《Python与数据挖掘》

第2章 Python基础入门

讲师：武永亮

目录

Python基础概述

- 本章是Python的基础章节，可以在这章中学习到丰富的Python基础知识。首先我们会从操作符和最简单的数字数据入手，然后就是流程控制，到这里能够对Python程序结构有一个清晰的认识。
- 接着是较复杂的数据结构，主要涉及Python的最常用5大内建数据类型：列表，字符串，元组，字典和集合。
- 这部分重点对这些数据结构的用法进行讲述，由于内容有限，并没有太多涉及它们的时间复杂度，空间复杂度和源码编写。我们并不认为这是可以忽略的，建议查阅其他资料对数据结构的复杂度有一定的认识。

操作符分类

- Python的常用操作符可分为4种，分别为算术操作符，赋值操作符，比较操作符和逻辑操作符。
- 算术操作符一般会返回一个数，而比较和逻辑操作符会返回布尔值True或False。
- 我们需要注意操作符的运算优先级，否则将得到与我们预料不到的结果。如果想改变运算的优先级，可以使用小括号。下面将逐一介绍每种操作符。

算术操作符

- 值得注意的是取商运算和除法运算。如果除号两侧的值都是整数，那么得到的结果是一个向下取整的整数。如果其中一个是浮点数，那么得到的结果最多保留17位有效数字。而取商运算正好是前面的相反，无论'//'两侧的值是浮点数和还是整数，返回的结果都会向下取整，但其数据类型是有一位小数点0的浮点数。

操作符	描述	实例
+	加法-返回两操作数相加的结果	3+2返回5
-	减法-返回左操作数减去右操作数的结果	3-2返回1
*	乘法-返回两操作数相乘的结果	3*2返回6
/	除法-返回右操作数除左操作数的结果	3/2返回1但3.0/2返回1.5
%	模-返回右操作数对左操作数取模的结果	5/3返回2
**	指数-执行对操作指数的计算	3**2返回9
//	取商-返回右操作数对左操作数取商的结果	3.0/2返回1.0

赋值操作符

- 赋值操作符主要是’ =’ ，其他都是运算操作符和’ =’ 的结合，其存在意义都是简化代码。

操作符	描述	例子
=	简单的赋值运算符，赋值从右侧操作数左侧操作数	c=a+b将 a和b相加的值赋值给 c
+=	加法AND赋值操作符，它增加了右操作数左操作数和结果赋给左操作数	c += a 相当于 c = c + a
-=	减法AND赋值操作符，它减去右边的操作数从左边操作数，并将结果赋给左操作数	c -= a 相当于 c = c - a
*=	乘法AND赋值操作符，它乘以右边的操作数与左操作数，并将结果赋给左操作数	c *= a 相当于 c = c * a
/=	除法AND赋值操作符，它把左操作数与正确的操作数，并将结果赋给左操作数	c /= a 相当于c = c / a
%=	模量AND赋值操作符，它需要使用两个操作数的模量和分配结果左操作数	c %= a相当于c = c % a
**=	指数AND赋值运算符，执行指数（功率）计算操作符和赋值给左操作数	c **= a 相当于 c = c ** a
//=	取商，并分配一个值，执行取商并将结果赋值给左操作数	c //= a 相当于 c = c // a

比较操作符

- Python的比较操作符与Java和C类似，同样很简单。

操作符	描述	实例
==	如果两个操作数的值相等则返回True，否则返回False	3==2返回False
!=	如果两个操作数的值不等则返回True，否则返回False	3!=2返回True
<>	与!=效果相同	3<>2返回True
>	如果左操作数大于右操作数则返回True，否则返回False	3>2返回True
<	如果左操作数小于右操作数则返回True，否则返回False	3<2返回False
>=	如果左操作数大于或等于右操作数则返回True，否则返回False	3>=3返回True
<=	如果左操作数小于或等于右操作数则返回True，否则返回False	2<=2返回True

逻辑操作符

- Python的逻辑操作符有and,or,not，分别对应逻辑学的与，或，非。逻辑操作符的两端一般是布尔值数据。

操作符	描述	实例
and	逻辑与运算符。当且仅当两个操作数为真则返回真，否则返回假。	True and False返回False
or	逻辑或运算符。当且仅当有两个操作数至少一个为真则返回真，否则返回假	True and False返回True
not	逻辑非运算符。用于反转操作数的逻辑状态。	not True 返回False

操作符优先级

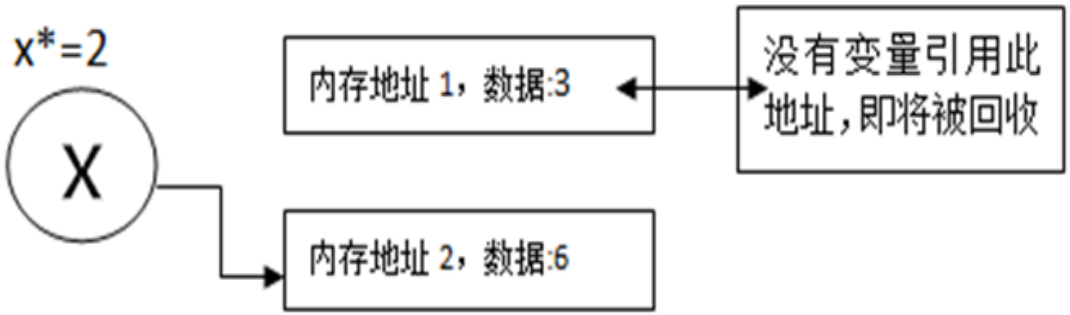
- 下表列出了上面提及的操作符从最高优先级到最低。

操作符	描述
**	幂
* / % //	乘，除，取模，取商
+ -	加，减
<= <>>>=	比较操作符
<> == !=	比较操作符
= %= /= //= -= += *= **=	赋值操作符
in not in	成员操作符
not or and	逻辑操作符

目录

变量与赋值

- 变量是我们广为熟悉的概念。程序语言中的变量和数学上的变量类似，如果需要对x赋值3，执行下面语句：>>> x = 3
- 这样程序将会为变量x申请地址并存储它。’ =’ 的这个操作称为赋值。
- 如果变量的值发生改变，Python会自动创建另一个对象申请另一块的内存，并改变变量的对象引用。这样做的优点是减少了重复的值对内存空间的占用，而缺点则是每次修改变量都需要重新开辟内存单元，给执行效率带来一定影响。
- 变量与地址的关系如下：



数字数据类型

- 数字的基本数据类型可分为整数，浮点数，布尔值。创建变量时，Python不需要声明数据类型，Python能够自动识别数据类型。x=3的数据类型是整数而x=3.3的数据类型是浮点数，函数type(x)可以查看数据的数据类型。布尔值只有True和False两种值，支持and not or 三种运算。
- 和数学运算不同的地方是，Python的整数结果仍然是整数，如果操作符两端其中一个操作数是浮点数，那么运算结果是浮点数。

目录

流程控制

- 流程控制是一门程序语言的基本，掌握Python流程控制语句就已经能够实现很多算法了。
- 本节主要介绍Python的条件分支结构if语句和两种主要循环结构while语句和for语句，并在最后详细讲解Python函数的用法。
- 如果有一定的编程基础，对条件分支，循环和函数这3种结构比较熟。

条件语句

- 如果要你的任务是输出两个数a和b之间的较大者，那么你的思路应该是这样的：如果a大于b：输出a　　否则：输出b
- 如果想通过Python实现上面的思想，就必须借助if语句实现条件分支。
- 布尔表达式：在前面我们简单介绍过布尔值，而布尔表达式是返回一个布尔值(或称为真值)的表达式。
- 在表达式运算的过程中，True会视为数值1，False会视为数值0，这与其他编程语言是相似的。逻辑表达式是布尔表达式的一种，逻辑表达式指的带逻辑操作符或比较操作符(如>,==)的表达式。逻辑表达式返回的是False或者True。

While语句

- 我们现有的知识要让程序重复地做一件事，那么我们只能够重复地写相同的代码，显然这不合理。为此，我们需要掌握一个重要的概念—循环。while循环是最常用的循环之一，它的格式如下：
- while 布尔表达式：
  - 程序段
- 只要布尔表达式为真，那么程序段将会被执行，执行完毕后，再次计算布尔表达式，如果结果仍然为真，那么再次执行程序段，直至布尔表达式为假。

break语句和continue语句

- 下面看两个简单的语句，它们只有嵌套在循环中才起作用，分别是break语句和continue语句。它们的作用如下：
- break：跳出最内层循环
- continue：跳到最内层循环的首行
- 简单来说，break用于中止循环，注意如果你一个while语句嵌套在另一个while语句内，即程序中有双层循环，内层循环中的break语句仅仅退出内层循环并回到外层循环。而continue语句是中断当前的循环并回到循环段的开头重新执行程序。

for循环

- for循环在Python中是一个通用的序列迭代器，可以遍历任何有序的序列。for语句可作用于字符串、列表、元组，这些数据结构在3.4节将会详细介绍，本节我们的例子需要用到这些数据结构。程序语言的学习是一个循环的学习过程，与其他学科不同，程序语言的知识是相互紧扣的。
- Python中的for语句接受可迭代对象，如序列和迭代器作为其参数，每次循环调取其中一个元素。
- 如果你希望Python能像C语言的格式进行循环，range()函数能够快速能成一个数字序列。

目录

数据类型

- Python中的绝大部分数据结构可以被最终分解为三种类型：标量（Scaler），序列（Sequence），映射（Mapping）。这表明了数据存储时所需的基本单位，其重要性如同欧式几何公理之于欧式空间。
- 序列是Python中最为基础的内建类型。它分为七种类型：列表、字符串、元组、Unicode字符串、字节数组、缓冲区和xrange对象。常用的是：列表（List）、字符串（String）、元组（Tuple）。
- 映射在Python的实现是数据结构字典（Dictionary）。作为第三种基本单位，映射的灵活使得它在多种场合中都有广泛的应用和良好的可拓展性。
- 集合（Set）是独立于标量，序列和映射之外的特殊数据结构，它支持数学理论的各种集合的运算。它的存在使得用程序代码实现数学理论变得方便。

列表

- 列表（List）是一个任意类型的对象的位置相关的有序集合。它没有固定的大小，更准确的说，它的大小是可变的。通过对偏移量进行赋值以及其他各种列表的方法进行调用，能够修改列表的大小和内容。
- 列表是序列的一种，Python的列表元素没有固定数据类型约束。列表是有序的，可以直接通过下标(即索引)访问其元素。注意下标是从0开始，Python的下标允许是负数，例如List2[-1]表示List2从后往前数的第一个元素。除了索引，列表支持切片。切片返回一个子列表。切片的索引有两个默认值，第一个索引默认为零，第二个索引默认为切片的列表的大小。

列表函数

- Python的列表与其他语言的数组有些类似，但是Python的列表强大得多，它具有很多灵活的函数。它能够做到像字符串一样自由插入，查找与合并。

函数名称	函数说明
list.append(x)	添加一个元素到列表的末尾，相当于a[len(a):]=[x]
list.extend(L)	将参数中的列表添加到自身的列表的末尾，相当于a[len(a):]=L
list.insert(i,x)	在下标为i的元素位置前插入一个元素，所以a.insert(0,x)相当于a.append(x)
list.remove(x)	删除列表第一个值为x的元素。如果没有这样的元素会报错。
list.pop([i])	删除列表指定位置的元素并返回它。[]表示这个参数是可选的，如果不输入这个参数，将删除并返回列表最后一个元素。
list.index(x)	返回列表第一个值为x的元素的下标。如果没有这样的元素会报错。
list.count(x)	返回列表中x出现的次数
list.sort(cmp=None,key=None,reverse=False)	排序列表中的元素，可参考2.4.4节字典的遍历的代码，里面讲述了一个使用sort()函数的例子。
list.reverse()	反转列表中的元素

字符串

- 字符串（String）是序列的一种，支持其中索引的操作。实际上，字符串是单个字符的字符串的序列。在所有编程语言中，字符串都是最基本的数据结构之一。在Python之中，字符串灵活的方法大大简化了程序。
- 虽然字符串和列表都可以通过[]来访问其中的有序数据，但是字符串具有不可变性，每个字符一旦创建，不能通过索引对其作任何修改。
- 字符串与列表一样，支持索引和切片。
- 创建字符串最简单的是用单引号或双引号，这两种方法几乎没有区别。

字符串函数

- Python字符串方法众多,能够满足程序员的各种要求。这里仅仅列出一些必需掌握的最重要的方法和相应的例子，如下表。值得注意的是，count和join方法在列表和字符串中都存在，且功能类似。

函数名称	函数说明
S.find(sub,[,start[,end]])	返回在字符串中找到的子字符串sub的最低索引，使得sub包含在切片s[start:end]中，如果未找到sub，则返回-1
S.split([sep[,maxsplit]])	返回字符串中的单词列表，使用sep作为分隔符字符串。如果给出maxsplit，则至多拆分maxsplit次（因此，列表中将最多有maxsplit+1个元素）。如果没有指定maxsplit或为-1，那么分割的数量没有限制（进行所有可能的分割）。
S.join(iterator)	连接字符串数组。将字符串、元组、列表中的元素以指定的字符(分隔符)连接生成一个新的字符串
S.strip([chars])	返回字符串的一个副本，删除前导和尾随字符。chars参数是一个字符串，指定要移除的字符集。如果省略或为None，则chars参数默认为删除空白字符。
S.lower()	将字符串所有大写字符变为小写
S.isalnum()	如果字符串至少有一个字符，并且所有字符都是数字或者字母，则返回true，否则返回false。
S.count(sub[,start[,end]])	返回在[start, end]范围内的子串sub非重叠出现的次数。可选参数start和end都以切片表示法解释。
S.replace(old,new[,count])	返回字符串的一个拷贝，其中所有的子串old通过new替换。如果指定了可选参数count，则只有前面的count个出现被替换。

元组

- 元组（Tuple）与列表和字符串一样，是序列的一种。而元组与列表的唯一不同的元组不能修改，元组和字符串都具有不可变性。
- 创建元组：元组没有固定的数据类型约束，它们编写在圆括号而不是方括号中，它们支持常见的序列操作。
- 元组有很多与列表相同的方法，但必须留意的是，append()和pop()等修改大小和内容的函数是元组不允许的。
- 元组的不可变性是关键，从某个角度说是它的天然优势。
- 例如Python的字典(后面会详细介绍)允许元组和字符串作为键值，但不允许列表。
- 原因就是元组和字符串的不可变性，字典的键值是必须保证唯一的。元组提供了一种完整性约束，这对于大型程序的编写是很重要的。

字典

- 字典（Dictionary）是基础数据结构映射(Mapping)的一种。序列是按照顺序来存储数据的，而字典是通过键存储数据。字典的内部实现是基于二叉树(Binary Tree)的，数据没有严格的顺序。字典将键映射到值，通过键来调取数据。如果键值本来是有序的，那么我们不应该使用字典，如映射：

$$\begin{cases} 1 \rightarrow A \\ 2 \rightarrow B \\ 3 \rightarrow C \end{cases}$$

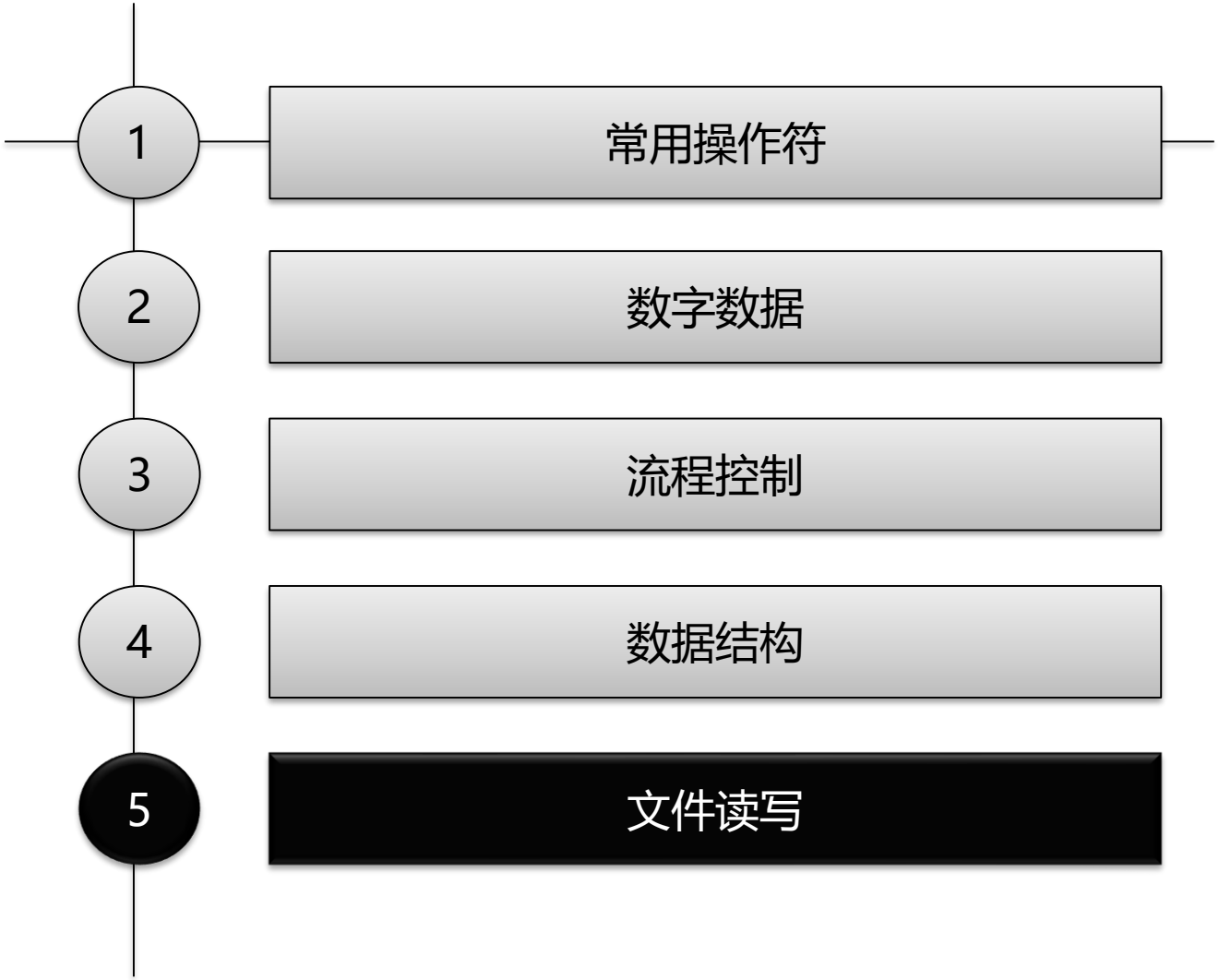
- 直接用列表[‘A’ ,’ B’ ,’ C’]即可，字典的效率比列表差得多。但是在很多情形下，字典比列表更加适用。比如我们手机的通讯录（假设人名均不相同）可以使用字典实现，把人的名字映射到一个电话号码，由于名字是无序的，不能直接用一个列表实现，使用字典直接高效。

集合

- Python有一种特殊的数据类型称为集合(Set)。之所以称它为特殊，是因为它既不是序列也不是映射类型，更不是标量。集合是自成一体的类型。集合是唯一的，不可变的对象是一个无序集合。集合对象支持与数学理论相对应的操作，如并和交，这也是这种数据类型被创建的最重要的目的。
- 集合能够通过表达式操作符支持一般的数学集合运算。这是集合特有的操作，序列和映射不支持这样的表达式。

表达式	结果	意义
x-y	set([1])	集合的差，返回包含在x且不包含在y中的元素集合
x|y	set([1, 2, 3, 4])	集合的并，返回包含在x或y中的元素集合
x&y	set([2, 3])	集合的交，返回既包含在x也在y的中的元素的集合
x^y	set([1, 4])	集合的异或，返回只被x包含或只被y包含的元素的集合
x>y	False	如果x真包含y，则返回True，否则返回False

目录



文件读写

- 文件访问是一门语言重要的一环，适当地进行文本读写能够保存一次程序运行下来的结果。
- 在数据挖掘的工作中，数据量很大，整个挖掘程序可以分为几部分，我们应该把每一部分运行的结果都保存下来，如果后面的程序出现错误，我们也不必再从头开始。
- 要进行文件的读写，首先要设置工作目录。如果使用脚本运行，那么默认的工作目录为脚本所在的目录。
- 要改变工作目录，首先要引入os模块，语句为：import os。查看当前工作目录的方法是os.getwd()，改变工作目录的方法是os.chdir(string)。

txt文件读取

- Python进行文件读写的函数是open或file。其格式如下：
  - file_handler = open(filename,mode=’r’)
- 其中filename是我们希望打开的文件的字符串名字,mode表示我们的读写模式，默认为read模式。如果此语句执行成功，那么一个文件句柄就会返回，后面的文件操作需依赖文件句柄的方法进行。
- 我们常用的文件读入函数是readline()和readlines()。
- 首先我们假设在我们脚本目录下有这样一个data.txt,其数据如下：
  - 1,2　　3,4
- 注意第一行中有一个换行符。如果我们采用readline()语句读取，执行f=open(‘data.txt’,’r’)和 a =f. readline()，那么就会将第一行以字符串的形式返回，此时a=’1,2\n’。

txt文件读取

- 同时文件指针指向第一行末尾，如果再执行语句b = f.readline()，那么b=’ 3,4’ ，此时文件指针就指向文件末尾，文件已读取完毕。可以使用下面的while循环读取所有语句：L=2#文件的行数
- for i in range(L):
  - a = readline()# 对该行的处理
- 如果我们想去掉第一行的读取的换行符，可以使用语a=a.strip(),strip()是一个可以去掉一个字符串开头和末尾的空白字符，包括换行符。
- 而readlines则返回一个列表，列表的包含了每一行的字符串数据。如执行a=f.readlines(),那么此时a=[‘1,2\n’ ,’ 3,4’]。最终保存的形式是一个二维列表，在后面的数据处理可以很容易的变换为numpy.array，大部分数据挖掘的算法都需要numpy.array作为数据存储的格式。

csv文件读取

- 我们习惯使用excel表存储数据，但excel表数据直接用Python读取是行不通的。
- 一个常用的办法是将文件另存为csv文件格式。csv是逗号分隔符的数据表，每两个数据单元间用逗号分隔，实际上和txt文件没有本质的区别。
- 数据文件的数据是用制表符分隔的，如果改成用逗号分隔，再把后缀名改成csv，那就转换成了csv文件。同理，csv文件读取的处理与txt几乎一样，使用语句f=open(‘data.csv’)读取,这里不再举例累赘阐述。
- 如果我们使用pandas模块，那么读入csv文件更快捷方便，直接使用pandas.read_csv()方法即可。

文件输出

- 我们把数据成功读入到程序中，现在我们考虑，假设我们的程序中得出了一个二维列表，我们重新输出到文件。
- 我们可以使用方法f.write(string)，并且借助字符串的join方法输出到文件中。
- 如果二维列表的元素不是字符类型而是整数类型，我们不能使用join方法，使用f.write(string)输出比较麻烦，这里介绍另一中更灵活的输出到文件的方式：print>>>f,...。这样就会把原本print函数输出到shell的内容改为输出到文件中。

JSON处理数据

- 保存数值型数据比保存字符串类型的数据容易得多。因为wtite(string)方法只能输出字符串，且read()函数只会返回字符串，想转化为数值型数据需用int()这样的函数。
- 当想保存想列表和字典这样复杂的数据结构时，单靠read()和write()去人工解析是很困难的。幸运的是，Python允许用户使用常用的数据交换格式JSON（JavaScript Object Noation）。
- 标准模块json可以接受Python数据结构，并将它们转换为字符串表示形式，此过程称为序列化(Serialize)。从字符串表示形式重新构建数据结构称为反序列化(Deserialize)。序列化和反序列化的过程中，表示该对象的字符串可以存储在文件中。

Thank You!