Lecture2

凡是红色标注的语句，显然是重点。
凡是**紫色加粗标注的语句**，说明在附件中有对应的代码示例可供参考，建议实践并掌握。
凡是蓝色下划线标注的语句，说明给出了参考阅读链接，可依兴趣阅读。

1 .Python函数

1.1 函数基础

和绝大多数编程语言一样，Python的函数是预先组织好的，可重复使用的，用来实现某种功能的代码块。在实际操作中，我们可以使用Python提供的内建函数；也可以通过引用，使用各大模块替我们编写好的函数；也可以自己编写函数并调用。

关于函数，我们主要关注两点：参数和返回值。可以将函数理解成一个黑箱，参数和返回值分别对应着输入值和输出值。

$$y = FunctionName(x1,x2\cdots\cdots)$$

如上式，在函数运行时，我们需要告知其所需的参数（输入值）x1，x2（放在括号里），而FunctionName函数运行的返回值（输出值）将会被赋予给变量y。

1.2 内置函数

Python提供了一些预设的内置函数，方便完成一些基本的功能。预设函数的特点是不需要导入任何模块，直接使用函数名( )就可以调用，上一节中提到的type( )，len( )，用以转换变量类型的int( )，bool( )，float( )等均为Python的内置函数。

补充一些未曾提过的常见内置函数如下表：

函数名	功能
abs(x)	返回x的绝对值
pow(x,y)	返回x的y次方
open(name[,mode])	以mode形式打开名称为name的文件，返回file对象
max(x,), min(x,)	返回括号中的最大/最小值
a.reverse( )	无返回值，将列表a中元素反转
a.next( )	返回当前可迭代对象a的下一个值
eval(s)	执行字符串表达式s，并返回该表达式的值

Python全部内置函数参考： https://www.runoob.com/python/python-built-in-functions.html

1.3 自定义函数

——函数定义的基本逻辑

实际工作中，我们往往需要自己定义函数。

```
def FunctionName(x1,x2):
    result = DoSomething(x1,x2)
    return result
```

对于上述提到的 FunctionName函数，其内部应遵守上图的逻辑，其中，def语句表示定义一个叫做FunctionName的函数，且要求参数为x1,x2。在函数体（缩进的部分）中对参数（输入值）做一定处理后，函数会使用return语句将某个变量（这里是result）作为返回值，赋予给调用函数时我们指定的变量y。至此函数的语句执行完毕，对于一个函数来说，return命令后面的语句均不会被执行。

注意⚠️，一个函数在编写时就已经确认好了将来调用时需要的参数，但由于默认参数、不定长参数的存在，这并不代表我们每次调用时都要告知函数全部的参数，告知一部分往往也是可以正常运行的。一个函数在编写时不一定要加入return语句指定返回值，但一定会有一个返回值。对于没有被指定返回值的函数，其返回值为None。

——函数的参数设置

在自定义函数时，我们可以设置必备参数、关键字参数、默认参数和不定长参数，以适应不同情况。

关于这四种参数的设置，见： https://www.runoob.com/python/python-functions.html

——使用*lambda*来创建匿名函数

Lambda语句被称为Python简介优雅的典型代表，它允许我们用一行语句定义一个小函数，且调用时能飞快地执行。对于一些简单的处理，lambda让我们能快速定义“即用即扔”的函数，而不需像def一样“长篇大论”，让代码更加简洁。

```
lambda [arg1 [,arg2,.....argn]]:expression
```

如上式，lambda后跟随的是一系列参数: arg1…argn，类似def语句里的参数；冒号后为使用这些参数表达的返回值expression，类似def语句里的return expression。

关于lambda和def的对比和示例，见附件3。

——函数变量作用域

一个程序的所有变量并不是在哪个位置都可以访问的，根据访问权限的不同，我们将所有变量划分成两个类型：全局变量与局部变量。

一般来说，定义在函数内部的变量为局部变量，仅能在被声明的函数内部访问；定义在函数外部的变量为全局变量，可以在程序任意地方被访问。

关于函数作用域的示例，见附件3。

2. Python面向对象

2.1 面向过程与面向对象

——面向过程

对于熟悉R/Matlab的同学，或常使用Python进行数据处理、画图的同学，可以简单粗暴地将执行这些任务时的编程思想归结为面向过程的编程。

在面向过程的编程任务中，我们就像是工厂流水线的设计者，在了解了原料（输入）和期待的成品（输出）后，就专注于每条生产线的步骤设置：该使用X函数进行第一步处理；该使用Y函数进行第二步处理……在这个过程中，我们的思维是线性的，往往可以先做完一步，再计划下一步。当流水线的最后一个步骤被执行完毕，程序也就完成了它的所有功能。

面向过程的编程非常符合直觉，也能自然地完成大多数轻量任务，但随着程序的复杂性逐渐上升，面向过程已经无法适应多人协作开发的生产流程。

——面向对象

面向对象编程最大的进步，就在于其使用了高度抽象的思维方法，将实际生活中的各个概念抽象成了对象。

继续使用上一个例子，在面向对象的编程任务中，我们不再是工厂流水线的设计者，而是整个工厂的设计者，为了让工厂能够正常运转，产出我们想要的结果，我们需要设计、安排好整个工厂不同部门的架构，而不仅仅专注于一条流水线。我们需要明确每个部门的大小规模、职责所在；我们需要给不同部门间安排通讯的方式；我们需要建立所有部门需要共同遵守的规章制度……

面向对象要求我们思考技术细节之外，还要思考更多架构上的问题，这也就是其能够适应更高级开发任务的原因。

更通俗地理解面向对象： https://www.zhihu.com/question/27468564/answer/575212256

2.2 类与实例化

Python是一门面向对象的语言，而面向对象在Python中的表现就是“一切皆对象”，但要解释“一切皆对象”对刚入门的同学比较困难，所以我们从类(Class)的概念入手，带大家了解Python中的面向对象设计。

类(Class)是用来描述具有相同属性和方法的对象的集合。它定义了该集合中，每个对象共有的属性和方法。对象(Object)是类(Class)的实例。

继续使用上一个例子，在面向对象编程任务中，我们需要关注工厂不同部门的设计，如开发部、销售部、财务部等，不同的部门在架构上有着很高的相似性，但在功能和细节上又各自不同，这是一个很适合用类(Class)来解决的问题。

我们将“部门”这个概念抽象成一个类(Class)，来描述一系列该概念公有的特性。开发部、销售部、财务部各自不同，但它们都从属于部门这个类。因此，我们定义Department这个类：

```
class Department():
    def __init__(self, name):
        self.name = name

    def callName(self):
        print(self.name)
```

如图，Python使用class语句来定义一个新类，这里将类名定义为Department。

类中可以包含属性(attributes)和方法(methods)，我们使用点(.)这个符号来调用实例化后的属性和方法。属性指类的一些描述性的值，以变量形式表示，如对于Department这个类，Department.name表示部门名，Department.number表示部门人数。方法指类的一些可执行的操作，以函数形式表示，如Department. 上班( )，Department. 解散( )等。

可以看到，在图片中，Department类具有一个name属性，以及两个方法:__init__( )和callName( )。其中，__init__( )方法为特殊方法，它规定了这个类在被实例化后执行的初始化操作，在这里，初始化操作为，将实例化该类时传入的name参数设置成该类的name属性。而我们自定义的callName( )方法则会将该实例的name属性print出来。

那什么是实例化呢？在完成了一个类的设计后，我们就可以复用它。我们称这个将抽象的类变成实际的对象的过程为实例化。例如，在设计完Department这个类后，我们可以将其变成一个个具体的部门，如开发部、销售部、财务部等。一个类只有在被实例化后，才可以查看它的属性(attributes)，调用它的方法(methods)。

在Python中，类的实例化非常简单。通常为如下形式：

$$y = ClassName(x1,x2\cdots\cdots)$$

与函数的调用很类似，该语句的含义为，使用ClassName类初始化所需的属性（在__init__方法里已经指出）创建一个新的ClassName类对象，并将其赋值给变量y。

关于类的实例化，见附件3。

2.3 多态、继承和封装

面向对象的三大基本特性为封装、继承、多态。

封装指将客观事物封装起来，成为抽象的类，并且把类自己的数据和方法只让可信的对象操作，对不可信的信息进行隐藏。例如，经过封装的财务部类，外部访问对象（如员工对象）只能得知自己的工资信息，不能从财务部类中获取到其他人的工资信息，而且也不了解财务部的详细工作流程。

继承指让某个类型的对象可以获得定一个类型的对象的属性和方法，继承可以让我们在设定和xx类相似的新类时，只需指定继承xx类就可获得xx类已有的属性和方法。例如，猴子类继承自哺

乳动物类，就自动获得了哺乳动物类的呼吸( )，哺乳( )等方法。在继承中，被继承的类被称作“父类”、“基类”或“超类”，继承的类被称作“子类”或“派生类”。

多态指不同内部结构的对象可以共享相同的外部接口。说人话就是尽管同一个类会经过实例化形成不同的对象，但这些对象都可以出现在该类可以出现的任何场合，享受同样的外部接口。例如，虽然开发部、销售部、财务部各自不同，但工厂想要评选年度最佳部门时，它们都可以成为年度最佳。

以上仅为封装、继承、多态的通俗解释，详细定义，以及更多面向对象的思想，在这里不做展开。

一篇不错的Python面向对象介绍：https://www.jianshu.com/p/9f3c8fca2e58

2.4 import语句

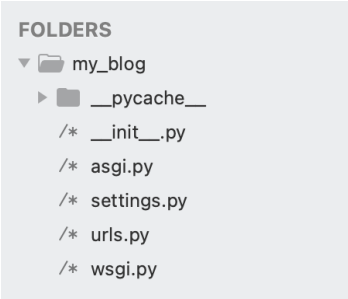
前文已经提到，Python为一门面向对象语言，“一切皆对象”。事实上，“一切皆对象”的原则不仅体现在类与实例化上，还体现在使用import语句进行模块(module)和包(package)的调用里。

——模块与包

模块(module)和包(package)是Python文件的两种不同组织方式。

通常，我们将一段Python保存于一个.py文件中，Python会将一个.py文件视为一个模块(module)。

而在更高级的开发中，若干个为同一整体服务的.py文件会被放在一个文件夹中，如果该文件夹中具有 __init__.py文件（该.py文件为空也可以），Python会将该文件夹视为一个包(package)。



如图，my_blog文件夹下有__pycache__文件夹，__init__. py文件等，因此my_blog为一个包，而包下面的asgi.py，wsgi.py等文件就为一个模块。

——使用import语句

我们使用import语句来调用其他文件中的Python代码，在这里我们参照上图进行讲解。

假设我们当前打开了 settings.py 文件，想要引用 urls.py 中的AFunction( )函数, 我们有两种写法：1. 在 settings.py 开头写下 import urls 命令，由于urls.py 与当前工作文件 settings.py 处于一个目录下（这很重要），Python可以顺利导入urls模块，此后在程序体中我们就可以使用

urls.AFunction( )来调用。 2.直接写下 from urls import AFunction， 此后在程序体中我们就可以使用AFunction( )来调用。

如果我们当前打开的文件并不是 settings.py ，而是某个和my_blog包平级的文件，那么为了达到相同的目的，我们得先导入my_blog包，上一段中的两个方法演变为：1. from my_blog import urls，使用urls.AFunction( )来调用。 2. from my_blog.urls import AFunction，使用AFunction( )来调用。

关于import语句的实践，见附件4。

——import语句与面向对象

很容易可以发现，import语句中，包、模块、函数/类的调用都使用了点(.)号来表示从属关系，如from my_blog.urls import AFunction。我们可以这样理解：每一个文件夹/文件都可以被看作一个对象，文件夹与.py文件、.py文件与其中定义的类/函数，在import语句的组织下可以互相调用，实现了复杂功能模块之间的互相联系，为高内聚、低耦合的面向对象设计建立了物理基础。

更多关于import语句： https://www.cnblogs.com/kungfupanda/p/5257174.html

3. Web爬虫基础

Web爬虫作为Python易上手、且实用的应用场景，非常适合来做我们第一部分学习的总结性实战。

3.1 爬虫的简明阐释

我们常常听到网络爬虫的概念，听上去很有黑客范儿，似乎和攻破数据库、大战防火墙有关。但事实上，网络爬虫仅仅是将公开在网站上的、本可以用人工收集但迫于效率低下的信息，用代码自动化的形式抓取而已。

因此，网络爬虫并不能抓取网上没有的信息，这些“网上没有”的信息往往存储在各大公司的服务器中。我没有能力，也不建议各位打这些数据的主意，因为这样的“爬虫”是会坐牢的。

网络爬虫可以简单分为Web端爬虫和App端爬虫，顾名思义，前者抓取网页端的信息，后者抓取移动端的信息。前者往往在电脑端就可以完成爬虫的全过程，后者可能会涉及到多设备之间的监听和抓包。今天，我们只介绍Web端爬虫，至于App端等有需要的时候再做讲解。

3.2 Web端爬虫流程

本部分代码见附件4

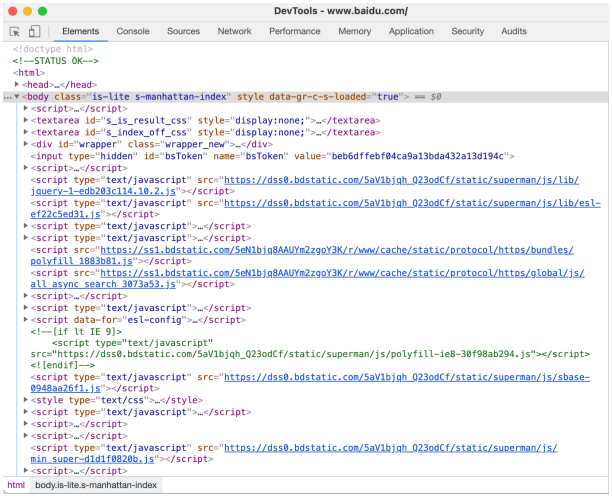
无论使用何种平台、何种方法，Web端爬虫往往遵循这三个流程：网站访问、定位数据、存储到本地。在这里，我们以requests+lxml的Web爬虫方案来过一遍这三个流程。

——网站访问

首先要明确的概念是，每一个网页都是以一个HTML文件的形式存在的，我们使用浏览器访问网站，本质上就是用浏览器解析.html文件，这和用Word解析.doc文件是一样的。

Lecture2

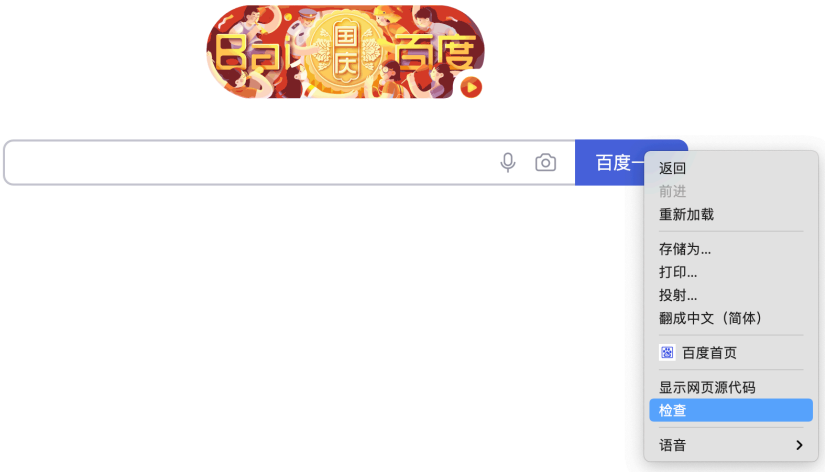
每个.html文件里的内容包含了网站的文字/图片/视频信息；网站的格式；网站的交互逻辑等。因此想要获取网站的信息，我们只需获得该网站.html文件里的信息即可。



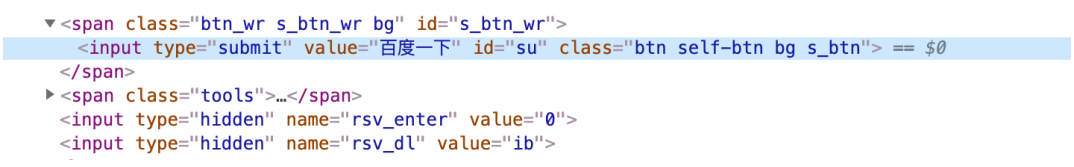
如上图，在浏览器（最好为Chrome系或Firefox）中打开百度首页，按下F12键，启动开发者工具，选择Elements/元素选项卡，即可看到百度首页对应的.html文件内容。

HTML文件有其基本语法，可见：https://www.runoob.com/html/html-tutorial.html

每个网页上的元素都被HTML文件中的某一标签表示，浏览器为我们提供了方便的定位元素的工具，如果想要了解网页上任一元素在HTML文件中的写法，只需右键点击，选择“检查”，即可跳转到对应的部分。



例如检查“百度一下”按钮后，浏览器会找到其所在位置，原来“百度一下”是某个input标签的value：



综上，想要进行网络爬虫，就必须先获取想要网站的HTML内容，Python为我们提供了可能是所有编程语言中最强大的网络请求模块——requests。

requests是一个第三方模块，需要安装和导入。我们使用pip来安装requests，pip的使用方法见Lecture1- 2.5。导入requests只需执行import requests即可。

使用requests访问某一网站非常直观自然(在这里使用get请求)：

$$r = requests.get(‘https://www.baidu.com’)$$

执行完语句后，r就变成了一个requests.models.Response对象，该对象有很多属性和方法，例如r.text表示当前网站html文本化后的内容，r.status_code表示此次访问的状态码。

requests的文档见： https://requests.readthedocs.io/zh_CN/latest/

——定位数据

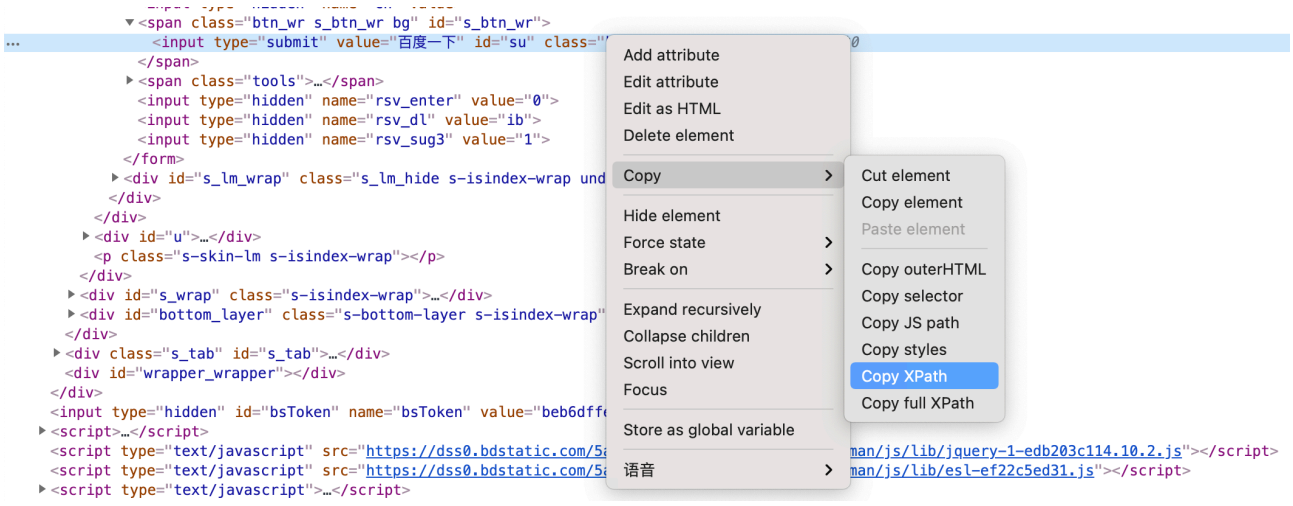
上一步，我们在r.text中得到了html的文本化数据，但这样的文本是杂乱无章的、非结构化的数据，计算机无法从中找到我们需要的内容。

HTML在编写时本身就是结构化的，它的不同元素层层嵌套，就像一棵树一样，非常利于查找和定位，但文本化的r.text里没有这样的定位信息。因此，我们需要让懂HTML的人来负责r.text的解析工作，Python的lxml库可以很方便地为我们自动识别。

lxml是一个第三方模块，同样需要安装和导入。我们使用lxml模块中的etree.HTML( )函数将r.text从繁杂的文本恢复成一棵便于查找和定位的树：

$$h = etree.HTML(r)$$

执行完语句后，h就变成了一个 lxml.etree._Element 对象，提供众多方便我们定位元素的方法，如h.xpath( )函数可以接受一个xpath地址，找到对应的元素。事实上，我们正是用xpath来定位html中不同的元素，xpath地址就像是现实生活中大家的居住地址。



浏览器为我们提供了方便的查看元素xpath地址的方法。如下图所示，我想要查询“百度一下”按钮的xpath地址，只需要右键点击该元素，选择copy-copy Xpath，即可获得xpath地址。

这样就可以进行轻松定位：

```
baiduyixia = h.xpath('//*[@id="su"]')
```

Xpath还可以用来定位多个元素、进行筛选等功能，非常强大
可以参见： https://www.runoob.com/xpath/xpath-tutorial.html

——存储到本地

本部分内容会在Lecture3中详细介绍，在这里不做展开，具体可以见代码。