Python 数据分析基础

学习笔记

曾　建

2018.12.25

目录

1. Python 基础知识 1
1.1. Python 简介 1
1.2. 解释型与编绎型 1
1.3. 搭建环境 2
1.4. 环境变量 Path 2
1.5. IDE 3
1.6. Pycharm 安装 3
1.7. 语法特点 4
1.7.1. 注释 4
1.7.2. 缩进 5
1.7.3. 语句断行 5
1.7.4. 编码规范 6
1.7.5. 命名规范 6
1.7.6. 保留字与标识符 6
1.8. 基本数据类型 8
1.8.1. 数字 8
1.8.2. 字符串 8
1.8.3. 布尔类型 10
1.8.4. 类型转换 11
1.8.5. 基本输入输出 11
1.9. 运算符 12
1.9.1. 算术运算符 12
1.9.2. 赋值运算符 13
1.9.3. 关系运算符 13
1.9.4. 逻辑运算符 14
1.9.5. 位运算符 14
1.9.6. 优先级 15
1.10. 条件表达式 15
2. 流程控制 16
2.1. 程序结构 16
2.2. 选择语句 16
2.3. 循环语句 18
2.3.1. while 循环 18
2.3.2. for 循环 18
2.3.3. 循环嵌套 19
2.3.4. 结束循环 20
3. 数据结构 21
3.1. 序列 21
3.1.1. 索引 21
3.1.2. 切片 22
3.1.3. 相加 22

3.1.4. 相乘.....22
3.1.5. 成员检查.....22
3.1.6. 计算长度和最大最小值.....22
3.2. 列表.....23
3.2.1. 创建列表.....23
3.2.2. 遍历列表.....24
3.2.3. 更新列表.....24
3.2.4. 列表统计计算.....25
3.2.5. 列表排序.....25
3.2.6. 列表推导式.....26
3.2.7. 列表的常用函数.....27
3.2.8. 二维列表.....27
3.3. 元组.....28
3.3.1. 元组与列表的区别.....28
3.3.2. 创建元组.....28
3.3.3. 访问元组.....29
3.3.4. 修改元组.....29
3.3.5. 元组推导式.....30
3.4. 字典.....30
3.4.1. 创建字典.....30
3.4.2. 访问字典.....32
3.4.3. 更新字典.....32
3.4.4. 字典推导式.....33
3.5. 集合.....33
3.5.1. 创建集合.....33
3.5.2. 添加删除元素.....34
3.5.3. 集合运算.....34
3.6. 字符串.....35
3.6.1. 拼接字符串.....35
3.6.2. 计算字符串长度.....35
3.6.3. 截取字符串.....35
3.6.4. 检索字符串.....35
3.6.5. 大小写转换.....36
3.6.6. 格式化字符串.....36
4. 函数.....37
4.1. 定义函数.....37
4.2. 调用函数.....38
4.3. 参数传递.....38
4.4. 返回值.....40
4.5. 变量的作用域.....40
5. 面向对象.....41
5.1. 对象与类.....41
5.2. 面向对象的特点.....41
5.2.1. 封装.....41

5.2.2. 继承.....41
5.2.3. 多态.....42
5.3. 类的定义.....42
5.4. 创建实例.....42
5.4.1. __init__()方法.....42
5.4.2. 实例方法.....43
5.4.3. 数据成员.....43
5.5. 访问限制.....43
5.6. 属性.....44
5.7. 继承.....44
5.8. 重写.....44
6. 模块.....45
6.1. 自定义模块.....45
6.2. 模块搜索目录.....45
6.3. 常用标准模块.....46
6.4. 第三方模块.....46
6.5. 包.....46
7. 异常处理.....47
8. 多维数组.....48
8.1. Numpy 模块.....48
8.2. 数组创建.....48
8.3. 数组生成函数.....50
8.4. 数据类型.....51
8.5. 数组访问.....52
8.6. 数组属性.....53
8.7. 数组形状.....54
8.8. 排序.....56
8.9. 基本运算.....56
8.9.1. 四则运算.....57
8.9.2. 比较运算.....57
8.9.3. 广播运算.....58
8.10. 常用函数.....58
8.10.1. 数学函数.....58
8.10.2. 统计函数.....59
8.11. 线性代数.....60
8.12. 随机模块.....60
9. 数据处理.....60
9.1. Pandas.....60
9.2. 序列.....61
9.2.1. 构建.....61
9.2.2. 访问.....63
9.2.3. 追加.....64
9.2.4. 删除.....64
9.2.5. 更新.....64

9.2.6. 排序.....65
9.3. 数据框.....65
9.3.1. 构建.....65
9.3.2. 访问.....66
9.3.3. 增加.....67
9.3.4. 删除.....68
9.3.5. 更新.....69
9.3.6. 显示数据.....70
9.4. 基本操作.....70
9.4.1. 重建索引.....70
9.5. 数据导入.....72
9.5.1. 文本数据.....72
9.5.2. 电子表格数据.....73
9.5.3. 数据库数据.....73
9.6. 数据导出.....73
9.7. 类型转换.....75
9.8. 数据清洗.....75
9.8.1. 重复值处理.....76
9.8.2. 缺失值处理.....76
9.8.3. 异常值处理.....78
9.9. 数据抽取.....79
9.9.1. 字段抽取.....79
9.9.2. 字段拆分.....79
9.9.3. 重置索引.....80
9.9.4. 条件抽取数据.....80
9.9.5. 索引抽取数据.....80
9.10. 数据修改.....81
9.11. 重置索引.....82
9.12. 透视表.....82
9.13. 合并连接.....84
9.14. 分组聚合.....85
10. 数据基本分析.....86
10.1. 基本统计分析.....86
10.2. 分组分析.....87
10.3. 分布分析.....88
10.4. 交叉分析.....90
10.5. 结构分析.....91
10.6. 相关分析.....93
11. 数据可视化.....94
11.1. 离散型变量.....94
11.1.1. 饼图.....94
11.1.2. 条形图.....97
11.2. 数值型变量.....100
11.2.1. 直方图.....100

11.2.2. 核密度图.....103
11.2.3. 箱线图.....105
11.2.4. 小提琴图.....105
11.2.5. 折线图.....105
11.3. 关系型数据.....107
11.3.1. 散点图.....108
11.3.2. 气泡图.....111
11.3.3. 热力图.....112
12. 上机作业题.....114
12.1. 数值交换.....114
12.2. 三数比较大小.....115
12.3. 计算人体健康 BMI.....115
12.4. 计算闰年.....116
12.5. 回文.....116
12.6. 九九乘法表.....117
12.7. 自然数求和.....117
12.8. 计算阶乘和.....117
12.9. 鸡兔同笼.....118
12.10. 百人分百饼.....118
12.11. 最大最小值.....118
12.12. 换零钱.....119
12.13. 登录功能.....120
12.14. 修改密码功能.....120
12.15. 质因数分解.....121
12.16. 冒泡排序.....122
12.17. **约瑟夫问题**.....122
12.18. 递归求自然数和.....123
12.19. 兔子数目.....123
12.20. 猜幸运数字.....124
12.21. 押大小游戏.....125
12.22. 五猴分桃.....126
12.23. 打印全年的日历.....127

1. Python 基础知识

1.1. Python 简介

Python 是 1989 年荷兰人 Guido van Rossum 发明的，它是一种面向对象的解释型高级编程语言。它的设计哲学是优雅、明确和简单。实际上，Python 也正是按着这个理念做的，以至于网络上现在流传“人生苦短，我用 Python”的说法，它有简单、开发速度快、节省时间和精力的特点。

Python 本身并非所有的特性和功能都集成到语言的核心，而是被设计成为可扩充的。它具有丰富和强大的库，能够把用其它语言制作的种和模块很轻松地联结在一起。为些，它也被称为胶水语言。

Python 版本主要有三个版本：

1、1994 年，发布 Python 1.0，现在已经过时；

2、2000 年，发布 Python 2.0，现在已经更新到 2.7.X；

3、2008 年，发布 Python 3.0，现在已经更新到 3.7.X；

Python 在版本升级的时候，并不是向下兼容的，它们在基本语法上会存在一些区别，但是它们的编程思想上是相通的，它们的主要区别不在此一一列举。Python 2.X 以后可能会停止更新，所以现在基本上建议直接学 Python 3.X，毕竟是大势所趋。

当然，选择 Python 3.X 也有缺点，就是很多扩展库的发行总是滞后于 Python 的发行版本。甚至目前还有很多库不支持 Python 3.X，所以在实际开发中，一定要考虑到做哪方面的开发，需要用到哪些库，然后再做出选择。

Python 能做什么：

1、Web 开发；

2、**大数据处理**；

3、人工智能；

4、自动化运维开发；

5、云计算；

6、爬虫；

7、游戏开发；

Python 的应用领域远比上面提到的要多得多，例如：图形图象处理、编程控制机器人、数据库编程、编写可移植性的维护操作系统的工具、自然语言分析处理等。

1.2. 解释型与编绎型

计算机不能直接理解**高级语言**，只能直接理解**机器语言**，所以必须要把**高级语言**翻译成**机器语言**，计算机才能执行**高级语言**编写的程序。

翻译的方式有两种，一个是**编译**，一个是**解释**。两种方式只是翻译的时间不同。

编译型语言写的程序执行之前，需要一个专门的编译过程，把程序编译成为机器语言的文件，比如 exe 文件，以后要运行的话就不用重新翻译了，直接使用编译的结果就行了(exe 文件)，因为翻译只做了一次，运行时不需要翻译，所以编译型语言的程序执行效率高，但也不能一概而论，部分解释型语言的解释器通过在运行时动态优化代码，甚至能够使解释型

语言的性能超过编译型语言。

解释性语言的程序不需要编译，省了道工序，解释性语言在运行程序的时候才翻译，比如解释性 basic 语言，专门有一个解释器能够直接执行 basic 程序，每个语句都是执行的时候才翻译。这样解释性语言每执行一次就要翻译一次，效率比较低。解释是一句一句的翻译。

编译型与解释型，两者各有利弊。前者由于程序执行速度快，同等条件下对系统要求较低，因此像开发操作系统、大型应用程序、数据库系统等时都采用它，像 C/C++、Pascal/Object Pascal(Delphi)等都是编译语言，而一些网页脚本、服务器脚本及辅助开发接口这样的对速度要求不高、对不同系统平台间的兼容性有一定要求的程序则通常使用解释性语言，如 Java、JavaScript、VBScript、Perl、Python、Ruby、MATLAB 等等。

但随着硬件的升级和设计思想的变革，编译型和解释型语言越来越笼统，主要体现在一些新兴的高级语言上，而解释型语言的自身特点也使得编译器厂商愿意花费更多成本来优化解释器，解释型语言性能超过编译型语言也是必然的。

1.3. 搭建环境

1、下载 Python 安装包：python-3.7.1-amd64.exe；

2、双击 python-3.7.1-amd64.exe，开始安装时注意：一定要勾选 Add Python 3.7 to Path 选项，然后可以默认安装。

3、测试是否安装成功，打开命令窗口，输入 python 命令；

1.4. 环境变量 Path

安装时注意勾选把 Python 添加到环境变量。

1.5. IDE

1、PyCharm

JetBrains 公司开发，社区和商用两个版本

2、Eclipse

开源的、基于 Java 的扩展平台，需要安装 Pydev 插件

3、Microsoft Visual Studio

微软公司开发，需要安装 PTVS 插件。

1.6. Pycharm 安装

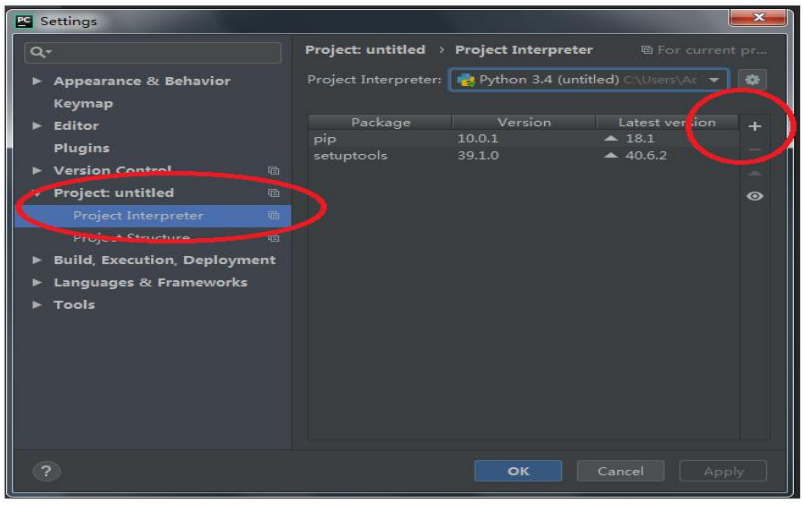
1、双击 pycharm-community-2018.2.4.exe 开始安装，可以默认；

2、IDE 安装好后，打开 PyCharm，随便建个项目打开 IDE。

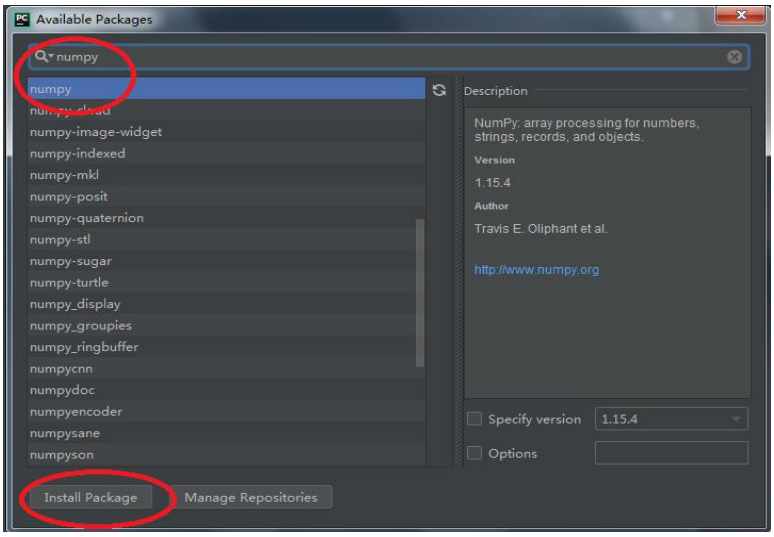
3、然后再依次安装第三方模块：Numpy、Pandas、Matplotlib、Seaborn。这四个模块的安装方法是一样的，如下：

（1）点击【File】→【Setting】，打开 Setting 对话框；

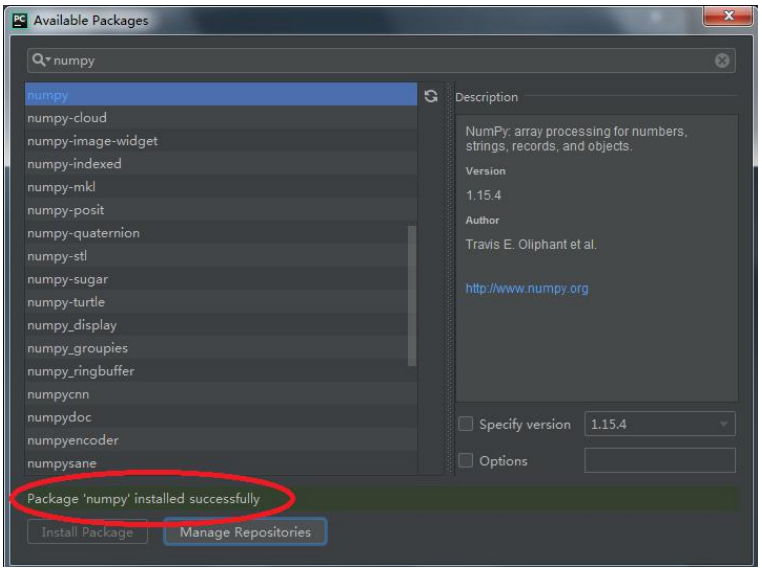
（2）选择【Project:untitled】→【Project:interpreter】



（3）点击右上边的 【+】，打开第三方模块安装对话框；



（4）在搜索框中搜索 numpy,选择点击左下角的【Install Package】开始安装，需要等半分钟，左下角会显示安装成功；



（5）搜索 Pandas 安装；
（6）搜索 Matplotlib 安装；
（7）搜索 Seaborn 安装；

1.7. 语法特点

1.7.1. 注释

所谓注释，就是解释说明此行代码的功能、用途，注释说明部分不被计算计执行。

1、单行注释

在 Python 中，使用“#”作为单行注释的符号，从符号“#”开始直到换行为此，其后面所有的内容都作为注释的内容，它是被 Python 编器忽略的。

语法：# print(“这是注释内容”)

注意：

（1）单行注释可以放在程序语句的上一行，也可以放程序语句的右侧；

（2）在添加注释的时候，一定要有意义，要充分地体现代码的作用。

```
# 这是一个注释
# print(“这条语句被注释掉了，不会执行”)
print(“我是第二条程序语句，我会被执行”)
print(“我是最后一条语句”)        # 执行完这一条语句，程序执行完毕
```

2、多行注释

多行注释用三个单引号 ''' 或者三个双引号 """ 将注释括起来。多行注释通常用来为 Python 文件、模块、类或者函数添加版权、功能等信息。

第一种形式：

```
'''
这是多行注释，用三个单引号
这是多行注释，用三个单引号
这是多行注释，用三个单引号
'''
print("Hello, World!")
```

第二种形式：

```
"""
这是多行注释，用三个双引号
这是多行注释，用三个双引号
这是多行注释，用三个双引号
"""
print("Hello, World!")
```

1.7.2. 缩进

1、Python 不像其它程序设计语言（如 C、C++、C#、Java），用“{}”分隔代码块，表示代码之间的层次，而是采用代码缩进和冒号“：”来区分代码之间的层次。

2、在 Python 中，对于类定义、函数定义、流程控制语句，以及异常处理语句等，行尾的冒号和下一行的缩进表示一个代码块的开始，而缩进结束，表示下一个代码块的开始。

3、Python 对代码的缩进要求非常严格，同一个级别的代码块的缩进量必须相同，如果采用不合理的代码缩进，将会抛出 SyntaxError 异常。例如：代码中有的缩进量是 4 个空格，有的是 3 个空格，就会出现 SyntaxError 异常。

4、在编码过程当中，一般以 4 个空格为一个缩进。

1.7.3. 语句断行

一般来说，Python 一条语句占一行，在每条语句的结尾处不需要使用“;”，但是也可

以使用“；”表示将两行简单语句写在一行。如果一条语句较长要分几行来写，可以使用“\”来换行。

1.7.4. 编码规范

1、每个 import 语句只导入一个模块，尽量避免一次导入多个模块；
2、不要在行尾加分号“；”，不要用分号把两条命令放在同一行；
3、每行尽量不要超过 80 个字母；
4、必要的时候可以增加空行增加代码的可读性；
5、通常情况，运算符两侧、函数参数两侧、逗号两侧，建议使用空格进行分隔；
6、应该避免在循环中使用“+”、“+=”操作符累加字符串。因为字符串是不可变的，这样做会创建很多不必要的临时对象，推荐做法是将每个子字符串加入列表，然后在循环结束后使用 join()方法连接列表。

1.7.5. 命名规范

命名规范在编写代码时有很重要的作用，虽然不遵循命令规范，程序也是可以运行的。但是使用命名规范可以更加直观地了解代码所代表的含义。
1、模块名尽量短小，并且全部使用小写字母，可以使用下划线分隔多个字母，例如：game_main、game_register 等；
2、包名尽量短小，并且全部使用小写字母，不推荐使用下划线。例如：com.mingrisoft、com.mr、com.mr.book。
3、类名采用单词首字母在大写的形式（Pascal 风格），例如，一个借书类，可以命名为：BorrowBook。
4、模块内部类采用下划线“”+Pascal 风格的类名组成。例如，在 BorrowBook 类中的内部类，可以使用_BorrowBook 命名。
5、函数、类的属性和方法的命名同模块类似，全部采用小写字母，多个字母可以用下划线“_”分隔。
6、常量全部使用大写字母；
7、使用单下划线“_”开头的模块变量或者函数是受保护的，在使用 import * from 语句从模块中导入时这些变量或者函数不能被导入；
8、使用双下划线“__”开头的实例变量或方法是类私有的。

1.7.6. 保留字与标识符

1、保留字

Python 语言中已经被赋予特定意义的一些单词，开发程序时，不可以把这些保留字作为变量、函数、类、模块和其它对象的名称来使用。这些保留字是区分大小写的。

and、as、assert、break、class、continue、def、del、elif、else、except、finally、for、from、False、global、if、import、in、is、lambda、nonlocal、not、None、or、pass、raise、return、try、True、while、with、yield

2、标识符

标识符可以理解为一个名字，比每个人都有自己的名字，它主要用来标识变量、函数、类、模块和其它对象的名称。它的命名规则如下：

（1）由字母、下划线“_”和数字组成，并且第一个字母不能是数字；

（2）不能使用 Python 保留字；

（3）区分大小写；

（4）Python 中以下划线开头的标识符有特殊的意义，避免使用相似的标识符。

3、变量

在 Python 中，严格意义上，变量应该称为名字，也可以理解为标签。当把一个值赋给一个名字时，Python 就称为**变量**。

相当于把值存储在变量中，意思是在计算机内存中的某个位置，你也不需要知道这个位置在哪里。例如，有个变量 string，它保存字符串“学会了 Python 可以飞”，你通过变量名 string 就可以得到这个字符串了。

打个比方，就像你的快递存放在货物架上，上面附着写有你的名字的标签，当你来取快递的时候，你并不需要知道它们存放在这个大型货物架上的某个位置，只需要提供你的名字，快递员就会把你的快递找出来给你。要取你的快递，你只要提供你的名字就可以了。变量也是一样，你不准确地知道数据存储在内存中的哪个位置，只需要记住存储变量时所有的名字，再使用这个名字就可以了。

4、定义变量

在 Python 中，不需要先声明变量名及其类型，直接赋值就可以创建各种类型的变量，需要注意的是，对于变量的命名并不是任意的，应该遵循以下几条规则：

（1）变量名必须是一个有效的标识符；

（2）变量名不能使用 Python 中的保留字；

（3）慎用小写字母 1 和大写字母 O；

（4）应该选择有意义的单词作为变量名；

为变量赋值通过“=”来实现。语法格式为：

变量名 = value

例如：

```
01  number = 1024                    #整形变量
02  nickname = “床前明月光”          #字符串变量
```

Python 是一名动态类型的语言，变量的类型可以随时变化。如果我们先定义了一个变量，给它赋值 1024，它就是数字型的；然后再给它赋值一个字符串，它就变成了字符串类型的；

```
01  number = 1024                    #首先定义一个变量，它是整形变量
02  number = “床前明月光”            #重新赋值以后，它变成了字符串变量
```

在 Python 中，可以用 type()函数返回变量的类型。

```
01  number = 1024                    #首先定义一个变量，它是整形变量
02  print(type(number))
```

程序输出：

```
<class ‘int’>
```

5、常量

常量就是程序运行过程中，值不能改变的量，比如现实生活中的居民身份证号码、数学运算中的圆周率，这些都是不会发生改变的，它们都可以定义为常量。但是在 Python 中，并没有提供定义常量的保留字。

在程序中定义常量尽量使用大写字母和下划线。

1.8. 基本数据类型

1.8.1. 数字

1、整数

整数用来表示整型数值，没有小数部分的数值。整数包括正整数、负整数和 0，并且它的位置是任意的。当超过计算机自身的计算功能时，会自动转用高精度计算，如果要指定一个非常大的整数，只需要写它的位数就可以了。

整数类型包含**十六进制整数**、十进制整数、八进制整数、二进制整数。

（1）十进制整数

是常用的整数表现形式，逢十进一，由数字 0、1、2、3、4、5、6、7、8、9 组成。

（2）十六进制整数

逢十六进一，由数字 0、1、2、3、4、5、6、7、8、9、A、B、C、D、E、F 组成。数值需要以 0X 或者 0x 开头。如 0x25，转换成十进制数就是 37。

（3）八进制整数

逢八进一，由数字 0、1、2、3、4、5、6、7 组成。数值需要以 0o 或者 0O 开头。如 0o25，转换成十进制数就是 21。

（4）**二进制整数**

逢二进一，由数字 0、1 组成。如 101，转换成十进制数就是 5。

2、浮点数

浮点数由整数部分和小数部分组成，主要用于处理包括小数的数，如 1.414、0.5，-1.732、3.1415926 等。浮点数也可用科学记数法，如 2.7e2、-3.4e5、5.25e-3。

3、复数

复数由实数部分和虚数部分构成，并且使用 j 或者 J 表示虚部，可以用 **a + bj** 表示，或者 **complex(a,b)**表示，复数的实部 a 和虚部 b 都是浮点型。

1.8.2. 字符串

字符串就是连续的字符序列，可以是计算机所能表示的一切字符的集合。字符串属于不可变序列，通常使用单引号“’’”双引号“””或者三引号“”””””””括起来。

注意：

1、这三种形式在语义上没有任何差别，只是在形式上有差别。

2、单引号和双引号中的字符串序列必须在同一行上，三引号可以连续分布在多行。

3、字符串开始和结尾使用的引号形式必须一致。

4、Python 中的字符串不能改变。

5、反斜杠可以用来转义，使用 r 可以让反斜杠不发生转义。 如 r"this is a line with \n" 则\n 会显示，并不是换行。

6、Python 没有单独的字符类型，一个字符就是长度为 1 的字符串。

7、字符串可以用 + 运算符连接在一起，用 * 运算符重复。

（1）访问字符串中的值

Python 中的字符串有两种索引方式，从左往右以 0 开始，从右往左以 -1 开始。

字符串的截取的语法格式如下：变量[头下标:尾下标:步长]

Python 访问子字符串，可以使用方括号来截取字符串，如下实例：

```
01    var1 = 'Hello World!'
02    var2 = "Runoob"
03
04    print ("var1[0]: ", var1[0])
05    print ("var2[1:5]: ", var2[1:5])
```

程序输出：

```
var1[0]:    H
var2[1:5]:    unoo
```

（2）转义字符：

\	(在行尾时) 续行符
\\	反斜杠符号
\'	单引号
\"	双引号
\a	响铃
\b	退格(Backspace)
\e	转义
\000	空
\n	换行
\v	纵向制表符
\t	横向制表符
\r	回车
\f	换页
\oyy	八进制数，yy 代表的字符，例如：\o12 代表换行
\xyy	十六进制数，yy 代表的字符，例如：\x0a 代表换行

（3）Python 字符串运算符

+	字符串连接 a + b
*	重复输出字符串 a*2
[]	通过索引获取字符串中字符

[:] 截取字符串中的一部分，遵循左闭右开原则，str[0,2] 是不包含第 3 个字符的。
in 成员运算符 - 如果字符串中包含给定的字符返回 True
not in 成员运算符 - 如果字符串中不包含给定的字符返回 True
r/R 原始字符串 - 原始字符串：所有的字符串都是直接按照字面的意思来使用，没有转义特殊或不能打印的字符。 原始字符串除在字符串的第一个引号前加上字母 r（可以大小写）以外，与普通字符串有着几乎完全相同的语法。

```
01  a = "Hello"
02  b = "Python"
03
04  print("a + b 输出结果：", a + b)
05  print("a * 2 输出结果：", a * 2)
06  print("a[1] 输出结果：", a[1])
07  print("a[1:4] 输出结果：", a[1:4])

08  if( "H" in a) :
09          print("H 在变量 a 中")
10  else :
11          print("H 不在变量 a 中")
12
13  if( "M" not in a) :
14          print("M 不在变量 a 中")
15  else :
16          print("M 在变量 a 中")
17
18  print (r'\n')
19  print (R'\n')
```

程序输出：

```
a + b 输出结果： HelloPython
a * 2 输出结果： HelloHello
a[1] 输出结果： e
a[1:4] 输出结果： ell
H 在变量 a 中
M 不在变量 a 中
\n
\n
```

1.8.3. 布尔类型

布尔类型表示真或者假的值，在 Python 中，True 和 False 被解释为布尔值。

布尔值可以转换为数值，其中 True 表示 1，False 表示 0。

所有的对象都可以进行真值测试，其中，下列几种情况为假，其他对象在判断条件中都表现为真:

1、False 或者 None
2、数值中的 0，包括 0，0.0，虚数 0
3、空序列，包括字符串、空元组、空列表、空字典
4、自定对象的实例，该对象的__bool__方法返回 False 或者__len__方法返回 0。

1.8.4. 类型转换

有时候，我们需要对数据内置的类型进行转换，数据类型的转换，你只需要将数据类型作为函数名即可。注意：非数字字符串不能转换成数字类型

以下几个内置的函数可以执行数据类型之间的转换。这些函数返回一个新的对象，表示转换的值。

int(x [,base])	将 x 转换为一个整数
float(x)	将 x 转换到一个浮点数
complex(real [,imag])	创建一个复数
str(x)	将对象 x 转换为字符串
repr(x)	将对象 x 转换为表达式字符串
eval(str)	用来计算在字符串中的有效 Python 表达式,并返回一个对象
tuple(s)	将序列 s 转换为一个元组
list(s)	将序列 s 转换为一个列表
set(s)	转换为可变集合
dict(d)	创建一个字典。d 必须是一个序列 (key,value)元组。
frozenset(s)	转换为不可变集合
chr(x)	将一个整数转换为一个字符
ord(x)	将一个字符转换为它的整数值
hex(x)	将一个整数转换为一个十六进制字符串
oct(x)	将一个整数转换为一个八进制字符串

1.8.5. 基本输入输出

1、input()函数输入

使用内置函数可以接受用户的键盘输出，基本用法如下：

variable = input(“提示文字”)

其中，variable 为保存输入结果的变量，双引号内的文字是用于提示要输入的内容。

注意：无论输入的是数字还是字符，都将被作为字符串读取。如果想要接收数值，需要把接收到的字符串进行类型转换。例如：

```
age = int(input(“请输出数字：”))
```

2、print()函数输出

内置的 print()函数可以将结果输出到标准控制台上。其基本语法格式如下：

print(输出的内容)

输出的内容可以是字符串，也可以是包含运算符的表达式，将会输出计算结果。

在默认情况下，一条 print()语句输出后会自动换行，如果想要一次输出多个内容，而且不换行，可以将要输出的内容使用逗号分隔。

```
print(a, b, c )
```

在输出时，也可以把结果输出到指定文件。

```
01  fp = open(r'D:\newfile.txt' , 'a+')
02  print("我是中国人，我爱我们的祖国" , file = fp)
03  fp.close
```

程序执行以后，将会在目录下生成一个名称为 newfile.txt 的文件。同时把"我是中国人，我爱我们的祖国"写入文件。

1.9. 运算符

1.9.1. 算术运算符

算术运算符是处理四则运算的符号，在数字处理中应用最多，常用的算术运算符如下：

```
+    加 - 两个对象相加
-    减 - 得到负数或是一个数减去另一个数
*    乘 - 两个数相乘或是返回一个被重复若干次的字符串
/    除 - x 除以 y
%    取模 - 返回除法的余数
**   幂 - 返回 x 的 y 次幂
//   取整除 - 向下取接近除数的整数
```

```
abs()：返回绝对值
power(x, y)：返回 x 的 y 次方
divmod(x, y)：返回(x//y, x%y)
```

演示代码：

```
01  a = 21
02  b = 10
03  c = 0
04
05  c = a + b
06  print ("1 - c 的值为：", c)
07
08  c = a - b
09  print ("2 - c 的值为：", c)
10
11  c = a * b
12  print ("3 - c 的值为：", c)
```

```
13
14  c = a / b
15  print ("4 - c 的值为：", c)
16
17  c = a % b
18  print ("5 - c 的值为：", c)
19
20  # 修改变量 a 、b 、c
21  a = 2
22  b = 3
23  c = a**b
24  print ("6 - c 的值为：", c)
25
26  a = 10
27  b = 5
28  c = a//b
29  print ("7 - c 的值为：", c)
```

1.9.2. 赋值运算符

赋值运算符主要用来给变量赋值，使用是，可以直接把基本运算符“=”右边的值赋给左边的变量，也可以进行某些运算后再赋值给左边的变量。

=	简单的赋值运算符	c = a + b 将 a + b 的运算结果赋值为 c
+=	加法赋值运算符	c += a 等效于 c = c + a
-=	减法赋值运算符	c -= a 等效于 c = c - a
*=	乘法赋值运算符	c *= a 等效于 c = c * a
/=	除法赋值运算符	c /= a 等效于 c = c / a
%=	取模赋值运算符	c %= a 等效于 c = c % a
**=	幂赋值运算符	c **= a 等效于 c = c ** a
//=	取整除赋值运算符	c //= a 等效于 c = c // a

演示代码：

1.9.3. 关系运算符

关系运算符，也称比较运算符，用于对变量或者表达式的结果进行大小、真假比较，如果比较结果为真，则返回 True，如果为假，则返回 False。

比较运算符通常在条件语句中做为判断的依据。

==	等于：	比较对象是否相等 (a == b) 返回 False。
!=	不等于：	比较两个对象是否不相等 (a != b) 返回 True。

> 大于： 返回 x 是否大于 y (a > b) 返回 False。
< 小于： 返回 x 是否小于 y。所有比较运算符返回 1 表示真，返回 0 表示假。这分别与特殊的变量 True 和 False 等价。注意，这些变量名的大写。 (a < b) 返回 True。
>= 大于等于： 返回 x 是否大于等于 y。 (a >= b) 返回 False。
<= 小于等于： 返回 x 是否小于等于 y。 (a <= b) 返回 True。

1.9.4. 逻辑运算符

逻辑运算符是对真和假两种布尔值进行运算，运算后的结果仍是一个布尔值，Python 中的逻辑运算符主要包括 and（逻辑与）、or（逻辑或）、not（逻辑非）。

表达式 1	表达式 2	and	or	not 表达式 1
True	True	True	True	False
True	False	False	True	False
False	False	False	False	True
False	True	False	True	True

1.9.5. 位运算符

1、按位与

“按位与”运算的运算符为“&”，它的运算法则是：两个操作数所的二进制表示，只有对应位都是 1 时，结果位才是 1，否则就是 0。如果两个操作数的精度不同，则结果的精度与精度高的操作数相同。

2、按位或

“按位或”运算的运算符为“|”，它的运算法则是：两个操作数所的二进制表示，只有对应位都是 0 时，结果位才是 0，否则就是 1。如果两个操作数的精度不同，则结果的精度与精度高的操作数相同。

3、按位异或

“按位或”运算的运算符为“^”，它的运算法则是：两个操作数所的二进制表示，只有对应位同时都是 0 时或者同时都是 1 时，结果位才是 1，否则就是 0。如果两个操作数的精度不同，则结果的精度与精度高的操作数相同。

4、按位取反

“按位取反”与叫做“按位非”运算，运算符是“~”，“按位取反”运算就是将操作数对应的二进制数中的 1 修改为 0，0 修改为 1。

5、左移位

左移位运算符“<<”是将一个二进制操作数向左移动指定的位数，左边溢出的们被舍弃，右边的空位用 0 来补。

左移运算符相当于乘以 2^n。

6、右移位

右移位运算符“>>”是将一个二进制操作数向右移动指定的位数，右边溢出的位被舍弃，而在填充左边的空位时，如果是正数，最高位是 0，左侧空位填入 0，如果是负数，最高位是 1，左侧空位填入 1。

右移运算符相当于除以 2^n。

由于位移运算的速度很快，在程序中遇到表达式乘以或者除以 2^n 的情况时，一般都是有移位运算来代替。

1.9.6. 优先级

所谓运算符的优先级，是指在应用中哪一个运算符先计算，哪一个后计算，与数学的四则运算应遵循的“先乘除后加减”是一个道理。Python 中的运算符的运算规则：是优先级高的运算先执行，优先级低的运算后执行，同一优先级的操作，按照从左到右的顺序进行。也可以像四则运算那样，使用小括号、括号内的运算最先执行。

在编写程序时，昼使用括号（），来限定运算次序，以名运算次序发生错误。

**	指数 (最高优先级)
~ 、+、 -	按位翻转, 一元加号和减号
*、 / 、% 、//	乘，除，取模和取整除
+ -	加法减法
>> 、<<	右移，左移运算符
&	位 'AND'
^ |	位运算符
<=、 < > 、>=	比较运算符
== != = %= /= //= -= += *= **=	赋值运算符
and、 or 、not	逻辑运算符

1.10. 条件表达式

在程序开发中，经常会根据表达式的结果有条件地进行赋值。例如，要返回两个数中较大的数，可以使用下面的 if 语句。

```
01  a = 10
02  b = 8
03  if a>b:
04          r=a
05  else:
06          r=b
```

上面的代码可以使用条件表达式来表达，如下：

```
01  a = 10
02  b = 8
03  r = a if a>b else b
```

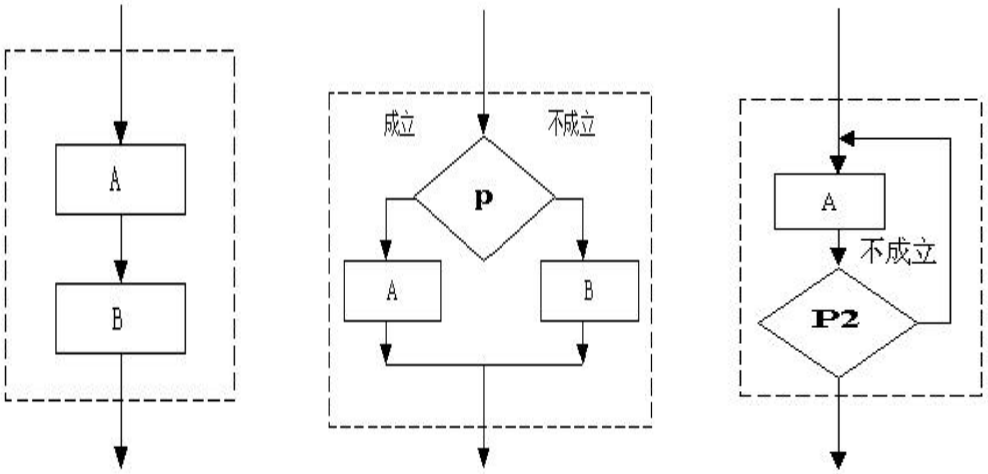
使用条件表达式时，先计算中间条件的值（a>b），如果结果为 True，返回 if 语句左边

的值，否则返回 else 右边的值。

2. 流程控制

2.1. 程序结构

计算机在解决某个具体问题时，主要有三种情况，分别是顺序执行所有的语句、选择执行部分语句和循环执行部分语句。对应程序设计中的 3 种基本结构是顺序结构、选择结构和循环结构。



2.2. 选择语句

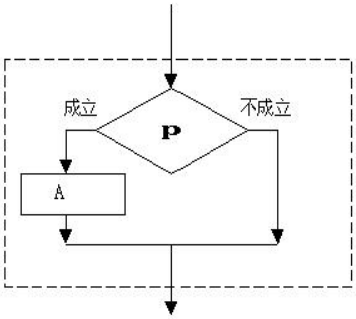
1、if 语句

最简单的法形式如下：

if 表达式：

语句块

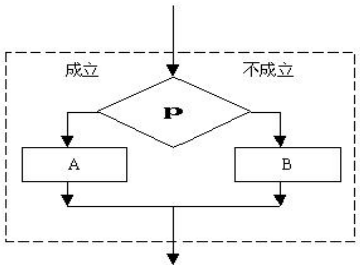
其中，表达式可以是一个单纯的布尔值或者变量，也可以是比较表达式或者逻辑表达式，如果表达式的值为真，则执行语句块；如果表达式的值为假，就跳过语句块。继续执行后面的语句。



注意：（1）if 语句后面有个冒号；

（2）使用 if 语句时，如果需要执行多个语句，所有的语句的缩进应该相同；

2、if…else 语句



如果遇到只能二先一的条件，可以用 if…else 语句，其语句格式如下：

```
if 表达式：
    语句块 1
else：
    语句块 2
```

使用 if…else 时，表达式可以是一个单纯的布尔值或者变量，也可以是比较表达式或逻辑表达式，如果满足条件，则执行 if 后面的语句块，否则，执行 else 后面的语句块。

else 不能单独使用。

3、if…elif…else 语句

如果遇到多先一的情况，可以用 if…elif…else 语句，其语句格式如下：

```
if 表达式 1：
    语句块 1
elif 表达式 2：
    语句块 2
elif 表达式 3：
    语句块 3
    ……
else：
    语句块 n
```

使用 if…else 时，表达式可以是一个单纯的布尔值或者变量，也可以是比较表达式或逻辑表达式，如果满足条件，则执行 if 后面的语句块，如果表达式为假，则跳过该语句，进进入下一个 elif 的判断，只有在所有的表达式都是假的情况下，地执行 else 后面的语句块。

elif、else 不能单独使用，必须跟 if 一起使用。

4、if 语句套嵌套

前面介绍了 3 种形式的选择语句，它们之间可以相互套嵌。

```
if 表达式 1：
    if 表达式 2：
```

```
        语句块 1
    else：
        语句块 2
else：
    if 表达式 3：
        语句块 3
    else：
        语句块 4
```

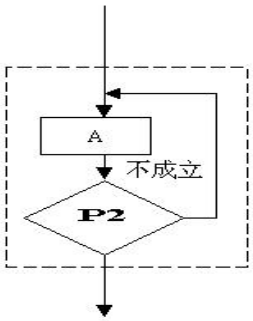
2.3. 循环语句

日常生活中很多问题都无法一次解决，如盖楼，所有的高层都是一层一层建起来的。还有有些事物，必须周而复始地运转，才能保证其存在的意义。例如，公交车、地铁等交通工具，必须每天在同样的时间，往返于始发站和终点站之间，类似这样的反复做同一件事情的情况，称之为循环。

循环主要有两种类型：

（1）重复一定次数的循环称为计次循环，如 for 循环；

（2）一直重复，直到条件不满足时才结束的循环，称之为条件循环，只要条件为真，这种循环会一直持续下去，如 while 循环。



2.3.1. while 循环

While 循环是通过一个条件来控制是否要继续反复执行循环体中的语句，语法如下：

```
while 条件表达式：
    循环体
```

循环体指一组被重复执行的语句。

当条件表达式的返回值为真的时候，则执行循环体中的语句，执行完毕后，重新判断条件表达式的返回值，直到条件表达式的返回结果为假，退出循环。

注意：一定不要忘记添加使循环条件改变为 False 的代码，否则，将会产生死循环。

2.3.2. for 循环

For 循环是一个计次循环，一般应用在循环次数已知的情况下。语法如下：

```
for 迭代变量 in 对象：
    循环体
```

其中，迭代变量用于保存读取出的值，对象为要遍历或迭代的对象，该对象可以是任何有序的序列对象，如字符串、列表和元组等，循环体为一组被重复执行的语句。

for 循环经常和 range()函数一起使用，它用于生成一系列连续的整数。具体用法：
range(start, end, step)
start：用于指定计数的起始值
end：用于指定计数的结束值，但是不包括该值。
step：用于指定步长，两个数之间的间隔。
range()函数如果只有一个参数，那表示指示结束值，如果两个参数，则表示起始值和结束值，只有三个参数时，最后一个值才表示步长。

2.3.3. 循环嵌套

循环体中可以套嵌入另一个循环体，这个就中循环套嵌。 for 循环和 while 循环可以相互套嵌。

（1） while 循环中套 while 循环

```
while 条件表达式 1：
    while 条件表达式 2：
        循环体 2
    循环体 2
```

（2） for 循环中套 for 循环

```
for 迭代变量 1 in 对象 1：
    for 迭代变量 2 in 对象 2：
        循环体 2

    循环体 1
```

（3） for 循环中套 while 循环

```
for 迭代变量 in 对象：
    while 条件表达式 1：
        循环体 2
    循环体 1
```

（4） while 循环中套 for 循环

```
while 条件表达式：
    for 迭代变量 in 对象：
        循环体 2
    循环体 1
```

循环语句之间可以实现多层的套嵌，在写程序时根据实际应用实现。

2.3.4. 结束循环

当循环条件一直满足时，程序会一直执行下去。如果希望在中间离开循环，有两种方法可以做到：使用 continue 和 break 语句。

1、break

break 语句用来终上当前循环，包括 while 和 for 在内的所有控制语句。

打个比方，某人独自一人沿着操场跑步，原来打算跑 10 圈，可是在跑到第 4 圈的时候，看到了自己的女神，于是果断地停下来，中止跑步。

Break 语句一般会结合 if 语句进行搭配使用，表示在某种条件下终止循环。如果使用的是套嵌循环，break 语句跳出最内层的循环。语法格式如下：

```
while 条件表达式 1：
    执行代码
    If 条件表达式 1:
        break

for 迭代变量 in 对象：
    执行代码
    If 条件表达式:
        break
```

2、continue

continue 语句没有 break 语句强大，它只能中止本次循环而提前进入下一次循环。

打个比方，某个独自一个沿着操场跑步，原来打算跑 10 圈，可是在跑到第 4 圈的时候，看到自己的女神也在跑步，于是果断停下来。跑回原点等待。制造一次美丽的邂逅。然后从第 5 圈开始跑。

Break 语句一般会结合 if 语句进行搭配使用，表示在某种条件下终止循环。如果使用的是套嵌循环，break 语句跳出最内层的循环。语法格式如下：

```
while 条件表达式 1：
    执行代码 1
    If 条件表达式 1:
        Continue
    执行代码 2

for 迭代变量 in 对象：
    执行代码 1
    If 条件表达式:
        Continue
    执行代码 2
```

3、**pass**

在 Python 中，还有一个 pass 语句，表示空语句，它不做任何事情，只起到占位的作用。方便以后扩展时进行某种处理。

```
for i in range(1,10):
    if i/2 == 0:                #判断是否是偶数
        print(i,end="")
    else:                       #不是偶数
        pass                    #占位符，不做任何事情
```

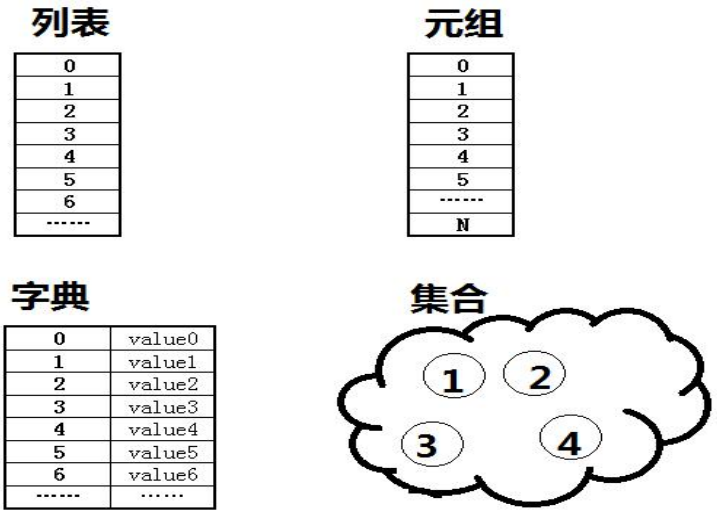
3. 数据结构

3.1. 序列

数据结构是以某种方式组合起来的数据元素集合。在 Python 中，最基本的数据结构是**序列**。

序列是一块用于存放多个值的连续内存空间，并且按照一定的顺序排列，每个值称为元素，都分配一个数字称为索引或位置，通过该索引可以取出相应的值，例如，我们可以把一家酒店看做成一个序列，那么酒店里的每个房间都可以看作是这个序列的元素，房间号就相当于索引，可以通过房间号找到对应的房间。

在 Python 中，序列结构主要有列表、元组、集合、字典和字符串。



对于序列有以下几个通用的操作：索引、切片、相加、相乘等。但是集合和字典不支持索引、切处、相加和相乘操作。

3.1.1. 索引

序列中的每个元素都有一个编号，也称索引。

这个索引是 0 开始递增，即下标为 0 表示第一个元素，下标为 1 表示第二个元素，以此类推。

序列的索引可以是负数，表示索引从右往左计数，也就是从最后一个元素开始计数。下标为-1 表示最后一个元素，-2 表示倒数第二个元素，以此类推。

注意：在采用负数做为索引时，是从-1 开始的，而不是从 0 开始，这是为了防止第一

元素在重合。

3.1.2. 切片

切片操作是访问序列元素的另一种方法，它可以访问一定范围内的元素。通过切片操作可以生成一个新的序列。实现切片操作的语法格式如下：

```
sname[start: end: step]
```

参数说明：

sname: 表示序列的名称

Start: 表示切片的开始位置（包含该位置），如果不指定，默认为 0

End: 表示切片的截止位置（不包括该位置），如果不指定，默认为序列的长度

Step: 表示切片的步长，如果省略，则默认为 1。当省略这个参数时，最后一个冒号也可以省略。

注意：步长不能为 0，否则无法向前移动，但可以是负数，也就是从右往左取数据。

3.1.3. 相加

在 Python 中，支持两种相同类型的序列相加操作，即将两个数序列进行连接，不会去除重复元素。使用“+”运算符实现。

在进行两个序列相加时，相同类型的序列是指，同为列表、元组、字符串等，序列中的元素类型可以不同。但是不能列表与元组相加，或者列表与字符串相加。

3.1.4. 相乘

使用数字 n 乘以一个序列会生成新的序列。新序列的内容为原来序列被重复 n 次的结果。

3.1.5. 成员检查

在 Python 中，可以用 **in** 关键字检查某个元素是否是序列的成员，即检查某个元素是否包含在该序列中。语法格式如下：

```
value in sequence
```

其中，value 表示要检查的元素，sequence 表示指定的序列。

当然，也可以 **not in** 关键字来检查某个元素是否不包括在指定的序列当中。

3.1.6. 计算长度和最大最小值

在 Python 中，提供了内置函数计算序列的长度、最大值和最小值。用 len()函数计算序列的长度，即返回序列包含多少个元素；使用 max()函数返回序列中的最大元素；使用 min()函数返回序列中的最小元素。

```
01    num = [0,1,2,3,4,5,6,7,8]
```

```
02  print("序列的长度：", len(num))
03  print("序列中的最大值：", max(num))
04  print("序列中的最小值：", min(num))
```

其它内置函数：

list(): 将序列转换为列表
str(): 将序列转换为字符串
sum(): 计算元素和
sorted(): 对元素进行排序
reversed(): 反向序列中的元素
enumerate(): 将序列组合为一个索引序列，多用在 for 循环。

3.2. 列表

列表由一系列按特定顺序排列的元素组成。它是 Python 中内置的可变序列。

在形式上，所有的元素都放在一对中括号“[]”中，两个相邻元素间使用逗号“，”隔开。

在内容上，可以将整数、实数、字符串、列表、元组等任何类型的内容放入列表中，并且同一个列表中，元素的类型可以不同，因为它们之间没有任何关系。

由此可见，Python 中的列表是相当灵活的。

3.2.1. 创建列表

1、使用赋值运算符直接创建列表

可以使用赋值运算符“=”直接将一个列表赋值给一个变量，具体的语法格式如下：

listname = [element1, element2, element3,…,elementN]

其中，listname 表示列表名称，可以是任何符合标志符命名规则的标识符。element1，element2， element3，…，elementN 表示列表中的元素，个数没有限制。

```
Superlist = [2, 3, 4, "人生苦短", "床前明月光", "银河落九天", [6, 7, "Python"]]
```

在使用列表的时候，虽然可以将不同的类型的数据放入同一个列表中，但是通常情况下，我们不这样做，而是在同一个列表中只放入同一类形的数据。

2、创建空列表

```
Emptylist = []                    #这个列表中什么都没有
```

3、创建数值列表

数值列表很常用，它的语法格式如下：

```
list(data)
```

其中，data 表示可以转换为列表的数据，其类型可以是 range 对象、字符串、元组、或者说其它迭代类型的数据。

```
list(range(10, 20 ,2))   #创建一个 10 到 20 中所有偶数的列表
```

4、删除列表

对于已经创建的列表，不再使用时，可能使用 del 语句将其删除，语法格式如下：

del listname

其中 listname 为要删除的列表名，del 语句并不常用，因为 python 自带的垃圾回收机制会自动销毁不用的列表，所以，就算我们不手工将其删除，Python 也会自动将其回收。

在删除列表之前，一定要保证列表是已经存在的列表。

3.2.2. 遍历列表

遍历列表中的所有元素是最常用的一种操作，在遍历的过程当中可以完成查询、处理等功能。遍历的方法很多，在这里介绍两种：

1、使用 for 循环实现

直接使用 for 循环遍历列表，只能输出元素的值。它的语法格式如下：

```
for item in listname
```

输出 item

其中 item 用于保存获取到的元素值，要输出元素内容时，直接输出该变量即可；listname 为列表名称。

2、使用 for 循环和 enumerate()函数实现

使用 for 循环和 enumerate()函数可以实现了同时输出索引值和元素内容。

```
for index, item in enumerate(listname)
```

输出 index 和 item

Index 用于保存元素的索引，item 用于保存获取到的元素值，listname 为列表名称。

3.2.3. 更新列表

添加、修改、删除列表中的元素称之为更新列表。在实际开发中，经常需要对列表进行更新。

1、添加元素

（1）列表对象的 append()方法用于在列表的末尾追加元素，它的语法格式如下：

listname.append(obj)

其中，listname 为要添加的元素的列表名称，obj 为要添加到列表末尾的对象。

（2）列表对象还提供了 insert()方法向列表中添加对象，该方法用于向指定的位置插入元素。但是执行效率没有 append()方法高，不推荐

listname.insert(index, obj)

Index 表示要在列表插入的位置。

（3）如果要将一个列表中的元素全部添加到另一个列表当中，可以使用列表的 extend()方法实现，extend()方法的具体语法如下：

listname.extend(seq)

Listname 为原列表，seq 为要添加的序列。

2、修改元素

修改列表中的元素只需要通过了索引获取该元素，然后再为其重新赋值即可。

```
01  listvalue = [0,1,2,3,4]
02  listvalue[2] = 10
```

3、删除元素

（1）根据索引删除

使用 del 语句实现。

```
01  listvalue = [0,1,2,3,4]
02  del listvalue[2]
```

删除的数据可以是切片操作的结果

（2）根据元素值删除

使用列表对象的 remove()方法

```
01  listvalue = [0,1,2,3,4]
02  listvalue.remove(3)
```

注意：使用列表对象的 remove()方法删除元素时，最好要先判断元素是否存在，如果不存在，程序会发生异常。

3.2.4. 列表统计计算

（1）指定元素出现的次数

列表对象的 count()方法用于判断指定元素出现的次数，返回结果为 0 时，表示不存在此元素。

Listname.count(obj)

Listname 是列表名称，obj 表示要判断的对象。

（2）获取提定元素首次出现的下标

使用列表对象的 index()方法可以获取指定元素的在列表中首次出现的位置，即索引。基本语法格式如下：

listname.index(obj)

listname 是列表名称，obj 表示要判断的对象。返回值就是首次出现的索引值。

如果元素不存在，会抛出异常。

（3）统计数值列表的元素和

列表对象提供了 sum()用于统计数值列表中各元素的和。

sum(iterable[, start])

Iterable 表示要统计的列表；

Start：表示统计结果从哪个数开始。

3.2.5. 列表排序

（1）使用列表对象的 sort()方法实现

列表对象提供了 sort()方法用于对原列表中的元素进行排序，排序后，原来列表中的顺

序将发生改变。语法格式如下：

listname.sort(key = None, reverse =False)

listname 表示要进行排序的列表。

key：可选参数，表示指定一个从每个列表中提取一个比较键，例如，设置 key=str.lower 表示排序时不区分大小写。

reverse：可选参数，如果将其值设置为 True，则表示降序排列。如果将其值设置为 False，则表示升序排列，默认为升序排序。

注意：使用 sort()方法对字符串列表进行排序的时候，采用的规则是先对大写字母进行排序，然后再对小写字母进行排序，也可以设定 key 的值忽略大小写。

Sort()对中文支持不是非常友好

（2）使用内置的 sorted()方法实现

Python 中有个内置的 sorted()函数，用于对列表进行排序，使用该函数进行排序后，原来列表中的元素顺序保持不变。语法格式如下：

sorted(iterable, key = None, reverse = False)

iterable 表示要进行排序的列表。

key：可选参数，表示指定一个从每个列表中提取一个比较键，例如，设置 key=str.lower 表示排序时不区分大小写。

reverse：可选参数，如果将其值设置为 True，则表示降序排列。如果将其值设置为 False，则表示升序排列，默认为升序排序。

注意：列表对象的 sort()方法，和内置的 sorted()函数的作用基本相同。所不同的是前者会改变原来列表中元素排序顺序。但是后者会建立一个原来列表的副本，该副本为排序后的列表。

3.2.6. 列表推导式

使用列表的推导式可以快速生一个新的列表。或者根据某个列表生成满足指定需求的列表。

（1）生成指定范围内的数值列表

list = [expression for var in range]

list: 表示生成列表的名称；

expression: 表达式，用于计算新列表的元素。

var: 循环变量

range: 采用 range()函数生成的 range 对象

```
01  import random
02  randomnumber = [random.radint(10, 100) for i in range(10)]
```

（2）根据列表生成指定需求的列表

Newlist = [expression for var i list]

Newlist: 表示新列表

expression: 表达式，用于计算新列表生成的元素

var: 变量，值为后面列表的每个元素值；

list：用来生成新列表的原列表

```
01  price = [300, 500, 600, 800, 900,1200]
02  sale = [int(x * 0.88) for x in price]
03  print(sale)
```

（3）从列表从选择符合条件的元素组成新的列表

```
Newlist = [expression for var i list if condition]
```

Newlist: 表示新列表
expression: 表达式，用于计算新列表生成的元素
var: 变量，值为后面列表的每个元素值；
list：用来生成新列表的原列表
condition: 条件表达式，用于指筛选的条件。

```
01  price = [300, 500, 600, 800, 900,1200]
02  sale = [x* 0.5 for x in price if x>600]
03  print(sale)
```

3.2.7. 列表的常用函数

append()：
extend()：
insert()：
remove()：
pop(i)：删除 index 为 i 的元素删除，并将删除的元素返回，若不指定，默认返回最后一个元素
clear()：清空列表
index()：
count()：统计列表中元素个数
reverse()：反向排序
sort()：排序
copy()：返回列表的副本

3.2.8. 二维列表

在 Python 中，由于列表元素可以是列表，所以它也支持二维列表的概念。

一个酒店有很多的楼层，这些楼层就像一个大列表。而每一层又有很多的房间，每一层也相当于一个列表。

二维列表中的信息以行和列的形式表示，第一个下标代表元素所有的行，第二个下标代表元素所在的列。

（1）直接定义列表

```
Listname = [[元素 11，元素 12，……，元素 1n],
            [元素 21，元素 22，……，元素 2n],
            [元素 31，元素 32，……，元素 4n],
```

……,
[元素 n1，元素 n2，……，元素 nn]
]

（2）用循环创建

首先创建一个列表，然后在这个列表中添加列表元素

（3）使用列表推导式创建

```
Listvalue = [[j for j in range(10)] for i in range(4)]]
```

（4）访问二维列表

创建二维列表后，可以通过下面的语法格式访问列表中的元素：

```
Listname[下标 1][下标 2]
```

3.3. 元组

3.3.1. 元组与列表的区别

元组是 Python 中另一个重要的序列结构，它与列表类似，也是由一系列按特定顺序排列的元素组成。但是它是不可变的序列。因此，元组也可以称为不可变的列表。

形式上，元组的所有元素都放在一对小括号“（）”中，两个相邻的元素间使用“，”分隔。

在内容上，可以将整数、实数、字符串、列表、元组等任何类型蛛内容放在元组中，并且在同一个元组中，元素的类型可以不同。因为它们之间没有任何关系。通常情况下，元组用于保存程序中不可修改的内容。

区别：

（1）列表属于可变序列，它的元素可以随时修改或者删除，而元组属于不可变的序列，其中的元素不可以修改，除非整体替换；

（2）列表可以使用 append()extend()insert()remove()pop()等方法实现添加和修改列表元素，而元组没有这几个方法，因此，不能向元组中添加和修改元素。同样，也不能删除元素。

（3）列表可以使用切片访问和修改列表中的元素，元组也支持切片，但是它只支持通过切片访问元组中的元素，不支持修改。

（4）元组比列表的访问速度和处理速度快，所以如果只需要对其中的元素进行访问，而不进行任何修改，建设使用元组。

（5）列表不能作为字典的键，而元组可以。

3.3.2. 创建元组

（1）使用赋值运算符直接创建元组

可以使用赋值运算符“=”直接将一个元素赋值给变量。具体的语法格式如下：

```
Tuplename = (element1,element2, element3,……, elementN)
```

创建元组的语法与创建列表的语法类似，只是列表用“[]”，而元组用“（）”

注意：

虽然元组使用小括号将所有的元素括起来，但是实际上，小括号不是必须的，只要一组值用逗号分隔开来，Python 就认为它是一个元组；

如果要创建的元组只包括一个元素，则在第一个元素后面加一个逗号，否则，解释器将会认为它是一个字符串。

Tuplename = (“床前明月光”，)

（2）创建空元组

```
emptyTuple = ()
```

空元组可以应用在为函数传递一个空值或者返回空值时。例如，定义一个函数必须传递一个元组类型的值，而我们还不想为它传递一组数据，那么就可以创建一个空元组传递给它。

（3）创建数值元组

可以使用 tuple()函数直接将 range()函数循环出来的结果转换为数组元组。

```
tuple(data)
tuple(range(10, 20, 2))
```

（4）删除元组

对于已经创建的元组，不再使用时，可能使用 del 语句将其删除。语法格式如下：

```
del tuplename
```

tuplename 为要删除的元组的名字。

在实际开发中，并不常用，因为 Python 有垃圾回收机制。

3.3.3. 访问元组

（1）索引方式

```
tuplename[index]
```

（2）切片方式

```
tuplename[start, end]
```

（3）for 循环方式

```
for item in tuplename
```

（4）for 循环加 enumerate()函数

```
for index, item in enumerate(tuplename)
```

3.3.4. 修改元组

（1）元组不能修改，不能单独修改元组中的一个元素；

```
Tuplename[3] = “重新赋值”      #程序会报异常
```

（2）可能对元组进行重新赋值；

```
Tuplename = (1,2,3,4,5,6)
Tuplename = (9,8,7,6,5,4)
```

（3）可能对元组进行连接组合操作，连接的内容必须都是元组。不能将元组与列表或者字符串进行连接。

```
Tuplename1 = (1,2,3,4,5,6)
Tuplename1 = Tuplename1 + (9,8,7,6,5,4)
```

3.3.5. 元组推导式

使用元组的推导式可以快速生成一个新的元组，它的表现形式与列表的推导式类似，只是将列表中的推导式中的中括号“[]”换成小括号“（）”

使用元组推导式生成的结果并不是一个元组或者是一个列表，而是一个生成器对象，这一点和列表推导式的是不同的。要使用该生成器对象，可以将其转换成列表或者元组，转换成元组使用 tuple()函数，而转换成列表使用 list()函数。

```
01  import random
02  randomnumber = (random.randint(10, 100) for i in range(10))
03  randomnumber = tuple(randomnumber )
```

生成器对象在遍历后，该对象就不存在了，如果还想再使用该生成器对象，都必须重新创建一个生成器对象。

3.4. 字典

字典（dictionary）和列表类似，也是可变序列，不过与列表不同的是，它是无序的可变序列。保存的内容是以“键-值对”的形式存放的。这类似于《新华字典》，它可以把拼音和汉字关联起来。通过拼音表可以快速找到想要的汉字。其中，拼音就相当于键，而对应的汉字就相当于值。键是唯一的，而值可以有很多个。

字典在定义一个包含多个命名字段的对象时，很有用。它的主要特征如下：

（1）通过键而不是通过索引来读取

字典有时也称为关联数组或者散列表，它是通键将一系列的值联系起来。这样就可以通过键从字典中获取指定项，但不能通过索引得到。

（2）字典是任意对象的无序组合

字典是无序的，各项是从左到右随机排序的，保存在字典中的项没有特定的顺序。这样可以提高查找顺序。

（3）字典是可变的，并且可以任意套嵌

字典可以在原处增加长或者缩短，并且它支持任意深度的套嵌，也就说它的值可以是其它的字典。

（4）字典中的键必须唯一

不允许同一个键出现两次，如果出现两次，则最后一个被记住。

（5）字典中的键必须不可变

字典中的键是不可变的，所以可以使用数字、字符串、或者元组，但不能使用列表。

3.4.1. 创建字典

（1）定义字典

每个元素都包含两个部分：键和值。并且在键和值之间使用冒号“：”隔开，相邻两个元素使用逗号分隔，所有的元素放在一个大括号“{}”中，语法格式如下：

dictionary = {'key1':'value1', 'key2':'value2', ……, 'keyN':'valueN' }

Dictionary: 表示字典名称

Key1,key2，……，keyN：表示元素的键，必须唯一，并且不可变，可以是字符串、数字或者元组；

value1,value2，……，valueN：表示元素的值，可是是任意数据类型，不必须唯一。

（2）空字典

dictionary = {}

dictionary = dict()

（3）通过映射函数创建

语法如下：

dictionary = dict(zip(list1, list2))

参数说明：

dictionary: 表示字典名称

zip(): 用于将多个列表或者元组对应位置的元素组合成元组，并返回包含这些内容的 zip 对象，如果想要得到元组，可以将 zip 对象使用 tuple()函数转换成元组；如果想要得到列表，则可以使用 list()函数将其转换成列表。

list1: 表示一个列表，用于指定生成字内的键

list2: 表示一个列表，用于指定生成字内的值

如果两个列表的长度不同，则与最短的列表长度相同。

（4）给定键值对创建

语法如下：

dictionary = dict(key1 = value1, key2 = value2, ……, keyn = valuen)

dictionary: 表示字典名称

key1,key2，……，keyN：表示元素的键，必须唯一，并且不可变，可以是字符串、数字或者元组；

value1,value2，……，valueN：表示元素的值，可是是任意数据类型，不必须唯一。

（5）创建键值为空的字典

可以使用 dict 对象的 fromkeys()创建键值为空的字典，语法如下：

dictionary = dict.fromkeys(list)

dictionary: 表示字典名称

list1: 表示一个列表，用于指定生成字内的键

```
mylist = ['一','二','三','四','五']
mydict = dict.fromkeys(mylist)
print(mydict)

输出：
{'二': None, '四': None, '五': None, '三': None, '一': None}
```

（6）删除字典

不再需要的字典可能通过 del 命令删除。

del dictionaryname

如果只是想删除字典中的全部元素，可能使用字典对象的 clear()方法，执行这个方法后，原字典将变成空字典。

除了上面介绍的两个方法，还可以使用字典对象的 pop()方法删除并返回指定的键的元素，以及使用字典对象的 popitem()方法删除并返回字典中的一个元素。

3.4.2. 访问字典

（1）通过键值

dictionary[key]

dictionary: 字典名称

key: 需要取值的键值

如果指定的键值不存在，将出抛出异常，需要使用 if 语句对不存在的情况进行处理，可以给一个默认值。

（2）通过 get()

语法格式：

dictionary.get(key[,default])

其中，dictionary 为字典对象，key 为指定的键，default 为可选项，用于当指定的键不存在时，返回一个默认值，如果省略，则返回 None。

（3）遍历字典

使用字典对象的 items()方法可以获取字典中全部的键值对列表。语法格式如下：

Dictionary.items()

其中，dictionary 为字典对象，返回值为可遍历的键值对元组列表。

```
01  dictionary = {‘一’:’1111’, ‘二’:’2222’, ‘三’:’3333’, ‘四’:’4444’ }
02  for item in dictionary.items():
03      print(item)
# 上面输出的是种个元组中的元素
04  for key,value in dictionary.items():
05      print(key, “:”, value)
```

3.4.3. 更新字典

字典是可变序列，所以可以随时在其中添加键值对，向字典中增加元素的语法格式如下：

dictionary[key] = value

参数说明：

dictionary：字典名称

key：要添加元素的键，必须是唯一的，并且不可变，如：字符串、数字、元组。

value：表示元素的值，可以是任意数据类型，不是必须唯一。

在字典中，键必须是唯一的，所以如果交添加的键与已经存在的键重复，那么将使用新的值替换掉原来的值。这相当于修改了字典中的元素。

当字典中的某个元素不需要时，可以使用 del 命令将其删除：del dictionary[key]
如果删除一个不存在的键，会抛出异常。

3.4.4. 字典推导式

使用字典的推导式可以快速生成一个字典，它的表现形式与列表推导式类似。

```
01  import random
02  randomdict = {i: random.randint(10,100) for i in range(1, 5)}
```

3.5. 集合

Python 中的集合（set）与数学中的集合概念类似，也是用于保存不重重的元素。它有可变集合（set）和不可变集合（frozenset）两种。

在形式上，集合的所有元素都放在一对大括号“{}”中，两个相邻的元素间使用“，”分隔。集合最好的应用就是去重，因为集合中的每个元素都是唯一的。

3.5.1. 创建集合

1、直接使用“{}”创建

语法格式如下：

```
setname = {element1, element2, element3,……, elementn}
```

setname：集合名称

element1, element2, element3,……, elementn：集合中的元素，个数没有限制。在创建集合时，如果有重复的元素，自动只保留一个。

2、使用 set()函数

语法格式如下：

```
setname = set(iteration)
```

setname：表示集合名称

iteration：表示要转换成集合的可迭代对象，可以是列表、元组、range 对象等，另外，也可以是字符串。如果是字符串，返回的集合将是包含全部不重复字符的集合。

在创建空集合时，只能使用 set()函数实现，而不能使用一对大括号“{}”实现，直接使用一对大括号“{}”，表示创建一个空字典。

推荐采用 set()创建函数。

3.5.2. 添加删除元素

1、添加元素

向集合中添加元素可以使用 add()方法实现，它的语法格式如下：

setname.add(element)

Setname：表示要添加元素的集合；

Element：表示要添加元素内容；这里只能使用字符串、数字及布尔型的 True 和 False 等，不能使用列表、元组等可迭代对象。

2、删除元素

可能使用 del 命令删除整个集合，也可以使用集合的 pop()方法或者 remove()方法删除一个元素，或者使用集合对象的 clear()方法清空集合，即删除集合中所有的全部元素，使其变为空集合。

3.5.3. 集合运算

集合最常用的操作就是进行交集、并集、差集和对称差集运算。

（1）进行交集运算时使用“&”符号；

以属于集合Ⅰ且属于集合Ⅱ的元素为元素的集合称为集合Ⅰ与集合Ⅱ的交（集），即图中的 B；

（2）进行并集运算时使用“|”符号；

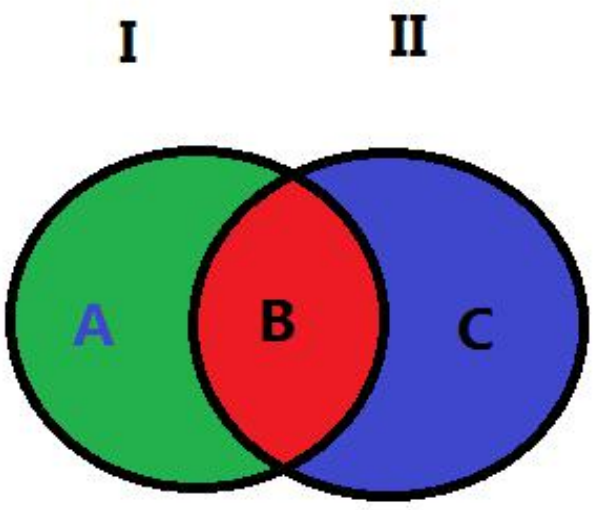
由所有属于集合Ⅰ或属于集合Ⅱ的元素所组成的集合，叫做Ⅰ、Ⅱ的并集，即图中的 A、B、C；

（3）进行差集运算则使用“-”符号；

Ⅰ，Ⅱ是两个集合，Ⅰ－Ⅱ则所有属于集合Ⅰ且不属于集合Ⅱ的元素构成的集合，即图中的 A，同理，Ⅱ－Ⅰ则所有属于集合Ⅱ且不属于集合Ⅰ的元素构成的集合，即图中的 C。

（4）进行对称差集运算时使用“^”符号；

Ⅰ＾Ⅱ为集合Ⅰ与集合Ⅱ中所有不属于Ⅰ∩Ⅱ的元素的集合,即图中的 A 和 C。



3.6. 字符串

3.6.1. 拼接字符串

使用“+”运算符可以完成对多个字符串的拼接，“+”运算符可以连接多个字符串并产生一个字符串对象。

字符串不允许直接与其它类型的数据拼接，如果要与整数拼接，要把整数转换成字符串。

3.6.2. 计算字符串长度

Python 中，提供了 len()函数计算字符串的长度

```
len(string)
```

数字、英文、小数点、下划线和空格占一个字节；一个汉字可能占 2-4 个字节，这个取决于采用的编码。汉字在 GBK\GB2312 编码中点 2 个字节，在 UTF\unicode 中一般点 3 个或 4 个字节。

3.6.3. 截取字符串

由于字符串也属于序列，所以要截取字符串，可以采用切片方法实现。通过切片截取字符串的语法格式如下：

```
String[start: end: step]
```

3.6.4. 检索字符串

1、 count()方法

检索一个字符串在另一个字符串中出现的次数，如果检索的字符串不存在，返回 0，否则返回出现的次数。

```
String.count(sub[,start[,end]])
```

2、 find()方法

用于检索是否包含指定的子字符串。如果检索的字符串不存在，则返回-1，否则返回首次出现的索引。

```
String.find(sub[,start[,end]])
```

3、 index()方法

Index()方法与 find()方法类似，也是用于检索是否包含指定的子字符串。只不过 index()在指定的字符串不存在的时候，会抛出异常。

```
String.index(sub[,start[,end]])
```

4、 startswith()方法

用于检索字符串是否以指定的字符串开头。

String.startswith(sub[,start[,end]])

5、 endswith()方法

用于检索字符串是否以指定的字符串结尾。

String.endswith(sub[,start[,end]])

3.6.5. 大小写转换

1、lower()

用于将字符串听全部大写字母转换成小写字母。

String.lower()

2、upper()

用于将字符串听全部大写字母转换成大写字母。

String.upper()

3.6.6. 格式化字符串

字符串对象提供了 format()方法用于进行字符串格式，其语法格式如下：

String.format(args)

在创建格式化模板的时候，需要用“{}”“：”指定占位符，基本格式如下：

{[index][:[[fill]align][sign][#][width][.pricision][type]]}

Index：用于指定要设置格式对象在参数列表中的索引位置

Fill：用于指定空白处的填充字符。

Align：用于指定对齐方式，左对齐“<”，右对齐“>”，内容居中“^”，内容右对齐符号最左侧“=”，需要配合 width 使用。

Sign：用于指定有无符号数，“+”表示正数加正号，负数加负号，“-”表示正数不变，负数加负号，空格表示正数加空格，负数加负号；

#：表示会显示“0b”“0x”“0o”前缀

Width：用于指家所占宽度

.pricision：用于指定保留小数位数

Type：用指定类型

格式化字符：

S：对字符串类型格式化；

D：十进制整数

C：将十进制自动转换成对应的 unicode 字符

E 或者 e：转换成科学计数法再格式化
b：转换成二进制再格式化
o：转换成八进制再格式化
X 或者 x：转换成十六进制数再格式化
F 或者 f：转换成浮点数再格式化
%：显示百分比

```
template = '编号：{:0<8s}\t 公司名称：{}\t 官网：http://www.{}.com'
mystring1 = template.format('7','百度','baidu')
mystring2 = template.format('8','腾讯','tencent')
print(mystring1)
print(mystring2)

输出：
编号：70000000    公司名称：百度    官网：http://www.baidu.com
编号：80000000    公司名称：腾讯    官网：http://www.tencent.com
```

4. 函数

4.1. 定义函数

使用 def 关键字定义函数，具体的语法格式如下：

```
def functionname([parameterlist])
    ["""comments"""]
    [functionbody]
```

functionname：函数名称
Parameterlist：用于指定向函数传递的参数，如果有多个参数，各参数间用逗号“，”隔开。如果不指定，表示函数没有参数。在调用时，也不用指定参数。
“””comments”””：表示为函数指定的注释，注释的内容通常是说明该函数的功能、参数的作用。
Functionbidy：函数体，函数调用后要执行的代码。如果函数有返回值，可以使用 return 语句返回。
注意：注释和函数体一定要与 def 保持一定的缩进

在定义函数的时候，可以给形式参数认定默认值。

```
def functionname([parameterlist, parameter1 = value1])
    ["""comments"""]
    [functionbody]
```

4.2. 调用函数

functionname([parametervalue])

parametervalue：用于指定向函数传递的参数，如果有多个参数，各参数间用逗号“，”隔开。如果函数没有参数，直接函数名后加“()”。

4.3. 参数传递

形式参数：在定义函数时，函数名后面括号中的参数。
实际参数：在调用函数时，函数名后面括号中的参数。

根据实际参数的类型不同，参数传递可以分为**值传递**和**引用传递**。当实际参数是不可变对象时，进行析是**值传递**；当实际参数为可变对象的时，进行的是**引用传递**；它们的区别就是，进行值传递后，改变形式参数的值，实际参数的值不改变。而进行引用传递后，改变形式参数的值，实际参数的值也会被改变

```
def test(testdata):
    print("原值：",testdata)
    testdata += testdata

mystring = "hello world"
test(mystring)
print('当前值：',mystring)

mylist = ['hello','world']
test(mylist)
print('当前值：',mylist)

输出：
原值： hello world
当前值： hello world
原值： ['hello', 'world']
当前值： ['hello', 'world', 'hello', 'world']
```

注意：
（1）在调用函数的时候，指定的实际参数的数量必须与形式参数的数量一致
（2）在调用函数的时候，指定的实际参数的位置也必须与形式参数的位置一致

关键字参数是指在使用形式参数据的名字来确定输入的参数值，通过该方式指定实际参数时，不再需要与形式参数的位置完全一致，只要将参数名写正确就可以了。

```
bmi(weight = 55 , height = 175)
```

调用函数时，如果没有指定某个参数将抛出异常，为了解决这个问题，我们可以为参数设置默认值，直接指定形式参数的默认值，这样，当没有传入参数时，则直接使用定义函数时设置的默认值。

语法如下：

def functionname(……parameter = “defaultValue”):

在 Python 中，还可以定义**可变参数**，可变参数也称为不定长参数，传入的实际参数可以是零个、一个、二个或者多个。主要有两种形式：*parameter 和**parameter。

（1）*parameter

这种形式表示接收任意多个实际参数并将其放置到一个元组中。

```
def testpara(* paralist):
    print(paralist)

testpara("Hello")
testpara("Hello","World")
testpara("Hello","World",1,2,3)
输出：
('Hello',)
('Hello', 'World')
('Hello', 'World', 1, 2, 3)
```

如果想要使用一个已经存在的列表作为函数的可变参数，可以在列表的名称前加*。

```
def testpara(* paralist):
    print(paralist)

mylist = ["Hello","World",1,2,3]
testpara(mylist)
testpara(* mylist)
输出：
(['Hello', 'World', 1, 2, 3],)
('Hello', 'World', 1, 2, 3)
```

（2）**parameter

这种形式表示接收任意多个类似关键字参数一样显式赋值的实际参数，并将其放置到一个字典中。

如果想要使用一个已经存在的字典作为函数的可变参数，可以在字典的名称前加“**”。

```
def testpara(** paralist):
    print(paralist)

testpara(一 = "张三",二 = "李四",三 = "王二",四 = "麻子",五 = "赵大")
```

```
mydict = {"一" : "张三","二" : "李四","三" : "王二","四" : "麻子","五" : "赵大"}
testpara(** mydict)
输出:
{'三': '王二', '一': '张三', '二': '李四', '五': '赵大', '四': '麻子'}
{'四': '麻子', '三': '王二', '一': '张三', '二': '李四', '五': '赵大'}
```

4.4. 返回值

在函数中可以用 return 语句为函数返回计算结果，返回值可以是任意类型，并且无论 return 语句出现在函数的什么位置，只要得到执行，就会直接结束函数的运行。

函数可以返回一个值，也可以返回多个值。

4.5. 变量的作用域

变量的作用域是指程序代码能够访问该变量的区域，如果超出该区域，再访问时就会出现错误。

在程序中，根据变量的有效范围可以将变量分为**局部变量**和**全局变量**

局部变量是指在函数内部定义并使用的变量，它只在函数内部有效，函数内部变量的名字只有函数运行时才会创建，在函数运行之前或者运行完成之后，这些名字是不存大的。在函数外部不能使用函数内部的变量。

如果一个变量在函数外部定义，那么不仅在函数外部可以访问到，在函数内部也可以访问得到，这个在函数外部定义的变量就是全局变量。

在函数体内定义的变量，并用使用 global 关键字修饰后，该变量也就变成了全局变量，在函数体外也能访问到该变量。

```
# 全局变量
myvariable =  100

def test():
    myvarible = 9   # 不影响全局变量赋值
    print("同名局部变量: ",myvariable)
    localvarible = 22
    print("局部变量: ",localvarible)

test()
print("全局变量: ", myvariable)
输出:
同名局部变量:  100
局部变量:  22
全局变量:  100
```

5. 面向对象

5.1. 对象与类

关于对象的理解很简单，在我们的身边，每一种事物的存在都是一种对象。总结为一句话也就是：对象就是事物存在的实体。下面举个简单的例子，比如人类就是一个对象，然而对象是有属性和方法的，那么身高，体重，年龄，姓名，性别这些是每个人都有的特征可以概括为属性，当然了我们还会思考，学习，这些行为相当于对象的方法。不过，不同的对象有不同的行为。

类是封装对象的属性和行为的载体，反过来说，具有相同属性和行为的一类实体被称为类。类是一个抽象的概念。

5.2. 面向对象的特点

5.2.1. 封装

封装隐藏了类的内部实现机制，可以在不影响使用的情况下改变类的内部结构，同时也保护了数据。对外界而已它的内部细节是隐藏的，暴露给外界的只是它的访问方法。

封装是面向对象编程的核心思想，将对象的属性和行为封装起来，而将对象的属性和行为封装起来的载体就是类。类通常对客户隐藏其实现细节，这个就是封装的思想。例如，用户使用计算机，只需要用手指敲敲键盘就可以实现一些功能，而无须知道计算机内部是如何工作的。

采用封装的思想保证了类内部数据结构的完整性，使用该类的用户不能直接看到类中的数据结构，而只能执行类允许公开的数据，这样就避免了外部对内部数据的影响，提高程序的可维护性。

5.2.2. 继承

继承是实现重复利用的重根手段，子类通过继承复用了父类的属性和行为的同时，又增加了子类特有的属性和行为。

矩形、菱形、平行四边形和梯形等都是四边形。因为四边形与它们具有共同的特征：拥有四条边。只要将四边形适当地延伸，就能得到它们。来平行四边形为例，如果把平行四边形看做是四边形的延伸，那么平行四边形就复用了四边形的属性和行为，同时增加了平行四边形特有的属性和行为，即平行四边形的对边相互平行且相等。四边形是一个类，那么平行四边形可以看成是继承四边形类的一个子类，将四边形类称之为平行四边形的父类或者是超类。

子类的实例都是父类的一个实例，可以说平行四边形是四边形，但不能说四边形都是平行四边形。

5.2.3. 多态

多态指的是一类事物有多种形态，（一个抽象类有多个子类，因而多态的概念依赖于继承），序列类型有多种形态：字符串，列表，元组，动物有多种形态：人，狗，猪

多态就是指程序中定义的引用变量所指向的具体类型和通过该引用变量发出的方法调用在编程时并不确定，而是在程序运行期间才确定，即一个引用变量倒底会指向哪个类的实例对象，该引用变量发出的方法调用到底是哪个类中实现的方法，必须在由程序运行期间才能决定。因为在程序运行时才确定具体的类，这样，不用修改源程序代码，就可以让引用变量绑定到各种不同的类实现上，从而导致该引用调用的具体方法随之改变，即不修改程序代码就可以改变程序运行时所绑定的具体代码，让程序可以选择多个运行状态，这就是多态性。

5.3. 类的定义

类的定义使用 class 关键字来实现，语法如下：

```
Class ClassName:
    """类的帮助信息"""    #类文档字符串
    stateMent                  #类体
```

className：用于指定类名，一般建议使用大写字母开头，如果类包含两个单词，则第二个单词的首字母也可以大写，这种命名方法也叫做驼峰式命名法

"""类的帮助信息"""：用于指定类的文档字符串，定义该字符串后，在创建类对象时，输出类名和左侧的括号，将显示该信息。

Statement：主要由类变量或类成员、方法和属性等定义语句组成。

5.4. 创建实例

定义类以后，并不会真正创建一个实例。这有点像一辆汽车的设计图，设计图可以告诉你汽车看上去怎么样，但设计图本身不是一辆汽车。你不能开走它，它只能用来制造真正的汽车。而且可以使用它制造很多的汽车。

创建类的实例的语法如下：

```
ClassName(parameterlist)
```

ClassName：用于指定类名

Parameterlist：是可选择参数，当创建一个类时，没有创建__init__()方法，或者该方法只有一个 self 参数时，可以省略。

5.4.1. __init__()方法

1、在创建类后，通常会创建一个__init__()方法，它是一特殊的方法，类似于 Java 语言中的构造方法。每创建一个类的新的实例的时候，都会自动执行它。

2、__init__()方法必须包括一个 self 参数，并且必须是第一个参数，self 参数是一个指向

实例本身的引用，用于访问类中的属性和方法。

3、在方法调用时，会自动传递实际参数 self。因此，当__init__()方法只有一个 self 参数时，就不需要指定实际参数了。

5.4.2. 实例方法

类的成员主要由实例方法和数据成员组成。在类中创建了类的成员后，可以通过类的实例进行访问。

所谓的实例方法，就是指在类中定义的函数。该函数是一种在类的实例上操作的函数。同__init__()方法一样，实例方法物质文明第一个参数必须是 self，并且必须得包含 self 参数。

创建实例方法的语法格式如下：

```
def functionName(self, parameterList)
    block
```

functionName：指定方法名，一般使用小写字母开头；
Self：必要参数，表示类的实例，其名称可以是 self 以外的单词
parameterList：用于指定除 self 以外的其它参数，各参数间用逗号隔开。
Blick：方法体

方法创建后，可以通过类的实例名和点（.）操作符进行访问。具体语法格式如下：

```
instanceName.functionName(parameterValue)
```

instanceName：实例名
functionName：指定方法名，一般使用小写字母开头；
parameterValue：用于指定除 self 以外的其它参数，各参数间用逗号隔开，个数必须与创建实例方法时的个数相同。

5.4.3. 数据成员

数据成员是指在类中定义的变量，即属性，根据定义的位置，又可以分类类属性和实例属性。

类属性是指在定义类中，并且在函数体外的属性。类属性可以在类中的所有实例中访问，也就是在所有的实例化对象中共用。

类属性可以通过类名称或者实例名称加点号（.）访问。

实例属性是指定义在类的方法中的属性，只作用于当前实例中。修改一个实例中的实例属性后，类的其它实例的该属性并不会改变。

5.5. 访问限制

在类的内部可以定义属性和方法，而在类的外部则可以直接调用属性和方法来操作数据，从而隐藏类内部的复杂逻辑。但是，Python 并没有对属性和方法的访问权限进行限制。为了保证类内部的某些属性或方法不被外部访问，可以在属性或方法名前面加单下划线

（_foo）、双下划线（__foo）、首尾双下划线（__foo__），从而限制访问权限。

1、_foo：以单下划线开头的表示 protected（保护）类型的成员，只允许类本身和子类进行要访问，保护属性可以通过实例名访问。不能使用 “from module import *”语句导入。

2、__foo：双下划线表示 private（私有）类型成员，只允许定义该方法的类本身进行访问，而且也不能通过类的实例进行访问，但是可以通过“类的实例名.类名__XXX”方式访问。

3、__foo__：首尾双下划线表示定义特殊方法，一般是系统定义名字，如__init__()

5.6. 属性

这里介绍的属性与前面介绍的类属性与成员属性不同，前面介绍的属性将返回所存储的值，这里的属性是一种特殊的属性，访问它将计算它的值。

在 Python 中，可以能过@property（装饰器）将一个方法转换成属性，从而实现用于计算的属性，将方法转换为属性后，可以直接通过方法名来访问方法，而不需要再添加一对小括号“()”，这样可以让代码更简洁。格式如下：

```
@property
def methodname(self):
    Block
```

注意：通过@property 转换后的属性不能赋值。

5.7. 继承

通过继承可以实现代码的复用，还可能通过继承理顺类与类之间的关系要，

在 Python 中，可以在类定义语句中，类名右侧使用一对小括号将要继承的基类名称括起来，从而实现类的继承。具体格式如下：

```
class ClassName(baseclasslist):
    “””类的帮助信息”””
    Statements
```

className：用于指定类名

Baseclasslist：用于指定基类，可以有多个，类名之间用逗号分隔。

Statements：类体，主要由类变量、类成员、方法和属性等定义语句组成。

5.8. 重写

基类的成员都会被派生类继承，当基类中的某个方法不完全适用于派生类时，就需要在派生类中重写父类的方法。

要让派生类调用基类的__init__()方法，需要在派生类使用 super()函数调用基本的__init__()方法。

6. 模块

在 Python 中，一个扩展名为“.py”的文件就是一个模块。

通常情况下，我们把能够实现某一特定功能的代码放在同一个文件中做一个模块，从而方便其它程序或者脚本导入并使用。使用模块也可以避免函数名和变量名冲突。

6.1. 自定义模块

自定义模块有两个作用，一个是规范代码，让代码更容易读，另一个是方便其它程序使用，重复利用已经编写得很好的代码，提高开发效率。

1、创建模块

创建模块就是将模块中相关的代码编写在一个单独的文件中，并且将该文件名命名为“模块名+.py”的形式。设置模块名的时候，不能用 Python 自带的标准模块名称。模块文件的扩展名必须是“.py”

2、导入模块

创建模块后，就可以在其它程序中使用该模块了，使用模块需要先以模块的形式加载模块中的代码，这可以使用 import 语句实现。Import 语句的基本语法：

```
import modulename [as alias]
```

modulename ：是要导入的模块的名称；

[as alias] ：给模块起的别名，通过别名也可以访问模块。

在调用模块中的变量、函数或者类时，需要在变量名、函数或者类名前添加“模块名.”作为前缀。

使用 import 语句可以一次导入多个模块，在导入多个模块时，模块名之间使用“，”隔开。但不建议这样做，最好一条导往个模块。

```
from modulename import member
```

Modulename：模块名称；

Member：用于指定要导入的变量、函数或者类。可以同时导入多个定义，各个定义之间使用“，”隔开，如果想导入全部定义，也可以使用通配符“*”代替。

在 import 导入模块时，每执行一条语句都会创建一个新命命空间，并且在该命名空间中执行配.py 文件相头的语句，所在有执行时，需要在具体的变量、函数和类名前加“模块名.”。而使用 from……import 语句导入模块后，不需要再添加前缀，直接通过具体的变量、函数和类名等访问即可。

6.2. 模块搜索目录

当使用 import 语句导入模块时，默认情况下，按照以下顺序进行行查找：

（1）在当前目录下查找；

（2）到 PYTHONPATH（环境变量）下的每个目录中查找；

（3）到 Python 的默认安装目录下查找

6.3. 常用标准模块

sys：与 Python 解释器及环境操作相关的标准库
time：提供与时间相关的各种函数标准库
os：提供了访问操作系统服务功能的标准库
calendar：提供与日期相关的各种函数的标准库
urllib：用于读取来自网上的数据的标准库
json：用于使用 json 序列化与反序列化对象
re：用于在字符串中执行正则表达式匹配和替换
math：提供标准算术运算函数的标准库
decimal：用于进行精确控制运算精度、有效数位和四舍五入操作的十进进制运算
shutil：用于进行高级文件操作，如复制、移动和重命名
logging：提供了灵活的记录事件、错误、警告和调试信息等日志信息的功能
tkinter：提代了 GUI 编程的标准库

6.4. 第三方模块

在使用第三方模块时，需要先下载并安装该模块，然后就可以使用标准模块一样导入并使用了。Pip 命令的语法格式如下：
pip <command> [modulename]

pip install numpy

6.5. 包

使用模块可以避免函数名和变量名重名引发冲突，那么，如果模块名重复应该怎么办呢？在 Python 中，提出了包的概念。

包是一个分层次的目录结构，它将一组功能相近的模块组织在一个目录下。这样，中以起到规范代码的作用，又能避免模块名重名引起的冲突。包的简单理解就是文件夹，只不过在该文件夹下面必须存在一个名称为“__init__.py”的文件。

创建包实际上就是建立一个文件夹，并用在该方件夹中创建一个名为“__init__.py”的 Python 文件，在__init__.py 文件中，可以不编写任何代码，也可以编写一些 Python 代码，其中的代码会在导入包时自动执行。

创建包以后，就可以在包中创建相应的模块，然后再使用 import 语句从包中加载模块，有 3 种方式：
（1）通过“import + 包名 + 模块名”形式加载指定模块，使用时要加包名.模块名缀。
（2）通过“from + 包名 + import +模块名”形式加载指定模块，使用时可以不用在包名前缀，但需要加包名。
（3）通过“from + 包名 + 模块名+ import + 定义名”形式加载指定模块，使用时可以不用

加前缀。

7. 异常处理

在程序运行的过程当中，经常会碰到各种各样的错误，这些错误统称为“异常”，这些错误有的是开发者一时疏忽将关键字敲错导致的，这类错误多数产生的是“SyntaxError：invalid syntax”（无效语法），这将导致程序不能运行，这类异常是显示的，在开发阶段很容易发现，还有一类是隐式的，通常跟使用都的操作有关。

1、常见异常

NameError：尝试访问一个没有声明的变量引发的错误
IndexError：索引超出序列范围引发的错误
IndentationError：缩进错误
ValueError：传入值错误
KeyError：请求一上不存在的字典关系字引发的错误
IOError：输出输去错误
ImportError：无法找到模块错误
AttributeError：尝试访问未知对象属性引发的错误
TypeError：类型不合适错误
MemoryError：内存不足
ZeroDivisionError：除数为 0 错误

在程序开发的时候，有些错误并不是每次运行都会出现的，但是在某些时候就会出现，导致程序异常并停止运行。在 Python 中，在开发程序时可以对可能出现异常的情况进行处理。

2、try……except 语句

在使用时，把可能产生异常的代码存在 try 语句块当中，把处理结果放在 except 语句块中，这样，当 try 语句块中的代码出现错误，就会执行 except 语句块中的代码，如果 try 语句块中没有错误，那么 except 语句块就不会执行。语法格式如下：

```
try：
    Block1
except [ExceptionName [as alias]]:
    Block2
```

ExceptionName [as alias]：可选参数，用于指定要捕捉的异常，其中，ExceptionName 表示要捕捉的异常名，如果在其右侧加上 as alias，则表示为当前的异常指定一个别名，通过该别名可以记录异常的具体内容。如果在 except 后面没有接异常名称，则表示接所有的异常。

使用 try except 语句捕获异常后，当程序出现错误，excepte 后面的程序处理后，程序会继续执行。

3、try……except……else 语句

如果在后面再添加一个 else 子句，用于指定当 try 语句块中没有发现异常时要执行的语句块。当 try 语句发现异常时，该语句块中的内容将不再执行。

4、try……except……finally 语句

完整的异常处理语句应该包含 finally 语句，表示无论程序中是否有异常产生，finally 代码中的代码都会被执行。

```
try：
    Block1
except [ExceptionName [as alias]]:
    Block2
finally：
    Block2
```

5、使用 raise 语句抛出异常

如果某个函数或者方法可能产生异常，但不想在当前函数或者方法中处理这个异常，则可以使用 raise 语句在函数中抛出这个异常。基本语法格式如下：

```
raise [ExceptionName [(reason)]]:
```

其中，ExceptionName [(reason)]为可选参数，用于指定抛出的异常，以及异常信息的相关描述。如果省略，就会把当前的异常原样抛出。Reason 也可以省略，如果省略，则在抛出异常时，不附带任何描述信息。

在应用 raise 抛出异常的时候，要尽量选择合理的异常对象，而不应该抛出一个与实际内容不相关的异常。

8. 多维数组

8.1. Numpy 模块

NumPy，是 Numerical Python 的简称，它是目前 Python 数值计算中最为重要的基础包，是 python 数值计算的基石，它提供了多种数据结构、算法以及大部分涉及 Phython 数值计算所需要的接口。NunPy 模块的核心就是基于数组的运算，相比于列表或其它数据结构，数组的效率是最高的。NumPy 的设计对含有大量数组的数据非常有效，它的方法比是 Python 方法要快 10 倍到 100 倍。大多数的计算包都提供了基于 NumPy 的科学函数功能，将 NumPy 的数组对象做为数据交换的数据容器。

NumPy 本身并不提供建模和数学函数，许多 python 的数值计算工具将 numpy 数组作为基础数据结构，或者与 Numpy 进行无缝互操作。理解 NumPy 的数组以及基于数组的计算将帮助你更高效地使用基于数组的工具，如 Pandas。虽然 NumPy 提供了操作数值数据的基础，但是大多数人还是想把 Pandas 做为统计分析的基石。

8.2. 数组创建

通过 numpy 模块中的 array()函数的实现数组的创建，如果向函数传入一个列表或者元

组，将构造简单的**一维数组**；如果向函数传入多个套嵌的列表或者数组，则可以构造出一个**二维数组**。

一个 ndarray 或者说是一个多维数组，它是一个通用的多维同类数据容器，构成数组的元素是同质的，数组中的每一个值都具有相同的数据类型。

一维数组：

```
# 导入模块，并重要命名为 np
import numpy as np
arr1 = np.array([1,2,3,4,5,6,7,8,9])
arr2 = np.array([1,2,3,4,5,6,7,8,9,'a'])
print("一维数组 arr1：",arr1)
print("一维数组 arr2：",arr2)
输出：
一维数组 arr1： [1 2 3 4 5 6 7 8 9]
<class 'numpy.ndarray'>
一维数组 arr2： ['1' '2' '3' '4' '5' '6' '7' '8' '9' 'a']
<class 'numpy.ndarray'>
```

二维数组：

```
arr2 = np.array([[11,12,13,14,15,16,17,18,19],
                 [21,22,23,24,25,26,27,27,29],
                 [31,32,33,34,35,36,37,38,39],
                 [41,42,43,44,45,46,47,48,49],
                 [51,52,53,54,55,56,57,58,59]])
print("二维数组 arr1：\n",arr2)
输出：
二维数组 arr1：
 [[11 12 13 14 15 16 17 18 19]
 [21 22 23 24 25 26 27 27 29]
 [31 32 33 34 35 36 37 38 39]
 [41 42 43 44 45 46 47 48 49]
 [51 52 53 54 55 56 57 58 59]]
<class 'numpy.ndarray'>
```

三维数组：

```
arr4 = np.array([[[1,2,3],[4,5,6],[7,8,9]],
                 [[10,11,12],[13,14,15],[16,17,18]],
                 [[19,20,21],[22,23,24],[25,26,27]],
                 [[28,29,30],[31,32,33],[34,35,36]],])
print("三维数组 arr1：\n",arr4)
print(type(arr4))
输出：
三维数组 arr1：
 [[[ 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 26 27]]

 [[28 29 30]
  [31 32 33]
  [34 35 36]]]
<class 'numpy.ndarray'>
```

8.3. 数组生成函数

array	将输入数据转换为 ndarray，可以是列表、元组、数组以及其它序列，默认复制所有的数据。
asarray	将输入转换为 ndarray，但如果输入已经是 ndarray 则不再复制
arange	Python 内建函数 range 的数组版，返回一个数组
ones	根据给定的形状和数据类型生成全 1 数组
ones_like	根据所给的数组生成一个一模一样的全 1 数组
zeros	根据给定的形状和数据类型生成全 0 数组
zeros_like	根据所给的数组生成一个一模一样的全 0 数组
empty	根据给定的形状生成一个没有初始化值的空数组
empty_like	根据所给的数组生成一个一模一样但是没有初始化值的空数组
full	根据给定的形状和数据类型生成指定数值的数组
full_like	根据给定的数组生成一模一样但是指定数值的数组
eye,identity	生成一个特征矩阵，对角线全是 1，其它都是 0

```
import numpy as np
data1 = np.zeros((3,6))
print(data1)

data2 = np.empty((2,3,3))
print(data2)

data3 = np.arange(10)
print(data3)
输出:
[[0. 0. 0. 0. 0. 0.]
```

```
 [0.  0.  0.  0.  0.  0.]
 [0.  0.  0.  0.  0.  0.]]
[[[0.  0.  0.]
  [0.  0.  0.]
  [0.  0.  0.]]

 [[0.  0.  0.]
  [0.  0.  0.]
  [0.  0.  0.]]]
[0 1 2 3 4 5 6 7 8 9]
```

8.4. 数据类型

数据类型，即 dtype，是一个特殊的对象，它包含了 ndarray 需要为某一种数据所申明的内块信息。

类型	代码	描述
int8、uint8	i1，u1	有符号和无符号的 8 位整数
int16、uint16	i2，u2	有符号和无符号的 16 位整数
int32、uint32	i4，u4	有符号和无符号的 32 位整数
int64、uint64	i8，u8	有符号和无符号的 64 位整数
float16	f2	半精度浮点型
float32	f4、f	标准单精度浮点型
float64	f8、d	标准双精度浮点型
float128	f16、g	拓展精度浮点型
complex64	c8	基于 32 位浮点数的复数
complex128	c16	基于 63 位浮点数的复数
complex256	c32	基于 128 位浮点数的复数
bool	?	布尔值
object	o	Python object 类型
string_	S	ASCII 字符串类型
unicode_	U	Unicode 类型

Numpy 包含的数据类型比较丰富，当需要转换数据格式时，可以使用 astype()函数。使用 astype()总是生成一个新的数组，尽管传入的 dtype 与之前的一样

```
import numpy as np
data = np.array([1,2,3,4,5,6,7,8,9],dtype = np.int64)
print("数据类型：",data.dtype)

floatdata = data.astype(np.float32)
print("数据类型：",floatdata.dtype)
```

```
strdata = np.array(['11','12','12.3','15','18.9','19.88'])
print("数据类型：",strdata.dtype)

numdata = strdata.astype(np.float32)
print("数据类型：",numdata.dtype)

输出：
数据类型：  int64
数据类型：  float32
数据类型：  <U5
数据类型：  float32
```

8.5. 数组访问

1、一维数组

（1）位置索引

使用位置索引可以实现数组元素的获取，列表中讲解过如何通过正向索引、负向索引、切片索引和无限索引获取元素。

（2）神奇索引

在一维数组中，列表的所有索引方法都可以用在数组上，而且还可以将任意位置的索引组装为列表，用作对元素的获取，这个也称之为**神奇索引**。神奇索引与其它的切片不同，它总是将数据复制到一个新的数组中。

2、二维数组

在二维数组中，位置索引必须写成**[rows, cols]**的形式，方括号前半部分用于控制行索引，后半部分用于控制列索引；如果需要获取所有的行或者列的元素，那么，对应的行索引或者列索引用冒号表示；

```
#一维数组元素的获取
print(arr1[[1,2,5,8]])

# 第二维数组元素的获取
# 第二行第三列
print(arr2[1,2])

# 第三行所有元素
print(arr2[2,:])

# 第三列所有元素
print(arr2[:,2])

# 第二行到第四行，第二到第八列
print(arr2[1:4,2:8])
```

上面程序输出：

```
[2 3 6 9]
23
[31 32 33 34 35 36 37 38 39]
[13 23 33 43 53]
[[23 24 25 26 27 27]
 [33 34 35 36 37 38]
 [43 44 45 46 47 48]]
```

3、如果要取二维数组的某几行，某几列，应该使用 ix_()函数。

```
# 第 1 行、第 3 行、最后一行，第二列、第三列、最后一列
print(arr2[np.ix_([0,2,-1],[1,2,-1])])

输出：
[[12 13 19]
 [32 33 39]
 [52 53 59]]
```

4、对切片的赋值，整个切片都会被重新赋值。

区别于 python 的内置列表，数组的切片是原数组的视图，这意味数据并不是被复制了，任何对于视图的修改都会反应到原数组上，这样设计非常适合处理非常大的数组。如果你还是想要一份数组切片的拷贝，而不是一份视图的话，你就必须显式地复制这个数组。如：Data[5:8].copy()

```
import numpy as np
data = np.array([1,2,3,4,5,6,7,8,9])
print("一维数组：",data)

dataslice = data[5:8]
dataslice[0:3] = 12
print("切片赋值后：",data)

slicecopy = data[5:8].copy()
slicecopy = 13
print("切片复制赋值后：",data)

输出：
一维数组： [1 2 3 4 5 6 7 8 9]
切片赋值后： [ 1  2  3  4  5 12 12 12  9]
切片复制赋值后： [ 1  2  3  4  5 12 12 12  9]
```

8.6. 数组属性

每个数组都有一个 shape 属性，用来表征数组每一维度的数量；

每个数组都有一个 dtype 属性，用来描述数组的数据类型；

```
print("二维数组的数据结构：",type(arr2))
print("二维数组的维数：",arr2.ndim)
print("二维数组的行列数：",arr2.shape)
print("二维数组的数据类型：",arr2.dtype)
print("二维数组的元素的个数：",arr2.size)

输出：
二维数组的数据结构： <class 'numpy.ndarray'>
二维数组的维数： 2
二维数组的行列数： (5, 9)
二维数组的数据类型： int32
二维数组的元素的个数： 45
```

8.7. 数组形状

数组形状处理的手段主要有 reshape()、resize()、ravel()、flatten()、vstack()、hstack()、row_stack()、colum_stack()。

reshap 和 resize 都是用来改变数组形状的方法，但是 reshape 方法只是返回改变形状后的预览，但是并未改变原来数组的形状，而 resize 方法会直接改变原数组的形状。

```
arr3 = np.array([[1,2,3],[4,5,6],[7,8,9],[10,11,12],[13,14,15],[16,17,18]])
print("数组形状：",arr3.shape)
print(arr3.reshape(2,9))
print("数组形状：",arr3.shape)
print(arr3.resize(2,9))
print("数组形状：",arr3.shape)

输出：
数组形状： (6, 3)
[[ 1  2  3  4  5  6  7  8  9]
 [10 11 12 13 14 15 16 17 18]]
数组形状： (6, 3)
None
数组形状： (2, 9)
```

如果需要将二维数组降维为一维数组，利用 ravel、flatten 和 reshape 方法可以解决。通过 flatten 方法实现的降维是复制，它的结果修改对原数据没有影响；ravel 与 reshape 返回的则是视图，通过对视图的修改会影响原数组。

```
arr4 = np.array([[1,2,3],[4,5,6],[7,8,9]])
print("原数组：\n",arr4)
print("数组降维 ravel：\n",arr4.ravel())
print("数组降维 flatten：\n",arr4.flatten())
print("数组降维 reshap：\n",arr4.reshape(-1))

print("数组降维 ravel：\n",arr4.ravel(order = 'F'))
```

```
print("数组降维 flatten: \n",arr4.flatten(order = 'F'))
print("数组降维 reshap: \n",arr4.reshape(-1,order = 'F'))

arr4.flatten()[0] = 99
print('flatten 方法: \n',arr4)
arr4.ravel()[1] = 88
print('ravel 方法: \n',arr4)
arr4.reshape(-1)[2] = 77
print('reshape 方法: \n',arr4)

输出:
原数组:
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
数组降维 ravel:
 [1 2 3 4 5 6 7 8 9]
数组降维 flatten:
 [1 2 3 4 5 6 7 8 9]
数组降维 reshap:
 [1 2 3 4 5 6 7 8 9]
数组降维 ravel:
 [1 4 7 2 5 8 3 6 9]
数组降维 flatten:
 [1 4 7 2 5 8 3 6 9]
数组降维 reshap:
 [1 4 7 2 5 8 3 6 9]
flatten 方法:
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
ravel 方法:
 [[ 1 88  3]
 [ 4  5  6]
 [ 7  8  9]]
reshape 方法:
 [[ 1 88 77]
 [ 4  5  6]
 [ 7  8  9]]
```

vstack()用于垂直方向的数组堆叠，其功能与 row_stack()函数一致，而 hstack()用于水平方向的数组合并，其功能与 column_stack()一致.如果是多个数组的垂直堆叠，必须保证每个数组的列数相同，如果将多个数水平堆叠，则必须保证所有数组的行数相同。

```
arr5 = np.array([11,22,33])
print("垂直合并数组，vstack：\n",np.vstack([arr4,arr5]))
print("垂直合并数组，row_stack：\n",np.row_stack([arr4,arr5]))

arr6 = np.array([[44],[55],[66]])
print("水平合并数组，hstack：\n",np.hstack([arr4,arr6]))
print("水平合并数组，column_stack：\n",np.column_stack([arr4,arr6]))

输出：
垂直合并数组，vstack：
 [[ 1 88 77]
 [ 4  5  6]
 [ 7  8  9]
 [11 22 33]]
垂直合并数组，row_stack：
 [[ 1 88 77]
 [ 4  5  6]
 [ 7  8  9]
 [11 22 33]]
水平合并数组，hstack：
 [[ 1 88 77 44]
 [ 4  5  6 55]
 [ 7  8  9 66]]
水平合并数组，column_stack：
 [[ 1 88 77 44]
 [ 4  5  6 55]
 [ 7  8  9 66]]
```

8.8. 排序

跟 python 列表类似，numpy 数组可以用数组的 sort 方法按位置排序；

可以在多维数组中根据传递的 axis 值，沿着轴向对每个一维数据段进行排序；

顶层的 np.sort()方法返回的是的是已经排序好的数组的拷贝，而不是对原来数组按位置排序。

8.9. 基本运算

Python 中的列表无法直接进行数学运算，一旦将列表转换为数组后，就可以实现各种常见的数学运算。如：四则运算、比较运算、广播运算。

8.9.1. 四则运算

在 numpy 模块中，它允许对数组进行批量操作而无需要任何 for 循环，这一操作也叫做向量化。任何两个尺寸相同的数组之间，都可以进行批量操作，不同尺寸的数组之间的操作，则需要用到广播特性。

实现四则运算可以使用符号，也可以使用函数，四则运算的符号是“+、-、*、/”，对应的模块分别是 np.add()、np.subtruct()、np.multiply()和 np.devide()。

必须保证操作数组具有相同的形状。

另外，还有“%、//、**”，也可使用 np.fmod()、np.modf()和 np.power()

```
arr7 = np.array([[1,2,3],[4,5,6],[7,8,9]])
arr8 = np.array([[11,22,33],[44,55,66],[77,88,99]])

print("加法：\n",arr7 + arr8)
print("减法：\n",arr8 - arr7)
print("乘法：\n",arr7 * arr8)
print("除法：\n",arr8 / arr7)

输出：
加法：
 [[ 12  24  36]
 [ 48  60  72]
 [ 84  96 108]]
减法：
 [[10 20 30]
 [40 50 60]
 [70 80 90]]
乘法：
 [[ 11  44  99]
 [176 275 396]
 [539 704 891]]
除法：
 [[11. 11. 11.]
 [11. 11. 11.]
 [11. 11. 11.]]
```

8.9.2. 比较运算

比较运算主要有六种“>、<、<=、>=、==、!=”用在从数组中查询满足条件的元素，另一个是根据结果执行的不同的操作。

numpy.where 函数是三元表达式 x if condition else y 的向量化版本。它有三个参数，分别是 condition、x、y，可以都是数组，第二个参数和第三个参数并不需要是数组，它们可以是标量。

```
# 取子集
# 取出 arr7 中大于 arr8 的所有元素
print("满足条件的二维数组元素：\n",arr7[arr7 < arr8])
# 从 arr1 中取出大于 3 的元素
print("满足条件的一维数组元素：\n",arr1[arr1 > 3])

# 判断操作
# 将 arr7 中大于 7 的元素改成 5，其它的不变
print("二维数组的操作：\n",np.where(arr7<7, 5, arr7))
# 将 arr1 中大于 7 的元素改成 5，其它的不变
print("一维数组的操作：\n",np.where(arr1<5, 1, 0))

输出：
满足条件的二维数组元素：
 [1 2 3 4 5 6 7 8 9]
满足条件的一维数组元素：
 [4 5 6 7 8 9]
二维数组的操作：
 [[5 5 5]
 [5 5 5]
 [7 8 9]]
一维数组的操作：
 [1 1 1 1 0 0 0 0 0]
```

8.9.3. 广播运算

8.10. 常用函数

8.10.1. 数学函数

np.pi	常数 π
np.e	常数 e
np.fabs(arr)	计算各元素的浮点型绝对值
np.ceil(arr)	各元素向上取整
np.floor(arr)	各元素向下取整
np.round(arr)	各元素四舍五入
np.fmod(arr1,arr2)	计算 arr1/arr2 的余数
np.modf(arr)	返回数组元素的整数部分和小数部分

np.sqrt(arr)	计算种元素的算术平方根
np.square(arr)	计算各元素的平方值
np.exp(arr)	计算以 e 为底的指数
np.power(arr)	计算各元素的指数
np.log2(arr)	计算以 2 为底的各元素的对数
np.log10(arr)	计算以 10 为底的各元素的对数

8.10.2. 统计函数

Np.min(arr,axis)	按着轴的方向计算最小值
Np.max(arr,axis)	按着轴的方向计算最大值
Np.mean(arr,axis)	按着轴的方向计算均值
Np.median(arr,axis)	按着轴的方向计算中位数
Np.sum(arr,axis)	按着轴的方向计算和
Np.std(arr,axis)	按着轴的方向计算标准差
Np.var(arr,axis)	按着轴的方向计算方差
Np.cumsum(arr,axis)	按着轴的方向计算累计和
Np.cumprod(arr,axis)	按着轴的方向计算累计乘积
Np.argmin(arr,axis)	按着轴的方向返回最小值所在的位置
Np.argmax(arr,axis)	按着轴的方向返回最大值所在的位置
Np.cov(arr)	计算协方差矩阵

这些统计函数中的 axis 参数，表示在统计数组元素时需要按照不同的轴方向计算，如果 axis=1，则表示按水平方向计算统计值，即计算每一行的统计值；如果 axis=0，则表示按垂直方向计算统计值，即计算每一列的统计值。

```
import numpy as np
data = np.array([[1,2,3],[4,5,6],[7,8,9]])

print(data)
print('垂直计算：\n',np.sum(data,axis = 1))
print('水平计算：\n',np.sum(data,axis = 0))

输出：
[[1 2 3]
 [4 5 6]
 [7 8 9]]
垂直计算：
 [ 6 15 24]
水平计算：
 [12 15 18]
```

8.11. 线性代数

8.12. 随机模块

Numpy.random 模块填补了 Python 内建的 random 模块的不足，可以高效地生成多种概率分布下的完整样本值数组。

产生的随机数都是伪随机数，因为它们是由具有确定的算法根据随机数生成器中的随机数种子生成的。

Numpy.random 中的数据生成函数公用了一个全局的随机数种子，为了避免全局状态，你可以使用 numpy.random.RandomState()生成一个随机数生成器，使数据独立于其它的随机数。

seed	向随机数生成器传递随机状态种子
permutation	返回一个序列的随机排列
shuffle	随机排列一个序列
rand	从均匀分布中抽取样本
randint	根据给定的由低到高的范围生成抽取随机整数
randn	从均值 0 方差 1 的正态分布中抽取样本
binomal	从二项分布中抽取样本
beta	从 beta 分布中抽取样本
normal	从正态分布中抽取样本
uniform	从均匀(0,1)分布中抽取样样本

9. 数据处理

9.1. Pandas

Pandas 是 Python 的一个数据分析包，它最初是被作为金融数据分析工具而开发出来的，所以，它对时间序列提供了非常好的支持。它所包含的数据结构和数据处理工具的设计使得在 Python 中进行数据清洗和分析非常的方便，Padas 做为强大的数据处理模块，它可以帮助数据分析师轻松地解决数据的预处理问题，如数据类型的转换、缺失值处理、描述性统计分析、数据汇总等。

Pandas 提供了高级数据结构和函数，这些数据结构和函数的设计使得利用结构化、表格化数据的工作快速、简单、有表现力。它将表格和关系型数据库的灵活数据操作能力与 numpy 的高性能数组计算的理念相结合，它提供了复杂的索引函数，使得数组重组、切块、

切片、聚合、子集选择更为容易。

Pandas 经常是和其它的数值计算工具结合使用，如 numpy、Scipy 以及数据可视化工 matplotlib。Pandas 采用了很多的 numpy 代码风格，不同的是 Pandas 是用来处理表格型或者异质型数据的，而 numpy 则是更适合用来处理同质型的数值类型数据。

Pandas 的数据结构是 **Series**、**DataFrame**、**Panel**。

（1）Series：一维数组系列，也可称序列，与 Numpy 中的一维数组类似，二者也与 Python 中基本的数据结构 list 相近。用于存储一行或一列的数据，以及与之相关的索引的集合。

（2）DataFrame：二维的表格型数据结构，用于存存储多行多列的数据集合，可以将 DataFrame 理解为 Series 的容器。类似于 excel 的二维表格。

（3）Panel：三维数组，可以理解为 DataFrame 的容器。

序列可以理解为数据集中的一个字段，数据框是指含有至少两个字段。

9.2. 序列

Series 对象本质上是一个 numpy 数组，因此，numpy 数组处理函数可以直接对 Series 进行处理。它是一种一维的数组型型对象，它包含一个值序列，并且包含了一个值序列，并且包含了数据标签，称之为索引。所以，Series 除了可以使用位置作为下标存取元素之外，还可以使用标签存取元素，这一点和字典类似。每一个 Series 对象实际上都由下面两个数组组成：

（1）index：它是从 numpy 数组继承的 index 对象，保存标签信息；

（2）values：保存值的数组。

注意：

（1）Series 是一种类似于一维数组的对象；

（2）它的数据类型没有限制，可以保存多种数据类型；

（3）它有索引，把索引当做数据的标签看看待，类似于字典；

（4）它同时具有数组和字典的功能，因此也支持字典的一些方法。

如果需要对序列进行数学函数的运算，一般选用 numpy 模块，pandas 模块在这方面比较缺乏。如果要对序列做统计运算，既可以用 numpy 模块，也可以用序列的方法，而且，序列的方法更加丰富。

9.2.1. 构建

可以用以下的方式构建一个序列：

序列也可以使用以下语法定义，默认的索引是 0,1,2,3,4,5……：

Series([dtat1, data2, data3,……], index = [index1, index2, index3,……])

（1）通过同质的列表或者元组构建；

（2）通过字典构建；

（3）通过 Numpy 中的一维数组构建；

（4）通过 DataFrame 中的某一列构建；

```
import numpy as np
import pandas as pd

myseries1 = pd.Series([2.8,3.01,8.99,8.59,5.18])
myseries2 = pd.Series({'北京':2.8,'上海':3.01,'广东':8.99,'江苏':8.59,'浙江
':5.18})

print(myseries1)
print(myseries2)

输出:
0    2.80
1    3.01
2    8.99
3    8.59
4    5.18
dtype: float64
上海    3.01
北京    2.80
广东    8.99
江苏    8.59
浙江    5.18
dtype: float64
```

序列的打印结果有两列，第一列属于序列的行索引，可理解为行号，自动从 0 开始，第二列才是序列的实际值。通过字典构建的序列，它的第一列不再是行号，而是具体的行名称，对应到字典中的键，第二列是序列的实际值，对应到字典中的值。

我们可以通过序列的索引属性 indexs 和 values 属性分别获取序列对象的索引和值。

从另外一个角度考虑 Series，可以认为它是一个长度固定且有序的字典，因为它将索引和数据值按位置配对，在可能会使用字典的上下文当中，也可以使用 Series。

Series 对象本身及其索引 index 都有 name 属性。

```
import pandas as pd

data = pd.Series(['AA','BB','CC','DD'],index =['a','b','c','d'])
data.name = "testseries"
data.index.name = 'label'
print(data)

输出:
label
a    AA
b    BB
```

```
c    CC
d    DD
Name: testseries, dtype: object
```

9.2.2. 访问

序列与一维数组有极高的相似性，获取一维数组元素的所有索引方法都可以应用在序列上，而且数组的数学和统计函数也同样可以应用到序列上，而且，序列还有更多的其它处理方法。

```
print('行号风格的序列：\n',myseries1[[0,3,4]])
print('行名称风格的序列：\n',myseries2[[0,3,4]])
print('行名称风格的序列：\n',myseries2[['上海','江苏','浙江']])
print('通过 numpy 方法：\n',np.log(myseries1))
print('通过 numpy 方法：\n',np.mean(myseries1))
print('通过序列的方法：\n',myseries1.mean())

输出：
行号风格的序列：
 0    2.80
3    8.59
4    5.18
dtype: float64
行名称风格的序列：
 上海    3.01
江苏    8.59
浙江    5.18
dtype: float64
行名称风格的序列：
 上海    3.01
江苏    8.59
浙江    5.18
dtype: float64
通过 numpy 方法：
 0    1.029619
1    1.101940
2    2.196113
3    2.150599
4    1.644805
dtype: float64
通过 numpy 方法：
 5.714
通过序列的方法：
 5.714
```

9.2.3. 追加

序列不可以追加单个元素，但可以追加序列。

```
import pandas as pd
myseries1 = pd.Series(['一','二','三','四','五'],index = [1,2,3,4,5])
# 不能追加单个元素
# myseries1.append('六')

myseries2 = pd.Series(['六'])
myseries3 = myseries1.append(myseries2)
print(myseries3)

输出:
1    一
2    二
3    三
4    四
5    五
0    六
dtype: object
```

9.2.4. 删除

通过索引名或者索引号删除，删除多个条目可以把索引放在一个列表当中。

```
import pandas as pd
myseries1 = pd.Series(['一','二','三','四','五'],index =
['first','second','third','forth','fifth'])
myseries2 = pd.Series(['一','二','三','四','五'])
print(myseries1)

myseries3 = myseries1.drop('second')
print(myseries3)

myseries4 = myseries2.drop(0)
print(myseries4)
```

9.2.5. 更新

访问直接赋值。

9.2.6. 排序

Series 的 sort_index(ascending=True)方法可以对 index 进行排序，ascending 参数用于控制升序或降序，默认为升序。

9.3. 数据框

9.3.1. 构建

数据框实际上是一个数据集，它的行代表一条观测，数据集的列代表各个变量，在一个数据框中可以存放不同的数据类型，而数组和序列没有这样的优势，它们只能存放同质的数据。

构造一个数据框可以用如下的方式：

（1）通过套嵌列表或元组构造；

（2）通过字典构造；

（3）通过二维数组构造；

（4）通过导入外部数据构造；

```
import numpy as np
import pandas as pd

mydata1 = pd.DataFrame([['张三',23,'男'],['李四',21,'女'],['王二',26,'女']])
mydata2 = pd.DataFrame({'姓名':['张三','李四','王二'],
                        '年龄':[23,21,26],
                        '性别':['男','女','女']})
mydata3 = pd.DataFrame(np.array([['张三',23,'男'],['李四',21,'女'],['王二',26,'
女']]))

print('套嵌列表构造：\n',mydata1)
print('字典构造：\n',mydata2)
print('二维数组构造：\n',mydata3)

输出：
套嵌列表构造：
     0   1  2
0  张三  23  男
1  李四  21  女
2  王二  26  女
字典构造：
    姓名  年龄 性别
0  张三  23  男
1  李四  21  女
2  王二  26  女
```

```
二维数组构造：
     0   1   2
0  张三  23   男
1  李四  21   女
2  王二  26   女
```

构建的 DataFrame 会为自动为 Series 分配索引，并且列会按着排序的顺序排列。DataFrame 也可能通过 index 来设置各行的索引。

如果 DataFrame 的索引和列拥有 name 属性，它也是一个 values 属性和 columns 属性

9.3.2. 访问

访问列：变量名[列名]
访问行：变量名[n: m]
访问块：变量名.loc[[行名],[列名]]
　　变量名.iloc[n1: n2,m1: m2]
　　变量名.iloc[[],[]]
指定位置：变量名.at[行名，列名]

```
import numpy as np
import pandas as pd

data = [['张三',23,'男',55],['李四',21,'女',60],['王二',26,'女',65],['赵大',26,'女',68]]
mydata1 = pd.DataFrame(data,index = ['a','b','c','d'],columns = ['name','age','sex','weight'])
print('原数据：\n',mydata1)
print("访问单列：\n",mydata1['name'])
print("访问多列：\n",mydata1[['name','age']])
print("访问行：\n",mydata1[0:2])
print("loc 访问块：\n",mydata1.loc[['a','b'] ,['name','weight']])
print("iloc 访问块 1：\n",mydata1.iloc[0:2 ,1:4])
print("iloc 访问块 2：\n",mydata1.iloc[[0,1,3] ,[1,3]])
print("访问指定位置：\n",mydata1.at['a','name'])

输出
原数据：
   name  age sex  weight
a   张三   23   男      55
b   李四   21   女      60
c   王二   26   女      65
d   赵大   26   女      68
访问单列：
 a    张三
```

```
b    李四
c    王二
d    赵大
Name: name, dtype: object
访问多列：
   name  age
a   张三   23
b   李四   21
c   王二   26
d   赵大   26
访问行：
   name  age sex  weight
a   张三   23   男      55
b   李四   21   女      60
loc 访问块：
   name  weight
a   张三      55
b   李四      60
iloc 访问块 1：
    age sex  weight
a   23   男      55
b   21   女      60
iloc 访问块 2：
    age  weight
a   23      55
b   21      60
d   26      68
访问指定位置：
 张三
```

9.3.3. 增加

1、增加列

如果被赋值的列不存在，则会生成一个新的列。

```
mydata1['phone'] = ['131********','132********','133********','134********']
```

2、增加行

```
mydata1.loc['e'] = ['尹红',26,'女',68]
```

3、append 方法

append 方法可以添加数据到一个 dataframe 中，注意 append 方法不会影响原来的

dataframe，会返回一个新的 dataframe。

语法：

DataFrame.append(otherData, ignore_index=False, verify_integrity=False)

其中 otherData 参数是要添加的新数据，支持多种格式。

ignore_index 参数默认值为 False，如果为 True，会对新生成的 dataframe 使用新的索引（自动产生），忽略原来数据的索引。

verify_integrity 参数默认值为 False，如果为 True，当 ignore_index 为 False 时，会检查添加的数据索引是否冲突，如果冲突，则会添加失败。

```
data2 = [['张三 1',23,'男',55],['李四 1',21,'女',60],['王二 1',26,'女',65],['赵大
1',26,'女',68]]
mydata2 = pd.DataFrame(data2,index = ['f','g','h','i'],columns =
['name','age','sex','weight'])
print(mydata2)
mydata3 = mydata1.append(mydata2)
mydata4 = mydata1.append(mydata2.loc['h'])
print(mydata3)
print(mydata4)
```

9.3.4. 删除

可以利用 drop 方法根据行标签删除值，如果删除多个，可以把行标签放在一个列表中。

也可以能过传递 axis=1 或者 axis='columns'从删除列。del 方法可以用于移除 DataFrame 的列，del frame[name]。

```
import numpy as np
import pandas as pd

data = [['张三',23,'男'],['李四',21,'女'],['王二',26,'女']]
mydata1 = pd.DataFrame(data,index = ['d','c','a'],columns = ['name','age','sex'])
print('原数据: \n',mydata1)

mydata2 = mydata1.drop('a')
print('删除行 a 数据: \n',mydata2)
print('原数据: \n',mydata1)

mydata3 = mydata1.drop('sex',axis = 1)
print('删除性别数据: \n',mydata3)
print('原数据: \n',mydata1)

del mydata1['age']
print('删除年龄数据: \n',mydata1)
```

```
输出：
原数据：
   name  age sex
d   张三   23   男
c   李四   21   女
a   王二   26   女
删除行 a 数据：
   name  age sex
d   张三   23   男
c   李四   21   女
原数据：
   name  age sex
d   张三   23   男
c   李四   21   女
a   王二   26   女
删除性别数据：
   name  age
d   张三   23
c   李四   21
a   王二   26
原数据：
   name  age sex
d   张三   23   男
c   李四   21   女
a   王二   26   女
删除年龄数据：
   name sex
d   张三   男
c   李四   女
a   王二   女
```

9.3.5. 更新

1、修改指定行或单元格数据

```
#其中 i 是行号，j 是列号，都是从 0 开始
df.iloc[i][j]= xxx
#  这样把指定列的所有数据都设置为同一个值。注如果指定的列不存在，会新增加列。
df['a'] = 12
```

DataFrame 中选择的列是数据的视图，不是拷贝，如果需要复制，则应当显式地使用 Series 的 copy 方法。

9.3.6. 显示数据

注意，head 和 tail 返回的是一个新的 dataframe，与原来的无关 。

```
mydata.head(n)  # 获取 mydata 中的前 n 行数据，n 不指定，则默认为 5
mydata.tail(n)  # 获取 mydata 中的后 n 行数据，n 不指定，则默认为 5
```

9.4. 基本操作

9.4.1. 重建索引

1、Series

可以使用 reindex 方法重新排序，如果调用 reindex 重排数据，使得它符合新的索引，如果索引值不存在，就会引入缺失数据值。

```
import pandas as pd

data1 = pd.Series([1.1,2.2,3.3,4.4,5.5],index = ['b','a','c','e','d'])
print(data1)

data2 = data1.reindex(['a','b','c','d','e','f'])
print(data2)

输出:
b    1.1
a    2.2
c    3.3
e    4.4
d    5.5
dtype: float64
a    2.2
b    1.1
c    3.3
d    5.5
e    4.4
f    NaN
dtype: float64
```

2、DataFrame

Reindex 可以改变行索引，列索引，也可以同时改变二者。当仅传入一个序列时，结果中的行会重建索引。

```
import numpy as np
import pandas as pd
```

```
data = [['张三',23,'男'],['李四',21,'女'],['王二',26,'女']]
mydata1 = pd.DataFrame(data,index = ['d','c','a'],columns = ['name','age','sex'])
print('原数据: \n',mydata1)

mydata2 = mydata1.reindex(['a','b','c','d'])
print('index 数据: \n',mydata2)

state = ['sex','name','age']
mydata3 = mydata1.reindex(columns = state)
print('columhs 数据: \n',mydata3)

mydata4 = mydata1.loc[['a','c','d'],state]
print('loc 数据: \n',mydata4)

输出:
原数据:
   name  age sex
d   张三   23   男
c   李四   21   女
a   王二   26   女
index 数据:
   name   age  sex
a   王二  26.0    女
b  NaN   NaN  NaN
c   李四  21.0    女
d   张三  23.0    男
columhs 数据:
   sex name  age
d   男   张三   23
c   女   李四   21
a   女   王二   26
loc 数据:
   sex name  age
a   女   王二   26
c   女   李四   21
d   男   张三   23
```

9.5. 数据导入

9.5.1. 文本数据

1、txt 文件的读取

pd.read_table()

函数的参数：

filepath_or_buffer：指定 txt 文件或者 csv 文件所在的路径；

sep：原数据集中各字段之间的分隔符，默认为 Tab 制表符；

header：是否需要将原数据集中的第一行做为表头，默认第一行用作字段名称；

names：如果原数据集没有字段，可以通过该打参数在数据集读取时给数据框添加具体的表头；

index_col：指定原数据集中的某些列作为数据框的行索引；

usecols：指定需要读取原数据集中的哪些变量名；

dtype：读取数据时，可以为原数据集的第个字段设置不同的数据类形；

converters：通过字典格式，为数据集中的某些字字段设置转换函数；

skiprows：读取数据时，指定需要跳过原数据集开头的行数；

skipfooter：读取数据时，指定需要跳过原数据集末尾的行数；

nrows：指定读取数据的行数；

na_values：指定原数据集中哪些特征的值做为缺省值；

skip_blank_lines：读取数据时是否需要跳过空白行，默认为 True；

parse_dates：如果参数值为 true，则尝试解析数据框的行索引；如果参数为列表，则尝试解析对应的日期列；如果参数是套嵌列表，则将某些列合并为日期列；如果参数是字典，则解析对应的列，并生成字段名；

thousands：指定千分位符；

comment：指定注释符；

encoding：如果文件中含有中文，有时需要指定字符编码；

```
from pandas import read_table
data = read_table(r'E:\TestData\testData.txt',sep= ',')
print(data.head())
print(data.columns)
```

2、CSV 文件的读取

pd.read_csv()

参数同上。

```
from pandas import read_csv
data = read_csv(r'E:\TestData\testData.csv')
print(data.head())
print(data.columns)
```

9.5.2. 电子表格数据

pd.read_excel()
函数的参数：
io：电子表格的具体路径；
sheetname：指定需要读取电子表格中的第几个 sheet，既可以传递整数值，也可以传递名称；
header：是否需要将原数据集中的第一行做为表头，默认第一行用作字段名称；
skiprows：读取数据时，指定需要跳过原数据集开头的行数；
skipfooter：读取数据时，指定需要跳过原数据集末尾的行数；
index_col：指定原数据集中的某些列作为数据框的行索引；
names：如果原数据集没有字段，可以通过该打参数在数据集读取时给数据框添加具体的表头；
parse_cols：指定需要解析的字段；
na_values：指定原数据集中哪些特征的值做为缺省值；
parse_dates：如果参数值为 true，则尝试解析数据框的行索引；如果参数为列表，则尝试解析对应的日期列；如果参数是套嵌列表，则将某些列合并为日期列；如果参数是字典，则解析对应的列，并生成字段名；
thousands：指定千分位符；
converters：通过字典格式，为数据集中的某些字字段设置转换函数；
convert_float：默认所有的数值内型都转换成浮点型。

```
from pandas import read_excel
data = read_excel(r'E:\TestData\testData.xls')
print(data.head())
print(data.columns)
```

9.5.3. 数据库数据

9.6. 数据导出

1、导出 CSV 文件
to_csv(file_path, sep = ”,” , index = True, header = True)
file_path：文件路径；
sep 为分隔符，默认是逗号；
index 表示是否导出行号，默认是 True；
header：表示是否导出列名，默认是 True；

2、导出 Excel 文件
to_excel(file_path, index = True, header = True)
file_path：文件路径；

index 表示是否导出行号，默认是 True；

header：表示是否导出列名，默认是 True；

```
import pandas as pd
import random

familyName = ['李','王','张','刘','陈','杨','黄','赵','周','吴',
        '徐','孙','朱','马','胡','郭','林','何','高','梁',
        '郑','罗','宋','谢','唐','韩','曹','许','徐','萧',
        '曾','程','彭','袁','潘','董','苏','叶','吕','魏',
        '蒋','田','杜','丁','沈','江','傅','汪','钟','崔',
        '任','陆','方','金','邱','姜','韦','雷','侯','白',
        '段','龙','薛','秦','孟','熊','贾','谭','夏','赫',
        '孔','邵','史','毛','常','万','顾','贺','铁','钱',
        '尹','严','施','牛','洪','汤','陶','温','易','莫',
        '樊','乔','文','安','颜','岳','阳','向','申','庞',
        '庄','章','邢','齐','蓝','俞','翟','葛','聂','区']
nameFirst1 = ['一','是','在','不','了','有','和','中','大','来',
              '为','上','个','国','时','用','生','作','地','于',
              '分','成','可','发','年','动','同','工','能','子',
              '方','多','定','行','学','法','所','民','得','经',
              '进','部','莹','雪','琳','晗','涵','晴','丽','美',
              '瑶','梦','茜','倩','希','梅','月','悦','乐','彤',
              '珍','依','沫','玉','灵','理','起','小','实','机',
              '点','业','本','好','黻','合','因','电','然','政',
              '那','社','义','平','形','相','全','表','间','样',
              '关','各','重','新','正','心','明','原','利','比',
              '质','气','道','变','意','建','月','公','军','立',
              '想','通','并','展','果','象','员','革','文','式',
              '长','及','根','运','治','北','志','艺','蕊','花']
nameFirst2 = ['一','是','在','不','了','有','和','中','大','来',
              '为','上','个','国','时','用','生','作','地','于',
              '分','成','可','发','年','动','同','工','能','子',
              '方','多','定','行','学','法','所','民','得','经',
              '进','部','莹','雪','琳','晗','涵','晴','丽','美',
              '瑶','梦','茜','倩','希','梅','月','悦','乐','彤',
              '珍','依','沫','玉','灵','理','起','小','实','机',
              '点','业','本','好','黻','合','因','电','然','政',
              '那','社','义','平','形','相','全','表','间','样',
              '关','各','重','新','正','心','明','原','利','比',
              '质','气','道','变','意','建','月','公','军','立',
              '想','通','并','展','果','象','员','革','文','式',
              '长','及','根','运','治','北','志','艺','蕊','花']

data = pd.DataFrame({'姓名':[familyName[random.randint(0,109)]
```

```
                            +nameFirst1[random.randint(0,129)]
                            +nameFirst2[random.randint(0,129)] for i in
range(1000)],
                    '学号':[i+100000 for i in range(1000)],
                    '性别':[random.randint(0,1) for i in range(1000)],
                    '语文':[random.randint(65,95) for i in range(1000)],
                    '数学': [random.randint(45, 100) for i in range(1000)],
                    '英语': [random.randint(59, 98) for i in range(1000)],
                    '物理': [random.randint(45, 100) for i in range(1000)],
                    '化学': [random.randint(55, 100) for i in range(1000)],
                    '地理': [random.randint(70, 100) for i in range(1000)],
                    '生物': [random.randint(65, 100) for i in range(1000)],
                    '历史':[random.randint(75,95) for i in range(1000)],
                    'phone1':[130+random.randint(0,10) for i in range(1000)],
                    'phone2':[1000+random.randint(0,8999) for i in range(1000)],
                    'phone3':[1000+random.randint(0,8999) for i in range(1000)],
                    '班级':[i%20 + 1 for i in range(1000)]})

data1 = data.astype(str)
tel = data1.phone1 + data1.phone2 + data1.phone3
data['手机'] = tel
del data['phone1']
del data['phone2']
del data['phone3']
'''
mydata1 = data.drop('phone1',axis = 1)
mydata2 = mydata1.drop('phone2',axis = 1)
mydata3 = mydata2.drop('phone3',axis = 1)
'''
data.to_csv(r'E:\TestData\testData.csv')
data.to_excel(r'E:\TestData\testData.xls')
```

9.7. 类型转换

9.8. 数据清洗

在数据分析时，海量的原始数据存在着大量的不完整、不一致、有异常的数据，严重影响到数据分析的结果，所以进行数据清洗显得尤为重要。数据清洗是数据价值链中最关键的步骤。垃圾数据，即使是通过最好的分析，也将产生错误的结果，并误导业务本身。因此，在数据分析过程中，数据清洗占据很大的工作量。

数据清洗就是处理缺失数据以及清除无意义的信息，如删除原始数据集中的无关数据、重复数据，平滑噪声数据，筛选掉与分析主题无关的数据，处理缺失值、异常值。

9.8.1. 重复值处理

Pandas 模块中去掉重复数据的步骤如下：

（1）利用 DataFrame 中的 duplicated 方法返回一个布尔型的 Series，显示是否有重复行，没有重复行显示为 Fasle，有重复行的则从重复的第二行起显示为 True。

（2）再利用 DataFrame 中的 drop_duplicates 方法返回一个移除了重复行的 DataFrame。

duplicated 方法的格式如下：

duplicated(self, subset = None, keep = 'first')

参数说明：

Subset：用于识别重复的列标签或列标签序列；

Keep = 'first'：表示除第一次出现外，其余相同的数据被标记为重复；

Keep = 'last'：表示除最后一次出现外，其余相同的数据被标记为重复；

Keep = False：表示相同的数据被标记为重复；

```
import numpy as np
import pandas as pd

data = [['张三',23,'男',55],['李四',21,'女',60],['王二',26,'女',65],['张三',26,'女',68]]
mydata1 = pd.DataFrame(data,index = ['a','b','c','d'],columns = ['name','age','sex','weight'])

print(mydata1.duplicated('name'))
print(mydata1.drop_duplicates('name'))
输出：
a    False
b    False
c    False
d     True
dtype: bool
  name  age sex  weight
a   张三   23   男      55
b   李四   21   女      60
c   王二   26   女      65
```

9.8.2. 缺失值处理

从统计上说，缺失的数据可能会产生有偏估计，从而使样本数据不能很好地代表总体，而现实中的绝大部分数据都包含缺失值，因此，处理缺失值非常的重要。

处理缺失值分为两个步骤，缺失数据的识别和缺失数据的处理。

1、缺失数据的识别

Pandas 使用浮点值 NaN 表示浮点和非浮点数组里的缺失数据，并使用 isnull()方法，和 notnull()方法来判断缺失情况。

```
from pandas import read_table

data = read_table(r'E:\TestData\grade.txt',sep= ',')
print("原数据: \n",data.head())
print("isnull 输出: \n",data.isnull())
print("notnull 输出: \n",data.notnull())
```

```
原数据:
姓名      学号  性别    数学    物理  班级  生物    英语  语文    手机
0  翟工涵  100000   1  96.0  70.0   1  85  76.0  78  13812446614
1  谭理琳  100001   0   NaN  52.0   2  81  67.0  89  13118076908
2  陈关学  100002   0  85.0   NaN   3  88  69.0  81  14045193027
3  曹重丽  100003   0  92.0  56.0   4  72   NaN  93  13948141420
4  程意公  100004   0  96.0  94.0   5  79  84.0  74  13596917594
5  聂电在  100005   1  57.0  56.0   6  79  73.0  65  13677299256
6  段员可  100006   0  81.0  71.0   7  77  98.0  91  13589376055
7  谢变和  100007   0  47.0  70.0   8  77  85.0  68  13625277801
8  钱梅灵  100008   1  57.0  63.0   9  98  65.0  95  13945042761
9  梁各莹  100009   1  45.0  82.0  10  74  70.0  72  13973637989

isnull 输出:
姓名     学号     性别     数学     物理     班级     生物     英语     语文   手机
0   False  False  False  False  False  False  False  False  False  False
1   False  False  False   True  False  False  False  False  False  False
2   False  False  False  False   True  False  False  False  False  False
3   False  False  False  False  False  False  False   True  False  False
4   False  False  False  False  False  False  False  False  False  False
5   False  False  False  False  False  False  False  False  False  False
6   False  False  False  False  False  False  False  False  False  False
7   False  False  False  False  False  False  False  False  False  False
8   False  False  False  False  False  False  False  False  False  False
9   False  False  False  False  False  False  False  False  False  False
10  False  False  False  False  False  False  False  False  False  False

notnull 输出:
姓名    学号    性别    数学    物理    班级    生物    英语    语文    手机
0   True  True  True   True   True  True  True   True  True  True
1   True  True  True  False   True  True  True   True  True  True
2   True  True  True   True  False  True  True   True  True  True
3   True  True  True   True   True  True  True  False  True  True
4   True  True  True   True   True  True  True   True  True  True
5   True  True  True   True   True  True  True   True  True  True
6   True  True  True   True   True  True  True   True  True  True
```

```
7   True  True  True   True   True  True  True   True  True  True
8   True  True  True   True   True  True  True   True  True  True
9   True  True  True   True   True  True  True   True  True  True
10  True  True  True   True   True  True  True   True  True  True
```

2、缺失数据的处理

对于缺失数据的处理方式，有数据补齐、删除对应行、不处理几种方式。

（1）dropna()：去除数据结构中值得为空白的行。

可以指定参数 how = 'all'，表示只有行里的数据全部都是空时才丢弃。如果想以相同的方式按列丢弃，可以传入 axis=1。

```
newdata = data.dropna()
newdata = data.dropna(axis = 1)
```

（2）fillna()：有其它数据来代替 NaN。

有时候，直接删除数据会影响分析的结果，可以对数据进行填补。

A、用前一个数据值替代，fillna(method = 'pad')

B、用后一个数据值替代，fillna(method = 'bfill)

C、用平均数来替代，fillna(df.mean())

D、df.fillna(df.mean()['填补列名':'计算均值的列名'])：使用别的列的均值进行处理

E、Df.fillna({'列名 1':值 1,'列名 2':值 2})：可以传入一个字典，对不同的值填充不同的值。

```
# newdata = data.fillna('?')
# newdata = data.fillna(method='pad')
# newdata = data.fillna(method='bfill')
# newdata = data.fillna(data.mean())
newdata = data.fillna({'数学':100,'物理':99})
print(newdata)
```

9.8.3. 异常值处理

1、清除字符型数据左右指定的字符，默认为空格

应用 strip()函数删除左右两边的空格；

应用 rstrip()函数删除左右两边的空格；

应用 lstrip()函数删除左右两边的空格；

```
print(data['姓名'].str.strip())
print(data['姓名'].str.rstrip())
print(data['姓名'].str.lstrip())
```

2、真正的异常值是指那些远离正常值的观测值，也就是不群的观测。导致异常值的出现一般是人为的记录错误或者是设备的故障等。异常值的出现会对模型的创建和预测产生严重的后果。

当然，异常值也不一定是坏事，有些情况下，通过寻找异常值就能给企业带来良好的发

展，如销毁“钓鱼”网站、关闭“薅羊毛”用户的权限等。

对于异常值的检测，一般采用两种方法：一种是 n 个标准差法，另一种是箱线图判别法。

9.9. 数据抽取

9.9.1. 字段抽取

字段抽取是指抽出某列上指定位置的数据做成新的列，其命令格式如下：

slice(start, stop)

start：表示开始位置；

stop：表示结束位置；

比方抽取手机号字段的前 3 位，识别服务商品牌，中间 4 位作为识别号码地区，最后 4 位显示用户看。

```
# 导入的电话号码可能是个数值形式
data = read_table(r'E:\TestData\grade.txt',sep= ',')
data['电话'] = data['电话'].astype(str)
bands = data['电话'].str.slice(0,3)
areas = data['电话'].str.slice(3,7)
users = data['电话'].str.slice(7,11)
```

9.9.2. 字段拆分

字段拆分是指按着指定的字符 sep，拆分已有的字符串，其命令格式如下：

Split(sep, n, expand = true)

Sep：表示用于分隔字符串的分隔符；

N：表示分割后新增的列数；

Expand：表示是否展开为数据框，默认为 false。

返回值： expand 为 true，返回 DataFrame，expand 为 false，返回 Series

```
data = read_table(r'E:\TestData\iputf8.txt',sep=',')
print(data)
newdata = data['IP'].str.split('.',2,True)
newdata.columns = ['IP1','IP2','IP3-4']
print(newdata)
输出:
   学号           手机              IP
0   1  13818121714  202.134.122.29
1   2  13817131615  202.133.132.26
2   3  13818141516  202.132.142.24
3   4  13819151417  202.131.152.22
4   5  13811161319  202.139.162.21
```

```
   IP1  IP2   IP3-4
0  202  134  122.29
1  202  133  132.26
2  202  132  142.24
3  202  131  152.22
4  202  139  162.21
```

9.9.3. 重置索引

重置索引是指指定某列为索引，以便于对其它数据进行操作。其命令格式如下：
df.set_index('列名')

9.9.4. 条件抽取数据

记录抽取指根据一定的条件，对数据进行抽取。其命令格式如下：
df[condition]

condition：表示过滤条件。
返回值：DataFrame
常用的 condition 类型有：
（1）比较运算：==、<、>、<=、>=、!=
（2）范围运算：between(left, right)
（3）空置运算：pandas.isnull(column)
（4）字符匹配：str.contains()
（5）逻辑运算：&、|、not

```
print(data[data['语文'] > 85])
print(data[data.语文.between(75,90)])
print(data[data.姓名.str.contains('五')])
print(data[(data['语文'] > 60) & (data['数学'] > 90)])
```

9.9.5. 索引抽取数据

在 Pandas 模块中实现数据框子集的获取可以使用 iloc、loc 和 ix 三种方法，这三种方法既可以对数据进行筛选，也可以实现对变量进行挑选，它们的语法可以表示成[rows_select, cols_select]

1、通过索引名

df.loc[行标签，列标签]

注意，这里的标签不是索引。第一个参数是行标签，第二个参数是列标签，它是可选参数，默认是所有的标签。如果两个参数都为列表，则返回 DataFrame，否则返回 Series。

同时抽取多行时，行的索引必须使用列表的形式，而不能简单地用逗号分隔，如抽取第一行和第二行，应该用 df.loc[[1, 2]]，而不要用 df.loc[1, 2]

2、通过索引号

df.iloc[行索引号，列索引号]

iloc 只能通过行号和列号进行数据筛选，读者可以将 iloc 中的 i 理解为“integer”，即只能向[rows_select, cols_select]指定定整数列表。该索引方式与数组的索引方式类似，都是是从 0 开始，可以间隔取号。

3、Ix 方法

Ix 是 iloc 和 loc 的结合体，根据索引号或者索引名都可以，但是当索引名是 int 类型时，只能用索引名索引。

9.10. 数据修改

修改数据是常有的事，比如数据串中的有些需要整体替换，有些需要个别修改等情况经常会出现。

1、整体替换

整行整列的替换，很容易做到。

df[‘数学’] = score_2，score_2 就是将要被填进去的数据列，可以是列表，也可以是 Series。

2、单值替换

命令格式：df.replace(‘B’, ‘A’)

用 A 替换 B

```
data = pd.DataFrame({'班级':[2,1,2,1,2,1,2,1,2,1],
                     '姓名':['  张一','张二','张三','张四','张五','张六','张七','张八','张九','张十'],
                     '性别':['男', '男', '女', '女', '女', '女', '男', '男', '女', '男'],
                     '语文':[60,65,57,83,76,84,90,'作弊',67,88],
                     '数学':[91,95,'缺考',88,84,83,87,100,68,79],
                     '历史':[91,93,92,95,94,92,90,89,98,97],
                     })
print(data)
newdata1 = data.replace('缺考',0)
print(newdata1)
newdata2 = newdata1.replace('作弊',-1)
print(newdata2)
```

3、指定列单值替换

命令格式：df.replace({‘数学’: ‘缺考’}, 0)

df.replace({‘数学’: ‘缺考’ , ‘英语’ : ‘作弊’}, 0)

第一个语句是用 0 来代替数学成绩中的缺考，后一个语句还用 0 来代替英语中的作弊。

4、多值替换

命令格式：df.replaec([‘成龙’ , ‘周怡’] , [‘陈龙’ , ‘周毅’])

用陈龙和周毅来代替成龙和周怡

9.11. 重置索引

1、set_index

命令格式：

DataFrame.set_index(keys , drop = True , append = False , inplace = False)

append = true 时，保留原索引并添加新索引；

drop 为 False 时，保留被作为索引的列；默认情况下，设置为索引的列会从数据框中删除。

inplace 为 True 时，在原数据集上修改；

2、reset_index

还原索引，使索引重新变为默认的整型索引，它是 set_index 的逆运算。`

9.12. 透视表

透视表就是应用 pivot_table 方法：

pivot_table(values , index , columns , aggfunc , fill_value)

参数说明：

values：表示数据透视表中的值；

index：表示数据透视表中的行；

column：表示数据透视表中的列；

aggfunc：统计函数，fill_value 表示 NA 值的统一替换。

fill_valve：指定一个标量，用于填充缺失值

1、最简单的透视，一个数据框，一个索引，选用“班级”做索引，默认的统计函数采用的是平均值。

```
import pandas as pd

data = pd.DataFrame({'班级':[2,1,2,1,2,1,2,1,2,1],
                     '性别': ['男', '男', '女', '女', '女', '女', '男', '男', '
女', '男'],
                     '姓名':['张一','张二','张三','张四','张五','张六','张七','
张八','张九','张十'],
                     '语文':[60,65,57,83,76,84,90,75,67,88],
                     '数学':[91,95,93,88,84,83,87,100,68,79],
                     '历史':[91,93,92,95,94,92,90,89,98,97],
                     '总分':[242,253,242,266,254,259,267,264,233,264],
                     '评价':['中','中','中','优','良','优','优','优','中','优']
                     })
print(data)
print('---------------------------------------')
print(pd.pivot_table(data,index=['班级']))
输出：
历史  姓名 性别   总分   数学  班级 评价  语文
0  91  张一  男  242  91  2  中  60
```

```
1  93  张二  男  253   95   1  中  65
2  92  张三  女  242   93   2  中  57
3  95  张四  女  266   88   1  优  83
4  94  张五  女  254   84   2  良  76
5  92  张六  女  259   83   1  优  84
6  90  张七  男  267   87   2  优  90
7  89  张八  男  264  100   1  优  75
8  98  张九  女  233   68   2  中  67
9  97  张十  男  264   79   1  优  88
------------------------------------
    历史   总分    数学  语文
班级
 1   93.2  261.2  89.0  79
 2   93.0  247.6  84.6  70
```

2、选用两个索引，“班级”和“性别”，它已经开始通过将“班级”列和“性别”列进行对应分组，来实现数据聚合和总结。：

```
print(pd.pivot_table(data,index=['班级','性别']))
------------------------------------------------
            历史         总分        数学        语文
班级 性别
 1    女   93.500000  262.500000  85.500000  83.500000
      男   93.000000  260.333333  91.333333  76.000000
 2    女   94.666667  243.000000  81.666667  66.666667
      男   90.500000  254.500000  89.000000  75.000000
```

3、如果我们只关心“总分”和“数学”，“历史”和“语文”不感兴趣，可设置 values 属性。

```
print(pd.pivot_table(data,index=['班级','性别'],values=['总分','数学']))
------------------------------------
            总分         数学
班级 性别
 1    女   262.500000  85.500000
      男   260.333333  91.333333
 2    女   243.000000  81.666667
      男   254.500000  89.000000
```

4、如果除开平均数，我对其它的统计数据也有兴趣，可以设置 aggfunc 参数，aggfunc 可以包含很多函数：

```
print(pd.pivot_table(data,index=['班级','性别'],values=['语文','数学','历史
'],aggfunc=[np.mean,np.sum]))
------------------------------------
           mean                        sum
           历史        数学        语文        历史   数学   语文
```

```
班级  性别
 1    女    93.500000  85.500000  83.500000  187  171  167
      男    93.000000  91.333333  76.000000  279  274  228
 2    女    94.666667  81.666667  66.666667  284  245  200
      男    90.500000  89.000000  75.000000  181  178  150
```

为了对你选择的不同值执行不同的函数，你可以向 aggfunc 传递一个字典，字典的键就会做为函数名的标签，那必须将标签做的更加简洁。

5、“columns”属性的应用：

```
print(pd.pivot_table(data,index=['班级','评价'],values=['语文','数学
'],columns=['性别'],aggfunc=[np.mean]))
-------------------------------------
           mean
           数学          语文
性别         女     男     女     男
班级 评价
 1   中     NaN   95.0    NaN   65.0
     优    85.5   89.5   83.5   81.5
 2   中    80.5   91.0   62.0   60.0
     优     NaN   87.0    NaN   90.0
     良    84.0   NaN    76.0   NaN
```

9.13. 合并连接

1、记录合并

记录合并就是指两个结构相同的数据框合并成一个数据框，也就是在一个数据框中追加另一个数据的数据记录。

命令格式：

Concat([dataframe1 , dataframe2,……])

返回一个新的数据框

```
print(pd.concat([mydata1,mydata2,mydata3]))
```

2、字段合并

字段合并是指同一个数据框中的不同列进行合并，形成新的列。

命令格式：

X = x1 + x2 + ……

返回值是 Series，要求合并成的序列长度一致。

```
data = data.astype(str)
phone = data['bands'] + data['areas'] +data['tells']
data['phone'] = phone
```

3、字段匹配

字段匹配是指不同结构的数据框，按着一定的条件进行匹配合并。有两个数据表，第一个表中有学号、姓名，第二个表中有学号、手机号，现需要整理一份数据表，包含学号、姓名、手机号，此时就可以用到 merge 函数。

命令格式：

Merge(x , y , left_on , right_on)

X 表示第一个数据框；

y 表示第二个数据框；

left_on 表示第一个数据框中用于匹配的列；

right_on 表示第二个数据框用于匹配的列。返回 DataFrame

```
data1 = pd.DataFrame({'班级':[2,1,2,1,2,1,2,1,2,1],
                      '姓名':['  张一','张二','张三','张四','张五','张六','张七
','张八','张九','张十'],
                      '性别':['男', '男', '女', '女', '女', '女', '男', '男', '
女', '男'],
                      '语文':[60,65,57,83,76,84,90,75,67,88],
                      '数学':[91,95,93,88,84,83,87,100,68,79]
                      })

data2 = pd.DataFrame({'班级':[2,1,2,1,2,1,2,1,2,1],
                      '姓名':['  张一','张二','张三','张四','张五','张六','张七
','张八','张九','张十'],
                      '性别':['男', '男', '女', '女', '女', '女', '男', '男', '
女', '男'],
                      '物理':[60,65,57,83,76,84,90,75,67,88],
                      '地理':[91,95,93,88,84,83,87,100,68,79]
                      })

print(pd.merge(data1,data2,left_on='姓名',right_on='姓名'))
```

9.14. 分组聚合

数据分组是指根据数据分析对象的特征，按着一定的数据指标，把数据划分为不同的区间来进行研究，以揭示其内在的联系和规律性。简单地说，就是新增一列，将原来的数据按照其性质归入新的类别中。

其命令格式如下：

cut(series , bins , ritht = True , labels = null)

其中：

series 表示需要分组的数据；

bins 表示分组的依据数据；

right 表示分组的时候右边是否闭合；

labels 表示分组的自定义标签，可以不自定义；

```
bins = [min(data.English)-1 , 60,70,80,max(data.English)+1]
labs = ['不及格','及格','中等','良好','优秀']
demo = pd.cut(data.English ,bins ,right = False ,labels = labs)
```

10. 数据基本分析

10.1. 基本统计分析

基本统计分析又叫描述性统计分析，一般统计某个变量的最小值、第一个四分位值、中值、第三个四分位值以及最大值。

数据的中心位置是我们最容易想到的数据特征，借助中心位置，我们可以知道数据的一个平均情况，如果要对数据进行预测，那么平均情况是非常直观的选择。

数据的中心位置可分为均值（Mean）、中位数（Median）和众数（Mode）。其中，中值和中位数用于定量的数据，众数用于定性的数据。对于定量的数据（Data）来说，均值是总和除以总量 N，中位数是数值大小位于中间的值，均值相对于中位数来说，包含的信息量更大。

描述性统计分析函数为 describe。该函数返回有均值、标准差、最大值、最小值、分位数。括号中可以带一些参数，如 percentiles = [0.2,0.4,0.6,0.8]，就是指定只计算 0.2,0.4,0.6,0.8 分位数，而不是默认的 1/4,1/2,3/4 分位数。

Numpy 可以用 mean 函数计算平均数，也可以用 average 函数计算平均成绩，也可以用 Pandas 的 DataFrame 对象的 mean 方法求值，计算中位数用 median 函数，使用 mode()计算众数。

常用的统计函数有：

size：计数，注意，此函数不需要用括号

sum：求和

mean：平均值

var：方差

std：标准差

```
import pandas as pd
import  random

data = pd.DataFrame({'语文':[random.randint(55,100) for i in range(1000)],
                     '数学': [random.randint(55, 100) for i in range(1000)],
                     '历史':[random.randint(55,100) for i in range(1000)]})

print(data['语文'].describe())
print(data[['语文','数学']].describe())
print('计数：',data.语文.size)
print('最大值：',data.语文.max())
print('最小值：',data.语文.min())
```

```
print('求和：',data.语文.sum())
print('平均值：',data.语文.mean())
print('方差：',data.语文.var())
print('标准差：',data.语文.std())
# 计算中位数
print('中位数:',data.median())
# 计算众数
print('众数:',data.mode())
```

10.2. 分组分析

分组分析是指根据分组字段将分析对象划分成不同的部分，以对比分析各组之间差异性的一种分析方法。

常用的统计指标有计数、求和、平均值。

常用命令形式：

Df.groupby(by = ['分类 1','分类 2', ……])['被统计的列'].统计函数

```
import pandas as pd

data = pd.DataFrame({'班级':[2,1,2,1,2,1,2,1,2,1],
                     '性别':['男','男','女','女','女','男','男','女','女','男'],
                     '姓名':['张一','张二','张三','张四','张五','张六','张七','
张八','张九','张十'],
                     '语文':[60,65,57,83,76,84,90,75,67,88],
                     '数学':[91,95,93,88,84,83,87,100,68,79],
                     '历史':[91,93,92,95,94,92,90,89,98,97]
                     })

print('平均值:\n',data.groupby(by=['班级','性别'])['语文','数学','历史'].mean())
print('最大值:\n',data.groupby(by=['班级','性别'])['语文','数学','历史'].max())
print('最小值:\n',data.groupby(by=['班级','性别'])['语文','数学','历史'].min())
print('人数：\n',data.groupby(by=['班级','性别'])['语文','数学','历史'].size())
print('标准差:\n',data.groupby(by=['班级','性别'])['语文','数学','历史'].std())
print('方差：\n',data.groupby(by=['班级','性别'])['语文','数学','历史'].var())


输出：

平均值：
            语文         数学         历史
班级 性别
  1  女   79.000000  94.000000  92.000000
```

```
     男   79.000000  85.666667  94.000000
  2  女   66.666667  81.666667  94.666667
     男   75.000000  89.000000  90.500000
最大值：
        语文   数学  历史
班级 性别
  1  女   83  100  95
     男   88   95  97
  2  女   76   93  98
     男   90   91  91
最小值：
        语文  数学  历史
班级 性别
  1  女   75  88  89
     男   65  79  92
  2  女   57  68  92
     男   60  87  90
人数：
 班级  性别
  1   女     2
      男     3
  2   女     3
      男     2
dtype: int64
标准差：
          语文        数学        历史
班级 性别
  1  女   5.656854   8.485281   4.242641
     男  12.288206   8.326664   2.645751
  2  女   9.504385  12.662280   3.055050
     男  21.213203   2.828427   0.707107
方差：
          语文         数学         历史
班级 性别
  1  女   32.000000   72.000000  18.000000
     男  151.000000  69.333333   7.000000
  2  女   90.333333  160.333333  9.333333
     男  450.000000   8.000000   0.500000
```

10.3. 分布分析

分布分析是指根据分析的目的，将数据进行等距或者不等距的分组，研究各组分布规律

的一种分析方法。

```
import pandas as pd

data = pd.DataFrame({'班级':[2,1,2,1,2,1,2,1,2,1],
                     '姓名':['张一','张二','张三','张四','张五','张六','张七','
张八','张九','张十'],
                     '语文':[60,65,57,83,76,84,90,75,67,88],
                     '数学':[91,95,93,88,84,83,87,100,68,79],
                     '历史':[91,93,92,95,94,92,90,89,98,97],
                     })
data['总分'] = data.语文 + data.数学 + data.历史
bins = [min(data.总分)-1,245,255,max(data.总分)+1]
labes = ['245 以下','245-255','255 以上']

newdata = pd.cut(data.总分,bins,labels=labes)
print('分段信息: \n',newdata)
data['分段'] = newdata
print(data)
输出:
分段信息:
 0      245 以下
1    245-255
2      245 以下
3      255 以上
4    245-255
5      255 以上
6      255 以上
7      255 以上
8      245 以下
9      255 以上
Name: 总分, dtype: category
Categories (3, object): [245 以下 < 245-255 < 255 以上]
   历史  姓名   数学  班级  语文   总分       分段
0  91  张一   91   2  60  242    245 以下
1  93  张二   95   1  65  253  245-255
2  92  张三   93   2  57  242    245 以下
3  95  张四   88   1  83  266    255 以上
4  94  张五   84   2  76  254  245-255
5  92  张六   83   1  84  259    255 以上
6  90  张七   87   2  90  267    255 以上
7  89  张八  100   1  75  264    255 以上
8  98  张九   68   2  67  233    245 以下
9  97  张十   79   1  88  264    255 以上
```

10.4. 交叉分析

交叉分析通常用于分析两个或两个以上分组变量之间的关系，以交叉表形式进行变量间关系的对比分析。

常用命令格式：

pivot_table(values , index , columns , aggfunc , fill_value)

参数说明：

values：表示数据透视表中的值；

index：表示数据透视表中的行；

column：表示数据透视表中的列；

aggfunc：统计函数，fill_value 表示 NA 值的统一替换。

fill_valve：指定一个标量，用于填充缺失值

```
import pandas as pd
import numpy as np

data = pd.DataFrame({'班级':[2,1,2,1,2,1,2,1,2,1],
                     '性别': ['男', '男', '女', '女', '女', '女', '男', '男', '
女', '男'],
                     '姓名':['张一','张二','张三','张四','张五','张六','张七','
张八','张九','张十'],
                     '语文':[60,65,57,83,76,84,90,75,67,88],
                     '数学':[91,95,93,88,84,83,87,100,68,79],
                     '历史':[91,93,92,95,94,92,90,89,98,97],
                     })
data['总分'] = data.语文 + data.数学 + data.历史
bins = [min(data.总分)-1,245,255,max(data.总分)+1]
labes = ['245以下','245-255','255以上']

newdata = pd.cut(data.总分,bins,labels=labes)
data['分段'] = newdata
print(data)
print(data.pivot_table(values='总分',
                       index=['分段','性别'],
                       columns='班级',
                       aggfunc=[np.mean,np.size]))
```

输出：

```
   历史  姓名 性别   数学  班级  语文   总分       分段
0  91  张一   男   91   2  60  242    245以下
1  93  张二   男   95   1  65  253  245-255
2  92  张三   女   93   2  57  242    245以下
3  95  张四   女   88   1  83  266    255以上
4  94  张五   女   84   2  76  254  245-255
```

```
5  92  张六  女   83   1  84  259    255 以上
6  90  张七  男   87   2  90  267    255 以上
7  89  张八  男  100   1  75  264    255 以上
8  98  张九  女   68   2  67  233    245 以下
9  97  张十  男   79   1  88  264    255 以上
                  mean          size
班级                1      2      1    2
分段      性别
245 以下   女      NaN  237.5    NaN  2.0
          男      NaN  242.0    NaN  1.0
245-255   女      NaN  254.0    NaN  1.0
          男    253.0    NaN    1.0  NaN
255 以上  女    262.5    NaN    2.0  NaN
          男    264.0  267.0    2.0  1.0
```

10.5. 结构分析

结构分析是在分组分析以及交叉分析的基础上，计算各组成部分所占的比重，进而分析总体的内部特征的一种分析方法。这里的分组主要是看指定性分组，定性分组一般看结构，它的重点在于计算各组成部分占总体的比重。

```
import pandas as pd
import numpy as np

data = pd.DataFrame({'班级':[2,1,2,1,2,1,2,1,2,1],
                     '姓名':['张一','张二','张三','张四','张五','张六','张七','张八','张九','张十'],
                     '性别':['男', '男', '女', '女', '女', '女', '男', '男', '女', '男'],
                     '语文':[60,65,57,83,76,84,90,75,67,88],
                     '数学':[91,95,93,88,84,83,87,100,68,79],
                     '历史':[91,93,92,95,94,92,90,89,98,97],
                     })
data['总分'] = data.语文 + data.数学 + data.历史
bins = [min(data.总分)-1,245,255,max(data.总分)+1]
labes = ['245 以下','245-255','255 以上']

newdata = pd.cut(data.总分,bins,labels=labes)
data['分段'] = newdata

data_pt = data.pivot_table(values='总分',
                           index='班级',
```

```
                    columns='性别',
                    aggfunc=[np.sum])

print(data_pt)
print('---------------------------------------')
print(data_pt.sum())          # 按行统计
print('---------------------------------------')
print(data_pt.div(data_pt.sum(axis=0),axis =1))
print('---------------------------------------')
print(data_pt.sum(axis = 1)) # 按列统计
print('---------------------------------------')
print(data_pt.div(data_pt.sum(axis=1),axis =0))
```

输出：

```
    sum
性别    女    男
班级
1   525  781
2   729  509
---------------------------------------
     性别
sum  女     1254
     男     1290
dtype: int64
---------------------------------------
        sum
性别        女         男
班级
1   0.41866  0.605426
2   0.58134  0.394574
---------------------------------------
班级
1    1306
2    1238
dtype: int64
---------------------------------------
         sum
性别          女         男
班级
1   0.401991  0.598009
2   0.588853  0.411147
```

10.6. 相关分析

判断两个变量是否具有线性相关性的最直观的方法是直接绘制散点图，看变量之间是否符合某个规律的变化。当需要考察多个变量间的相关关系时，一一绘制它们间的简单散点图是比较麻烦的。此时可以利用散点矩阵较长同时绘制多种变量间的散点图，从而快速发现多个变量之间的主要相关性，这在进行多元线性回归时显得尤为重要。

相关分析是研究现象之间是否存在着某种依存关系，并对具体有依存关系现象探讨其相关方向以及相关程度，是研究随机变量之间的相关关系的一种统计方法。

为了更加准确地描述变量之间的线性相关程度，通过相关系数来进行相关分析，在二元变量的相关分析过程中，比较常用的有 Pearson 相关系数、Spearman 秩相关系统和判定系数。

相关系数可以用来描述定量变量之间的关系。相关系数与相关程度之间的关系如下：

0 <= |r| < 0.3	低度相关
0.3 <= |r| < 0.8	中度相关
0 .8<= |r| <= 1	高度相关

相关分析函数有：

DataFrame.corr()

Series.corr(other)

如果由 DataFrame 调用 corr 方法，那么将会计算每列两两之间的相关度，它会返回一个相关度的 DataFrame。如果由序列调用，那么只计算该序列与传入序列之间的相关度，直接返回相关度。

```
import pandas as pd

mydata1 = pd.Series([3,5,7,9,11,13,15,17])
mydata2 = pd.Series([4,6,8,10,12,14,15,18])
mydata3 = pd.Series([5,4,5,3,7,8,5,12,9])

print('相关度 1：',mydata1.corr(mydata2))
print('相关度 2：',mydata1.corr(mydata3))

mydata4 = pd.DataFrame({'数学':[99,96,80,95,55],
                        '物理':[100,97,85,93,60],
                        '语文':[88,86,79,83,86],
                        '英语':[90,91,90,89,88,]})
print('相关度 3:\n',mydata4.corr())
输出：
相关度 1： 0.9977119678953293
相关度 2： 0.6854242365806001
相关度 3:
      数学        物理        英语        语文
数学  1.000000  0.993008  0.730217  0.128427
物理  0.993008  1.000000  0.789455  0.097351
```

```
英语   0.730217   0.789455   1.000000 -0.012504
语文   0.128427   0.097351 -0.012504   1.000000
```

11. 数据可视化

文不如字，字不如表，表不如图，这句话充分说明了可视化的重要性。从事与数据相关的工作者经常会作一些总结或者展望性的报告，如果报告中密密麻麻的都是文字，相信听众和老板一定会烦，如果报告中呈现的是大量的图形化结果，就会边到众人的喜爱，因为图形更加直观醒目。本章的重点就是利用 Python 绘制常见的统计图形，如条形图、饼图、直方图、折线图、散点图等。这些图形的绘制可以通过 matplotlib 模块、pandas 模块或者 seaborn 模块实现。

Matplotlib 是最流行的用于制图及其它二维数组可视化的库，它与生态系统的其他库良好整合。

11.1. 离散型变量

11.1.1. 饼图

饼图，也叫饼状图，是一个划分为几个扇区的圆形统计图表，用于描述量、频率、或百分比之间的相对关系。在饼图中，每个扇区的弧长（圆心角和面积）大小为其所表示的数量的比例。这些扇区合在一起刚好是一个完整的圆形，这些扇区就好像拼成一个切开的饼形图案。

1、Matplotlib 模块

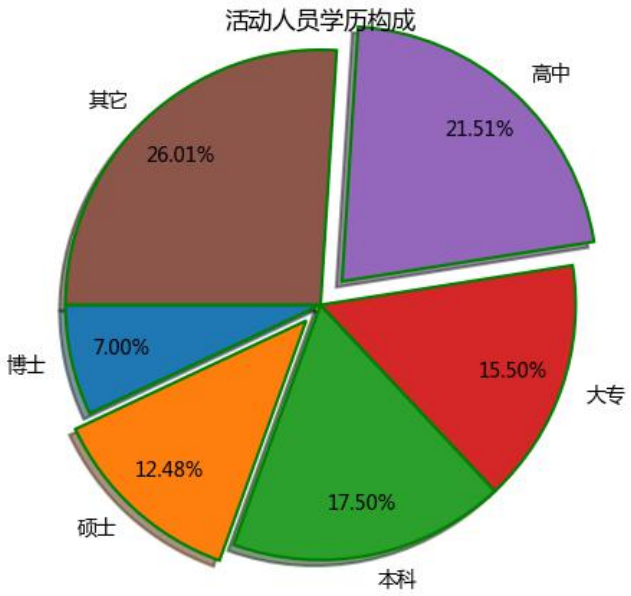
首先要导入 matplotlib 模块的子模块 pyplot，然后利用模块中的 pie 函数。

（1）示例 1

```
import matplotlib.pyplot as plt
# 处理中文乱码
plt.rcParams['font.sans-serif'] = ['Microsoft YaHei']
#构造数据
data = [255,455,638,565,784,948]
labels = ['博士','硕士','本科','大专','高中','其它']

#绘制饼图
plt.pie(x = data,                  # 数据
        labels = labels,           # 标签
        autopct = '%.2f%%'         # 格式
        )
plt.show()
```

（2）示例 2

```
import matplotlib.pyplot as plt
# 处理中文乱码
plt.rcParams['font.sans-serif'] = ['Microsoft YaHei']
plt.rcParams['axes.unicode_minus'] = False
# 标准化处理，确保饼图是个正圆，否则是椭圆
plt.axes(aspect='equal')

#构造数据
data = [255,455,638,565,784,948]
labels = ['博士','硕士','本科','大专','高中','其它']
# 设置突出显示的部分
explode = [0,0.1,0,0,0.15,0]

#绘制饼图
plt.pie(x = data,                    # 数据
        labels = labels,             # 标签
        autopct = '%.2f%%',          # 格式
        explode = explode ,          # 突出显示某个部分
        pctdistance = 0.8,           # 百分比标签到圆心的距离
        labeldistance = 1.1,         # 标签到圆心的距离
        startangle = 180,            # 饼图初始角度
        radius = 1.2,                # 饼图半径
        shadow= True,                # 阴影效果
        wedgeprops = {'linewidth':1.5, 'edgecolor':'green'}, # 内外边界属性值
        textprops = {'fontsize':10, 'color':'black'}         # 设置文本标签属性值
        )
plt.title('活动人员学历构成')
plt.show()
```



注意：

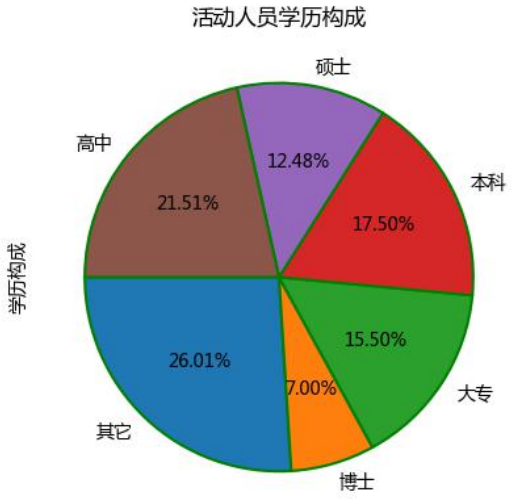
（A）如果绘制的图中涉及中文及数字中的负号，需要通过 rcParams 进行了控制；

（B）不加修饰的饼图更像是一个椭圆，所以需要利用 axes 函数强制为正圆。

2、Pandas 模块

（1）示例

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

# 处理中文乱码
plt.rcParams['font.sans-serif'] = ['Microsoft YaHei']

data = pd.Series({'博士':255,'硕士':455,'本科':638,'大专':565,'高中':784,'其它
':948,})
data.name = '学历构成'
data.plot(kind = 'pie',
          autopct = '%.2f%%',
          radius = 1,
          startangle = 180,
          title = '活动人员学历构成',
          wedgeprops = {'linewidth':1.5, 'edgecolor':'green'}, # 内外边界属性值
          textprops = {'fontsize':10, 'color':'black'}       # 设置文本标签属性值
          )
#显示图形
plt.show()
```



11.1.2. 条形图

条形图亦称条图、条状图、棒形图、柱状图，是一种以长方形的长度为变量的统计图表。长条图用来比较两个或者两个以上的数值（不同时间或者不同条件），只有一个变量，通常利用最小的数据集分析。

长条图也可横向排列，也可用多维表达方式。

1、matplotlib 模块

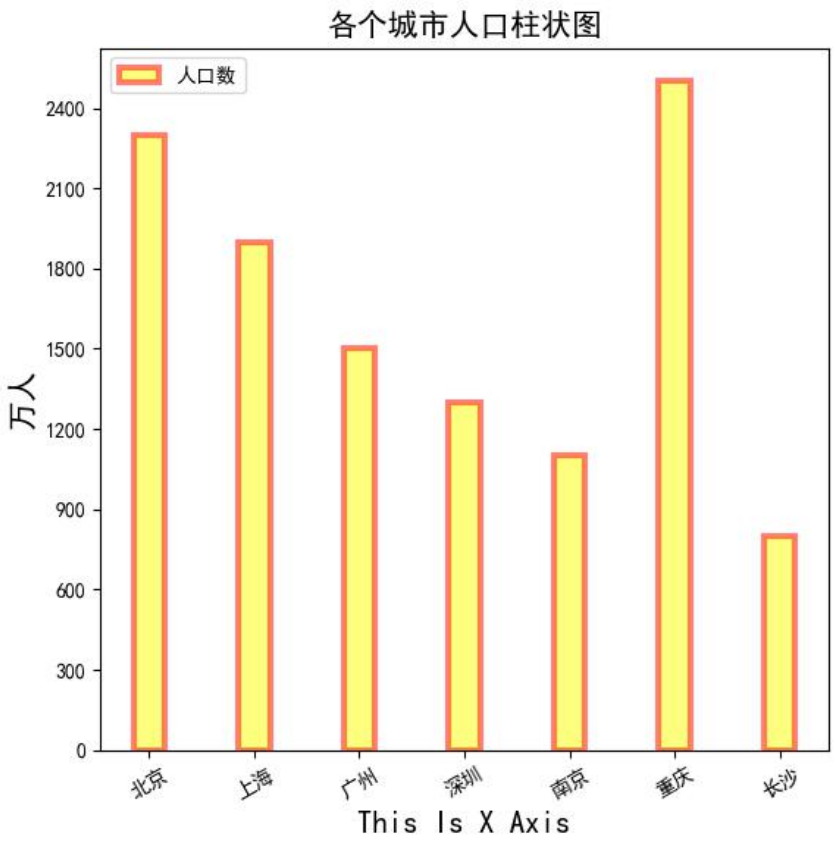
matplotlib.pyplot.bar(left, height, alpha=1, width=0.8, color=, edgecolor=, label=, lw=3)

1. left：x 轴的位置序列，一般采用 range 函数产生一个序列，但是有时候可以是字符串

2. height：y 轴的数值序列，也就是柱形图的高度，一般就是我们需要展示的数据；
3. alpha：透明度，值越小越透明
4. width：为柱形图的宽度，一般这是为 0.8 即可；
5. color 或 facecolor：柱形图填充的颜色；
6. edgecolor：图形边缘颜色
7. label：解释每个图像代表的含义，这个参数是为 legend()函数做铺垫的，表示该次 bar 的标签
8. linewidth or linewidths or lw：边缘 or 线的宽

```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
# 设置中文字体
mpl.rcParams['font.sans-serif'] = ['SimHei']

xdata = ["北京","上海","广州","深圳","南京","重庆","长沙"]
ydata = [2300,1900,1500,1300,1100,2500,800]

plt.bar(xdata, ydata, alpha=0.5, width=0.3, color='yellow', edgecolor='red',
```

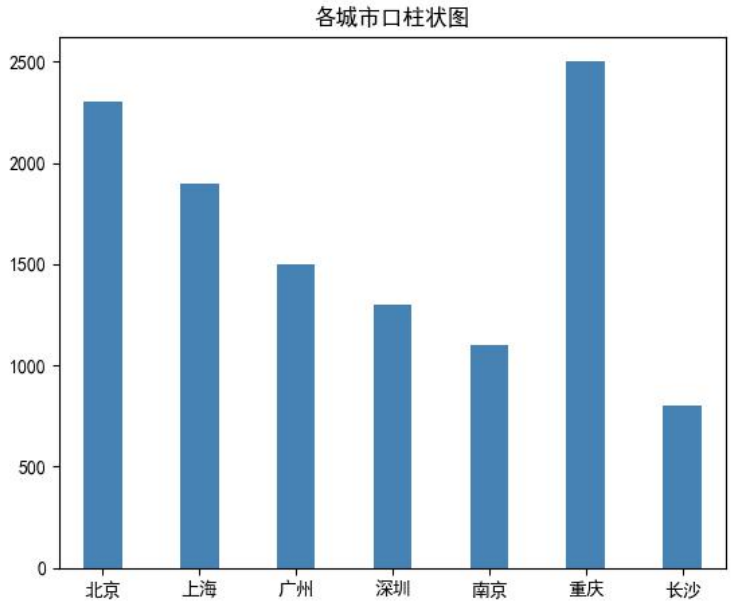
```
label='人口数', lw=3)
plt.legend(loc='upper left')

plt.xlabel('This Is X Axis', fontsize=15)
plt.ylabel('万人', fontsize=15)
plt.title('各个城市人口柱状图', fontsize=15)

# 如果想自己给 x 轴的 bar 加上标签,rotation 控制倾斜角度
plt.xticks(rotation=30)
plt.yticks(np.arange(0, 2500, 300))

plt.show()
```



2、pandas 模块

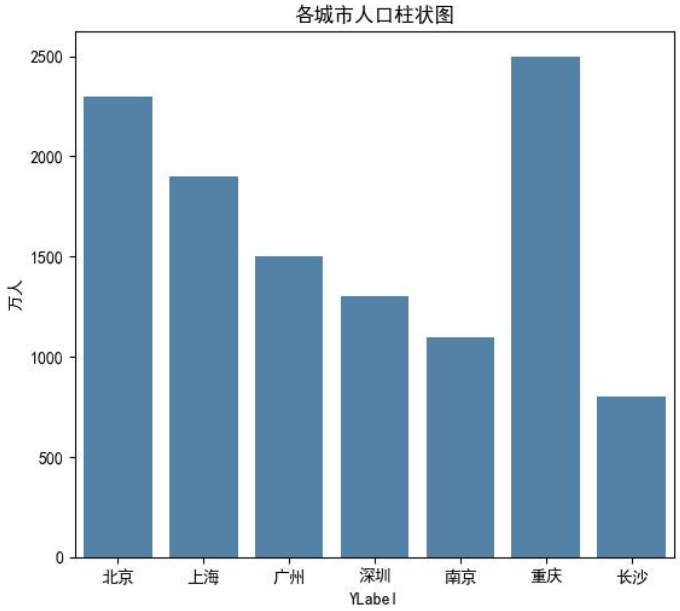
```
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
# 设置中文字体
mpl.rcParams['font.sans-serif'] = ['SimHei']

data = pd.Series([2300,1900,1500,1300,1100,2500,800],
                 index = ["北京","上海","广州","深圳","南京","重庆","长沙"])
```

```
data.plot(kind = 'bar',
          width =0.4,
          rot = 0,
          color='steelblue',
          title = '各城市口柱状图')
plt.show()
```



3、seaborn 模块

```
import seaborn as sns
import matplotlib as mpl
import matplotlib.pyplot as plt

# 设置中文字体
mpl.rcParams['font.sans-serif'] = ['SimHei']

xdata = ["北京","上海","广州","深圳","南京","重庆","长沙"]
ydata = [2300,1900,1500,1300,1100,2500,800]

sns.barplot(y = ydata,
            x = xdata,
            color = 'steelblue'
            )

plt.xlabel('YLabel')
plt.ylabel('万人')
plt.title('各城市人口柱状图')

plt.show()
```



11.2. 数值型变量

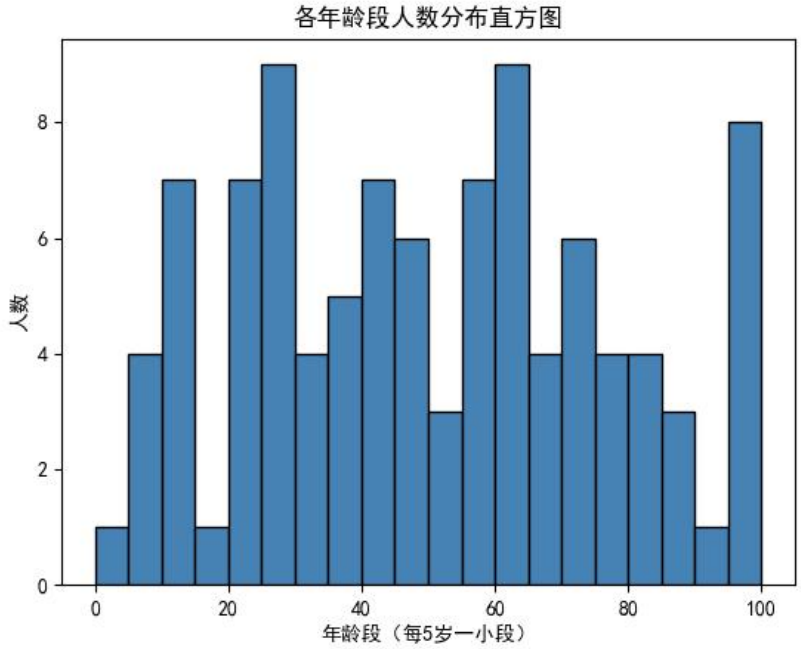
11.2.1. 直方图

直方图一般用来观察数据的分布形态，横坐标代表数值的均匀分段，纵坐标代表每个段内的观测数量，一般直方图都会与核密度图搭配使用，目的是为了更加清晰地掌握数据的分布特征。

1、matplotlib 模块

```
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import random

# 设置中文字体
mpl.rcParams['font.sans-serif'] = ['SimHei']

data = pd.DataFrame({'age':[random.randint(0, 100) for i in range(100)]})

plt.hist(x = data['age'],          # 数据
         bins = 20,                # 列数
         range = (0,100),          # 范围
         color = 'steelblue',      # 颜色
         edgecolor = 'black')      # 边界颜色

plt.xlabel('年龄段（每 5 岁一小段）')
```

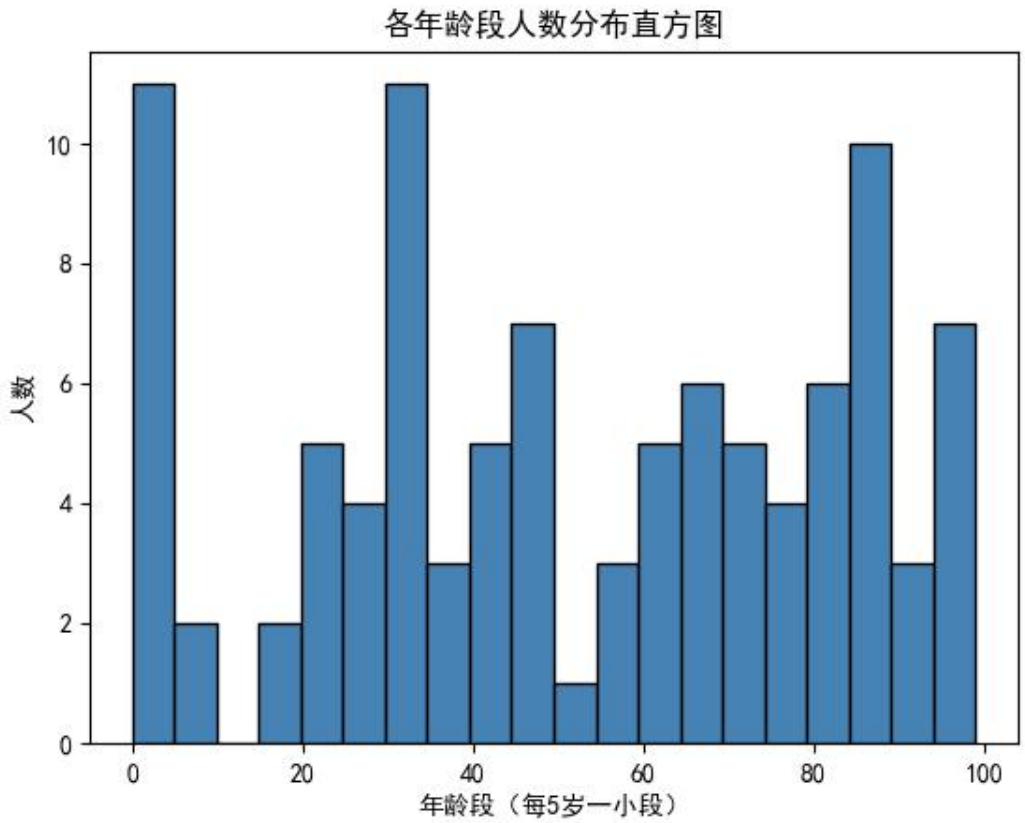
```
plt.ylabel('人数')
plt.title('各年龄段人数分布直方图')

plt.show()
```



2、pandas 模块

```
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import random

# 设置中文字体
mpl.rcParams['font.sans-serif'] = ['SimHei']

data = pd.Series([random.randint(0, 100) for i in range(100)])

data.plot(kind = 'hist',
          bins = 20,
          color = 'steelblue',
          edgecolor = 'black',
          label = '直方图'
          )

# data.plot(kind = 'kde',color = 'red',label = '核密度图')
```

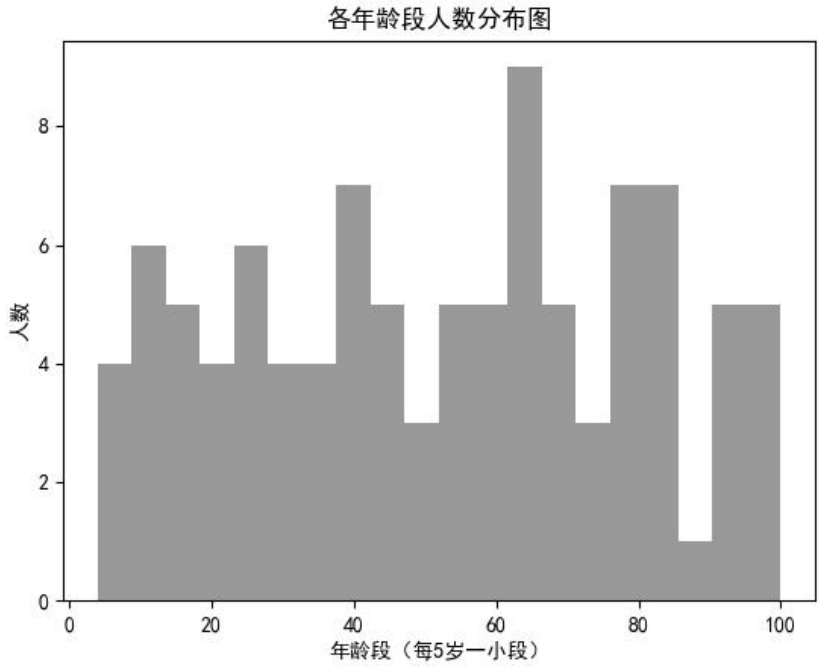
```
plt.xlabel('年龄段（每 5 岁一小段）')
plt.ylabel('人数')
plt.title('各年龄段人数分布直方图')

plt.show()
```



3、seaborn 模块

```
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sbn
import random

# 设置中文字体
mpl.rcParams['font.sans-serif'] = ['SimHei']

data = pd.Series([random.randint(0, 100) for i in range(100)])
```

```
sbn.distplot(data,
             bins = 20,
             kde = False,
             hist_kws = {'color':'black'},
             label = '年龄分布'
)

plt.xlabel('年龄段（每 5 岁一小段）')
plt.ylabel('人数')
plt.title('各年龄段人数分布图')

plt.show()
```

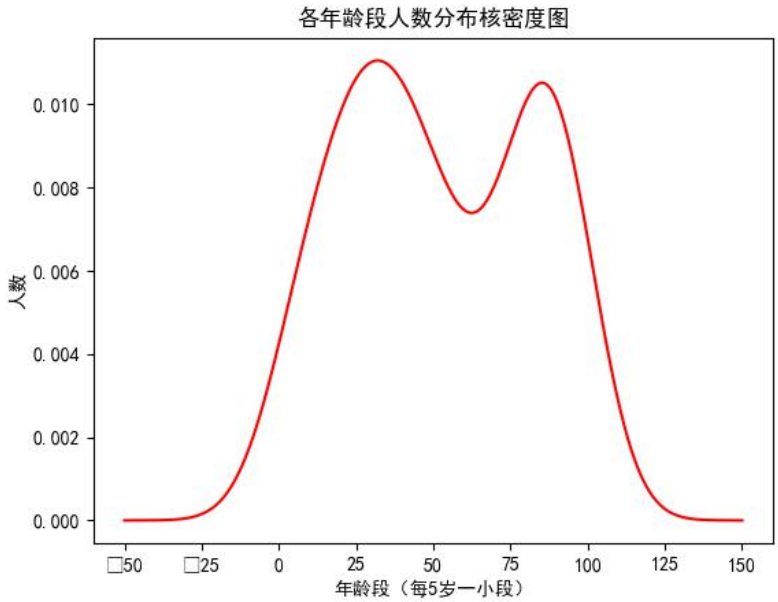


11.2.2. 核密度图

1、pandas 模块

```
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import random

# 设置中文字体
mpl.rcParams['font.sans-serif'] = ['SimHei']

data = pd.Series([random.randint(0, 100) for i in range(100)])
```

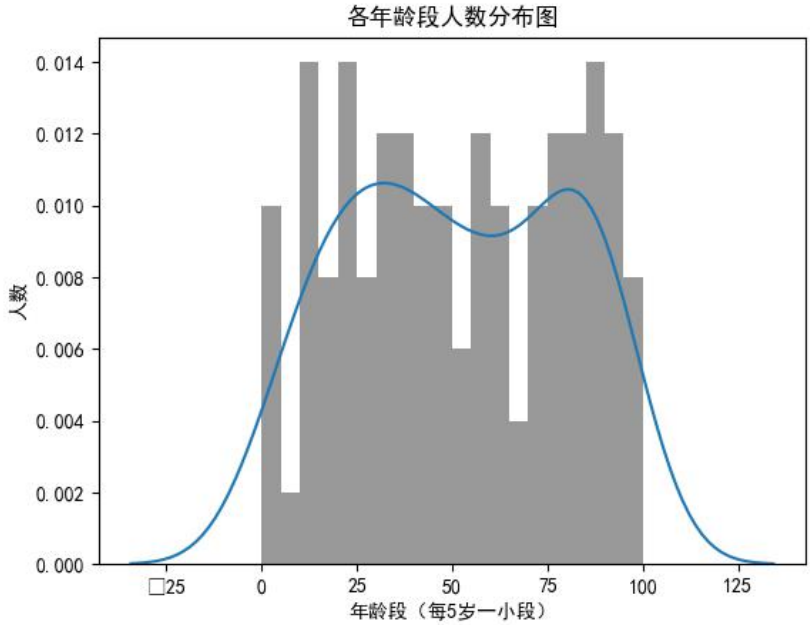
```
data.plot(kind = 'kde',
          color = 'red',
          label = '核密度图')

plt.xlabel('年龄段（每 5 岁一小段）')
plt.ylabel('人数')
plt.title('各年龄段人数分布核密度图')

plt.show()
```



2、seaborn 模块

```
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sbn
import random

# 设置中文字体
mpl.rcParams['font.sans-serif'] = ['SimHei']

data = pd.Series([random.randint(0, 100) for i in range(100)])

sbn.distplot(data,
             bins = 20,
             kde = True,
             hist_kws = {'color':'black'},
             label = '年龄分布'
```

```
)

plt.xlabel('年龄段（每 5 岁一小段）')
plt.ylabel('人数')
plt.title('各年龄段人数分布图')

plt.show()
```



11.2.3. 箱线图

11.2.4. 小提琴图

11.2.5. 折线图

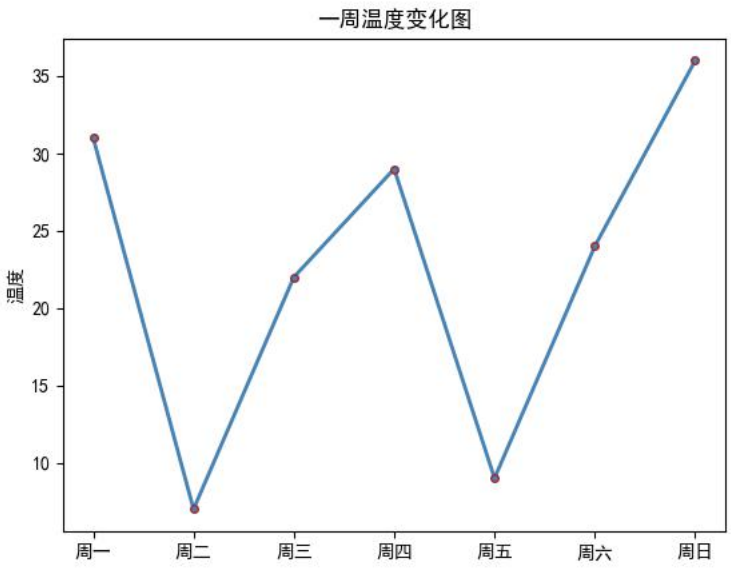
对于时间序列数据而言，一般都会使用折线图反映数据背后的超势。通常折线的横坐标指代日期数据，纵坐标代表某个数值型变量，当然，还可以使用第三个离散变量对折线图进行分组处理。

Python 中的 matplotlib 模块和 pandas 模块实现折线图的绘制。但是 seaborn 模块也可以绘制，它的 tsplot 函数非常不合理，所以不介绍。

1、matplotlib 模块

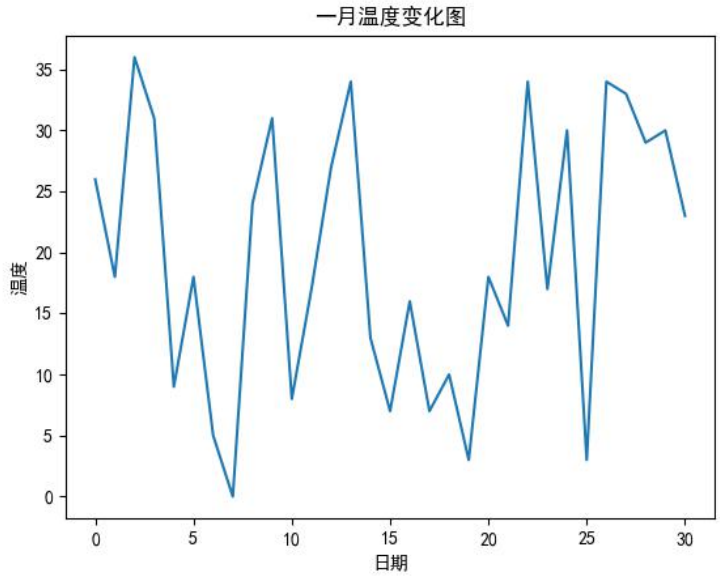
Matplotlib 模块中的 plot 函数可以绘制折线图。

```
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import random

# 设置中文字体
mpl.rcParams['font.sans-serif'] = ['SimHei']

data = pd.Series([random.randint(0, 37) for i in range(7)],
                 index = ['周一','周二','周三','周四','周五','周六','周日'])

plt.plot( data.index,
          data,
          linestyle = '-',
          linewidth = 2,
          color = 'steelblue',
          marker = 'o',
          markersize = 4,
          markeredgecolor = 'brown'
)

plt.ylabel('温度')
plt.title('一周温度变化图')

plt.show()
```



2、pandas 模块

```
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import random

# 设置中文字体
mpl.rcParams['font.sans-serif'] = ['SimHei']
data = pd.Series([random.randint(0, 37) for i in range(31)])

data.plot()

plt.ylabel('温度')
plt.xlabel('日期')
plt.title('一月温度变化图')

plt.show()
```



11.3. 关系型数据

在众多的可视化图形中，有一类图形专门用来探究两个或者三个变量之间的关系，例如，散点图用于发现两个数值变量之间的关系，气泡图用来展现三个数值之间的关系，热力图则体现了两个离散变量之间的组合关系。

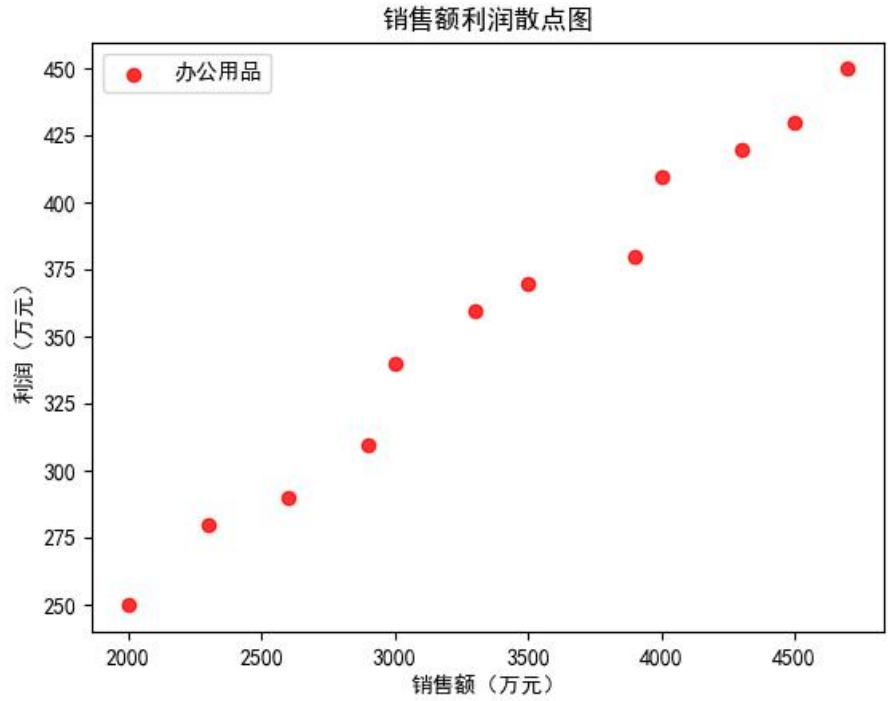
11.3.1. 散点图

如果需要研究两个数值变量之间是否存在某种关系，例如正向的线性关系，或者是趋势性的非线性关系，那么散点图将是最佳的选择。

1、matplotlib 模块

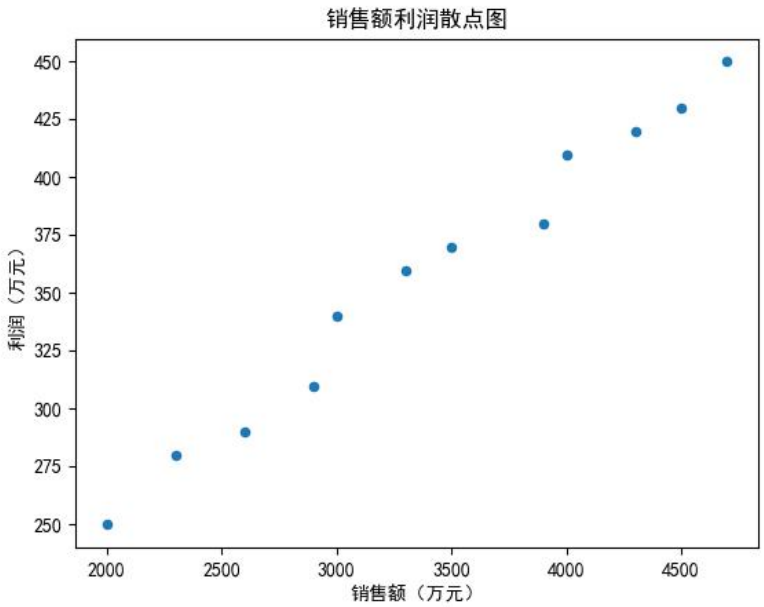
Matplotlib 模块中的 scatter 函数可以非常方便地绘制两个数值型变量的散点图。

```
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import random

# 设置中文字体
mpl.rcParams['font.sans-serif'] = ['SimHei']

data = pd.DataFrame(
{'销售额':[2000,2300,2600,2900,3000,3300,3500,3900,4000,4300,4500,4700],
'利润': [250,280,290,310,340,360,370,380,410,420,430,450]},
                    index = [1,2,3,4,5,6,7,8,9,10,11,12]
                  )
print(data)

plt.scatter(x=data['销售额'],
            y=data['利润'],
            label = '办公用品',
            color = 'red',
            alpha= 0.8
)

plt.xlabel('销售额（万元）')
plt.ylabel('利润（万元）')

plt.title("销售额利润散点图")
plt.legend()
plt.show()
```



2、pandas 模块

```
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import random

# 设置中文字体
mpl.rcParams['font.sans-serif'] = ['SimHei']

data = pd.DataFrame({'销售额
':[2000,2300,2600,2900,3000,3300,3500,3900,4000,4300,4500,4700],
                     '利润': [250,280,290,310,340,360,370,380,410,420,430,450]},
                    index = [1,2,3,4,5,6,7,8,9,10,11,12]
                   )
print(data)

data.plot(x = '销售额',
          y = '利润',
          kind = 'scatter')

plt.xlabel('销售额（万元）')
plt.ylabel('利润（万元）')

plt.title("销售额利润散点图")
plt.show()
```



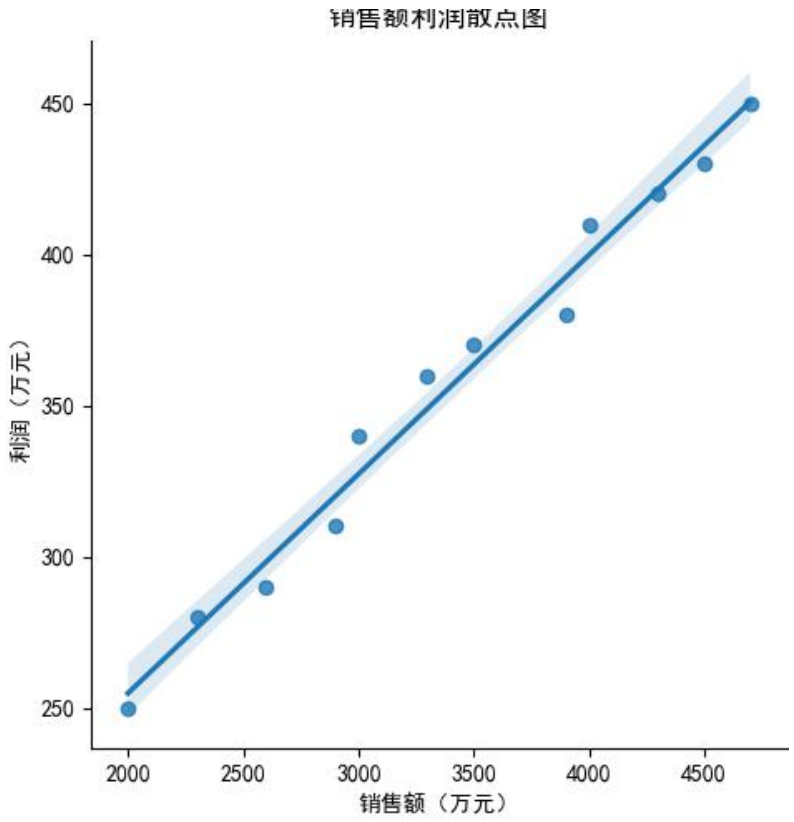
3、seaborn 模块

尽管使用 matplotlib 和 pandas 可以非常方便地绘制出散点图，但是绘制分组散点图会稍微复杂一点了。如果使用 seaborn 中的 lmplot 函数，那就非常方便了，而且，它还可以根据散点图添加线性拟合线。

Lmplot 函数还可以指定 lowess 参数拟合局部多项式回归，指定 logistic 参数拟合逻辑回归，指定 order 参数做多项式回归，指定 robust 做鲁棒回归。

```
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns

# 设置中文字体
mpl.rcParams['font.sans-serif'] = ['SimHei']

data = pd.DataFrame({'销售额
':[2000,2300,2600,2900,3000,3300,3500,3900,4000,4300,4500,4700],
                     '利润': [250,280,290,310,340,360,370,380,410,420,430,450]},
                    index = [1,2,3,4,5,6,7,8,9,10,11,12]
                   )

sns.lmplot(x = '销售额',
           y = '利润',
           data = data,
           legend_out= False,
           truncate= True )
```

```
plt.xlabel('销售额（万元）')
plt.ylabel('利润（万元）')
plt.title("销售额利润散点图")
plt.show()
```

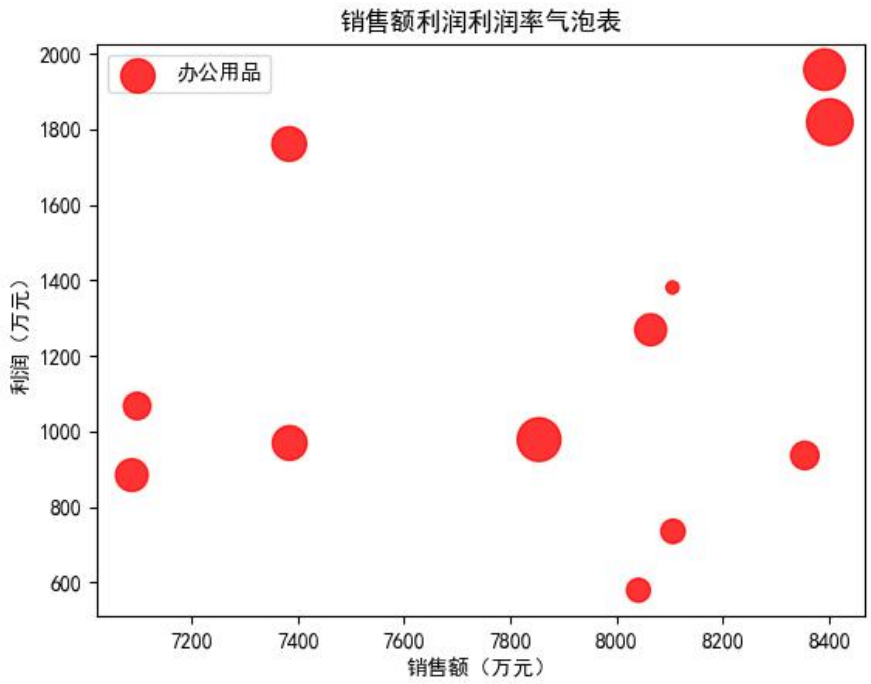


11.3.2. 气泡图

散点图一般用来反应两个离散型数据变量的关系，如果还想通过散点图添加第三个数值变量的信息，一般可以使用用气泡图。气泡图的实质就是通过第三个数值型变量控制每个散点的大小，点越大，代表第三维的数值越高，反之亦然。

```
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import random

# 设置中文字体
mpl.rcParams['font.sans-serif'] = ['SimHei']

data = pd.DataFrame({'销售额':[random.randint(7000,9000) for i in range(12)],
                     '利润': [random.randint(500, 2000) for i in range(12)],
                     '利润率': [random.randint(1, 100)*5 for i in range(12)]},
```

```
                    index = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
                )
print(data)

plt.scatter(x=data['销售额'],
            y=data['利润'],
            s=data['利润率'],
            label = '办公用品',
            color = 'red',
            alpha= 0.8
)

plt.xlabel('销售额（万元）')
plt.ylabel('利润（万元）')

plt.title("销售额利润利润率气泡表")
plt.legend()
plt.show()
```



pandas 与 seaborn 中没有绘制气泡图的方法或函数。
Python 中的 bokeh 模块，也可以实现绘制气泡图的功能。

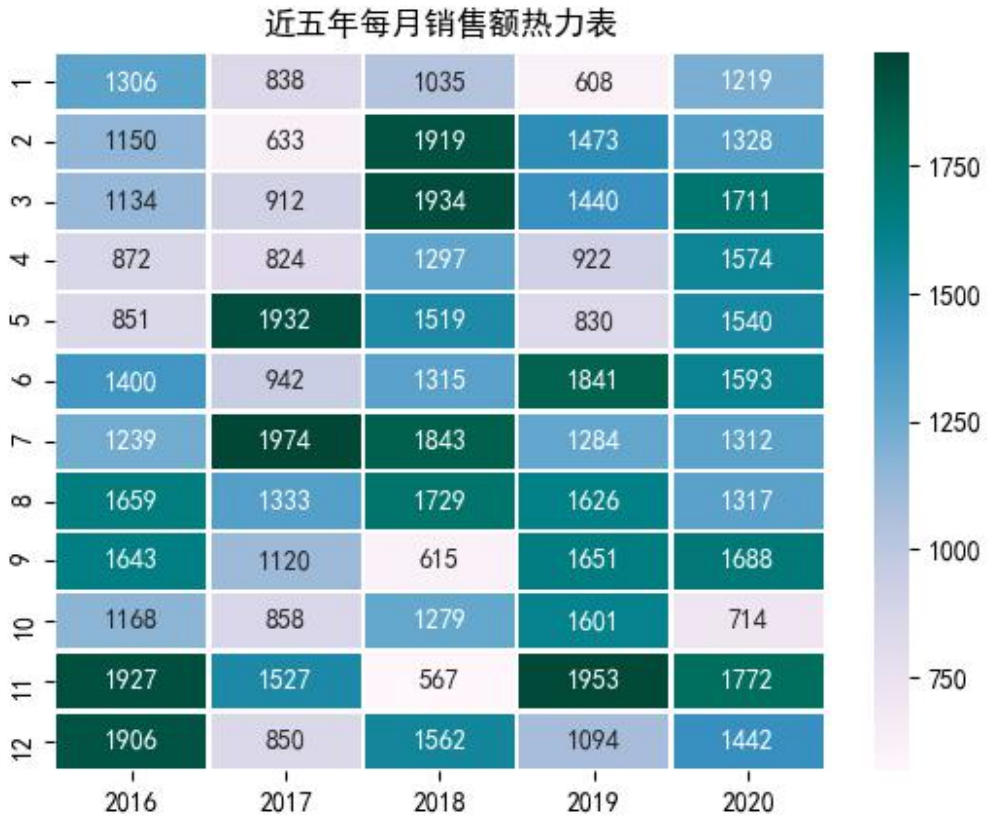
11.3.3. 热力图

热力图，也叫做交叉填充图，它的用法是实现列联表的可视化。通过图形的方式展现两

个离散变量之间的组合关系，

Seaborn 中的 heatmap 函数，可以完成热力图的绘制。

```
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
import random

# 设置中文字体
mpl.rcParams['font.sans-serif'] = ['SimHei']
data = pd.DataFrame({'2016':[random.randint(500,2000) for i in range(12)],
                     '2017': [random.randint(500, 2000) for i in range(12)],
                     '2018': [random.randint(500, 2000) for i in range(12)],
                     '2019': [random.randint(500, 2000) for i in range(12)],
                     '2020': [random.randint(500, 2000) for i in range(12)],},
                    index = [1,2,3,4,5,6,7,8,9,10,11,12]
                   )
print(data)
sns.heatmap(data = data,
            linewidths=1,
            cmap= 'PuBuGn', # 填充颜色
            annot = True,   # 显示数值
            fmt= ''         # 不用科学计数法
            )


plt.title("近五年每月销售额热力表")
plt.show()
输出:
     2016  2017  2018  2019  2020
1   1306   838  1035   608  1219
2   1150   633  1919  1473  1328
3   1134   912  1934  1440  1711
4    872   824  1297   922  1574
5    851  1932  1519   830  1540
6   1400   942  1315  1841  1593
7   1239  1974  1843  1284  1312
8   1659  1333  1729  1626  1317
9   1643  1120   615  1651  1688
10  1168   858  1279  1601   714
11  1927  1527   567  1953  1772
12  1906   850  1562  1094  1442
```



12. 上机作业题

12.1. 数值交换

要求：（1）给两个变量分别赋不同的整数值，然后利用程序交换两个变量的值。
代码：

```
value1 = int(input('请给变量 1 赋值：'))
value2 = int(input('请给变量 2 赋值：'))

print('你输入变量 1 的值是：',value1)
print('你输入变量 2 的值是：',value2)

# 交换两个变量的值
value3 = value1
value1 = value2
value2 = value3

print('交换后变量 1 的值是：',value1)
print('交换后变量 2 的值是：',value2)
```

12.2. 三数比较大小

题目：输入三个整数 x，y，z，请把这三个数由小到大输出。
代码：

```
value1 = int(input('请给变量 1 赋值：'))
value2 = int(input('请给变量 2 赋值：'))
value3 = int(input('请给变量 3 赋值：'))
value4 = 0

print('你输入的三个数分别是：',value1,',',value2,',',value3)

if value1 > value2 :
    value4 = value1
    value1 = value2
    value2 = value4

if value1 > value3 :
    value4 = value1
    value1 = value3
    value3 = value4

if value2 > value3:
    value4 = value2
    value2 = value3
    value3 = value4

print('从小到大输出：',value1,',',value2,',',value3)
```

12.3. 计算人体健康 BMI

要求：输入身高（米）和体重值（千克），计算体重指数 BMI，然后输出判断结果。指数小于 18.5，偏瘦；指数在 18.5 与 24.9 之间，正常；指数在这 24.9 与 29.9 之间，偏重；指数大于 29.9，肥胖。
BMI = 体重 /（身高 * 身高）
实现代码：

```
height = float(input('请输入你的身高（厘米）：'))
weight = float(input('请输入你的体重（千克）：'))

print('你的身高是：',height,'厘米')
print('你的体重是：',weight,'千克')
```

```
height /= 100  # 单位转换成米
bmi = weight/(height * height)

print('你的 BMI 指数是：',bmi)
if bmi < 18.5 :
    print('你的体重太轻了，偏瘦，要注意锻炼！')
elif bmi < 24.9 :
    print('你的体重非常完美，注意保持！')
elif bmi < 29.9 :
    print('你的体重偏重啦，请注意锻炼！')
else :
    print('请注意，你已经很胖啦，^@_@^')
```

12.4. 计算闰年

题目：判断某一年是不是闰年，整百年是 400 的倍数是闰年，不是整百年是 4 的倍数是闰年。
代码：

```
year = int(input("输入一个年份: "))
if (year % 4) == 0:
   if (year % 100) == 0:
       if (year % 400) == 0:
           print(year,"是闰年")  # 整百年能被 400 整除的是闰年
       else:
           print(year,"不是闰年")
   else:
       print(year,"是闰年")       # 非整百年能被 4 整除的为闰年
else:
   print(year,"不是闰年")
```

12.5. 回文

题目：输入一个字符串，判断其是否为回文。回文字符串是指从左到右读和从右到左读完全相同的字符串。

```
mystr = input('请输入要判断的英文字符串：')

length = len(mystr)
i = 0
flag = True
while i <= length/2:
    if mystr[i] == mystr[length - 1 - i]:
        i += 1
    else:
        flag =False
```

```
        break
if flag :
    print("你的字符串是回文")
else:
    print("你的字符串不是回文")
```

12.6. 九九乘法表

题目：打印九九乘法表

```
for i in range(1, 10):
    for j in range(1, i+1):
        #print('{}x{}={}\t'.format(j, i, i*j), end='')
        print(j,"*",i,"=",i*j, end='\t')
    print()
```

12.7. 自然数求和

要求：求前 N 个自然数据的和。
代码：

```
value = int(input('你要计算前多少个自然数的和：'))
i = 0
sum = 0
while i <= value:
    sum += i
    i += 1
print('前',value,'个自然数的和是：',sum)
```

12.8. 计算阶乘和

题目：计算 1! + 2! + 3! + …… + N!

```
myvalue = int(input("请输入你要计算前几个数的阶乘之和："))
sum = 0
fac = 1
for i in range(1 , myvalue + 1):
    fac = 1
    for j in range(1 , i + 1):
        fac = fac * j
    sum += fac
```

```
print("前",myvalue,"个自然数的阶乘之和是：",sum)
```

12.9. 鸡兔同笼

题目：今有鸡兔同笼，上有 35 头，下有 94 足，鸡兔分别有几只？
代码：

```
chicken = 0
rabbat = 35 - chicken

while chicken <= 35 :
    if 2 * chicken + 4 * rabbat == 94 :
        print("**************************")
        print("找到答案")
        print("鸡的数量是：",chicken)
        print("兔的数量是：",rabbat)
        print("**************************")
    chicken += 1
    rabbat = 35 - chicken
```

12.10. 百人分百饼

题目：有一百个人，分一百个饼，其中，大人每人分 3 个，小孩子 3 个人分 1 个，请问有多少个小孩，多少个大人。
代码：

```
child = 0
adult = 100 - child
while child <= 100 :
    # print("尝试：",child,',大人：',adult)
    if 3 * adult + child/3 == 100 :
        print("找到答案：")
        print("小孩的人数是：",child)
        print("大人的人数是：",adult)
    child += 1
    adult = 100 - child
```

12.11. 最大最小值

题目：利用循环比较的方法，打出一个列表中的最大值和最小值，不要用 max 函数和 min 函。

```
import random
mylist =[]
# 给列表随机赋值
for item in range(0,10):
    mylist.append(random.randint(10,100))

print("随机生成的数值列表：", mylist)

maxnum = mylist[1];
minnum = mylist[1];

for item in mylist:
    if item < minnum:
        minnum = item
    if item > maxnum:
        maxnum = item

print("最大值是：", maxnum)
print("最小值是：", minnum)
print('利用 max 函数所得最大值是：',max(mylist))
print('利用 min 函数所得最大值是：',min(mylist))
```

12.12. 换零钱

题目：把 1 元钱换成 1 分、2 分和 5 分的硬币，请问有多少种换法？
代码：

```
coin1 = 0
coin2 = 0
coin5 = 0

max1 = 100
max2 = int(100/2)
max5 =int( 100/5)

count = 0

while coin1 <= max1:
    coin2 = 0
    while coin2 <= max2:
        coin5 = 0
        while coin5 <= max5:
            if coin1 + 2*coin2 +5*coin5 == 100 :
                count += 1
                print('方案',count,':\t 1 分',coin1,'个\t2 分',coin2,'个\t5 分:
```

```
',coin5,'个')
            coin5 += 1
        coin2 += 1
    coin1 += 1
print('总方案数：',count)

'''
for coin1 in range(max1+1):
    for coin2 in range(max2+1):
        for coin5 in range(max5+1):
            if coin1 + 2*coin2 +5*coin5 == 100 :
                count += 1
                print('方案',count,':\t 1 分',coin1,'个\t2 分',coin2,'个\t5 分：
',coin5,'个')
'''
```

12.13. 登录功能

题目：模拟实现一个系统登录功能，最多允许输错 4 次；

```
username = "admin"
password = "abc888"
flag = False
trytimes = 0

while not flag:
    myname = input("请输入你的用户名：")
    mypassword = input("请输入你的密码：")
    trytimes += 1
    if myname == username and mypassword == password:
        flag = True
        print('您登录成功，欢迎你进入系统')
    else:
        print('用户名错误或者密码错误')
        print('你还有{:d}次尝试机会'.format(4 - trytimes))
        if trytimes >= 4:
            print('你已经输错 4 次，为了账户安全，账号被锁定')
            break
```

12.14. 修改密码功能

题目：模拟用户修改密码的功能。

```
username = "admin"
password = "abc888"
flag = False
trytimes = 0

while not flag:
    oldpassword  = input("请输入你的旧密码：")
    trytimes += 1
    if oldpassword == password:
        while not flag:
            newpassword1 = input("请输入你的新密码:")
            newpassword2 = input("请再次输入你的新密码:")
            if newpassword1 == newpassword2:
                password = newpassword1
                print('你的密码修改成功')
                flag = True
            else:
                print('你两次输入的密码不一致，请重新输入')
    else:
        print('你输入的旧密码错误，你还有{:d}次尝试机会'.format(4 - trytimes))
        if trytimes >= 4:
            print('你已经输错 4 次，为了账户安全，账号被锁定')
            break
```

12.15. 质因数分解

题目：输出一个大的整数，然后对这个数进行因数分解。

```
number = int(input("请输入一个要进行质因数分析的数字："))
mylist = []
test = 2
while test <= number :
    if number % test == 0:
        mylist.append(test)
        number /= test
        continue
    if number == 1:
        break;
    test += 1
print(mylist)
```

12.16. 冒泡排序

冒泡排序的思想: 每次比较两个相邻的元素, 如果他们的顺序错误就把他们交换位置。
冒泡排序原理: 每一趟只能将一个数归位, 如果有 n 个数进行排序,只需将 n-1 个数归位, 也就是说要进行 n-1 趟操作(已经归位的数不用再比较)
缺点: 冒泡排序解决了桶排序浪费空间的问题, 但是冒泡排序的效率特别低
比如有五个数: 12, 35, 99, 18, 76, 从大到小排序, 对相邻的两位进行比较。
第一趟:
第一次比较: 35, 12, 99, 18, 76
第二次比较: 35, 99, 12, 18, 76
第三次比较: 35, 99, 18, 12, 76
第四次比较: 35, 99, 18, 76, 12
经过第一趟比较后, 五个数中最小的数已经在最后面了, 接下来只比较前四个数, 依次类推
第二趟
99, 35, 76, 18, 12
第三趟
99, 76, 35, 18, 12
第四趟
99, 76, 35, 18, 12
比较完成

```
import random

mylist =[]
# 给列表随机赋值
for item in range(0,10):
    mylist.append(random.randint(10,100))

print("随机生成的数值列表: ", mylist)

for i in range(len(mylist) - 1):          # 这个循环负责设置冒泡排序进行的次数
    for j in range(len(mylist) - i - 1):  # j 为列表下标
        if mylist[j] > mylist[j + 1]:
            mylist[j], mylist[j + 1] = mylist[j + 1], mylist[j]
    print('第',i+1,"趟排序后的数值列表: ", mylist)    #输出每一趟排序的结果

print("冒泡排序后的数值列表: ", mylist)
```

12.17. 约瑟夫问题

题目: 约瑟夫问题是一个有趣的数学游戏, 游戏规则是, N 个人围成一个圈, 编号从 1 开始, 依次到 N。编号为 M 的游戏参与者开始报数, 报数从 1 开始, 后面的人报数接龙, 直到 K

为止，报数为 K 的人将出局。出局者的下一个玩家接着从 1 开始报数，如此循环，直到剩下一个玩家时游戏结束，这个玩家就是游戏获胜者。哪个编号是游戏获胜者呢？

```
def josephus1(n,k):
    link = list(range(1,n+1))     #初始化列表
    print(link)
    killnum = 0
    while len(link) > 1:
        killnum += k - 1
        if killnum >= len(link):
            killnum %=  len(link)

        print('本轮杀死：', link[killnum])
        del link[killnum]
    print('最后活下来的是：',link[0])

josephus1(5,3)
josephus1(10,3)
```

12.18. 递归求自然数和

```
def sum(n):
    if n==1:
        return 1
    else:
        return n + sum(n-1)

print("前 100 个自然数之和：",sum(100))
```

12.19. 兔子数目

题目：有一对兔子，从出生后第 3 个月起每个月都生一对兔子，小兔子长到第三个月后每个月又生一对兔子，假如兔子都不死，问每个月的兔子总数为多少？
首先，得拿起你的纸笔演算出有限数量的变化规律，也就是分析:
第一个月----------1
第二个月----------1
第三个月----------2
第四个月----------3
第五个月----------5
第六个月----------8
第七个月----------13
然后通过你的细心观察，终于发现，除前两个月外，每个月的兔子数都是前两个月的兔子数

之和，如何你再有点想象力，假设有个第 0 月，兔子数是 0，那么除第一个月外每个月的兔子数量都是前两个月的兔子数量之和，所以得到了一个规律：

斐波那契数列, $S_n = S_{n-1}+S_{n-2}$

```
def calcu( n ):
    if n == 1:
        return 1
    elif n == 2:
        return 1
    else:
        return calcu(n - 1) + calcu(n - 2)

print('一个月兔子的数量：',calcu(1))
print('二个月兔子的数量：',calcu(2))
print('三个月兔子的数量：',calcu(3))
print('四个月兔子的数量：',calcu(4))
print('五个月兔子的数量：',calcu(5))
print('六个月兔子的数量：',calcu(6))
print('七个月兔子的数量：',calcu(7))
print('八个月兔子的数量：',calcu(8))
print('九个月兔子的数量：',calcu(9))
print('十个月兔子的数量：',calcu(10))
```

12.20. 猜幸运数字

题目：系统随机生成一个 1-1000 之间的数字，然后用户开始猜，程序提示大了或者是小了，给 10 次猜数字的机会。

```
import random

result = random.randint(1, 999)
success = False
print("你有十次机会猜幸运数字(1--1000)")
for i in range(1,11):
    guess = int(input("请输入你猜的数字:"))
    if guess == result:
        print("你运气真好，猜对了，幸运数字就是：",result)
        success = True
        break
    if(guess < result):
        print("你输入的数字太小了：", guess)
    elif(guess > result):
        print("你输入的数字太大了：", guess)
    print("你还有",10-i,"次机会")
```

```
if not success:
    print("你没有猜出幸运数字，幸福数字是：", result)
```

12.21. 押大小游戏

题目：赌大小游戏
代码：

```
import random

name = input('请输入你的名字：')
money = 5000
print('你总共有', money, '元')

while money > 0:
    stake = int(input('请输入下注金额:'))
    print('你下注：', stake)
    print()

    if stake > money:
        print('你没有这么多钱下注，你只剩下：', str(money))
        continue
    print('输入 1 表示押大，输入 0 表示小')
    num = int(input('押大还是押小:'))
    if num == 0:
        print('你此局押小')
    else:
        print('你此局押大')
    print()

    randomNum = random.randint(0, 1)
    print()
    print('*****************')
    if randomNum == 0:
        print('*** 此局开小 ***')
    else:
        print('*** 此局开大 ***')
    print('*****************')
    print()
    if num == randomNum:
        money += stake
        print(name, '你赢了，你现在的余额为:', money)
```

```
    else:
        money -= stake
        print(name, '你输了，你现在的余额为:', money)
    print('**************************************')
    print()
    if money > 0:
        print(name, '，赌局继续')
    elif money > 10000:
        print(name, '，你今天运气真好，不跟你玩了')
        break

print(name, ',运气真背，你输光了，明天再来吧')
```

12.22. 五猴分桃

题目：5 只猴子一起摘了 1 堆桃子，因为太累了，它们商量决定，先睡一觉再分。过了不知多久，来了 1 只猴子，它见别的猴子没来，便将这 1 堆桃子平均分成 5 份，结果多了 1 个，就将多的这个吃了，拿走其中的 1 堆。又过了不知多久，第 2 只猴子来了，他不知道有 1 个同伴已经来过，还以为自己是第 1 个到的呢，于是将地上的桃子堆起来，平均分成 5 份，发现也多了 1 个，同样吃了这 1 个，拿走其中的 1 堆。第 3 只、第 4 只、第 5 只猴子都是这样。问这 5 只猴子至少摘了多少个桃子?

```
def apple(n):
    """
    判断苹果数是不是符合题义
    :param n:
    :return:
    """
    for i in range(1,6):
        n = n - 1               # 猴子吃掉一个
        if n % 5 == 0:          # 如果能分成五份
            n = n - n / 5
            continue
        else:                   # 如果不能分成五份，则返回 False
            return False
    return True

isright = False
trynum = 1
while not isright:
    if apple(trynum):
        isright = True
        break
    trynum += 1
```

```
print("最小的苹果数为：",trynum)
for i in range(1, 6):
    trynum = trynum - 1
    trynum = trynum - trynum / 5
    print("第{:d}猴子吃 1 个，拿走：{:.0f}个，还剩{:.0f}".format(i,int(trynum/4),
trynum))
```

12.23. 打印全年的日历

题目：输入一个年份，要求按日历格式打印出一年的所有日历。

```
import datetime

def isleapyear(year):
    """
    用于判断一年是不是闰年
    """
    if year % 4 == 0:             # 如果是 4 的倍数
        if year % 100 == 0:       # 如果是 100 倍数
            if year % 400 == 0:   # 如果是 400 倍数
                return True
            else:
                return False
        else:
            return True
    else:
        return False

year = int(input("请输入你要打印的年份："))

# 用来保存每个月有多少天数
listmonth = [31,28,31,30,31,30,31,31,30,31,30,31]

# 确定 2 月具体的天数
if isleapyear(year):
    listmonth[1] = 29
else:
    listmonth[1] = 28

# 用来计数，每七个换行
count = 0

# 确定第一个月第一天开始打印的位置
```

```
start = datetime.datetime(year,1,1).weekday()

#需要打印的日期字符串
daystr = ""

for month in range(0,12):
    print("---------------------------")
    print("公元{}年{}月: {}天".format(year , month + 1,listmonth[month]))
    print("---------------------------")
    print("一\t 二\t 三\t 四\t 五\t 六\t 日")
    daystr = ""
    count  = 1
    for day in range(1,listmonth[month]+1):
        if day == 1:                    # 如果是第一天，处理第一行
            daystr = "\t" * start
            count += start              # 添加多少空格，计数需要增加多少
            count %= 7

        if count == 0:                   # 如果计数满 7 个
            daystr += str(day)
            print(daystr)                # 打印
            daystr = ""                  # 打印后把日期字符串设置为空
        else:
            daystr += str(day) + "\t"  # 计数不满 7 个，直接把日期接在后面

        if day == listmonth[month]:  # 如果是最后一天，处理最后一行
            if len(daystr) != 0:       # 如果打印字符串为空，则不打印
                print(daystr)
            start = count              # 下个月开始打印的位置
            count += 7 - start         #  最后一行剩余空格，则计数需要增加
        count += 1                     # 每个循环计数增加 1
        count %= 7
```