
Table of Contents

介绍	1.1
KickOff Party	1.2
1.1 教学目的	1.2.1
1.2 软件和工具	1.2.2
1.3 课前自学	1.2.3
Python 简介	1.3
2.1 数据工程介绍	1.3.1
2.2 Python 介绍	1.3.2
2.3 Python 语法	1.3.3
2.4 Python 请求	1.3.4
2.5 Python 语法进阶	1.3.5
爬虫	1.4
3.1 HTML	1.4.1
3.2 爬虫实战	1.4.2
3.3 选学：JSON 简介	1.4.3
3.4 选学：爬虫的难点	1.4.4
数据处理	1.5
4.1 初见数据处理	1.5.1
4.2 数据处理应用实例	1.5.2
4.3 选学：Pandas	1.5.3
Demo练习	1.6
大作业	1.7
分享与后续学习建议	1.8

概述

此教程是专门为 Girls Coding Day Python 爬虫 女性编程日工作坊而设计。

本教程由吕戈设计，文洋校订。

This work is licensed under a [Creative Commons Attribution-NonCommercial 4.0 International License](#).

Kickoff Party

本章节目标

1. 下载和安装工作坊所需要的软件
2. 了解本工作坊的学习目标
3. 自学 HTML 和 Python 基本知识

教学目标

- 了解数据工程，获得数据工程的大体印象。
- 了解数据工程师的成长路径，知道如何去获取相关资源。
- 掌握相关工具完成数据获取和分析的任务。

知识点目标

- 了解一些编程思维，打破对编程的神秘感。
- 对数据工程和爬虫有大体印象。
- 了解 Python 的语法，基本数据结构，会使用条件，循环和输入/输出，知道如何输出数据到文件。
- 了解网站、网页、请求、响应的大致原理。
- 掌握 HTML 的基本元素，类和 id，选择器，并会使用浏览器查看 DOM 和选择器。
- 了解什么是 Web Service。了解 JSON 的基本语法。
- 了解 Python 发请求，处理 HTML 和 JSON 的相应库的使用。
- 了解一些 Web 服务或者信息网站，知道如何搜索常见网站的 API。
- 学习基本的数据处理方式。
- 设计一个爬虫应用，并展示爬取的数据（图表或者爬出来的数据）。

软件和工具

Cooking time: 30min active / 30-45min passive

- 下载并安装 浏览器 Chrome
- 下载并安装 集成开发环境 Anaconda，记得选择Python 3.7 Version



- 你是否好奇「集成开发环境」是什么？那么赶紧用搜索引擎搜索一下关键字「集成开发环境」吧。
 - 略读 [集成开发环境 - 维基百科](#)
- 注意：
- Anaconda 的默认安装已包含教程中涉及的 Spyder, Pandas, Requests, BeautifulSoup 等软件。
 - Anaconda 的安装包比较大，最好提前下载安装好。

Tasks For Coaches: 请教练简单介绍这些软件。Tips：尽量使用类比方法，让学员快速理解这些软件的用途。

辅助资料

Python 小抄：

[Laurent Pointal's Python 3 Cheat Sheet](#)

方便的话可以双面彩打到一张 A4 纸上作为学员手边的参考。

1.3 课前作业

Cooking time: 3 hours active / 3-5 hours passive

1. 从你的同组学员中，找一个小伙伴，两两结对互相监督学习

2. 完成下来自学任务

- 阅读HTML教程

手机上有 W3C School的App，大家可自行搜索下载。充分利用上下班通勤时间学习。HTML只需要看下列知识点即可

- [HTML 简介](#)
- [基本的 HTML 标签 - 四个实例](#)
- [HTML 元素](#)
- [HTML 属性](#)
- 根据 [HTML 参考手册](#) 掌握下列知识点：
 - 常见元素(div, a, img)
 - 常见属性(id, class, href, src)

- 完成Python基础

仅完成Python基础部分即可，其他部分可不阅读。

Tasks For Coaches:

1. 每天找一个合适的时间（根据各组情况也可以两天一次，我们鼓励每天都联系），跟大家语音or视频开站会：了解学员的学习情况，鼓励她们积极学习。

- 站会一般不超过15分钟；
- 建议在站会问的三个问题（更多信息 [每日站会，自组织的量度 - Scrum中文网](#)）：
 - 我今天做了什么帮助我完成学习目标？
 - 我明天打算做什么帮助我完成目标？
 - 我在学习过程中遇见了什么困难疑惑？

2. 如果已经完成上述材料的阅读，鼓励学员预习本教程后续内容。

初见 Python

本章节目标

1. 掌握 Python 的基本语法
2. 掌握爬虫所需第三方库的调用方法

数据工程介绍

Cooking time: 10 mins active / 15 mins passive

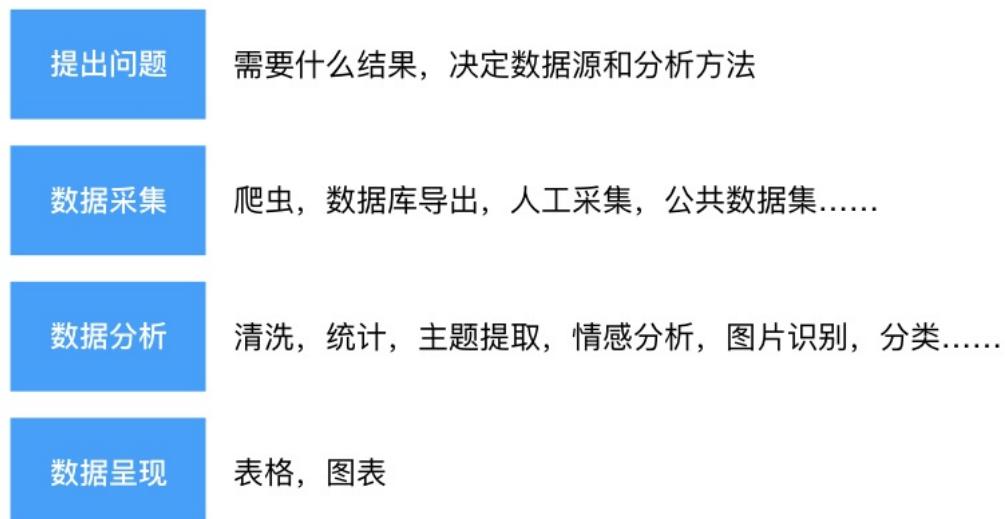
Tasks For Instructors: 讲师介绍数据工程概述，帮助学员了解数据工程全局观。

什么是数据工程？

数据（資料），Data（Datum 的复数）经常和「信息」「資訊」可以互用。

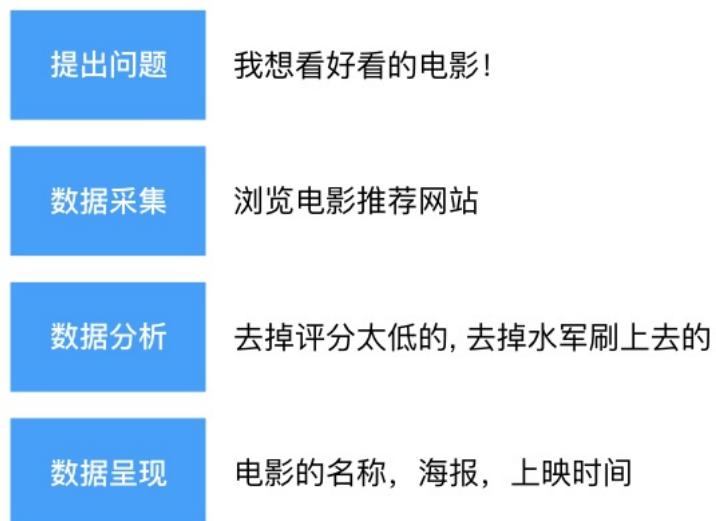
人类每天都会生产出大量的数据，不仅人类，各种各样的检测仪器都会产生数据。数据可以是数字，文本，图片，图像等各种可以被数字化的信息。

数据工程，就是用工程的手段来处理数据。数据工程基本会涉及下面几个方面：



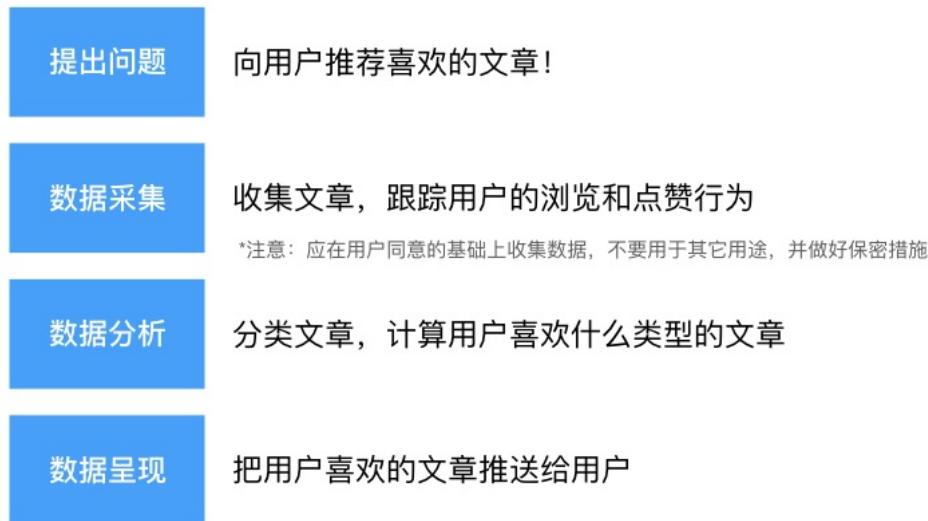
一个运用数据处理的例子：

例 1



上面的例子用到了一个 IT 工具：浏览器。而数据工程师处理的问题经常是这样的：

例 2



要处理这个问题，需要掌握更多更强大的工具，并对数据有更深刻的认识。

数据工程师的成长路径

初级入门：

- 有一定的英文基础
- 掌握编程技能（如 Python, R, SAS 等）
- 理解 Web 的原理，会搭建简单的 Web 服务和编写网页
- 学习统计学和回归分析（需要概率论和微积分基础）
- 学习数据存储和管理：数据库、SQL、搜索引擎
- 掌握采集数据的手段：爬虫，日志收集，Analytics 等

锻炼编程技能，扩展知识面，加深对数据的理解：

- 学习算法，计算机网络基础
- 学习 Linux 系统管理
- 学习机器学习和数据挖掘
- 学习自然语言处理和图像模式识别

当数据来源和种类都变得越来越多，变成了大数据：

- 学习数据仓库，ETL
- 学习分布式系统，应用大数据处理框架
- 掌握深度学习，用大数据训练更复杂的神经网络
- 设计数据产品，思考从数据中可以挖掘的价值

现在有很多优秀的网络课程教授数据工程，例如 <https://www.coursera.org/specializations/jhu-data-science>。任何人只要肯努力就可以掌握这项技能。

数据分析职业的分支

随着大数据的兴起，数据分析职业也越来越专业和细分，相关的职业有：

- 业务分析员 - 用技术提升业务效率和产出
- 数据分析师 - 运用数据模型解决问题
- 数据科学家 - 数据分析师的进阶，往往包括设计专用的数据分析模型
- 数据工程师 - 运用数据采集和分析工具，实现算法
- 数据架构师 - 统筹整合数据资源，架构数据平台
- 市场分析师 - 研究市场数据，帮助营销决策
- 量化分析师 - 运用数据工程进行量化交易，自动交易
- 统计专员 - 应用统计学知识分析数据，设计和制作报表
-

参考 <https://www.mastersindatascience.org/careers/data-analyst/>

一些问题

问：我想成为数据工程师，数学不好怎么办？

答：首先，很多分析工具不需要完整的数学知识就能使用。其次，掌握一些编程思维后，对学习数学会有不少启示和帮助。再其次，这次活动并不需要数学基础。

问：我并不想成为数据工程师，了解这些有用吗？

答：工程思维、编程思维对其他行业和学科也有意义。更重要的是：如今人们能获取的信息渠道都是被企业和组织牢牢掌握，信息流产品大行其道，我们获取信息的方式越来越被动。如果我自己不掌握主动采集、分析信息的能力，那就很难获取到及时的、真实的信息。

数据工程师的一天

数据工程师的一天，by 毛苏晗：

数据工程师的一天当然是搬砖啦！

数据就是一块块砖，每一块砖都要经过无数次的搬运才能从砖场放到建筑中的合适的位置。搬砖的过程就叫ETL。当然啦，要是砖搬得不好，质量出了问题，那就要全部推倒重来啦！我们辛辛苦苦的一层一层盖好了楼，最怕的就是都快封顶了，结果人家说地基有问题，需要把楼拆了重建，那就是真是倒了血霉了。毕竟数据都是一层一层有着依赖关系，最源头的数据质量必然会影响到之后的每一层，所以只能老老实实的重新将搬砖进行到底。好不容易把楼搭好了，还要定期检查砖的质量，是不是有砖松了，有砖老化了，有砖被虫子蛀掉了（什么虫子这么强），有问题的砖就要重新补或者换掉，真是一个繁琐的工作啊！我们需要对每个表的每一行的每一个列负责，这就是数据工程师的使命。

虽然是搬砖，听上去很无聊，但是其实我们的生活还是很多彩的。每天第一件事就是打开工作流页面，就会看到里面各种的任务呈现出了五彩斑斓的颜色，一个屏幕满满的都是五彩的节点，黄色的是失败了，蓝色的是正在运行的，绿色的是成功的，黄色的是被阻塞的，红色的是被取消了，灰色的是正在排队。而且每天都会有新的工作流重新生成，每个有问题的节点都要点开查找原因，是不是很酸爽？！我们这一行，最怕的就是色盲，你说要是把失败的看成是成功的，那直接就是P1 issue了有木有！

虽然有这么多吐槽，但是不可否认的是这是一个很有难度的工作，能够将庞大复杂的数据治理好，绝对是非常有挑战非常值得自豪的一件事情。

Python 简介

Cooking time: 15 mins active / 20 mins passive

Python 是计算机入门首选语言之一，编写简单，有丰富的生态系统和库，在很多行业和学科都有出色的表现。当前的稳定版本是 3.7。

运行 Python 程序需要 Python 解释器（其实是编译+执行）。

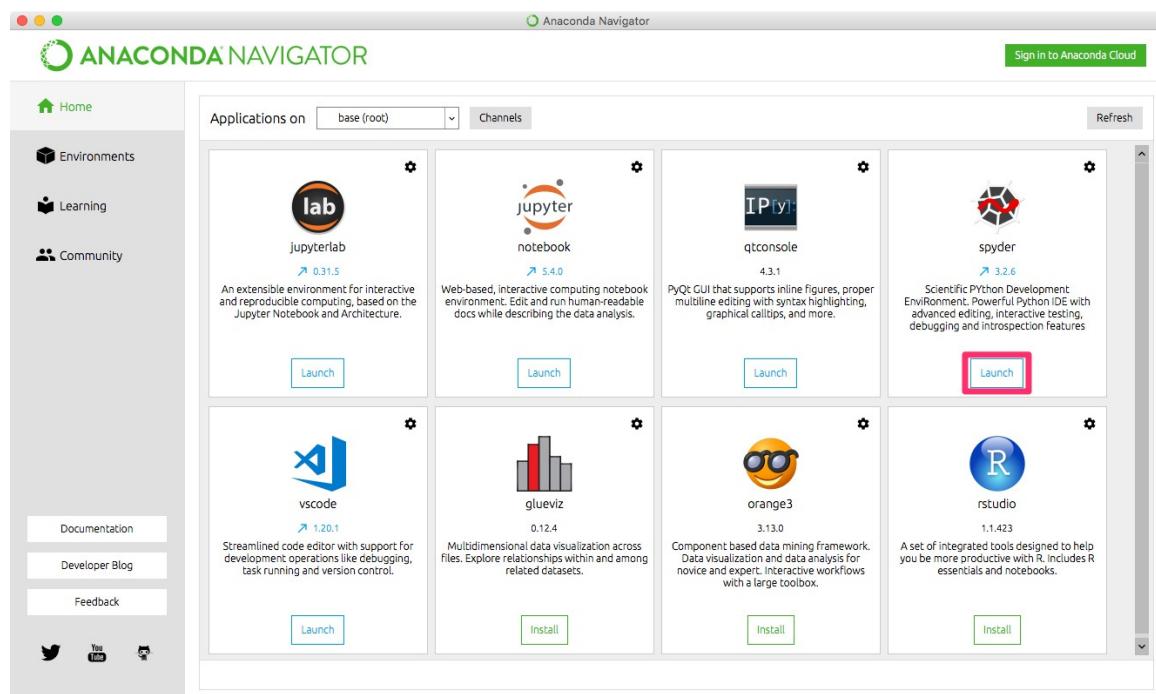
Python 的源代码文件扩展名是 `.py`。

Task for Coaches:

- 解释什么是解释器
- 解释什么是编译
- 解释什么是执行
- 解释什么是源代码

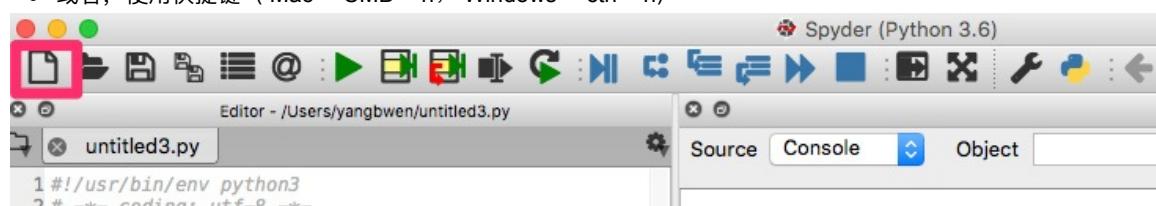
第一个 Python 程序

1. 使用IDE Anaconda打开 spyder

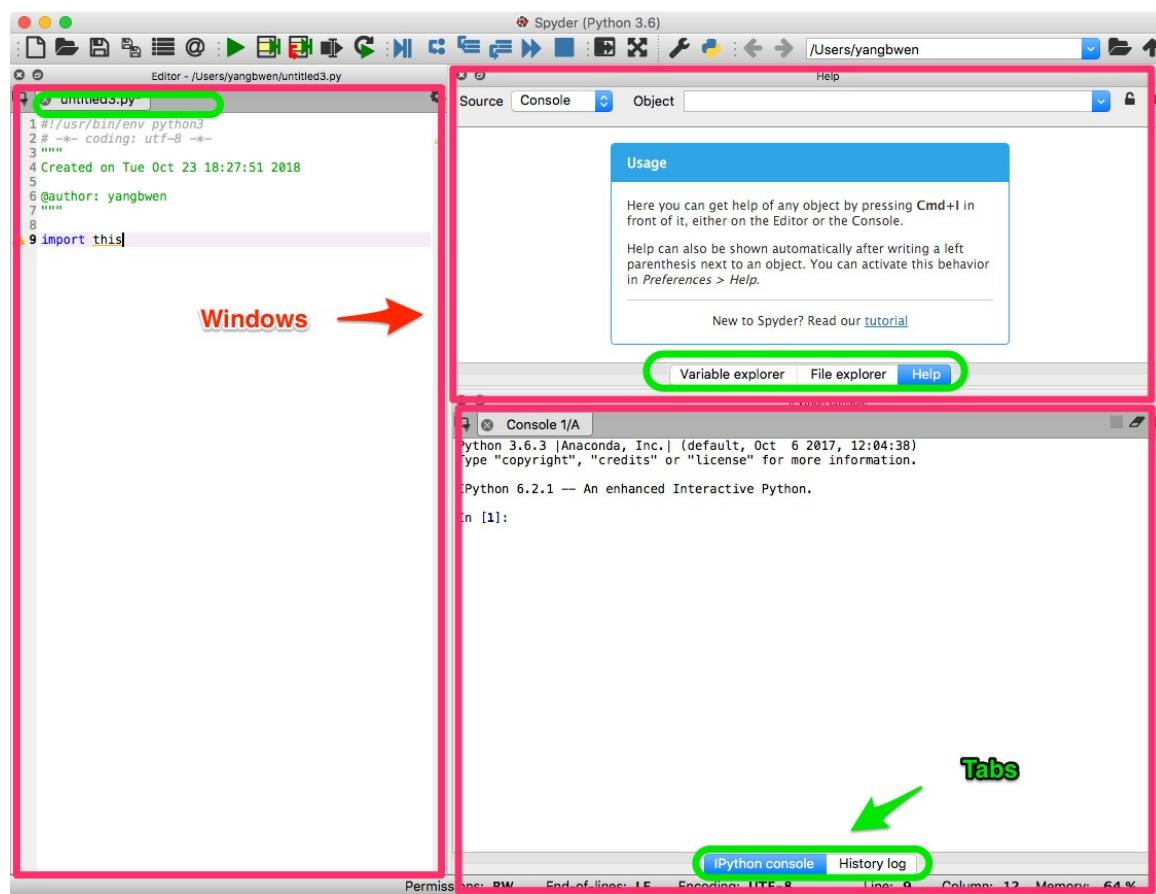


2. 新建python文件：

- 点击 spyder 右上角第一个图标
- 或者，使用快捷键（ Mac : CMD + n; Windows : ctrl + n）



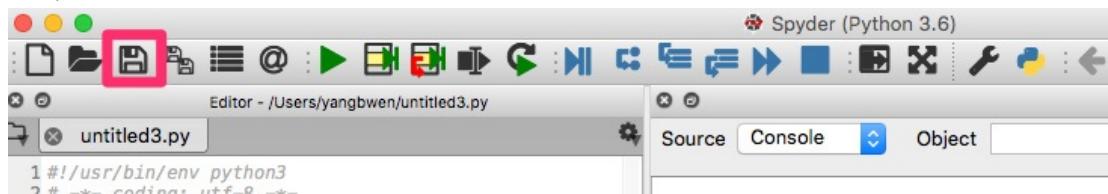
3. 我们先看看spyder的界面， spyder包含多个单独的窗口（标有红色方框）， 每个窗口都有赞成的标签（标有绿色圆框）



4. 在编辑器中输入 `import this`
 5. 保存Python文件（注意Python文件都是以 .py 结尾的文件）：

Task for Learners: 请使用搜索引擎搜索「[Python文件命名规范](#)」，随意阅读几篇介绍Python命名规范的文章。

- 点击 spyder 右上角第三个图标
- 或者，使用快捷键（ Mac : CMD + s; Windows : ctrl + s）



6. 运行Python文件：
 ◦ 点击 spyder 右上角第七个图标
 ◦ 或者，使用快捷键（ Mac : F5; Windows : F5）

The screenshot shows the Spyder IDE interface. On the left is the Editor pane with a file named 'my_first_program.py' containing the following code:

```

1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 Created on Tue Oct 23 18:27:51 2018
5
6 @author: yangbwen
7 """
8
9 import this

```

The right side of the interface has a 'Console' tab active, displaying the output of the code execution. A red box highlights the output area, which contains the 'Zen of Python' poem by Tim Peters:

```

In [1]: runfile('/Users/yangbwen/Desktop/my_first_program.py', wdir='/Users/yangbwen/Desktop')
The Zen of Python, by Tim Peters

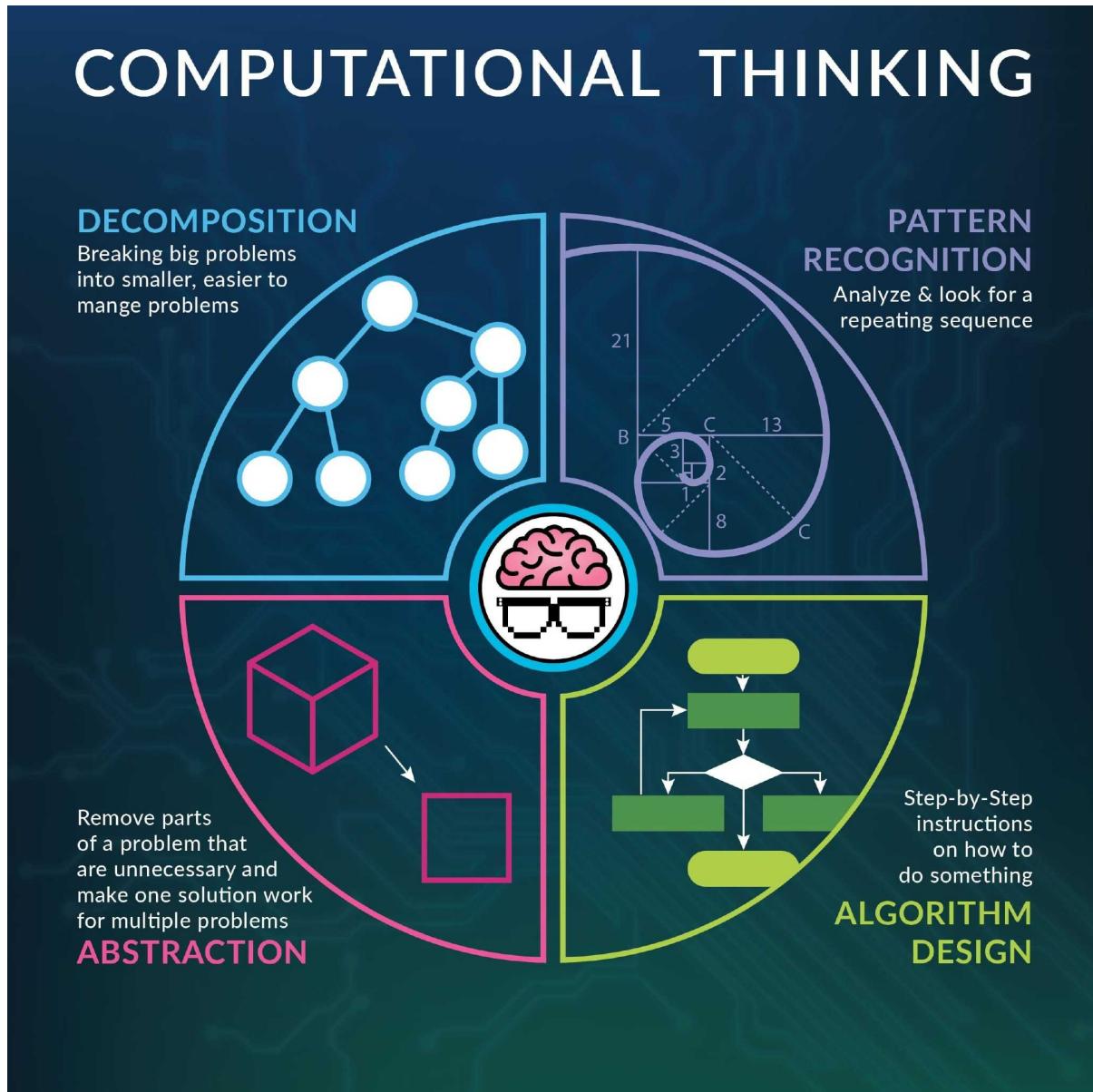
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!

```

可以看到 Python 的哲学，描述于一首诗中。上面的代码中，`import` 语句的作用是引入一个包，而这个包 `this` 的作用是打印这首诗。

编程思维 - 任务分解

编程的第一步就是把要做的事情分解成一步一步，用计算机语言讲清楚，让计算机可以执行。



Donald Knuth 说过：一个人并不真正了解一件事，除非他把一件事交给计算机——特别是用算法把它描述出来。

It has often been said that a person does not really understand something until he teaches it to someone else. Actually a person does not really understand something until after teaching it to a computer, i.e., express it as an algorithm.

Learning by Additional Reading: [Google for Education: Computational Thinking](#) (需要科学上网，拓展阅读，非本工作坊所需)

程序出错

有一个说法叫：计算机很笨却听话（The computer is very dumb but obedient）。当遇到语法错误时，即使是很微小的错误，它也不会自己修正它。程序员每天都会遇到很多各种语法错误。只要仔细阅读错误信息，找到并修正就可以了。

Learning by Additional Reading: [Introduction to Computer Programming - What Is It](#)

打印一段话

Learning by Doing:

Python 的 `print` 函数不是打印到纸上，而是打印到屏幕上（确切来说是标准输出流）

```
print("I can program!")
```

Python 语法

Cooking time: 30 mins active / 45 mins passive

Learning by Doing:

- 建议切换到英文输入法编程，可以避免一些用错符号的错误。
- Tasks for Learners:** 请学员创建第二个 Python 文件 `python_syntax_practice.py`，对下列 Python 语法进行练习。
- Tasks for Coaches:** 请教练对学员的练习提供一定的辅助，请控制好时间。

魔法注释

惯例：把这一行写在 Python 文件的头一行，表达这个文件的非 ASCII 字符的编码为 UTF-8，不然遇到中文会出现编码错误。

```
# encoding: utf-8
```

注释

注释是写给阅读代码的人看的，计算机会忽略注释。一行中以 `#` 开始直到行末的部分都属于注释。

```
# 可以在代码中穿插写一些笔记，不影响程序的执行。
```

基本数据类型

整数，类型为 `int`

```
1234  
-1234
```

小数（浮点数），类型为 `float`，例如

```
3.1415926
```

字符串，类型为 `str`

```
"双引号字符串" # 注意要用半角英文引号包起来  
'单引号字符串'
```

字符串中，空格也是字符：

```
" " # 这也是一个字符串
```

有一些字符也是没有对应的字形（Glyph），但是可以用 转义字符 表达：

```
"\n" # 换行  
"\r" # 回车
```

```
"\t" # tab
```

以上的字符串写法有个限制：只能写在一行内。Python 支持三个引号的多行字符串语法，方便书写多行的内容：

```
"""
床前明月光
疑是地上霜
举头望明月
低头思故乡
"""
```

特殊量 -- 代表什么都没有

```
None
```

查看数据的类型

```
type(3) # int
```

运算

在 IPython 中输入下面的表达式试试：

```
1 + 1
2.5 * 3
4 / 2
```

% 返回除法的余数：

```
5 % 3
```

除以 0 会得到 NaN (Not a Number)

```
1 / 0 # NaN
```

字符串也可以进行连接运算（使用 + 运算符）

```
"hello" + " world"
```

注意：全角符号和半角符号的区别。

变量

用一个名称代表一个值 (= 叫做赋值运算符)

```
a = 3
some_variable_1 = "一个值" # 变量名可以包含字母、数字和下划线，但不能以数字打头
```

这个符号代表的值可以重新赋值，所以这个量称为变量 -- variable。

运算的结果可以赋予变量，变量也可以参加运算，例如下面计算平均值的代码：

```
sum = 1 + 2 + 3 + 4 + 5
average = sum / 5.0
print(average)
```

条件

在 Python 中表达逻辑：“如果”满足某种条件“那么”你就给我做这件事情。

布尔类型（注意大小写）

```
True
False
```

条件语法

```
if True:
    print("真")
if False:
    print("假")
```

注意：

- 「如果」部分以关键字 `if` 开始，接着是条件表达式，后面必须再加上冒号 `:`
- 「那么」部分的缩进要比 `if` 行深，推荐用 4 个空格缩进（参考 [PEP8](#)）

上面两个条件内容互斥，我们可以用双路条件 `if else :`

```
t = True
if t:
    print("真")
else:
    print("假")
```

布尔类型上可以用布尔运算：

- 而且 `and`
- 或者 `or`
- 否定 `not`

练习：思考下面几个表达式的值并在 IPython 验证：

```
True and True
not False
False or True
```

比较运算符可以判断两个值是否相等或者孰大孰小：

- `==` 等于（两个等号，注意和赋值运算的不同）
- `>` 大于
- `<` 小于
- `!=` 不等于
- `>=` 大于或者等于
- `<=` 小于或者等于

练习：思考下面几个表达式的值并在 IPython 验证：

```
1 <= 3
5 > 4 and 4 > 3
```

综合运用条件语句和比较运算符：「如果天气晴朗并且低于 30 度就出去玩」

```
weather = "rainy"
temperature = 20

if weather == "sunny" and temperature < 30:
    print("lets go out!")
```

循环

计算机是处理数据的机器。循环（迴圈）告诉机器重复做相同的工作。

`while` 循环打印 0 到 100

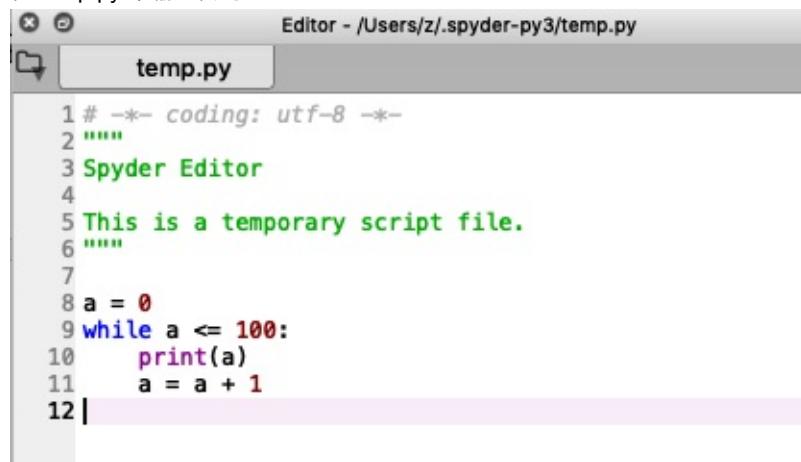
```
a = 0
while a <= 100:
    print(a)
    a = a + 1
```

同样注意别忘了 `:` 和缩进

循环一般由循环条件和循环体组成。

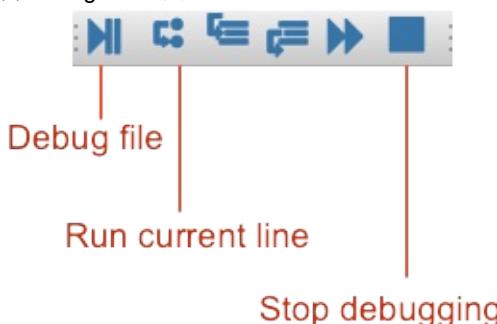
练习：使用 Spyder 观察程序的执行：

1. 在 temp.py 中输入代码

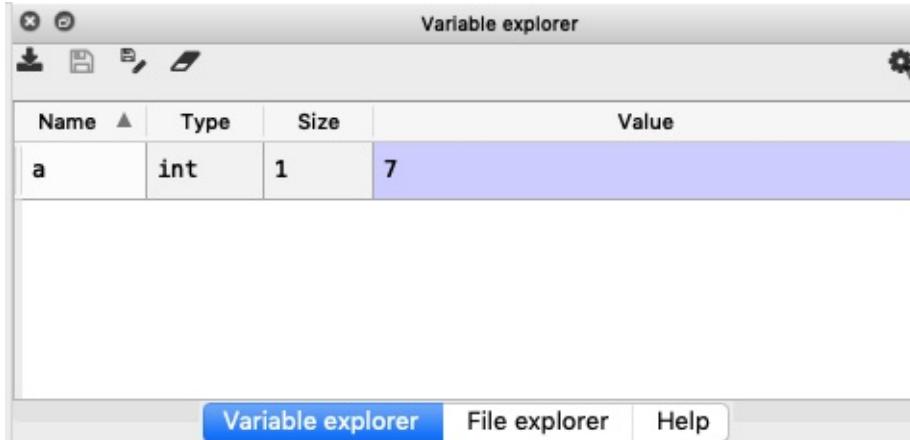


```
temp.py
1 # -*- coding: utf-8 -*-
2 """
3 Spyder Editor
4
5 This is a temporary script file.
6 """
7
8 a = 0
9 while a <= 100:
10     print(a)
11     a = a + 1
12 |
```

2. 点击 Debug file 工具



3. 多次点击 Run current line 工具，观察 Variable explorer 的变化



4. 最后点击 Stop debugging 结束 debug

死循环 的意思是：不会结束的循环。

```
import time
while True:
    print("I am stuck")
    time.sleep(1)
```

要结束程序，可以按 Ctrl + C 退出。（一个手指按住 Ctrl 键，然后另一个手指按 C 键，退出时解释器会报告一段错误，意思是收到了中断信号）。

循环控制：

- `continue` 进入下一个轮回
- `break` 结束循环

例：下面的代码打印 1 到 10，通过判断 `i` 是否增加到 10 决定结束循环：

```
i = 1
while True:
    print(i)
    if i == 10:
        break
    i = i + 1
```

例：下面的代码打印 1 到 10 之间的偶数，当 `i` 是奇数时通过 `continue` 跳过循环体中的其余代码。

```
i = 1
while i <= 10:
    if i % 2 == 1:
        i = i + 1
        continue
    print(i)
    i = i + 1
```

函数

我们学习了用一个词代表值（变量），在 Python 中我们也能用一个词代表一段代码。

定义函数用 `def` 关键字，接上函数名称，参数列表和冒号。试理解下面的代码：

```
def double(x):
    return x * 2
```

在缩进的代码块中编写函数体，使用 `return` 返回需要的内容。

之前我们学习的 `print`, `len` 都是函数，而且是 Python 自带的函数 (built-in functions)。

调用函数：

```
double(4)
```

由于函数返回的是一个值，这个值也可以赋予给变量：

```
twice = double(4)
four_times = double(twice)
```

函数返回的值也可以传递给下一个函数：

```
print(double(4))
```

小提示：Python 的运算符 `+`, `-`, `*`, `/` 等等其实都是函数。

输入和输出简介

IO 是输入/输出 (input / output) 的缩写。

IO 是程序能从真实世界获取信息 (输入)，对真实世界产生影响 (输出) 的基础。

典型的输入包括：用户键盘输入，鼠标点击，读取文件，读取网络数据，获取传感器的读数，……

典型的输出包括：在屏幕显示文字，在屏幕绘制图像，输出到打印机，发出声音，发送数据到网络端口，……

读写文件

读写文件是 Python 里最基本的 IO 操作之一。

`open` 函数打开一个文件，`"r"` 模式是读取，`"w"` 模式是写入。

以读取模式打开一个文件 "foo.csv"，并返回对应的文件对象：

```
f = open("foo.csv", "r")
```

关闭一个文件对象：

```
f.close()
```

下面的代码打开文件，读取文件的内容，然后关闭文件。

```
f = open("foo.csv", "r")
data = f.read()
f.close()

print(data)
```

但是，在对文件操作的时候有可能发生异常！例如文件编码不正确，或者出现了没考虑到的某种状况，就会出错。

没处理各种异常情况的一段程序，不能成为一个产品

Python 提供了 `with .. as` 语法，不用处理异常和写 `close`，就能保证处理文件后能自动关闭它。

读取一个文件

```
with open("foo.csv", "r") as f:  
    data = f.read()
```

写入一个文件

```
with open("foo.csv", "w") as f:  
    f.write(data)
```

Python 请求

Cooking time: 15 mins active / 20 mins passive

Tasks For Instructors: 请讲师简单介绍 Python 发请求, HTTP 知识。

本节稍微了解一下便可

HTTP 简介

HTTP 是超文本传输协议 (hypertext transfer protocol) 的缩写。

(常见语病: 「HTTP 协议」。「P」和「协议」同义重复了, 只用「HTTP」即可)

HTTP 规定了客户端和服务器之间传输网页和文件的办法。

客户端: 例如浏览器、爬虫、手机应用。服务器: 用网址定位, 一般运行于一台电脑上的 80 或者 443 端口。

在 HTTP 语境下, 网页或文件一般统称资源 (resource)。

HTTP 请求由以下部分组成:

- 请求行, 包含请求动作 (verb) 和请求路径, HTTP 的版本, 例如 `GET /images/logo.gif HTTP/1.1`
- 多个请求头, 例如告诉服务器我希望获得中文的响应 `accept-language: zh`
- 一个空行
- 可选的消息体 (body)

常用的 HTTP 请求动作 (verb) 有

- `GET` 一般用来获取一个资源, 例如 `GET /images/logo.gif HTTP/1.1`。
- `POST` 一般用来提交一个资源, 常用于表单提交或登录。

(verb 的使用只是一种惯例, 也有网站不遵守的)

HTTP 响应由以下部分组成:

- 响应行, 包含 HTTP 版本和响应代码, 例如 `HTTP/2 200`
- 多个响应头, 例如告诉客户端消息体的格式 `content-type: image/gif`
- 一个空行
- 可选的消息体

常见的响应代码

- 2XX 正常工作, 如 200 OK, 201 Created
- 3XX 重定向, 如 302 跳转, 304 网页没变化
- 4XX 客户端出问题了, 如 400 非法请求, 404 资源没找到
- 5XX 服务器出问题了, 如 500 服务器内部错误, 504 后台服务超时

使用 Chrome 开发者工具查看请求

打开菜单: 视图 -> 开发者 -> 开发者工具, 选择 "网络" 标, 刷新页面

• POST 一般用来提交一个资源，常用于表单提交或登录。
(verb 的使用只是一种惯例，也有网站不遵守的)

Name

- 4-python-request.html
- /2.Python
- style.css
- /gitbook
- website.css
- /gitbook/gitbook-plugin-highlight
- search.css
- /gitbook/gitbook-plugin-search
- website.css
- /gitbook/gitbook-plugin-fontsettings
- gitbook.js
- /gitbook
- theme.js
- /gitbook
- plugin.js
- /gitbook/gitbook-plugin-livereload
- search-engine.js
- /gitbook/gitbook-plugin-search

x Headers Preview Response Cookies Timing

General

Request URL: http://localhost:4000/2.Python/4-python-request.html
Request Method: GET
Status Code: 200 OK
Remote Address: ::1:4000
Referrer Policy: no-referrer-when-downgrade

Response Headers view source

Accept-Ranges: bytes
Cache-Control: public, max-age=0
Connection: keep-alive
Content-Length: 19116
Content-Type: text/html; charset=UTF-8
Date: Fri, 26 Oct 2018 08:12:58 GMT
ETag: W/"4aac-166af6efbac"
Last-Modified: Fri, 26 Oct 2018 08:12:58 GMT
X-Current-Location: /2.Python/4-python-request.html

Requests 的使用

`Requests` 是一个方便的 HTTP 请求库。

```
import requests
response = requests.get('https://c.xkcd.com/random/comic/')
print(response.text) # 打印网页的 HTML 源代码
```

可以看到以 `<!DOCTYPE html>` 开头的 HTML 内容，下面我们先讲解一下 Python 进阶语法，然后再介绍如何从 HTML 中提取信息。

如果请求的是一张图片，图片文件往往包含大量的非字符数据，需要用 `response.content` 获得内容

```
response = requests.get('https://ws4.sinaimg.cn/large/006tNbRwly1fwict5oyqdj31kw1kw124.jpg')
with open("crawled_image.jpg", 'rb') as f:
    f.write(response.content)
```

因为写入的是二进制数据，这里的 `open` 使用了 `'rb'` 参数表明用二进制输出流写文件。

`requests` 还提供了方便的方法获取网页上的所有链接：

```
response.links
```

Python 语法进阶

Cooking time: 15 mins active / 20 mins passive

Learning by Doing:

- 提示：切换到英文输入法编程可以避免一些用错符号的错误。
- Tasks for Learners:** 请学员创建第3个Python文件 `python_advanced_syntax_practice.py`，对下列Python语法进行练习。
- Tasks for Coaches:** 请教练对学员的练习提供一定的辅助，请控制好时间。

下面介绍更多的 Python 语法。

如果对某个语法不熟悉，推荐到 Python 的官方网站寻找详细的 Python 语法参考：<https://docs.python.org/3.7/reference/index.html>。

拼接字符串

使用 `f"..."` 和 `{}` 可以把变量的值拼接到字符串中去。试试下面的例子：

```
me = "程序员"
sentence = f"hello, {me}"
print(sentence)
```

列表

列表（`list`）是一种顺序的数据。列表中的每个元素都对应一个整数下标，从 0 数起。

创建一个空的列表：

```
li = []
```

创建一个带 3 个元素的列表：

```
li = ["zero", "one", "two"]
```

查看列表的长度：

```
len(li)
```

取出列表中的元素：

```
li[1]
```

往列表中添加成员：

```
li.append(1)
```

用 `for in` 遍历列表：

```
for e in li:
    print(e)
```

这个循环也可以用 `while` 来写，相当于：

```
i = 0
while i < len(li):
    print(li[i])
    i = i + 1
```

`for` 比较方便迭代列表。

字典

创建一个字典（注意用英文花括号，不要用中文的花括号）：

```
di = {}
```

用 `for` 遍历字典：

```
for key, value in di.items():
    print(f'{key} {value}')
```

方法

方法（method）是定义在对象上的函数，可以通过 `.` 语法调用一个方法。

你可能会疑惑.....

问：程序员都记得所有的语法和函数吗？

答：并不！程序员也需要查阅文档，重新学习各种库和函数的用法，甚至到网上提问。网上还有不少问答网站，例如 [StackOverflow](#) 和 [SegmentFault](#) 都记录了很多常见的问答。如果找不到需要的答案，你也可以在上面提出自己的问题。

初见爬虫

本章节目标

1. 掌握 HTML 的基本元素，类和 id，选择器，并会使用浏览器查看 DOM 和选择器。
2. 了解一些 Web 服务或者信息网站，知道如何搜索常见网站的 API。

HTML 与 爬虫

Cooking time: 20 mins active / 25 mins passive

Tasks For Instructors: 讲师讲解和演示该节内容，若相关内容认为需要教练辅助，请讲师灵活应变。

为什么学习爬虫要掌握 HTML 基础

HTML 是超文本标记语言 (hypertext markup language) 的缩写。大部分网页都是以 HTML 写成的。

(常见语病：「HTML 语言」。「L」和「语言」同义重复了，直接说「HTML」即可)

网络爬虫的工作就是代替人去阅读网页，抽取数据。人看到的是渲染好的网页投射进眼睛的影像，而爬虫看到的是 HTML。

注：MDN 是一个比较全面的 Web 初学者的教程，介绍 HTML，CSS 和 JavaScript，上面还有非常全面的 Web 参考。

HTML 的构造

```
<!DOCTYPE html>
<header>...</header>
<body>...</body>
</html>
```

常见元素

- 块 `<div>`
- 行内 ``
- 链接 `<a>`，爬虫往往需要特别关注它的 `href` 属性
- 图片 ``
- 视频 `<video>`

常见的属性

- `href` 链接所指向的地址
- `src` 图片的地址
- `id` 用来指定元素唯一的身份证号码
- `class` 表示元素属于哪些类别，多个类别之间用空格隔开

体验：在 Chrome 浏览器中，右键查看元素。鼠标移到不同的元素上面，观察页面哪一部分高亮了。

路径和地址

我们在页面上找到的链接可能是相对路径、绝对路径或者地址。常见的地址格式：

- `http://foo.com/bar` 这是一个 HTTP 地址
- `/some_page?n=3` 这是一个绝对路径，完整的链接需要和它所在页面对应的域名联立起来
- `//foo.com/bar` 这是一个地址，省略的 scheme 和它所在页面相同，可以补足为 `http://foo.com/bar` 或者 `https://foo.com/bar`

CSS 选择器

CSS 是层叠样式表 (Cascade Stylesheet) 的缩写。

(常见语病：「CSS 样式表」。「S」和「样式表」同义重复了，直接说「CSS」即可)

在 CSS 中，选择器 (selector) 可以用于定位我们想要样式的 HTML 元素。虽然我们这里不需要给 HTML 元素指定外观和样式，但选择器可以帮助我们选取需要的元素。

- 元素选择器 `div`, `span`, `a`
- id 选择器
- class 选择器

例：假设有以下的 HTML 页面

```
<p id="question">What color do you like?</p>
<div>I like blue.</div>
<p class="paragraph red">I prefer red!</p>
```

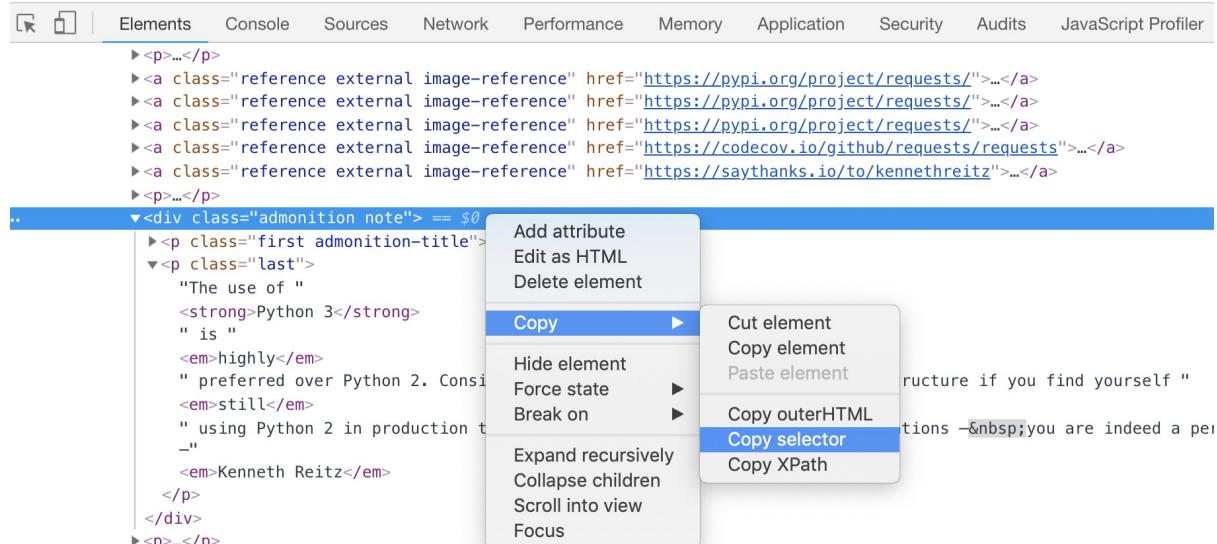
- 用 `#question` 选择器可以获得 `<p id="question">What color do you like?</p>`
- 用 `div` 选择器可以获得 `<div>I like blue.</div>`
- 用 `.red` 选择器可以获得 `<p class="paragraph red">I prefer red!</p>`

组合选择器

`div.red a` 匹配 `<div>` 元素里面的 `red` 类任意元素里面的 `<a>` 元素

快速获取选择器

打开网页，使用 Chrome 开发者工具，选择对应的元素，右键复制选择器



用 Python 解析 HTML

[Beautiful Soup](#) 是一个可以从 HTML 中提取数据的 Python 库。

假设我们获取了下面的页面并赋值给了变量 `page` (回忆 Python 的多行字符串语法)

```
page = """
<p id="question">What color do you like?</p>
<a href="/blue">I like blue.</a>
<p class="paragraph red">I prefer red!</p>
```

....

假设我们要获取 `id` 为 `question` 的元素的内容，以及链接的地址 `/blue`，使用 BeautifulSoup 可以这样解析：

```
from bs4 import BeautifulSoup
soup = BeautifulSoup(page) # 创建一个 soup 对象

matched_elements = soup.select('#question')
links = soup.select('a')
```

`select` 函数返回的结果是列表，回忆之前学的列表，我们可以用下标访问：

```
question = matched_elements[0].text # element.text 是
print(question)
```

也可以遍历它：

```
for a in links:
    print(a['href'])
```

爬虫实战

Cooking time: 30 mins active / 45 mins passive

Tasks For Instructors: 讲师讲解和演示该节内容，若相关内容认为需要教练辅助，请讲师灵活应变。

- 如果网站提供了 API，那么用 API 比从网页解析要更简单。
 - 如果 API 不能满足需求，再去考虑做网页爬虫。
 - 总结网址的规律，写成循环。

总结规律

例如贴吧的一个帖子，首页，第二页，第三页的网址分别如下：

<https://tieba.baidu.com/p/1433563243>

<https://tieba.baidu.com/p/1433563243?pn=2>

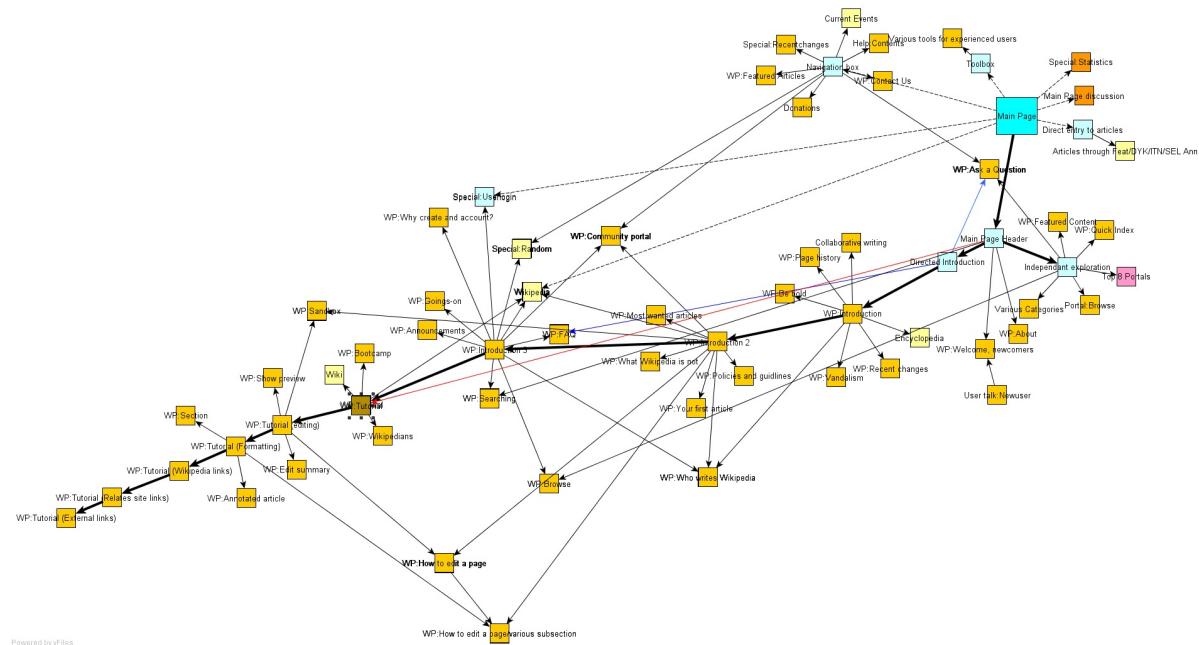
<https://tieba.baidu.com/p/143b3563243?pn=3>

那么只要我们知道一共有多少页，然后循环迭代 `pn=...`，就能遍历所有的页面。我们可以用 `range(from, to)` 构造一个迭代器，从 `from` 迭代到 `to - 1`：

```
for page in range(1, 4)
    url = f"https://tieba.baidu.com/p/1433563243?pn={page}
```

网站的结构

网站由一个或者多个网页组成，各个网页之间以超链连接起来，画成图会像这样



全站爬虫的构造

爬虫分为定向爬虫和全站爬虫。

定向爬虫只关注特定的内容，全站爬虫的目标却是爬下整个网站。

全站爬虫一般应用于搜索引擎、发现新内容、保存网络快照等目的。

为了实现爬虫的连续工作，我们可以把任务分解开来。

编程中如何分解任务？一种思维模式是：

- 可以先考虑简单的、特殊的情况。
- 把简单情况的代码写出来后，再把它改成更通用的情况 -- 例如改成一个函数。
- 把分解的函数组装起来，调试完成任务。

最简单的状况，网站只有一个页面的时候，我们回忆 `requests` 的用法，直接发出请求就可以了。

稍微复杂一点的情况，网站有一个入口页和多个次级页面，我们分析入口页的所有链接，得到次级页面的链接爬取汇总。

```
links = BeautifulSoup(page).select('a')
urls = []
for link in links:
    urls.append(link['href'])
```

再复杂一点，网站的次级页面上还有到更次一级的页面的链接，我们可以把上面的代码提炼成一个函数：

```
def extract_links(page):
    links = BeautifulSoup(page).select('a')
    urls = []
    for link in links:
        urls.append(link['href'])
    return urls
```

把请求网页的代码也组合到一起：

```
def get_page(url):
    page = requests.get(url)
    urls = extract_links(page.text)
    for sub_url in urls:
        get_page(sub_url)
```

这个函数调用了自己，叫做 **递归函数**。

上面的代码还有一些问题，如果子页面恰好链接到了父页面，它就停不下来了（实际上会出现一个栈溢出错误）。我们可以用集合来保证不会爬取相同的网页。

```
crawled = set()
def get_page(url):
    if url in crawled:
        return # 下次碰到相同的网页，它就不会再爬取了
    crawled.add(url)
    page = requests.get(url)
    urls = extract_links(page.text)
    for sub_url in urls:
        get_page(sub_url)
```


JSON 简介

JSON 是一种数据格式。

最准确又简洁的介绍：<https://json.org/json-zh.html>

处理 JSON 的库

标准的 JSON 库可以把 JSON 类型和 Python 类型相互转换

```
import json

json_string = json.dumps([1, 2, {"foo": true, "bar": None}])
json.loads(json_string)
```

获取 JSON 响应

```
response = requests.get("http://some.website.com/data.json")
response.json()
```

Web API

API -- 应用程序编程接口（Application Programming Interface）的缩写。

很多网站提供了基于 JSON 和 HTTP 的 API，你可以通过阅读文档，学习其 API 的使用方法。

爬虫的难点

- 网站需要登录：登录后才能看见网站的内容。
- 动态渲染的页面：越来越多的网站使用 JavaScript 动态构建页面，最终浏览器看到的页面和请求返回的页面不一样。
- 网站的反爬虫措施：根据访问特征例如 IP 地址限流。
- 爬取网页的速度太慢。

模拟登录

网站一般使用 Cookie 来记住登录状态。

当网站设置 Cookie 时，响应会加上 `Set-Cookie` 头。

爬虫请求通过加上 `Cookies` 头去告诉网站自己的登录状态。

模拟渲染

当静态页面没有包含。

Chrome Driver 是自动化的 Chrome 浏览器，我们可以通过编程的方式控制它访问页面并获得动态渲染后的内容。

Selenium 是自动化网页操作的框架。

结合 Selenium 和 Chrome Driver，我们可以完成动态网页的爬取。

模拟多 IP 访问

最常用的反爬虫措施就是检验请求的 IP。

爬虫方的应对办法就是使用代理，例如使用 Crawlera 的服务，每个请求都用不同的 IP。

提高性能

爬虫的大部分的时间都损耗在等待网络返回上面，使用异步框架如 `scrapy` 可以增加爬虫的吞吐量。使用多进程可以进一步充分利用 CPU 的多核心。

数据处理

本章节目标

1. 学习基本的数据处理方式
2. 展示爬取的数据

数据处理初步

Cooking time: 20 mins active / 30 mins passive

Tasks For Instructors: 讲师讲解和演示该节内容，若相关内容认为需要教练辅助，请讲师灵活应变。

我们收集的数据，往往不符合我们需要的格式，带有不同的噪音，看不出本质……需要对数据进行转换加工才能获得有用的信息。

爬虫获取的是原料，要经过数据处理的加工，最后才能获得产品。

Python 字符串

很多人会说最重要的数据结构是数组，但最常用的数据结构却是字符串。数据处理中使用最多的操作也是字符串操作。

按分隔符拆分字符串（split）

```
"a b c d".split(" ") # ["a", "b", "c", "d"]
```

用分隔符合并字符串（join）

```
" ".join(["a", "b", "c", "d"]) # "a b c d"
```

取子字符串

```
"this is good"[5:]
```

找到子字符串的位置

```
"foo bar baz".index('bar')
"foo".index("bar") # ValueError
```

子字符串判断：

```
"sub" in "string with sub"
str.startswith("prefix")
str.endswith("suffix")
```

类型转换

前面 Python 基础提到，数据有不同的类型，有时它的类型不是我们想要的。

例如从网页中提取出来的内容是字符串：

```
a = "1234"
b = "3.4"
c = "-12"
```

而我们想把它表示的数值加起来。这样做是不行的：

```
a + b + c # "12343.4-12"
```

注意对字符串做的 `+` 运算只是把它们串连起来。

为了进行数值计算，需要先把它们转换成数值类型。试试下面的代码：

```
int("123")
float("3.4")
```

转换成数值类型后，就能进行数值计算了：

```
a = "1234"
b = "3.4"
c = "-12"
int(a) + float(b) + float(c) # 1225.4
```

容器类型的处理

`list` , `dict` , `set` 等数据类型的处理。

处理一：排序

```
li.sort()                      # 纯粹的排序
li.sort(key=lambda x: x[2]) # 按照第二列排序
```

上面的排序使用了 `lambda` 函数。`lambda` 语法定义了一个匿名的函数，写法为

```
lambda 参数1, 参数2, ...: 表达式
```

表达式的值会作为函数的返回值。

处理二：过滤 - 去掉不需要的数据

```
result = []
for e in li:
    if ...
        result.append(e)
```

处理三：重塑 - 通过循环生成新的集合数据

```
value = []
for key, value in dict.items():
    result.append(value)
```

数据统计

假设我们有一个 `list` , 赋予了变量 `lst` , 里面包含若干项数据，我们要计算它的最大值：

```
max(lst)
```

最小值

```
min(lst)
```

和

```
sum = 0
for n in lst:
    sum += n
```

平均值

```
float(sum) / len(lst)
```

注：如果 `sum` 是整数相加得来，那它还是个整数对象（`int`），为了除法可以得到小数结果，需要先转换成浮点数。

理解数据

Cooking time: 20 mins active / 30 mins passive

Tasks For Coaches: 教练讲解和演示该节内容, 请教练灵活应变。

理解数据的几个层次:

- 形式理解: 了解数据文件的格式
- 逻辑理解: 理解数据的逻辑组织方式, 哪个字段代表什么意思
- 统计理解: 理解数据的分布, 是否异常, 是否可以解释等等, 有些可以通过绘图直观的表达

常见的数据格式:

- Excel 数据表格, 文档结构复杂, 需要用专门的库去解析。Pandas 集成了读写 Excel 文件的库。
- CSV 源文件容易阅读的数据表格, 首行是表头, 从第二行起每行是一条记录。
- JSON 如前述, 可以用 Python 自带的 JSON 库去处理。
- XML 类似于 HTML, 但是标签需要严格闭合, 也可以用 BeautifulSoup 处理。

应用示例（提取信息）

示例: 从免费词典资源提取一个简化版 (只含单词和定义) 的词典。

假设我们能下载到这么一个词典数据 <https://raw.githubusercontent.com/skywind3000/ECDICT/master/ecdict.csv>

下载:

```
import requests
response = requests.get("https://raw.githubusercontent.com/skywind3000/ECDICT/master/ecdict.csv")
with open("ecdict.csv", "w") as f:
    f.write(response.text)
```

它是一个 CSV 文件, 用编辑器或者表格处理软件打开, 观察文件的结构:

```
word,phonetic,definition,translation,pos,collins,oxford,tag,bnc,frq,exchange,detail,audio
hood,hud,,,"n. 罩; 风帽; (布质)面罩; 学位连领帽(表示学位种类)\nv. 覆盖; 用头巾包; 使(马、鹰等)戴头罩; 给...加罩\n[网络] 胡德; 兔帽; 引擎盖",,,,0,0,,,
```

由第一行表头信息可知, 单词 (word) 是第一列 (对应下标 0), 定义 (definition) 是第三列 (对应下标 2)。

Python 内建了处理 CSV 的库函数, 我们可以写一个循环去提取这两列:

```
# encoding: utf-8
import csv
result = {}
with open("ecdict.csv", "r") as f:
    for row in csv.reader(f):
        word = row[0]
        definition = row[2]
        result[word] = definition
```

然后把结果写进文件里:

```
with open("simplified.csv", "w") as f:
    for word, definition in result.items():
```

```
f.write(word)
f.write(":")
f.write(definition)
f.write("\n")
```

注：此示例参考了 <https://blog.plover.com/lang/ambiguous.html>

思考：如果这是一个英语到中文的词典，而我们想要得到中文到英文的解释，可以怎么做？

延伸阅读：网上搜索“倒排索引”，并尝试理解。

Pandas

Cooking time: 20 mins active / 30 mins passive

Tasks For Coaches: 教练讲解和演示该节内容, 请教练灵活应变。

Python 里进行数据操作 (manipulation) 和数据处理的函数库。

核心功能: DataFrame, Series, 图表, 统计等。

DataFrame

DataFrame 类似于数据库的表, 利用 DataFrame 可以对各种数据进行统一的处理。

Pandas 支持从多种数据格式创建 DataFrame, 例如 Excel 表格:

```
import pandas  
data_frame = pandas.read_excel("data.xls")
```

绘制曲线图

```
data_frame.plot()
```

绘制直方图

```
data_frame.hist("chart title")
```

Demo 练习

Cooking time: 45 mins active / 60 mins passive

Learning by Doing:

- **Tasks for Learners:** 请学员创建Python文件 `demo.py`，对下列Python语法进行练习。在练习过程中，请弄懂demo中每一行代码的意义。先学会阅读，接着模仿，再自己创造。
- **Tasks for Coaches:** 请教练对学员的练习提供一定的辅助，请控制好时间。

题目

假如我想把 xkcd.com 上的漫画图片爬下来 100 张。能写一个 Python 脚本完成吗？

【提示】数学家波利亚在[怎样解题](#)一书中讲到，解决一个问题的四个步骤：

- 理解题目：我们有什么，目标是什么？
- 拟定方案：通过什么思路去解决问题？条件和结果之间还有哪些缺失的环节？
- 执行方案：roll your sleeve，付诸行动。期间可能会需要修改方案中不合理的地方，请保持尝试和练习。
- 回顾：得到的结果是否符合预期？有什么收获？

工程是结合了整体和细节的技术。工程师通常先把总体规划分解成很多个细节，然后集中注意力解决每一个细节，然后再把细节组合成新的整体，重新理解和调整。

爬虫的任务可以大致划分为；遍历链接、分析页面、下载数据、输出结果。

1. 遍历链接

- 问题：页面的链接有什么规律？

访问网站点几下 "prev"，观察地址栏的变化发现链接的格式是 `https://xkcd.com/{数字}/`

- 问题：如何用 Python 生成所有的链接？（回顾[字符串拼接](#)）

```
for n in range(100):
    link = f"https://xkcd.com/{n}/"
```

2. 分析和下载

- 问题：如何获得页面或者图片？（回顾[Requests 的使用](#)）
- 问题：漫画图片的选择器是什么？（回顾[选择器](#)）

用 Chrome 开发者工具，复制选择器，得到 `#comic > img`

- 问题：如何用 Python 提取图片元素的 `src` 属性？如何补足成完整的图片地址？（回顾[BeautifulSoup 的使用](#)）

```
src = BeautifulSoup(page).select("#comic > img")[0]["src"]
image_link = f"http:{src}"
```

3. 输出结果

- 问题：结果格式是图片，如何保存二进制文件？（回顾[保存文件的方法](#)）
- 问题：保存多个文件如何命名？（回顾[循环语句](#)）

编程是一项需要注意力和细心的工作，大部分程序员也不能一次就把程序写对，除了阅读出错信息以外，往往还需要编写一些代码帮助了解程序的执行状况。

1. 问题：输出些什么让你知道程序正在运行？

使用 `print()` 打印程序在做的事情，做到了哪一步。

[Demo 参考代码](#)：你能读懂每行代码吗？

Learning by Thinking：

- 每个图片都有一个 `title` 属性，能否把它也顺便保存下来？
- 你有没有过一些想法，不知道如何去实现，掌握了简单的爬虫技术后，是否变得可能？（如果感觉掌握的知识技能还不足以支撑实现，可向教练请教或者组队完成）
- 你掌握了写程序解决问题的思路吗？是否从中掌握了新的思维方式？

大作业

Cooking time: 120 mins active / 150 mins passive

Learning by Doing:

- **Tasks for Learners:** 请学员创建Python文件 `project.py`, 实现自己人生中第一个数据工程作品!
- **Tasks for Coaches:** 请教练对学员的练习提供一定的辅助, 请控制好时间。

让我们再运用数学家波利亚在[怎样解题](#)一书中解决一个问题的四个步骤:

- 理解题目: 我们有什么, 目标是什么?
- 拟定方案: 通过什么思路去解决问题? 条件和结果之间还有哪些缺失的环节?
- 执行方案: roll your sleeve, 付诸行动。期间可能会需要修改方案中不合理的地方, 请保持尝试和练习。
- 回顾: 得到的结果是否符合预期? 有什么收获?

1. 【理解题目】你想用爬虫解决生活和工作中什么问题?

2. 【拟定方案】如何用编程思维中的「任务分解」把大问题分解成demo中的小问题?

3. 【执行方案】每一个小问题中需要的工具和知识点是什么? 是否都已经掌握?

4. 【回顾】

题外话: 跟自己聊聊天: 有没有一个小梦想埋藏在自己心中很多年, 却缺少勇气卖出哪一步? 可否用编程思维回答自己, 赋予自己信心与勇气, 探索人生更多的可能性!

分享 & 学习建议

分享

请与教练和小组的伙伴一起上台分享你的作品和参会体验，这是一天当中感动满满的时刻。

学习建议

1. 打好基础。

例如在线的统计学课程：[可汗学院公开课：统计学](#)
锻炼编程技能，可以从做题开始？例如：[Leet Code](#)

2. 问题导向。

可以学习的东西很多，多年编程经验的程序员遇到某些领域时，还是会有一堆黑盒子的感觉。一开始就弄清楚选什么科目进行学习是比较困难的。

可以在做项目的过程中，边做边学，遇到不懂的就查，是快速理解一个领域的办法。

3. 参与交流。

例如在 [GitHub](#) 上阅读开源代码，参与开源项目。

例如关注 [Hacker News](#)，了解业内动态。

参考书目

- [笨方法学 Python 3](#)
- [Python 网络数据收集](#)
- [Python 数据处理](#)