Python程序设计

列表 - 基本概念和方法

刘安
苏州大学，计算机科学与技术学院

http://web.suda.edu.cn/anliu/

本节知识点

- range对象
- sum函数
- 列表的构造方法：[], list()函数、列表推导
- 列表的基本操作：索引、分片、连接、重复
- 列表的常用方法
  - 查询类：in、len、min、max、count、index
  - 修改类：append、extend、insert、clear、pop、remove、del、reverse、reversed
- 列表和引用

range

- range对象：不可变的整数序列
- range(stop)： 0, 1, 2, …, stop-1
- range(start, stop)：start, start+1, start+2, …, stop-1
- range(5) : 0, 1, 2, 3, 4
- range(3, 7) : 3, 4, 5, 6
- range(-3, 3) : -3, -2, -1, 0, 1, 2

🧐 https://docs.python.org/3/library/stdtypes.html#range

range

- range(start, stop, step)
  - r[i] = start + step * i where i >= 0 and r[i] < stop
  - r[i] = start + step * i where i >= 0 and r[i] > stop
    - 如果r[0]不满足约束条件，那么构造一个空range对象
- range(0, 11, 2)：0，2，4，6，8，10
- range(10, 3, -3)：10，7，4
- range(10, 3, 3)：空range对象

range是一种可迭代（iterable）对象

- range对象对应一个不可变整数序列，但该序列没有实际存储在内存中，而是按照计算需求一次产生一个整数对象

- range对象支持两种迭代模式：手动和自动

```
>>> I = iter(range(5)) #获得range对象的迭代器
>>> next(I) #通过迭代器的next方法获得下一个元素
0
>>> next(I)
1
>>> for i in range(5): #for循环自动进行迭代
        print(i, end = ' ')


0 1 2 3 4
```

😎 无论一个range对象对应多少个整数，其占用的存储空间是一样的（而且非常小）

内置函数sum

- sum(iterable [, start])：返回start和iterable中所有元素的和，如果没有指定start，start设为0

```
>>> sum(range(5))
10
>>> sum(range(5), 100)
110
>>> sum(range(-3, 3))
-3
>>> sum(range(0, 11, 2))
30
>>> sum(range(10, 3, 3))
0
```

https://docs.python.org/3/library/functions.html#sum

列表的基本概念

- 列表是一组任意类型的对象，对象在列表中具有固定的位置，位置用非负整数0,1,…表示，也称为索引

1	3	5	7	9	11
0	1	2	3	4	5

- 可以通过索引来访问列表的元素

```
>>> L = [1, 3, 5, 7, 9, 11]
>>> L[0]
1
>>> L[5] = 13
>>> L
[1, 3, 5, 7, 9, 13]
```

列表的构造方法

- 使用[ ]创建空列表
- 使用[ ]并在其中放置用逗号分开的一组对象
- 使用函数list([iterable])：使用iterable中的所有元素创建一个集合。如果没有指定iterable，创建空列表
- 使用列表推导[expression for x in iterable]

使用[ ]创建列表

- 使用[]创建列表

```
>>> empty_list = [] #空列表
>>>
>>> primes = [2, 3, 5, 7, 11]
>>>
>>> mixed = [3.14, 'bob', 10]
>>>
>>> matrix = [[1, 0, 0],
              [0, 1, 0],
              [0, 0, 1], #可以有逗号
              ]
```

使用list函数创建列表

- 使用list函数创建列表

```
>>> empty_list = list()
>>>
>>> list(range(5))
[0, 1, 2, 3, 4]
>>>
>>> list(range(1, 10, 2))
[1, 3, 5, 7, 9]
>>>
>>> list('abc')
['a', 'b', 'c']
```

使用列表推导创建列表

- [expression for x in iterable]

```
>>> [x for x in range(5)] # equals to list(range(5))
[0, 1, 2, 3, 4]
>>>
>>> [x ** 2 for x in range(5)]
[0, 1, 4, 9, 16]
>>>
>>> [[] for x in range(5)]
[[], [], [], [], []]
>>>
>>> import math
>>> [math.sqrt(x) for x in range(5)]
[0.0, 1.0, 1.4142135623730951, 1.7320508075688772, 2.0]
```

带条件的列表推导

- [expression for x in iterable if condition]
- 根据条件condition来过滤元素，将符合条件的x对应的expression值放入列表中

```
>>> [x for x in range(10) if x % 2 == 1]
[1, 3, 5, 7, 9]
>>>
>>> [x for x in 'alice' if x in 'aeiou']
['a', 'i', 'e']
>>>
>>> L = [[1, -2, 3], [-4, 5], [-6], [7, -8]]
>>> [x for x in L if sum(x) < 0]
[[-6], [7, -8]]
```

平方和

- 编写一个函数，接收一个正整数n，返回1~n的平方和S
  - $S = 1^2 + 2^2 + \cdots + n^2$
- 如果你不知道平方和公式，怎么办？可以使用列表吗？

```
>>> def sum_of_square(n):
    L = [x ** 2 for x in range(n+1)]
    return sum(L)

>>> sum_of_square(3)
14
>>> sum_of_square(10)
385
```

奇偶排序

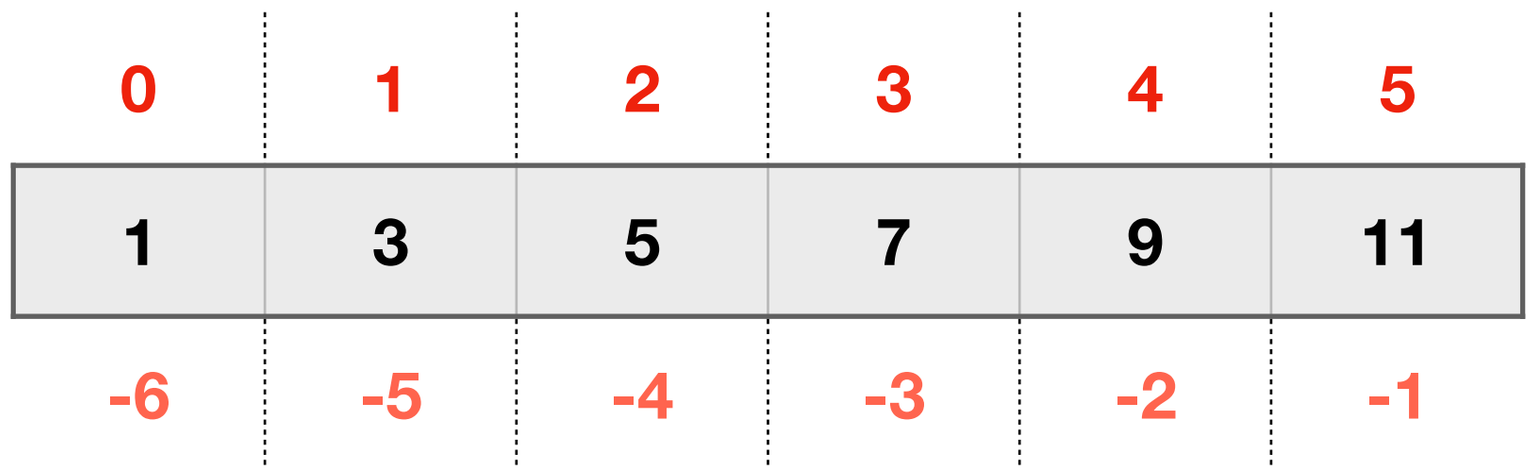
- 编写一个函数，接受一个整数列表，对其排序，要求排序以后所有的奇数在一起，所有的偶数在一起，且奇数在偶数的前面

- 思路：奇数放入一个列表，偶数放入一个列表，然后合并

```
1 def parity_sort(L):
2     odd  = [x for x in L if x % 2 == 1]
3     even = [x for x in L if x % 2 == 0]
4     return odd + even
```

通过索引获取单个元素

- 索引可以为正，有效范围：0 ~ len(L)-1
- 索引可以为负，有效范围：-1 ~ -len(L)
- 如果索引越界（超出有效范围），会抛出异常

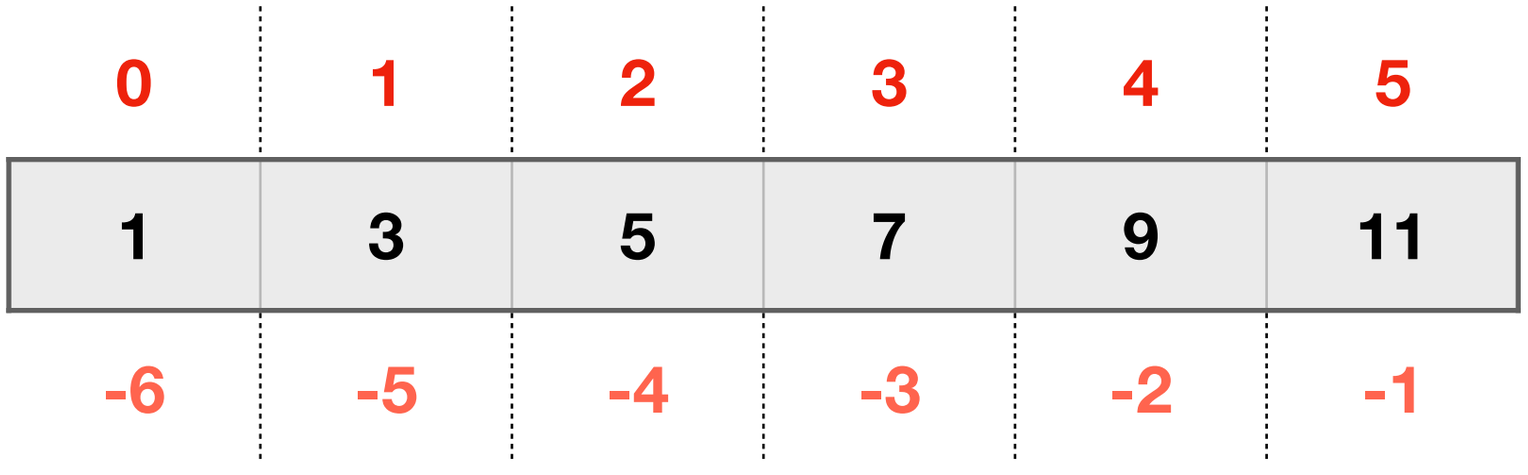



```
>>> L[0]
1
>>> L[-1]
11
>>> L[-6]
1
```

```
>>> L[6]
Traceback (most recent call last):
  File "<pyshell#84>", line 1, in <module>
    L[6]
IndexError: list index out of range
```

通过分片获取子列表

- L[m:n]返回从索引m到索引n的子列表（不包含L[n]）
- 如果m或者n大于len(L)，将其看成len(L)
- 如果m大于n，分片是空列表
- 如果省略m，将其看成0
- 如果省略n，将其看成len(L)

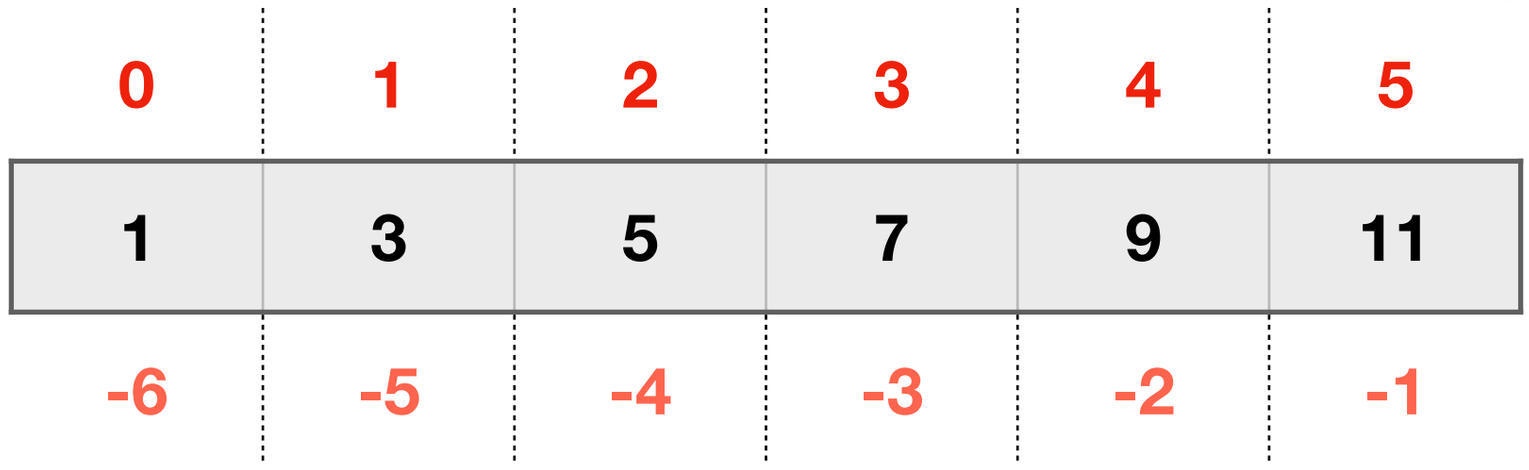


分片的上下边界没有限制，不会产生越界错误

```
>>> L[1:4]
[3, 5, 7]
>>> L[-6:-1]
[1, 3, 5, 7, 9]
>>> L[1:10]
[3, 5, 7, 9, 11]
>>> L[2:0]
[]
>>> L[-4:0] # 同上
[]
>>> L[:3]
[1, 3, 5]
>>> L[3:]
[7, 9, 11]
>>> L[:]
[1, 3, 5, 7, 9, 11]
```

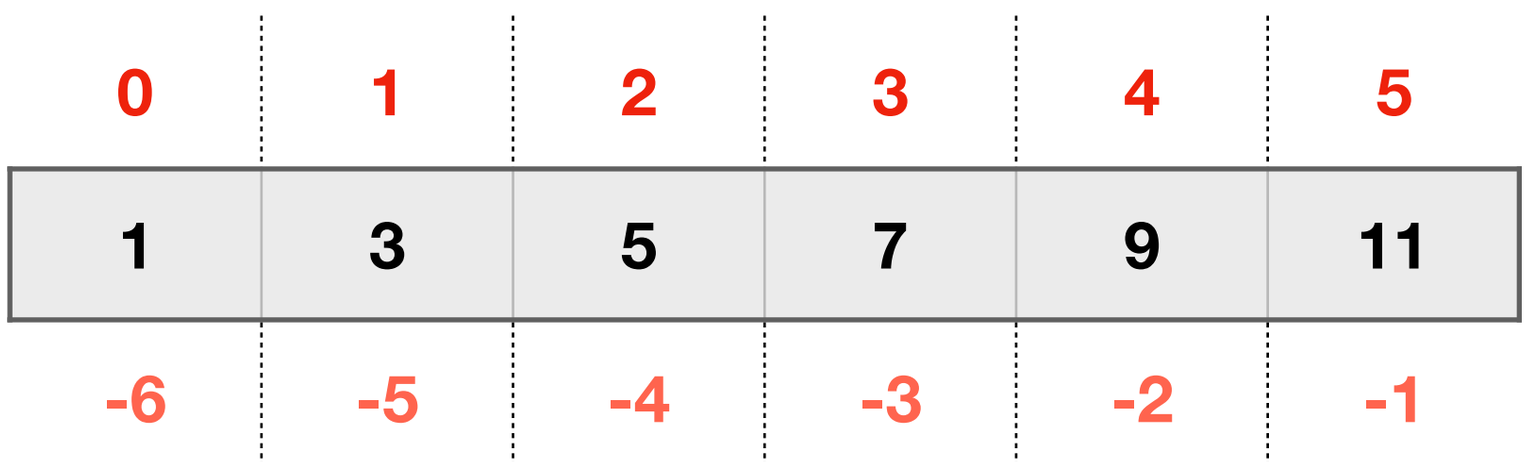
在分片中考虑步长[m:n:s]

- 假设子列表中元素的索引为x，那么
  - x = m + t * s，其中整数t的取值范围是 0 <= t < (n-m)/s
- [0:5:2]：t的值是0,1,2，索引是0, 2, 4，子列表是[1,5,9]
- [1:4:3]：t的值是0，索引是1，子列表是[3]
- [0:5:-2]：没有满足条件的t，所以子列表为空
- [5:0:-2]：t的值是0,1,2，索引是5,3,1，子列表是[11,7,3]



在分片中考虑步长[m:n:s]

- 假设子列表中元素的索引为x，那么
  - x = m + t * s，其中整数t的取值范围是 0 <= t < (n-m)/s
- 当s大于0，如果m或n大于len(L)，将其看成len(L)
- 当s小于0，如果m或n大于len(L)，将其看成len(L)-1
- [10:1:-3]：s小于0，所以m的值看成5，t的值是0和1，索引是5和2，所以子列表是[11, 5]

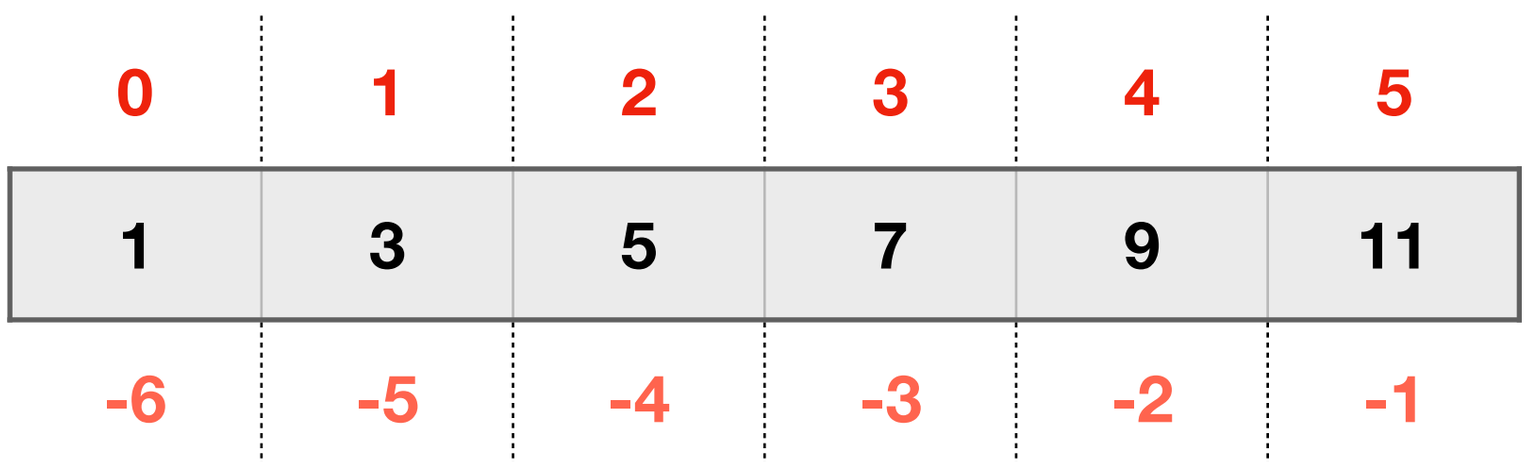



```
>>> L[10:1:-3]
[11, 5]
```

在分片中考虑步长[m:n:s]

- 假设子列表中元素的索引为x，那么
  - x = m + t * s，其中整数t的取值范围是 0 <= t < (n-m)/s
- 如果省略m或n，将其看成边界值，具体情况取决于s的正负
- s不能为0，如果省略s，将其看成1

- [:4:3]：省略m且s为正，将m看成边界值0
- [:4:-3]：省略m且s为负，将m看成边界值5




```
>>> L[1::3]
[3, 9]
>>> L[1::-3]
[3]
>>> L[:4:3]
[1, 7]
>>> L[:4:-3]
[11]
```

分片的常用技巧

- 分片生成的子列表都是副本（即新的列表）
- [::]对原有列表进行浅拷贝
- [::-1]生成原始列表的逆序（等同于列表的reverse方法）
- [:x]和[x:]将列表分成不重叠的两个部分

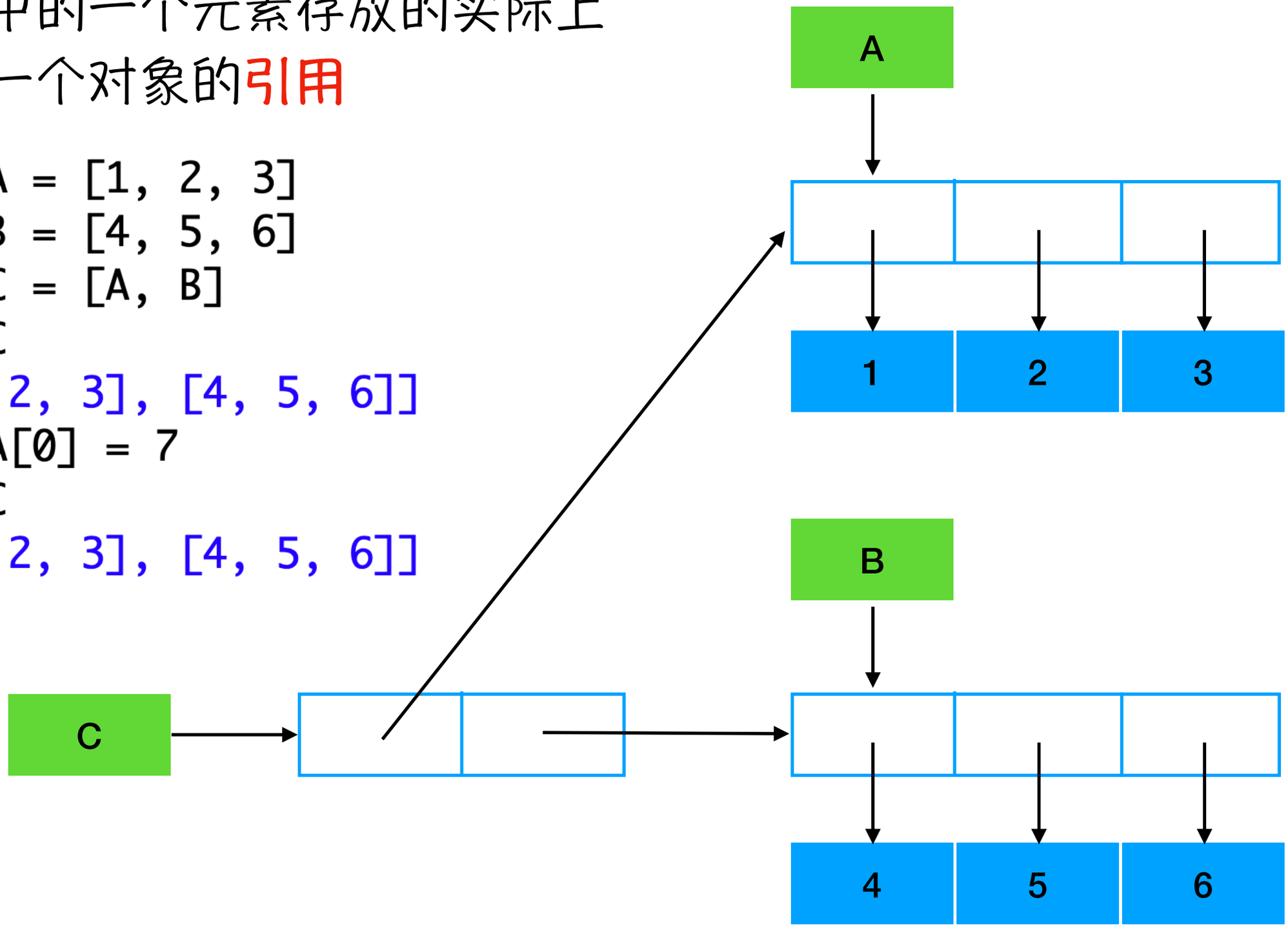
```
>>> A = L[::]
>>> A
[1, 3, 5, 7, 9, 11]
>>> L is A # 测试两者是否是同一个对象
False
>>> L == A # 测试两者的值是否相等
True
```

```
>>> L[::-1]
[11, 9, 7, 5, 3, 1]
>>> L[:1]
[1]
>>> L[1:]
[3, 5, 7, 9, 11]
```

列表存储对象的引用

- 列表中的一个元素存放的实际上是对一个对象的引用

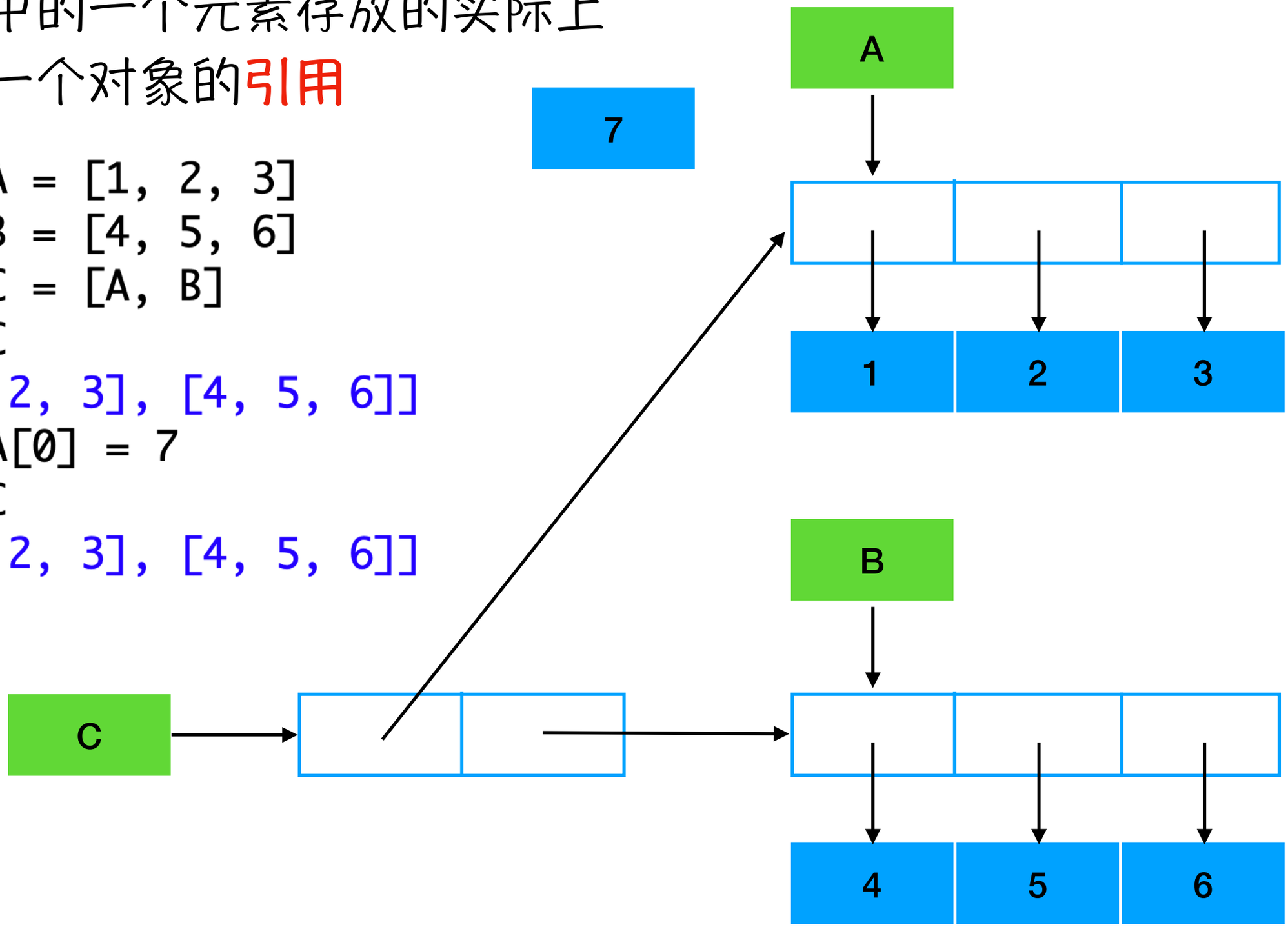
```
>>> A = [1, 2, 3]
>>> B = [4, 5, 6]
>>> C = [A, B]
>>> C
[[1, 2, 3], [4, 5, 6]]
>>> A[0] = 7
>>> C
[[7, 2, 3], [4, 5, 6]]
```



列表存储对象的引用

- 列表中的一个元素存放的实际上是对一个对象的引用

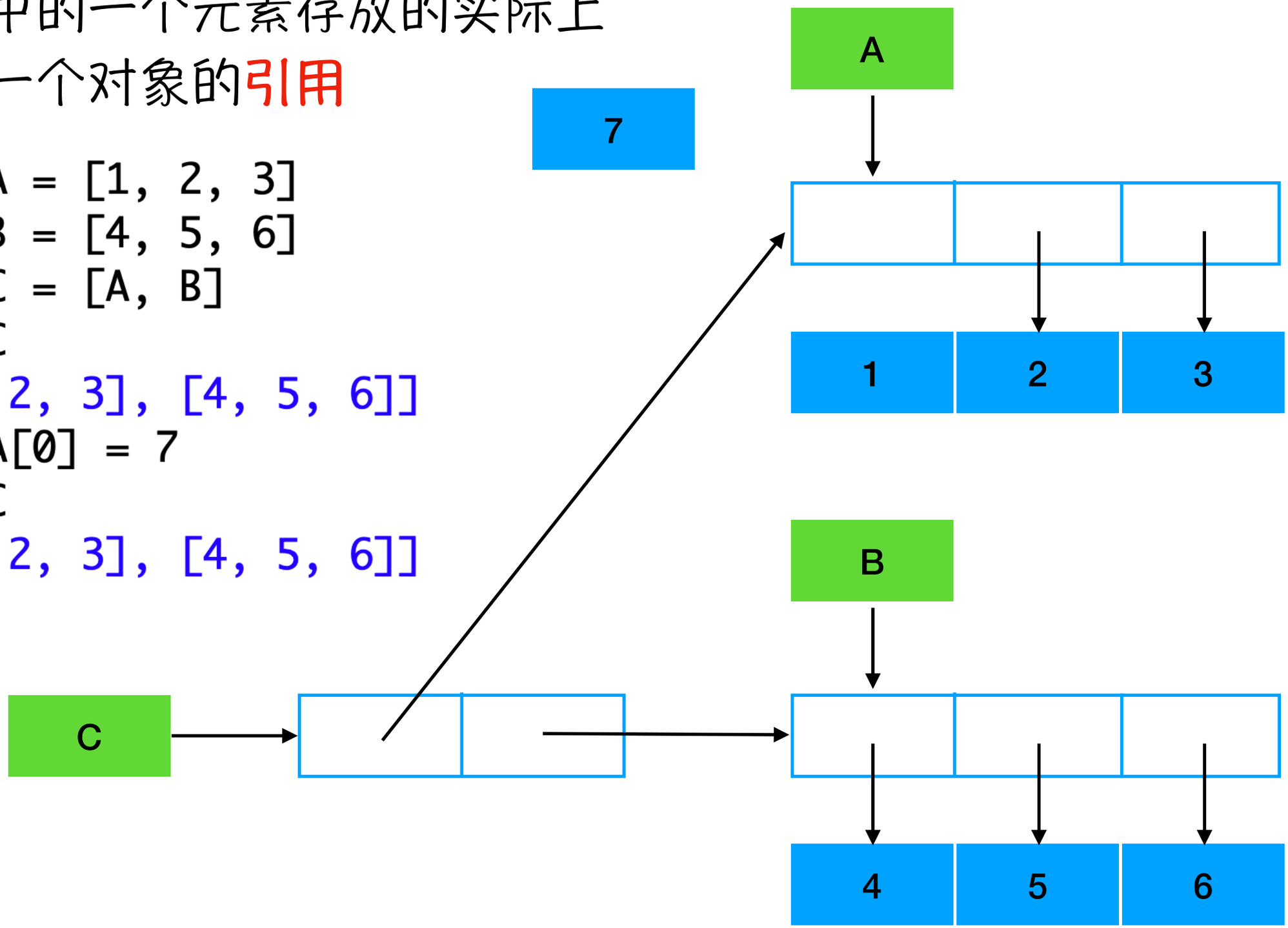
```
>>> A = [1, 2, 3]
>>> B = [4, 5, 6]
>>> C = [A, B]
>>> C
[[1, 2, 3], [4, 5, 6]]
>>> A[0] = 7
>>> C
[[7, 2, 3], [4, 5, 6]]
```



列表存储对象的引用

- 列表中的一个元素存放的实际上是对一个对象的引用

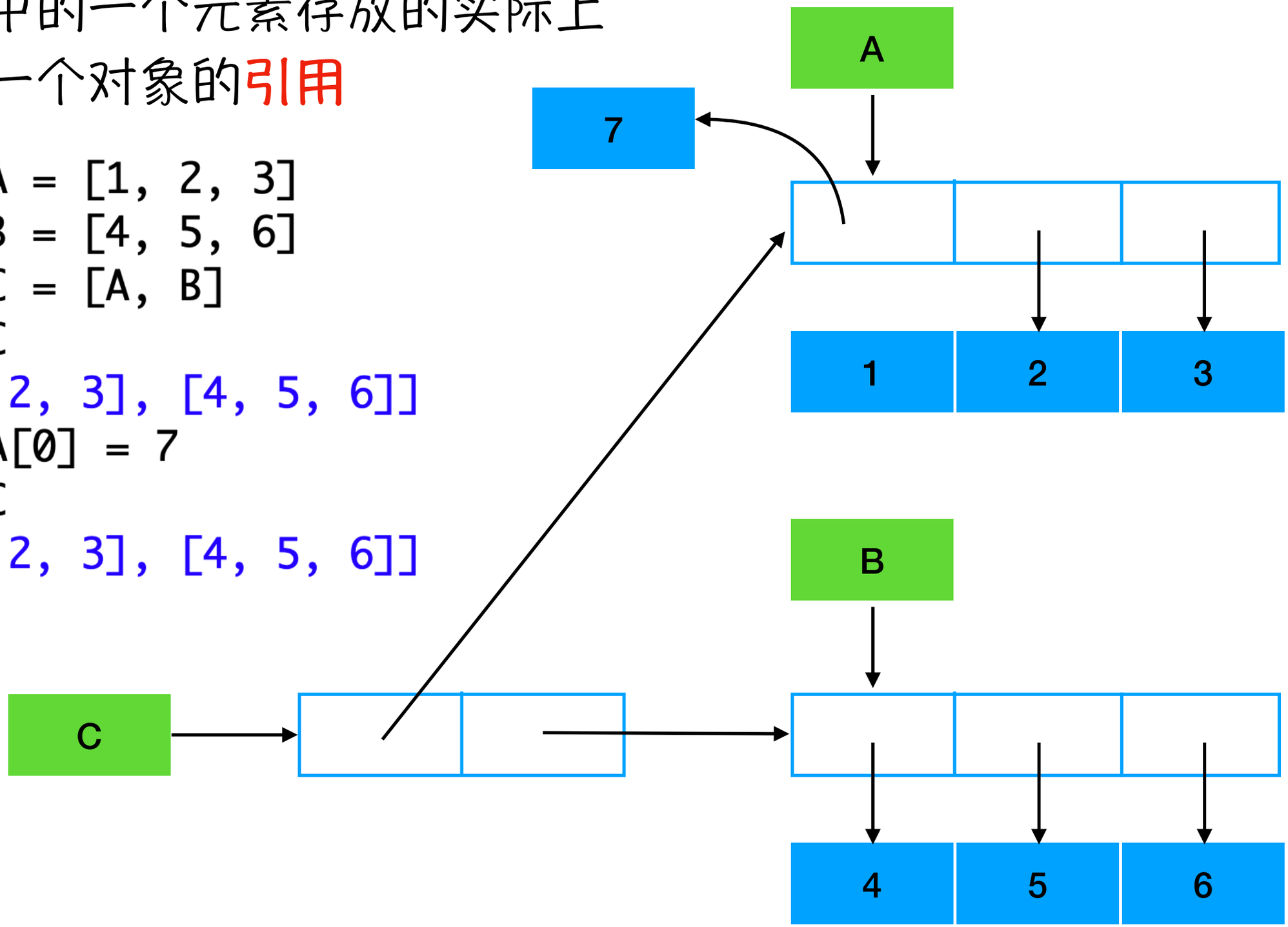
```
>>> A = [1, 2, 3]
>>> B = [4, 5, 6]
>>> C = [A, B]
>>> C
[[1, 2, 3], [4, 5, 6]]
>>> A[0] = 7
>>> C
[[7, 2, 3], [4, 5, 6]]
```



列表存储对象的引用

- 列表中的一个元素存放的实际上是对一个对象的引用

```
>>> A = [1, 2, 3]
>>> B = [4, 5, 6]
>>> C = [A, B]
>>> C
[[1, 2, 3], [4, 5, 6]]
>>> A[0] = 7
>>> C
[[7, 2, 3], [4, 5, 6]]
```



列表的连接和重复操作

- s+t：将s的元素和t的元素连接起来，形成一个新列表
- s*n 或者 n*s (n是一个整数)：将s的元素重复n次，形成一个新列表

```
>>> [1, 2] + [3, 4]
[1, 2, 3, 4]
>>>
>>> [1, 2] * 3
[1, 2, 1, 2, 1, 2]
>>> 3 * [1, 2]
[1, 2, 1, 2, 1, 2]
```

神秘的数学常数

- 圆周率 $\pi \approx 3.141592653589793$
- 自然底数 $e \approx 2.718281828459045$
- 欧拉常数 $\gamma \approx 0.577215664901532$

```
>>> import math
>>> math.pi
3.141592653589793
>>> math.e
2.718281828459045
```

😎 更多神秘的数学常数参http://www.matrix67.com/blog/archives/3682

神秘的数学常数

- 将这些神秘的数学常数的各位数字存放在列表中

```
>>> pi = [3, 1, 4, 1, 5, 9]
>>> e = [2, 7, 1, 8]
>>> gamma = [0, 5, 7, 7, 2, 1, 5, 6]
```

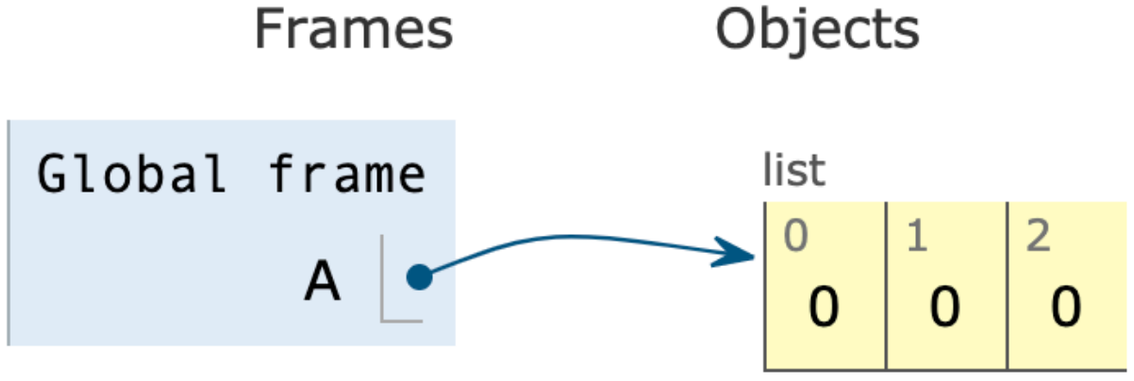
- 通过列表的分片以及连接操作来构造下面的列表
  - [2, 0, 1, 9] - 2019
  - [1, 1, 9] - 119
  - [1, 2, 0] - 120
  - [1, 2, 3, 4, 5, 6, 7, 8, 9, 0] - 1234567890

神秘的数学常数

```
>>> pi = [3, 1, 4, 1, 5, 9]
>>> e = [2, 7, 1, 8]
>>> gamma = [0, 5, 7, 7, 2, 1, 5, 6]
>>>
>>> e[0:1] + gamma[0:1] + gamma[5:6] + pi[5:6]
[2, 0, 1, 9]
>>> pi[1::2]
[1, 1, 9]
>>> gamma[5:3:-1] + gamma[0:1]
[1, 2, 0]
>>> gamma[5:3:-1] + pi[::2] + gamma[7:0:-4] + e[3:4] +
pi[5:6] + gamma[:1]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
```

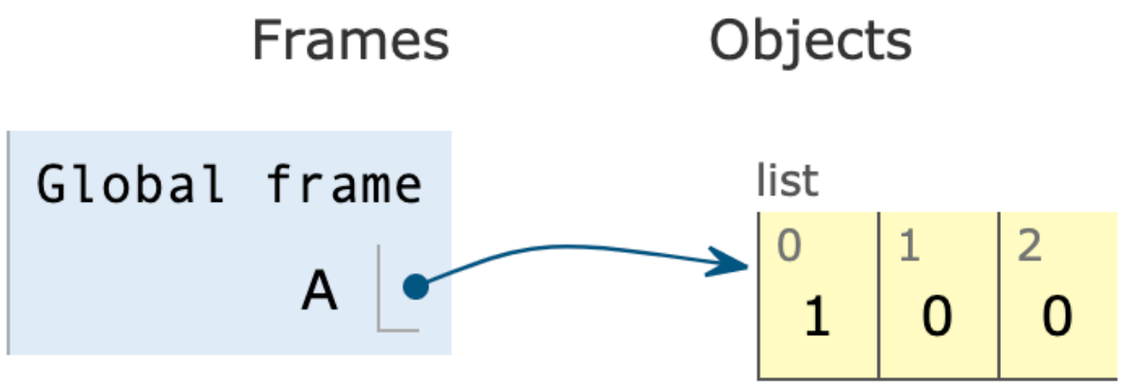
对象引用对重复操作的影响

- 列表中的一个元素存放的实际上是对一个对象的引用
- 数值属于不可变类型
- 执行赋值语句A[0]=1



- 创建对象1，将其引用存放在A[0]

```
>>> A = [0] * 3
>>> A
[0, 0, 0]
>>> A[0] = 1
>>> A
[1, 0, 0]
```



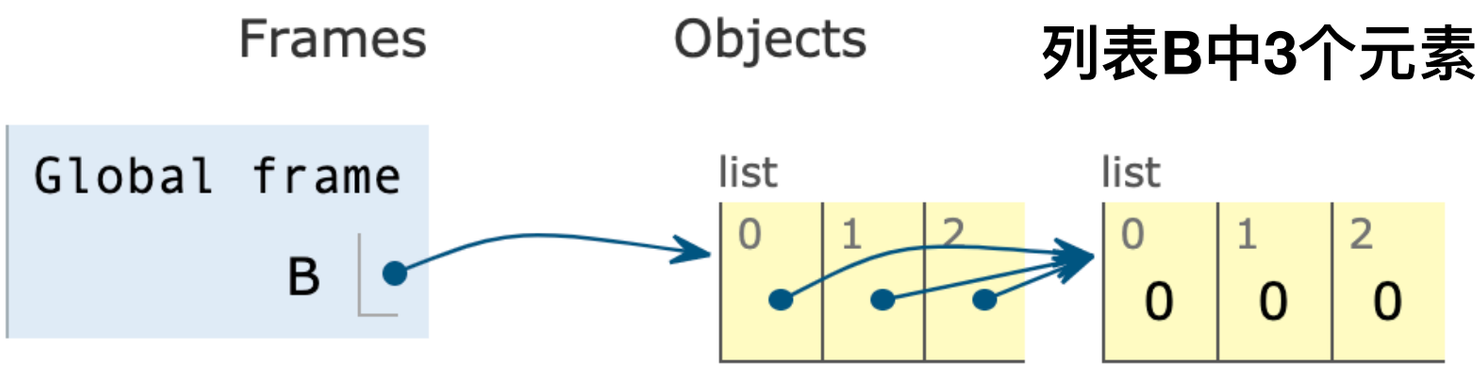
上图通过代码可视化工具生成 - http://www.pythontutor.com/

这里简化了不可变类型对象的表示，没有画出对应的引用

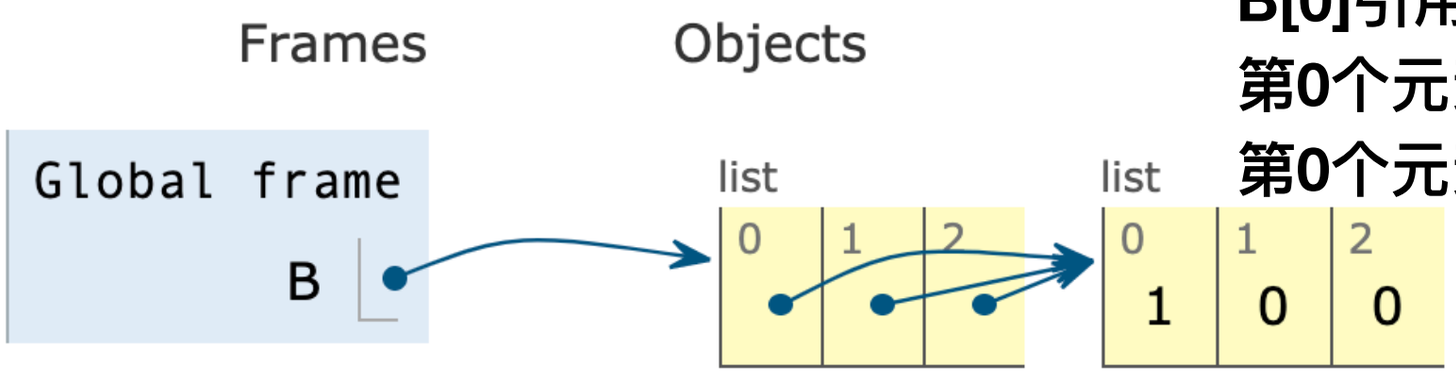
对象引用对重复操作的影响

- 如果列表中存放的是可变类型，重复操作只是复制了引用

```
>>> B = [[0, 0, 0]] * 3
>>> B
[[0, 0, 0], [0, 0, 0], [0, 0, 0]]
>>> B[0][0] = 1
>>> B
[[1, 0, 0], [1, 0, 0], [1, 0, 0]]
```



列表B中3个元素实际是对同一个列表的引用



B[0]引用一个列表，赋值操作将该列表的第0个元素修改成1，更准确的，B[0]中第0个元素引用的对象从整数0变成整数1

列表常用的方法

- x in L：如果L中有一个元素和x相等，返回True，否则False
- len(L)：返回列表的长度
- min(L)/max(L)：返回列表中的最小值/最大值
- L.count(x)：返回x在L中的出现次数
- L.index(x[, i[, j]])：x在L第一次出现的位置

```
>>> L = [x + x % 4 for x in range(1, 11)]
>>> L
[2, 4, 6, 4, 6, 8, 10, 8, 10, 12]
>>> 10 in L
True
>>> len(L), min(L), max(L)
(10, 2, 12)
>>> L.count(4)
2
```

这些方法没有对列表进行修改，
所以它们也适用于其他的不可变序列，
包括字符串、元组、range

index方法

- L.index(x[, i[, j]])：x在L第一次出现的位置
  - 只考虑L[i:j]中的元素
  - 如果x没有出现，抛出异常

```
>>> L = [1, 2, 3, 1, 1, 2, 3]
>>> L.index(2)
1
>>> L.index(2, 2, 5)
Traceback (most recent call last):
  File "<pyshell#192>", line 1, in <module>
    L.index(2, 2, 5)
ValueError: 2 is not in list
>>> L.index(2, 4)
5
```

范围查询

- 编写一个函数，接受一个列表L和两个数值lb和ub，返回列表中满足条件lb <= x <= ub 的元素x的个数

```
def count_range(L, lb, ub):
    lst = [x for x in L if lb <= x <= ub]
    return len(lst)
```

```
>>> L
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> count_range(L, 3, 8)
6
>>> count_range(L, -5, 77)
10
>>> count_range(L, 12, 18)
0
>>> count_range([], 1, 6)
0
```

改进的平均值

- 在分析数据时倾向于去除数据中的极端最大或者极端最小值（也称离群点），因为这些值通常是系统受外部干扰造成的，会影响数据分析的精度

- 编写一个函数，接受一个长度大于2的列表，去除其中的最大值后计算剩余元素的均值，注意最大值可能不止一个

```
>>> def advanced_avg(L):
        t = max(L)
        freq = L.count(t)
        total = sum(L) - t * freq
        cnt = len(L) - freq
        return total / cnt
```

```
>>> L = [1, 2, 3, 4, 100]
>>> advanced_avg(L)
2.5
```

```
>>> L = [1, 2, 3, 4, 100, 100]
>>> advanced_avg(L)
2.5
```

修改列表常用的方法

- 添加元素
  - append、extend、insert
- 删除元素
  - clear、pop、remove
  - del
- 分片用于赋值实现添加、删除、修改元素
- 反转列表（将列表中所有元素逆序）
  - reverse、reversed、分片

🧐 **上述方法也适用于所有的序列类型，包括不可变序列字符串、元组、range**

向列表添加元素

- append(x)：将x添加至列表末端
- extend(iterable)：将iterable中的多个元素添加至列表末端
- insert(i, x)：将x添加至索引为i的位置

```
>>> L = [1, 2]
>>> L.append(3)
>>> L
[1, 2, 3]
>>> L.extend([4, 5])
>>> L
[1, 2, 3, 4, 5]
>>> L.extend(range(6,8))
>>> L
[1, 2, 3, 4, 5, 6, 7]
```

```
>>> L = [1, 2]
>>> L.extend([3, 4])
>>> L
[1, 2, 3, 4]
>>> L.append([3, 4])
>>> L
[1, 2, 3, 4, [3, 4]]
```

向列表添加元素

- append(x)：将x添加至列表末端
- extend(iterable)：将iterable中的多个元素添加至列表末端
- insert(i, x)：将x添加至索引为i的位置

```
>>> L = [2, 3, 5, 7, 11]
>>> L.insert(0, 1)
>>> L
[1, 2, 3, 5, 7, 11]
>>> L.insert(6, 13)
>>> L
[1, 2, 3, 5, 7, 11, 13]
>>> L.insert(4, 6)
>>> L
[1, 2, 3, 5, 6, 7, 11, 13]
```

😎 不检查i是否越界，同时也支持负数索引

删除列表中的元素

- clear方法删除列表中的所有元素（成为空列表）
- pop([i])方法删除索引为i的元素，如果没有指定i，i取-1
- remove(x)方法删除列表中第一次出现的x

```
>>> L = [1, 2, 1, 3, 1, 4]
>>> L.pop(4)
1
>>> L
[1, 2, 1, 3, 4]
>>> L.pop()
4
>>> L
[1, 2, 1, 3]
```

```
>>> L.remove(1)
>>> L
[2, 1, 3]
>>> L.remove(1)
>>> L
[2, 3]
>>> L.clear()
>>> L
[]
```

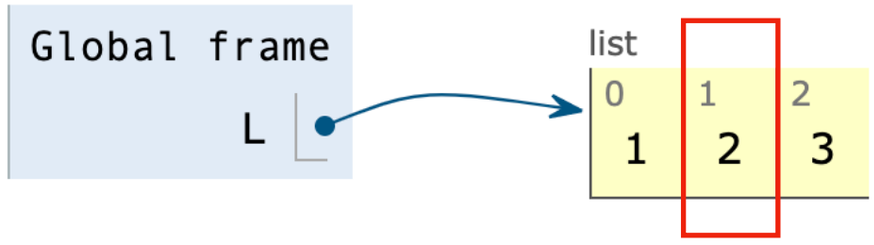
删除列表中的元素

- 语句del L[i]删除索引为i的元素
  - 索引i可以是负数，但不能超过有效范围，否则引发异常
- 语句del L[i:j]从L中删除分片L[i:j]包含的元素
- 语句del L[i:j:k]从L中删除分片[i:j:k]包含的元素

```
>>> L = [1, 2, 3, 4, 5, 6, 7, 8]
>>> del L[::3]
>>> L
[2, 3, 5, 6, 8]
>>> del L[1:4]
>>> L
[2, 8]
>>> del L[1]
>>> L
[2]
```

分片用于赋值操作

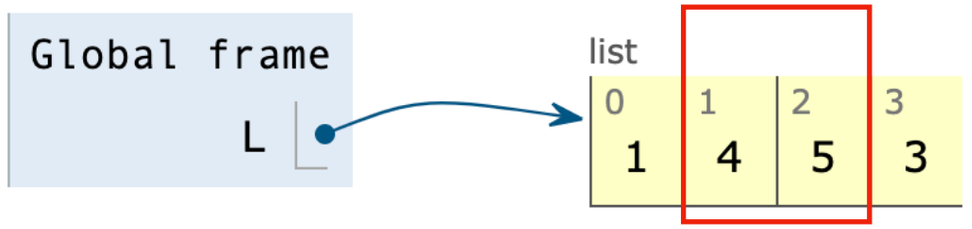
- 分片位于赋值运算符的左侧
- 注意分片的长度以及右侧值（列表）的长度

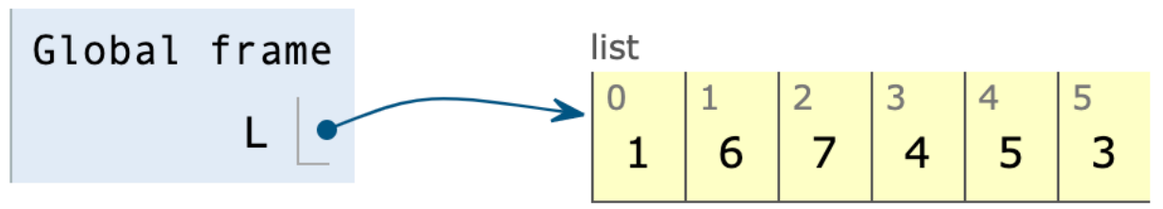


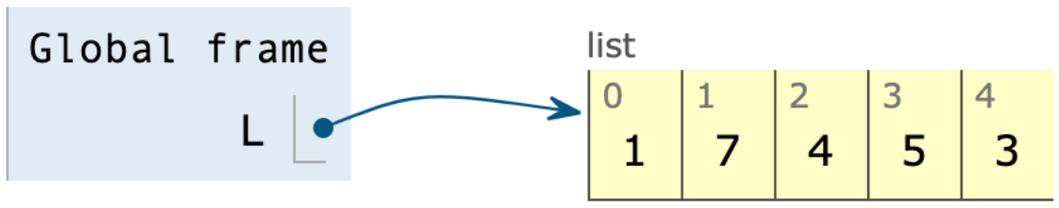
L[1:2]

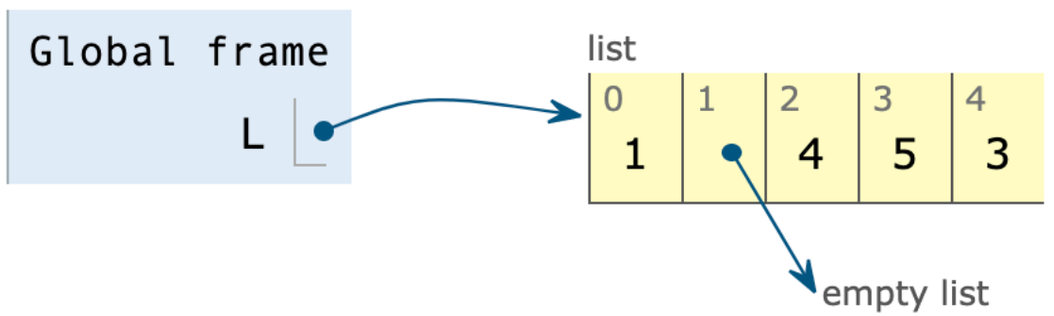
```
>>> L = [1, 2, 3]
>>> L[1:2] = [4, 5]
>>> L
[1, 4, 5, 3]
>>> L[1:1] = [6, 7]
>>> L
[1, 6, 7, 4, 5, 3]
>>> L[1:2] = []
>>> L
[1, 7, 4, 5, 3]
>>>
>>> L[1] = []
>>> L
[1, [], 4, 5, 3]
```

这里不是分片！









反转列表中的所有元素

- reverse方法：原地反转，修改L
- reversed(L)函数：将L中的元素反转后作为一个可迭代对象返回（可将其放入list()函数中生成一个反转后的列表）
- 分片：返回一个新的列表

```
>>> L = [1, 2, 3, 4, 5]
>>> L.reverse()
>>> L
[5, 4, 3, 2, 1]
>>> list(reversed(L))
[1, 2, 3, 4, 5]
>>> L[::-1]
[1, 2, 3, 4, 5]
>>>
>>> L
[5, 4, 3, 2, 1]
```

基于均值的划分

- 编写一个函数，接受一个长度大于2的列表，计算去除其中最大的2个元素后的剩余元素的均值，以该均值为标准，构造并返回一个新的列表，该列表的第0个元素是所有低于均值的元素构成的列表，第1个元素是所有等于均值的元素构成的列表，第2个元素是所有大于均值的元素构成的列表
- 比如L = [1, 2, 3, 4, 5, 99, 100]，最大的两个元素是99和100，去除之后剩余元素的均值是3，所以最终返回的列表是[[1, 2], [3], [4, 5]]

基于均值的划分

- 参考代码

```
def partition_avg(L):
    L.remove(max(L))
    L.remove(max(L))
    avg = sum(L) / len(L)
    left = [x for x in L if x < avg]
    mid = [x for x in L if x == avg]
    right = [x for x in L if x > avg]
    return [left, mid, right]
```

```
>>> L = [1, 2, 3, 4, 100, 100]
>>> partition_avg(L)
[[1, 2], [], [3, 4]]
>>>
>>> L = [10, 100, 1000]
>>> partition_avg(L)
[[], [10], []]
```