# 数据结构

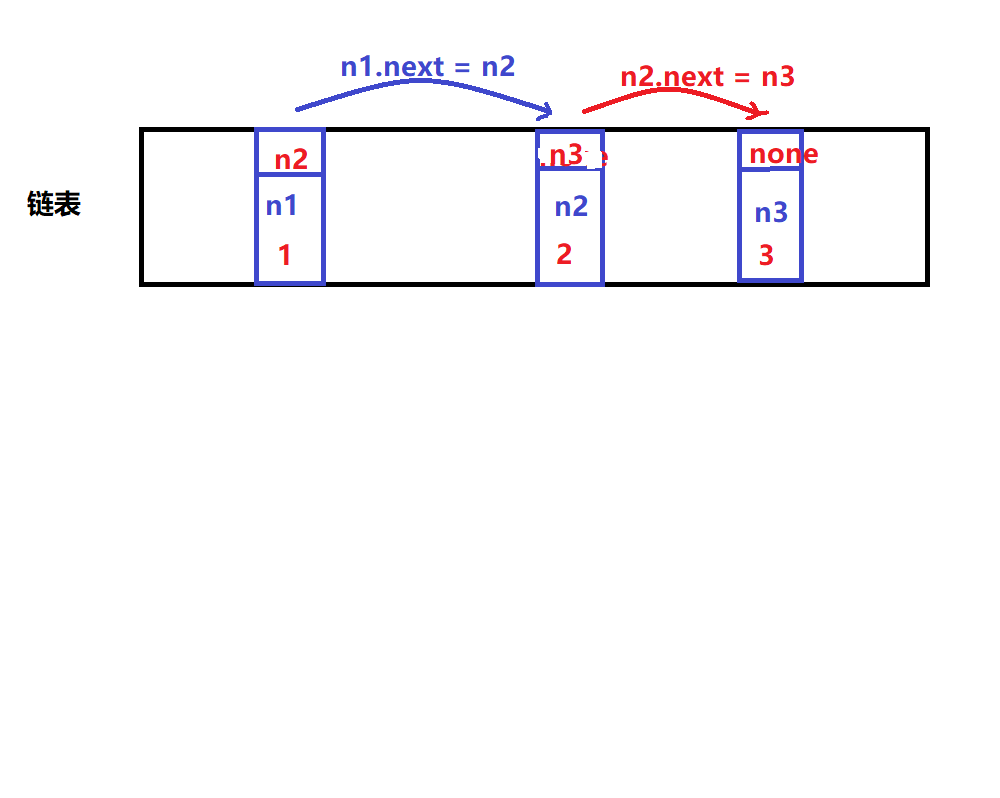
## Day1

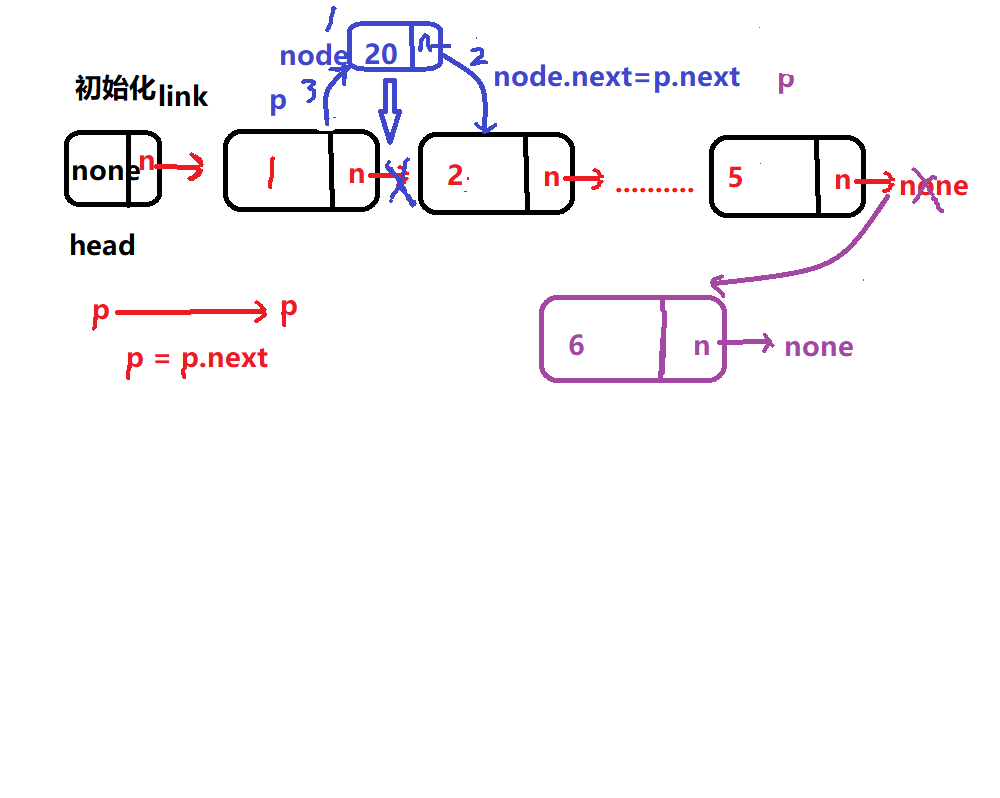
作业 : 1. 学习markdown的使用方法

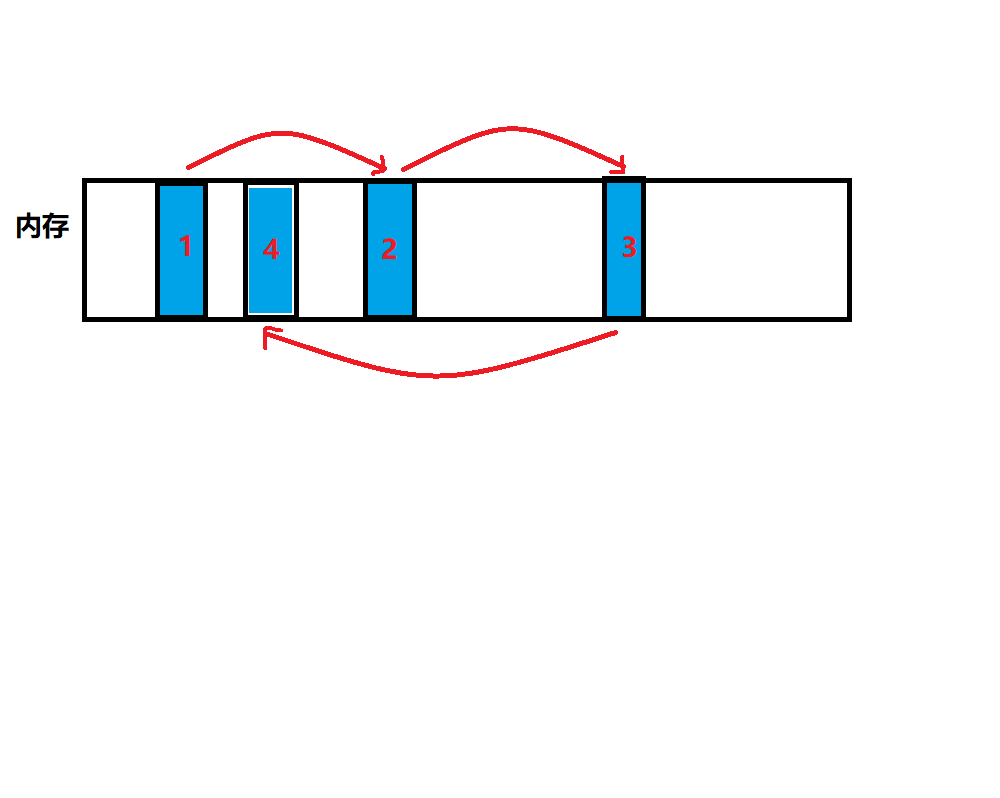
2. 三个数入栈的顺序为1,2,3 那么出栈的顺序不可能是什么

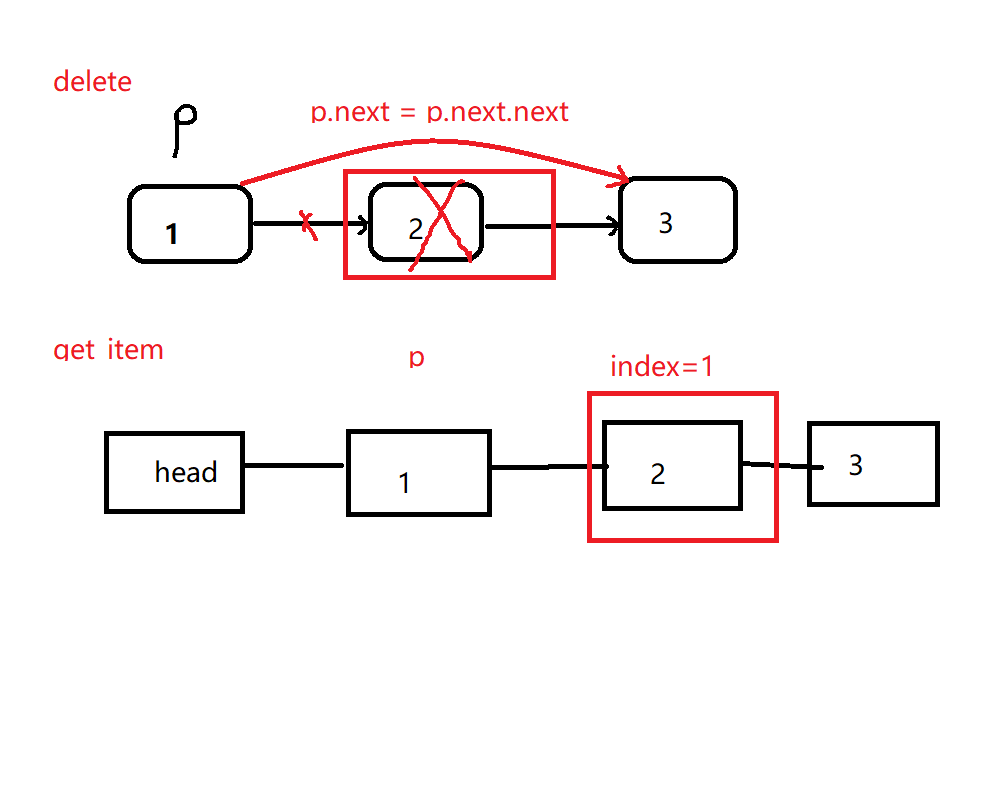
3. 将linklist代码逐个方法走流程

*"""  
linklist.py 链表程序实现  
重点代码  
  
思路分析  
1. 创建节点类，生成节点对象  
　　包含数据和下一个节点的引用  
2. 链表类，生成链表对象  
　　可以对链表进行数据操作  
"""  
  
# 节点类***class** Node:  
 **def** \_\_init\_\_(self, data, next=**None**):  
 self.data = data  
 self.next = next  
  
  
*# 链表类***class** Linklist:  
 *"""  
 建立链表模型  
 进行链表操作  
 """* **def** \_\_init\_\_(self):  
 *"""  
 初始化链表,生成一个头节点,表示链表开始节点  
 """* self.head = Node(**None**)  
  
 *# 初始添加一组链表节点* **def** init\_list(self, list\_):  
 p = self.head *# ｐ为移动变量* **for** i **in** list\_:  
 p.next = Node(i)  
 p = p.next *# ｐ向后移动一个节点  
  
 # 遍历链表* **def** show(self):  
 p = self.head.next *# 第一个有效节点* **while** p **is not None**:  
 print(p.data, end=**' '**)  
 p = p.next  
 print() *# 换行  
  
 # 获取链表长度* **def** get\_length(self):  
 n = 0  
 p = self.head  
 **while** p.next **is not None**:  
 n += 1  
 p = p.next  
 **return** n  
  
 *# 判断链表是否为空* **def** is\_empty(self):  
 **if** self.get\_length() == 0:  
 **return True  
 else**:  
 **return False** *# 清空链表* **def** clear(self):  
 self.head.next = **None** *#　尾部插入节点* **def** append(self,data):  
 node = Node(data) *#　生成新节点* p = self.head  
 **while** p.next **is not None**:  
 p = p.next  
 p.next = node  
  
 *#　选择位置插入节点* **def** insert(self,index,data):  
 **if** index < 0 **or** index > self.get\_length():  
 **raise** IndexError(**"index out of range"**)  
  
 *# 定义ｐ移动到插入位置的前一个* p = self.head  
 **for** i **in** range(index):  
 p = p.next  
  
 node = Node(data) *#　生成节点  
  
 # 插入* node.next = p.next  
 p.next = node  
  
 *# 删除节点* **def** delete(self,data):  
 p = self.head  
 **while** p.next **and** p.next.data != data:  
 p = p.next  
 *#　判断ｗｈｉｌｅ循环结束原因* **if** p.next **is None**:  
 **raise** ValueError(**"value is error"**)  
 **else**:  
 p.next = p.next.next  
  
 *# 获取节点值* **def** get\_item(self,index):  
 **if** index < 0 **or** index >= self.get\_length():  
 **raise** IndexError(**"index out of range"**)  
 p = self.head.next  
 **for** i **in** range(index):  
 p = p.next  
 **return** p.data  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 *# 链表对象　　  
 # link-->head  
 # link.head --> data == None  
 # link.head --> next == None* link = Linklist()  
 l = [1, 2, 3, 4, 5]  
 link.init\_list(l)  
 link.show() *#　遍历数据* print(**"Link length:"**,link.get\_length())  
 link.append(6)  
 link.insert(4,100)  
 link.show()  
 link.delete(3)  
 link.show()  
 print(link.get\_item(3))









## Day2

1. 什么是数据结构

程序 = 数据结构 + 算法

2. 逻辑结构和存储结构

逻辑结构 : 线性结构 , 树形结构

存储结构 : 顺序存储 , 链式结构

3. 链表的编写

\* 节点类

\* 链表操作类

4. 栈模型 : 先进后出

练习 : 一段文字,在文字中可能存在括号配对错误的情况.要求写一段代码,取检测这段文字中有没有括号书写错误. 括号包含 {} [] ()

**from** day2.sstack **import** SStack  
text = **"Open source software is made better when users (can easily) contribute code and [documentation] to fix bugs and add {fea(tures).} Python {strongly} encoura[ges community] (in{vo}lv[em]ent) in improving the software. Learn more about how to make Python better for everyone."**parens = **"{}[]()"** *#　需要验证的字符*left\_parens = **"{[("***#　验证配对是否正确*opposite = {**'}'**:**'{'**,**']'**:**'['**,**')'**:**'('**}  
  
st = SStack() *#　初始化一个栈  
  
# 负责提供遍历到的括号***def** parent(text):  
 *"""  
 遍历字符串,提供括号字符和其位置  
 """  
 #　ｉ记录索引位置* i,text\_len = 0,len(text)  
 **while True**:  
 *# 循环遍历字符串  
 # 到结尾结束，遇到括号提供给ｖｅｒ* **while** i < text\_len **and** text[i] **not in** parens:  
 i += 1  
  
 **if** i >= text\_len:  
 **return  
 else**:  
 **yield** text[i],i  
 i += 1  
  
  
*#　字符是否匹配的验证工作***def** ver():  
 **for** pr, i **in** parent(text):  
 **if** pr **in** left\_parens:  
 st.push((pr,i)) *#　左括号入栈* **elif** st.is\_empty() **or** st.pop()[0] != opposite[pr]:  
 print(**"Unmatching is found at %d for %s"**%(i,pr))  
 **break** *#　ｆｏｒ循环正常结束* **else**:  
 **if** st.is\_empty():  
 print(**"All parentheses are matched"**)  
 **else**:  
 *#　剩下左括号了* p = st.pop()  
 print(**"Unmatching is found at %d for %s"** % (p[1], p[0]))  
  
*# 主程序只负责做括号的验证*ver()

练习 : 基于链式栈,完成一个逆波兰表达式的接口程序,只完成加法减法即可

**from** lstack **import** \*  
  
ls = LStack()  
  
**while True**:  
 exp = input()  
 tmp = exp.split(**' '**)  
 **for** i **in** tmp:  
 **if** i **not in** [**'+'**,**'-'**,**'p'**]:  
 ls.push(float(i)) *#　入栈数字* **elif** i == **'+'**:  
 x = ls.pop()  
 y = ls.pop()  
 ls.push(y + x)  
 **elif** i == **'-'**:  
 x = ls.pop()  
 y = ls.pop()  
 ls.push(y - x)  
 **elif** i == **'p'**:  
 print(ls.top()) *#　查看栈顶元素*

作业: 1. 编写一个函数,传入一个整数,返回该整数的阶乘.

2. 将线性结构代码整理总结(类似操作和存储差别)