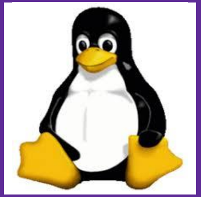
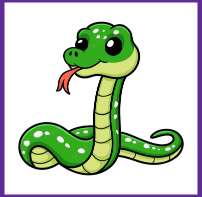
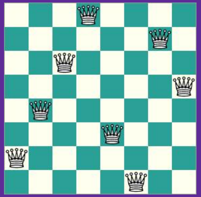
Introduction to Computation

Autumn, 2024

Prof. Hongfei Fu
Shanghai Jiao Tong University
(slides by Prof. Fang Cheng)

fuhf@cs.sjtu.edu.cn
https://jhc.sjtu.edu.cn/~hongfeifu/
https://github.com/ichengfan/itc

5

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)

高效社会管理

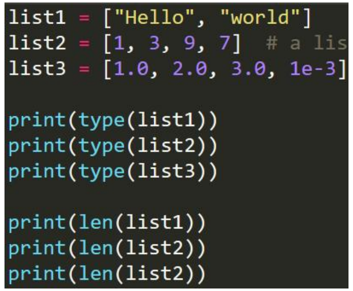
List 列表

如何处理10^{20}个整数？定义10^{20}个变量？

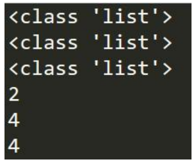
List: creation

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

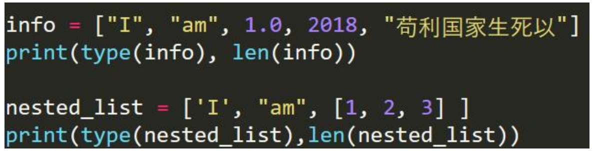
- list1 = [“Hello”, “world”]   # a list with two string
  list2 = [1, 3, 9, 7] # a list with four integers
  list3 = [1.0, 2.0, 3.0, 1e-3]   # a list with four floats
- The elements of a list don’t have to be the same type
- A list within another list is called to be nested (嵌套)
- Recall: assignment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



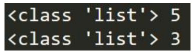
```
list1 = ["Hello", "world"]
list2 = [1, 3, 9, 7]  # a lis
list3 = [1.0, 2.0, 3.0, 1e-3]

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

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



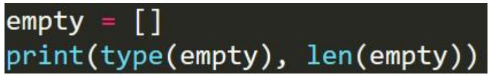
```
<class 'list'>
<class 'list'>
<class 'list'>
2
4
4
```



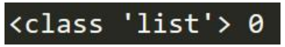
```
info = ["I", "am", 1.0, 2018, "苟利国家生死以"]
print(type(info), len(info))

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))
```

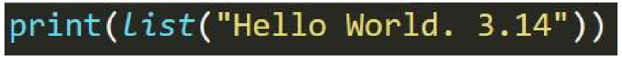


```
<class 'list'> 0
```

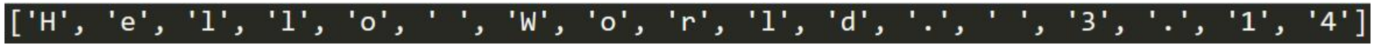
严格按照语法：[, , ,], 不是{, , ,}, 也不是[; ; ;]

List: list()构造函数

- 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

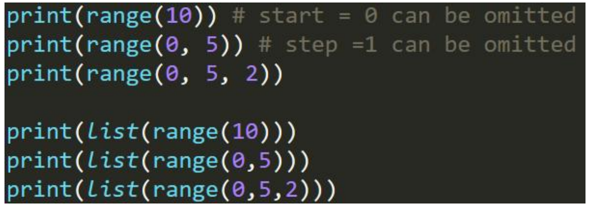


```
print(list("Hello World. 3.14"))
```

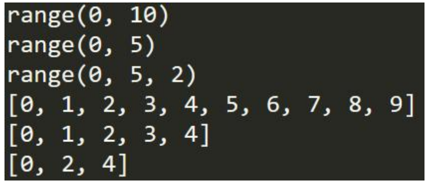


```
['H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd', '.', ' ', '3', '.', '1', '4']
```

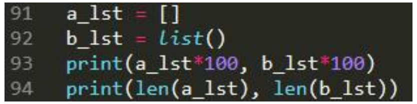
- List with range(). range(start, stop, step) returns a sequence from start to stop with step
    - When start = 0, start can be omitted
    - When step = 1, step can be omitted
    - Together with list(), it generates a list from start to stop with step



```
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 (序数，顺序排列)
- Elements in the list (a) are ordered starting from 0, 1, …, len(a)-1 （超出这个范围就是越界错误）
- To access or modify 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
```

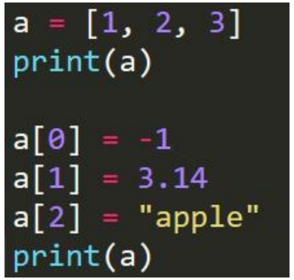
```
print(info[-6])
```

```
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
```

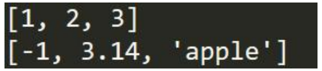
世界上最常见的错误: out of range

Elements Modification

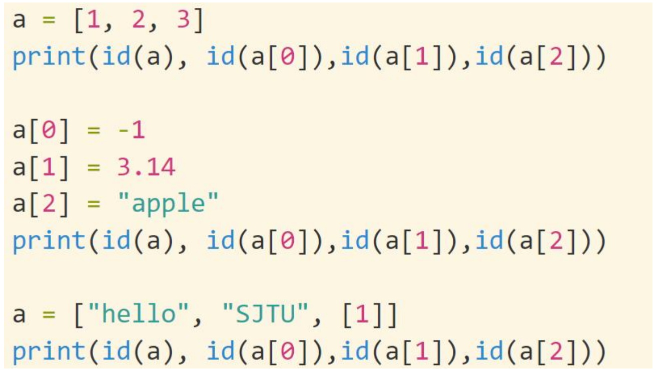
- Assignment statement could be used to modify the elements of a list
  - list[i] = new_element
- List is mutable (可修改的) . However, the id remains unchanged.
  - That is, the content of the list is changed, not its location



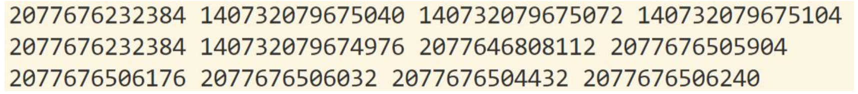
```python
a = [1, 2, 3]
print(a)

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



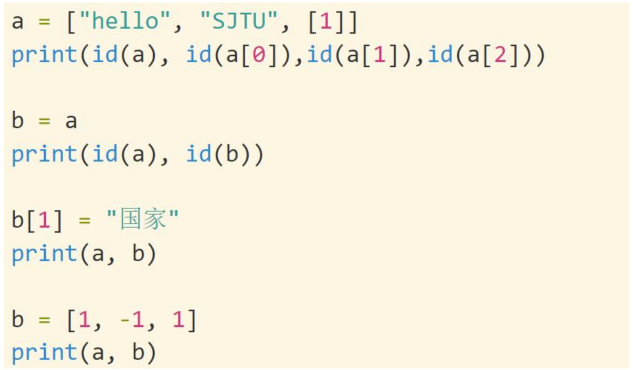
```
[1, 2, 3]
[-1, 3.14, 'apple']
```



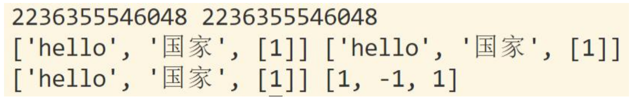
```python
a = [1, 2, 3]
print(id(a), id(a[0]),id(a[1]),id(a[2]))

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)
```

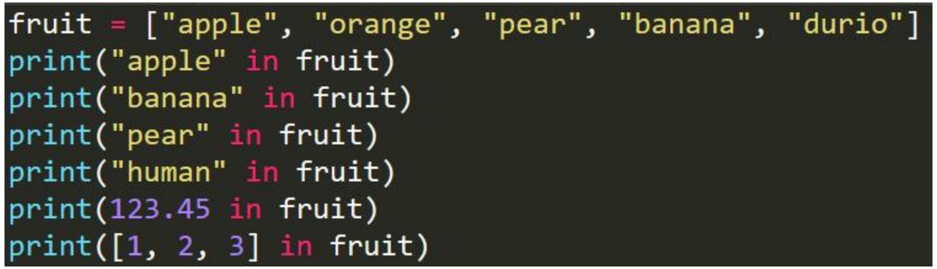


```
2236355546048 2236355546048
['hello', '国家', [1]] ['hello', '国家', [1]]
['hello', '国家', [1]] [1, -1, 1]
```

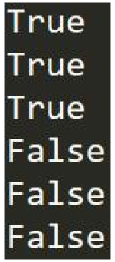
List可修改，重新赋值和修改差别很大
数据是否可以被修改是一个关键因素

List: membership

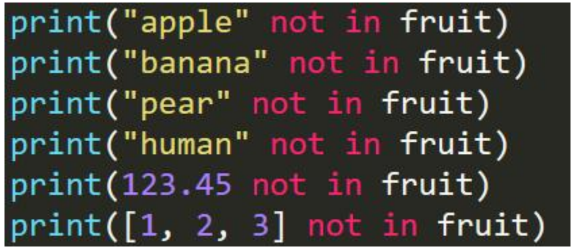
- One can use in and not in to test whether an element belonging to a list
- in: x in S, return True if $x \in S$, otherwise False
- not in: x not in S, return True if $x \notin S$, otherwise False



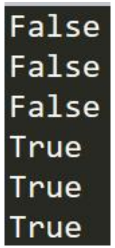
```
fruit = ["apple", "orange", "pear", "banana", "durio"]
print("apple" in fruit)
print("banana" in fruit)
print("pear" in fruit)
print("human" in fruit)
print(123.45 in fruit)
print([1, 2, 3] in fruit)
```



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



```
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

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()
```

可迭代的(iterable)

List: +, *

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

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

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

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

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

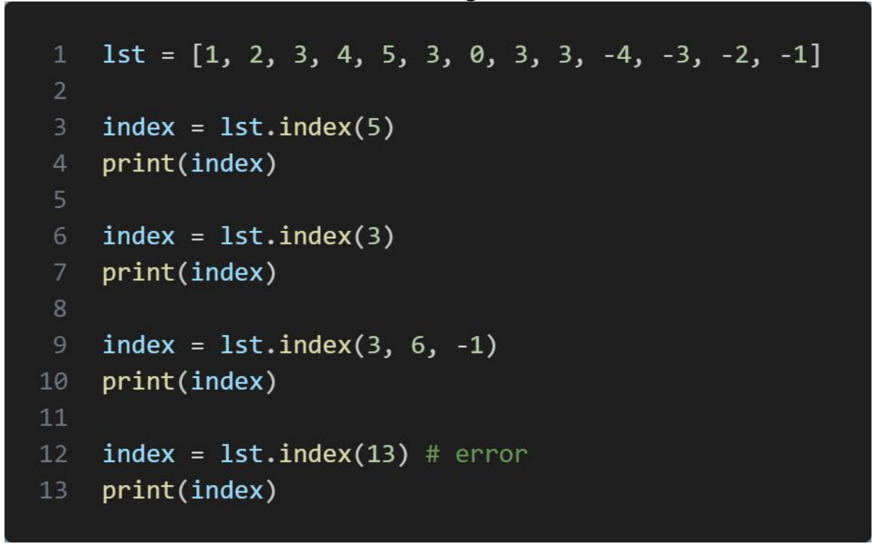
- *: a * n will repeat a list a given number of times: n

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

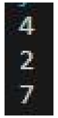
```
[0, 0, 0]
[3.14, 2.718282, 1j, 3.14, 2.718282, 1j, 3.14, 2.718282, 1j]
['+1s', '+1s', '+1s', '+1s', '+1s', '+1s', '+1s']
```

List.index()

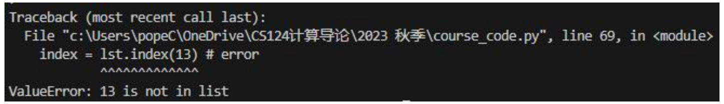
- list.index(x[, start[, end]])
    - Return zero-based index in the list of the first item whose value is equal to x. Raises a ValueError if there is no such item.
    - The optional arguments start and end are interpreted as in the slice notation and are used to limit the search to a particular subsequence of the list. The returned index is computed relative to the beginning of the full sequence rather than the start argument.



```python
lst = [1, 2, 3, 4, 5, 3, 0, 3, 3, -4, -3, -2, -1]

index = lst.index(5)
print(index)

index = lst.index(3)
print(index)

index = lst.index(3, 6, -1)
print(index)

index = lst.index(13) # error
print(index)
```



```
4
2
7
```

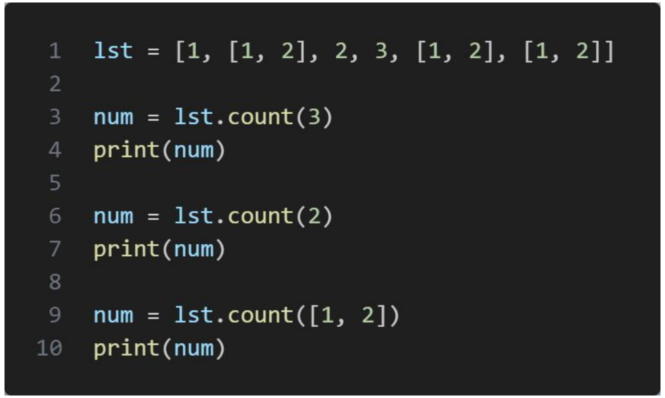


```
Traceback (most recent call last):
  File "c:\Users\popeC\OneDrive\CS124计算导论\2023 秋季\course_code.py", line 69, in <module>
    index = lst.index(13) # error
            ^^^^^^^^^^^^^
ValueError: 13 is not in list
```

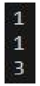
类函数的调用: Class.func()

List.count()

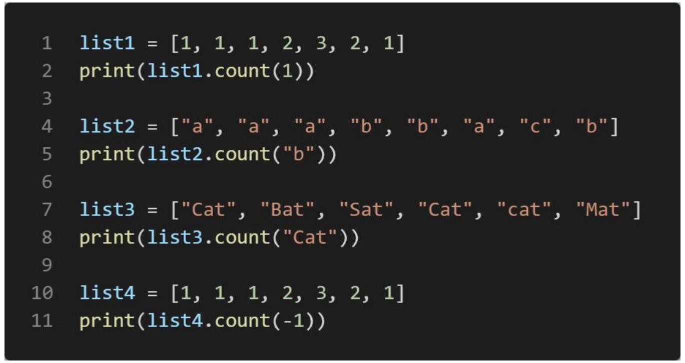
- list.count(x)
  - Return the number of times x appears in the list.



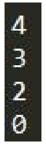
```python
lst = [1, [1, 2], 2, 3, [1, 2], [1, 2]]

num = lst.count(3)
print(num)

num = lst.count(2)
print(num)

num = lst.count([1, 2])
print(num)
```



```
1
1
3
```

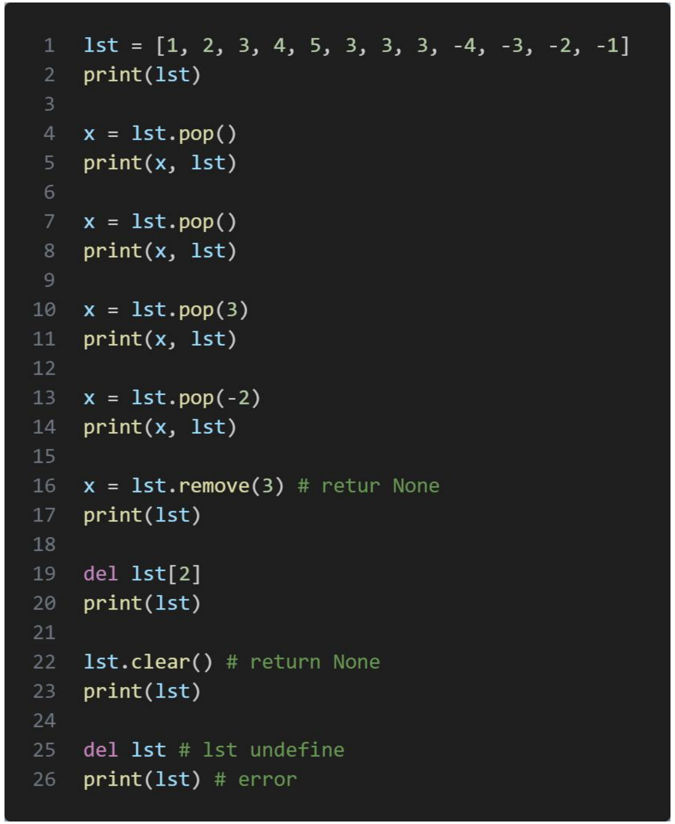


```python
list1 = [1, 1, 1, 2, 3, 2, 1]
print(list1.count(1))

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
```

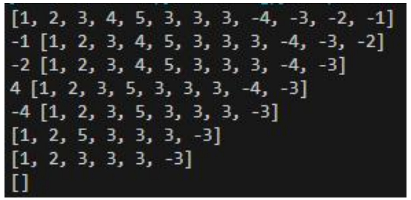
Remove an Item



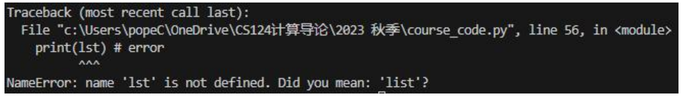
```python
lst = [1, 2, 3, 4, 5, 3, 3, 3, -4, -3, -2, -1]
print(lst)

x = lst.pop()
print(x, lst)

x = lst.pop()
print(x, lst)

x = lst.pop(3)
print(x, lst)

x = lst.pop(-2)
print(x, lst)

x = lst.remove(3) # retur None
print(lst)

del lst[2]
print(lst)

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

del lst # lst undefine
print(lst) # error
```

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
- del x: Delete variable x from Python system
  - Remove items by index or slice: del
- Remove multiple items that meet the condition: List comprehensions



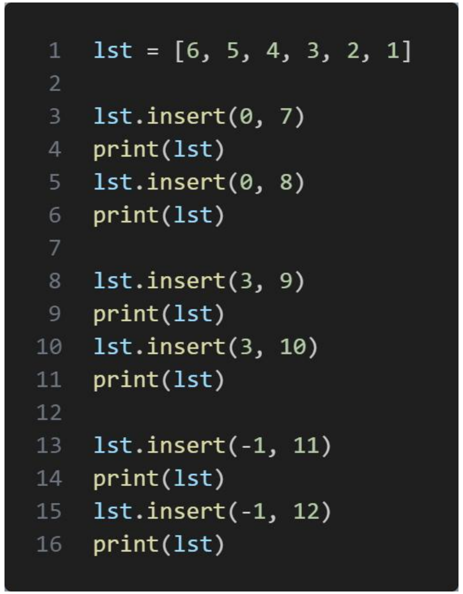
```
[1, 2, 3, 4, 5, 3, 3, 3, -4, -3, -2, -1]
-1 [1, 2, 3, 4, 5, 3, 3, 3, -4, -3, -2]
-2 [1, 2, 3, 4, 5, 3, 3, 3, -4, -3]
4 [1, 2, 3, 5, 3, 3, 3, -4, -3]
-4 [1, 2, 3, 5, 3, 3, 3, -3]
[1, 2, 5, 3, 3, 3, -3]
[1, 2, 3, 3, 3, -3]
[]
```



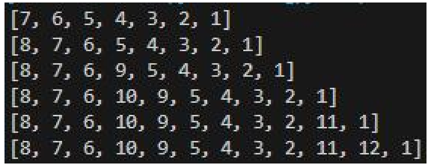
```
Traceback (most recent call last):
  File "c:\Users\popeC\OneDrive\CS124计算导论\2023 秋季\course_code.py", line 56, in <module>
    print(lst) # error
          ^^^
NameError: name 'lst' is not defined. Did you mean: 'list'?
```

List.insert()

- list.insert(i, x): Insert an item x at a given position i. The first argument is the index of the element before which to insert.
  - Return None. Modification list in place
  - a.insert(0, x) inserts at the front of the list, and a.insert(len(a), x) is equivalent to a.append(x)



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

lst.insert(0, 7)
print(lst)
lst.insert(0, 8)
print(lst)

lst.insert(3, 9)
print(lst)
lst.insert(3, 10)
print(lst)

lst.insert(-1, 11)
print(lst)
lst.insert(-1, 12)
print(lst)
```

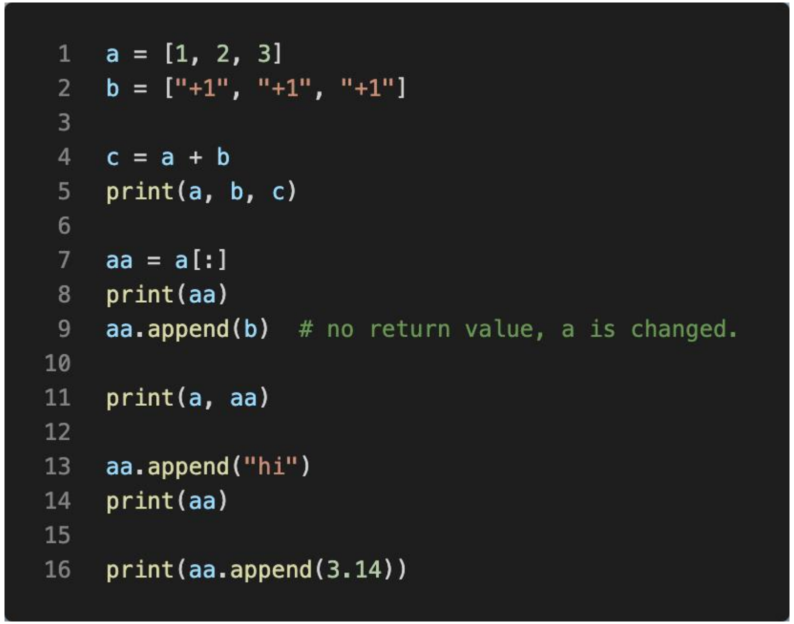


```
[7, 6, 5, 4, 3, 2, 1]
[8, 7, 6, 5, 4, 3, 2, 1]
[8, 7, 6, 9, 5, 4, 3, 2, 1]
[8, 7, 6, 10, 9, 5, 4, 3, 2, 1]
[8, 7, 6, 10, 9, 5, 4, 3, 2, 11, 1]
[8, 7, 6, 10, 9, 5, 4, 3, 2, 11, 12, 1]
```

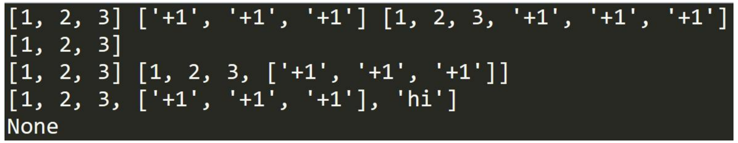
返回值为None

List.append()

- The append() method appends (追加、添加) an element to the end of the list itself
  - No return 没有返回值, return None
  - The list itself will be changed



```python
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))
```

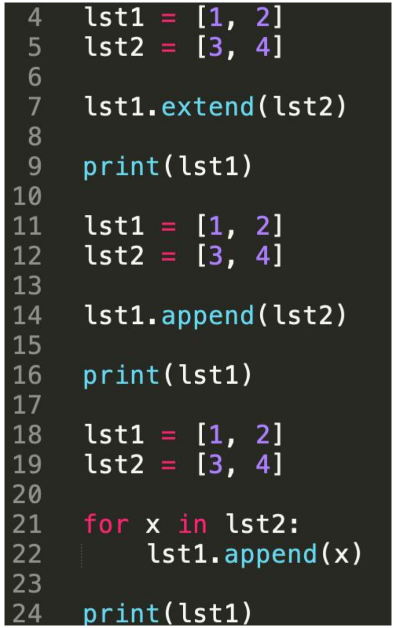


```
[1, 2, 3] ['+1', '+1', '+1'] [1, 2, 3, '+1', '+1', '+1']
[1, 2, 3]
[1, 2, 3] [1, 2, 3, ['+1', '+1', '+1']]
[1, 2, 3, ['+1', '+1', '+1'], 'hi']
None
```

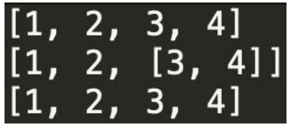
最最最常见错误 a=a.append(x)
返回值为None

List.extend()

- The extend() (扩展)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



```
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]
```

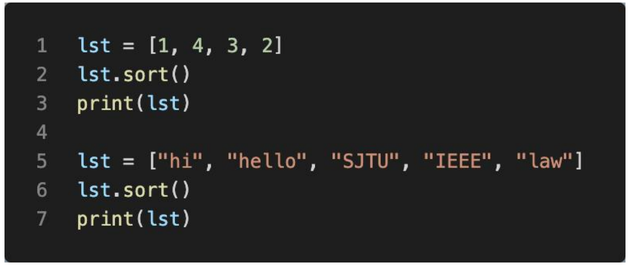
辨析extend和append的区别

返回值为None

List.sort()

- The sort() (排序)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)

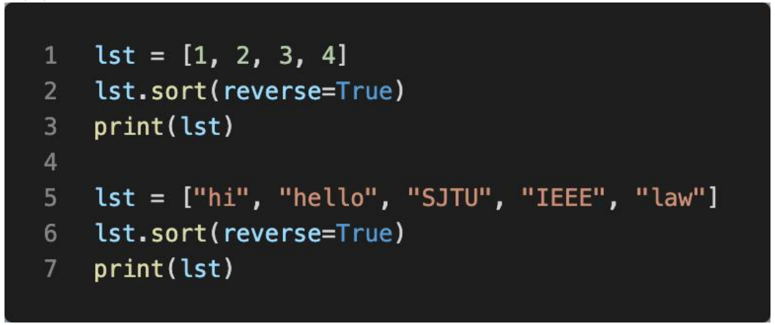


```python
lst = [1, 4, 3, 2]
lst.sort()
print(lst)

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

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

```python
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'
```



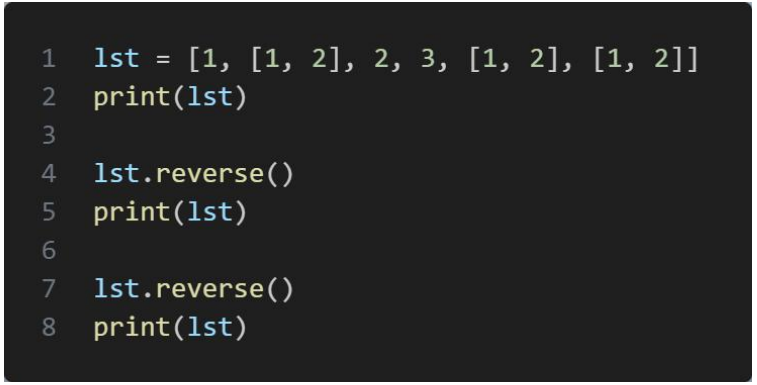
```python
lst = [1, 2, 3, 4]
lst.sort(reverse=True)
print(lst)

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

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

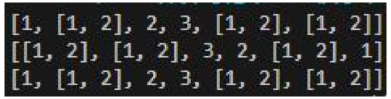
返回值为None

List.reverse()

- Reverses the elements of the list, in place
  - Return None



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

lst.reverse()
print(lst)

lst.reverse()
print(lst)
```

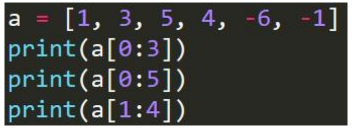


```
[1, [1, 2], 2, 3, [1, 2], [1, 2]]
[[1, 2], [1, 2], 3, 2, [1, 2], 1]
[1, [1, 2], 2, 3, [1, 2], [1, 2]]
```

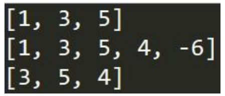
返回值为None

List: slice (切片)

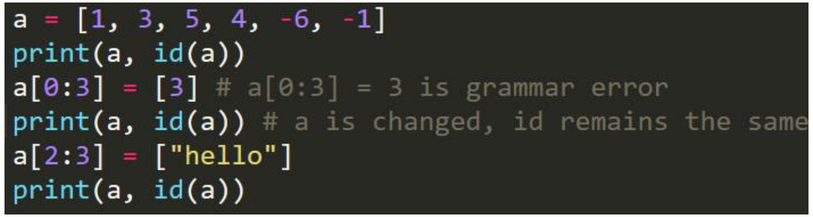
- Given a list list_a, you could get a slice of list_a via: list_a[i:j], it is the sub-list of list_a[i], list_[i+1], …, list_a[j-1]. (list_a[j] is not included!)
  - The type of list_a[i:j] is still list, which is a new copy from a
- In default, the slice is started from 0 and ended till the end of 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)
- We can update several elements at once. The types of the left and right hand should match [] = []
  - We can also remove elements from a list by assigning the empty list to the slice



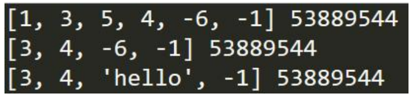
```
a = [1, 3, 5, 4, -6, -1]
print(a[0:3])
print(a[0:5])
print(a[1:4])
```



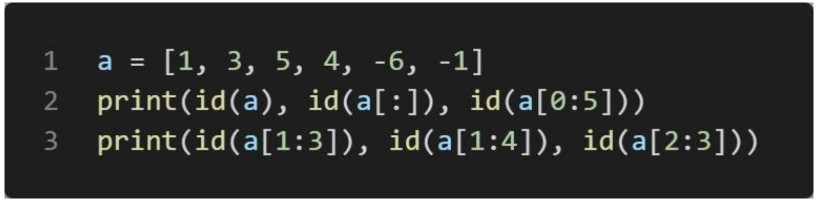
```
[1, 3, 5]
[1, 3, 5, 4, -6]
[3, 5, 4]
```



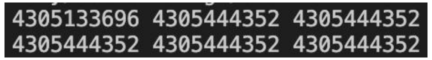
```
a = [1, 3, 5, 4, -6, -1]
print(a, id(a))
a[0:3] = [3] # a[0:3] = 3 is grammar error
print(a, id(a)) # a is changed, id remains the same
a[2:3] = ["hello"]
print(a, id(a))
```



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



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

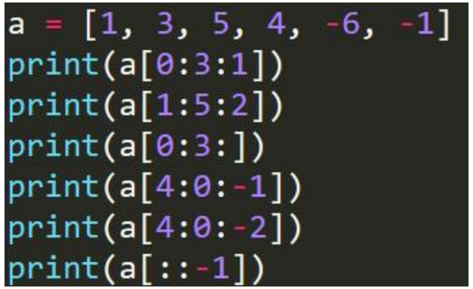


```
4305133696 4305444352 4305444352
4305444352 4305444352 4305444352
```

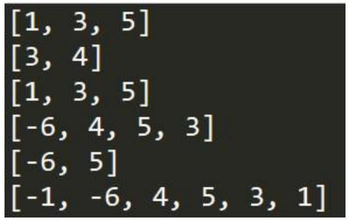
id(a[:]) ≠id(a)
a and a[:] are not in the same address, albeit they have identical elements

List: slice with step

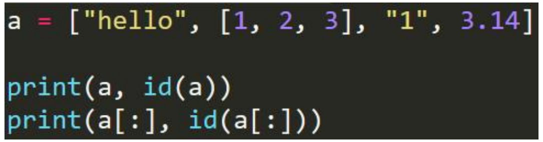
- Another usage of list slice is to set the step size from the start to the stop:
  a[start: stop: step] （stop is not included!）
- We get the slice a[start], a[start+step], ….
- When step =1, step can be skipped. step could be negative
- a[::-1] returns the inverse of a



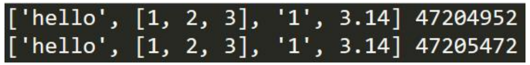
```
a = [1, 3, 5, 4, -6, -1]
print(a[0:3:1])
print(a[1:5:2])
print(a[0:3:])
print(a[4:0:-1])
print(a[4:0:-2])
print(a[::-1])
```



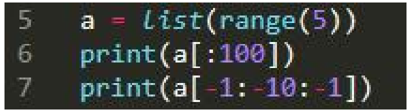
```
[1, 3, 5]
[3, 4]
[1, 3, 5]
[-6, 4, 5, 3]
[-6, 5]
[-1, -6, 4, 5, 3, 1]
```



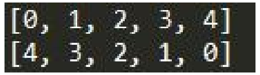
```
a = ["hello", [1, 2, 3], "1", 3.14]

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



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

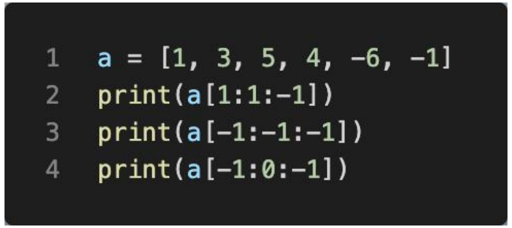


```
a = list(range(5))
print(a[:100])
print(a[-1:-10:-1])
```

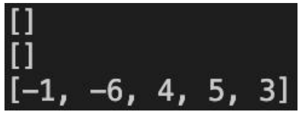


```
[0, 1, 2, 3, 4]
[4, 3, 2, 1, 0]
```

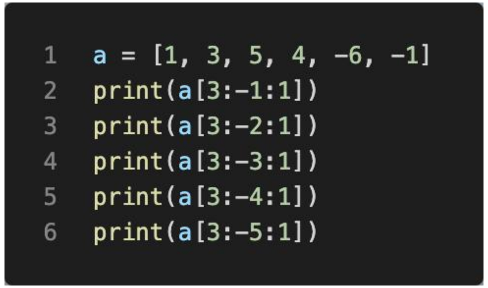
slice 不会越界



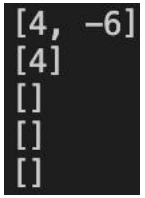
```
a = [1, 3, 5, 4, -6, -1]
print(a[1:1:-1])
print(a[-1:-1:-1])
print(a[-1:0:-1])
```



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



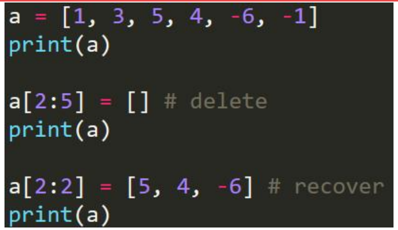
```
a = [1, 3, 5, 4, -6, -1]
print(a[3:-1:1])
print(a[3:-2:1])
print(a[3:-3:1])
print(a[3:-4:1])
print(a[3:-5:1])
```



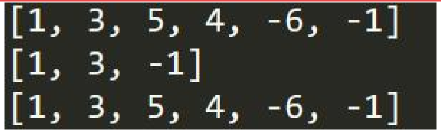
```
[4, -6]
[4]
[]
[]
[]
```

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

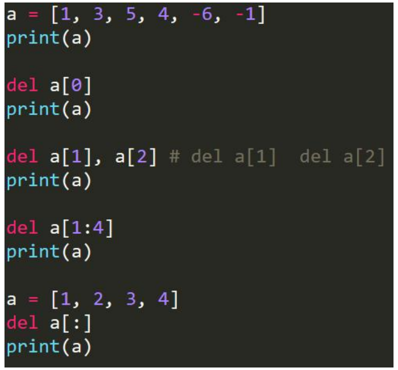


```python
a = [1, 3, 5, 4, -6, -1]
print(a)

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

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

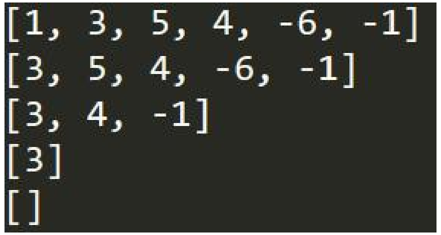


```
[1, 3, 5, 4, -6, -1]
[1, 3, -1]
[1, 3, 5, 4, -6, -1]
```

- 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)
```



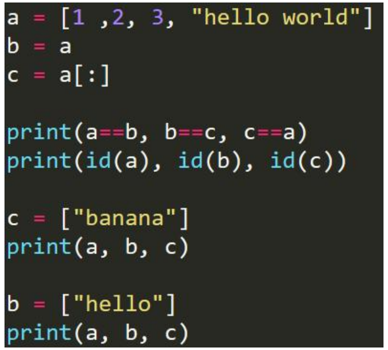
```
[1, 3, 5, 4, -6, -1]
[3, 5, 4, -6, -1]
[3, 4, -1]
[3]
[]
```

How about del a? ! =del a[:]

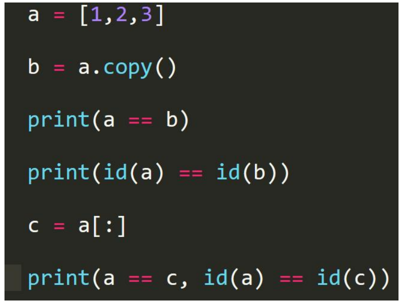
List: cloning

Stop

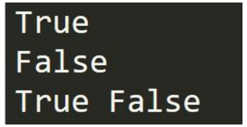
- If we want to modify a list and keep a copy of the original, we need to make a copy of the list itself. This process is sometimes called cloning (克隆)
- The easiest way to clone a list is to use the slice operator: newlist = list[:]
  or newlist = list.copy()



```
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)
```

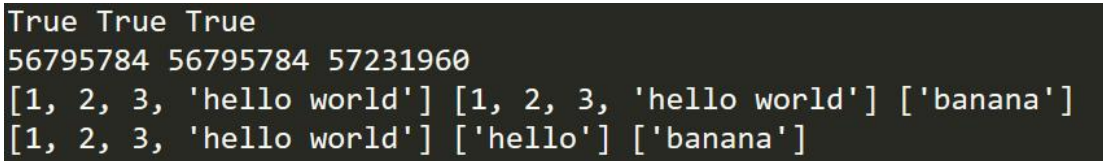


```
a = [1,2,3]

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？

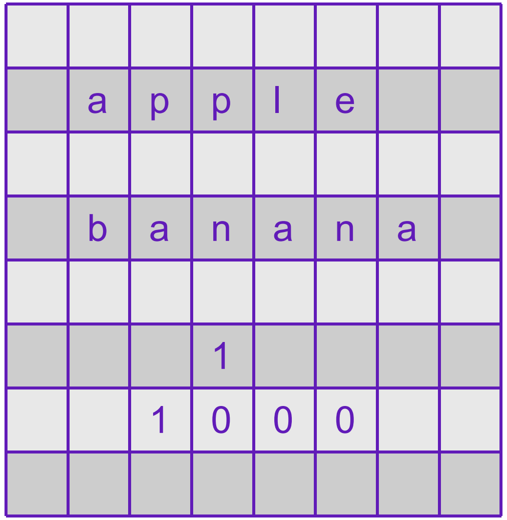


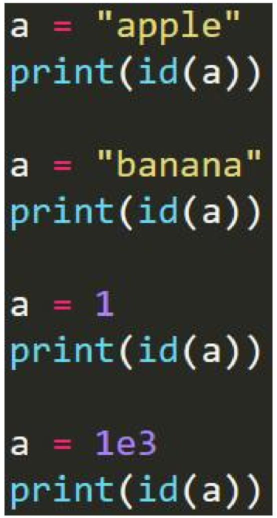
```
True True True
56795784 56795784 57231960
[1, 2, 3, 'hello world'] [1, 2, 3, 'hello world'] ['banana']
[1, 2, 3, 'hello world'] ['hello'] ['banana']
```

Data model

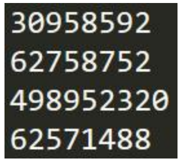
- Recall: Every variable points to a memory location in python. The location is referred to as id of the variable.
  - Every variable x in python has an id. Function id(x) returns the id of x.
  - id(x): 变量x所指向的数据的地址
  - When variable is reassigned, the location (id) will be different
  - https://docs.python.org/3/library/functions.html#id

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。

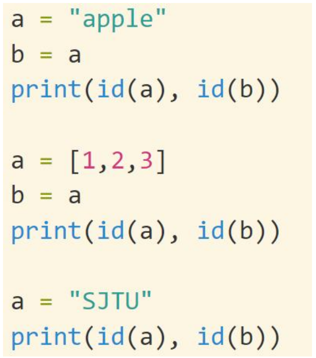




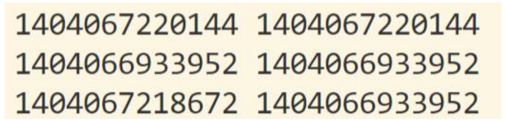
```
a = "apple"
print(id(a))

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

a = 1
print(id(a))

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



```
30958592
62758752
498952320
62571488
```

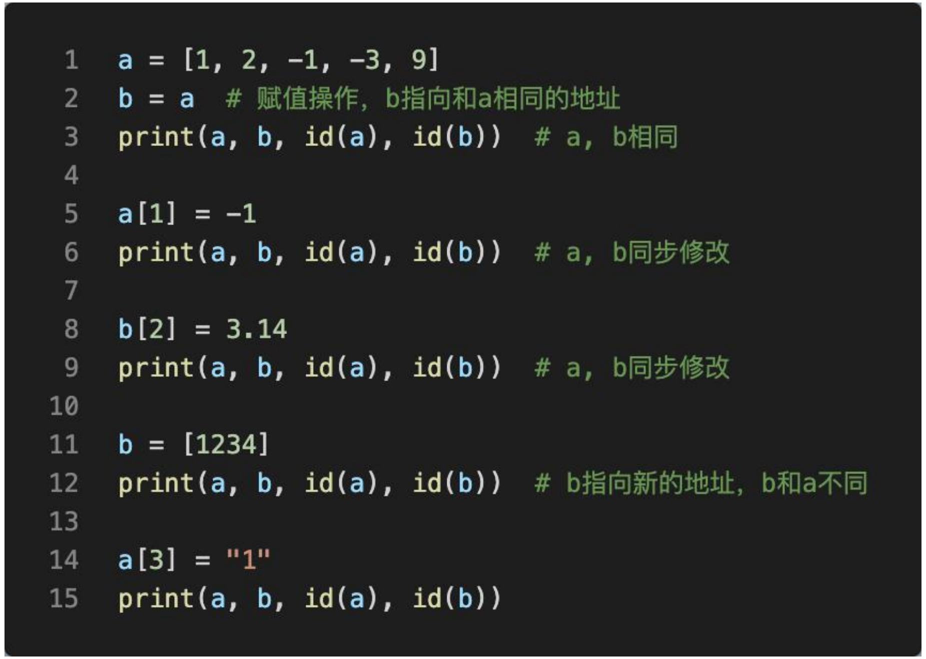


```
a = "apple"
b = a
print(id(a), id(b))

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

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

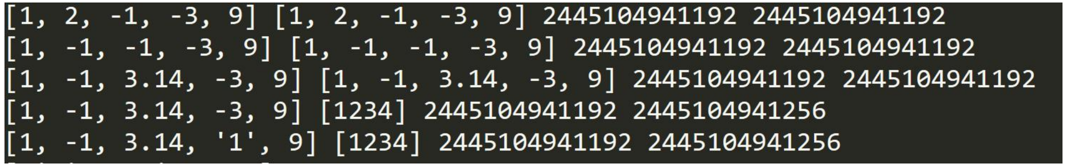


```
1404067220144 1404067220144
1404066933952 1404066933952
1404067218672 1404066933952
```

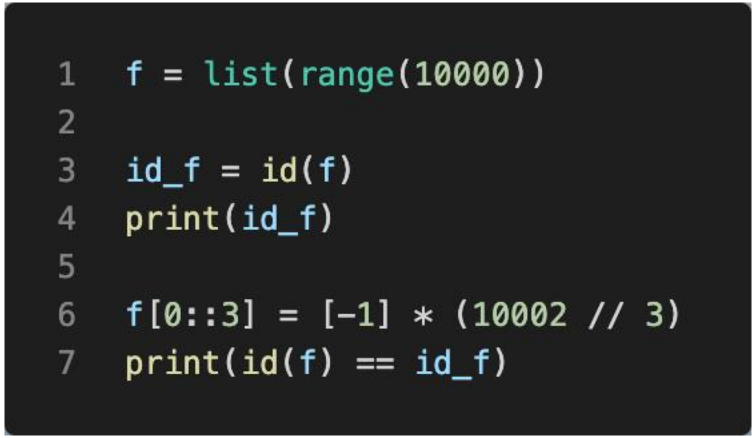
Reassignment Vs. Modification



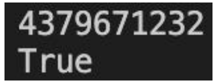
```python
a = [1, 2, -1, -3, 9]
b = a   # 赋值操作，b指向和a相同的地址
print(a, b, id(a), id(b))  # a, b相同

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

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



```
[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
```



```python
f = list(range(10000))

id_f = id(f)
print(id_f)

f[0::3] = [-1] * (10002 // 3)
print(id(f) == id_f)
```



```
4379671232
True
```

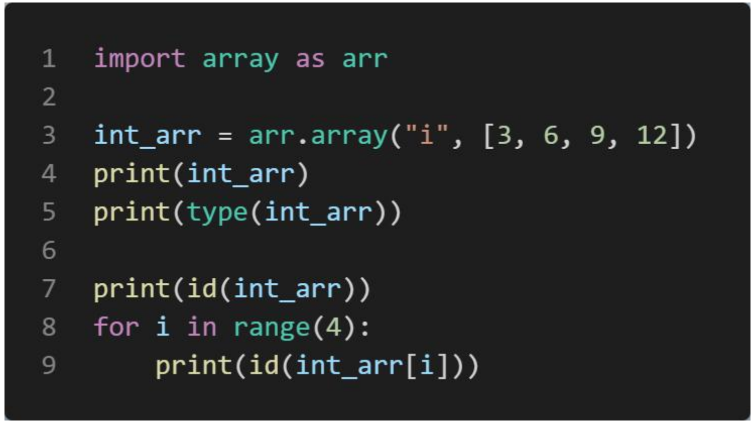
严格区分：修改还是重新赋值
不可修改类型只能重新赋值
x = 123
x = 321
x被重新赋值，并没有修改123

Magic behind List[i]

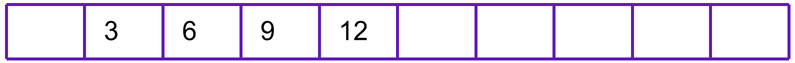
- Every variable x in Python points to location referred by its address, i.e., id(x)
  - The id of a variable is set by Python interpreter, and it is hardly possible for programmers to manage it themselves
  - In general, we need a variable to deal each data
- The ith element of a list is accessed by list[i]. All those elements are accessed via two variables, i.e., the name of the list and the index of the element
- How to store 108 integers in memory. As we know all of them are integer, we may allocate the same size of memory to store each of them. Assume each integer occupies a unit memory. We may further allocate 108 contiguous units of memory to store them all
- By this means, for a list a=[…], if we know the address of a[0] is a0, then the address of the a[i] is a0 + unit*i. That is, we can access a[i] by several arithmetic operations, which is very efficient
- The challenge for list is that the type of each element can be arbitrary. The solution above cannot be directly applied here. The reason is that, even if we store all those elements in the contiguous units, as the data may be of any type, we cannot get its address by a0 + unit*i
- The solution is, we don't store data directly in a list but store the addresses of the data in the list. The address are belonging to the same type, which can be stored by a series of contiguous units of memory

List Internal: array

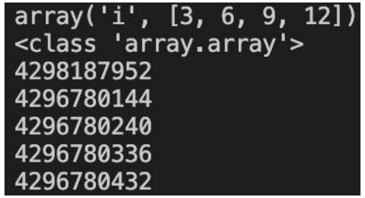
- An array is a vector containing homogeneous elements i.e., belonging to the same data type. Elements are allocated with contiguous (连续) memory locations. Typically, the size of an array is fixed
- Arrays of the array module are a thin wrapper over C arrays and are useful when you want to work with homogeneous data
- They are also more compact and take up less memory and space which makes them more size efficient compared to lists



```python
import array as arr

int_arr = arr.array("i", [3, 6, 9, 12])
print(int_arr)
print(type(int_arr))

print(id(int_arr))
for i in range(4):
    print(id(int_arr[i]))
```



前提, 数据类型相同，连续排列:
如果首id为 a0, 那么第i个元素的id为 a0+size*i



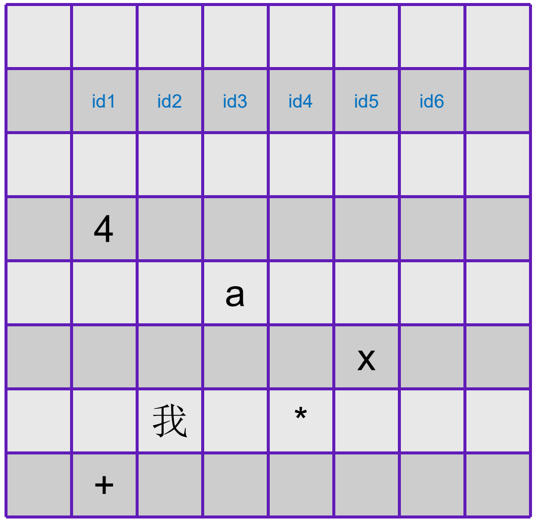
```
array('i', [3, 6, 9, 12])
<class 'array.array'>
4298187952
4296780144
4296780240
4296780336
4296780432
```

https://docs.python.org/3.8/library/array.html
https://www.geeksforgeeks.org/difference-between-list-and-array-in-python/

数组a的两个特性，保证了通过首地址a和下标i，可以计算出a[i]的地址，从而顺序访问

List Internal: array → list

- The ids (addresses, id()) of all the data in Python share the same format and type
- Albeit the types of elements in the list are arbitrary, their ids are of the same type
- List[i] stores the id of the ith element of the data. That's why we can access the ith element via its index



- Lst = [4, 'a', 'x', '我', '*', '+']
- id1, …, id6 are contiguous
- id1 → 4 , id2 → 'a', id3→ 'x', id4 → '我', id5 → '*', id6 → '+'
- $addr2 = addr1 + size$
- $addrk = addr1 + size * (k - 1),\ k = 1, \ldots, 6$
- 地址可能保存的是一个地址，如果反复迭代，就会形成地址1→地址2→地址3, …, →地址n的地址链
- 例如： [[[[[[1]]]]]]

列表：内部用array保存了每个地址。所有不同数据类型的地址的类型是一样的。从而间接达到了array的效果

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 ☺. Next page

```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)
```

基本原则

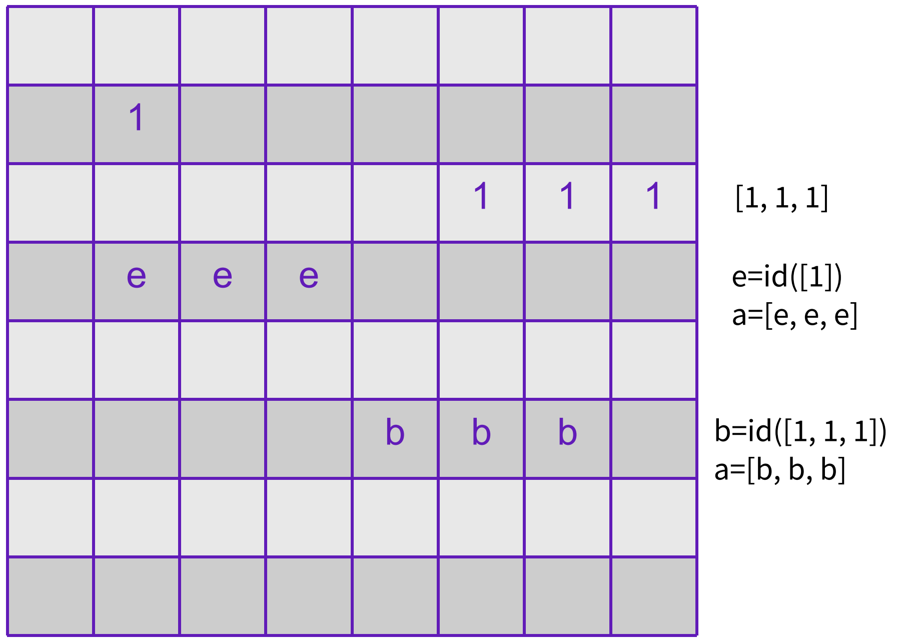
1. 是否存在(隐藏的)赋值、复制、copy、*n等操作
2. 判断数据类型是不是可修改的，不可修改的只能重新赋值
3. 可修改的类型，判断是直接修改还是重新赋值
4. 直接修改的，判断哪些会受到影响

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

注意，对于不可修类型，不相关变量也可能会指向同一地址，这样可以节约空间，同时不影响程序运行结果(不可修该)

```python
x = 1
y = 1
print(id(x), id(y))
```

```
140710880863016 140710880863016
```



```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

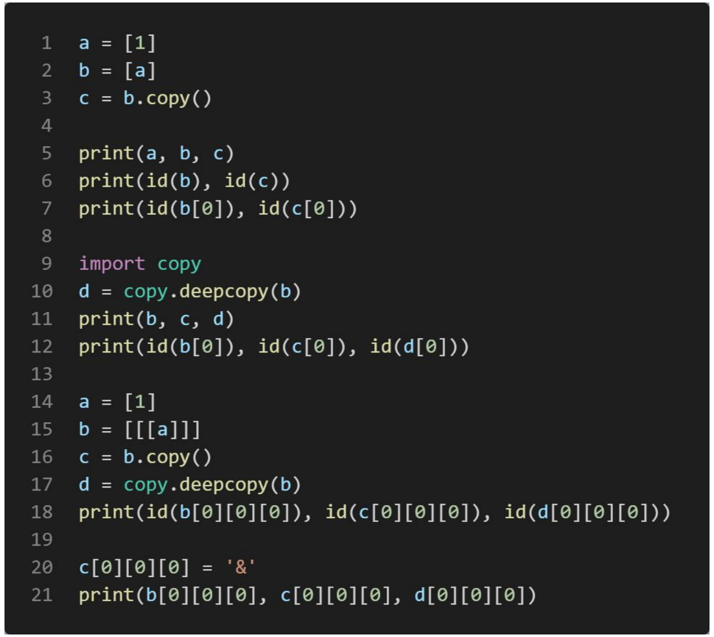
```
True True
True False
True True True
```

每个元素保存的是id

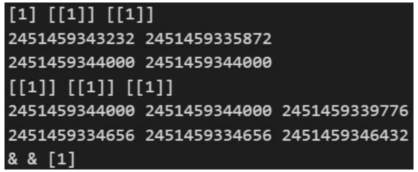
Nested list: deepcopy

Nested list非常非常非常容易出错：地址的地址的地址([[[[[[1]]]]]])⋯⋯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)



```python
1  a = [1]
2  b = [a]
3  c = b.copy()
4
5  print(a, b, c)
6  print(id(b), id(c))
7  print(id(b[0]), id(c[0]))
8
9  import copy
10 d = copy.deepcopy(b)
11 print(b, c, d)
12 print(id(b[0]), id(c[0]), id(d[0]))
13
14 a = [1]
15 b = [[[a]]]
16 c = b.copy()
17 d = copy.deepcopy(b)
18 print(id(b[0][0][0]), id(c[0][0][0]), id(d[0][0][0]))
19
20 c[0][0][0] = '&'
21 print(b[0][0][0], c[0][0][0], d[0][0][0])
```

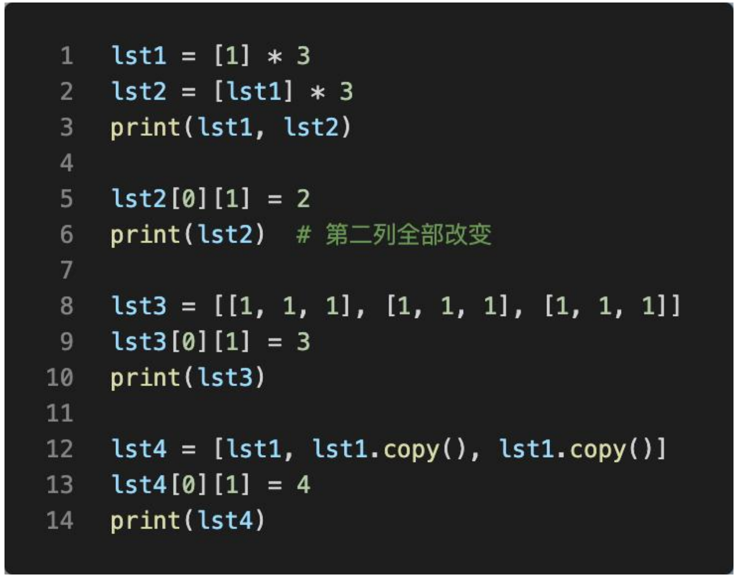


```
[1] [[1]] [[1]]
2451459343232 2451459335872
2451459344000 2451459344000
[[1]] [[1]] [[1]]
2451459344000 2451459344000 2451459339776
2451459334656 2451459334656 2451459346432
& & [1]
```

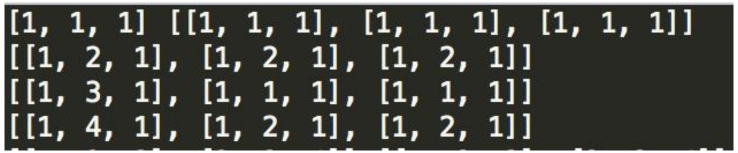
刘备和曹操的后代的后代可能是同一个人

1. Line 3-7, c是b的copy，所以b和c的地址不一样，但是c[0]的内容和b[0]一样，都是a的地址
2. Line 9-12， deepcopy复制的程度比copy更深（整个地址链），所以d没有影响
3. Line 14-21, 是deepcopy更复杂的例子，但本质上都是copy只复制一层，deepcopy复制整个地址链（按家谱复制刘备和曹操的后人）

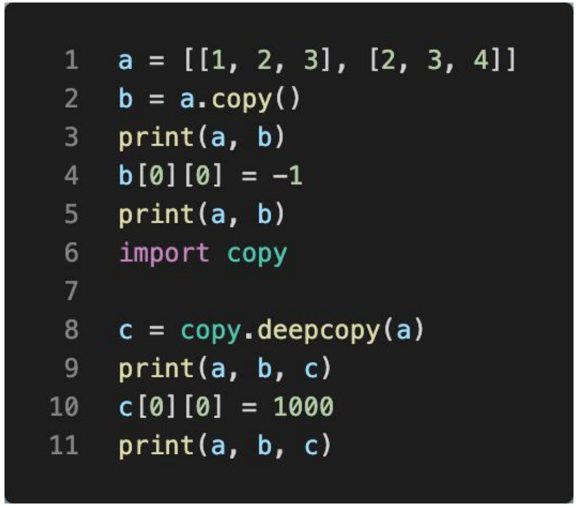
copy.deepcopy()



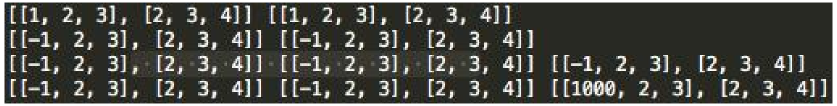
```python
lst1 = [1] * 3
lst2 = [lst1] * 3
print(lst1, lst2)

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]]
```



```python
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)
```

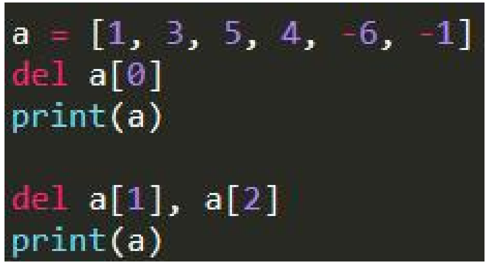


```
[[1, 2, 3], [2, 3, 4]] [[1, 2, 3], [2, 3, 4]]
[[-1, 2, 3], [2, 3, 4]] [[-1, 2, 3], [2, 3, 4]]
[[-1, 2, 3], [2, 3, 4]] [[-1, 2, 3], [2, 3, 4]] [[-1, 2, 3], [2, 3, 4]]
[[-1, 2, 3], [2, 3, 4]] [[-1, 2, 3], [2, 3, 4]] [[1000, 2, 3], [2, 3, 4]]
```

- 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、连根拔起)

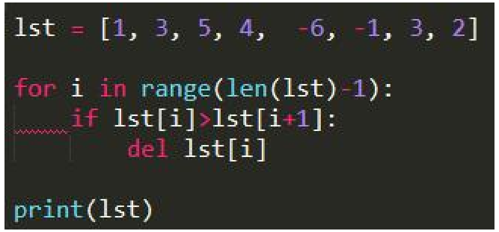
List: Traps of deletion

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

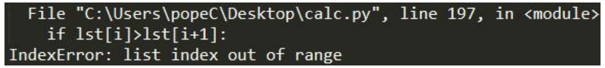


```python
a = [1, 3, 5, 4, -6, -1]
del a[0]
print(a)

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

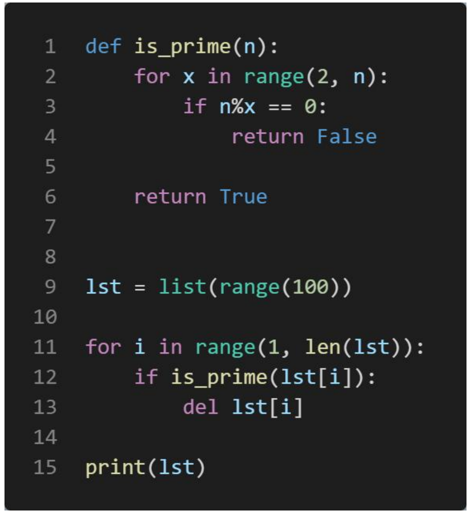
a[2]是原来的"a[3]"



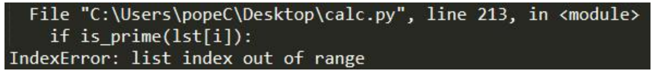
```python
lst = [1, 3, 5, 4,  -6, -1, 3, 2]

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

print(lst)
```



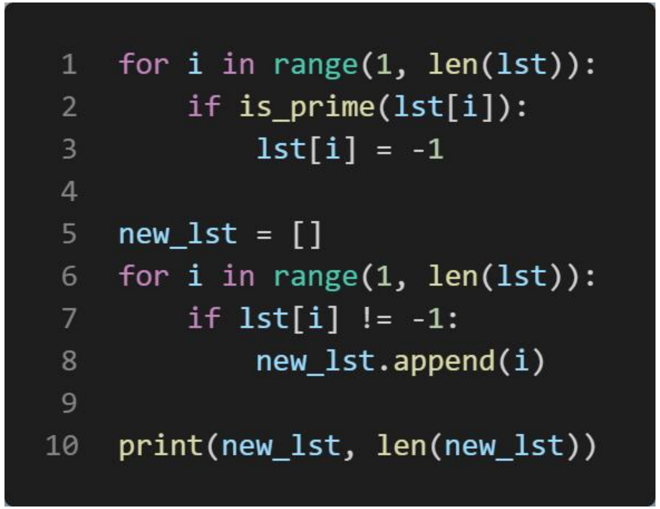
```
  File "C:\Users\popeC\Desktop\calc.py", line 197, in <module>
    if lst[i]>lst[i+1]:
IndexError: list index out of range
```



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

    return True


lst = list(range(100))

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

new_lst = []
for i in range(1, len(lst)):
    if lst[i] != -1:
        new_lst.append(i)

print(new_lst, len(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
3. 或者，用一个新的列表保留结果

List comprehension (列表推导)

- Grammar

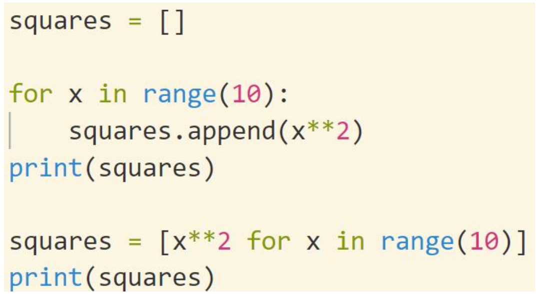
  单重循环 [expression for target in iterable if conditon]

  多重循环 [expression for target1 in iterable1 if condition1

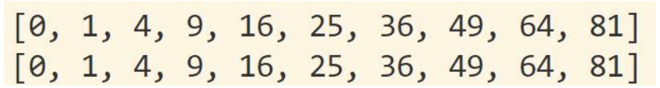
  for target2 in iterable2 if condition2 ..

  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



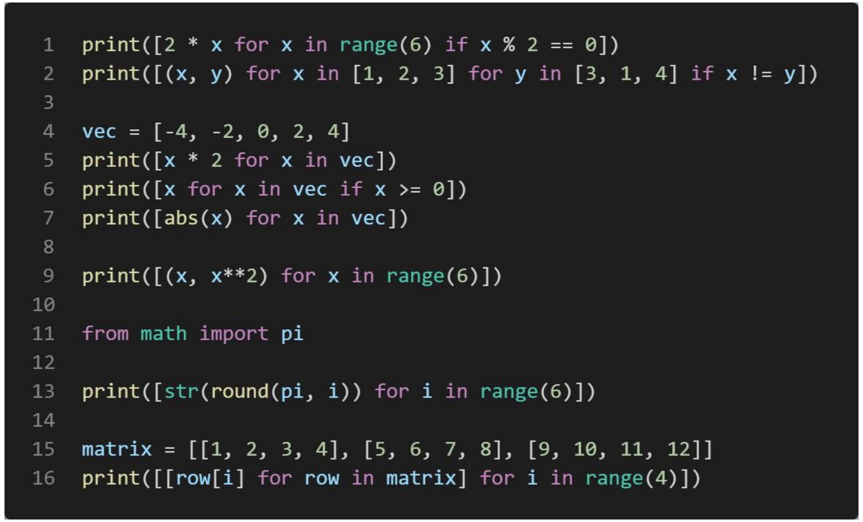
```
squares = []

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

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



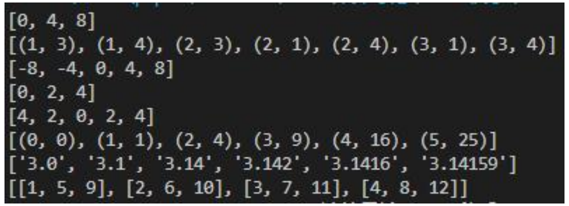
```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

推荐：Pythonic

Example

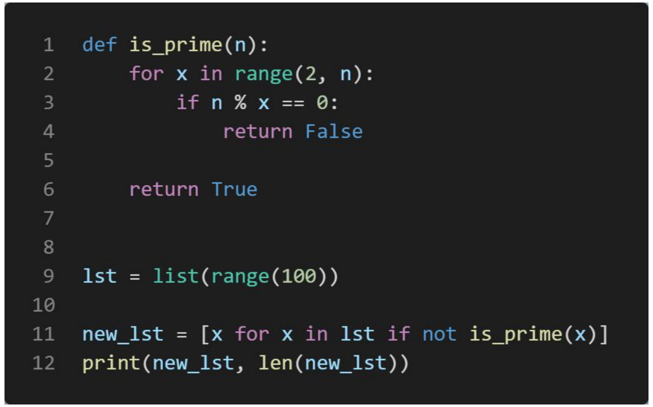


```python
 1  print([2 * x for x in range(6) if x % 2 == 0])
 2  print([(x, y) for x in [1, 2, 3] for y in [3, 1, 4] if x != y])
 3
 4  vec = [-4, -2, 0, 2, 4]
 5  print([x * 2 for x in vec])
 6  print([x for x in vec if x >= 0])
 7  print([abs(x) for x in vec])
 8
 9  print([(x, x**2) for x in range(6)])
10
11  from math import pi
12
13  print([str(round(pi, i)) for i in range(6)])
14
15  matrix = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]
16  print([[row[i] for row in matrix] for i in range(4)])
```

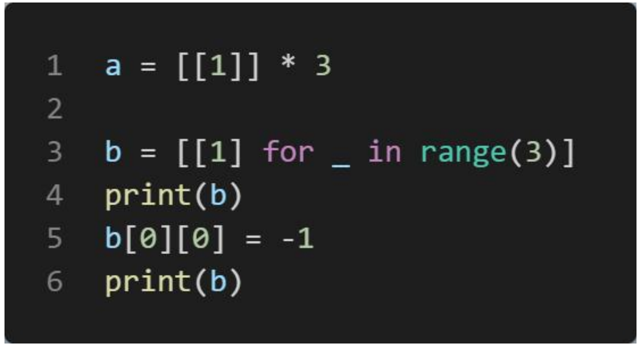


```
[0, 4, 8]
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
[-8, -4, 0, 4, 8]
[0, 2, 4]
[4, 2, 0, 2, 4]
[(0, 0), (1, 1), (2, 4), (3, 9), (4, 16), (5, 25)]
['3.0', '3.1', '3.14', '3.142', '3.1416', '3.14159']
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

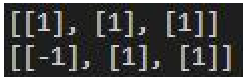
Line 15-16, 内层的列表是一个表达式，可以看成一个函数，这样就和Line 7一样理解



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

    return True


lst = list(range(100))

new_lst = [x for x in lst if not is_prime(x)]
print(new_lst, len(new_lst))
```



```python
a = [[1]] * 3

b = [[1] for _ in range(3)]
print(b)
b[0][0] = -1
print(b)
```



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

处理list更便捷、bug更少

Pythonic

List slice is fast

```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
0.0031674883365631103
Test by Loop
0.027055989503860473
True
```

大概相差一个数量级

如何创建新的list

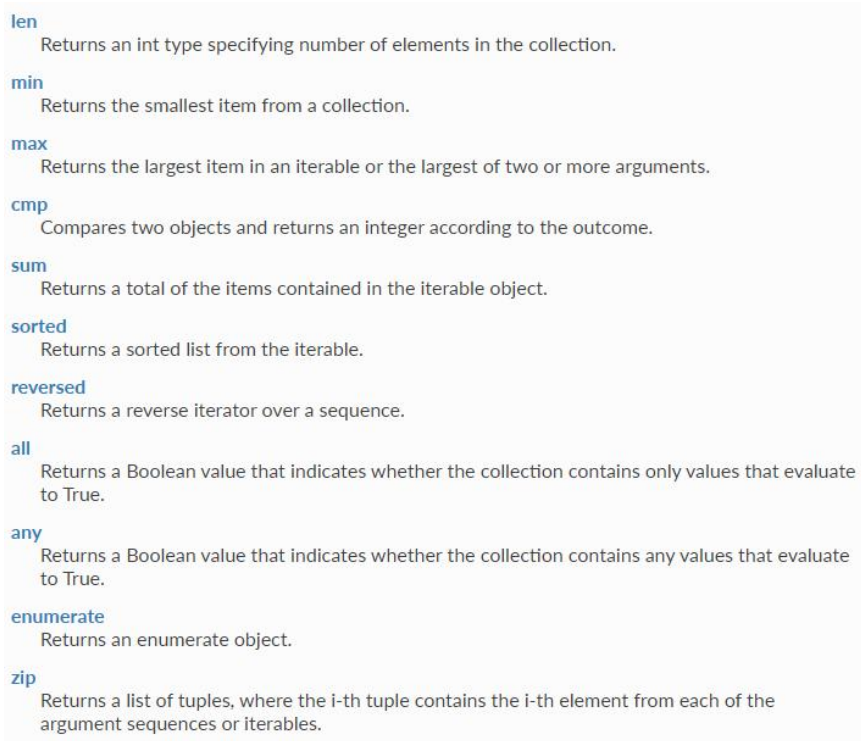
- for
- [0]*n
- list(range()), list(set), list(str)
- list comprehension

推荐List Comprehension

✓ 清晰，可读，正确

✓ 比for更快

Built-in Functions



len
: Returns an int type specifying number of elements in the collection.

min
: Returns the smallest item from a collection.

max
: Returns the largest item in an iterable or the largest of two or more arguments.

cmp
: Compares two objects and returns an integer according to the outcome.

sum
: Returns a total of the items contained in the iterable object.

sorted
: Returns a sorted list from the iterable.

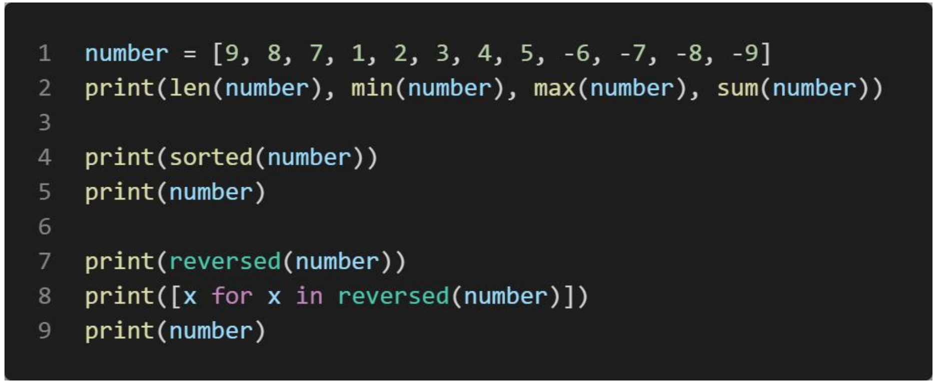
reversed
: Returns a reverse iterator over a sequence.

all
: Returns a Boolean value that indicates whether the collection contains only values that evaluate to True.

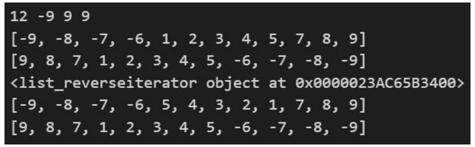
any
: Returns a Boolean value that indicates whether the collection contains any values that evaluate to True.

enumerate
: Returns an enumerate object.

zip
: Returns a list of tuples, where the i-th tuple contains the i-th element from each of the argument sequences or iterables.



```python
number = [9, 8, 7, 1, 2, 3, 4, 5, -6, -7, -8, -9]
print(len(number), min(number), max(number), sum(number))

print(sorted(number))
print(number)

print(reversed(number))
print([x for x in reversed(number)])
print(number)
```



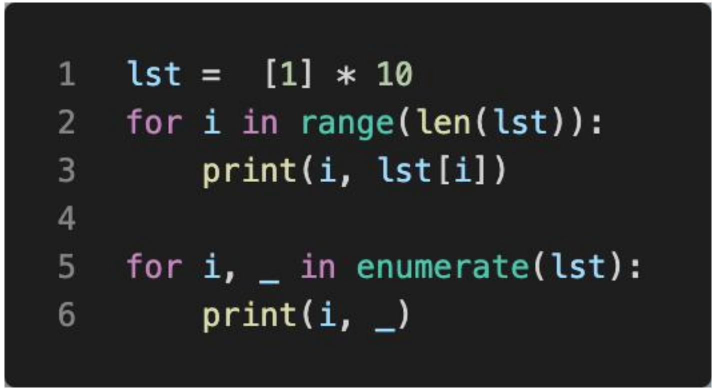
```
12 -9 9 9
[-9, -8, -7, -6, 1, 2, 3, 4, 5, 7, 8, 9]
[9, 8, 7, 1, 2, 3, 4, 5, -6, -7, -8, -9]
<list_reverseiterator object at 0x0000023AC65B3400>
[-9, -8, -7, -6, 5, 4, 3, 2, 1, 7, 8, 9]
[9, 8, 7, 1, 2, 3, 4, 5, -6, -7, -8, -9]
```

reversed()返回一个迭代器，需要用list()转换为list

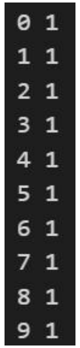
生产新的，不修改

enumerate 枚举

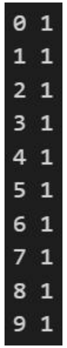
- for x in list无法获得list中元素的序号，要用for i in range(len(list))
- enumerate(list): Generating both offsets (元素的序号) and items
  - enumerate 取代 range(len(a))



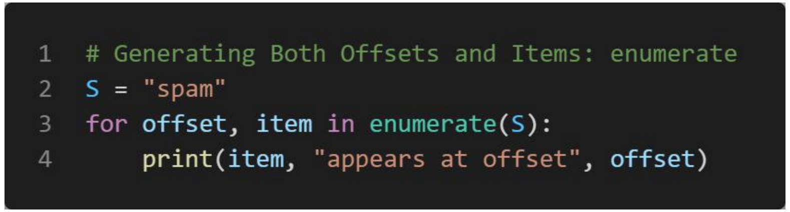
```python
lst =  [1] * 10
for i in range(len(lst)):
    print(i, lst[i])

for i, _ in enumerate(lst):
    print(i, _)
```



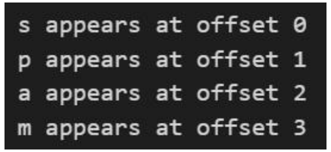
```
0 1
1 1
2 1
3 1
4 1
5 1
6 1
7 1
8 1
9 1
```



```
0 1
1 1
2 1
3 1
4 1
5 1
6 1
7 1
8 1
9 1
```



```python
# Generating Both Offsets and Items: enumerate
S = "spam"
for offset, item in enumerate(S):
    print(item, "appears at offset", offset)
```



```
s appears at offset 0
p appears at offset 1
a appears at offset 2
m appears at offset 3
```

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]]