数据结构与算法（Python）-03/第4周

北京大学 陈斌

2021.03.30

线下课堂

- 关于线上学习的几个问题
- 本周内容小结
- 问题解答
- 抽象数据类型ADT与Python类定义
- 慕课作业讲解
- 课堂讨论
- 课堂练习

几个课程学习的问题

提问的艺术

学习过程中有问题比没问题好，但不要做伸手党，先努力自己找答案

课程安排相关的，Canvas公告、单元、页面/gis4g/微信群公告；

教学内容相关的，先动手试试，搜索慕课讨论区，搜索微信群；

学习过程中花费的这些时间都不会白费，你会获得知识以外的强大能力。

作业不等于学习

但这些所有事情的前提，是先要看完教学视频

学习是一个过程，作业只是评价点，会做作业并不代表学会

通过教别人加强学习

多参与慕课讨论区的讨论交流

同学的提问一般体现了课程内容的难点

本课中的样例

- **为了聚焦核心算法，课程样例采用简明的约定**

  表达式采用了简明的空格分隔操作数与操作符：A + B * 5

- **在课程样例中并不涉及各种错误的处理**

  特殊的输入格式处理，如：表达式23.45+33.7这样

  由于输入数据引起的错误处理，如：23a+45这样

- **也留给同学们自己完善和扩展的空间**

本周内容小结W03：基本结构（上）

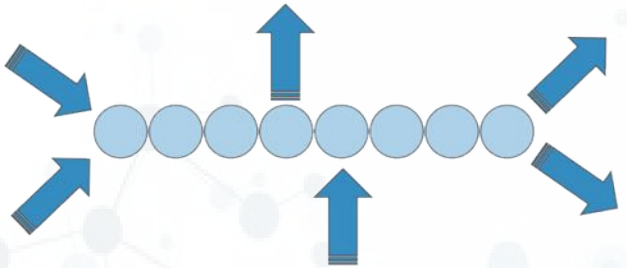
301 什么是线性结构

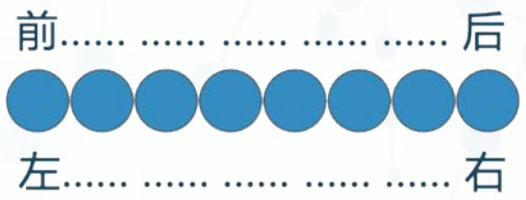
数据项的集合，每个数据项都有唯一的前驱和后继。（不然呢？）

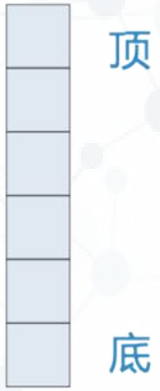
线性结构总有两端。

不同的线性结构差别在于数据项的增减限制

线性结构是应用最广泛的数据结构







本周内容小结W03：基本结构（上）

302 栈抽象数据类型及Python实现

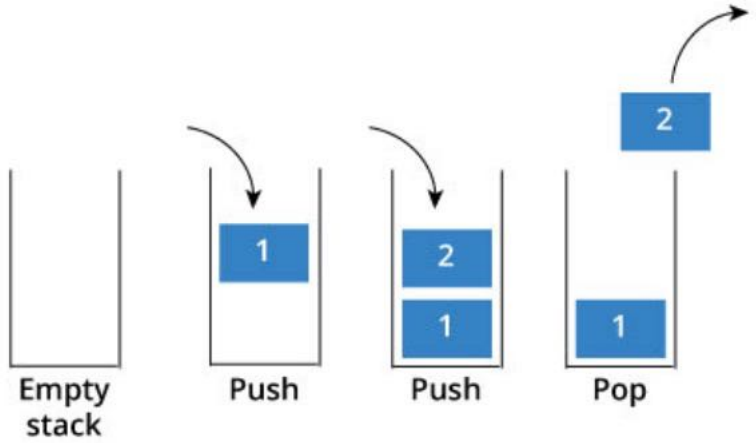
数据项的加入和移除只能发生在一端

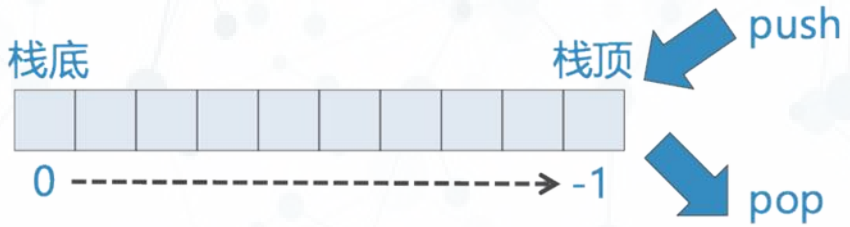
“后进先出LIFO”，具有次序反转的特性

ADT Stack（push, pop, peek, size, isEmpty）

ADT Stack的一种实现：用Python list数据类型来实现

理解ADT Stack不同的实现方法



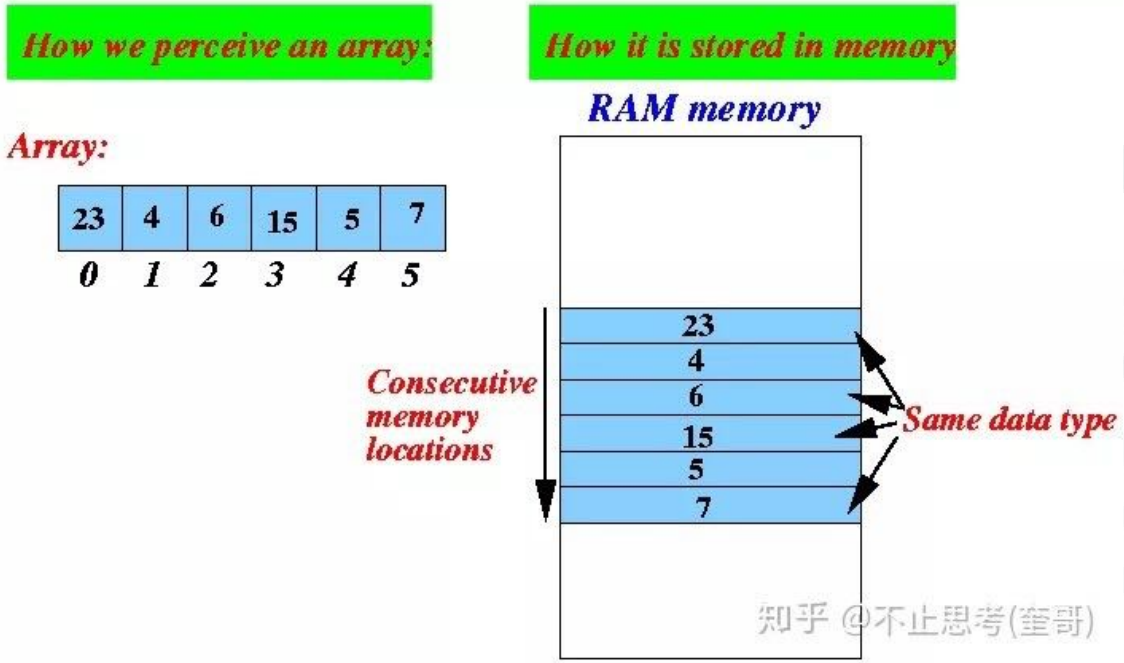


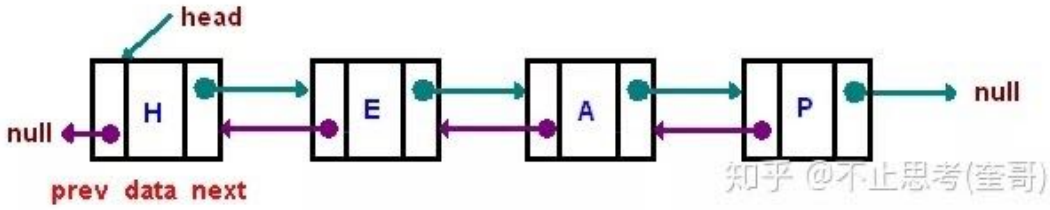
本周内容小结W03：基本结构（上）

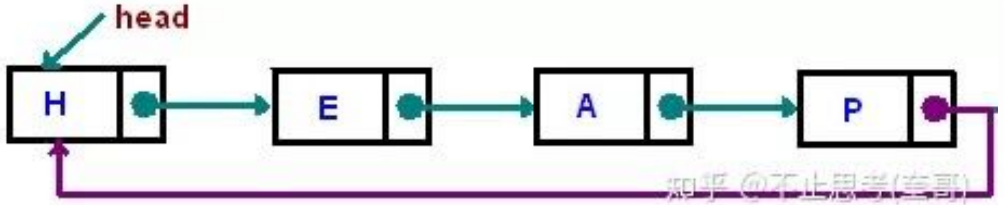
- **303 栈的应用：简单括号匹配**
  次序反转：最早打开的左括号，匹配最后关闭的右括号
- **304 栈的应用：十进制转换为二进制**
  次序反转：最先计算出来的余数是低位，要最后输出
- **305/306 表达式转换**
  次序反转：低优先级的操作符，即使先出现，也要后计算
- **307 后缀表达式求值**
  次序反转：碰到操作符之前，最后出现的数，最先计算

问题解答

› 栈的结构除了本身调用Stack和使用list 还有其他实现方式吗？







问题解答

6 多选 (3分) 以下哪些关于栈的说法是正确的?

- A. 栈的pop操作时间复杂度是O(n)
- B. 括号匹配算法需要栈结构的参与
- C. 在Python中栈结构可以由list来实现 ✔1.00/3.00
- D. 栈的特性是后进先出(LIFO) ✔1.00/3.00

这是第三周测验的一道题，按照分数的分配方式来看，应该有三个正确的选项，那就是A,B中有一个是正确的。

1.（为什么A不正确）

选项中只是一般地说stack的pop的时间复杂度，并没有具体地提及其实践方法，因此实际上其时间复杂度是无法确认的。

2.（为什么B不正确）

问题解答

- **括号匹配演算法 “可以应用” stack的方法来处理，但并不是唯一的办法，因此 "不一定需要" 用stack的办法。**

例子：可以用 queue 的办法，见下：【参考：Check for balanced parentheses in Python - GeeksforGeeks】Approach #2

链接：
https://www.geeksforgeeks.org/check-for-balanced-parentheses-in-python/

```
# Python3 code to Check for
# balanced parentheses in an expression
def check(expression):

    open_tup = tuple('({[')
    close_tup = tuple(')}]')
    map = dict(zip(open_tup, close_tup))
    queue = []

    for i in expression:
        if i in open_tup:
            queue.append(map[i])
        elif i in close_tup:
            if not queue or i != queue.pop():
                return "Unbalanced"
    if not queue:
        return "Balanced"
    else:
        return "Unbalanced"

# Driver code
string = "{[]{()}}"
print(string, "-", check(string))

string = "(((("
print(string,"-",check(string))
```

抽象数据类型ADT

- **抽象数据类型是仅定义了接口的数据类型**
  只说它操作起来会是什么结果
  不说明这些接口具体如何实现
- **栈是个抽象数据类型（ADT Stack）**
- **我们用Python的类定义来实现ADT Stack**
  可以有多种实现方法
  不同实现方法的复杂度不同
  所以不会说ADT Stack的某个接口有时间复杂度

作为子类继承list的一种实现

```
class Stack(list):
    def push(self, item):
        self.append(item)
    def peek(self):
        return self[-1]
    def isEmpty(self):
        return self==[]
    def size(self):
        return len(self)

st = Stack()
```

Python类定义

- **如何定义一个类？怎么就算是个完整的类了？**

  class语句定义类

  只要有个名字，就算是完整的类啦

```
>>> class MyClass:
        pass

>>> a = MyClass()
>>> type(a)
<class '__main__.MyClass'>
```

Python类定义

http://pyln.fun/p/view/4d8c88f1-3904-4a93-a2fd-d545ce276f83/

怎么输出一个类？

确切的说，是把对象内容以字符串形式展现出来

通过定义__str__和__repr__特殊方法

把对象的什么属性，以什么格式展现，你说了算

```
1  class Stack(list):
2      def push(self, item):
3          self.append(item)
4      def peek(self):
5          return self[-1]
6      def isEmpty(self):
7          return self==[]
8      def size(self):
9          return len(self)
10     def __repr__(self):
11         l = len(self) * 7
12         s = "|"+ "-" * l + ")\n|"
13         for a in self:
14             s += "| %-5s" % a
15         s += "\n|" + "-" * l + ")"
16         return s
17     __str__ = __repr__
```

```
20 st = Stack()
21 st.push(45)
22 st.push(56)
23 st.push(123)
24 st.push(999)
25 st.push("yes")
26 print(st)
27
```

Shell

```
>>> %Run mystack.py
  |-------------------------------)
  || 45   | 56   | 123  | 999  | yes
  |-------------------------------)
>>>
```

慕课OJ作业讲解之前

极简class Stack
C & P

```python
class Stack(list):
    def push(self, item):
        self.append(item)
    def peek(self):
        return self[-1]
    def isEmpty(self):
        return self==[]
    def size(self):
        return len(self)

st = Stack()
```

慕课作业讲解：OJ-W03-1有效的括号


```python
st = Stack()
d = {"(":")", "[":"]", "{":"}"}
s= input()
for a in s:
    if a in "({[":
        st.push(a)
    elif st.isEmpty() or not a == d[st.pop()]: # 右括号多出来，或者不匹配
        print("False")
        break
else:
    if st.isEmpty():
        print("True") # 正好匹配
    else:
        print("False") # 左括号多出来
```

慕课作业讲解：OJ-W03-2一维消消乐


```
st = Stack()
s = input()
for a in s:
    if not st.isEmpty() and a==st.peek(): # 相同的就消
        st.pop()
    else:
        st.push(a)
if st.isEmpty():
    print("None")
else:
    print("".join(st))
```

慕课作业讲解：OJ-W03-3洗碗工

不可能的出栈序列

```
st = Stack()
s= input()
n = 0 #正在洗的盘子编号
i = 0 #取盘子的次序，s[i]是取得盘子的编号
while i < 10 and n < 10:
    k = int(s[i])
    # 洗盘子
    # 如果顾客取到了编号k的盘子，那么正在洗的盘子到k之间的所有盘子都洗好叠放
    if n <= k:
        for m in range(n, k+1):
            st.push(m)
            #print("PUSH", m)
        n = k+1 # 正在洗下一个盘子
    # 取盘子
    # 逐个从顶上取盘子，从k开始取，一直取到对不上号，说明要去取的还没洗
    while not st.isEmpty() and st.peek()==int(s[i]):
        m= st.pop()
        #print("POP", m)
        i += 1

# 能取的都取完了，如果盘子堆里还有盘子，说明取的序列不对
if st.isEmpty():
    print("Yes")
else:
    print("No")
```

课堂讨论

- 线性结构的基本特征？
- 栈Stack作为一种特殊的线性结构，有什么进一步的限制？
- 栈Stack主要应用在哪类问题求解？

课堂练习【K03】展示后缀表达式计算过程的栈变化

- 请正确运行后缀表达式求值程序，成功对下列表达式求值，并通过自己定义Stack的__str__和__repr__，来展示计算过程中栈的变化

```
2 3 * 4 +
1 2 + 3 + 4 + 5 +
1 2 3 4 5 * + * +
```

- 请将代码和运行的结果截图做成文档提交。

（pdf, doc, docx, ppt, pptx）

【H2】栈与队列编程作业

- H2-1：中缀表达式求值；
- H2-2：基数排序；
- H2-3： HTML标记匹配；
- H2-4：链表实现栈和队列；
- H2-5：双链无序表。
- DDL是4月14日18:00，不用着急

```
12  # ======= 1 中缀表达式求值 =======
13  # 通过把“中缀转后缀”和“后缀求值”两个算法功能集成在一起（非简单的顺序
14  # 实现对中缀表达式直接求值，新算法还是从左到右扫描中缀表达式，
15  # 但同时使用两个栈，一个暂存操作符，一个暂存操作数，来进行求值。
16  #
17  # 创建一个函数，接受参数为一个字符串，即一个中缀表达式，
18  # 其中每个数字或符号间由一个空格隔开；
19  # 返回一个浮点数，即求值的结果。（支持 + - * / ^ 五种运算）
20  #   其中“ / ”定义为真除True DIV，结果是浮点数类型
21  # 输入样例1:
22  # ( 2 + 3 ) * 6 + 4 / 2
23  # 输出样例1:
24  # 32.0
25  # 输入样例2:
26  # 2 ^ 3 + 4 * 5 - 16 / 2
27  # 输出样例2:
28  # 20.0
29  # 输入样例3:
30  # ( 5 + 1 ) * 2 / 3 - 3 ^ ( 2 + 8 / 4 ) / 9 + 6
31  # 输出样例3:
32  # 1.0

55  def calculate(s) -> float:
56      # 请在此编写你的代码（可删除pass语句）
57      pass
58      # 代码结束
59
60
61  # 调用检验
62  print("======== 1-calculate ========")
63  print(calculate("( 2 + 3 ) * 6 + 4 / 2"))
64  print(calculate("2 ^ 3 + 4 * 5 - 16 / 2"))
65  print(calculate("( 5 + 1 ) * 2 / 3 - 3 ^ (
66
```

下课！