

# Python 入门公开课 讲义



如果有更新迭代的建议，请发送邮件至 [kylie@udacity.com](mailto:kylie@udacity.com) 并抄送 [april.chen@udacity.com](mailto:april.chen@udacity.com)。谢谢。

## 0 使用说明

本讲义内含五部分：

1. 基本语言操作
2. 脚本运行与文件操作
3. 函数和模块
4. NumPy 和 Pandas
5. 项目导览

本课程+项目总时长为4周。讲义内的内容建议分三个公开课时讲解完毕，为同学必须要了解的知识内容，最后的一节公开课助教可以根据同学的需求/常见问题自由控制主题。

公开课和讲义中包含 \* 为拓展内容，不是必须讲解的部分。

建议时间安排：

第一课时：基本语言操作 + 文件路径存储路径

第二课时：脚本运行、异常处理 + 函数和模块

第三课时：NumPy 和 Pandas + 项目导览

## 1 基本语言操作

此处的基本语言操作，并不涉及到很多基础语法的讲解，只是一些容易出错的 Python 基础知识点。

### 1.1 缩进

#### 1.1.1 介绍

Python 是通过空白符（制表符 Tab 或空格）来组织代码的，不像其他语言（如 R、C++、Java 等）用的是大括号。以 if-else 语句为例：

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

冒号表示一段缩进代码块的开始，其后的所有代码都必须缩进相同的量，直到代码块结束为止。

使用空白符的主要好处是，它能使大部分 Python 代码在外观上看起来差不多。也就是说，当你阅读某段不是自己编写的（或是一年前匆忙编写的）代码时，不怎么样容易出现“认知失调”。在那些空白符无实际意义的语言中，你可能会发现一些格式不统一的代码，毕竟只要有花括号，程序就可以识别到哪部分才是 if 语句条件达成时需要执行的所有代码。

### 1.1.2 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。

### 1.1.3 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。

以 Sublime 为例：

```
{  
    "draw_white_space": "all",  
    "tab_size": 4,  
    "translate_tabs_to_spaces": true,  
}
```

## 1.2 关于代码行

### 1.2.1 多行语句

Python 语句中一般以换行作为语句的结束符。如果一行语句太长，我们也可以使用一些连接方式将一行代码换行来写。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（出现这样的多行语句时，要将连在一起的多行代码看作一行代码，以第一行的缩进层级为准）

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

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

### 1.2.2 行长度规范

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')
```

### 1.2.3 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。

### 1.2.4 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，有助于提高代码可读性，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。

## 1.3 注释

### 1.3.1 注释——动词

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 + /

### 1.3.2 注释——名词

代码中的注释文字，通常用来解释代码的功能和作用。

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
```

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
"""
```

## 1.4 引号

### 1.4.1 介绍

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，在文件的特定位置，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"""
```

### 1.4.2 易错点

- 开始的引号与结束的引号不统一；
- 引号括起来的字符串中包含引号；

```
# 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!'
```

## 1.5 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。

### 1.5.1 使用示例

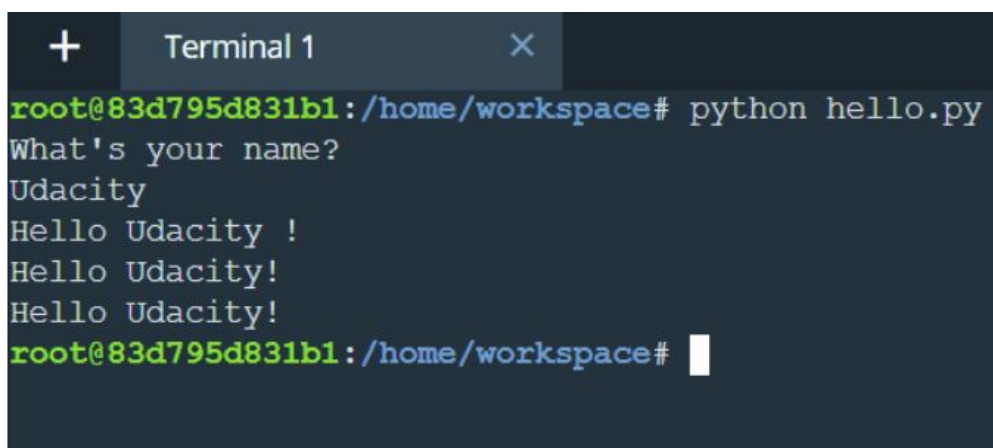
```
# 获取用户输入并赋值给 name 变量
name = input("What's your name?\n")

# 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))
```

### 1.5.2 运行效果



```
+ Terminal 1 X
root@83d795d831b1:/home/workspace# python hello.py
What's your name?
Udacity
Hello Udacity !
Hello Udacity!
Hello Udacity!
root@83d795d831b1:/home/workspace#
```

## 2 脚本运行与文件操作

这一小节主要目的是帮助同学完成本地的脚本运行操作。

### 2.1 使用命令行导航

#### 2.1.1 Mac 终端

更改当前工作目录：cd 目标路径，比如修改到桌面的 python 文件夹：

```
cd desktop/python
```

cd 是 change directory 的缩写。

列出当前目录中的所有文件和文件夹：`ls`

ls 是 list 的缩写。

#### 2.1.2 Windows 命令提示符

更改当前工作目录：cd 目标路径，比如修改到桌面的 python 文件夹：

```
cd desktop/python
```

cd 是 change directory 的缩写。

列出当前目录中的所有文件和文件夹：`dir`

dir 是 directory 的缩写。

Windows 与 Mac 还有一处不同：Windows 存在不同的盘符，比如默认文件夹是 C 盘，但是我们还会经常用到 D 盘或者 E 盘等。如果从 C 盘的某个文件夹，直接 `cd d:/python` 是没办法修改到这个目录的，需要再输入一次 `d:` 更改盘符。所以从 C 盘更改到 D 盘，需要两步操作（这两步的顺序没有强制要求）：

```
> d:
> cd d:/python
```

### 2.2 使用 Python 操作导航\*

使用 os 包 `import os`

获得当前工作目录：`print(os.getcwd())`

当前工作目录下的所有文件和文件夹：`print(os.listdir())`

更改当前目录：`os.chdir("d:/python")`

### 2.3 运行脚本

当前目录下运行脚本，先修改目录到脚本文件所在目录 path，然后运行脚本

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

直接将路径与文件名一起输入，运行其他目录下的脚本（终端可以实现一种操作，是直接将 python 脚本文件拖动到终端中运行，就是这样的效果）

```
$ 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。

## 2.4 绝对路径和相对路径

```
open('test1.txt') #1
open('/temp/test2.txt') #2
open('D:\\user\\test3.txt') #3
open(r'D:\user\temp\test4.txt') #4
```

#1 和 #2 为相对路径，#3 和 #4 为绝对路径。

相对路径相对的是当前工作目录，也就是终端和命令提示符所运行的当前工作目录，并不一定是 python 脚本所在目录。

关于斜线，Windows 比较特立独行地使用了 \ 作为路径目录之间的分隔线，但是 \ 本身在字符串中，还被作为转义字符使用，是有特殊用途的字符。要使用字符本身，而不是字符的特殊效果，需要用转义字符转义。比如在字符串中有引号时，可以使用转义字符，将引号识别为普通字符，而不是用来闭合字符串的特殊字符：'That\'s right'。转义字符同样可以转义它自己，所以 \\ 就相当于正常的 \ 字符串。

在字符串前面添加一个 r 也可以实现类似的功能。r 代表了 raw，表示这个字符串都识别为原始字符串而不是特殊含义。

## 2.5 读写文件\*

### 2.5.1 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
```

对于可迭代对象，可以使用 for 循环迭代：

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

```
f.close()
```

时间能打开的文件数量也是有限的。

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()
```

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())
```

### 2.5.2 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 `append`。

我们可以反复调用 `write()` 来写入文件，但是务必要调用 `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!')
```

### 2.5.3 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'
```

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')
```

## 2.6 异常处理

### 2.6.1 介绍

Python 使用异常对象（Exception Object）来管理程序执行期间发生的错误。每当发生让 Python 不知所措的错误时，他都会创建一个异常对象。如果你编写了处理该异常的代码，程序将继续运行；如果你未对异常进行处理，程序将停止，并显示一个 `Traceback` 回溯，其中包含有关异常的报告。

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

当你认为可能发生了错误时，可编写一个 `try-except` 代码块来处理可能引发的异常。让 Python 尝试（`try`）运行一些代码，并告诉它如果这些代码引发了指定的异常（`except` 异常名字），则执行另外一段代码。

```
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'
```

### 2.6.2 常见异常

异常名字	描述
<code>FileNotFoundError</code>	路径下找不到文件
<code>KeyError</code>	映射中（字典、 <code>DataFrame</code> 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

## 3 函数和模块

函数是 Python 中最主要也是最重要的代码组织和复用手段。这一小节主要讲解函数的概念、如何定义函数、如何调用函数等。

### 3.1 什么是函数

函数是带名字的代码块，用于完成具体的工作。要执行函数定义的特定任务，可调用该函数。需要在程序中多次执行同一项任务时，你无需反复编写完成该任务的代码，而只需要调用执行该任务的函数，让 Python 运行函数中的代码。使用函数，程序的编写、阅读、测试和修复都更加容易操作。



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。

#### 3.1.1 函数示例

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)
```

## 3.2 创建函数

### 3.2.1 def 关键字

函数定义的第一行，使用 def 关键字来告诉 Python 你要定义一个函数。

`def greet_user():` 向 Python 指明了函数名，并在括号中指出函数为完成其任务需要什么参数（即便不需要参数，括号也必不可少）。

最后，定义以冒号结束。

### 3.2.2 函数体

紧跟在 `def greet_user():` 后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!。

## 3.3 调用函数

要调用函数，可以指定函数名以及用括号括起必要的参数，如 `sum_up(10)` 和 `greet_user()`。

Python中不允许在函数未定义之前，对其进行调用。

### 3.3.1 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。

### 3.3.2 关键字参数

让调用者通过参数名字来区分参数，允许参数不按顺序。如果没有指定参数名字，则属于位置实参，传入实参的顺序需要与形参的顺序相同。

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

### 3.3.3 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
```

### 3.3.4 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

### 3.3.5 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)
```

## 3.4 变量作用域

到底什么是变量？你可以把变量看作是值的名字。在执行 `x=1` 赋值语句之后，名称 `x` 引用到值 `1`。这就象个字典一样，键引用值，当然，变量和所对应的值用的是个“不可见”的字典。实际上这么说已经很接近真实情况了。内建的 `vars` 函数可以返回这个字典：

```
>>> x=1
>>> scope = vars()
>>> scope['x']
1
```

这类“不可见字典”叫做命名空间或者作用域。除了全局作用域外，每个函数调用都会创建一个新的作用域。

Python 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
```

## 3.5 模块、包 Package 和库 Library

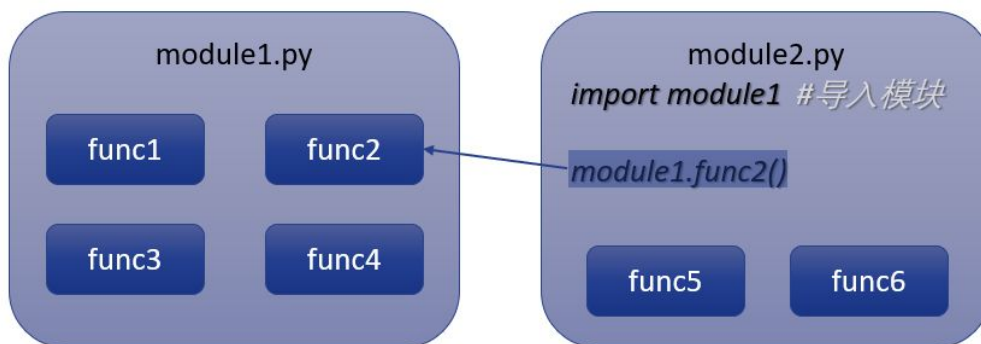
### 3.5.1 进一步抽象——模块

函数的优点之一是，使用它们可以将代码块抽象出来，与主程序分离。通过给函数指定描述性名称，可以让主程序更加容易理解。我们可以再进一步抽象，将函数储存在一个独立的文件中，称为**模块 (module)**。使用 `import` 语句，将模块导入主程序，这样即可在当前程序中使用模块中的代码。

导入操作，只需要 `import module_name` 即可。

导入之后，脚本文件 `module_name.py` 中的所有函数都可以在新的文件中被调用。

如图所示，`import module1` 之后，`module2` 中可以调用 `module1` 中的函数。也可以导入一些 python 内置的模块：比如 `import math`



### 3.5.2 数据分析常用的库

Python 标准库中包含了很多常用的模块、内置函数或方法。除此之外，NumPy、pandas 和 matplotlib 是数据分析经常用到的库。

约定俗成的导入方式和别名：

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

### 3.5.3 查询文档，搜索代码示例

搜索 `python/python3/pandas + 函数名` 即可找到对应的文档和教程

- [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》](#)

## 4 NumPy 和 Pandas 基础操作

NumPy 和 Pandas 初步接触，理解向量化的数据结构和操作方式，初步进行一些统计计算。

### 4.1 NumPy 向量运算



数组与数组			数组与标量		
1 2 3	×	4 5 6	=	4 10 18	
a b c	+	d e f	=	ad be cf	
1 2 3	+	5	=	6 7 8	
a b c	×	3	=	aaa bbb ccc	

向量与向量之间的运算，按位计算。与标量的运算，按位依次计算。

## 4.2 切片和索引

### 4.2.1 示例

```
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

{  
↑  
a

{  
↑  
arr\_slice

数组切片返回的都是视图（view），如果要复制副本（copy），可以使用 `arr[5:8].copy()`

### 4.2.2 布尔值索引示例

```
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])
```

`names == 'Bob'` 生成一个布尔型数组，可用于数组索引。

## 4.3 Pandas 数据结构

pandas 是基于 NumPy 构建的，让以 NumPy 为中心的应用变得更加简单。

### 4.3.1 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。

数据标签默认为 0到N-1的整数型索引，也可以设定特定的索引标记。通过 Series 的 `values` 和 `index` 属性获取其数组表示形式和索引对象。

NumPy 数组运算（如根据布尔型数组进行过滤、标量乘法、应用数学函数等）都会保留索引和值之间的链接。

```
In [11]: obj = pd.Series([4, 7, -5, 3])
Out[11]: 0    4
         1    7
         2   -5
         3    3
         dtype: int64

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)
```

### 4.3.2 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），其中面向行和面向列的操作基本上是平衡的。

DataFrame 以二维结构保存数据（而不是列表、字典或别的一维数据结构）。

```
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

### 4.3.3 索引、选取和过滤

Series 索引 (obj[...]) 的工作方式类似于 NumPy 数组的索引，只不过 Series 的索引值不只是整数，还有标签。利用标签的切片运算与普通的 Python 切片运算不同，其末端是包含的：

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

In [118]: obj	In [119]: obj['b']	In [121]: obj[2:4]	In [125]: obj['b':'c']
Out[118]:	Out[119]: 1.0	Out[121]:	Out[125]:
a 0.0		c 2.0	b 1.0
b 1.0	In [120]: obj[1]	d 3.0	c 2.0
c 2.0	Out[120]: 1.0	dtype: float64	dtype: float64
d 3.0			
dtype: float64			

用一个值或序列对 DataFrame 进行索引其实就是获取一个或多个列，可以使用布尔型 DataFrame 进行数据筛选操作，与 NumPy 数组的语法很像。

```
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	In [130]: data['two']	In [133]: data[data['three'] > 5]
Out[129]:	Out[130]:	Out[133]:
	Ohio 1	
Ohio 0 1 2 3	Colorado 5	Colorado 4 5 6 7
Colorado 4 5 6 7	Utah 9	Utah 8 9 10 11
Utah 8 9 10 11	New York 13	New York 12 13 14 15
New York 12 13 14 15	Name: two, dtype: int64	

### 4.3.4 汇总和计算描述统计

pandas 对象拥有一组常用的数学和统计方法。它们大部分都属于约简和汇总统计，用于从 Series 中提取单个值（如 sum 或 mean）或从 DataFrame 的行或列中提取一个 Series。跟对应的 NumPy 数组方法相比，它们都是基于没有缺失数据的假设而构建的。

```
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
```

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

## 5 项目导览

对项目进行简要介绍，理解项目所需要实现的功能。

### 5.1 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。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。

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 (6) 列：

- 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）

芝加哥和纽约市文件（chicago.csv 和 new\_york\_city.csv）还包含以下两列：

- 性别 **Gender**
- 出生年份 **Birth Year**

数据预览， 以下是纽约市的前 10 条数据：

	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*

## 5.2 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）中哪个月份最常见？
- 起始时间（Start Time）中， 周几最常见？
- 起始时间（Start Time）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， 最常见的是哪一年？

## 5.3 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

## 5.4 项目相关知识

### 5.4.1 字典

字典是另一种可变容器模型， 且可存储任意类型对象。

字典的每个键和值之间用冒号：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
```

### 5.4.2 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。

```
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

### 5.4.3 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，所以需要控制有效输入的范围。

思考如何用代码实现如下功能：

- ✓ 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。

### 5.4.4 mode

项目涉及到最多的计算是众数计算。pandas 中使用 mode 来计算众数。但是与普通的 sum mean 等函数不同，众数可能不只一个。

mode 函数，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，它也会以 Series 的类型返回。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
```

## 感谢以下资深助教对本辅导资料的贡献

李伟伟 (lvy)

本资料由Udacity（中国）官方审核发布，最终解释权归Udacity（中国）所有。