	2	66	B	82	R	98	b	114	r				
0011	3	3	♥	^C	ETX	正文结束	19	!!	^S	DC3	设备控制 3		35	#	51	3	67	C	83	S	99	c	115	s				
0100	4	4	♦	^D	EOT	传输结束	20	¶	^T	DC4	设备控制 4		36	$	52	4	68	D	84	T	100	d	116	t				
0101	5	5	♣	^E	ENQ	查询	21	§	^U	NAK	反确认		37	%	53	5	69	E	85	U	101	e	117	u				
0110	6	6	♠	^F	ACK	确认	22	■	^V	SYN	同步空闲		38	&	54	6	70	F	86	V	102	f	118	v				
0111	7	7	•	^G	BEL	震铃	23	↨	^W	ETB	传输块结束		39	'	55	7	71	G	87	W	103	g	119	w				
1000	8	8	◘	^H	BS	退格	24	↑	^X	CAN	取消		40	(	56	8	72	H	88	X	104	h	120	x				
1001	9	9	○	^I	TAB	水平制表符	25	↓	^Y	EM	媒体结束		41	)	57	9	73	I	89	Y	105	i	121	y				
1010	A	10	◙	^J	LF	换行/新行	26	→	^Z	SUB	替换		42	*	58	:	74	J	90	Z	106	j	122	z				
1011	B	11	♂	^K	VT	竖直制表符	27	←	^[	ESC	转意		43	+	59	;	75	K	91	[	107	k	123	{				
1100	C	12	♀	^L	FF	换页/新页	28	∟	^\	FS	文件分隔符		44	,	60	<	76	L	92	\	108	l	124	|				
1101	D	13	♪	^M	CR	回车	29	↔	^]	GS	组分隔符		45	-	61	=	77	M	93	]	109	m	125	}				
1110	E	14	♫	^N	SO	移出	30	▲	^6	RS	记录分隔符		46	.	62	>	78	N	94	^	110	n	126	~				
1111	F	15	☼	^O	SI	移入	31	▼	^-	US	单元分隔符		47	/	63	?	79	O	95	_	111	o	127	Δ	Back space			

注：表中的ASCII字符可以用:ALT + “小键盘上的数字键” 输入

link：

https://github.com/zzqnot996/coding/blob/main/%E6%A1%88%E4%BE%8B-%E9%80%86%E6%B3%A2%E5%85%B0%E8%AE%A1%E7%AE%97%E5%99%A8.ipynb