



优达学城
UDACITY

Python入门

DAND VIP班公开课





目录 Contents

01



基本语言操作

脚本运行与文件操作



02

03



函数和模块

NumPy 和 pandas



04

05



项目导览



基本语言操作 01





缩进

学习 Python 与其他语言最大的区别就是, Python 的代码块不使用大括号 `{}` 来控制类、函数以及其他逻辑判断, 而是用缩进来组织代码。

缩进相同的一组语句构成一个代码块。像 `if-else`、`while`、`def` 和 `class` 这样的复合语句, 首行以关键字开始, 以冒号(`:`)结束, 该行之后的一行或多行代码构成代码块。

```
def judge(x):  
    if x:  
        print('True')  
        print(x)  
    else:  
        print('False')  
        print(x)
```



缩进易错点

- 同一层级的代码块缩进不统一；
- 混用空格和 Tab 键进行缩进；
- 应该缩进的地方没缩进，会影响代码逻辑。

```
if True:
    print('True')
else:
    print('False')
    print('Answer')
# 同一代码块中没有统一缩进会报错
```

```
def judge(x):
    if x:
        print('True')
    else:
        print('False')
```



缩进代码规范

PEP8 建议每级缩进都使用四个空格，即可提高可读性，又留下了足够的多级缩进空间。

各种代码编辑器都有对应的设置，可以将制表符转换为空格，还可以将制表符与空格分别都显示出来，空格是点，制表符为一条线。

Sublime 设置示例：

```
—— "draw_white_space": "all", // 绘制所有空白符  
—— "tab_size": 4, // 一个制表符相当于多少个空格  
—— "translate_tabs_to_spaces": true, ... // 输入 Tab 时自动转换为空格
```




多行语句

Python语句中一般以新行作为语句的结束符。

但是我们可以使用斜杠(\)将一行的语句分为多行显示

如果包含[], {}或()则不需要添加斜杠:

```
total = item_one + \
        item_two + \
        item_three
```

VS

```
days = ['Monday', 'Tuesday',
        'Wednesday', 'Thursday',
        'Friday']
```



代码行长度

PEP8 规范, 每行限制为79个字符;

换行的首先方式是使用Python中{}、[]、()的隐式行连接。

隐式行连接的使用方式应该优先于使用右斜杠(\)符号的方式。

利用隐式行连接处理长字符串换行的示例:

```
s = ('This is a very long long long long'  
     'long long long long long long string')
```




同一行显示多条语句

Python可以在同一行中使用多条语句, 语句之间使用分号(;)分割, 以下是一个简单的实例:

```
a = 5; b = 6; c = 7
```

注意: 在一行放置多条语句的做法在 Python 中一般是不推荐的, 因为这往往会降低代码的可读性。



空行

函数之间或类的方法之间用空行分隔，表示一段新的代码的开始。类和函数入口之间也用一行空行分隔，以突出函数入口的开始。

空行与代码缩进不同，空行**并不是Python语法的一部分**。书写时不插入空行，Python解释器运行也不会出错。但是空行的作用在于**分隔两段不同功能或含义的代码**，便于日后代码的维护或重构。

记住：空行也是程序代码的一部分。



注释

注释的文本 Python 解释器不会运行，可以利用这一点来注释掉一些暂时不需要运行又不想删除的代码。

在代码前添加一个 # 号即可使其变成注释文本。

Windows 快捷键: Ctrl + / Mac 快捷键: Command + /

```
# if True:
#     print('True')
# else:
#     print('False')
```



行注释与块注释

对于复杂的操作，可以在代码块开始前进行块注释，解释整块代码。

对于某一行代码的注释，可以在行尾添加行注释。

```
# We use a weighted dictionary search to find out where i is in  
# the array. We extrapolate position based on the largest num  
# in the array and the array size and then do binary search to  
# get the exact number.
```

```
if i & (i-1) == 0:           # true iff i is a power of 2
```



文档字符串

python 中使用三个单引号('')或三个双引号('\"'\")的多行注释, 通常用于文档字符串(docstring), 是对函数、类、模块等功能的文档注解。可以通过调用函数的 `__doc__` 属性来读取。

```
'''
```

```
这是一个多行注释, 使用三个单引号  
这是一个多行注释, 使用三个单引号  
这是一个多行注释, 使用三个单引号
```

```
'''
```

```
"""
```

```
这是一个多行注释, 使用三个双引号  
这是一个多行注释, 使用三个双引号  
这是一个多行注释, 使用三个双引号
```

```
"""
```



引号

Python 可以使用单引号('single')、双引号("double")、三引号("""triple""" 或 '''triple''') 来表示字符串, 引号的开始与结束必须为相同类型的。

其中三引号可以由多行组成, 编写多行文本的快捷语法, 常用于文档字符串, 在文件的特定地点, 被当做注释。

```
single = 'single quotes'
double = "double quotes"
triple = """triple quotes
三重引号可以支持多行字符串"""
```

```
# 如果字符串本身包含引号
with_quote = "That's right!"
with_quote = 'That\'s right!'
with_quote = 'That's wrong!'
```




输入与输出

input 语句用于获取用户输入，输入的内容作为字符串传入代码中

print 语句用于程序的输出打印，可以打印固定字符串或者变量的值

```
# 获取用户输入并赋值给 name 变量
name = input("What's your name?\n")

# 输出字符串和变量名，逗号会自动添加一个空格
print("Hello", name, "!")

# 可以做字符串的加法组成新的字符串
# 字符串相加不会自动添加空格，需要手动添加
print("Hello " + name + "!")

# 还可以使用 format 方法
# 其中 {} 为 format 括号内的值所要显示的地方
print("Hello {}".format(name))
```



运行效果(教室工作区测试)

A terminal window titled "Terminal 1" with a dark background. The prompt "root@83d795d831b1:/home/workspace#" is displayed in green and blue text, followed by a white cursor. The terminal window has a plus sign on the left and a close button on the right.

```
+ Terminal 1 ×  
root@83d795d831b1:/home/workspace#
```

脚本运行与文件操作

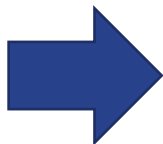
02



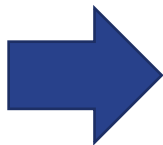


使用命令提示符或终端进行导航

Mac 终端



Windows 命令提示符

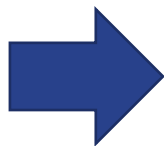


修改目录	列出当前目录的文件
<i>cd path</i> change directory	<i>ls</i> list
<i>cd path</i> 修改盘符 <i>d</i> :	<i>dir</i> directory



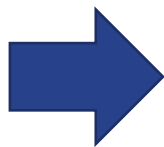
* 使用 Python 操作

获得当前路径



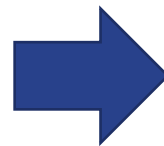
```
cwd=os.getcwd()  
print(cwd)
```

得到当前文件夹下的所有文件和文件夹



```
print(os.listdir())
```

更改当前目录



```
os.chdir( "C:\\123")
```



运行脚本

运行其他目录下的脚本

```
$ python path/filename.py
```

此时当前工作目录不等于 python 脚本所在目录，所以如果脚本中存在相对路径的文件读写操作，可能出现找不到文件的情况。建议将脚本所在目录修改为当前工作目录后，再运行脚本，避免相对路径与预期不符的问题。

当前目录下运行脚本

```
$ cd path  
$ python filename.py
```




绝对路径和相对路径：

相对路径相对的是当前工作目录

```
open('test1.txt') #1
```

```
open('/temp/test2.txt') #2
```

```
open('D:\\user\\test3.txt') #3
```

```
open(r'D:\user\temp\test4.txt') #4
```

相对路径

绝对路径



读文件*

打开一个文件用open()方法(open()返回一个文件对象, 它是可迭代的):

```
f = open('test.txt', 'r')
```

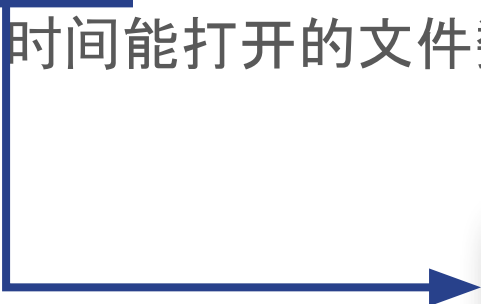
```
for line in f:  
    pass
```

如果文件不存在, open()函数就会抛出一个IOError的错误, 并且给出错误码和详细的信息告诉你文件不存在。



读文件*

文件使用完毕后**必须关闭** 因为文件对象会占用操作系统的资源, 并且操作系统同一时间能打开的文件数量也是有限的。



```
f.close()
```



读文件*

由于文件读写时都有可能产生IOError, 一旦出错, 后面的f.close()就不会调用。所以, 为了保证**无论是否出错**都能正确地关闭文件, 我们可以使用try ... finally来实现:

```
try:
    f = open('/path/to/file', 'r')
    print(f.read())
finally:
    if f:
        f.close()
```

- try中报错则运行finally



读文件*

但是每次都这么写实在太繁琐，所以，Python引入了with语句来自
动帮我们调用close()方法：

```
with open('/path/to/file', 'r') as f:  
    print(f.read())
```



写文件*

写文件和读文件是一样的，唯一区别是调用open()函数时，传入标识符'w'或者'wb'表示写文本文件或写二进制文件：

```
f = open('test.txt', 'w') # 若是'wb'就表示写二进制文件
f.write('Hello, world!')
f.close()
```

* 'w'的意义：如果没有这个文件，就创建一个；如果有，那么就会先把原文件的内容清空再写入新的东西。所以若不想清空原来的内容而是直接在后面追加新的内容，就用'a'这个模式。



写文件*

我们可以反复调用write()来写入文件，但是务必要调用f.close()来关闭文件。当我们写文件时，操作系统往往不会立刻把数据写入磁盘，而是放到内存缓存起来，空闲的时候再慢慢写入。只有调用close()方法时，操作系统才保证把没有写入的数据全部写入磁盘。忘记调用close()的后果是数据可能只写了一部分到磁盘，剩下的丢失了。所以，还是用with语句来得保险：

```
with open('test.txt', 'w') as f:  
    f.write('Hello, world!')
```



字符编码*

要读取非UTF-8编码的文本文件，需要给open()函数传入encoding参数，例如，读取GBK编码的文件：

```
f = open('test.txt', 'r', encoding='gbk')  
f.read()  
'测试'
```



字符编码*

遇到有些编码不规范的文件，你可能会遇到UnicodeDecodeError，因为在文本文件中可能夹杂了一些非法编码的字符。遇到这种情况，open()函数还接收一个errors参数，表示如果遇到编码错误后如何处理。最简单的方式是直接忽略：

```
f = open('test.txt', 'r', encoding='gbk', errors='ignore')
```



异常处理

```
Traceback (most recent call last):  
  File "p2-test.py", line 138, in <module>  
    print(float('some'))  
ValueError: could not convert string to float: 'some'
```

```
def attempt_float(x):  
    try:  
        return float(x)  
    except ValueError:  
        return x  
  
In [20]: attempt_float('1.23')  
Out[20]: 1.23  
  
In [21]: attempt_float('some')  
Out[21]: 'some'
```



常见异常

异常名字	描述
FileNotFoundError	路径下找不到文件
KeyError	映射中(字典、DataFrame 等)没有这个键
NameError	未声明/初始化对象 (没有属性)
SyntaxError	语法错误
ValueError	传入无效的参数
TypeError	对类型无效的操作
IndentationError	缩进错误
IOError	输入/输出操作失败

函数和模块 03





什么是函数

函数是带名字的代码块，用于完成具体的工作。定义后可以使用函数名来重复调用。



将函数**抽象理解**为一个功能按钮，点击不同按钮实现不同的功能

输入是传入函数的参数，输出是函数的返回值。

Python中存在无参数或者无返回值的函数，无返回值则默认返回 None



无参数或者无返回值的函数
无返回值默认返回 None

```
def greet_user():  
    """显示简单的问候语"""  
    print('Hello!')  
  
# 调用函数  
greet_user()
```

有参数有返回值的函数

```
def sum_up(n):  
    """  
    传入一个正整数 n,  
    求 1+2+...+n 的累加值  
    """  
    result = 0  
    for i in range(n):  
        result += i  
    return result  
  
# 调用函数  
sum_to_10 = sum_up(10)  
print(sum_to_10)
```



创建函数

(1) def 关键字

(2) 函数体

```
def greet_user():  
    """显示简单的问候语"""  
    print('Hello!')  
  
# 调用函数  
greet_user()
```

紧跟在 def 语句下面的所有缩进行构成了函数体。

上图中引号括起来的部分是文档字符串的注释，描述函数是做什么的。

代码行 print('Hello!') 是函数体内唯一一行代码，该函数只实现一项工作：打印 Hello! 。



形参和实参

写在 `def` 语句中函数名后面括号内的变量通常叫做函数的**形参**，而**调用**函数时提供的值是**实参**，一般都可以称为函数的**参数**。

- 形参出现在函数定义中，在整个函数体内都可以使用，离开该函数则不能使用。
- 实参出现在函数外部的代码中，进入被调函数后，实参变量名也不能使用。
- 形参和实参的功能是作数据传送。发生函数调用时，代码把实参的值传送给被调函数的形参从而实现向被调函数的数据传送。



调用函数

(1) 调用函数: `function_name()`

```
sum_up(10)
```

```
greet_user()
```

Python中不允许在函数未声明之前, 对其进行调用

(2) 关键字参数

```
sum_to_10 = sum_up(n=10) # 关键字参数  
print(sum_to_10)
```

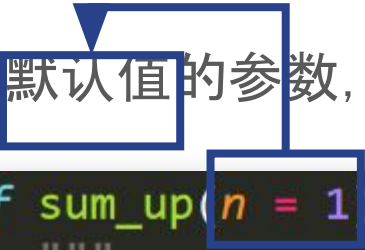
让调用者通过参数名字来区分参数, 允许参数**不按顺序**。



调用函数

(3) 默认参数:

默认参数是声明了默认值的参数, 允许调用者不向该参数传入值。



```
def sum_up(n = 1):  
    """  
    传入一个正整数 n, 默认为 1  
    求 1+2+...+n 的累加值  
    """  
    result = 0  
    for i in range(n):  
        result += i  
    return result
```



调用函数

(4) 参数组

把元组(非关键字参数)或字典(关键字参数)作为**参数组**传给函数。

```
func(positional_args, keyword_args, *tuple_grp_nonkw_args, **dict_grp_kw_args)
```

- 单个星号代表这个位置接收任意多个非关键字参数, 在函数的*b位置上将其转化成元组
- 双星号代表这个位置接收任意多个关键字参数, 在**b位置上将其转化成字典



* 传递函数

Python中函数就像其他对象，函数可以被引用(访问或者以其它变量作为其别名)，也可以作为参数传入函数，以及可以作为列表和字典等容器对象的元素。函数同其他对象区别开来的特征是函数可以调用。

```
def func_a(func, *args, **kwargs):  
    print(func(*args, **kwargs))  
  
def func_b(*args):  
    return args  
  
if __name__ == '__main__':  
    func_a(func_b, 1, 2, 3)
```




变量作用域

搜索标识符的顺序: 先从局部作用域开始, 再到全局域。未找到, 抛出NameError异常。

```
name = 'Tim' #全局变量

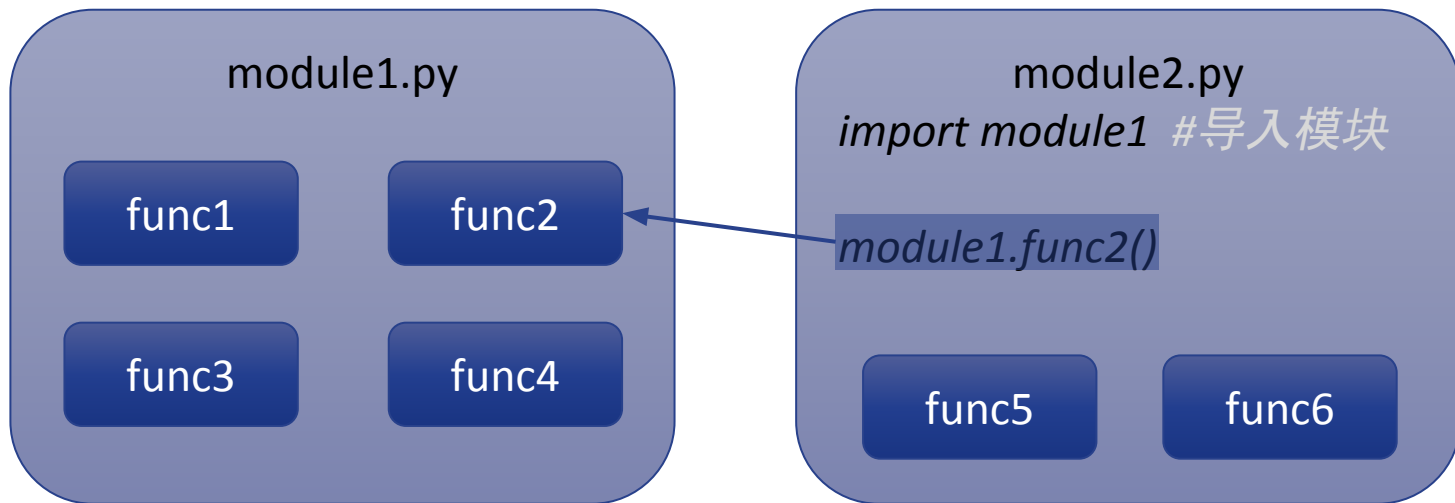
def f1():
    age = 18 #局部变量
    print(age, name)

def f2():
    age = 19 #局部变量
    print(age, name)

f1()
f2()
>>>
18 Tim
19 Tim
```



进一步抽象——模块 Module



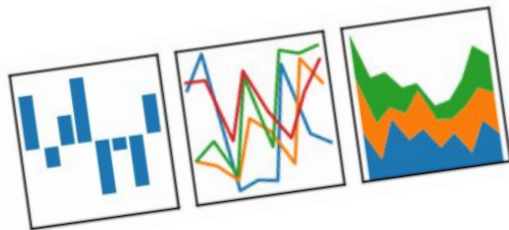


模块、包 Package 或库 Library



pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$



matplotlib



查询文档、搜索示例

- [Python 标准库官方文档](#)
- [pandas 官方文档](#)
- Python3 教程：[菜鸟教程](#)、[廖雪峰](#)
- pandas 教程：[《利用 Python 进行数据分析》](#)

NumPy 和 pandas 基础操作

04





NumPy 向量运算

数组与数组	数组与标量																				
<table><tr><td>1</td><td rowspan="3">×</td><td>4</td><td rowspan="3">=</td><td>4</td></tr><tr><td>2</td><td>5</td><td>10</td></tr><tr><td>3</td><td>6</td><td>18</td></tr></table>	1	×	4	=	4	2	5	10	3	6	18	<table><tr><td>1</td><td rowspan="3">+</td><td rowspan="3">5</td><td rowspan="3">=</td><td>6</td></tr><tr><td>2</td><td>7</td></tr><tr><td>3</td><td>8</td></tr></table>	1	+	5	=	6	2	7	3	8
1	×		4		=	4															
2			5			10															
3		6	18																		
1	+	5	=	6																	
2				7																	
3				8																	
<table><tr><td>a</td><td rowspan="3">+</td><td>d</td><td rowspan="3">=</td><td>ad</td></tr><tr><td>b</td><td>e</td><td>be</td></tr><tr><td>c</td><td>f</td><td>cf</td></tr></table>	a	+	d	=	ad	b	e	be	c	f	cf	<table><tr><td>a</td><td rowspan="3">×</td><td rowspan="3">3</td><td rowspan="3">=</td><td>aaa</td></tr><tr><td>b</td><td>bbb</td></tr><tr><td>c</td><td>ccc</td></tr></table>	a	×	3	=	aaa	b	bbb	c	ccc
a	+		d		=	ad															
b			e			be															
c		f	cf																		
a	×	3	=	aaa																	
b				bbb																	
c				ccc																	



切片和索引

```
arr = np.array([1, 2, 3, 5, 8])
```

索引: `a = arr[0]`

切片: `arr_slice = arr[2:4]`

arr

index	0	1	2	3	4
value	1	2	3	5	8

└─┬─┘└─┬─┬─┬─┘
↑↑
aarr_slice



布尔值索引

```
In [11]: names = np.array(['Bob', 'Joe', 'Will', 'Bob', 'Joe'])  
        scores = np.array([64, 85, 76, 95, 76])
```

```
In [12]: names == 'Bob' # 生成一个布尔型数组，可用于数组索引
```

```
Out[12]: array([ True, False, False,  True, False])
```

```
In [13]: names[names == 'Bob']
```

```
Out[13]: array(['Bob', 'Bob'], dtype='<U4')
```

```
In [14]: scores[names == 'Bob']
```

```
Out[14]: array([64, 95])
```




pandas 数据结构 —— Series

Series 是一种类似于一维数组的对象，它由一组数据(各种NumPy数据类型)以及一组与之相关的数据标签(即索引)组成。

```
In [11]: obj = pd.Series([4, 7, -5, 3])
```

```
In [12]: obj
```

```
Out[12]:
```

```
0    4
```

```
1    7
```

```
2   -5
```

```
3    3
```

```
dtype: int64
```

```
In [13]: obj.values
```

```
Out[13]: array([ 4,  7, -5,  3])
```

```
In [14]: obj.index # like range(4)
```

```
Out[14]: RangeIndex(start=0, stop=4, step=1)
```



pandas 数据结构 —— DataFrame

DataFrame 是一个表格型的数据结构，它含有一组有序的列，每列可以是不同的值类型(数值、字符串、布尔值等)。DataFrame 既有行索引也有列索引，它可以被看作由 Series 组成的字典(共用同一个索引)。

```
data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada', 'Nevada'],  
        'year': [2000, 2001, 2002, 2001, 2002, 2003],  
        'pop': [1.5, 1.7, 3.6, 2.4, 2.9, 3.2]}  
frame = pd.DataFrame(data)
```

In [45]: frame

Out[45]:

	pop	state	year
0	1.5	Ohio	2000
1	1.7	Ohio	2001
2	3.6	Ohio	2002
3	2.4	Nevada	2001
4	2.9	Nevada	2002
5	3.2	Nevada	2003



索引、选取和过滤——Series

```
In [117]: obj = pd.Series(np.arange(4.), index=['a', 'b', 'c', 'd'])
```

```
In [118]: obj
```

```
Out[118]:
```

```
a    0.0
```

```
b    1.0
```

```
c    2.0
```

```
d    3.0
```

```
dtype: float64
```

```
In [119]: obj['b']
```

```
Out[119]: 1.0
```

```
In [120]: obj[1]
```

```
Out[120]: 1.0
```

```
In [121]: obj[2:4]
```

```
Out[121]:
```

```
c    2.0
```

```
d    3.0
```

```
dtype: float64
```

```
In [125]: obj['b':'c']
```

```
Out[125]:
```

```
b    1.0
```

```
c    2.0
```

```
dtype: float64
```



索引、选取和过滤——DataFrame

```
In [128]: data = pd.DataFrame(np.arange(16).reshape((4, 4)),  
.....:                        index=['Ohio', 'Colorado', 'Utah', 'New York'],  
.....:                        columns=['one', 'two', 'three', 'four'])
```

```
In [129]: data
```

```
Out[129]:
```

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

```
In [130]: data['two']
```

```
Out[130]:
```

Ohio	1
Colorado	5
Utah	9
New York	13

Name: two, dtype: int64

```
In [133]: data[data['three'] > 5]
```

```
Out[133]:
```

	one	two	three	four
Colorado	4	5	6	7
Utah	8	9	10	11
New York	12	13	14	15



汇总和计算描述统计

```
In [230]: df = pd.DataFrame([[1.4, np.nan], [7.1, -4.5],  
.....:                      [np.nan, np.nan], [0.75, -1.3]],  
.....:                      index=['a', 'b', 'c', 'd'],  
.....:                      columns=['one', 'two'])
```

```
In [231]: df
```

```
Out[231]:
```

	one	two
a	1.40	NaN
b	7.10	-4.5
c	NaN	NaN
d	0.75	-1.3

```
In [232]: df.sum()
```

```
Out[232]:
```

one	9.25
two	-5.80

dtype: float64

```
In [235]: df.idxmax()
```

```
Out[235]:
```

one	b
two	d

dtype: object



汇总统计方法

方法	说明
describe	针对series或各dataframe列计算汇总统计
min, max	计算最小值和最大值
idxmin, idxmax	计算能够获取到最小值和最大值的索引值
sum	值的总和
mean	值的平均数
value_counts	计算一个Series中各值出现的频率, 按值频率降序排列

项目导览 05





项目介绍

在过去十年内，自行车共享系统的数量不断增多，并且在全球多个城市内越来越受欢迎。自行车共享系统使用户能够按照一定的金额在短时间内租赁自行车。用户可以在 A 处借自行车，并在 B 处还车，或者他们只是想骑一下，也可以在同一地点还车。每辆自行车每天可以供多位用户使用。





项目介绍



在此项目中，你将使用 Motivate 提供的探索自行车共享使用模式，Motivate 是一家入驻美国很多大型城市的自行车共享系统。你将比较以下三座城市的系统使用情况：芝加哥、纽约市和华盛顿特区。



项目介绍

我们提供了三座城市 2017 年上半年的数据。

三个数据文件都包含相同的核心**六列**(英文粗体是列名):

- 起始时间 **Start Time**(例如 2017-01-01 00:07:57)
- 结束时间 **End Time**(例如 2017-01-01 00:20:53)
- 骑行时长 **Trip Duration**(例如 776 秒)
- 起始车站 **Start Station**(例如百老汇街和巴里大道)
- 结束车站 **End Station**(例如塞奇威克街和北大道)
- 用户类型 **User Type**(会员 Subscriber/Registered 或散客 Customer/Casual)





项目介绍

芝加哥和纽约市文件的数据多了两列：

- 性别 **Gender**
- 出生年份 **Birth Year**

	Start Time	End Time	Trip Duration	Start Station	End Station	User Type	Gender	Birth Year
0	2017-01-01 00:00:21	2017-01-01 00:11:41	680	W 82 St & Central Park West	Central Park West & W 72 St	Subscriber	Female	1965.0
1	2017-01-01 00:00:45	2017-01-01 00:22:08	1282	Cooper Square & E 7 St	Broadway & W 32 St	Subscriber	Female	1987.0
2	2017-01-01 00:00:57	2017-01-01 00:11:46	648	5 Ave & E 78 St	3 Ave & E 71 St	Customer	NaN	NaN
3	2017-01-01 00:01:10	2017-01-01 00:11:42	631	5 Ave & E 78 St	3 Ave & E 71 St	Customer	NaN	NaN
4	2017-01-01 00:01:25	2017-01-01 00:11:47	621	5 Ave & E 78 St	3 Ave & E 71 St	Customer	NaN	NaN
5	2017-01-01 00:01:51	2017-01-01 00:12:57	666	Central Park West & W 68 St	Central Park West & W 68 St	Subscriber	Male	2000.0
6	2017-01-01 00:05:00	2017-01-01 00:14:20	559	Broadway & W 60 St	9 Ave & W 45 St	Subscriber	Male	1973.0
7	2017-01-01 00:05:37	2017-01-01 00:19:24	826	Broadway & W 37 St	E 10 St & Avenue A	Subscriber	Female	1977.0
8	2017-01-01 00:05:47	2017-01-01 00:10:02	255	York St & Jay St	Carlton Ave & Flushing Ave	Subscriber	Male	1989.0
9	2017-01-01 00:07:34	2017-01-01 00:18:08	634	Central Park West & W 72 St	Columbus Ave & W 72 St	Subscriber	Male	1980.0

Data for the first 10 rides in the `new_york_city.csv` file



项目需要实现的统计计算

你将编写代码并回答以下关于自行车共享数据的问题：

- 起始时间(Start Time 列)中**哪个月份**最常见？
- 起始时间中，**周几**最常见？ 提示：可以使用 `datetime.weekday()`
- 起始时间中，一天当中**哪个小时**最常见？
- **总骑行时长**(Trip Duration)是多久，平均骑行时长是多久？
- 哪个起始车站(Start Station)**最热门**，哪个结束车站(End Station)最热门？
- 哪一趟行程最热门(即，哪一个起始站点与结束站点的组合最热门)？
- 每种**用户类型(User Type)**有多少人？
- 每种性别(Gender)有多少人？
- 出生年份(Birth Year)最早的是哪一年、最晚的是哪一年，最常見的是哪一年？



项目选修练习的目的：

- 理解数据结构，观察各列的列名、取值和各个值的频率等信息
- 了解如何对数据集 DataFrame 进行筛选
- 对某些需要统计分析的列使用统计函数：
 - value_counts
 - mode
 - min/max



项目相关知识:字典

字典是另一种可变容器模型, 且可存储任意类型对象。

字典的每个键值 key=>value 对用冒号 : 分割, 每个键值对之间用逗号 , 分割, 整个字典包括在花括号 {} 中 ,格式如下所示:

```
CITY_DATA = {'chicago': 'chicago.csv',
              'new york city': 'new_york_city.csv',
              'washington': 'washington.csv'}

print(CITY_DATA['chicago']) # 打印结果是 chicago.csv
```



项目相关知识:read_csv

pandas 提供了一些用于将表格型数据读取为 DataFrame 对象的函数。

其中 read_csv 是从文件、URL、文件型对象中加载带分隔符的数据，默认分隔符是逗号。

* 可以用于读取项目文件

```
In [9]: df = pd.read_csv('examples/ex1.csv')
```

```
In [10]: df
```

```
Out[10]:
```

	a	b	c	d	message
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo



项目相关知识: 用 while 循环获取有效输入

Python3.x 中 `input()` 函数接受用户输入内容, 返回为 `string` 类型。

输入错误的值, 会导致后续程序报错, 需要控制有效输入。

思考如何用代码实现:

- 如果用户输入错误, 则一直循环 `input` 语句请求用户输入;
- 输入正确则跳出循环。



项目相关知识: mode

沿着某个选择的轴返回(一组)**众数**。每一列的众数都是一个 Series, 即使这一列只有一个众数。每个众数都会增加一行和一个label, 对缺失行用nan填充。如果只需要某一系列的一个众数, 则可以使用 `df['A'].mode()[0]`

```
In [15]: df = pd.DataFrame({'A': [1, 2, 1, 2, 1, 2, 3]})  
        df.mode()
```

```
Out[15]:
```

	A
0	1
1	2

```
In [16]: df['A'].mode()
```

```
Out[16]:
```

0	1
1	2

dtype: int64

```
In [17]: df['A'].mode()[0]
```

```
Out[17]: 1
```

Acknowledgement

感谢以下资深助教对本辅导资料的贡献

张一凡、李伟伟



优达学城
UDACITY

