2019秋本科生《交通信息融合与挖掘》课程

第六讲
Python数据分析基础及典型案例

沈煜
嘉定校区交通运输工程学院311室
yshen@tongji.edu.cn
http://yushen.scripts.mit.edu/home/
2019年10月16日

Why Python?

- 简单
  - 适合阅读
  - 易学
  - 开源
  - 跨平台
  - 解释性（逐行执行）
  - 面向过程+面向对象
  - 可扩展（胶水）
  - 库
  - ……
- 推荐版本
  - 3.4 +

- 功能丰富
  - 数据处理：numpy, pandas, sqlite3
  - 爬虫：urllib2, requests, beautifulsoup
  - 可视化：matplotlib
  - 运筹优化：pygurobi
  - 机器学习：scikit-learn, tensorflow, keras….
  - 空间分析：pysal
  - 建站：django
  - 游戏：pygame
  - 交通仿真
  - ……

How to learn Python?

- 能动手尽量别吵吵
- 学习别人的代码
  - 琢磨别人解决问题的思路
  - 自己动手敲一遍
  - 不要Ctrl+C，Ctrl+V!
- 借助互联网社区
  - Google, GitHub, Stack Overflow

Python基础

- 表达式、变量、变量类型
- 逻辑运算与比较运算
- 函数
- 条件
- 列表与元组
- 字典
- 循环与循环控制
- 类
- 一些规范

表达式、变量、变量类型

```
print(3, -1, 3.14159, -2.0)
输出: 3 -1 3.14159 -2.0
print(type(-3))
输出: <class 'int'>
print(type(3.0))
输出: <class 'float'>
print(2.718281828459045235360287471352)
输出: 2.718281828459045 #最多17位有效数字
print(1 + 2 ** 3 / (4 * 5))
输出: 1.4
print(5 // 2, 5 % 2) #整除与求余数
输出: 2 1
```

表达式、变量、变量类型

```
print(type(True))
输出：<class 'bool'>
print(type('True'))
输出：<class 'str'>
print(type(b'True'))
输出：<class 'bytes'>
print(str(True))
输出：True
print(bytes('同济', encoding='utf-8'))
输出：b'\xe5\x90\x8c\xe6\xb5\x8e'
print("tong"+"ji") #字符串相连
输出：tongji
```

表达式、变量、变量类型

```
my_name = "Yu Shen"   #字符串string加引号
print(my_name)   #输出my_name变量的值
输出: Yu Shen
print(type(my_name))
输出: <class 'str'>
fahrenheit = 74   #整型变量int
print(type(fahrenheit))
输出:  <class 'int'>
celsius = 5 / 9 * (fahrenheit - 32)
print(celsius)   #python2和3的结果有区别
输出: 23.333333333333336
print(type(celsius))
输出: <class 'float'>
```

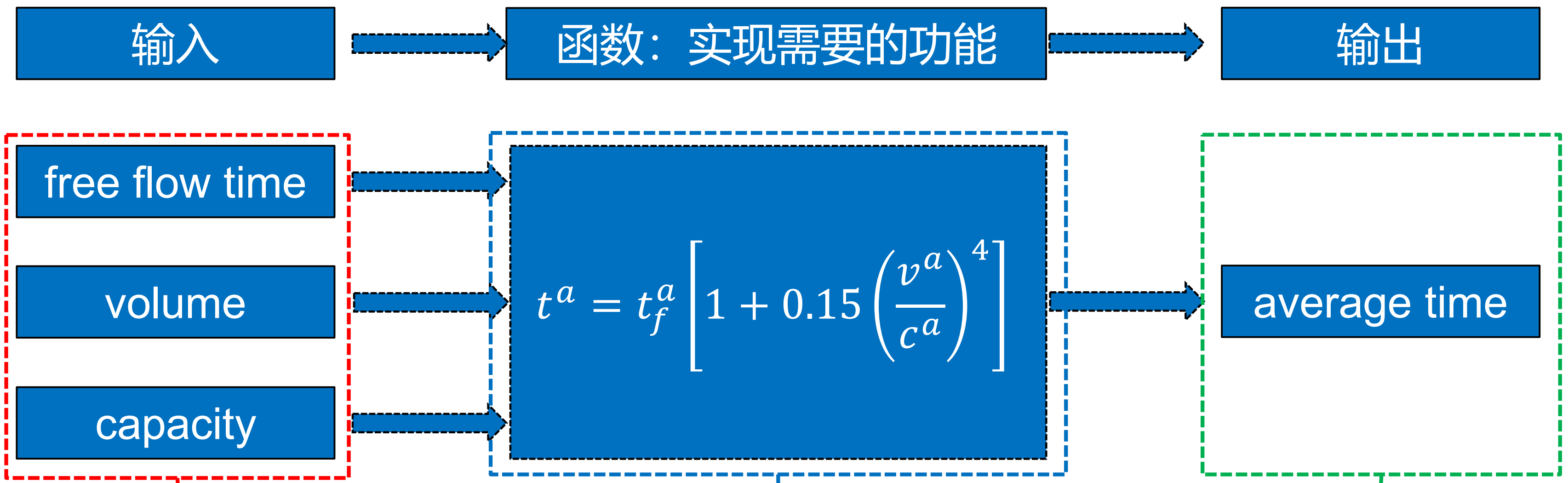
逻辑运算与比较运算

```
'''
逻辑运算
'''

a = True
b = False
print(a and b)
输出: False
print(a or b)
输出: True
print(not a)
输出: False
print(a and (not b))
输出: True
```

逻辑运算与比较运算

```
'''
比较运算
'''

a = 2 > 1
print(a)
输出: True
print(2 < 2, 2 >= 1, 2 <= 2)
输出: False True True
print("tongji" == 'tongji')
输出: True
print(2 != 2)
输出: False
```

函数



```
'''
The Bureau of Public Roads (BPR)'s link congestion function
'''

def bpr(ff_time, volume, capacity):
    mean_time = ff_time * (1 + 0.15 * (volume / capacity) ** 4)
    return mean_time
print(bpr(30, 2500, 3000))
```

输出：32.17013888888889

```python
import math

def haversine(lat1, lon1, lat2, lon2):
    radius = 6373.0
    r_lat1 = math.radians(lat1)
    r_lon1 = math.radians(lon1)
    r_lat2 = math.radians(lat2)
    r_lon2 = math.radians(lon2)

    dist_lat = r_lat2 - r_lat1
    dist_lon = r_lon2 - r_lon1

    a = math.sin(dist_lat/2)**2 + math.cos(r_lat1) * \
        math.cos(r_lat2) * math.sin(dist_lon/2)**2
    c = 2 * math.atan2(a**0.5, (1-a)**0.5)

    distance = radius * c
    return distance

dist = haversine(31.288644, 121.213237, 31.282218, 121.505649)
print('The direct distance between two campuses is {:.2f} km.'
      .format(dist))
```

调用math包

注意冒号

调用math中的radians函数，如果用from math import *，则在调用函数时写为：
r_lat1 = randians(lat1)

换行，有括号不用

输出结果：
The distance between two campuses is 27.80 km.

{}内保留小数点2位

在{}内填入dist

条件

```python
from math import *

def choice(time_bus, cost_bus, time_taxi, cost_taxi, time_bike):
    u_bus = -0.35 * time_bus - 0.5 * cost_bus
    u_taxi = -0.65 * time_taxi - 0.1 * cost_taxi
    u_bike = -0.25 * time_bike
    p_bus = exp(u_bus) / (exp(u_bus) + exp(u_taxi) + exp(u_bike))
    p_taxi = exp(u_taxi) / (exp(u_bus) + exp(u_taxi) + exp(u_bike))
    p_bike = 1 - p_bus - p_taxi
    if p_bus > p_taxi and p_bus > p_bike:
        return 'bus', p_bus
    elif p_taxi > p_bus and p_taxi > p_bike:
        return 'taxi', p_taxi
    else:
        return 'bike', p_bike

my_choice, prob = choice(time_bus=110, cost_bus=10,
                         time_taxi=50, cost_taxi=130, time_bike=180)
print("My choice is {} with the probability of {:.2f}."
      .format(str(my_choice), prob))
```

输出结果:

```
My choice is bus with the probability of 0.74.
```

列表与元组(list & tuple)

```
a = [3, 2, 7, 5, 11, 3]
a.extend([17, 13])
print(a)
输出: [3, 2, 7, 5, 11, 3, 17, 13]
a.append(19)
print(a)
输出: [3, 2, 7, 5, 11, 3, 17, 13, 19]
print(a.count(3))
输出: 2
print(a.index(11))
输出: 4
```

列表与元组(list & tuple)

```
[3, 2, 7, 5, 11, 3, 17, 13, 19]
```

```
a.remove(3)
print(a)
输出: [2, 7, 5, 11, 3, 17, 13, 19]
a.sort(reverse=False)
print(a)
输出: [2, 3, 5, 7, 11, 13, 17, 19]
a.insert(-1, 23)
print(a)
输出: [2, 3, 5, 7, 11, 13, 17, 23, 19]
a.reverse()
print(a)
输出: [19, 23, 17, 13, 11, 7, 5, 3, 2]
```

列表与元组(list & tuple)

```
[19, 23, 17, 13, 11, 7, 5, 3, 2]
```

```
a.pop(0)
print(a)
输出: [23, 17, 13, 11, 7, 5, 3, 2]
print(a[:3])
输出: [23, 17, 13]
print(a[2:])
输出: [13, 11, 7, 5, 3, 2]
print(a[4])
输出: 7
```

字典(dictionary)

```
bike_gps = {"065033209": [[1498751902, 103.807374, 1.451625],
                           [1498776159, 103.807374, 1.451625],
                           [1498776222, 103.808221, 1.452058],
                           [1498797735, 103.808221, 1.452058],
                           [1498798179, 103.808235, 1.451961],
                           [1498817388, 103.808235, 1.451961]
                          ],
             "065034465": [[1498751904, 103.742517, 1.319935],
                           [1498884837, 103.742517, 1.319935],
                           [1498885083, 103.743094, 1.320579],
                           [1498885221, 103.743094, 1.320579],
                           [1498885731, 103.739962, 1.321119],
                           [1498890654, 103.739962, 1.321119]
                          ]
            }

print(bike_gps["065033209"])
print(bike_gps["065033209"][0])
```

{key1: value1, key2: value2, …}

每个key对应的value是一个list：由list组成的list

字典(dictionary)

```
print(bike_gps["065033209"])
```
输出：
```
[[1498751902, 103.807374, 1.451625], [1498776159, 103.807374, 1.451625],
[1498776222, 103.808221, 1.452058], [1498797735, 103.808221, 1.452058],
[1498798179, 103.808235, 1.451961], [1498817388, 103.808235, 1.451961]]
```

```
print(bike_gps["065033209"][0])
```
输出：
```
[1498751902, 103.807374, 1.451625]
```

```
if "065033209" in bike_gps:
    bike_gps["065033209"].append([1498817643,
                                  103.807188, 1.451811])
```

```
print(bike_gps)
```
输出：
```
{‘065033209’: [[1498751902, 103.807374, 1.451625],…,[1498817388, 103.808235,
1.451961], [1498817643, 103.807188, 1.451811]], '065034465': [[1498751904,
103.742517, 1.319935],…,[1498890654, 103.739962, 1.321119]]}
```

循环

```python
init_list = list(range(26))
print(init_list)
alpha_list = []
for i in init_list:
    alpha_list.append(i+97)

print(alpha_list)

alpha_list = [97]
while len(alpha_list) < 26:
    k = alpha_list[-1]
    alpha_list.append(k+1)

print(alpha_list)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
12, 13, 14, 15, 16, 17, 18, 19, 20,
21, 22, 23, 24, 25]
```

```
[97, 98, 99, 100, 101, 102, 103,
104, 105, 106, 107, 108, 109,
110, 111, 112, 113, 114, 115, 116,
117, 118, 119, 120, 121, 122]
```

循环+条件

```python
alpha_list = [97]
while len(alpha_list) < 26:
    k = alpha_list[-1]
    alpha_list.append(k+1)

cipher_dict = {}
for k in alpha_list:
    step = 3
    if k + step > 122:
        n = k + step - 26
    else:
        n = k + step
    cipher_dict[chr(k)] = chr(n)
```

```
print(cipher_dict)
{'a': 'd', 'b': 'e', 'c': 'f', 'd':
'g', 'e': 'h', 'f': 'i', 'g': 'j',
'h': 'k', 'i': 'l', 'j': 'm', 'k':
'n', 'l': 'o', 'm': 'p', 'n': 'q',
'o': 'r', 'p': 's', 'q': 't', 'r':
'u', 's': 'v', 't': 'w', 'u': 'x',
'v': 'y', 'w': 'z', 'x': 'a', 'y':
'b', 'z': 'c'}
```

```
print(cipher_dict.items())
dict_items([('a', 'd'), ('b', 'e'),
('c', 'f'), ('d', 'g'), ('e', 'h'),
('f', 'i'), ('g', 'j'), ('h', 'k'),
('i', 'l'), ('j', 'm'), ('k', 'n'),
('l', 'o'), ('m', 'p'), ('n', 'q'),
('o', 'r'), ('p', 's'), ('q', 't'),
('r', 'u'), ('s', 'v'), ('t', 'w'),
('u', 'x'), ('v', 'y'), ('w', 'z'),
('x', 'a'), ('y', 'b'), ('z', 'c')])
```

循环+字典

```python
def encoder(message):
    encoded_message = ""
    for ch in message:
        encoded_message += cipher_dict[ch]
    return encoded_message


print(encoder("python"))
```

密钥

```
{'a': 'd', 'b': 'e', 'c': 'f', 'd': 'g', 'e': 'h', 'f': 'i',
'g': 'j', 'h': 'k', 'i': 'l', 'j': 'm', 'k': 'n', 'l': 'o',
'm': 'p', 'n': 'q', 'o': 'r', 'p': 's', 'q': 't', 'r': 'u',
's': 'v', 't': 'w', 'u': 'x', 'v': 'y', 'w': 'z', 'x': 'a',
'y': 'b', 'z': 'c'}
```

输出：sbwkrq

循环+字典

```python
def decoder(message):
    decoded_message = ""
    for ch in message:
        for key, value in cipher_dict.items():
            if ch == value:
                decoded_message += key
    return decoded_message

print(decoder("sbwkrq"))
```

```
dict_items([('a', 'd'), ('b', 'e'), ('c', 'f'), ('d', 'g'), ('e', 'h'),
('f', 'i'), ('g', 'j'), ('h', 'k'), ('i', 'l'), ('j', 'm'), ('k', 'n'),
('l', 'o'), ('m', 'p'), ('n', 'q'), ('o', 'r'), ('p', 's'), ('q', 't'),
('r', 'u'), ('s', 'v'), ('t', 'w'), ('u', 'x'), ('v', 'y'), ('w', 'z'),
('x', 'a'), ('y', 'b'), ('z', 'c')])
```

输出：python

循环控制（break）

- 在语句块执行过程中终止循环，并且跳出整个循环。

```
count = 1
sum = 0

while count <= 100:
    sum = sum + count
    if sum > 1000:
        break
    count = count + 1

print(count, sum)
```

输出：

```
45 1035
```

循环控制（continue）

- 在语句块执行过程中终止当前循环，跳出该次循环，执行下一次循环。

```python
count = 1
sum = 0
while count <= 100:
    if count % 2 == 0:
        count = count + 1
        continue
    sum = sum + count
    count = count + 1
print(count, sum)
```

输出：101 2500

循环控制（pass）

- 什么都不做。

```
count = 1
sum = 0
while count <= 100:
    if count % 2 == 0:
        count = count + 1
        pass
    sum = sum + count
    count = count + 1

print(count, sum)
```

输出：102 2601

类（Class）

- 类是用来描述具有相同的属性和方法的对象的集合。它定义了该集合中每个对象所共有的属性和方法。
- 类是对象的模板，对象是类的实例。

类

```python
class Vehicle(object):
    class_name = "vehicle"

    def __init__(self, brand, license_plate):
        self.brand = brand
        self.license_plate = license_plate

    def my_info(self):
        print("The brand is {} with license plate {}"
              .format(str(self.brand),
                      str(self.license_plate)))

car_1 = Vehicle("Toyota", "ABC123")
print(car_1.class_name)
car_1.my_info()
```

输出:

```
vehicle
The brand is Toyota with license plate ABC123
```

```python
import math


class Vehicle(object):
    class_name = "vehicle"

    def __init__(self, brand, license_plate):
        self.brand = brand
        self.license_plate = license_plate

    def my_info(self):
        print("The brand is {} with license plate {}"
              .format(str(self.brand), str(self.license_plate)))

    def haversine(self, lat1, lon1, lat2, lon2):
        radius = 6373.0
        r_lat1 = math.radians(lat1)
        r_lon1 = math.radians(lon1)
        r_lat2 = math.radians(lat2)
        r_lon2 = math.radians(lon2)

        dist_lat = r_lat2 - r_lat1
        dist_lon = r_lon2 - r_lon1
        a = math.sin(dist_lat / 2) ** 2 + math.cos(r_lat1) * \
            math.cos(r_lat2) * math.sin(dist_lon / 2) ** 2
        c = 2 * math.atan2(a ** 0.5, (1-a) ** 0.5)

        distance = radius * c

        return distance
```

类

```python
car_1 = Vehicle("Toyota", "ABC123")
print(car_1.class_name)
car_1.my_info()

car_2 = Vehicle("Volkswagen", "DEF456")
print(car_2.class_name)
car_2.my_info()
dist = car_2.haversine(31.288644, 121.213237,
                       31.282218, 121.505649)
print(dist)
```

输出:

```
vehicle
The brand is Toyota with license plate ABC123
vehicle
The brand is Volkswagen with license plate DEF456
27.804696570232892
```

子类

```python
class Luxury(Vehicle):

    def __init__(self, brand, license_plate, price):
        Vehicle.__init__(self, brand, license_plate)
        self.price = price

    def declaration(self):
        print("I am a luxury car!")


car_3 = Luxury("BMW", "GHI789", 300000)
print(car_3.class_name)
car_3.my_info()
print(car_3.price)
car_3.declaration()
```

输出:

```
vehicle
The brand is BMW with license plate GHI789
300000
I am a luxury car!
```

类

（父）类：车

品牌
车牌号

品牌
车牌号

价格
我是豪车

（子）类：豪车

继承

实例：
车牌ABC123的Toyota

实例：
车牌GHI的BMW
输出“我是豪车”

命名规范

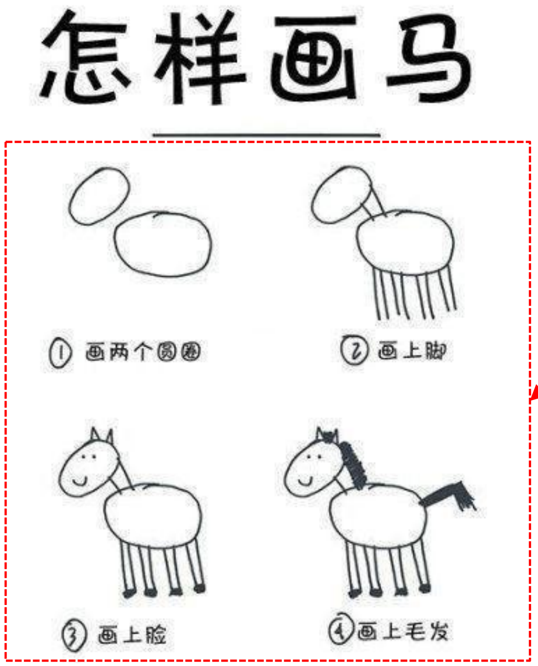
- 包、模块名全小写：module_name, package_name
- 类名首字母大写单词串：ClassName
- 方法、函数名全小写：method_name, function_name
- 变量全小写：local_var_name
- 全局变量（常量）全大写：GLOBAL_VAR_NAME
- 避免：
  - 单字符名称，除计数器与迭代器
  - 包与模块中的连字符（-）
  - 双下划线开头（Python保留）
- 约定俗成的缩写：
  - fn, txt, obj, cnt, num等

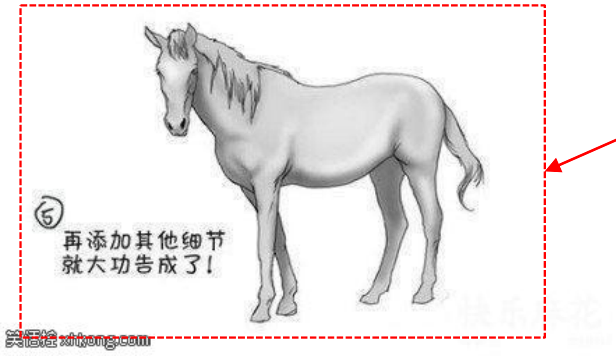
```
for i in range(10):
    print(i**2)


if __name__ == "__main__":
    pass
```

格式规范

- 分号：不要在行尾加分号, 也不要用分号将两条命令放在同一行
- 行长度：每行不超过80个字符
  - 长的导入模块语句、注释里的URL除外
- 括号：该用的时候再用
- 缩进：用4个空格来缩进代码
- **空行：**顶级定义之间空两行, 方法定义之间空一行
- **空格：**按照标准的排版规范来使用标点两边的空格，括号内不要有空格
- **导入格式：**每个导入应该独占一行
- **语句：**通常每个语句应该独占一行

作业





今天的学习内容

作业：
根据摩拜单车GPS轨迹，画出出行时间、距离与速度的分布直方图

作业要求

- Python源代码
  - 发邮箱：yshen@tongji.edu.cn
    - 以第一封邮件为准
  - 截止时间：**2019年10月23日13:29**
    - 以邮件时间戳为准
  - 代码命名规则：学号_姓拼音_名拼音.py
    - 例如：1750919_Zhang_Chenkeng.py
  - 尽量不要用Python标准库以外的模块
    - 可以用matplotlib画图，也可以用其它软件画图
    - 在别人的电脑上可以直接运行
- 纸质实验报告
  - 截止时间：**2019年10月23日上课前**
  - 默认页边距、双面打印，**不许超过10页**（保护森林资源）
  - 不要封面页，首页注明标题、姓名、学号
  - 正文格式：1.5倍行距、小四（12号）、衬线字体（图片和表格不受此限制）

可能需要用到的功能：读文件

```python
import csv

file = 'mobike_reordered.txt'
bike_list = []

with open(file, newline='') as f:
    reader = csv.reader(f, delimiter=',', quotechar='"')
    next(reader)
    for row in reader:
        order_id = row[2]
        bike_id = row[3]
        user_id = row[4]
        start_time = float(row[0])
        start_time = int(start_time)
        start_lon = float(row[5])
        start_lat = float(row[6])
        trip_info = [order_id, bike_id, user_id, start_time, (start_lon, start_lat)]
        bike_list.append(trip_info)

print(bike_list[0:3])
```

参考输出：

```
[['17086', '110', '110', 1469980862, (121.459, 31.192)], ['17090', '2635', '2635',
1469980894, (121.457, 31.317)], ['17093', '2484', '2484', 1469981765, (121.46,
31.197)]]
```

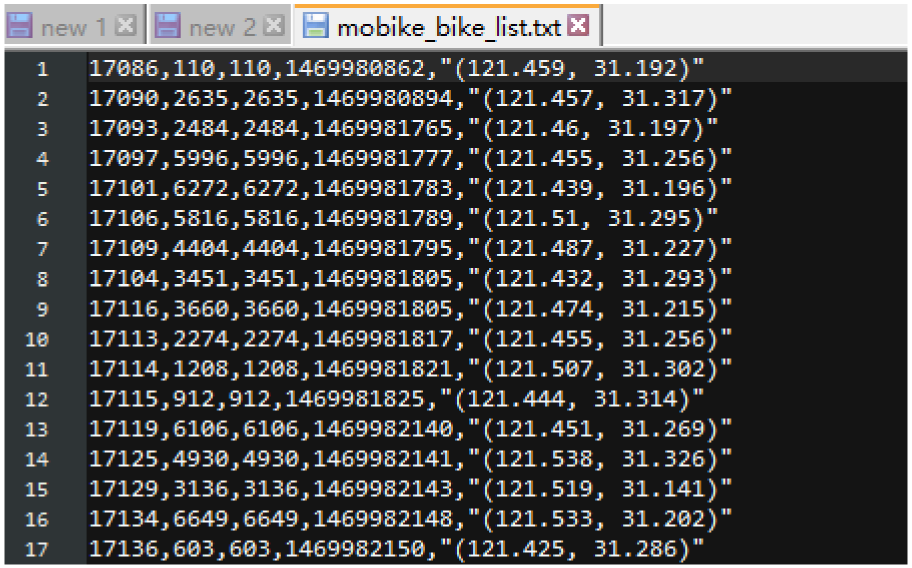
可能需要用到的功能：写文件

bike_list的前三行:

```
[['17086', '110', '110', 1469980862, (121.459, 31.192)], ['17090',
'2635', '2635', 1469980894, (121.457, 31.317)], ['17093', '2484',
'2484', 1469981765, (121.46, 31.197)]]
```

```python
with open("mobike_bike_list.txt", "w") as f:
    writer = csv.writer(f, lineterminator='\n')
    writer.writerows(bike_list)
```

输出txt文件:



```
17086,110,110,1469980862,"(121.459, 31.192)"
17090,2635,2635,1469980894,"(121.457, 31.317)"
17093,2484,2484,1469981765,"(121.46, 31.197)"
17097,5996,5996,1469981777,"(121.455, 31.256)"
17101,6272,6272,1469981783,"(121.439, 31.196)"
17106,5816,5816,1469981789,"(121.51, 31.295)"
17109,4404,4404,1469981795,"(121.487, 31.227)"
17104,3451,3451,1469981805,"(121.432, 31.293)"
17116,3660,3660,1469981805,"(121.474, 31.215)"
17113,2274,2274,1469981817,"(121.455, 31.256)"
17114,1208,1208,1469981821,"(121.507, 31.302)"
17115,912,912,1469981825,"(121.444, 31.314)"
17119,6106,6106,1469982140,"(121.451, 31.269)"
17125,4930,4930,1469982141,"(121.538, 31.326)"
17129,3136,3136,1469982143,"(121.519, 31.141)"
17134,6649,6649,1469982148,"(121.533, 31.202)"
17136,603,603,1469982150,"(121.425, 31.286)"
```

可能需要用到的功能：str转list

```python
track = "121.459,31.197#121.461,31.198#121.462,31.198#121.463,31.198"
track_list = track.split("#")
print(track_list)
```

输出结果：

```
['121.459,31.197', '121.461,31.198', '121.462,31.198',
'121.463,31.198']
```

第六讲结束

沈煜
嘉定校区交通运输工程学院311室
yshen@tongji.edu.cn
http://yushen.scripts.mit.edu/home/
2019年10月16日