



数据结构与算法（Python版）

栈抽象数据类型及Python实现

陈斌 北京大学 gischen@pku.edu.cn

栈Stack: 什么是栈?

- ❖ 一种有次序的数据项集合，在栈中，数据项的加入和移除都仅发生在同一端
这一端叫栈“顶top”，另一端叫栈“底base”
- ❖ 日常生活中有很多栈的应用
盘子、托盘、书堆等等



栈Stack：什么是栈？

❖ 距离栈底越近的数据项，留在栈中的时间就越长

而最新加入栈的数据项会被最先移除

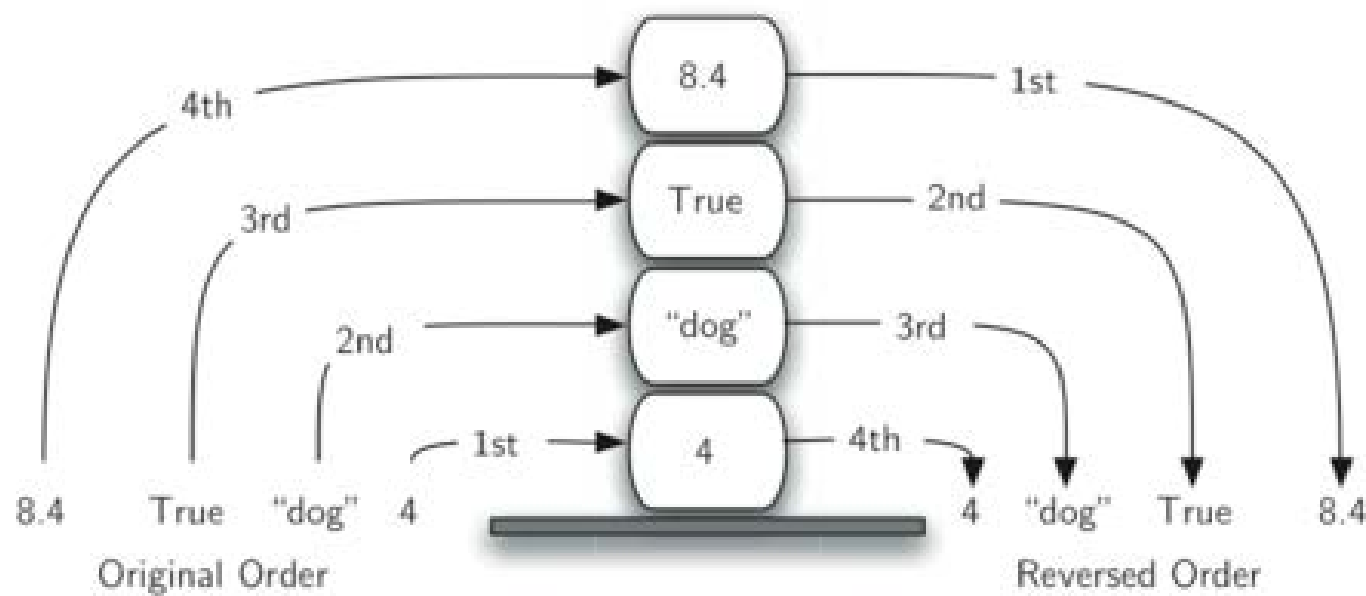
❖ 这种次序通常称为“后进先出LIFO”：
Last in First out

这是一种基于数据项保存时间的次序，时间越短的离栈顶越近，而时间越长的离栈底越近

栈的特性：反转次序

❖ 我们观察一个由混合的python原生数据对象形成的栈

进栈和出栈的次序正好相反



栈的特性：反转次序

❖ 这种访问次序反转的特性，我们在某些计算机操作上碰到过

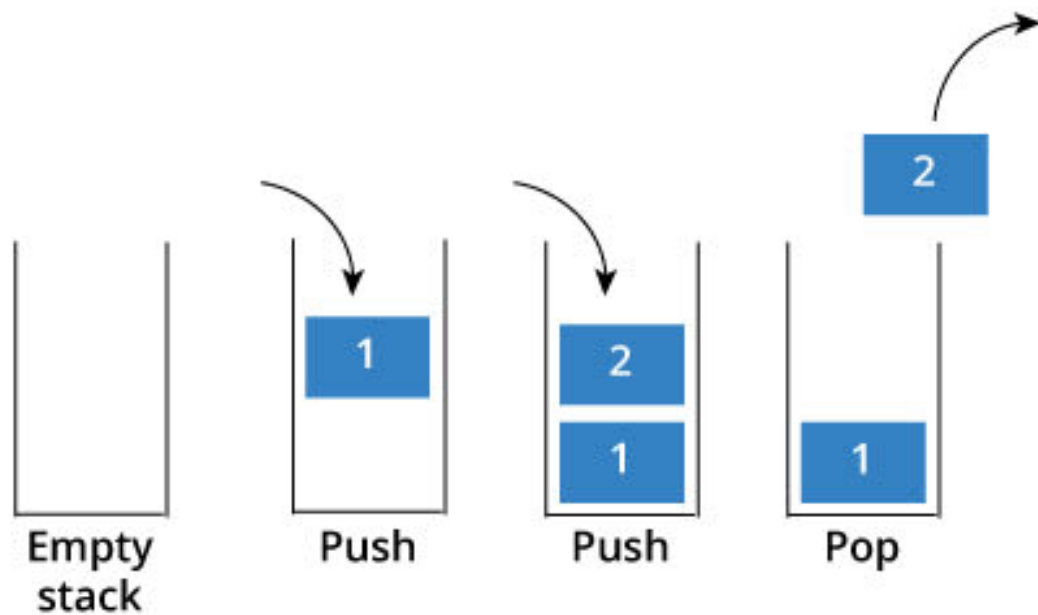
浏览器的“后退back”按钮，最先back的是最近访问的网页

Word的“Undo”按钮，最先撤销的是最近操作



抽象数据类型Stack

- ❖ 抽象数据类型“栈”是一个有次序的数据集，每个数据项仅从“栈顶”一端加入到数据集中、从数据集中移除，栈具有后进先出LIFO的特性



抽象数据类型Stack

❖ 抽象数据类型“栈”定义为如下的操作

Stack(): 创建一个空栈, 不包含任何数据项

push(item): 将item加入栈顶, 无返回值

pop(): 将栈顶数据项移除, 并返回, 栈被修改

peek(): “窥视”栈顶数据项, 返回栈顶的数据项但不移除, 栈不被修改

isEmpty(): 返回栈是否为空栈

size(): 返回栈中有多少个数据项

抽象数据类型Stack：操作样例

Stack Operation	Stack Contents	Return Value
s= Stack()	[]	Stack object
s.isEmpty()	[]	True
s.push(4)	[4]	
s.push('dog')	[4, 'dog']	
s.peek()	[4, 'dog']	'dog'
s.push(True)	[4, 'dog', True]	
s.size()	[4, 'dog', True]	3
s.isEmpty()	[4, 'dog', True]	False
s.push(8.4)	[4, 'dog', True, 8.4]	
s.pop()	[4, 'dog', True]	8.4
s.pop()	[4, 'dog']	True
s.size()	[4, 'dog']	2

用Python实现ADT Stack

- ❖ 在清楚地定义了抽象数据类型Stack之后，我们看看如何用Python来实现它
- ❖ Python的面向对象机制，可以用来实现用户自定义类型
 - 将ADT Stack实现为Python的一个Class
 - 将ADT Stack的操作实现为Class的方法
 - 由于Stack是一个数据集，所以可以采用Python的原生数据集来实现，我们选用最常用的数据集List来实现

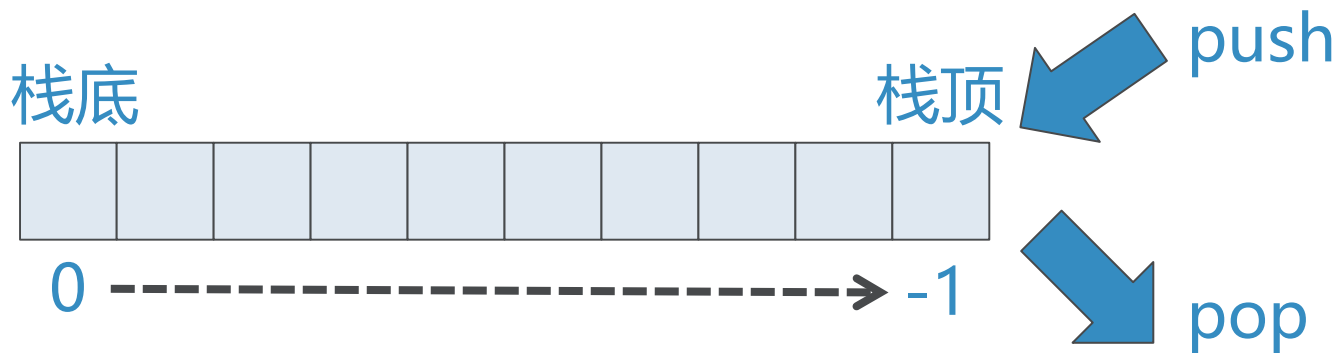
用Python实现ADT Stack

❖ 一个细节：Stack的两端对应list设置

可以将List的任意一端（ $\text{index}=0$ 或者 -1 ）设置为栈顶

我们选用List的末端（ $\text{index}=-1$ ）作为栈顶

这样栈的操作就可以通过对list的append和pop来实现，很简单！



用Python实现ADT Stack

```
class Stack:
    def __init__(self):
        self.items = []

    def isEmpty(self):
        return self.items == []

    def push(self, item):
        self.items.append(item)

    def pop(self):
        return self.items.pop()

    def peek(self):
        return self.items[len(self.items)-1]

    def size(self):
        return len(self.items)
```

课程配套代码：pythonds模块

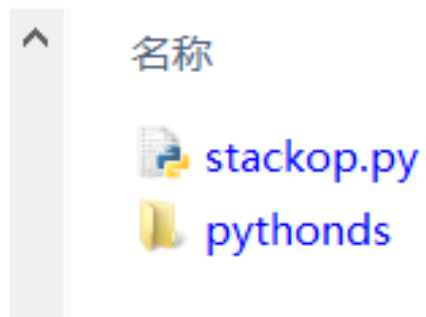
❖ 解包拷贝到练习目录下，与练习程序平级

❖ 调用方法

```
from pythonds.basic.stack import Stack
```

❖ 在与pythonds目录平级的stackop.py文件里面，按照上面的import导入后，就可以直接调用Stack类了

TEMPDATA (K:) ▶ pythontest



Stack测试代码

```
from pythonds.basic.stack import Stack
```

```
s=Stack()
```

```
print(s.isEmpty())
```

```
s.push(4)
```

```
s.push('dog')
```

```
print(s.peek())
```

```
s.push(True)
```

```
print(s.size())
```

```
print(s.isEmpty())
```

```
s.push(8.4)
```

```
print(s.pop())
```

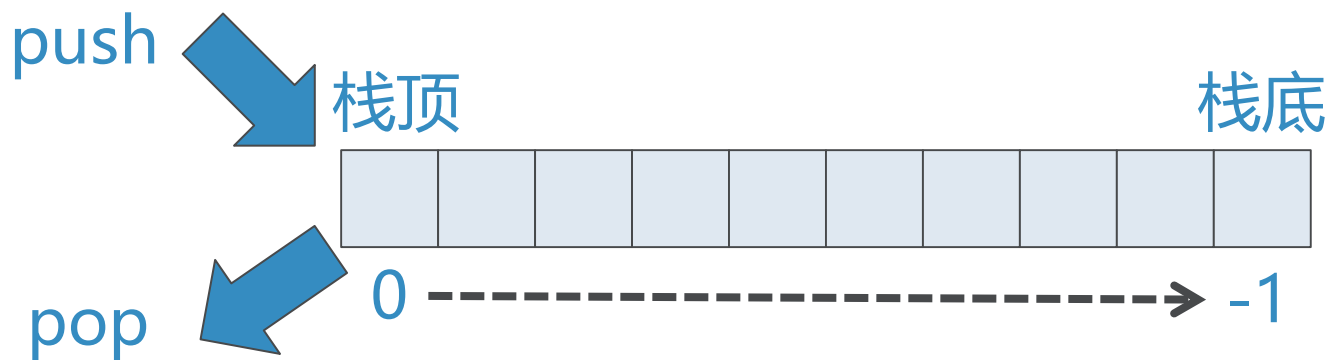
```
print(s.pop())
```

```
print(s.size())
```

```
>>> =====
>>>
True
dog
3
False
8.4
True
2
>>> |
```

ADT Stack的另一个实现

- ❖ 如果我们把List的另一端（首端 $\text{index}=0$ ）作为Stack的栈顶，同样也可以实现Stack



ADT Stack的另一个实现

- ❖ 不同的实现方案保持了ADT接口的稳定性
但性能有所不同，栈顶首端的版本（左），其
push/pop的复杂度为 $O(n)$ ，而栈顶尾端的实现
（右），其push/pop的复杂度为 $O(1)$

```
class Stack:
    def __init__(self):
        self.items = []
```

```
def isEmpty(self):
    return self.items == []
```

```
def push(self, item):
    self.items.insert(0, item)
```

```
def pop(self):
    return self.items.pop(0)
```

```
def peek(self):
    return self.items[0]
```

```
def size(self):
    return len(self.items)
```

```
class Stack:
    def __init__(self):
        self.items = []
```

```
def isEmpty(self):
    return self.items == []
```

```
def push(self, item):
    self.items.append(item)
```

```
def pop(self):
    return self.items.pop()
```

```
def peek(self):
    return self.items[len(self.items)-1]
```

```
def size(self):
    return len(self.items)
```

