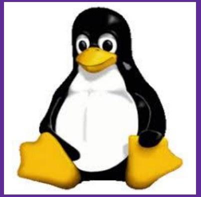
Introduction to Computation

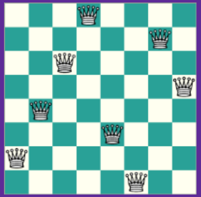
Autumn, 2023

Prof. Fan Cheng

Shanghai Jiao Tong University

chengfan85@gmail.com
https://github.com/ichengfan/itc





6

Outline

- List

Data structure (数据结构)

Suppose we have lots of data, how could we find and modify the right information efficiently?

Student id	Name	GPA	Math	Physics	Python	….
001	James	3.5	90			
002	Ivan	3	80			
…	…					
…	…					
…	…					

- We need to store the data in a **structural manner**
- In computer science, a **data structure** is a data organization, management and storage format that enables efficient **access and modification**
- More precisely, a data structure is **a collection of data values**, **the relationships** among them, and **the functions or operations** that can be applied to the data

data structure = data representation + data operation (functions)

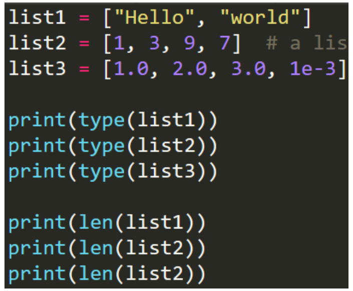
List 列表

如何处理10^{20}个整数？定义10^{20}个变量？

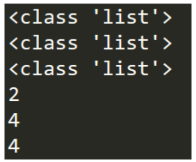
List: creation

Enclose the set of elements in square brackets [], separated by “,”

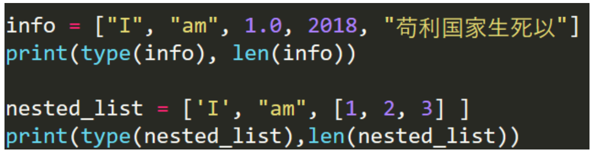
- list1 = [“Hello”, “world”]   # a list with two string
  list2 = [1, 3, 9, 7] # a list with four integers
  list3 = [1.0, 2.0, 3.0, 1e-3]   # a list with four floats
- The elements of a list don’t have to be the same type
- A list within another list is called to be nested (嵌套)
- Recall: assignment statement (=) can create a variable
  type() could be used to determine the type of list1, list2, list3
  len() could be used to determine the length of a list, like list1, list2, list3



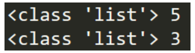
```
list1 = ["Hello", "world"]
list2 = [1, 3, 9, 7]  # a lis
list3 = [1.0, 2.0, 3.0, 1e-3]

print(type(list1))
print(type(list2))
print(type(list3))

print(len(list1))
print(len(list2))
print(len(list2))
```



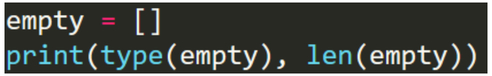
```
<class 'list'>
<class 'list'>
<class 'list'>
2
4
4
```



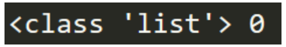
```
info = ["I", "am", 1.0, 2018, "苟利国家生死以"]
print(type(info), len(info))

nested_list = ['I', "am", [1, 2, 3] ]
print(type(nested_list),len(nested_list))
```



```
<class 'list'> 5
<class 'list'> 3
```



```
empty = []
print(type(empty), len(empty))
```

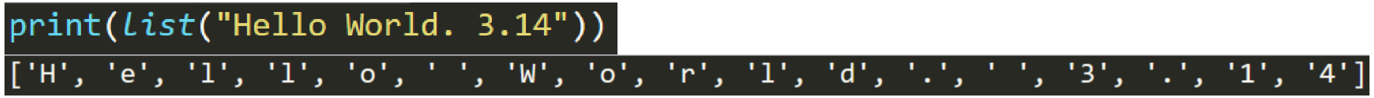


```
<class 'list'> 0
```

严格按照语法：[,,,,,,,,], 不是{,,,,,,}, 也不是[;;;;;]
[],(),{} are different

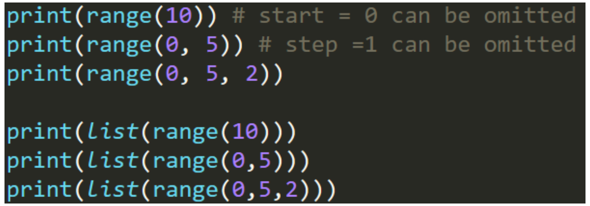
List: list()

- The type of a list is “list”. Python provides a built-in type conversion function called list() that tries to turn whatever you give it into a list.
    - list() is called the constructor (构造函数，每个类型都有自己的构造函数)
- Pass string to list() will split the string to list form

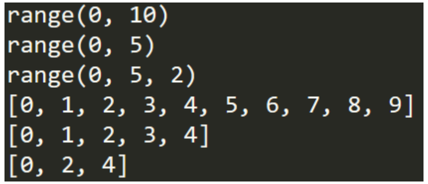


```
print(list("Hello World. 3.14"))
['H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd', '.', ' ', '3', '.', '1', '4']
```

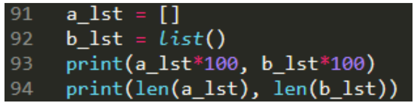
- List with range(). range(start, stop, step) returns a sequence from start to stop with step
    - When start = 0, start can be omitted
    - When step = 1, step can be omitted
    - Together with list(), it generates a list from start to stop with step



```
print(range(10)) # start = 0 can be omitted
print(range(0, 5)) # step =1 can be omitted
print(range(0, 5, 2))

print(list(range(10)))
print(list(range(0,5)))
print(list(range(0,5,2)))
```



```
range(0, 10)
range(0, 5)
range(0, 5, 2)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 1, 2, 3, 4]
[0, 2, 4]
```

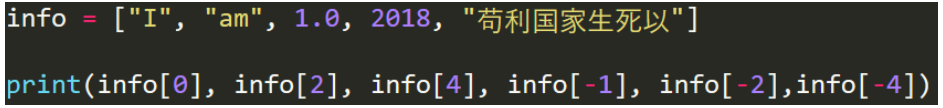


```
a_lst = []
b_lst = list()
print(a_lst*100, b_lst*100)
print(len(a_lst), len(b_lst))
```

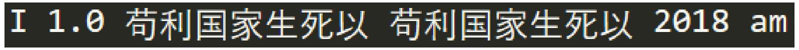
Empty list

List: access

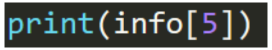
- A list is an ordered set of values, where each value is identified by an index
- Elements in the list (a) are ordered starting from 0, 1, …, len(a)-1
- To access the i-th element of a list: a[i]
  - If i exceeds the max length of a, there will be an error
- If an index has a negative value, it counts backward from the end of the list
  - a[-1] is the last element of list a (可以看作 mod len(a))



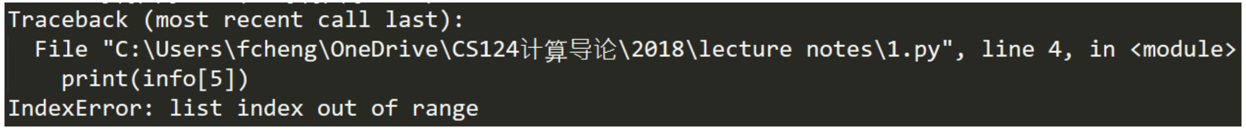
```
info = ["I", "am", 1.0, 2018, "苟利国家生死以"]

print(info[0], info[2], info[4], info[-1], info[-2],info[-4])
```



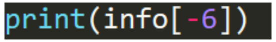
```
I 1.0 苟利国家生死以 苟利国家生死以 2018 am
```



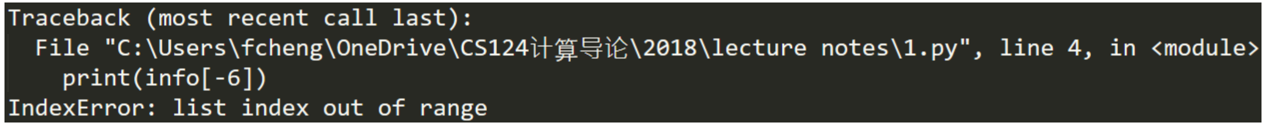
```
print(info[5])
```



```
Traceback (most recent call last):
  File "C:\Users\fcheng\OneDrive\CS124计算导论\2018\lecture notes\1.py", line 4, in <module>
    print(info[5])
IndexError: list index out of range
```



```
print(info[-6])
```



```
Traceback (most recent call last):
  File "C:\Users\fcheng\OneDrive\CS124计算导论\2018\lecture notes\1.py", line 4, in <module>
    print(info[-6])
IndexError: list index out of range
```

世界上最常见的错误: out of range

for + list

- for x in collection_a: # 合集
            do_sth()

  #The type of collection_a could be list, tuple, dict, string, set, etc.

```python
lst1 = [1, 2, 3, -1, "Hello", 3.14]
for x in lst1:
    print(x, end=" ")
print()

lst1 = list(range(7))
for x in lst1:
    print(x, end=" ")
print()
```

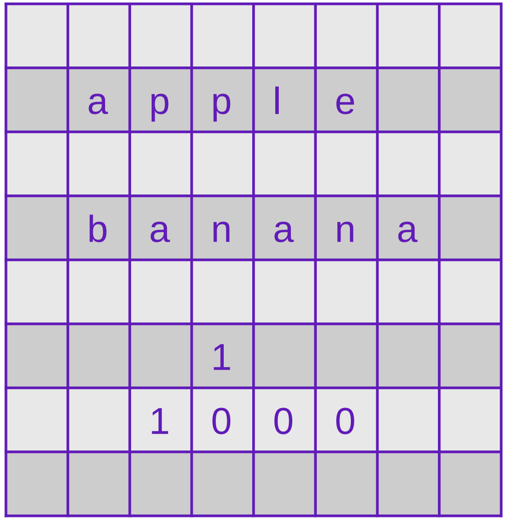
```
1 2 3 -1 Hello 3.14
0 1 2 3 4 5 6
```

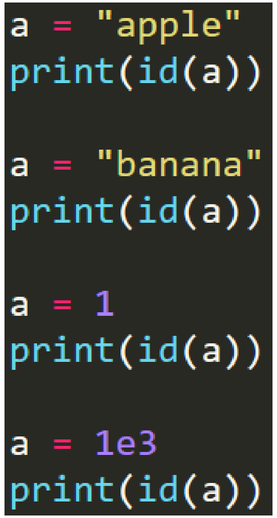
```python
i = 0
while i < len(lst1):
    print(lst1[i], end=" ")
    i += 1
print()
```

Data model

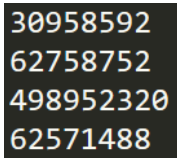
- Recall: Every variable points to a memory location in python. The location is referred to as id of the variable.
  - Every variable x in python has an id. Function **id(x)** returns the id of x.
  - id(x): 变量x所指向的数据的地址
  - When variable is reassigned, the location (id) will be different
  - https://docs.python.org/3/library/functions.html#id

赋值语句x=y中，x和y会指向同一块内存区域，具有相同的id。
如果x的类型可以修改，那么修改y相当于修改x。

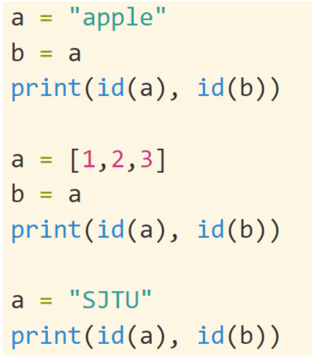




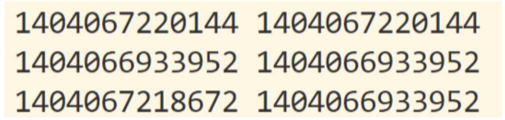
```
a = "apple"
print(id(a))

a = "banana"
print(id(a))

a = 1
print(id(a))

a = 1e3
print(id(a))
```



```
30958592
62758752
498952320
62571488
```



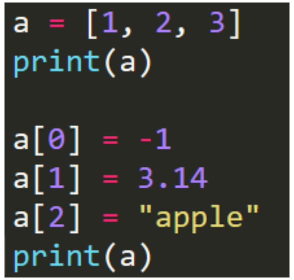
```
a = "apple"
b = a
print(id(a), id(b))

a = [1,2,3]
b = a
print(id(a), id(b))

a = "SJTU"
print(id(a), id(b))
```



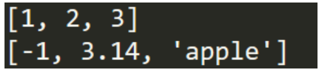
```
1404067220144 1404067220144
1404066933952 1404066933952
1404067218672 1404066933952
```

List: modification

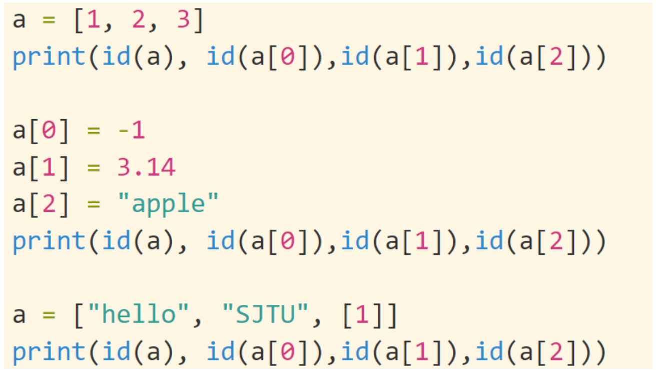
- Assignment statement could be used to modify the elements of a list
    - list[i] = new_element
- List is mutable (可修改的) . However, in some situation, the id remains unchanged. That is, the content of the list is changed, not its location



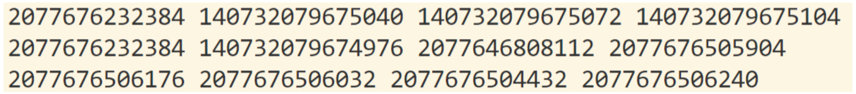
```python
a = [1, 2, 3]
print(a)

a[0] = -1
a[1] = 3.14
a[2] = "apple"
print(a)
```



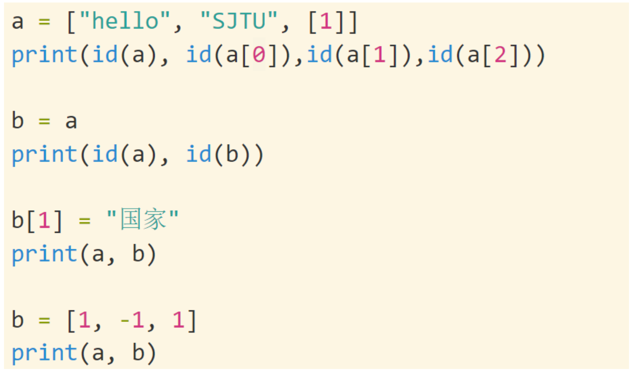
```
[1, 2, 3]
[-1, 3.14, 'apple']
```



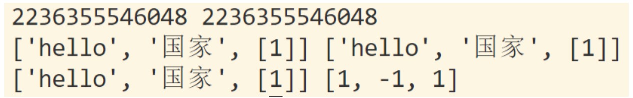
```python
a = [1, 2, 3]
print(id(a), id(a[0]),id(a[1]),id(a[2]))

a[0] = -1
a[1] = 3.14
a[2] = "apple"
print(id(a), id(a[0]),id(a[1]),id(a[2]))

a = ["hello", "SJTU", [1]]
print(id(a), id(a[0]),id(a[1]),id(a[2]))
```



```
2077676232384 140732079675040 140732079675072 140732079675104
2077676232384 140732079674976 2077646808112 2077676505904
2077676506176 2077676506032 2077676504432 2077676506240
```



```python
a = ["hello", "SJTU", [1]]
print(id(a), id(a[0]),id(a[1]),id(a[2]))

b = a
print(id(a), id(b))

b[1] = "国家"
print(a, b)

b = [1, -1, 1]
print(a, b)
```

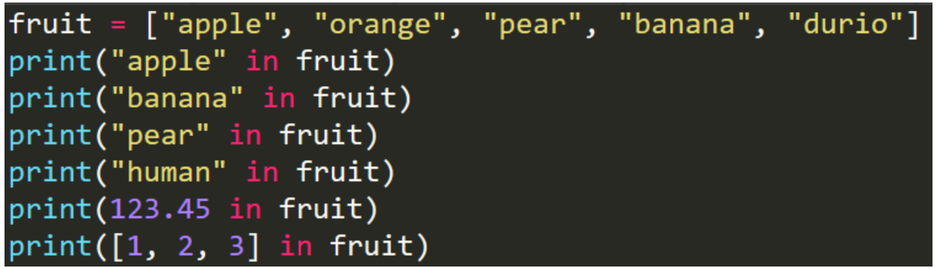


```
2236355546048 2236355546048
['hello', '国家', [1]] ['hello', '国家', [1]]
['hello', '国家', [1]] [1, -1, 1]
```

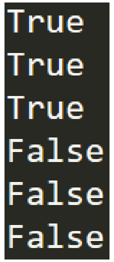
List可修改，重新赋值和修改差别很大
数据是否可以被修改是一个关键因素

List: membership

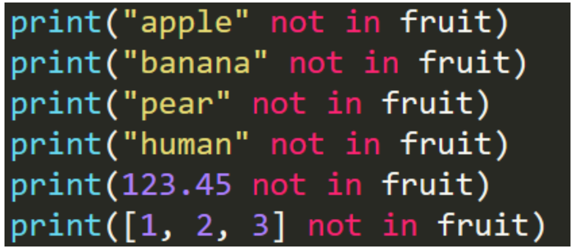
- One can use in and not in to test whether an element belonging to a list
- in: x in S, return True if $x \in S$, otherwise False
- not in: x not in S, return True if $x \notin S$, otherwise False



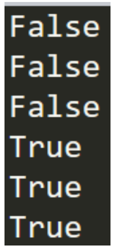
```python
fruit = ["apple", "orange", "pear", "banana", "durio"]
print("apple" in fruit)
print("banana" in fruit)
print("pear" in fruit)
print("human" in fruit)
print(123.45 in fruit)
print([1, 2, 3] in fruit)
```



```
True
True
True
False
False
False
```



```python
print("apple" not in fruit)
print("banana" not in fruit)
print("pear" not in fruit)
print("human" not in fruit)
print(123.45 not in fruit)
print([1, 2, 3] not in fruit)
```

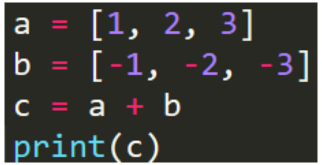


```
False
False
False
True
True
True
```

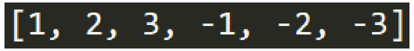
in 和 not in是关键字

List: +, *

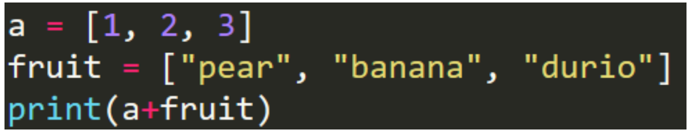
- +: a+b will concatenate two lists a and b



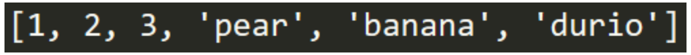
```python
a = [1, 2, 3]
b = [-1, -2, -3]
c = a + b
print(c)
```



```
[1, 2, 3, -1, -2, -3]
```

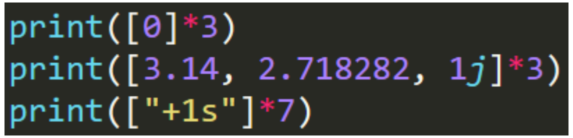


```python
a = [1, 2, 3]
fruit = ["pear", "banana", "durio"]
print(a+fruit)
```

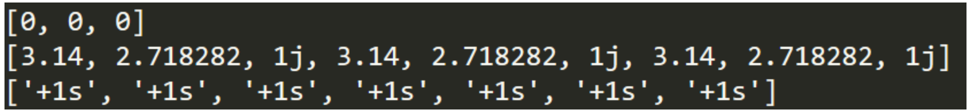


```
[1, 2, 3, 'pear', 'banana', 'durio']
```

- *: a*n will repeat a list a given number of times: n



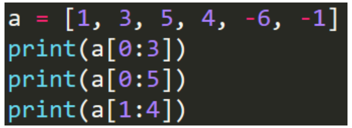
```python
print([0]*3)
print([3.14, 2.718282, 1j]*3)
print(["+1s"]*7)
```



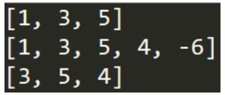
```
[0, 0, 0]
[3.14, 2.718282, 1j, 3.14, 2.718282, 1j, 3.14, 2.718282, 1j]
['+1s', '+1s', '+1s', '+1s', '+1s', '+1s', '+1s']
```

List: slice (切片)

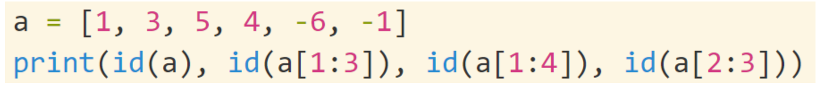
- Given a list list_a, you could get a slice of list_a via: list_a[i:j], it is the sub-list of list_a[i], list_[i+1], …, list_a[j-1]. (list_a[j] is not included!)
- The type of list_a[i:j] is still list
- In default, the slice is started from 0 and ended till the end of the list
  - a[0:3] is identical to a[:3] (0 can be omitted)
  - a[2:6] is identical to a[2:] (6 can be omitted)
  - a[0:6] is identical to a[:] (Both 0 and 6 can be omitted)
- With the slice operator we can update several elements at once
  - We can also remove elements from a list by assigning the empty list to the slice



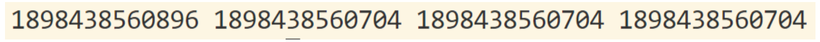
```python
a = [1, 3, 5, 4, -6, -1]
print(a[0:3])
print(a[0:5])
print(a[1:4])
```



```
[1, 3, 5]
[1, 3, 5, 4, -6]
[3, 5, 4]
```

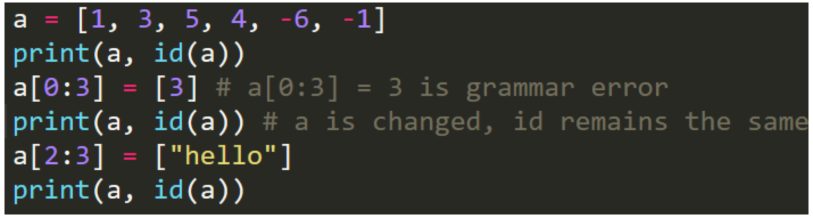


```python
a = [1, 3, 5, 4, -6, -1]
print(id(a), id(a[1:3]), id(a[1:4]), id(a[2:3]))
```

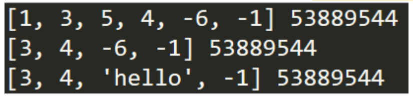


```
1898438560896 1898438560704 1898438560704 1898438560704
```

id(a[0:4]) ≠id(a)
a and a[:] are not in the same address, but they have identical elements



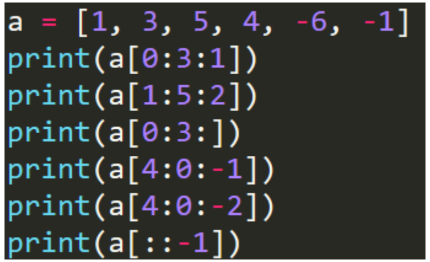
```python
a = [1, 3, 5, 4, -6, -1]
print(a, id(a))
a[0:3] = [3] # a[0:3] = 3 is grammar error
print(a, id(a)) # a is changed, id remains the same
a[2:3] = ["hello"]
print(a, id(a))
```



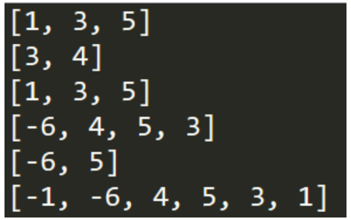
```
[1, 3, 5, 4, -6, -1] 53889544
[3, 4, -6, -1] 53889544
[3, 4, 'hello', -1] 53889544
```

List: slice with step

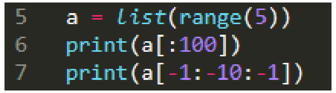
- Another usage of list slice is to set the step size from the start to the stop:
  a[start: stop: step]   (stop is not included!)
- We get the slice a[start], a[start+step], ….
- When step =1, step can be skipped. step could be negative
- a[::-1] returns the inverse of a



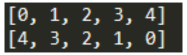
```
a = [1, 3, 5, 4, -6, -1]
print(a[0:3:1])
print(a[1:5:2])
print(a[0:3:])
print(a[4:0:-1])
print(a[4:0:-2])
print(a[::-1])
```



```
[1, 3, 5]
[3, 4]
[1, 3, 5]
[-6, 4, 5, 3]
[-6, 5]
[-1, -6, 4, 5, 3, 1]
```

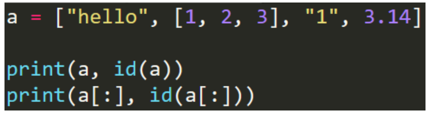


```
a = list(range(5))
print(a[:100])
print(a[-1:-10:-1])
```

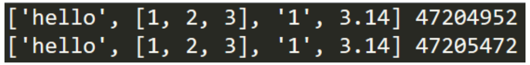


```
[0, 1, 2, 3, 4]
[4, 3, 2, 1, 0]
```

不会越界



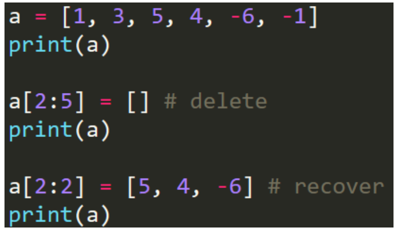
```
a = ["hello", [1, 2, 3], "1", 3.14]

print(a, id(a))
print(a[:], id(a[:]))
```



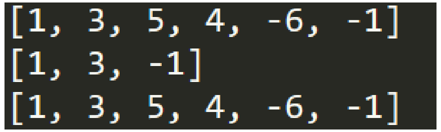
```
['hello', [1, 2, 3], '1', 3.14] 47204952
['hello', [1, 2, 3], '1', 3.14] 47205472
```

List: slice (cont'd)

- We can remove elements from a list by assigning the empty list to the slice
- We can add elements to a list by squeezing them into an empty slice at the desired location

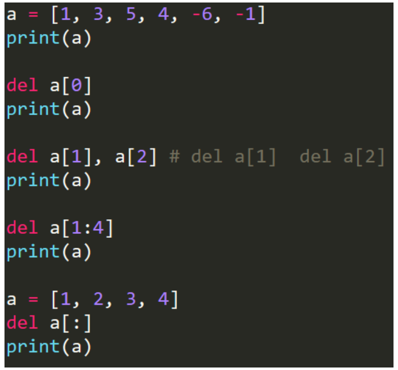


```python
a = [1, 3, 5, 4, -6, -1]
print(a)

a[2:5] = [] # delete
print(a)

a[2:2] = [5, 4, -6] # recover
print(a)
```

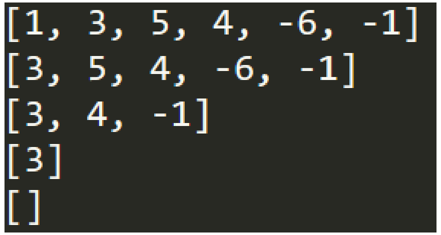


```
[1, 3, 5, 4, -6, -1]
[1, 3, -1]
[1, 3, 5, 4, -6, -1]
```

- Python provides an alternative operation del that is more readable



```python
a = [1, 3, 5, 4, -6, -1]
print(a)

del a[0]
print(a)

del a[1], a[2] # del a[1]  del a[2]
print(a)

del a[1:4]
print(a)

a = [1, 2, 3, 4]
del a[:]
print(a)
```

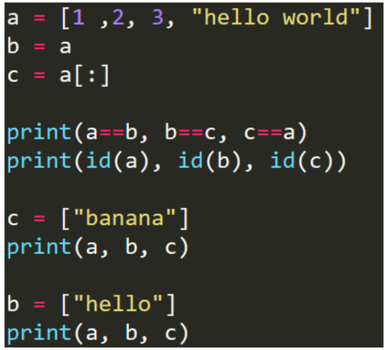


```
[1, 3, 5, 4, -6, -1]
[3, 5, 4, -6, -1]
[3, 4, -1]
[3]
[]
```

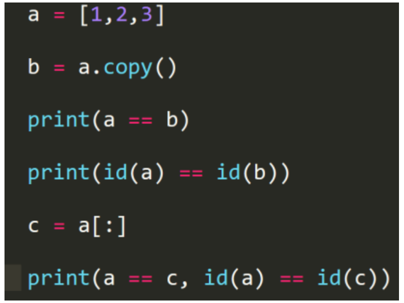
How about del a?

List: cloning

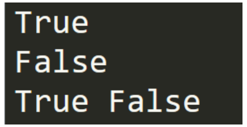
- If we want to modify a list and keep a copy of the original, we need to make a copy of the list itself
- This process is sometimes called cloning (克隆)
- The easiest way to clone a list is to use the slice operator: newlist = list[:]
- newlist = list.copy()



```python
a = [1 ,2, 3, "hello world"]
b = a
c = a[:]

print(a==b, b==c, c==a)
print(id(a), id(b), id(c))

c = ["banana"]
print(a, b, c)

b = ["hello"]
print(a, b, c)
```

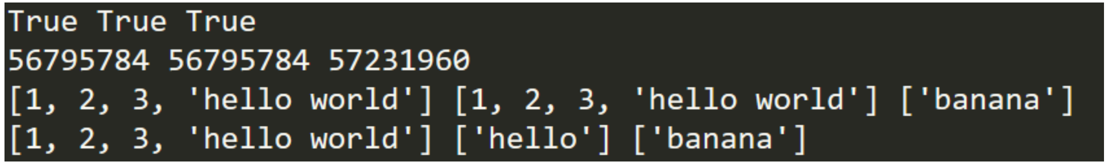


```python
a = [1,2,3]

b = a.copy()

print(a == b)

print(id(a) == id(b))

c = a[:]

print(a == c, id(a) == id(c))
```



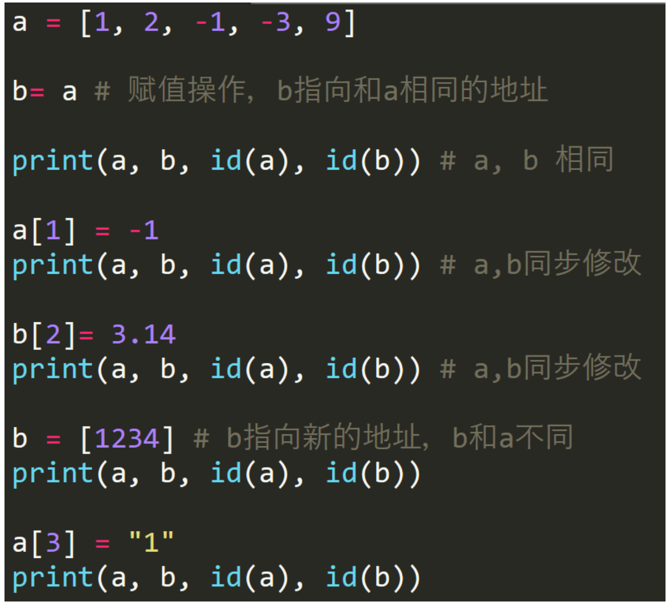
```
True
False
True False
```

思考：List 中，何时是生成新的，何时是直接修改？

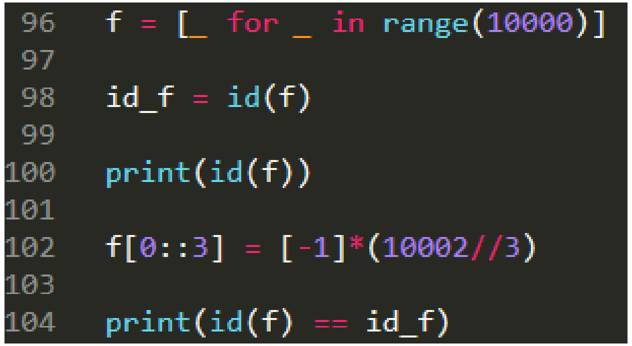


```
True True True
56795784 56795784 57231960
[1, 2, 3, 'hello world'] [1, 2, 3, 'hello world'] ['banana']
[1, 2, 3, 'hello world'] ['hello'] ['banana']
```

重新赋值 Vs. 修改

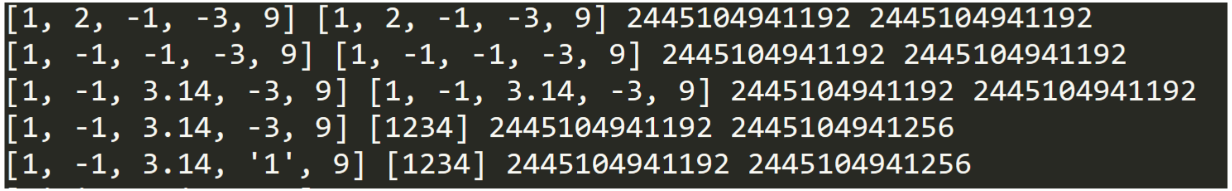


```python
a = [1, 2, -1, -3, 9]

b= a # 赋值操作，b指向和a相同的地址

print(a, b, id(a), id(b)) # a, b 相同

a[1] = -1
print(a, b, id(a), id(b)) # a,b同步修改

b[2]= 3.14
print(a, b, id(a), id(b)) # a,b同步修改

b = [1234] # b指向新的地址，b和a不同
print(a, b, id(a), id(b))

a[3] = "1"
print(a, b, id(a), id(b))
```



```python
f = [_ for _ in range(10000)]

id_f = id(f)

print(id(f))

f[0::3] = [-1]*(10002//3)

print(id(f) == id_f)
```

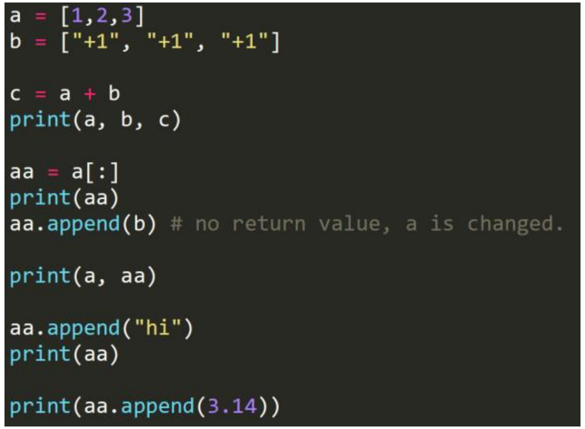
In place修改，节省空间



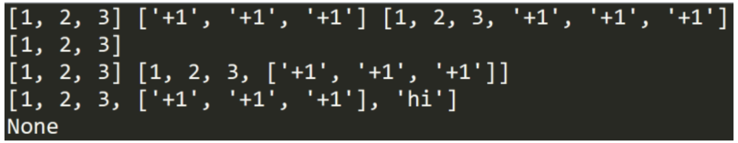
```
[1, 2, -1, -3, 9] [1, 2, -1, -3, 9] 2445104941192 2445104941192
[1, -1, -1, -3, 9] [1, -1, -1, -3, 9] 2445104941192 2445104941192
[1, -1, 3.14, -3, 9] [1, -1, 3.14, -3, 9] 2445104941192 2445104941192
[1, -1, 3.14, -3, 9] [1234] 2445104941192 2445104941256
[1, -1, 3.14, '1', 9] [1234] 2445104941192 2445104941256
```

append() 追加、添加

- The append() method appends an element to the end of the list itself
  - No return 没有返回值
  - The list itself will be changed



```
a = [1,2,3]
b = ["+1", "+1", "+1"]

c = a + b
print(a, b, c)

aa = a[:]
print(aa)
aa.append(b) # no return value, a is changed.

print(a, aa)

aa.append("hi")
print(aa)

print(aa.append(3.14))
```

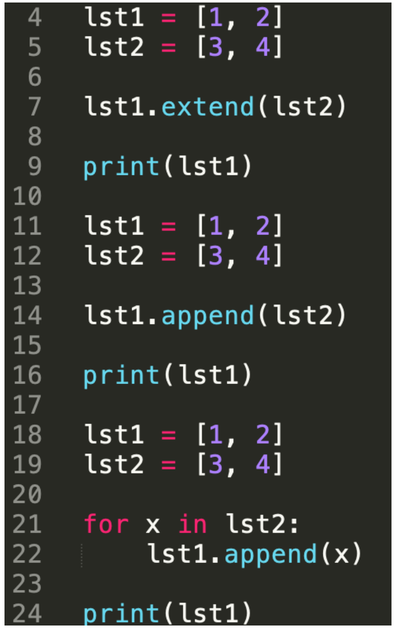


```
[1, 2, 3] ['+1', '+1', '+1'] [1, 2, 3, '+1', '+1', '+1']
[1, 2, 3]
[1, 2, 3] [1, 2, 3, ['+1', '+1', '+1']]
[1, 2, 3, ['+1', '+1', '+1'], 'hi']
None
```

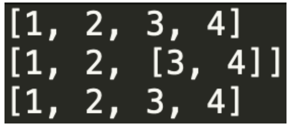
最最最常见错误
a=a.append(x)

extend() 扩展

- The extend() method adds all the elements of an iterable (list, tuple, string etc.) to the end of the list.
  - The same with append(). No return. Modified in place. 没有返回值



```python
lst1 = [1, 2]
lst2 = [3, 4]

lst1.extend(lst2)

print(lst1)

lst1 = [1, 2]
lst2 = [3, 4]

lst1.append(lst2)

print(lst1)

lst1 = [1, 2]
lst2 = [3, 4]

for x in lst2:
    lst1.append(x)

print(lst1)
```



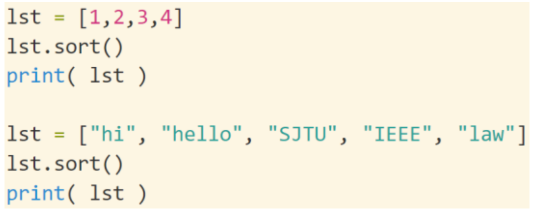
```
[1, 2, 3, 4]
[1, 2, [3, 4]]
[1, 2, 3, 4]
```

辨析extend和append的区别

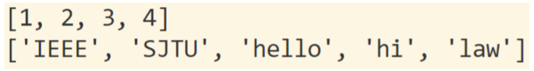
sort() 排序

- The sort() method will sort the list by ascending order. 按照从小到大排序
  - list.sort() has no return value
  - The elements should be the same type
- Parameters

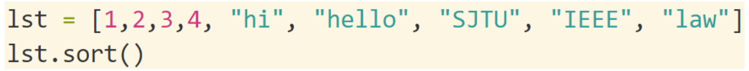
reverse	Optional. reverse=True will sort the list descending. Default is reverse=False
key	Optional. A function to specify the sorting criteria(s)



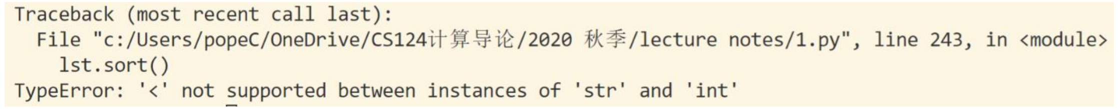
```
lst = [1,2,3,4]
lst.sort()
print( lst )

lst = ["hi", "hello", "SJTU", "IEEE", "law"]
lst.sort()
print( lst )
```



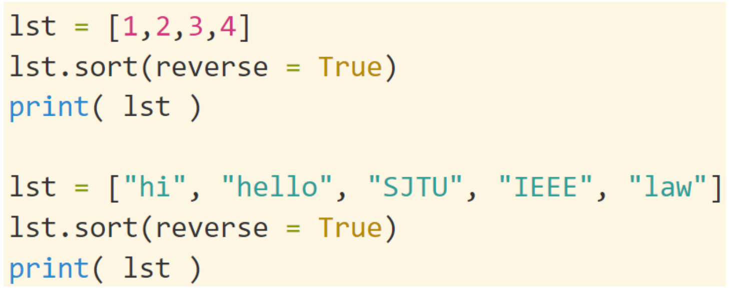
```
[1, 2, 3, 4]
['IEEE', 'SJTU', 'hello', 'hi', 'law']
```



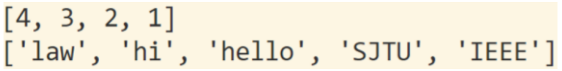
```
lst = [1,2,3,4, "hi", "hello", "SJTU", "IEEE", "law"]
lst.sort()
```



```
Traceback (most recent call last):
  File "c:/Users/popeC/OneDrive/CS124计算导论/2020 秋季/lecture notes/1.py", line 243, in <module>
    lst.sort()
TypeError: '<' not supported between instances of 'str' and 'int'
```



```
lst = [1,2,3,4]
lst.sort(reverse = True)
print( lst )

lst = ["hi", "hello", "SJTU", "IEEE", "law"]
lst.sort(reverse = True)
print( lst )
```



```
[4, 3, 2, 1]
['law', 'hi', 'hello', 'SJTU', 'IEEE']
```

Nested list

- A list can contain another list as its element: a=[…, […], …]
- To access the inside list, a[i][j]
- One way to create a matrix: matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]] (Not recommended)

```
a = ["hello", [1, 2, 3], "1", 3.14]
print(a)
print(a[1])
print(a[1][0])
print(a[1][2])
```

```
['hello', [1, 2, 3], '1', 3.14]
[1, 2, 3]
1
3
```

```
a[1][2] = "world"
print(a)
```

```
['hello', [1, 2, 'world'], '1', 3.14]
```

列表的每个元素都是一个变量，都指向一个内存区域

思考题：写出如下程序的运行结果

```
a = [1]*3
a[1] = -1
print(a)

a=[[1]]*3
a[1] = -1
print(a)

a=[[1]]*3
a[1][0] = -1
print(a)

a = [[1]*3]*3
a[1][1] = 3.14
print(a)
```

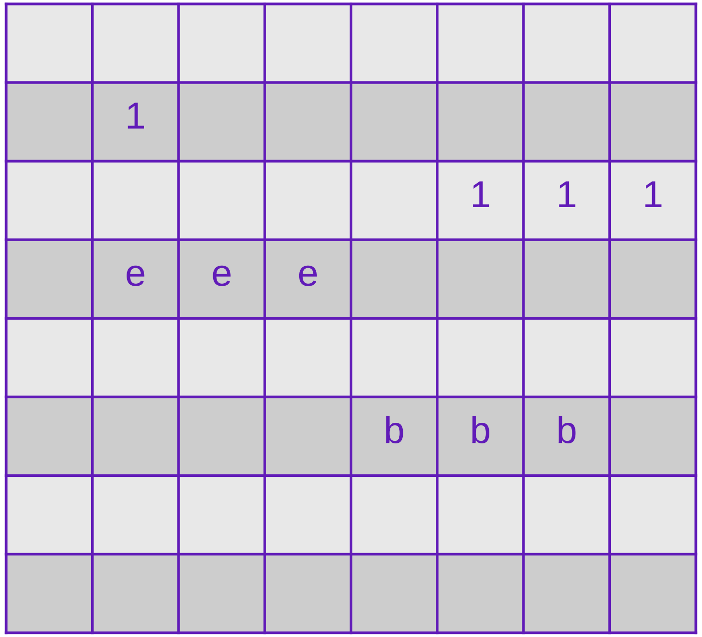
Remove to see
the answer

```python
a = [1, [1,2,3] ,3]
b = a
print(a==b, id(a)==id(b))

c = a.copy()
print(c==a, id(c)==id(a))
print(id(c[0])==id(a[0]), id(c[1])==id(a[1]), id(c[2])==id(a[2]))
```

思考题：为什么a和c的id不一样，
但是它们的每个元素的id一样

```
True True
True False
True True True
```

每个元素保存的是id



[1,1,1]

e=[1]

b=[1,1,1]

```python
a = [1]*3
a[1] = -1
print(a)

a=[[1]]*3
a[1] = -1
print(a)

a=[[1]]*3
a[1][0] = -1
print(a)

a = [[1]*3]*3
a[1][1] = 3.14
print(a)
```

“=”: 重新赋值 or 修改

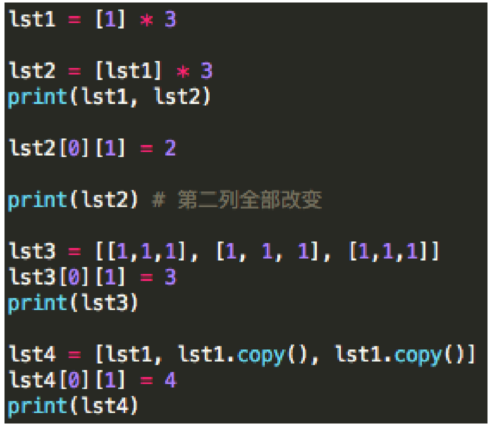
```
[1, -1, 1]
[[1], -1, [1]]
[[-1], [-1], [-1]]
[[1, 3.14, 1], [1, 3.14, 1], [1, 3.14, 1]]
```

Nested list

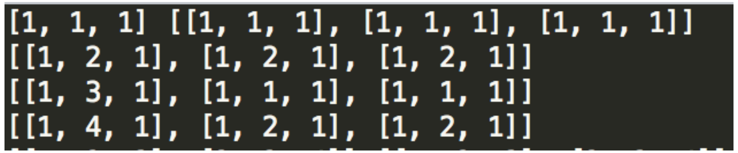
后代的后代的后代可能是同一个人

Nested list非常非常非常容易出错：地址的地址的地址……可能是同一个内存区域

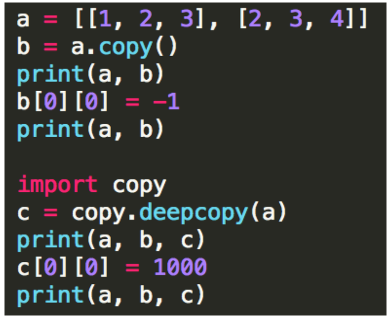
1. a.copy() 对于单层list
2. deepcopy() 对于nested list (https://docs.python.org/3/library/copy.html)



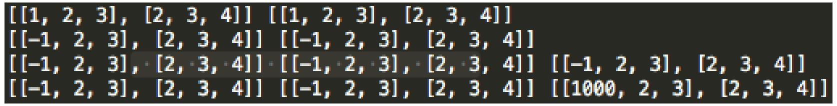
```
lst1 = [1] * 3

lst2 = [lst1] * 3
print(lst1, lst2)

lst2[0][1] = 2

print(lst2) # 第二列全部改变

lst3 = [[1,1,1], [1, 1, 1], [1,1,1]]
lst3[0][1] = 3
print(lst3)

lst4 = [lst1, lst1.copy(), lst1.copy()]
lst4[0][1] = 4
print(lst4)
```



```
[1, 1, 1] [[1, 1, 1], [1, 1, 1], [1, 1, 1]]
[[1, 2, 1], [1, 2, 1], [1, 2, 1]]
[[1, 3, 1], [1, 1, 1], [1, 1, 1]]
[[1, 4, 1], [1, 2, 1], [1, 2, 1]]
```



```
a = [[1, 2, 3], [2, 3, 4]]
b = a.copy()
print(a, b)
b[0][0] = -1
print(a, b)

import copy
c = copy.deepcopy(a)
print(a, b, c)
c[0][0] = 1000
print(a, b, c)
```



```
[[1, 2, 3], [2, 3, 4]] [[1, 2, 3], [2, 3, 4]]
[[-1, 2, 3], [2, 3, 4]] [[-1, 2, 3], [2, 3, 4]]
[[-1, 2, 3], [2, 3, 4]] [[-1, 2, 3], [2, 3, 4]] [[-1, 2, 3], [2, 3, 4]]
[[-1, 2, 3], [2, 3, 4]] [[-1, 2, 3], [2, 3, 4]] [[1000, 2, 3], [2, 3, 4]]
```

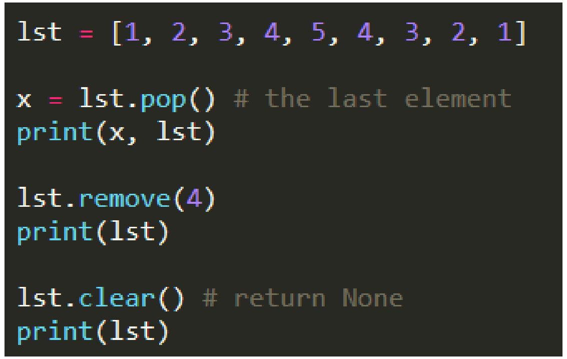
注意：list某些地方
保存的 是（地址的地址）

- copy () 的时候，虽然a 和 b指向不同的内存块，但是这两个内存块保存了相同的内容。所以对于嵌套的list，两次下标操作[][]，修改b也就修改了a
- deepcopy会把内容全部复制一遍，无论有多少层地址（所有的后代都会被复制）

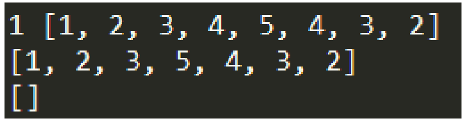
List: Remove an Item

Remove an item from a list in Python (clear, pop, remove, del)

- Remove all items: clear()
- Remove an item by index and get its value: pop(index=-1)
  - In default, the last element
- Remove an item by value: remove(value)
  - Remove the first element that matches the value
- Remove items by index or slice: del
- Remove multiple items that meet the condition: List comprehensions



```python
lst = [1, 2, 3, 4, 5, 4, 3, 2, 1]

x = lst.pop() # the last element
print(x, lst)

lst.remove(4)
print(lst)

lst.clear() # return None
print(lst)
```

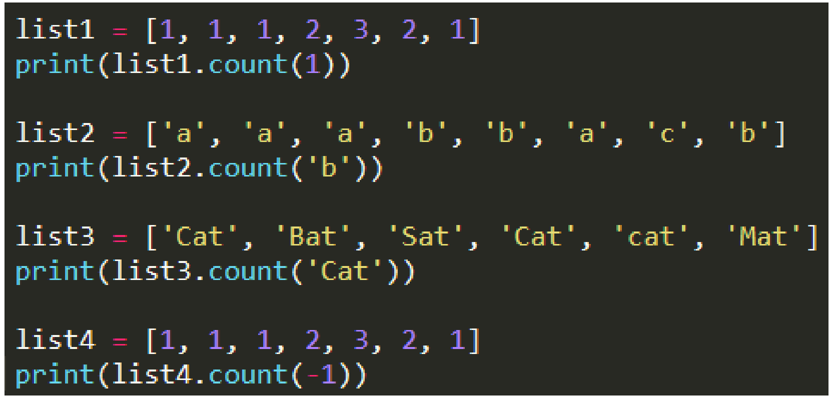


```
1 [1, 2, 3, 4, 5, 4, 3, 2]
[1, 2, 3, 5, 4, 3, 2]
[]
```

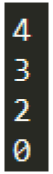
List: count() method

count() returns the count of how many times a given item occurs in a List.

- list_name.count(object)



```
list1 = [1, 1, 1, 2, 3, 2, 1]
print(list1.count(1))

list2 = ['a', 'a', 'a', 'b', 'b', 'a', 'c', 'b']
print(list2.count('b'))

list3 = ['Cat', 'Bat', 'Sat', 'Cat', 'cat', 'Mat']
print(list3.count('Cat'))

list4 = [1, 1, 1, 2, 3, 2, 1]
print(list4.count(-1))
```



```
4
3
2
0
```

List comprehension (列表推导)

- Grammar

  单重循环 [expression for target in iterable if conditon]

  多重循环 [expression for target1 in iterable1 if condition1

  for target2 in iterable2 if condition2 ..

  for targetN in iterableN if conditionN]

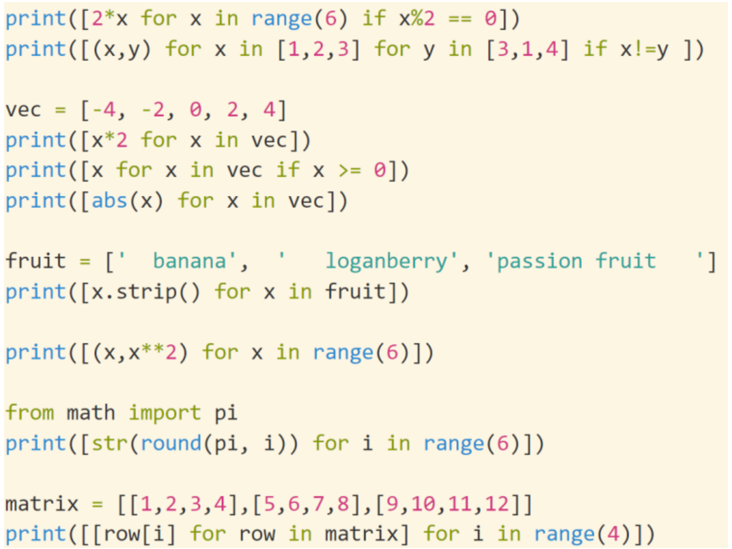
  - A list comprehension consists of brackets [ ] containing an expression followed by a for clause, then zero or more for or if clauses
  - The result will be a new list resulting from evaluating the expression in the context of the for and if clauses which follow it
  - List comprehension can be nested

```python
squares = []

for x in range(10):
    squares.append(x**2)
print(squares)

squares = [x**2 for x in range(10)]
print(squares)
```

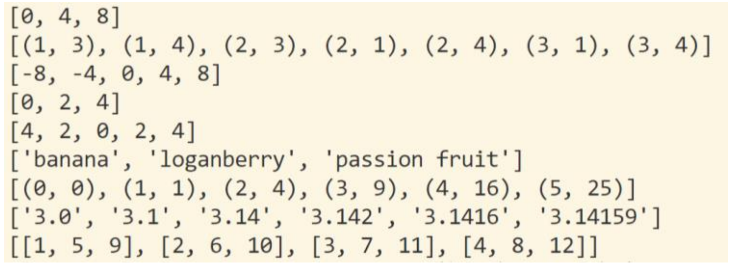
```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Pythonic

Sample Code



```python
print([2*x for x in range(6) if x%2 == 0])
print([(x,y) for x in [1,2,3] for y in [3,1,4] if x!=y ])

vec = [-4, -2, 0, 2, 4]
print([x*2 for x in vec])
print([x for x in vec if x >= 0])
print([abs(x) for x in vec])

fruit = ['  banana', '   loganberry', 'passion fruit   ']
print([x.strip() for x in fruit])

print([(x,x**2) for x in range(6)])

from math import pi
print([str(round(pi, i)) for i in range(6)])

matrix = [[1,2,3,4],[5,6,7,8],[9,10,11,12]]
print([[row[i] for row in matrix] for i in range(4)])
```



```
[0, 4, 8]
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
[-8, -4, 0, 4, 8]
[0, 2, 4]
[4, 2, 0, 2, 4]
['banana', 'loganberry', 'passion fruit']
[(0, 0), (1, 1), (2, 4), (3, 9), (4, 16), (5, 25)]
['3.0', '3.1', '3.14', '3.142', '3.1416', '3.14159']
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

如何创建新的list

- for
- list comprehension
- [0]*n
- list(range()), list(set), list(str)

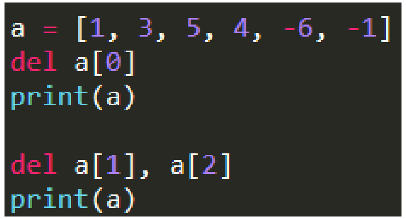
推荐List Comprehension

✓ 根据需要合理使用
✓ 清晰，可读，正确
✓ 比for更快

Pythonic

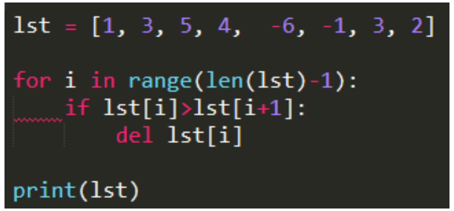
List: 删除陷阱

列表删除一个元素后，会改变后面元素的序号，特别容易出错

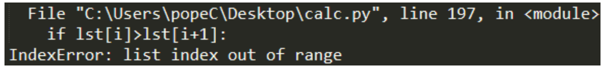


```python
a = [1, 3, 5, 4, -6, -1]
del a[0]
print(a)

del a[1], a[2]
print(a)
```

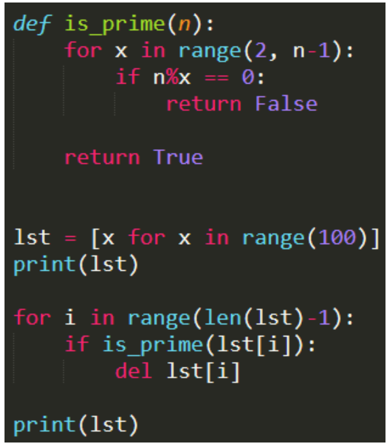
a[2]是原来的”a[3]”



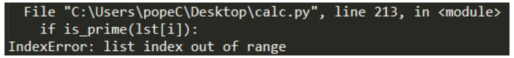
```python
lst = [1, 3, 5, 4,  -6, -1, 3, 2]

for i in range(len(lst)-1):
    if lst[i]>lst[i+1]:
        del lst[i]

print(lst)
```



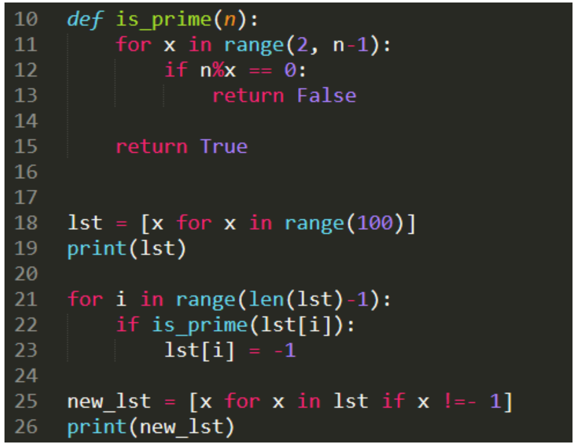
```
  File "C:\Users\popeC\Desktop\calc.py", line 197, in <module>
    if lst[i]>lst[i+1]:
IndexError: list index out of range
```



```python
def is_prime(n):
    for x in range(2, n-1):
        if n%x == 0:
            return False

    return True


lst = [x for x in range(100)]
print(lst)

for i in range(len(lst)-1):
    if is_prime(lst[i]):
        del lst[i]

print(lst)
```



```
  File "C:\Users\popeC\Desktop\calc.py", line 213, in <module>
    if is_prime(lst[i]):
IndexError: list index out of range
```

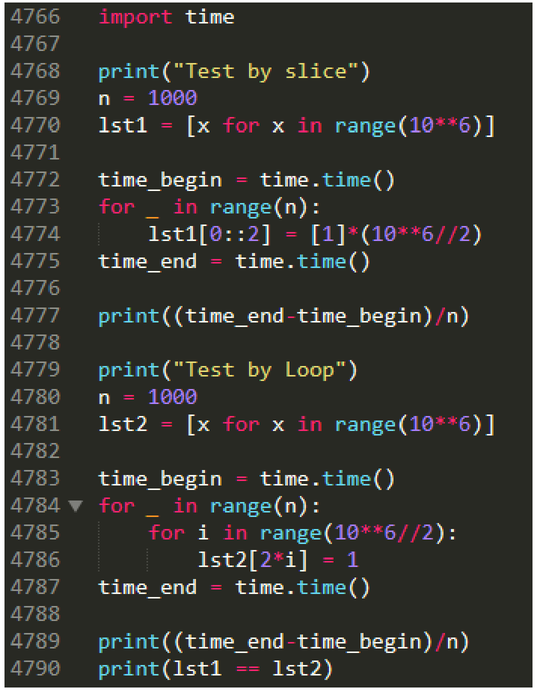


```python
def is_prime(n):
    for x in range(2, n-1):
        if n%x == 0:
            return False

    return True


lst = [x for x in range(100)]
print(lst)

for i in range(len(lst)-1):
    if is_prime(lst[i]):
        lst[i] = -1

new_lst = [x for x in lst if x !=- 1]
print(new_lst)
```

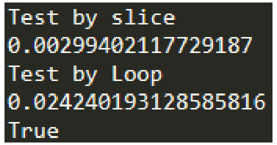
尽可能不要循环动态删除一个数据结构，很容易出bug

1. 列表在内存中是顺序排列的，删除后要重新恢复，速度慢
2. 标记为不存在的元素：例如-1，或者None

多用切片，快



```python
import time

print("Test by slice")
n = 1000
lst1 = [x for x in range(10**6)]

time_begin = time.time()
for _ in range(n):
    lst1[0::2] = [1]*(10**6//2)
time_end = time.time()

print((time_end-time_begin)/n)

print("Test by Loop")
n = 1000
lst2 = [x for x in range(10**6)]

time_begin = time.time()
for _ in range(n):
    for i in range(10**6//2):
        lst2[2*i] = 1
time_end = time.time()

print((time_end-time_begin)/n)
print(lst1 == lst2)
```



```
Test by slice
0.0029940211772918
Test by Loop
0.024240193128585816
True
```

大概相差一个数量级

Quick Test

- Given two strings: str1 = “Hello World”, str2 = “Machine Learning”
    - Create two lists list1and list2, from str1 and str2, respectively
    - Reverse list2
    - Print the characters of list1 at the even positions
    - Concatenate list1 and list2
    - Repeat list1 3 times and list2 twice
- How to extract a list from a string, e.g., “[1,2,3,-1,-5, [1,2,3]]” to [1,2,3,-1,-5, [1,2,3]]