Table of Contents

1 序列类型

1.1 定义

1.1.1 `list` 列表

`list` 是可变序列类型

[1]:
```
s = [1, 2, 3]
type(s)
```

[1]:
```
list
```

1.1.2 `tuple` 元组

`tuple` 是不可变序列

[2]:
```
s = (1, 2, 3)
type(s)
```

[2]:
```
tuple
```

1.2 通用序列操作

1.2.1 `x in s`：如果 `s` 中的某项等于 `x`，则结果为 `True`，否则为 `False`

[3]:
```
s = [1, 2, 3, 4, 'a', 'b']
```

[4]:
```
[1, 2] in s
```

[4]:
```
False
```

[5]:
```
'a' in s
```

[5]: True

1.2.2 s + t：s 与 t 拼接

[6]:
```
s = [1, 2, 3]
t = ['a', 'b']
s + t
```

[6]: [1, 2, 3, 'a', 'b']

1.2.3 s * n：s 与自身进行 n 次拼接

[7]:
```
s = [1, 'ab']
s * 3
```

[7]: [1, 'ab', 1, 'ab', 1, 'ab']

1.2.4 s[i]：s 的第 i 项

[8]:
```
s = [1, 'a', 2, 'b']
```

正数第 i 项（从 0 开始）

[9]:
```
s[0]
```

[9]: 1

[10]:
```
s[1]
```

[10]: 'a'

倒数第 i 项（从-1 开始）

[11]:
```
s[-1]
```

[11]: 'b'

[12]:
```
s[-2]
```

[12]: 2

1.2.5 切片

[13]:
```
s = [1, 'a', 2, 'b', 3, 'c', 4567]
```

s[i:j]：s 从 i 到 j 的切片。i 表示开始位置，j 表示结束位置的前一个位置。

[14]:
```
# 将 s[1] - s[3] 进行切片
s2 = s[1:5]
s2
```

[14]: ['a', 2, 'b', 3]

省略 j 表示到末尾

[15]:
```
s3 = s[1:]
s3
```

[15]: ['a', 2, 'b', 3, 'c', 4567]

s[i:j:k]：s 从 i 到 j，以步长为 k 进行切片。i 表示开始位置，j 表示结束位置的前一个位置，k 表示步长。

[16]:
```
# 将 s[1], s[1+2] 进行切片（s[1+2+2]=s[5] 不能要）
s4 = s[1:5:2]
s4
```

[16]: ['a', 'b']

省略 j 表示到末尾

[17]:
```
# 将 s[1], s[1+2], s[1+2+2] 进行切片
s5 = s[1::2]
s5
```

[17]: ['a', 'b', 'c']

1.2.6 len(s)：s 的长度

[18]:
```
s = []
len(s)
```

[18]: 0

[19]:
```
s = [[]]
len(s)
```

[19]: 1

[20]:
```
s = [1, 2, '中文']
len(s)
```

[20]: 3

1.2.7 min(s) & max(s)：s 的最小 / 大值

[21]:
```
s = [100, 999, 66.89]
min(s), max(s)
```

[21]: (66.89, 999)

[22]:
```
s = ['abc', 'def', 'azx']
min(s), max(s)
```

[22]: ('abc', 'def')

1.2.8 s.index(x)：x 在 s 中首次出现项的索引号

[23]:
```
s = [1, 2, 'ab', 3, 'cd']
s.index('ab')
```

[23]: 2

1.2.9 s.count(x)：x 在 s 中出现的总次数

[24]:
```
s = ['a', 'b', 1, 2, 'a', 1]
s.count('a')
```

[24]: 2

[25]:
```
s = [[], [], []]
s.count([])
```

[25]: 3

1.3 可变序列操作

1.3.1 s.append(x)：将元素 x 添加到 s 中

[26]:
```
s = ['ab', 'cd', 1]
s.append(2)
s
```

[26]: ['ab', 'cd', 1, 2]

1.3.2 s.extend(t)：用 t 的内容扩展 s

[27]:
```
s = [1, 2, 3]
t = ['a', 'b']
```

[28]:
```
s.extend(t)
s
```

[28]: [1, 2, 3, 'a', 'b']

[29]:
```
s.append(t)
s
```

[29]: [1, 2, 3, 'a', 'b', ['a', 'b']]

1.3.3 `s[i] = x`：将 `s` 的第 `i` 项替换为 `x`

[30]:
```
s = [1, 2, 4, 'a', 'c']
s[2] = 3
s
```

[30]:
```
[1, 2, 3, 'a', 'c']
```

1.3.4 `[]`：空列表

[31]:
```
s = []
s
```

[31]:
```
[]
```

1.3.5 `[[]]`：一个包含了一个空列表的单元素列表

`[[]] * 3` 结果中的三个元素都是对一个空列表的引用：

[32]:
```
s = [[]] * 3
s[0].append(1)
s[1].append(3)
s[2].append(5)
s
```

[32]:
```
[[1, 3, 5], [1, 3, 5], [1, 3, 5]]
```

创建以不同列表为元素的列表：

[33]:
```
s = [[], [], []]
s[0].append(1)
s[1].append(3)
s[2].append(5)
s
```

[33]:
```
[[1], [3], [5]]
```

1.4 tuple 中引用 list

[34]:
```
s1 = [1, 2]
s2 = [3, 4]
s3 = [5, 6]
```

[35]:
```
s = (s1, s2, s3)
s
```

[35]: ([1, 2], [3, 4], [5, 6])

list 改变，但 tuple 不变

[36]:
```
s2 = ['a', 'b']
s
```

[36]: ([1, 2], [3, 4], [5, 6])

list 改变，tuple 也改变

[37]:
```
s1.append('ab')
s
```

[37]: ([1, 2, 'ab'], [3, 4], [5, 6])

2 文本序列类型 —— str 字符串

由 Unicode 码位构成的不可变序列

2.1 定义

2.1.1 单引号：带转移字符，并且不能同时带有' 和"

[38]:
```
s = 'I\'m a batman.'
s
```

[38]: "I'm a batman."

2.1.2 双引号：可直接输出'，但" 需要带转义字符

[39]:
```
s = "John said: \"I'm a batman.\""
s
```

[39]:
```
'John said: "I\'m a batman."'
```

2.1.3 三重引号：句子里既有' 和" 时用"""，常用于代码前的大段注释

[40]:
```
s = """This is a python document.
Revise on 05.01

@author: Iris1946"""
s
```

[40]:
```
'This is a python document.\nRevise on 05.01\n\n@author: Iris1946'
```

2.2 注意点

2.2.1 r：告诉 python 解释器不要把\看成转义字符，而就是\本身

[41]:
```
s = 'd:\\dir\\a.txt'
print(s)
```

```
d:\dir\a.txt
```

[42]:
```
s = r'd:\dir\a.txt'
print(s)
```

```
d:\dir\a.txt
```

2.2.2 Unicode 编码，正确处理中文

[43]:
```
s = "中文 abc"
len(s), s[0]
```

[43]:
```
(5, ' 中')
```

2.2.3 字符串拼接

[44]:
```
s = '英文' + "abc" + str(24)
s
```

[44]:
```
' 英文 abc24'
```

2.3 格式化

[45]:
```
name = 'Mike'
age = 12
```

2.3.1 +

[46]:
```
name + " is " + str(age) + " years old."
```

[46]:
```
'Mike is 12 years old.'
```

2.3.2 %

str % tuple:【printf 风格】字符串 str 包含字符串字面值以及需要替换的% 字符。

[47]:
```
'%s is %03d years old.' % (name, age)
```

[47]:
```
'Mike is 012 years old.'
```

2.3.3 format

str.format tuple: 字符串 str 包含字符串字面值以及花括号 {} 括起来的替换域。

[48]:
```
'{0} is {1} years old.'.format(name, age)
```

[48]:
```
'Mike is 12 years old.'
```

[49]:
```
'{0} is {2} years old, and gonna to be {1} years old next year.'.format(name,␣
↪age + 1, age)
```

[49]:
```
'Mike is 12 years old, and gonna to be 13 years old next year.'
```

2.3.4 f

f str：字符串 str 包含字符串字面值以及花括号 {} 括起来的替换域。

[50]:
```
f'{name} is {age} years old.'
```

[50]:
```
'Mike is 12 years old.'
```

2.4 字符串的方法

2.4.1 seq.join(list)

【list→str】返回一个由 list 中的字符串拼接而成的字符串，调用 seq 作为元素间的分隔。

[51]:
```
name = ['Alice', 'Bob', 'Clark']
", ".join(name)
```

[51]:
```
'Alice, Bob, Clark'
```

2.4.2 str.replace(old, new)

返回字符串的副本，其中出现的所有子字符串 old 都被替换为 new。

[52]:
```
s = "Mike is 12 years old."
s.replace('Mike', 'Lisa')
```

[52]:
```
'Lisa is 12 years old.'
```

2.4.3 str.split(sep)

【str→list】返回一个由字符串内单词组成的列表，使用 sep 作为分隔字符串。

[53]:
```
text = '1, 2, 3, a, b, c'
text.split(', ')
```

[53]:
```
['1', '2', '3', 'a', 'b', 'c']
```

3 结构控制语句

3.1 if 语句

[54]:
```
score = 88
```

[55]:
```
if score < 60:
    print('C')
elif score >= 60 and score < 90:
    print('B')
else:
    print('A')
```

```
B
```

3.2 while 语句

[56]:
```
i = 0
```

[57]:
```
while i < 3:
    print(i)
    i += 1
```

```
0
1
2
```

3.3 for 语句

[58]:
```
scores = [66, 45, 93]
```

3.3.1 for 语句的句式

[59]:
```
for score in scores:
    print(score)
```

```
66
45
93
```

3.3.2 range 对象

range(stop): [0, stop - 1] 的值

[60]:
```
list(range(10))
```

[60]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

range(start, stop, step): **[start, stop - 1]** 中步长为 step 的值

[61]:
```
list(range(1, 10, 2))
```

[61]: [1, 3, 5, 7, 9]

range 在 for 语句中的应用

[62]:
```
for i in range(3):
    print(i)
```

```
0
1
2
```

[63]:
```
for i in range(len(scores)):
    print(scores[i])
```

```
66
45
93
```

[64]:
```
for i in range(len(scores) - 1, -1, -1):
    print(scores[i])
```

```
93
45
66
```

3.3.3 enumerate 对象

enumerate(seq): 将 seq 组合为一个索引序列，同时列出数据 value 和数据下标 index

[65]:
```
list(enumerate(scores))
```

[65]:
```
[(0, 66), (1, 45), (2, 93)]
```

enumerate(seq, start)：将 seq 组合为一个索引序列，同时列出数据 value 和数据下标 index（index 从 start 开始编号）

[66]:
```
list(enumerate(scores, 1))
```

[66]:
```
[(1, 66), (2, 45), (3, 93)]
```

enumerate 在 for 语句中的应用

[67]:
```
for idx, score in enumerate(scores):
    print(idx, score)
```

```
0 66
1 45
2 93
```

[68]:
```
for idx, score in enumerate(scores, 1):
    print(idx, score)
```

```
1 66
2 45
3 93
```

3.3.4 zip 对象

zip(it, ...)：将 it 中对应的元素打包成元组，然后返回由这些元组组成的列表。

[69]:
```
names = ['Alice', 'Bob', 'Clark']
```

[70]:
```
list(zip(names, scores))
```

[70]:
```
[('Alice', 66), ('Bob', 45), ('Clark', 93)]
```

zip 在 for 语句中的应用

[71]:
```
for name, score in zip(names, scores):
    print(name, score)
```

```
Alice 66
Bob 45
Clark 93
```

4 映射集合 —— dict

字典 dict 属于**可变对象**

4.1 字典的创建

字典 dict 可以通过以，分割的 key: value 对列表包含于花括号 {} 来创建：

[72]:
```
scores = {'Alice': 66, 'Bob': 45, 'Clark': 93}
```

dict 对象会将 hashable 值映射到任意对象，因此字典的 **key** 必须必是 **hashable** 的，包含 int、float、bool、str、tuple，而 **value** 可以是**任何值**（包括非 hashable 的 list 对象）。

4.2 字典的访问

4.2.1 d[key]：返回 d 中以 key 为键的值

[73]:
```
scores['Alice']
```

[73]: 66

4.2.2 d.get(key, default)

不设置 default：如果 key 存在于 d 中则返回 key 的 value，否则返回 None

[74]:
```
scores.get('Bob')
```

[74]: 45

[75]:
```
print(scores.get('Mike'))
```

```
None
```

设置 default：如果 key 存在于 d 中则返回 key 的 value，否则返回 default

[76]:
```
scores.get('Mike', 0)
```

[76]: 0

4.3 字典的方法

4.3.1 d[key] = value：将 d[key] 设为 value

[77]:
```
scores['Bob'] = 88
scores
```

[77]: {'Alice': 66, 'Bob': 88, 'Clark': 93}

4.3.2 del d[key]：将 d[key] 从 d 中移除

[78]:
```
del scores['Bob']
scores
```

[78]: {'Alice': 66, 'Clark': 93}

4.3.3 key in d：如果 d 中存在 key 则返回 True，否则返回 False

[79]:
```
'Clark' in scores
```

[79]: True

4.4 字典的遍历

[80]:
```
scoreSheet = {
    'Mike': [77, 87, 38],
    'Jack': [19, 98, 81],
    'Lisa': [100, 98, 88]
}
```

[81]:
```
for name in scoreSheet:
    print(name, scoreSheet[name])
```

```
Mike [77, 87, 38]
Jack [19, 98, 81]
Lisa [100, 98, 88]
```

4.4.1 d.keys(): 返回由字典 key 组成的一个新视图

[82]:
```
scoreSheet.keys()
```

[82]:
```
dict_keys(['Mike', 'Jack', 'Lisa'])
```

[83]:
```
for key in scoreSheet.keys():
    print(key, scoreSheet[key])
```

```
Mike [77, 87, 38]
Jack [19, 98, 81]
Lisa [100, 98, 88]
```

4.4.2 d.values(): 返回由字典 value 组成的一个新视图

[84]:
```
scoreSheet.values()
```

[84]:
```
dict_values([[77, 87, 38], [19, 98, 81], [100, 98, 88]])
```

[85]:
```
for value in scoreSheet.values():
    print(value)
```

```
[77, 87, 38]
[19, 98, 81]
[100, 98, 88]
```

4.4.3 d.items(): 返回由字典 (key, value) 组成的一个新视图

[86]:
```
scoreSheet.items()
```

[86]:
```
dict_items([('Mike', [77, 87, 38]), ('Jack', [19, 98, 81]), ('Lisa', [100, 98,
88])])
```

[87]:
```
for key, value in scoreSheet.items():
    print(key, value)
```

```
Mike [77, 87, 38]
Jack [19, 98, 81]
Lisa [100, 98, 88]
```

5 函数

5.1 函数的定义

[88]:
```
def foobar():
    pass
```

[89]:
```
foobar()
```

5.2 参数

5.2.1 引用传递

形参改变，但实参不改变

[90]:
```
def foobar(x):
    x = 2
```

[91]:
```
x = 1
x
```

[91]: 1

[92]:
```
foobar(x)
x
```

[92]: 1

形参改变，实参也改变

[93]:
```
def foobar(x):
    x[1] = 'a'
```

[94]:
```
x = [1, 2, 3]
x
```

[94]: [1, 2, 3]

[95]:
```
foobar(x)
x
```

[95]:
```
[1, 'a', 3]
```

5.2.2 实参的顺序

[96]:
```
def minus(x, y):
    return x - y
```

如果**实参**按 def 中 parameter_list 的**顺序输入**，则无需表明 parameter

[97]:
```
minus(6, 1)
```

[97]:
```
5
```

如果实参是**非顺序输入**，则传递时需要标明 parameter 的**名称**

[98]:
```
minus(y = 1, x = 6)
```

[98]:
```
5
```

5.2.3 形参的默认值

在定义 def 时可以设定 parameter 的默认值，如果在调用 def 时没有传递实参值，则用默认值进行计算

[99]:
```
def plus(x = 6, y = 3):
    return x + y
```

[100]:
```
plus()
```

[100]:
```
9
```

[101]:
```
plus(1)
```

[101]:
```
4
```

[102]:
```
plus(y = 7)
```

[102]:
```
13
```

注意：需要把**没有默认值**的 parameter 写在**前面!!!**

[103]:
```
def multiply(a, b, c = 1):
    return a * b * c
```

[104]:
```
multiply(4, 5, 3)
```

[104]: 60

5.2.4 *args: [list] 不定参数

[105]:
```
def foobar(x, y, *args):
    print(args)
```

[106]:
```
foobar(1, 2, 'abc', 2001, 'M')
```

```
('abc', 2001, 'M')
```

5.2.5 **kwargs: [dict] 带名字的不定参数

传参语法：key = value

[107]:
```
def foobar(x, y, *args, **kwargs):
    print(args)
    print(kwargs)
```

[108]:
```
foobar('a', 'b', 1, 2, Zhang = 97, Lei = 100)
```

```
(1, 2)
{'Zhang': 97, 'Lei': 100}
```

5.3 first class

5.3.1 变量引用

[109]:
```
def fun(x):
    return 3 * x ** 2 + 2 * x - 5
```

[110]:
```
y = fun
```

[111]:
```
y(2)
```

[111]: 11

5.3.2 参数传递

[112]:
```
def fun(x):
    return 3 * x ** 2 + 2 * x - 5
```

[113]:
```
def diff(a, f):
    delta = 1e-6
    return int((f(a + delta) - f(a)) / delta)
```

[114]:
```
diff(2, fun)
```

[114]: 14

5.3.3 加入到集合当中

[115]:
```
def fun1(x):
    return x ** 2
```

[116]:
```
def fun2(x):
    return x ** 3
```

[117]:
```
def fun3(x):
    return x ** 4
```

[118]:
```
funs = [fun1, fun2, fun3]
```

[119]:
```
for fun in funs:
    print(fun(5))
```

```
25
125
625
```

5.3.4 函数闭包

[120]:
```
def fun1(x):
    def fun2(y):
        return x * y
    return fun2
```

传入 x 的值后 fun1(x) 是一个 function 类，因此再次传入 y 的值

[121]:
```
f = fun1(2)
```

[122]:
```
f(3)
```

[122]: 6

以上步骤可简写为：

[123]:
```
fun1(3)(4)
```

[123]: 12

5.3.5 lambda 表达式

以下 lambda 表达式与 def 函数等价：

[124]:
```
lambda x : 3 * x ** 2 + 2 * x - 5
```

[124]: <function __main__.<lambda>(x)>

[125]:
```
def lambda_x(x):
    return 3 * x ** 2 + 2 * x - 5
```

lambda 表达式在 def 函数中的应用：

[126]:
```
def diff(a, f):
    delta = 1e-6
    return int((f(a + delta) - f(a)) / delta)
```

[127]:
```
diff(2, lambda x : 3 * x ** 2 + 2 * x - 5)
```

[127]: 14

6 异常处理

try 语句

[128]:
```
try:
    pass   # 可能会报错的语句
except Exception as e:
    pass   # 异常处理
finally:
    pass   # 无论怎样都会执行
```

example 1:

[129]:
```
try:
    x = float('abc')
except Exception as e:
    print(e.args)
```

```
("could not convert string to float: 'abc'",)
```

example 2：当 x 输入无误的值时，就让 x 正常赋值；而出现异常的话，则让 x 为 0。

[130]:
```
try:
    x = 0
    x = float('abc')
except Exception as e:
    print(e.args)
```

```
("could not convert string to float: 'abc'",)
```

[131]:
```
print(x)
```

```
0
```

example 3：出现多种 Error 时可以使用多个 except。

[132]:
```
try:
    x = float('abc')
    f = open('new.txt', 'r')
except ValueError as v:
```

```
    print(v.args)
except FileNotFound as f:
    print(f.args)
except Exception as e:
    print(e.args)
```

```
("could not convert string to float: 'abc'",)
```

example 4：input 输入的值出现异常。

[133]:
```
def askUser():
    while True:
        try:
            return float(input('P1z input a floating number:'))
        except ValueError as v:
            print(v.args)
```

askUser()

7 文件处理

7.1 读取文件

7.1.1 with 语句

with EXPRESSION as TARGET: SUITE

7.1.2 使用 with 语句读取文本

[134]:
```
with open('news.txt', 'r', encoding = 'UTF-8') as f:
    lines = f.readlines()
```

7.2 文本处理

7.2.1 str.strip()：返回字符串的副本，移除其中的前导和末尾字符。

[135]:
```
for line in lines:
    line = line.strip()
    tokens = line.split("。")
    print(tokens)
```

[' 一家刚刚成立两年的网络支付公司，它的目标是成为市值 100 亿美元的上市公司', '']
['']
[' 这家公司叫做快钱，说这句话的是快钱的 CEO 关国光', ' 他之前曾任网易的高级副总裁，负责过网易的上市工作',
' 对于为什么选择第三方支付作为创业方向，他曾经对媒体这样说：“我能看到这个胡同对面是什么，别人只能看到这个胡同',
'”自信与狂妄只有一步之遥——这几乎是所有创业者的共同特征，是自信还是狂妄也许需要留待时间来考证', '']
['']
[' 对于市值 100 亿美元的上市公司，他是这样算这笔账的，“百度上市时广告客户数量只有 4 万，而且它所做的只是把客户吸引过来，就可以支撑起现有的庞大市值；而我们几年
后的客户数量是几千万，而且这些客户都是能直接带来利润的，说市值 100 亿美元一点都不夸张', '”']
['']
[' 这家公司 2005 年年底注册用户达到 400 万，计划今年注册用户突破 1000 万，号称是国内最大的第三方网络支付平台',
'“在美国跟支付相关的收入已经超过了所有商业银行本身利差收入的总和，我所查到的数据是 3000 亿美元，其中超过 70%是个人消费者带来的收入',
'”关国光喜欢借用美国支付产业的现状与中国的情况进行比较', ' 虽然美国和中国差异显著，但他坚信中国的第三方支付市场前景非常广阔', '']

7.2.2 清除空列表（len == 0）：

for 循环

[136]:
```
sents = []
```

[137]:
```
for line in lines:
    line = line.strip()
    tokens = line.split('。')
```

```
    for token in tokens:
        if(len(token) > 0):
            sents.append(token)
```

[138]:
```
print(sents)
```

[' 一家刚刚成立两年的网络支付公司，它的目标是成为市值 100 亿美元的上市公司', ' 这家公司叫做快钱，说这句话的是快钱的 CEO 关国光',
' 他之前曾任网易的高级副总裁，负责过网易的上市工作',
' 对于为什么选择第三方支付作为创业方向，他曾经对媒体这样说：“我能看到这个胡同对面是什么，别人只能看到这个胡同',
'”自信与狂妄只有一步之遥——这几乎是所有创业者的共同特征，是自信还是狂妄也许需要留待时间来考证', ' 对于市值 100 亿美元的上市公司，他是这样算这笔账的，“百
度上市时广告客户数量只有 4 万，而且它所做的只是把客户吸引过来，就可以支撑起现有的庞大市值；而我们几年后的客户数量是几千万，而且这些客户都是能直接带来利润的，说市
值 100 亿美元一点都不夸张', '”', ' 这家公司 2005 年年底注册用户达到 400 万，计划今年注册用户突破 1000 万，号称是国内最大的第三方网络支付平台',
'“在美国跟支付相关的收入已经超过了所有商业银行本身利差收入的总和，我所查到的数据是 3000 亿美元，其中超过 70%是个人消费者带来的收入',
'”关国光喜欢借用美国支付产业的现状与中国的情况进行比较', ' 虽然美国和中国差异显著，但他坚信中国的第三方支付市场前景非常广阔']

filter 内置函数 filter(function, list)：用 list 中函数 function 返回 True 的那些元素，构成一个新的迭代器。

[139]:
```
numbers = [1, 2, 3, 4, 5]
```

[140]:
```
it = filter(lambda x : x % 2 == 0, numbers)
```

[141]:
```
list(it)
```

[141]: [2, 4]

filter 改写 for

[142]:
```
sents = []
```

[143]:
```
for line in lines:
    line = line.strip()
    tokens = line.split('。')
    it = filter(lambda x : len(x) > 0, tokens)
    sents.extend(it)
```

[144]:
```
print(sents)
```

```
[' 一家刚刚成立两年的网络支付公司，它的目标是成为市值 100 亿美元的上市公司', ' 这家公司叫做快钱，说这句话的是快钱的 CEO 关国光',
' 他之前曾任网易的高级副总裁，负责过网易的上市工作',
' 对于为什么选择第三方支付作为创业方向，他曾经对媒体这样说：“我能看到这个胡同对面是什么，别人只能看到这个胡同',
'”自信与狂妄只有一步之遥——这几乎是所有创业者的共同特征，是自信还是狂妄也许需要留待时间来考证', ' 对于市值 100 亿美元的上市公司，他是这样算这笔账的，“百
度上市时广告客户数量只有 4 万，而且它所做的只是把客户吸引过来，就可以支撑起现有的庞大市值；而我们几年后的客户数量是几千万，而且这些客户都是能直接带来利润的，说市
值 100 亿美元一点都不夸张', '”', ' 这家公司 2005 年年底注册用户达到 400 万，计划今年注册用户突破 1000 万，号称是国内最大的第三方网络支付平台',
'“在美国跟支付相关的收入已经超过了所有商业银行本身利差收入的总和，我所查到的数据是 3000 亿美元，其中超过 70%是个人消费者带来的收入',
'”关国光喜欢借用美国支付产业的现状与中国的情况进行比较', ' 虽然美国和中国差异显著，但他坚信中国的第三方支付市场前景非常广阔']
```

7.2.3 使用 “列表生成式” 处理文本

for 语句

[145]:
```
lens = []
```

[146]:
```
for sent in sents:
    lens.append(len(sent))
```

[147]:
```
print(lens)
```

```
[36, 24, 24, 54, 47, 121, 1, 55, 67, 27, 32]
```

列表生成式

[148]:
```
[(x, y) for x in range(1, 4) for y in range(2, 5)]
```

[148]: [(1, 2), (1, 3), (1, 4), (2, 2), (2, 3), (2, 4), (3, 2), (3, 3), (3, 4)]

列表生成式改写 for 语句

[149]:
```
lens = [len(sent) for sent in sents]
```

[150]:
```
print(lens)
```

[36, 24, 24, 54, 47, 121, 1, 55, 67, 27, 32]

7.2.4 查找文字

不导 re 包进行查找

[151]:
```
for sent in sents:
    if '公司' in sent:
        print(sent)
```

一家刚刚成立两年的网络支付公司，它的目标是成为市值 100 亿美元的上市公司
这家公司叫做快钱，说这句话的是快钱的 CEO 关国光
对于市值 100 亿美元的上市公司，他是这样算这笔账的，“百度上市时广告客户数量只有 4 万，而且它所做的只是把客户吸引过来，就可以支撑起现有的庞大市值；而我们几年后的客户数量是几千万，而且这些客户都是能直接带来利润的，说市值 100 亿美元一点都不夸张
这家公司 2005 年年底注册用户达到 400 万，计划今年注册用户突破 1000 万，号称是国内最大的第三方网络支付平台

导入 re 包进行查找

[152]:
```
import re
```

查找 “公司”

[153]:
```
regex = '公司'
```

[154]:
```
for sent in sents:
    if re.search(regex, sent) is not None:
        print(sent)
```

一家刚刚成立两年的网络支付公司，它的目标是成为市值 100 亿美元的上市公司
这家公司叫做快钱，说这句话的是快钱的 CEO 关国光

对于市值 100 亿美元的上市公司，他是这样算这笔账的，“百度上市时广告客户数量只有 4 万，而且它所做的只是把客户吸引过来，就可以支撑起现有的庞大市值；而我们几年后的客户数量是几千万，而且这些客户都是能直接带来利润的，说市值 100 亿美元一点都不夸张
这家公司 2005 年年底注册用户达到 400 万，计划今年注册用户突破 1000 万，号称是国内最大的第三方网络支付平台

查找 “市 + x”

[155]:
```
regex = '市.'
```

[156]:
```
for sent in sents:
    if re.search(regex, sent) is not None:
        print(sent)
```

一家刚刚成立两年的网络支付公司，它的目标是成为市值 100 亿美元的上市公司
他之前曾任网易的高级副总裁，负责过网易的上市工作
对于市值 100 亿美元的上市公司，他是这样算这笔账的，“百度上市时广告客户数量只有 4 万，而且它所做的只是把客户吸引过来，就可以支撑起现有的庞大市值；而我们几年后的客户数量是几千万，而且这些客户都是能直接带来利润的，说市值 100 亿美元一点都不夸张
虽然美国和中国差异显著，但他坚信中国的第三方支付市场前景非常广阔

查找 “百 or 上”

[157]:
```
regex = '[百上]'
```

[158]:
```
for sent in sents:
    if re.search(regex, sent) is not None:
        print(sent)
```

一家刚刚成立两年的网络支付公司，它的目标是成为市值 100 亿美元的上市公司
他之前曾任网易的高级副总裁，负责过网易的上市工作
对于市值 100 亿美元的上市公司，他是这样算这笔账的，“百度上市时广告客户数量只有 4 万，而且它所做的只是把客户吸引过来，就可以支撑起现有的庞大市值；而我们几年后的客户数量是几千万，而且这些客户都是能直接带来利润的，说市值 100 亿美元一点都不夸张

查找 “年份” （2xxx）

[159]:
```
regex = '2{0-9}{3}'
```

[160]:
```
for sent in sents:
    if re.search(regex, sent) is not None:
        print(sent)
```

查找作为一个整体的数字

[161]:
```
regex = '[0-9]+'
```

[162]:
```
for sent in sents:
    if re.search(regex, sent) is not None:
        print(re.findall(regex, sent))
        print(sent)
```

```
['100']
一家刚刚成立两年的网络支付公司，它的目标是成为市值 100 亿美元的上市公司
['100', '4', '100']
对于市值 100 亿美元的上市公司，他是这样算这笔账的，“百度上市时广告客户数量只有 4 万，而且它所做的只是把客户吸引过来，就可以支撑起现有的庞大市值；而我们几年后的客户数量是几千万，而且这些客户都是能直接带来利润的，说市值 100 亿美元一点都不夸张
['2005', '400', '1000']
这家公司 2005 年年底注册用户达到 400 万，计划今年注册用户突破 1000 万，号称是国内最大的第三方网络支付平台
['3000', '70']
“在美国跟支付相关的收入已经超过了所有商业银行本身利差收入的总和，我所查到的数据是 3000 亿美元，其中超过 70%是个人消费者带来的收入
```

7.2.5 写入文件

[163]:
```
with open('output.txt', 'w') as f:
    sents = [sent + '\n' for sent in sents]
    f.writelines(sents)
```