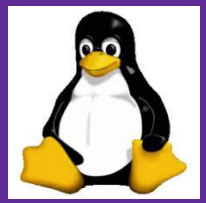
Introduction to Computation

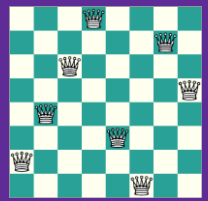
Autumn, 2023

Prof. Fan Cheng

Shanghai Jiao Tong University

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





7

Outline

- Tuple
- Dict
- String
- Set
- Summary

Tuple

Tuple (元组)

- Tuples are written with round brackets () (the round brackets can be omitted)

  tuple_a = (element1, elmement2, …,)
- To create a tuple with a single element, we need to add a “,” at the end

  tuple_a = (element1,)
- tuple() method will make a tuple (similar to list()) (tuple类型的构造函数)
- Tuple is ordered (顺序排列) and immutable (不可变).
  - len(), type(), [], in and not in, for
  - When you modify tuple in program, there will be a grammar error

```
a = (2.0, 9, 3)
a[0] = 9
```

```
Traceback (most recent call last):
  File "C:\Users\fcheng\OneDrive\CS124计算导论\2018\lecture notes\1.py", line 2, in <module>
    a[0] = 9
TypeError: 'tuple' object does not support item assignment
```

```
tup = (1,2,3)
tup_1 = 1,2,3
print(tup, tup_1)
```

```
(1, 2, 3) (1, 2, 3)
```

```
tup = ("hello world", 1896, 2018, 3.14, "Frog", [1,2,3], (3,4,5))
print(tup, type(tup), len(tup))
print(tup[4])
print(tup[-1])
print(tup[1:3])
print(tup[:3])
print(tup[5:2:-1])
```

```
('hello world', 1896, 2018, 3.14, 'Frog', [1, 2, 3], (3, 4, 5)) <class 'tuple'> 7
Frog
(3, 4, 5)
(1896, 2018)
('hello world', 1896, 2018)
([1, 2, 3], 'Frog', 3.14)
```

```
tup_1 = (1)
tup_2 = 1,
tup_3= (1,)
print(type(tup_1), type(tup_2), type(tup_3))
```

```
<class 'int'> <class 'tuple'> <class 'tuple'>
```

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

```
(1, 2, 3, 2, 1, 4, 1, 3, 4)
```

元组：不可修改的list

Immutable and Mutable

- Tuple is immutable. The elements of tuples could be mutable
- Reassignment is allowed
- del an element is not allowed, but del the whole tuple is allowed

```
1  tup = (1, 2, 3, 4)
2  print(tup)
3
4  tup = ([1], [1, 2], (1, 2, 3))  # reassignment
5  print(tup)
6
7  #tup[0] = 1 # error
8  #del tup[0] # error
9  del tup
```

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

```
5  x = [1, 2, 3]
6  a = (x)
7  b = (x,)
8  print(a, b)
```

```
[1, 2, 3] ([1, 2, 3],)
```

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

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

```
16  lst = [1, 2, 3]
17  st = (1, 2)
18
19  print(lst, id(lst))
20  print(st, id(st))
21
22
23  lst += [4]
24  st += (3, 4)
25
26  print(lst, id(lst))
27  print(st, id(st))
```

```
[1, 2, 3] 2742421490048
(1, 2) 2742421183680
[1, 2, 3, 4] 2742421490048
(1, 2, 3, 4) 2742421108544
```

list是直接修改，而tuple是创建一个新的

Tuple: packing and unpacking

- Packing: serval variables can be grouped into a tuple
  - x = a, b, c, d # x will be a tuple with four elements
- Unpacking: a tuple can be broken up into several elements
  - x1, x2, x3, x4 = x # Then x1 = a, x2=b, x3 =c, x4 =d
  - The left and right should have the same number of elements
- Unpacking and packing可以嵌套
  - ((a, b), c) = ((1, 2), 3)
- Simultaneous assignment is based on Tuple：

  x1, x2, x3, x4 = a, b, c, d

```
x, y, z, w = 1,"2", 3.0, [4j]
a = x, y, z, w
print(type(a))
a1, a2, a3, a4 = a
print(a1, a2, a3, a4)
```

```
<class 'tuple'>
1 2 3.0 [4j]
```

```
a = 1, 2, 3, 4
x, y, z = a
```

```
Traceback (most recent call last):
  File "C:\Users\fcheng\OneDrive\CS124计算导论\2018\lecture notes\1.py", line 8, in <module>
1 2 3.0 [4j]
    x, y, z = a
ValueError: too many values to unpack (expected 3)
```

元组：函数中同时return 多个值

Tuple: immutable list

- tuple_a.count(x): Returns the number of times a specified value occurs in a tuple
- tuple_a.index(x): Searches the tuple for a specified value and returns the position of where it was found
  - When x is not in tuple_a, an error will arise
- [], [::], +, *

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

print(a.count(1))
print(a.count(2))
print(a.count(4))

print(a.index(3))
print(a.index(4))
```

```
(1, 2, 3, 2, 1, 4, 1, 3, 4)
3
2
2
2
5
```

```
print(a.index(-1))
```

```
ValueError: tuple.index(x): x not in tuple
```

```
a = (2.0, 9, 3)
print(a)
b = a *3
a = a + (4,)
print(a)
print(b)
```

```
(2.0, 9, 3)
(2.0, 9, 3, 4)
(2.0, 9, 3, 2.0, 9, 3, 2.0, 9, 3)
```

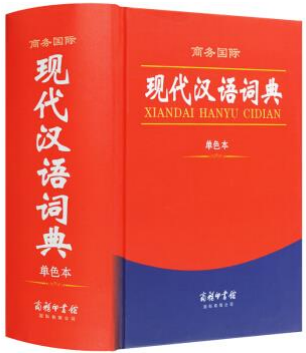
元组：不可修改的list

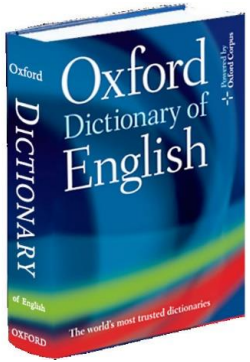
Quick test

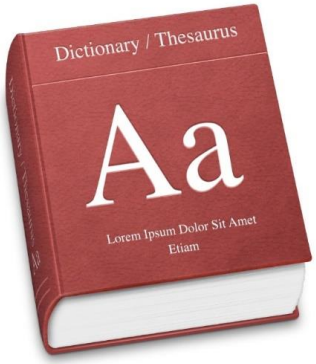
- Given a tuple a = (1,2,3)
  - What is the result of a[0] = -1?
  - What is print(a[2:1:-1])
  - x, y, z = a, what are x, y and z?
- a=1与a=(1,)的差异

Dict

Dictionary







- List search: Search from the first item to the last one
- Dictionary search: We use keywords to search the dictionary, not a list search
- In Python Dict, we use keys to index values; it is unordered, not an ordered list
  - Dict: Words and their interpretations
  - Word -- Key(关键字)
  - Interpretation – Value(值)

Dict is more efficient than list

Dict: Creation

- The elements of dict should be a pair like: **key: value. “林则徐”: “民族英雄”**
- All the pairs are enclosed by **{}**

```
a = {} # empty dict
print(type(a))
a = {"hello" : 1} # dict with 1 element
print(type(a))
b = {"test": "SJTU", "print": 1} # dict with 2 elements
print(type(b))
```

```
<class 'dict'>
<class 'dict'>
<class 'dict'>
```

- The key can be any immutable type. 关键字是唯一的. List type (mutable) cannot be the key

```
a = {"test":1, 1.2:3, 3:"hello", 5:[1,2,3]}
print(a, type(a))
```

```
{'test': 1, 1.2: 3, 3: 'hello', 5: [1, 2, 3]} <class 'dict'>
```

```
a = {[1,2,3]:4} #TypeError: unhashable type: 'list'
```

- Values can be any type. Nested dict.

```
1  nested_dt = {"Ronaldo":{"Age":18, "No.":"7"},
2               "Messi":{"Age":16, "No.":"10"}}
3  print(nested_dt['Messi']['No.'])
```

```
10
```

- len(dict) returns the length of dict

```
a = {"test":1, 1.2:3, 3:"hello", 5:[1,2,3]}
print(len(a))
```

```
4
```

dict()

- dict类的构造函数
- class dict(**kwarg)
- class dict(mapping, **kwarg)
- class dict(iterable, **kwarg) (dt = dict({}))

```python
dt = {"Tsinghua":1911, "PKU":1898, "SJTU":1896}

dt1 = dict(Tsinghua=1911, PKU=1898, SJTU=1896)

dt2 = dict((("Tsinghua",1911), ("PKU",1898), ("SJTU",1896)))

dt3 = dict(zip(("Tsinghua", "PKU", "SJTU"), (1911, 1898, 1896)))

print(dt)
print(dt1)
print(dt2)
print(dt3)

print(dt == dt1, dt1 == dt2, dt2 == dt3)
```

```
{'Tsinghua': 1911, 'PKU': 1898, 'SJTU': 1896}
{'Tsinghua': 1911, 'PKU': 1898, 'SJTU': 1896}
{'Tsinghua': 1911, 'PKU': 1898, 'SJTU': 1896}
{'Tsinghua': 1911, 'PKU': 1898, 'SJTU': 1896}
True True True
```

Dict: []

- Values of a dictionary can be accessed via dict[key]. If the key is not existed in the dict, an error will arise
- Suppose we have a dict as follows:
  - chn = {"Hello":"你吼啊", "World":"世界", "SJTU":"上海交大", "PI":3.14, "Second":"秒"}
  - All its keys are: “Hello”, “World”, “SJTU”, “PI”, “Second”
  - An error will arise if you use other values to access the chn

```
chn = {"Hello":"你吼啊", "World":"世界", "SJTU":"上海交大", "PI":3.14, "Second":"秒"}
print(chn["Hello"])
print(chn["SJTU"])
print(chn["PI"])
```

```
你吼啊
上海交大
3.14
```

```
print(chn[0]) # KeyError: 0
print(chn[1]) # KeyError: 1
```

- Unless the index is a keyword, dict cannot be accessed by an index (KeyError)

```
1  chn = {1:2, 2:3, 4:5}
2  print(chn[1], chn[2], chn[4])
3  print(chn[3]) # error, 3 is not a kw
```

```
2 3 5
```

```
Traceback (most recent call last):
  File "c:\Users\popeC\OneDrive\CS124计算导论\2023 秋季\course_code.py", line 18, in <module>
    print(chn[3]) # error, 3 is not a kw
KeyError: 3
```

In principle, there is no order on the keys, unless you specify it. For example: a = {"test":1, 1.2:3, 3:"hello", 5:[1,2,3]}

get() and setdefault()

- If we try to access a value of the key that doesn't exist in the dictionary, Python will return a KeyError.
- To get around this problem, we can use the get() method that will return the value if its key is in the dictionary, or it will return some default value that we set:
- setdefault(). The setdefault() method is often confused with the get() method. They perform more or less the same task. Indeed, if we suspect that there is a non-existing key in the dictionary, we can use this method to return a default value. However, in contrast to get(), this method inserts the default value of this key in the dictionary

```
1  dt = {1:2, 2:3, 3:4}
2  print(dt[4]) #error
```

```
Traceback (most recent call last):
  File "c:\Users\popeC\OneDrive\CS124计算导论\2023 秋季\course_code.py", line 20, in <module>
    print(dt[4])
KeyError: 4
```

```
1  dt = {1:2, 2:3, 3:4}
2  print(dt.get(4, -1))
3  print(dt[4])
```

```
-1
```

```
1  dt = {1:2, 2:3, 3:4}
2  print(dt.setdefault(4, -1))
3  print(dt[4])
```

```
-1
-1
```

Dict: Removing Items

- delete dict[key]: The del keyword removes the item with the specified key name. The del keyword can also delete the dictionary completely
- Dict.clear(): make an empty dict
- dict.pop(): The pop() method removes the item with the specified key name
- dict.popitem(): The popitem() method removes the last inserted item (in versions before 3.7, a random item is removed instead ☺

```
chn = {"Hello":"你吼啊", "World":"世界", "SJTU":"上海交大", "PI":3.14, "Second":"秒"}

chn.pop("Hello")
print(chn)

chn.popitem()
print(chn)

del chn["PI"]
print(chn)

chn.clear()
print(chn)
```

```
{'World': '世界', 'SJTU': '上海交大', 'PI': 3.14, 'Second': '秒'}
{'World': '世界', 'SJTU': '上海交大', 'PI': 3.14}
{'World': '世界', 'SJTU': '上海交大'}
{}
```

```
chn = {"Hello":"你吼啊", "World":"世界", "SJTU":"上海交大", "PI":3.14, "Second":"秒"}
print(chn)

del chn
print(chn)
```

```
{'Hello': '你吼啊', 'World': '世界', 'SJTU': '上海交大', 'PI': 3.14, 'Second': '秒'}
Traceback (most recent call last):
  File "c:/Users/popeC/OneDrive/CS124计算导论/2020 秋季/lecture notes/course_code.py", line 348, in <module>
    print(chn)
NameError: name 'chn' is not defined
```

Multi-Line Statements

- Explicit line continuation: \, use the line continuation character (\) to split a statement into multiple lines
- Implicit line continuation: Implicit line continuation is used when you split a statement using either parenthesis ( ), brackets [ ], and braces { }

```
msg = "A fox is \
in the hole.\n\
The old man was \
dreaming about \
the lions."

print(msg)

total = 1 + \
    2 - \
    3 / \
    3
print(total)
```

```
A fox is in the hole.
The old man was dreaming about the lions
2.0
```

\后面不能有空格(转义字符)

```
top3 = ("Shanghai "
     "Jiao "
     "Tong "
     "University.")
print(top3)


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

total = (1 +
       2 -
       3 /
       3)
print(total)
```

```
Shanghai Jiao Tong University.
[5, 4, 3, 2, 1]
2.0
```

```
chn = {"Hello": "你吼啊",
       "World": "世界",
       "SJTU": "上海交大",
       "PI": 3.14,
       "Second": "秒"}

chn.pop("Hello")
print(chn)

chn.popitem()
print(chn)

del chn["PI"]
print(chn)

chn.clear()
print(chn)
```

Dict: modification, copy and merge

- The values can be reassigned/updated/added via its key

```
a = {"1":1, "2":2}
a["2"] = -1 # update
a["3"] = 3.14 #add a new one
print(a)
```

```
{'1': 1, '2': -1}
```

```
a = {} # a is empty now
```

- The dictionary in python is **mutable**. You need to make a copy in case of modification: **dict.copy()**
  - It is also possible to use the dict() constructor to make a new dictionary

```
a = {"test":1, 1.2:3, 3:"hello", 5:[1,2,3]}
b = a
c = a.copy()
print(id(a), id(b), id(c))
print(a==b, b==c, c==a)
```

```
49958320 49958320 51525456
True True True
```

b is the alias of a; c is a copy of a

```
chn = {"Hello":"你吼啊", "World":"世界", "SJTU":"上海交大", "PI":3.14, "Second":"秒"}
chn1 = dict(chn)
print(chn1 == chn)
print(id(chn1) == id(chn))
```

```
True
False
```

Merge: **, |, update()

- Merge two dictionaries x and y, z = {**x, **y}

```
a = {"test":1, 1.2:3}
b = {3:"hello", 5:[1,2,3]}
c = {**a, **b}
print(c)
```

```
{'test': 1, 1.2: 3, 3: 'hello', 5: [1, 2, 3]}
```

```
a = {"test":1, 1.2:3}
b = {2:1, 1.2:-1}
c = {** a, ** b}
print(c)
```

```
{'test': 1, 1.2: -1, 2: 1}
```

- dta.update(dtb): return None, update dta in place
- dta | dtb

```
a = {"test":1, 1.2:3}
b = {2:1, 1.2:-1}
a.update(b) #return None
print(a, b)

a = {"test":1, 1.2:3}
b = {2:1, 1.2:-1}
c = a|b
print(c)
```

```
{'test': 1, 1.2: -1, 2: 1} {2: 1, 1.2: -1}
{'test': 1, 1.2: -1, 2: 1}
```

Keys(), values(), items()

- For a dictionary dict, there are several methods to get its useful information:
  - dict.keys(), returns all its keys
  - dict.values(), returns all its values
  - dict.items(), returns all its items (key:value pairs)

```
a = {"test":1, 1.2:3, 3:"hello", 5:[1,2,3]}
print(a.keys(), type(a.keys()), list(a.keys()))
print(a.values(), type(a.values()), list(a.keys()))
print(a.items(), type(a.items()), list(a.items()))
```

```
dict_keys(['test', 1.2, 3, 5]) <class 'dict_keys'> ['test', 1.2, 3, 5]
dict_values([1, 3, 'hello', [1, 2, 3]]) <class 'dict_values'> ['test', 1.2, 3, 5]
dict_items([('test', 1), (1.2, 3), (3, 'hello'), (5, [1, 2, 3])]) <class 'dict_items'>
[('test', 1), (1.2, 3), (3, 'hello'), (5, [1, 2, 3])]
```

dict.keys(), dict.values(), and dict.items() will not directly return a list. You should transform them yourself.

- in and not in are used to test the membership

```
print("test" in a)
print("🐕" not in a)
print("哈哈" in a)
```

```
True
True
False
```

Loop Through a Dictionary

- Print all key names in the dictionary, one by one:
  - for x in dict: print(x)
- Print all values in the dictionary, one by one:
  - for x in dict: print(dict[x])
- You can also use the values() method to return values of a dictionary:
  - for x in thisdict.values(): print(x)
- Loop through both keys and values, by using the items() method:
  - for x in thisdict.items(): print(x,y)

```
chn = {"Hello":"你吼啊", "World":"世界", "SJTU":"上海交大", "PI":3.14, "Second":"秒"}
for x in chn:
    print(x)

for x in chn:
    print(chn[x])

for x in chn.values():
    print(x)

for x, y in chn.items():
    print(x, y)
```

```
Hello
World
SJTU
PI
Second
你吼啊
世界
上海交大
3.14
秒
你吼啊
世界
上海交大
3.14
秒
Hello 你吼啊
World 世界
SJTU 上海交大
PI 3.14
Second 秒
```

Dictionary Methods

- Python has a set of built-in methods that you can use on dictionaries.

Method	Description
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and value
get()	Returns the value of the specified key
items()	Returns all the tuples for each key value pair
keys()	Returns all the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns all the values in the dictionary

Dict应用：统计数量

- Dict的结构中key:value, 其中value可用来保存有用的信息，譬如数量
- 给定一篇文章，统计每个单词出现的数量
  https://www.abrahamlincolnonline.org/lincoln/speeches/gettysburg.htm（葛底斯堡演说）

speech = '''
Four score and seven years ago our fathers brought forth on this continent, a new nation, conceived in Liberty, and dedicated to the proposition that all men are created equal.
…
'''

```
#print(speech)
lst = speech.split()

dt = {}

for x in lst:
    if x in dt:
        dt[x] += 1
    else:
        dt[x] = 1

ndt = sorted(dt.items(), key=lambda kv: kv[1], reverse = True)
print(ndt)
```

dict查找、插入、删除比list快很多

```
[('that', 13), ('the', 9), ('to', 8), ('we', 8), ('a', 7), ('--', 7), ('and', 6), ('can', 5), ('of', 5), ('have', 5), ('for', 5), ('here', 5), ('not', 5), ('in', 4), ('this', 3), ('nation,', 3), ('dedicated', 3), ('are', 3), ('great', 3), ('so', 3), ('who', 3), ('It', 3), ('is', 3), ('they', 3), ('us', 3), ('shall', 3), ('people,', 3), ('our', 2), ('on', 2), ('new', 2), ('conceived', 2), ('or', 2), ('nation', 2), ('long', 2), ('We', 2), ('dedicate', 2), ('gave', 2), ('The', 2), ('here,', 2), ('far', 2), ('what', 2), ('be', 2), ('which', 2), ('from', 2), ('these', 2), ('dead', 2), ('devotion', 2), ('Four', 1), ('score', 1), ('seven', 1), ('years', 1), ('ago', 1), ('fathers', 1), ('brought', 1), ('forth', 1), ('continent,', 1), ('Liberty,', 1), ('proposition', 1), ('all', 1), ('men', 1), ('created', 1), ('equal.', 1), ('Now', 1), ('engaged', 1), ('civil', 1), ('war,', 1), ('testing', 1), ('whether', 1), ('any', 1), ('dedicated,', 1), ('endure.', 1), ('met', 1), ('battle-field', 1), ('war.', 1), ('come', 1), ('portion', 1), ('field,', 1), ('as', 1), ('final', 1), ('resting', 1), ('place', 1), ('those', 1), ('their', 1), ('lives', 1), ('might', 1), ('live.', 1), ('altogether', 1), ('fitting', 1), ('proper', 1), ('should', 1), ('do', 1), ('this.', 1), ('But,', 1), ('larger', 1), ('sense,', 1), ('consecrate', 1), ('hallow', 1), ('ground.', 1), ('brave', 1), ('men,', 1), ('living', 1), ('dead,', 1), ('struggled', 1), ('consecrated', 1), ('it,', 1), ('above', 1), ('poor', 1), ('power', 1), ('add', 1), ('detract.', 1), ('world', 1), ('will', 1), ('little', 1), ('note,', 1), ('nor', 1), ('remember', 1), ('say', 1), ('but', 1), ('it', 1), ('never', 1), ('forget', 1), ('did', 1), ('here.', 1), ('living,', 1), ('rather,', 1), ('unfinished', 1), ('work', 1), ('fought', 1), ('thus', 1), ('nobly', 1), ('advanced.', 1), ('rather', 1), ('task', 1), ('remaining', 1), ('before', 1), ('honored', 1), ('take', 1), ('increased', 1), ('cause', 1), ('last', 1), ('full', 1), ('measure', 1), ('highly', 1), ('resolve', 1), ('died', 1), ('vain', 1), ('under', 1), ('God,', 1), ('birth', 1), ('freedom', 1), ('government', 1), ('by', 1), ('perish', 1), ('earth.', 1), ('Abraham', 1), ('Lincoln', 1), ('November', 1), ('19,', 1), ('1863', 1)]
```

Dict: more examples

- Replace conditional expression
- Graph theory: weighted graph

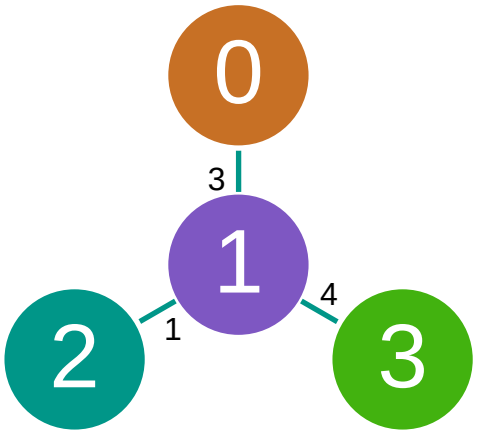
```python
def add(x, y):
    return x + y


def sub(x, y):
    return x - y


tool = {"add": add, "sub": sub}
print(tool["add"](1, 2), tool["sub"](3, 3))
```

```
3 0
```

比if-elif-else更清晰, 配合λ表达式

```python
weight_graph = {(0, 1): 3, (1, 2): 1, (1, 3): 4}
print(weight_graph[(1, 2)])
```

```
1
```



String

String: Review of Lecture 3

- String can be defined by “…”, ’…’, and “““…”””
- str()
- Escape character: “\n”, “\t”
- input(): always returns a string
- String is an immutable list. You cannot change a string. Yet you can reassign or create it
  - []: String can be accessed by str[i]
  - str[start: stop: step]: The rule for a slice of the string are the same with list and tuple
  - len(str), returns the length of a string
  - +: str1 + str2, returns the concatenation of two strings
  - *: str*n, repeats the string n times (What if $n \leq 0$?)
  - Loop: for x in str1: do_sth()
- Alphabetical order : >, <, >=, <=, !=, ==
- String format
  - C-style
  - “{}. ..{}”.format(x, y)
  - F-string: f“{x}…{y}”

string is an immutable list!

split(): split a string

- str.split(): Split a string into a list where each word is a list item:

  “I am from SJTU.” → [“I”, “am”, “from”, “SJTU”]

- Parameter:

  separator: Optional. Specifies the separator to use when splitting the string. By default any whitespace is a separator

```
str1 = "123 13.4 Helloworld"
a = str1.split()
print(a)

str1 = "I am from SJTU."
a = str1.split()
print(a)
```

```
['123', '13.4', 'Helloworld']
['I', 'am', 'from', 'SJTU.']
```

默认分隔符：空格

```
str1 = "123*456*3.14*SJTU-IEEE"
a = str1.split('*')
print(a)

str1 = "hello, my name is Peter, I am 26 years old"
a = str1.split(", ")
print(a)
```

```
['123', '456', '3.14', 'SJTU-IEEE']
['hello', 'my name is Peter', 'I am 26 years old']
```

自定义分隔符

```
str1 = input("Please enter the lengths of edges of a triangle: \n")
a = str1.split()
x, y, z = float(a[0]), float(a[1]), float(a[2])
if x>0 and y>0 and z>0 and x+y>z and y+z>x and z+x>y:
    print("It is a triangle.")
else:
    print("It is not a triangle")
```

```
Please enter the lengths of edges of a triangle:
123 1 1
It is not a triangle
```

实际应用

join(): concatenate strings

- str.join(): The join function is the inverse of split(). It takes a list of strings and concatenates the elements with a space between each pair

```
str1 = "a string, with stuff"
print(str1.split())
str2 =  "苟利国家生死以,  岂因祸福避趋之。"
print(str2.split())
```

```
['a', 'string,', 'with', 'stuff']
['苟利国家生死以,', '岂因祸福避趋之。']
```

List!

```
li1 = ['a', "string", "with", "stuff"]
print("".join(li1))
li2 =  ["苟利国家生死以, ", "岂因祸福避趋之。"]
print("".join(li2))
```

```
astringwithstuff
苟利国家生死以, 岂因祸福避趋之。
```

- str.join(): could also specify the separator letter

```
myTuple = ("John", "Peter", "Vicky")
x = "#".join(myTuple)
print(x)

myTuple = ("John", "Peter", "Vicky")
x = ":))".join(myTuple)
print(x)
```

```
John#Peter#Vicky
John:))Peter:))Vicky
```

```
myDict = {"name": "John", "country": "Norway"}
mySeparator = "TEST"

x = mySeparator.join(myDict)
print(x)
```

```
nameTESTcountry
```

str: methods

- Python string module provides several built-in functions to process strings
  - https://docs.python.org/3/library/stdtypes.html
- str.find(sub, start, end): Return the lowest index in the string where substring sub is found within the slice s[start:end]. Optional arguments start and end are interpreted as in slice notation. Return -1 if sub is not found.

```
fruit = 'banana'
print(fruit.find('a'))
print(fruit.find('na'))
print(fruit.find('na', 3))
print(fruit.find('na', 1, 3))
```

```
1
2
4
-1
```

- str.count(sub[, start[, end]]): Return the number of non-overlapping occurrences of substring sub in the range [start, end]. Optional arguments start and end are interpreted as in slice notation.

```
str = "exciting!"
print(str.count("i"))
print(str.count("z"))
```

```
2
0
```

str: more

- str.isdigit(): Return true if all characters in the string are digits and there is at least one character, false otherwise
- str.upper(), str.lower(): Return a copy of the string with all the characters converted to upper/lower case
- str.strip() method removes any whitespace from the beginning or the end
  - str.rstrip(), str.lstrip(): Return a copy of the string with tailing/trailing characters removed
- str.replace(old, new[, count]): Return a copy of the string with all occurrences of substring old replaced by new
- String can be processed directly on the content without defining a variable

```
three = '3'
print(three.isdigit())

str1 = str.upper("Exciting!")
print(str1)

test = "newlines  \t \n\n\n"
print(test.rstrip())

s = "a string, with stuff"
print(s.replace("stuff", "characters"))
print(s.replace("s", "X", 1))
```

```
True
EXCITING!
newlines
a string, with characters
a Xtring, with stuff
```

```
1  name = '  SJTU is top3  '
2  print(name.lstrip(), name.rstrip(), name.strip(), name)
```

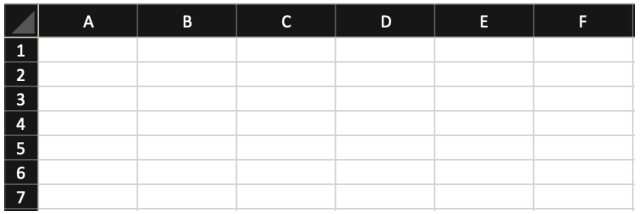
```
SJTU is top3     SJTU is top3 SJTU is top3   SJTU is top3
```

```
print("Hello world".lower())
print("I am from Shanghai".split())
```

```
hello world
['I', 'am', 'from', 'Shanghai']
```

\t and expandtabs()

- \t: tab. How many blanks in print('\t')? It is subtle
- \t in excel: Jump from one cell to the next
- In default, each row of the python console is divided into cells with length equals to 8 spaces
  - 8, 16, 32, …, 8n, …
- \t will move the console to the beginning the next cell
- str.expandtabs(n): set the current tab size to be n



微软office办公文档中，对于每个单元格，有左对齐、右对齐、居中

```
unit = "12345678"
print(unit)
print('\t-')
print('1\t-')
print('12\t-')
print(unit*2)
print('\t1\t-')
```

```
12345678
        -
1       -
12      -
1234567812345678
        1       -
```

```
print("1\t1".expandtabs(0))
print("1\t1".expandtabs(1))
print("1\t1".expandtabs(2))
print("1\t1".expandtabs(3))
print("1\t1".expandtabs(4))
```

```
11
1 1
1 1
1  1
1   1
```

Format print()

f-string中的格式化控制符号与str.format()一样，为了演示方便，这里采用了str.format()的方式来编码，推荐优先使用f-string

- {:format}：Alignment(对齐方式: <, =, >), width(宽度), 小数位
  - 每个单元格，有宽度。所谓宽度(width)为10，就是输出的时候，用当前光标位置后面10个位置来输出数据。有三种情况：
    - 数据长度大于10。默认输出
    - 小于10。左对齐和右对齐（默认右对齐）
    - 等于10。 正常输出即可
  - 对于浮点数，要进一步指明小数点后面的位数。10.3f就是宽度为10，后面保留3位, f表示浮点数

```
print('{:>10}'.format('SJTU'))
print('{:<10}'.format('IEEE'))
print('{:10.3f}'.format(3.1415926535897 93))
```

```
      SJTU
IEEE
     3.142
```

- 宽度为10，右对齐 >
- 宽度为10， 左对齐 <
- 宽度为10，保留三位小数

```
print("{:>5} {:>5} {:>5} {:>5} {:>5}".format(1, 2, 3, 4, 5))
print("{:>5} {:>5} {:>5} {:>5} {:>5}".format(6, 7, 8, 9, 10))
print("{:>5} {:>5} {:>5} {:>5} {:>5}".format(11, 12, 13, 14, 15))
print("{:>5} {:>5} {:>5} {:>5} {:>5}".format(16, 17, 18, 19, 20))
```

```
    1     2     3     4     5
    6     7     8     9    10
   11    12    13    14    15
   16    17    18    19    20
```

宽度为5，>右对齐

str.format()

“I am {} from {}”.format(name, city)

- {}: placeholder 占位符
- format(): formats the specified value(s) and insert them inside the string's placeholder. The format() method returns the formatted string.
- You may use index or variable to denote the placeholders

```
txt1 = "My name is {fname}, I'am {age}".format(fname = "John", age = 36)
txt2 = "My name is {0}, I'am {1}".format("John",36)
txt3 = "My name is {}, I'am {}".format("John",36)
print(txt1)
print(txt2)
print(txt3)
```

```
My name is John, I'am 36
My name is John, I'am 36
My name is John, I'am 36
```

```
txt1 = "My name is {name1}. He is  {name2} and she is {name3}".format( name1= "John", name2 = "Bill", name3 = "Lily")
txt2 = "My name is {name1}. He is  {name1} and she is {name1}".format( name1= "John", name2 = "Bill", name3 = "Lily")
txt3 = "My name is {0}. He is  {1} and she is {2}".format( "John", "Bill", "Lily")
txt4 = "My name is {2}. He is  {1} and she is {2}".format( "John", "Bill", "Lily")
print(txt1)
print(txt2)
print(txt3)
print(txt4)
```

```
My name is John. He is  Bill and she is Lily
My name is John. He is  John and she is John
My name is John. He is  Bill and she is Lily
My name is Lily. He is  Bill and she is Lily
```

#Accessing arguments by position or name

str.format(): b, o, x

The bin() method converts and returns the binary equivalent string of a given integer

```
str1 = "0101"
x1 = int(str1)
x2 = int(str1, base=2)
print(x1, x2)

str1 = "{:b}".format(x1)
str2 = "{:b}".format(x2)
print(str1, str2)
```

```
101 5
1100101 101
```

```
str1 = "{:o}".format(x1)
str2 = "{:o}".format(x2)
print(str1, str2)

str1 = "{:x}".format(x1)
str2 = "{:x}".format(x2)
print(str1, str2)
```

```
145 5
65 5
```

```
x = 1234
xbs = bin(x)
print(xbs, type(xbs))
```

```
0b10011010010 <class 'str'>
```

d: 10进制(默认)
b, o, x: 二、八、十六进制

str.format(): f, d, c, s

f： 浮点数　d：10进制整数
c： 字符　　s： 字符串

```
print("The lengths are {:f}, {:f}, {:f}.".format(10.1, 8.5, 6.9))
print("The lengths are {:.0f}, {:.0f}, {:.0f}.".format(10.1, 8.5, 6.9))
print("The lengths are {:.3f}, {:.3f}, {:.3f}.".format(10.1, 8.5, 6.9))
print("The lengths are {:.9f}, {:.9f}, {:.9f}.".format(10.1, 8.5, 6.9))

print("The lengths are {:10.3f}, {:10.3f}, {:10.3f}.".format(10.1, 8.5, 6.9))
print("The lengths are {:>10.3f}, {:<10.3f}, {:^10.3f}.".format(10.1, 8.5, 6.9))
```

默认6位小数

```
The lengths are 10.100000, 8.500000, 6.900000.
The lengths are 10, 8, 7.
The lengths are 10.100, 8.500, 6.900.
The lengths are 10.100000000, 8.500000000, 6.900000000.
The lengths are     10.100,      8.500,      6.900.
The lengths are     10.100, 8.500     ,   6.900   .
```

```
print("The length is {:+d}".format(12))
print("The length is {:+d}".format(-12))
print("The length is {:-d}".format(12))
print("The length is {:-d}".format(-12))
print("The length is {: d}".format(12))
print("The length is {: d}".format(-12))
print("The length is {:=5d}".format(12))
print("The length is {:=5d}".format(-12))
```

```
The length is +12
The length is -12
The length is 12
The length is -12
The length is  12
The length is -12
The length is    12
The length is -  12
```

```
print("The length is {:12s}".format("Hello"))
print("The ASCII/unicode character is {:c}".format(65))
```

```
The length is Hello
The ASCII/unicode character is A
```

https://www.w3schools.com/python/ref_string_format.asp
https://docs.python.org/3.8/library/string.html

str.format(): format types

:<	Left aligns the result (within the available space)
:>	Right aligns the result (within the available space)
:^	Center aligns the result (within the available space)
:=	Places the sign to the left most position
:+	Use a plus sign to indicate if the result is positive or negative
:-	Use a minus sign for negative values only
:	Use a space to insert an extra space before positive numbers (and a minus sign befor negative numbers)
:,	Use a comma as a thousand separator
:_	Use a underscore as a thousand separator
:b	Binary format
:c	Converts the value into the corresponding unicode character
:d	Decimal format

str.format(): format types

:e	Scientific format, with a lower case e
:E	Scientific format, with an upper case E
:f	Fix point number format
:F	Fix point number format, in uppercase format (show inf and nan as INF and NAN)
:g	General format
:G	General format (using a upper case E for scientific notations)
:o	Octal format
:x	Hex format, lower case
:X	Hex format, upper case
:n	Number format
:%	Percentage format

f-String: Formatted string literals

- Python version ≥ 3.6
- Formatted string literals (also called f-strings for short) let you include the value of Python expressions inside a string by prefixing the string with f or F and writing expressions as {expression}.

```
name = "Eric"
age = 74
txt = f"Hello, {name}. You are {age}."
num = f"{2 * 37}"
print(txt)
print(num)
```

```
Hello, Eric. You are 74.
74
```

```
exp1 = f"{70 + 4}"
exp2 = f"{{70 + 4}}"
exp3 = f"{{{70 + 4}}}"
exp4 = f"{{{{70 + 4}}}}"

print(exp1, exp2, exp3, exp4)
```

```
74 {70 + 4} {74} {{70 + 4}}
```

```
import math
print(f'The value of pi is approximately {math.pi:.3f}.')
```

```
The value of pi is approximately 3.142.
```

- f-strings support = for self-documenting expressions and debugging (Py 3.8)

```
1  from datetime import date
2
3
4  user = 'eric_idle'
5  member_since = date(1975, 7, 31)
6  print(f'{user=} {member_since=}')
```

```
user='eric_idle' member_since=datetime.date(1975, 7, 31)
```

String module (Not recommended)

- Python provides string module, where many useful constants and methods are included
  - https://docs.python.org/3/library/string.html
  - To use this module, import string

```
import string
print(string.ascii_letters)
print(string.punctuation)
print(string.ascii_lowercase)
```

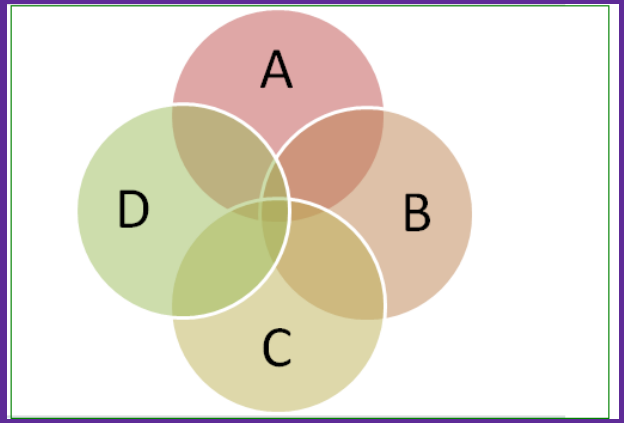
```
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
abcdefghijklmnopqrstuvwxyz
```

- string.asci_letters includes all the letters
- string.punctuation includes all the punctuation
- string.asci_lowercase includes all the lowercase letters

string.ascii_letters
string.ascii_lowercase
string.ascii_uppercase
string.digits
string.hexdigits
string.octdigits
string.punctuation
string.printable
string.whitespace

formatter — Generic output formatting: Deprecated since version 3.4: Due to lack of usage, the formatter module has been deprecated.

Set



Set: create

- A set is an unordered collection with no duplicate elements.
  - Basic uses include membership testing and eliminating duplicate entries（重复）.
- Set objects also support mathematical operations like union, intersection, difference, and symmetric difference.
- Curly braces {} or the set() function can be used to create sets.

Note: to create an empty set you have to use set(), not {}; the latter creates an empty dictionary

fruit = {'apple', 'orange', 'pear', 'banana'}

```
my_set = {1, 2, 3}
print(my_set)

my_set = {1.0, "Hello", (1, 2, 3)}
print(my_set)

my_set = {} # it is a dict.
print(type(my_set))

my_set = set()
print(type(my_set))
```

```
{1, 2, 3}
{1.0, (1, 2, 3), 'Hello'}
<class 'dict'>
<class 'set'>
```

{}: dict

https://docs.python.org/3/tutorial/datastructures.html#sets

set()

- The parameter of set() can be type of str, list, dict
  - List, tuple: all the elements
  - Dict: all the keywords
  - String: all the characters
- You cannot access items in a set by referring to an index, since sets are unordered the items has no index
- To add one item to a set use the add() method
- To add more than one item to a set use the update() method
- To remove an item in a set, use the remove(), or the discard() method
- The clear() method empties the set
- All the above methods can be implemented by basic set operations

return None: set.add(), set.update()

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

a = set("123")
print(a)

a = set((1,2,3))
print(a)

a = set({1:2, 2:3, 3:4})
print(a)
```

```
set([1, 2, 3])
set(['1', '3', '2'])
set([1, 2, 3])
set([1, 2, 3])
[Finished in 0.0s]
```

```
# a = set(123) #error
# a = set(1,2,3) #error
```

```
a = set(range(10))
print(a)

a.add(100) # single element
a.remove(6) # single element

a.update([10,12]) # add multiple element
print(a)

a.clear()
print(a)
```

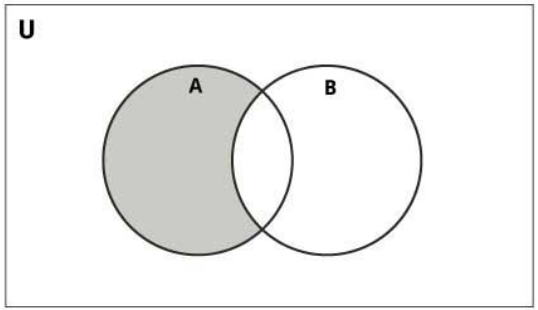
```
set([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
set([0, 1, 2, 3, 4, 5, 7, 8, 9, 10, 12, 100])
set([])
```

```
4798    x = [[2]]
4799    set(x)
```

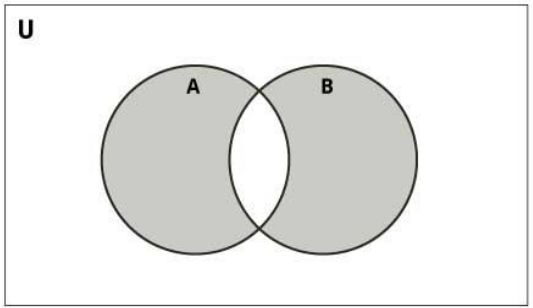
TypeError: unhashable type: 'list'

Sets operations

- A|B: Returns a set which is the union of sets A and B
- A& B: Returns a set which is the intersection of sets A and B
- A-B: Returns the set difference of A and B (the elements included in A, but not included in B)
- A^B: Returns the symmetric difference of sets A and B (the elements belonging to either A or B, but not to both sets simultaneously)



A-B



A^B

- A<=B: Returns true if A is a subset of B
- A>=B: Returns true if B is a subset of A
- Similar, A<B, A>B

```python
basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
print(basket,type(basket))

print("apple" in basket)
print("Frog" in basket)
print("pear" not in basket)

a = set("Hello world")
b = set("Shanghai Jiao Tong University")
print(a) # unique letters in a
print(b)

print(a-b) # set minus
print(a|b) # set union
print(a&b) # set intersection
print(a^b) # elements in only one set
```

```
{'apple', 'pear', 'banana', 'orange'} <class 'set'>
True
False
False
{'H', ' ', 'o', 'e', 'd', 'l', 'r', 'w'}
{'i', 'a', 'r', ' ', 'o', 'e', 'y', 'g', 'T', 'U', 'h', 's', 't', 'J', 'n', 'S', 'v'}
{'H', 'd', 'w', 'l'}
{'i', 'o', 'y', 'T', 'l', 't', 'U', 'n', 'v', 'H', 'a', ' ', 'e', 'S', 'd', 'g', 'h', 's', 'J', 'r', 'w'}
{'r', ' ', 'o', 'e'}
{'i', 'y', 'T', 't', 'l', 'U', 'n', 'v', 'H', 'a', 'S', 'd', 'g', 'h', 's', 'J', 'w'}
```

```python
A = {1, 2, 3, 4, 5, 6, 7}
B = {1, 3, 5, 7, 0 , -1, -3, -5}
print(A<=B)
print(A==B)
```

```
False
False
```

Code talks!

Summary

List, tuple, str, dict, set

- Mutable: list, dict, set
- Immutable: tuple, str
- Ordered: list, tuple, str
- Unordered: dict, set
- Elements of dicts and sets should be immutable. Element of tuples could be immutable
- List vs tuple: if we don't need to alter the elements, tuple is preferred; e.g., return values
- List, dict, set. List is ordered while dict and set are not. List[::] is allowed but not legal for dict and set
- Insert, alter, delete operation are much faster in set and dict than list
- dict vs set: set is a simplified version of dict. If set is enough for the problem, then we should keep it as simple as possible. Dict and used to represent a collection with duplicates while set cannot
- Methods of data structures: modified in place or create a new one
    - List.sort()

KISS: Keep It Stupid Simple

Loop在编程中的意义

- 遍历一个集合的元素
- 根据循环结束时各个变量的状态，判断现在程序的状态，执行下一阶段的命令（合理利用程序运行中各种信息）
  - 利用循环结束时x的状态，进行下一步操作

```
lst = [1, 3, 4, 5, 8, 10, 11, 14, 17]
for x in lst:
    if x % 3 == 2:
        break
    if x % 4 == 3:
        break

if x % 3 == 1:
    pass

if x % 4 == 3:
    pass
```

for x in collection_a: 状态不变

- 当for开始运行时，x的遍历序列就定义好了
  - 改变x不会改变collection_a
  - 改变x不会改变x的遍历序列

```
a = [1,2,3,4,5,6,7]
print(range(len(a)))
for i in range(len(a)):
    print("i={}, range(len(a))={}".format(i,  range(len(a))))
    if i<len(a) and a[i] % 2== 0:
        a.append(a[i])

    print(range(len(a)))

print(range(len(a)))
```

```
range(0, 7)
i=0, range(len(a))=range(0, 7)
range(0, 7)
i=1, range(len(a))=range(0, 7)
range(0, 8)
i=2, range(len(a))=range(0, 8)
range(0, 8)
i=3, range(len(a))=range(0, 8)
range(0, 9)
i=4, range(len(a))=range(0, 9)
range(0, 9)
i=5, range(len(a))=range(0, 9)
range(0, 10)
i=6, range(len(a))=range(0, 10)
range(0, 10)
range(0, 10)
```

i=0, 1, …, 6不变

```python
L = [1,2,3,4,5]
for x in L:
    x += 1
print(L)

for i in range(len(L)):
    L[i] += 1
print(L)

lst = [1,-2,3,-4,5,-6,7,-8]

for x in lst:
    print(x)
    x = x**2
    print(x)
```

```
[1, 2, 3, 4, 5]
[2, 3, 4, 5, 6]
1
1
-2
4
3
9
-4
16
5
25
-6
36
7
49
-8
64
```

for的陷阱: 应对

- Code that modifies a collection while iterating over that same collection can be tricky to get right.
  - for内部的循环机理实现需要定义iter()（高级议题，后面讲到）
- 尽量不要一边修改集合collection_a, 一边进行for x in collection_a操作
- 在for x in xxx: do_sth()
  - do_sth()对x的修改并不改变x的遍历顺序for和range结合的时候也有这种现象
- 解决思路：
  - 用while 手工操作
  - 用集合collection_a.copy()来处理
  - 用for x in reversed(lst):
  - 列表推导

https://docs.python.org/3/tutorial/controlflow.html#for-statements
https://sopython.com/canon/95/removing-items-from-a-list-while-iterating-over-the-list/

```python
lst = [1, 2, 3, 4, 5, 6, 7, 8]
for x in lst:
    if x < 6:
        lst.remove(x)
print(lst)

lst = [1, 2, 3, 4, 5, 6, 7, 8]
print([x for x in lst if x >= 6])

lst = [1, 2, 3, 4, 5, 6, 7, 8]
for x in reversed(lst):
    if x < 6:
        lst.remove(x)
print(lst)
```

```
[2, 4, 6, 7, 8]
[6, 7, 8]
[6, 7, 8]
```

血泪教训：for 和remove尽量不要一起用

特别注意：不要一边修改一个数据结构，一边遍历它，否则遍历的顺序可能会很古怪

+=, /=, *=

- In python, when you write an expr like a = a + b, you may just write it as a += b
  - +=, -=, *=, /=, //=
- However, you should pay attention to the situations when a is mutable: list, dict
- 不可被修改
  - 基本类型： int, float, complex
  - 数据结构: tuple, string
- 可被修改
  - List, dict, set
  - 自定义复合类型

```
a = 1
b = 12.3
c = "Hell world"
d = [1, 2, 3, 4]

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

a += 1 # a = a + 1
b += 1
c += "!"
d += [5]

print(a, b, c, d)
print(id(a), id(b), id(c), id(d))
# d的地址没有变, += 直接操作
```

```
dd = [1,2,3,4]
print(id(dd))
dd = dd +[5]
print(id(dd))
# dd的地址会改变
```

特别注意区别
创造新变量， 改造旧变量

Mutability of parameters

If a mutable parameter is passed into a function, it may be altered inside the function

```
def f(inta, listb):
    print("The paramters are: ", inta, listb)
    inta -= 3
    listb += [100]
    print("The paramters after modification are: ", inta, listb)

a = 2008
b = [1,3,7]
print("a = {}, b = {}".format(a, b))
f(a, b)
print("a = {}, b = {}".format(a, b))
```

```
a = 2008, b = [1, 3, 7]
The paramters are:  2008 [1, 3, 7]
The paramters after modification are:  2005 [1, 3, 7, 100]
a = 2008, b = [1, 3, 7, 100]
```

a: 基本int类型，不可修改。
b: 是list，可以被修改。
参数传递：
inta = a
listb = b
修改listb就修改了b

从程序设计的角度而言，你认为参数应该被修改吗？