大数据分析基础

Python基础

- 关于计算机编程
- 初识Python
- Python基础语法
- 正则表达式

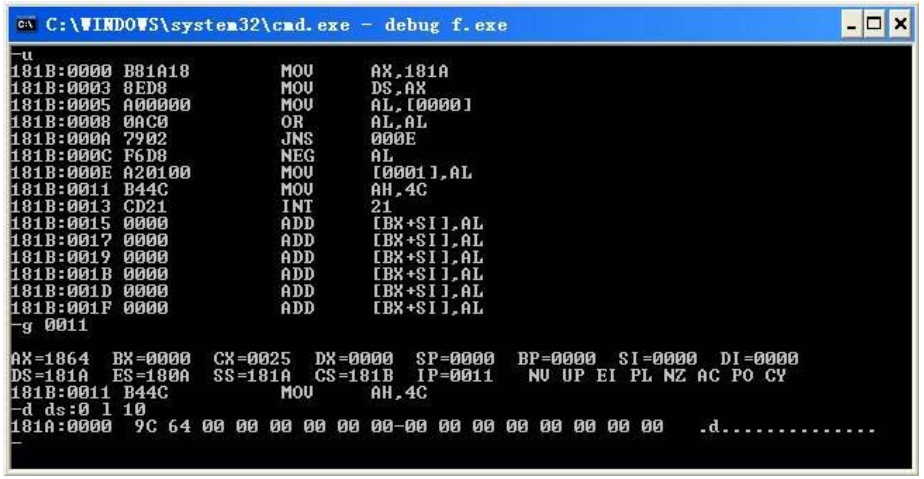
关于计算机编程

- 我们为什么要学习计算机编程？
  - 处理数据而言，Excel已经足够强大
  - 确实如此吗？
    - 你甚至可能都无法打开文件

- 计算机编程难学吗？
  - 大家眼中的计算机编程：
  - 实际上的计算机编程：
    - print("hello world")


```
word_count = 711
if word_count > 800:
    print("字数达到要求")
else:
    print("字数不达标")
```

- 计算机语言即用于人与计算机之间通讯的语言
  - 所谓计算机编程就是用计算机语言与计算机进行沟通
- 计算机语言的分类
  - 机器语言（第一代计算机语言）
    - 指一台计算机全部的指令集合
    - 一串串由"0"和"1"组成的指令序列
    - 每一台计算机的指令集合都不尽相同
  - 汇编语言（第二代计算机语言）
    - 用一些简洁的英文字母、符号串来替代一个特定的指令的二进制串
    - 比如，用"ADD"代表加法，"MOV"代表数据传递等等
    - 依赖于机器硬件，移植性不好，但效率仍非常高，因此很多软件仍然使用汇编语言编写
  - 高级语言（第三代计算机语言）



- 高级语言
  - 高级语言主要是相对于汇编语言而言，它并不是特指某一种具体的语言
  - C，C++，Java，Python等都是高级语言
  - 计算机不能直接的理解高级语言，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
  - 根据翻译过程的不同，高级语言又可以分成编译性语言和解释性语言
    - 编译性语言需要先经过编译成为机器可直接执行指令（比如.exe、.dll、.ocx文件）才能运行，编译后的文件不可以跨平台运行
    - 解释性语言则是有一个专门的解释器，一边运行一边解释，一般都可以跨平台运行
    - 编译性语言有C、C++等，解释性语言有JavaScript、Python等
    - 部分语言如Java兼具解释性和编译性

高级语言

	解释性语言	编译性语言
优点	可移植性好，只要有解释环境，可以在不同的操作系统上运行。将解释器移植到不同的系统上，程序不用改动就可以在移植了解释器系统上运行。	运行速度快，代码效率高，编译后程序不可以修改，保密性好。
缺点	运行需要解释环境，运行起来比编译的要慢，占用的资源也要多一些，代码效率低，代码修改后就可以运行，不需要编译过程。因为不仅要给用户程序分配空间，解释器本身也占用了宝贵的系统资源。其封装底层代码，程序严重依赖平台。不能同C++，VB那样直接操作底层。	代码需要经过编译方可运行，可移植性差，只能在兼容的操作系统上运行。

- 编程的思维：面向过程、面向对象
  - 面向过程
    - 一种以过程为中心的编程思想
    - 是一种基础的顺序的思维方式
    - C，Pascal等
  - 面向对象
    - 面向对象是按人们认识客观世界的系统思维方式，采用基于对象（实体）的概念建立模型，模拟客观世界分析、设计、实现软件的办法
    - C++，C#，Java，Objective-C，Swift，Python等

- 面向对象编程（Object oriented programming）
  - 对象（Object）
    - 对象是人们要进行研究的任何事物
    - 对象具有唯一性
  - 类（Class）
    - 具有相同特性（数据元素）和行为（功能）的对象的抽象就是类
    - 类具有抽象性
    - 从类到对象的过程就叫做**实例化**
  - 方法（Method）
    - 类中操作的实现过程叫做方法
    - 一个方法有方法名、返回值、参数、方法体

面向对象编程

封装

- 把一个或多个元素封闭在一个物理的或者逻辑的包中
- 利用访问修饰符进行权限控制

Java关键词	范围	Python
public	所有对象都可以访问	默认设置
private	对象本身在对象内部可以访问	变量名前加__
protected	只有该类对象及其子类对象可以访问	无
default	同一个程序集的对象可以访问	无

注意：
Python中实际上不存在protected的概念，变量名前加一条下划线_的实际效果和不加一样，是一个约定俗成的惯例，表示“虽然我可以被访问，但是，请把我视为私有变量，不要随意访问”

面向对象编程

继承

- 在现有类（基类、父类）上建立新类（派生类、子类）的处理过程称为继承
- 继承是软件复用的一种形式
- 使用继承可以复用现有类的数据和行为，为其赋予新功能而创建出新类
- 派生类能自动获得基类的除了构造函数和析构函数以外的所有成员，可以在派生类中添加新的属性和方法扩展其功能
- 单重继承（一个类只能派生自一个基类）和多重继承（一个类可以派生自多个类）

面向对象编程

多态

- 是指程序设计中存在同名不同方法的存在
- 主要通过子类对父类的覆盖来实现多态
- 设计原则之一就是要依赖于抽象，而不依赖于具体，增加灵活性

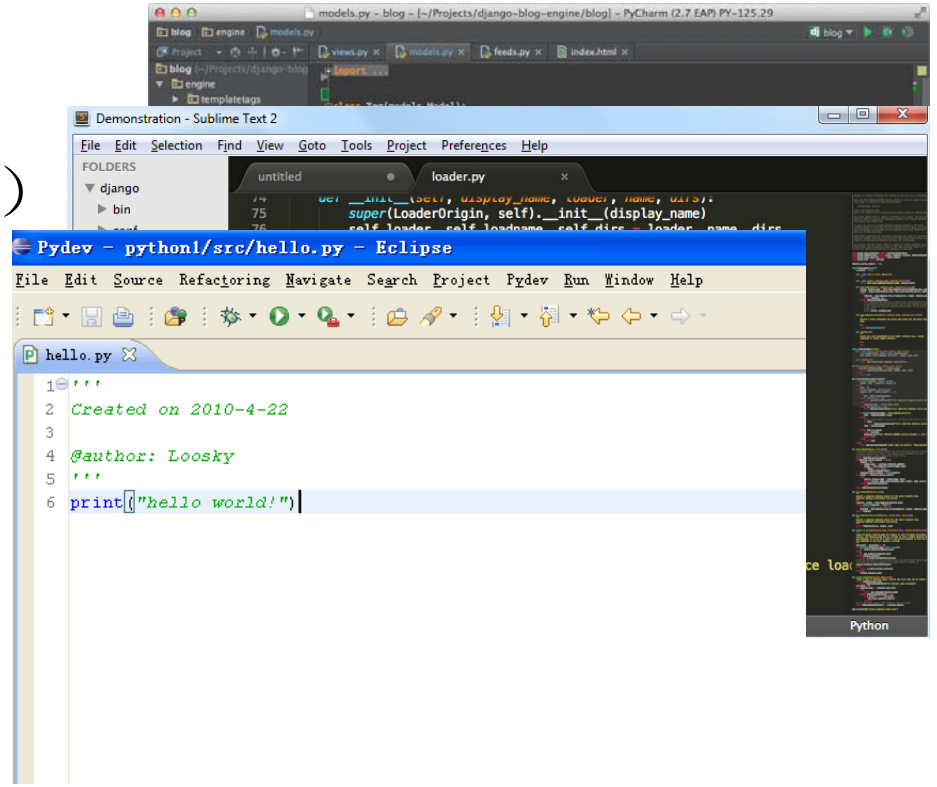
面向对象编程

	面向过程	面向对象
特点	模块化、流程化	抽象、封装、继承、多态
优点	性能比面向对象高。因为类调用时需要实例化，比较消耗资源；比如单片机、嵌入式开发、Linux/Unix等一般采用面向过程开 发。	易维护、易复用、易扩展，由于面向对象有封装、继承、多态性的特性，可以设计出低耦合的系统，使系统更加灵活、更加易于维护
缺点	没有面向对象易维护、易复用、易扩展	性能比面向过程低

- 为什么选择Python？
  - 语法简洁，接近于英语的自然语言
  - 学习曲线平坦
  - 丰富的第三方库，特别是在大数据分析处理和人工智能方面
  - 可扩展性和可嵌入性：可以将Python代码嵌入到其它语言中，反之亦可
  - 可移植性：Unix/Linux/Mac OS等一般自带Python，在移动平台稍弱
  - 代码规范性极强，易于阅读
  - Python作为脚本语言，更适合开发小的应用
    - 脚本语言是为了缩短传统的编写-编译-链接-运行（edit-compile-link-run）过程而创建的计算机编程语言

- Python 2.x 还是Python 3.x？
  - Python 3无法兼容Python 2
- 两者语法上的区别
  - print 语句取消，统一改为函数
  - Python 3源码文件默认使用utf-8编码
  - 除法运算（a/b）：a、b均为整数时，Python 2返回整数，Python 3返回浮点数
  - 整型数统一为long
  - 不等号：取消了<>作为不等号，只保留!=
  - 其它的一些细节变化
- Python核心团队计划2020年停止支持Python 2，一些重要的第三方库也表示即将停止支持Python 2

- 下载安装Python
  - https://www.python.org/downloads/
- IDE（Integrated development environment）
  - Python自带IDE：IDLE
  - PyCharm、Sublime Text等
  - Eclipse：需要安装PyDev
  - Jupyter Notebook
    - 一个交互式笔记本
    - 本质是一个 Web 应用程序，支持40多种语言
- Anaconda
  - Anaconda是一个开源的Python发行版本，包含了conda、Python等180多个包
    - https://www.anaconda.com/distribution/



Python基础语法：常量与变量

- 常量
  - 一旦初始化后就不能修改的固定值
  - Python中主要的值有数值、布尔型、字符串
- 数值
  - 整型数、浮点数、复数
  - Python中默认的整数类型为long，可以任意长
- 布尔型
  - 取值为True或者False，用于逻辑运算
- 字符串
  - 一对单引号(‘)或者双引号(“)括起来的字符序列
  - 三引号（’’’或者”””）表示一个多行的字符串

- 转义字符
  - 假如有一个字符串包含单引号（’），如何表示这个字符串呢？
  - 在单引号前面加一个\，比如’That\’s great!’
  - 常见的特定转义字符
    - \t：一个制表符　　\n：ASC II换行符　　\r：ASC II回车符
    - \\：反斜杠\　　\’：单引号　　\”：双引号
    - 注意：如果字符串本身是用单引号括起来的，那么在字符串中添加双引号就不需要使用转义字符；反之亦然。
    - 如果想字符串不被特殊处理，例如像转义序列，那么就需要在字符串前面附加r 或R 来指定自然字符串，例如r’That\’s great!’

- 字符串拼接
  - 如果是多个字符串拼接，那么直接用+就可以，如’小明考了’+’100分’
    - 这种方式由于要重新申请内存，所以相对来说效率比较低
  - 如果是字符串和数值混合拼接
    - 方法1：将数值用str()方法转换成字符串后拼接，如’小明考了’+str(100)+’分’
    - 方法2：用format()方法，如’小明考了{0}分’.format(100)
      - 0表示format()中的参数位置，如果{}中不填写位置，则按照format中的参数排列顺序填入
      - 注意：{}的个数一定要和format中参数的个数一致

- 变量
  - 变量就是代表某个数据(值)的名称
  - 变量是计算机中存储信息的一部分内存
  - 变量可以有不同类型的值，称之为数据类型
    - 整型、浮点、字符串、布尔型等
    - Python中变量没有固定的数据类型
- 变量命名规则
  - 变量名的第一个字符必须是字母表中的字母（大写或小写）或者一个下划线(_)
  - 变量名的其他部分可以由字母（大写或小写）、下划线(_)或数字（0-9）组成。
  - 变量名是对大小写敏感的。例如， myname 和myName 不是一个标识符。
    - 有效变量名的例子：i 、__my_name 、name_23 和a1b2_c3 。
    - 无效变量名的例子：2things 、this is spaced out 和my-name 。

缩进

- Python中每行开头的空白都很重要，称之为缩进
- 行首的主要的空白（空格键和制表符）用来决定逻辑行缩进的层次，从而来决定语句分组
- 同一层次的语句必须有相同的缩进，每一组这样的语句称为一个块
- 错误的缩进会引发错误
- 注意：在同一个代码文件中，缩进所用的符号和符号的数量必须一致！

- 逻辑行和物理行
  - 物理行是你在编写程序时所看见的
  - 逻辑行是Python 看见的单个语句，如print(‘Hello World’)
  - 默认地，Python 希望每行都只使用一个语句，这样使得代码更加易读。
  - 如果想要在一个物理行中使用多于一个逻辑行，那么需要使用分号（;）来特别地标明这种用法
- 注释
  - 单行注释：#
  - 多行注释：三对单引号（’’’注释内容’’’）或者双引号（”””注释内容”””）
  - 为什么需要注释？
    - 与人方便于己方便：准确告诉代码的阅读者这段代码的目的、功能、细节等必要信息

算数操作符

符号	含义
+	加
-	减
*	乘
/	除
**	幂运算，如4**2 = 4^2
//	地板除（Floor），返回除法结果向下取整的整数，如4//3=1，(-4)//3=-2
%	模除，返回余数

比较操作符

符号	含义
>	大于
<	小于
>=	大于等于
<=	小于等于
==	等于
!=	不等于

字符串也可以用这些进行比较大小，比较时候按照ASCII码来比较

- 逻辑操作符

符号	含义
and	且，T and T = T，T and F = F，F and F = F
or	或，T or T = T，T or F = T，F or F = F
not	非，not T = F，not F = T

位操作符

位操作符针对的是二进制数

符号	含义
~	将二进制数取反，即0变1、1变0（包括符号位，“~x”的结果为“-(x+1)”）
<<	左移，a << b表示将整数a的二进制形式向左移动b位
>>	右移，a >> b表示将整数a的二进制形式向右移动b位
&	与，同为真则真，有一个为假，则为假
|	或，有一个为真，则为真，两个都是假，才是假
^	异或，同为0，异为1

（&，|）和（and，or）是两组比较相似的运算符

- 如果a, b是数值变量， 则&， |表示位运算， and，or则依据是否非0来决定输出
  - 位运算后结果为0则F，为1则T
- 如何a, b是逻辑变量， 则两类的用法基本一致

优先级

- 改变优先级
  - 用括号()
- 结合顺序
  - 同一优先级的运算符通常是从左往右结合
  - 例如，2+3+4 的顺序是(2+3)+4
  - 也有一些运算符是从右往左，如赋值
  - 思考：10==10==10的结果？
    - Python中a==b==c等同于(a==b)&(b==c)

Python基础语法：优先级

运算符	描述
lambda	Lambda表达式
or	布尔“或”
and	布尔“与”
not x	布尔“非”
in，not in	成员测试
is，is not	同一性测试
<，<=，>，>=，!=，==	比较
|	按位或
^	按位异或
&	按位与
<<，>>	移位
+，-	加法与减法
*，/，%	乘法、除法与取余
+x，-x	正负号
~x	按位翻转
**	指数
x.attribute	属性参考
x[index]	下标
x[index:index]	寻址段
f(arguments...)	函数调用
(experession,...)	绑定或元组显示
[expression,...]	列表显示
{key:datum,...}	字典显示
'expression,...'	字符串转换

- 主要有if，for，while三种
  - if是判断控制

if完整结构：

```
if 条件1:
    #满足条件1则执行语句块1
    语句块1
elif 条件2:
    #不满足条件1且满足条件2则执行语句块2
    #elif可以出现任意多次，也可以不出现
    语句块2
else:
    #条件1、2均不满足则执行语句块3
    #else最多只能出现一次，可以不出现
    语句块3
```

- 主要有if，for，while三种
  - for，while都是循环控制

while完整结构：

```
while 条件:
    #满足条件则执行语句块1
    语句块1
    #执行完语句块1后继续判断是否满足条件
else:
    #条件不满足则执行语句块2
    #else最多只能出现一次，可以不出现
    语句块2
    #执行完语句块2后退出循环
```

for完整结构：

```
for 一个序列:
    #依次调用序列中的值并执行语句块1
    语句块1
else:
    #序列中的值调用完毕则执行语句块2
    #else最多只能出现一次，可以不出现
    语句块2
    #执行完语句块2后退出循环
```

for i in range(开始值，结束值，步长)
- 开始值可省略，默认为0
- 步长可省略，默认为1

break和continue语句：

- break跳过语句块中剩下的语句并停止循环（不执行else语句块）
- continue跳过语句块中剩下的语句并执行下一个循环

- 函数是重用的程序段
- 它们允许你给一个语句块一个名称，然后你用这个名字可以在你的程序的任何地方，任意多次地运行这个语句块，这被称为**调用函数**
  - 函数用关键字def 来定义
  - def 关键字后跟一个函数的标识符名称，然后跟一对圆括号
  - 圆括号之中可以包括一些变量名（即传入的参数），该行以冒号结尾
  - 接下来是一块语句，它们是函数体
  - 可以用return语句定义函数调用后的返回值
  - return可以有多个
  - 程序执行完return后就立即跳出函数

```
def add(a,b):
    c = a + b
    return c

#调用函数add
print(add(3,4))
```

- 变量的作用域
  - 全局变量和局部变量
  - 当你在函数定义内声明变量的时候，它们与函数外具有相同名称的其他变量没有任何关系，即**变量名称对于函数来说是局部**的
  - 这称为变量的作用域
  - 变量的作用域是它们被定义的块，从被定义的那点开始

```
x = 50#全局变量x

def func(x):
    print('未定义局部变量前x={0}'.format(x))
    x = 2#定义了一个局部变量x
    print('x={0}'.format(x))
    x = 30
    print('更改局部变量x={0}'.format(x))


func(x)
print('全局x={0}'.format(x))
```

```
未定义局部变量前x=50
x=2
更改局部变量x=30
全局x=50
```

- 数据结构是用来存储一组相关数据的
  - 列表（list）
    - 一组有序元素的数据结构，即可以在一个列表中存储一个序列的元素
  - 元组（tuple）
    - 元组和列表十分类似，只不过元组是不可变的
  - 字典（dict）
    - 字典类似于你通过联系人名字查找地址和联系人详细情况的地址簿，即把键（名字）和值（详细情况）联系在一起
    - 注意，键必须是唯一的
  - 集合（set）
    - 集合是没有顺序的简单对象的聚集

- 列表
  - list1 = ['physics', 'chemistry', 1997, 2000]　　[]改成()就是元组
  - 列表中的元素的数据类型可以不同
  - 一旦创建了一个列表，可以添加、删除或是搜索列表中的元素
  - 搜索
    - list1[0]，列表位置为0的元素（即‘physics’），python是从0开始排序的
  - 添加
    - list1.append(新元素)，在列表末尾添加新元素
    - list1.insert(i，新元素)，在列表指定位置i添加新元素
  - 删除
    - list1.pop()，删除列表末尾的元素
    - list1.pop(i)，删除列表位置i的元素

- 字典
  - 键值对在字典中以这样的方式标记：d = key1 : value1, key2 : value2
  - dict = {'a': 1, 'b': 2, 'c': '3'}
  - 值可以取任何数据类型，如字符串，数字，列表，字典等，同一个字典中的类型可以不同
  - 键也可以取任何数据类型，除了列表和字典，同一个字典中的类型可以不同
  - 访问字典里面的值：dict['a']
  - 修改字典里面的值：dict['a']=5
  - 删除某个元素：del dict['a']

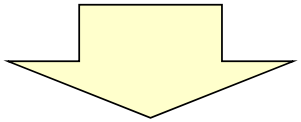
Python基础语法：输入输出

- 系统输入
  - something = input()
  - 系统得到所输入的字符串
- 系统输出
  - print()
- 文件读入
  - f = open('filename ')
- 文件输出
  - f = open('filename', 'w')
  - 读模式（’r’）、写模式（’w’）或追加模式（’a’）
  - 默认情况下，open() 用’r’ead 模式打开

Python基础语法：异常处理

- 当程序中出现某些异常的状况的时候，异常就发生了
  - 比如当你想要读某个文件的时候，而那个文件不存在
  - 如果不对异常进行处理，那么程序就会终止运行
- 处理异常
  - 使用try...expect语句来捕捉异常
  - 异常准确处理完后，程序继续运行

- 如何从文本中提取出需要的信息？
  - 从下面这一段话中提取数据
    - *初步核算，全年国内生产总值**900309**亿元，比上年增长**6.6%**。其中，第一产业增加值**64734**亿元，增长**3.5%**；第二产业增加值**366001**亿元，增长**5.8%**；第三产业增加值**469575**亿元，增长**7.6%**。第一产业增加值占国内生产总值的比重为**7.2%**，第二产业增加值比重为**40.7%**，第三产业增加值比重为**52.2%**。*



	增加值	增长	增加值占比
第一产业	64734	3.5%	7.2%
第二产业	366001	5.8%	40.7%
第三产业	469575	7.6%	52.2%

- 正则表达式（regex）通常用于在文本中查找匹配的字符串
  - Python中需要先导入模块re
  - 使用re.compile(pattern[, flags])来生成一个正则表达式
    - pattern : 一个字符串形式的正则表达式
    - flags : 可选，表示匹配模式，比如忽略大小写，多行模式等，具体参数为：
      - re.I 忽略大小写
      - re.L 表示特殊字符集 \w, \W, \b, \B, \s, \S 依赖于当前环境
      - re.M 多行模式
      - re.S 即为 . 并且包括换行符在内的任意字符（. 不包括换行符）
      - re.U 表示特殊字符集 \w, \W, \b, \B, \d, \D, \s, \S 依赖于 Unicode 字符属性数据库
      - re.X 为了增加可读性，忽略空格和 # 后面的注释

- 正则表达式（regex）通常用于在文本中查找匹配的字符串
  - 利用match()、search()来确定是否能够成功匹配
    - match(string)，search(string)
    - match()和search()的区别
      - match只匹配字符串的开始，如果字符串开始不符合正则表达式，则匹配失败，函数返回None
      - search匹配整个字符串，直到找到一个匹配
  - 然后findall(string[, pos[, endpos]])来找出字符串中的所有匹配项
    - string : 待匹配的字符串。
    - pos : 可选参数，指定字符串的起始位置，默认为 0。
    - endpos : 可选参数，指定字符串的结束位置，默认为字符串的长度。

常见正则表达式字符

字符

- .: 除换行符\n外的任意字符
- [abc]: a、b、c中的任意一个字符，如果以^开头（[^abc]）表示非a、b、c
- [0-9]: 任意一个数字，[1-5]匹配1-5的任意一个数字
- \d: 与[0-9相同]，\D 非数字（即[^\d]）
- \w: 单词字母[a-zA-Z0-9_]，\W非单词字母（即[^\W]）
- \s: 空白字符，[<空格>\t\r\n\f\v]，\S非空白字符
- \: 转义字符，如果字符串中需要匹配.，则\.

- 常见正则表达式字符
  - 数量词
    - *: 前面一个字符出现大于等于0次
    - +: 前面一个字符出现大于等于1次
    - ?: 前面一个字符出现0次或1次
    - {m}:前面一个字符出现m次
    - {m, n}:前面一个字符出现m-n次

- 常见正则表达式字符
  - 分组、逻辑
    - (…): 匹配结束后返回括号内匹配成功的字符串
    - abc|bcd: 匹配abc或者bcd
      - 作用域为整个表达式
      - 但是如果出现在(…)中，则作用范围仅限于(…)
  - 贪婪匹配
    - 即在长度不确定的情况下，尽可能多地匹配字符
    - 在abbbbbc中使用(b{1,3})就会返回bbb
    - 使用?将贪婪匹配转变为非贪婪匹配: (b{1,3}?)
  - For more: https://docs.python.org/3.7/library/re.html