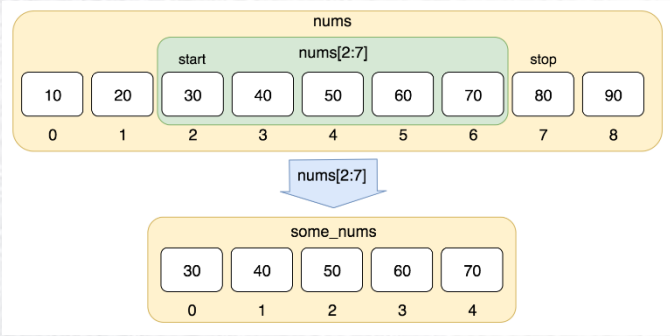
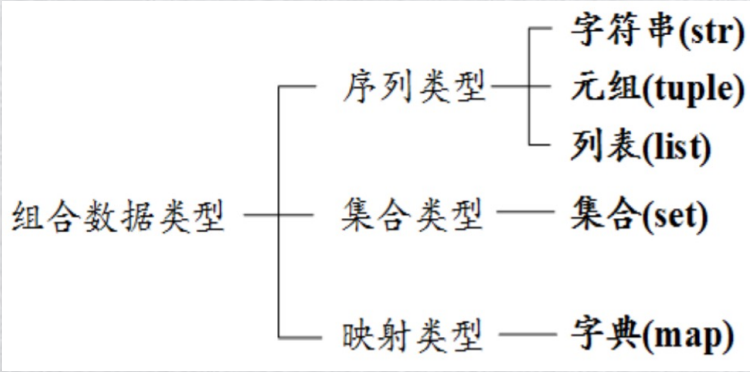
大数据分析导论——IO、多维数据、类和对象

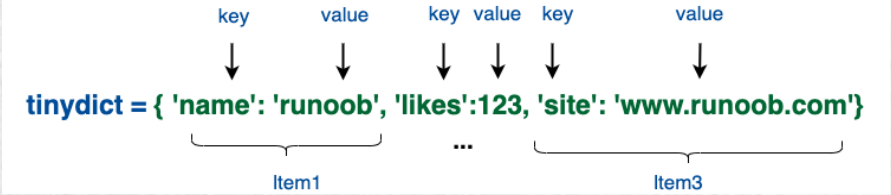
上次课回顾：组合数据类型

- 列表



- 字典





大数据分析导论
IO编程

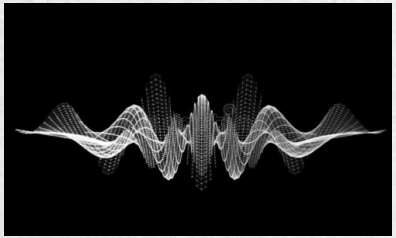
提纲

- IO编程
- 多维数据的格式化与处理
- 类和对象
- 两个高级程序

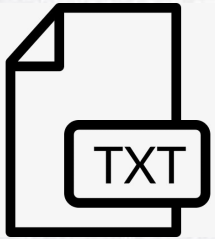
文件概述与读写

- 文件概述
  - 文件是存储在外部介质（存储器）上的数据序列，可以包含多种数据内容，例如程序文件、数字图像、音频、视频等。
  - 文件是数据的集合和抽象，如图像是像素数据的集合，文本是文字数据的集合
  - 文件是一种有序的数据序列
  - 文件包括两种类型：文本文件和二进制文件

图像文件.jpg



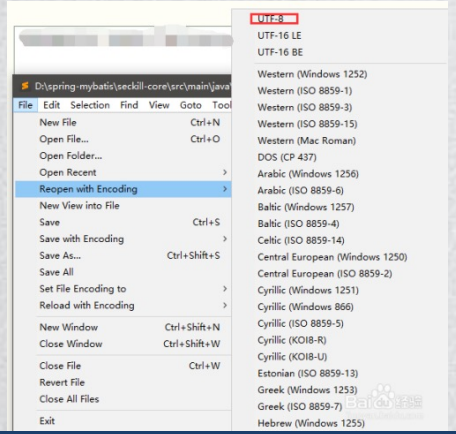
音频文件.wav



文本文件.txt

文件概述与读写

- 文本文件
  - 一般由单一特定编码的字符组成，如UTF-8编码、ASCII编码、GBK编码等。
  - 文本文件可以通过文本编辑软件或文字处理软件创建、修改和阅读，如sublime。
  - 文本文件可以看做为存储在外部介质上的长字符串。




```
1  #!/usr/bin/python
2
3  print ("你好，世界")
4
5
6
```

```
SyntaxError: Non-ASCII character '\xe4' in file test.py on line 2, but no encodi
ng declared; see http://python.org/dev/peps/pep-0263/ for details
```

```
1  #!/usr/bin/python
2  # -*- coding: UTF-8 -*-
3
4  print ("你好，世界")
5
6
```

```
你好，世界
```

文件概述与读写

- 二进制文件
  - 由比特1和比特0所组成，没有统一的字符编码，文件内部数据的组织格式和文件用途有关。
  - 二进制文件与文本文件的主要区别在于是否有统一的字符编码。
  - 二进制是信息按照非字符但特定格式形成的文件，数字图像的JPEG和PNG，数字音频的MP3和WAV等。
  - 由于二进制文件没有统一的字符编码，故只能当做字节流，而不能看做是字符串。

文件概述与读写

- 文本文件与二进制文件的区别
  - 二进制文件是变长的，相对于文本文件更加节省空间
  - 读取与存储区别

```
In [9]: text_file = open('人工智能与Python程序设计.txt', 'rt')

In [10]: print(text_file.readline())
人工智能与Python程序设计
```

文本文件读取

```
In [11]: binary_file = open('人工智能与Python程序设计.txt', 'rb')

In [12]: print(binary_file.readline())
b'\xe4\xba\xba\xe5\xb7\xa5\xe6\x99\xba\xe8\x83\xbd\xe4\xb8\x8ePython\xe7\xa8\x8b\xe5\xba\x8f\xe8\xae\xbe\xe8\xae\xa1'
```

二进制文件读取

文件概述与读写

- 编码
  - 编码是信息从一种形式转换为另一种形式的过程
  - ASCII编码、Unicode编码、UTF-8编码等
  - Unicode编码：
    - 跨语言、跨平台进行文本转换和处理
    - 对每种语言中字符设定统一且唯一的二进制编码
    - 每个字符两个字节长
    - 65536 个字符的编码空间
  - UTF-8编码
    - 可变长度的Unicode的实现方式

https://tool.chinaz.com/tools/unicode.aspx

文件概述与读写

- 文件操作
  - 文件读写是最常见的IO操作，遵循打开-操作-关闭步骤。
  - Python内置了读写文件的函数，用法和C是兼容的。
  - 在磁盘上读写文件的功能都是由操作系统提供的，现代操作系统不允许普通的程序直接操作磁盘
  - 读写文件即请求操作系统打开一个文件对象，进而对此文件对象进行操作
- 文件的打开
  - 利用Python内置的open()函数，以读文件的模式打开一个文件对象
  -
```
<variable> = open(<name>, <mode>)
```
  - <name>：文件路径及名称
  - <mode>：打开模式
  - <variable>：文件对象名

文件概述与读写

- 文件的打开

```
file = open('人工智能与Python程序设计.txt', 'rt')
```

  - 标识符 r 代表读取（只读），t 代表以文本文件读取方式进行解码（默认UTF-8编码）

文件打开模式	含义
r	只读。如果文件不存在，则返回异常
w	覆盖写。文件不存在则自动创建，存在则覆盖原文件
x	创建写。文件不存在则自动创建；存在则返回异常
a	追加写。文件不存在则自动创建；存在则在原文件最后追加写
b	二进制文件模式
t	文本文件模式（默认）
+	与r/w/x/a一同使用，在原功能基础上增加同时读写功能

文件概述与读写

- 文件的读取
  - 文件打开后，可根据打开方式对文件进行读写操作。
  - 文件读取操作：

操作方法	含义
<variable>.readall()	读入整个文件内容，返回字符串或字节流
<variable>. read(size=-1)	根据给定的size参数，读取前size长度的字符串或字节流
<variable>. readline(size=-1)	根据给定的size参数，读取该行前size长度的字符串或字节流
<variable>. readlines(hint=-1)	给定参数，从文件中读入前hint行，并以每行为元素形成一个列表

文件概述与读写

- 文件的读取

逐行读取

```
file = open('人工智能与Python程序设计.txt', 'rt')
for i in range(3):
    text = file.readline()
    print(text)
file.close()
```

```
人工智能与Python程序设计 第一课

人工智能与Python程序设计 第二课

人工智能与Python程序设计 第三课
```

```
file = open('人工智能与Python程序设计.txt', 'rt')
for i in range(3):
    text = file.readline()
    print(text[:-1])
file.close()
```

```
人工智能与Python程序设计 第一课
人工智能与Python程序设计 第二课
人工智能与Python程序设计 第三课
```

换行符

按给定长度读取

```
file = open('人工智能与Python程序设计.txt', 'rt')
for i in range(3):
    text = file.readline(5)
    print(text)
file.close()
```

```
人工智能与
Pytho
n程序设计
```

```
file = open('人工智能与Python程序设计.txt', 'rt')
for i in range(3):
    text = file.readline(5)
    print(text)
    text = file.readline()
    print(text)
file.close()
```

```
人工智能与
Python程序设计 第一课

人工智能与
Python程序设计 第二课

人工智能与
Python程序设计 第三课
```

文件概述与读写

- 文件的写入
  - 从计算机内存向文件写入数据
  - Python提供3个与文件内容写入有关的方法

操作方法	含义
<variable>.write()	向文件写入一个字符串或字节流
<variable>.writelines()	将一个元素全为字符串的列表写入文件
<variable>.seek(offset)	改变当前文件操作指针的位置，offset的值： 0--文件开头；1--当前位置； 2--文件结尾

文件概述与读写

- 文件的写入

```
file_write = open('人工智能与Python程序设计_write.txt', 'w')
file_write.writelines(['人工智能',' ','Python程序设计', '\n', '人工智能与Python程序设计'])
file_write.close()
file_read = open('人工智能与Python程序设计_write.txt', 'r')
print(file_read.read())
file_read.close()
```

```
人工智能 Python程序设计
人工智能与Python程序设计
```

文件概述与读写

- 文件读写的注意事项
  - 文件读取：调用read()会一次性读取文件的全部内容，如果文件有10G，内存就溢出。为保险起见，可以反复调用read(size)方法，每次最多读取size个字节的内容。
  - 文件关闭：只有调用close()方法时，操作系统才保证把没有写入的数据全部写入磁盘。忘记调用close()的后果是数据可能只写了一部分到磁盘，剩下的丢失了。

```
with open('人工智能与Python程序设计.txt', 'w') as f:
    f.write('Hello, world!')
with open('人工智能与Python程序设计.txt', 'r') as f:
    print(f.read())
```

```
Hello, world!
```

大数据分析导论
IO编程

提纲

- IO编程
- **多维数据的格式化与处理**
- 类和对象
- 两个高级程序

多维数据的格式化与处理

- 数据组织的维度
  - 一维数据：由对等关系的有序或无序数据构成，采用线性方式组织，对对应于数学中的数组和集合等概念。

北京、上海、广州、深圳、成都、重庆、杭州、武汉、西安、天津、苏州、南京、郑州、长沙、东莞、沈阳、青岛、合肥、佛山

哲学院、文学院、历史学院、国学院、新闻学院、经济学院、统计学院、法学院、信息学院、环境学院、理学院、数学学院、高瓴人工智能学院

多维数据的格式化与处理

- 数据组织的维度
  - 二维数据：也称表格数据，由关联关系数据构成，采用表格方式组织，对应于数学中的矩阵，常见的表格都属于二维数据。

		Polynomial			Exponential		
Benchmark	Error rate	Computation Required (Gflops)	Environmental Cost (CO_2)	Economic Cost ($)	Computation Required (Gflops)	Environmental Cost (CO_2)	Economic Cost ($)
ImageNet	Today: 11.5%	10^{14}	10^{6}	10^{6}	10^{14}	10^{6}	10^{6}
	Target 1: 5%	10^{19}	10^{10}	10^{11}	10^{27}	10^{19}	10^{19}
	Target 2: 1%	10^{28}	10^{20}	10^{20}	10^{120}	10^{112}	10^{112}
MS COCO	Today: 46.7%	10^{14}	10^{6}	10^{6}	10^{15}	10^{7}	10^{7}
	Target 1: 30%	10^{23}	10^{14}	10^{15}	10^{29}	10^{21}	10^{21}
	Target 2: 10%	10^{44}	10^{36}	10^{36}	10^{107}	10^{99}	10^{99}
SQuAD 1.1	Today: 4.621%	10^{13}	10^{4}	10^{5}	10^{13}	10^{5}	10^{5}
	Target 1: 2%	10^{15}	10^{7}	10^{7}	10^{23}	10^{15}	10^{15}
	Target 2: 1%	10^{18}	10^{10}	10^{10}	10^{40}	10^{32}	10^{32}
CoLLN 2003	Today: 6.5%	10^{13}	10^{5}	10^{5}	10^{13}	10^{5}	10^{5}
	Target 1: 2%	10^{43}	10^{35}	10^{35}	10^{82}	10^{73}	10^{74}
	Target 2: 1%	10^{61}	10^{53}	10^{53}	10^{181}	10^{173}	10^{173}
WMT 2014 (EN-FR)	Today: 54.4%	10^{12}	10^{4}	10^{4}	10^{12}	10^{4}	10^{4}
	Target 1: 30%	10^{23}	10^{15}	10^{15}	10^{30}	10^{22}	10^{22}
	Target 2: 10%	10^{43}	10^{35}	10^{35}	10^{107}	10^{99}	10^{100}

多维数据的格式化与处理

- 数据组织的维度
  - 高维数据：由键值对类型的数据构成，采用对象方式组织，属于整合度更好的数据组织方式。高维数据在网络系统中十分常用，HTML、XML、JSON等都是高维数据组织的语法结构。

```
<nav style="background-color: #dcdcdc;">
    <span class="container">
        <span class="navbar-group">
            <ul class="navbar-list">
                <li class="nav-item"><a href="/Home">Home</a></li>
                <li class="nav-item"><a href="/Products">Products</a></li>
                <li class="nav-item"><a href="/Services">Services</a></li>
            </ul>
        </span>
        <span class="navbar-group">
            <ul class="navbar-list">
                <li class="nav-item"><a href="/About">About Us</a></li>
                <li class="nav-item"><a href="/Privacy">Privacy</a></li>
            </ul>
        </span>
    </span>
</nav>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<Print_Records>
<form1>
    <Name>Ego ille</Name>
    <Address>345 Park Aven</Address>
    <City>San Jose</City>
    <State>CA</State>
    <ZipCode>94087</ZipCode>
    <Country>USA</Country>
</form1>
<form1>
    <Name>Johnson</Name>
    <Address>1 Almaden Blvd</Address>
    <City>San Jose</City>
    <State>CA</State>
    <ZipCode>94089</ZipCode>
    <Country>USA</Country>
</form1>
</Print_Records>
```

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [],
  "spouse": null
}
```

多维数据的格式化与处理

- 一、二维数据的存储格式
  - 一维数据是最简单的数据组织类型，有多种存储格式，常用特殊字符分割，分隔方式如下：
    - 用一个或多个空格分隔，例如：北京 上海 广州 深圳
    - 用逗号分割，例如：北京，上海，广州，深圳
    - 用其他符号或符合组合分隔，例如：北京！上海！广州！深圳
  - 逗号分隔数值的存储格式叫做CSV格式（comma-Separated Values)
    - CSV是一种通用的、相对简单的文件格式，在商业和科学上广泛应用，尤其应用在程序之间转移表格数据。

多维数据的格式化与处理

- CSV格式遵循如下基本规则
  - 纯文本格式，通过单一编码表示字符。
  - 以行为单位，开头不留空行，行之间没有空行。
  - 每行表示一个一维数据，多行表示二维数据。
  - 以逗号（英文，半角）分隔每列数据，列数据为空也要保留逗号。
  - 对于表格数据，可以包含或不包含列名，包含时列名放置在文件第一行。

多维数据的格式化与处理

- CSV示例

Error rate	Gflops	Env Cost(CO2)	Economic Cost($)
11.50%	10^14	10^6	10^6
5%	10^19	10^10	10^11
1%	10^28	10^20	10^20

ImageNet相关指标预测[1]

```
1  Error rate,Gflops,Env Cost(CO2),Economic Cost($)
2  11.50%,10^14,10^6,10^6
3  5%,10^19,10^10,10^11
4  1%,10^28,10^20,10^20
```

CSV存储示例

CSV格式存储的文件一般采用.CSV为扩展名，可以通过windows平台上的记事本或excel工具打开，也可以在其他操作系统平台上用文本编辑工具打开，如sublime。

[1] https://www.stateof.ai/

多维数据的格式化与处理

- 一、二维数据的表示与读取
  - CSV文件的每一行是一维数据，可以使用Python中的列表类型表示，整个CSV文件是一个二维数据，由表示每一行的列表类型作为元素，组成一个二维列表。
  - CSV文件的读取

Error rate	Gflops	Env Cost(CO2)	Economic Cost($)
11.50%	10^14	10^6	10^6
5%	10^19	10^10	10^11
1%	10^28	10^20	10^20

```
ls = []
with open('imagenet.csv', 'r') as f:
    for line in f:
        line=line.replace("\n", "")
        ls.append(line.split(","))
print(ls)
```

```
[['Error rate', 'Gflops', 'Env Cost(CO2)', 'Economic Cost($)'], ['11.5
0%', '10^14', '10^6', '10^6'], ['5%', '10^19', '10^10', '10^11'], ['1%',
'10^28', '10^20', '10^20']]
```

注意：以split(“,”)方法从CSV文件中获得内容时，每行最后一个元素后面包含一个换行符（”\n”）。对于数据的表达和使用而言，该换行符是多余的，可以通过使用字符串的replace()方法将其去掉。

多维数据的格式化与处理

- 一、二维数据的表示与读取
  - CSV文件的读取
    - Python内置了文件的处理模块，可方便调用

```
ls = []
with open('imagenet.csv', 'r') as f:
    for line in f:
        line=line.replace("\n", "")
        ls.append(line.split(","))
print(ls)
```

```
[['Error rate', 'Gflops', 'Env Cost(CO2)', 'Economic Cost($)'], ['11.5
0%', '10^14', '10^6', '10^6'], ['5%', '10^19', '10^10', '10^11'], ['1%',
'10^28', '10^20', '10^20']]
```

```
import csv
with open('imagenet.csv', 'r', newline='') as f:
    f_csv = csv.reader(f)
    for row in f_csv:
        print(row)
```

```
['Error rate', 'Gflops', 'Env Cost(CO2)', 'Economic Cost($)']
['11.50%', '10^14', '10^6', '10^6']
['5%', '10^19', '10^10', '10^11']
['1%', '10^28', '10^20', '10^20']
```

一、二维数据的格式化与处理

- 一、二维数据的表示与读取
  - CSV文件的读取
    - 逐行处理

```
import csv
with open('imagenet.csv', 'r', newline='') as f:
    f_csv = csv.reader(f)
    for row in f_csv:
        items = ""
        for item in row:
            items += "{}\t".format(item)
        print(items)
```

```
Error rate      Gflops  Env Cost(CO2)   Economic Cost($)
11.50%  10^14   10^6    10^6
5%      10^19   10^10   10^11
1%      10^28   10^20   10^20
```

多维数据的格式化与处理

- 一、二维数据的表示与读取
  - CSV文件的写入
    - 基于列表类型

```
import csv

headers = ['Error rate', 'Gflops', 'Env Cost(CO2)', 'Economic Cost($)']
rows = [['11.50%', '10^14', '10^6', '10^6'],
        ['5%', '10^19', '10^10', '10^11'],
        ['1%', '10^28', '10^20', '10^20']]

with open('imagenet.csv','w')as f:
    f_csv = csv.writer(f)
    f_csv.writerow(headers)
    f_csv.writerows(rows)
```

	A	B	C	D
1	Error rate	Gflops	Env Cost(CO	Economic Cost($)
2	11.50%	10^14	10^6	10^6
3	5%	10^19	10^10	10^11
4	1%	10^28	10^20	10^20
5				

多维数据的格式化与处理

- 一、二维数据的表示与读取
  - CSV文件的写入
    - 基于字典类型

class csv.**DictWriter**(*f*, *fieldnames*, *restval=''*, *extrasaction='raise'*, *dialect='excel'*, **args*, ***kwds*)

Create an object which operates like a regular writer but maps dictionaries onto output rows. The *fieldnames* parameter is a `sequence` of keys that identify the order in which values in the dictionary passed to the `writerow()` method are written to file *f*. The optional *restval* parameter specifies the value to be written if the dictionary is missing a key in *fieldnames*. If the dictionary passed to the `writerow()` method contains a key not found in *fieldnames*, the optional *extrasaction* parameter indicates what action to take. If it is set to `'raise'`, the default value, a `ValueError` is raised. If it is set to `'ignore'`, extra values in the dictionary are ignored. Any other optional or keyword arguments are passed to the underlying `writer` instance.

```
import csv

headers = ['Error rate', 'Gflops', 'Env Cost(CO2)', 'Economic Cost($)']
rows = [
        {'Error rate':'11.50%', 'Gflops':'10^14', 'Env Cost(CO2)':'10^6', 'Economic Cost($)':'10^6'},
        {'Error rate':'5%', 'Gflops':'10^19', 'Env Cost(CO2)':'10^10', 'Economic Cost($)':'10^11'},
        {'Error rate':'1%', 'Gflops':'10^28', 'Env Cost(CO2)':'10^20', 'Economic Cost($)':'10^20'}
    ]

with open('imagenet.csv','w',newline='')as f:
    f_csv = csv.DictWriter(f,headers)
    f_csv.writeheader()
    f_csv.writerows(rows)
```

大数据分析导论
IO编程

提纲

- IO编程
- 多维数据的格式化与处理
- **类和对象**
- 两个高级程序

Python 面向对象编程

- **面向过程（Procedure Oriented Programming，POP）**的程序设计把计算机程序视为一系列的命令集合，即一组函数的顺序执行。为了简化程序设计，面向过程把函数继续切分为子函数，即把大块函数通过切割成小块函数来降低系统的复杂度。
- **面向对象编程（Object Oriented Programming， OOP）**是一种程序设计思想。OOP把对象作为程序的基本单元，一个对象包含了数据和操作数据的函数
- 面向对象的程序设计把计算机程序视为一组对象的集合，而每个对象都可以接收其他对象发过来的消息，并处理这些消息，计算机程序的执行就是一系列消息在各个对象之间传递。

Python 面向对象编程

在Python中，所有数据类型都可以视为对象，也可以自定义对象。自定义的对象数据类型就是面向对象编程中的类（class）的概念。

Python 面向对象编程

- 以＂处理学生成绩＂举例，我们来说明POP 与OOP 在程序流程上的不同之处：

假如有几个学生和他们的考试成绩，在POP中可用一个dict进行表示：

```
std1 = {'name': 'Michael', 'score': 98 }
std2 = {'name': 'Bob', 'score': 81 }
std3 = {'name': 'Kristen', 'score': 93 }
```

如果想处理学生成绩，可通过函数实现，如打印学生的成绩：

```
def print_score(std):
    print('%s: %d' % (std['name'], std['score']))

print_score(std1)
```

```
Michael: 98
```

Python 面向对象编程

- 以＂处理学生成绩＂举例，我们来说明POP 与OOP 在程序流程上的不同之处：

若采用OOP，首先考虑的不是程序的执行流程，而是观察这些学生的共性特点，定义一个Student 类型，其实例（即Student 对象）拥有name和score这两个共有属性（Property）。

```
std1 = {'name': 'Michael', 'score': 98 }
std2 = {'name': 'Bob', 'score': 81 }
std3 = {'name': 'Kristen', 'score': 93 }
```

Python 面向对象编程

若要打印一个学生的成绩，先创建出这个学生对应的对象，然后给对象发送一个打印（print_score）消息，让对象把自己的数据打印出来。

```
class Student(object):

    def __init__(self, name, score):
        # 类的构造函数，  self代表类的实例
        self.name  = name
        self.score = score

    def print_score(self):
        # 类的方法
        print('%s: %d' % (self.name, self.score))
```

Python 面向对象编程

给对象发消息实际上就是调用对象对应的关联函数，我们称之为对象的方法（Method）：

```
Michael = Student('Michael', 98) # 创建实例对象
Kristen = Student('Kristen', 93) # 创建实例对象

Michael.print_score() # 调用类的方法
Kristen.print_score() # 调用类的方法
```

```
Michael: 98
Kristen: 93
```

类和实例

- OOP 的设计思想来源于自然界，因为在自然界中，类(class)和实例(instance)的概念非常自然。
  - 类(class): 用来描述具有相同的属性和方法的对象的集合。比如我们定义的Class--Student，是指学生这个概念。
  - 实例(instance)：创建一个类的实例，类的具体对象。比如一个个具体的Student，Michael和Kristen是两个具体的student。

类和实例

- 类和实例是OOP中最为重要的概念
  - 类是抽象的模板，比如Student 类，
  - 实例是根据类创建出来的一个个具体的“对象”，每个对象都拥有相同的方法，但各自的数据可能不同。

```
Michael = Student('Michael', 98) # 创建实例对象
Kristen = Student('Kristen', 93) # 创建实例对象

Michael.print_score() # 调用类的方法
Kristen.print_score() # 调用类的方法
```

```
Michael: 98
Kristen: 93
```

类和实例

- 仍以“处理学生成绩”举例，我们来说明类和实例的概念
  - Python中，定义类是通过class关键字
  - class 后为类名，类名通常是大写开头的单词
  - 类名（object），表示该类从object父类继承下来。通常，如果没有合适的继承类，就使用object类，这是所有类最终都会继承的类。

```
class Student2(object):
    pass
```

类和实例

- 仍以“处理学生成绩”举例，我们来说明类和实例的概念
  - 定义好了Student2 类，就可以根据它来创建出其实例

```
Michael = Student2()
Kristen = Student2()
```

```
print(Michael)
print(Kristen)
```

```
<__main__.Student2 object at 0x7fad4b7d5ca0>
<__main__.Student2 object at 0x7fad4b7d5730>
```

```
print(Student2)
```

```
<class '__main__.Student2'>
```

类和实例

- 仍以“处理学生成绩”举例，我们来说明类和实例的概念
  - 定义好了Student2 类，就可以根据它来创建出其实例

```
Michael = Student2()
Kristen = Student2()
```

```
print(Michael)
print(Kristen)
```

```
<__main__.Student2 object at 0x7fad4b7d5ca0>
<__main__.Student2 object at 0x7fad4b7d5730>
```

```
print(Student2)
```

```
<class '__main__.Student2'>
```

Student2 本身是一个类，而Michael指向的是一个Student2 的实例，0x7fef7387a160是其内存地址，每个object 的地址都不一样。而Student2本身则是一个类

类和实例

- 仍以“处理学生成绩”举例，我们来说明类和实例的概念
  - Student2类实例化后，可以自由地给一个实例变量绑定属性

```
Michael.name="Michael Simon"
print(Michael.name)
```

```
Michael Simon
```

```
Kristen.name
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-24-41b55b5ff35e> in <module>
----> 1 Kristen.name

AttributeError: 'Student' object has no attribute 'name'
```

类和实例

- 类起到模板的作用，可以在创建实例的时候，把一些我们认为必须绑定的属性强制填写进去。
- 通过定义一个特殊的__init__方法（构造函数），在创建实例的时候，就把name，score等属性绑上去
  - __init__前后分别有两根下划线
  - __init__()的第一个参数永远是self，表示创建的实例本身。因此，在__init__()内部，就可把各种属性绑定到self，因为self就指向创建的实例本身。

```
class Student(object):

    def __init__(self, name, score):
        # 类的构造函数，  self代表类的实例
        self.name  = name
        self.score = score
```

类和实例

- 有了__init__()，在创建实例时，就不能传入空的参数了，必须传入与__init__()相匹配的参数，但self不需要传，Python 解释器自己会把实例变量传进去。

```
Michael = Student('Michael', 98) # 创建实例对象
print(Michael.name)
print(Michael.score)
```

```
Michael
98
```

和普通的函数相比，在类中定义的函数只有一点不同，就是第一个参数永远是实例变量self，并且调用时，不用传递该参数。除此之外，类的方法和普通函数没有显著区别，所以，仍然可以用默认参数、可变参数、关键字参数和命名关键字参数。

数据封装

- OOP 的一个重要特点就是数据封装。
  - Student类中，每个实例拥有各自的name和score数据
  - 可以通过函数来访问实例的数据，如打印某个学生的成绩
- Student 的实例本身就拥有这些数据，故要访问它们，可以直接在Student类的内部定义访问数据的函数，进而把“数据”封装。
- 封装数据的函数是和Student类本身是关联起来的，我们称之为类的方法

```
class Student(object):

    def __init__(self, name, score):
        # 类的构造函数，  self代表类的实例
        self.name  = name
        self.score = score

    def print_score(self):
        # 类的方法
        print('%s: %d' % (self.name, self.score))
```

我们发现，要定义一个方法，除了第一个参数是self外，其他和普通函数一样。

数据封装

- 要调用一个方法，只需要在实例变量上直接调用即可。除了self不用传递，其他参数正常传入。
  - 从调用方来看Student 类，只需在创建实例时给定name和score
  - score打印会在Student 类的内部实现。这些数据和逻辑被『封装』起来，调用会变得容易。

```
Michael = Student('Michael Simon', 98)
Michael.print_score()
```

```
Michael Simon: 98
```

数据封装

- 通过封装，可以给class增加新的方法。

```
class Student(object):

    def __init__(self, name, score):
        # 类的构造函数， self代表类的实例
        self.name  = name
        self.score = score

    def print_score(self):
        # 类的方法
        print('%s: %d' % (self.name, self.score))

    def get_grade(self):
        if self.score >= 90:
            return 'A'
        elif self.score >= 60:
            return 'B'
        else:
            return 'C'
```

大数据分析导论
IO编程

提纲

- IO编程
- 多维数据的格式化与处理
- 类和对象
- **两个高级程序**

拼写检查

- http://norvig.com/spell-correct.html
- **How to Write a Spelling Corrector**
  - One week in 2007, two friends (Dean and Bill) independently told me they were amazed at Google's spelling correction. Type in a search like [speling] and Google instantly comes back with **Showing results for: *spelling***. I thought Dean and Bill, being highly accomplished engineers and mathematicians, would have good intuitions about how this process works. But they didn't, and come to think of it, why should they know about something so far outisde their specialty?I figured they, and others, could benefit from an explanation. The full details of an industrial-strength spell corrector are quite complex (you can read a little about it here or here). But I figured that in the course of a transcontinental plane ride I could write and explain a toy spelling corrector that achieves 80 or 90% accuracy at a processing speed of at least 10 words per second in about half a page of code.

Language	Lines Code	Author (and link to implementation)
Awk	15	Tiago "PacMan" Peczenyj
Awk	28	Gregory Grefenstette
C	184	Marcelo Toledo
C++	98	Felipe Farinon
C#	43	Lorenzo Stoakes
C#	69	Frederic Torres
Clojure	18	Rich Hickey
Coffeescript	21	Daniel Ribeiro
D	23	Leonardo M
Erlang	87	Federico Feroldi
F#	16	Dejan Jelovic
F#	34	Sebastian G
Go	57	Yi Wang
Groovy	22	Rael Cunha
Haskell	24	Grzegorz
Java 8	23	Peter Kuhar
Java	35	Rael Cunha
Javascript	53	Panagiotis Astithas
Lisp	26	Mikael Jansson
OCaml	148	Stefano Pacifico
Perl	63	riffraff
PHP	68	Felipe Ribeiro
R	2	Rasmus B邋th
Rebol	133	Cyphre
Ruby	34	Brian Adkins
Scala	20	Pathikrit Bhowmick
Scheme	45	Shiro
Swift	108	Airspeed Velocity

考试成绩换算

- 一次期末考试成绩

学生	基本分 x_1	附加分 x_2	调整后成绩 y
001	90	10	100
002	80	5	92
003	85	0	92
004	78	10	93
005	75	5	89
006	66	15	89
007	52	5	75
张三	83	10	?

$y = f(x_1, x_2)$

f是什么？

$$y = 10 \times (\sqrt{x_1} + \frac{x_2}{20})$$

作业

- 输入：多个文本文件（已分词）
- 输出：统计出文档中所有的词的词频
  - （1）按照词频由高到低排序并以CSV格式输出到一个文件中；
  - （2：选做）参考Python书中第7章第5节，将此CSV文件转换为HTML文件并用浏览器展示。

- 要求：
  - 提交代码、结果、代码实现和运行结果文档说明
  - 鼓励OOP编程，用类封装实现相关功能

谢谢！