

Tutorial

djangogirls

目錄

介绍	1.1
安装	1.2
互联网是如何工作的？	1.3
命令行介绍	1.4
Python的安装	1.5
代码编辑器	1.6
Python 简介	1.7
Django 是什么？	1.8
Django安装	1.9
你的第一个Django 项目！	1.10
Django模型	1.11
Django管理	1.12
部署！	1.13
Django urls	1.14
Django视图 - 抓紧时间去创建！	1.15
HTML简介	1.16
Django ORM (查询集)	1.17
模板中的动态数据	1.18
Django模板	1.19
CSS - 让其更漂亮	1.20
模板扩展	1.21
扩展您的应用	1.22
Django表单	1.23
接下来呢？	1.24

Django Girls 教程

[gitter](#) [join chat](#)

这项工作是根据公共创意贡献署名国际协议4.0的要求完成的。若要查看本许可证的副本，请访问
<https://creativecommons.org/licenses/by-sa/4.0/>

介绍

你有没有觉得世界越来越多的是关于技术，而你已经在某种程度上被甩在了后面？你有没有想去做一个网站，但是却从未有过足够的动力去开始吗？你有没有想过软件世界太过复杂，即使你想做一些属于自己的东西？

好的，我们跟你讲一个好消息！编程没有看上去那么复杂，我们将给你展示它是多么的有趣。

本教程不会神奇地将你转变为程序员。如果你想变得擅长于此，你需要数月乃至数年的时间去学习和练习。但是我们想展示给你编程和创建网站没有看起来那么复杂。我们试着用深入浅出的方法解释，所以你不会觉得被技术吓到。

我们希望我们可以让你变得更爱技术，像我们一样！

在本教程里你会学到什么？

一旦你完成本教程，你会有一个简单可用的web应用：你自己的博客。我们会教你怎么把它放到网上，这样别人就能看到你的作品！

它将（或多或少）看起来像这个样子：

The screenshot shows a web browser window titled "Django Girls Blog" at the URL "127.0.0.1:8000". The page has a bright orange header with the "Django Girls" logo and navigation icons. Below the header, there are three blog post cards:

- Nulla facilisi** (published: 28-06-2014)
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras bibendum sapien interdum, posuere massa et, hendrerit leo. Nam commodo facilisis sapien vitae ornare. Integer eget purus posuere, vesti...
- Fusce vehicula feugiat augue eget consectetur** (published: 28-06-2014)
Pellentesque venenatis elit tortor, eu dictum magna accumsan in. Aenean vestibulum velit arcu, eleifend mattis purus suscipit a. Ut vitae pellentesque lorem. Integer lobortis orci in est molestie t...
- Duis quis imperdiet justo** (published: 28-06-2014)
Nulla ut metus luctus, tristique massa sit amet, venenatis eros. Aliquam hendrerit ligula nec viverra euismod. Vivamus eu sagittis diam, eget pharetra libero. Vestibulum ante ipsum primis in faucibus...

如果你独自学习这个教程而苦于没有一位能帮你答疑解惑的教练，我们给你提供交流的平台：[\[gitter\]](#) [\[join chat\]](#)
[1]。我们要求我们的教练以及之前的参与者时不时到那里去，帮助其他人参与本教程！不要担心什么，尽管大胆地提问吧！

好嘞，我们从头开始...

关于和贡献

这个教程由[DjangoGirls](#)维护。如果你发现本网有任何错误或想更新教程，请按照[贡献指导行动](#).

中文版的贡献者

本教程的中文版由下列人员参与翻译,按名字首字母排名:

- Abraham Kan (abrahamkan)
- adv_zxy
- BigBagBOOM
- c2j
- classicalmu (zhanyanqiao)
- cnbabylinke
- Javen (javen_liu)
- jeff kit (bbmyth)
- Joker Qyou (Joker_Qyou)

- Kate Chan (kateismex0)
- liuzhongjun
- meelo
- neal1991
- neoblackcap
- olasitarska
- pchnet
- rurong (defrur)
- Shit_JinP
- suusatoshigi (SuuSatoshigi)
- wangjiaxi
- wangshunping
- wly522
- yinhm
- yobo (yfrancesc)
- Yue Du (ifduyue)
- zhangyi2099
- Zhiheng Lin (onlyice0328)
- 刘奕聪(MrLYC)
- 芝加哥中文Meetup (chicagochinesemeetup)
- 陈艺虹(laceychen1993)

你愿意帮我们翻译将本教程至其他语言吗？

目前，所有的翻译都储存于crowdin.com的平台，网址是：

<https://crowdin.com/project/django-girls-tutorial>

如果你在crowdin的列表里没看到自己的语言，请以[新开issue](#)的方式通知我们，以便我们加进去。

如果你在家做这个练习

如果你在家做这个练习，而不是在一个 [Django Girls 活动](#) 中，你现在完全可以跳过这一章，直接阅读[互联网是如何工作的？一章](#)。

这是因为下面的这些步骤包含在整个教程当中，这只是一个额外的页面，把所有的安装说明收集在了一起。Django Girls活动有一个“开学礼”环节，活动中会一次性安装完成所有的软件，因而在之后的线下活动中就无需再进行安装，这对我们很有帮助。

如果你觉得很有帮助，你也可以跟着阅读这一章。但是如果你想在安装一堆软件以前就开始学习一些东西，最好跳过这一章，我们会稍后解释安装的部分。

祝你好运！

安装

在线下教程中，你会建立一个博客，有一些配置任务最好能够在之前完成，那样你就可以在那一天直接开始编程。

安装 Python

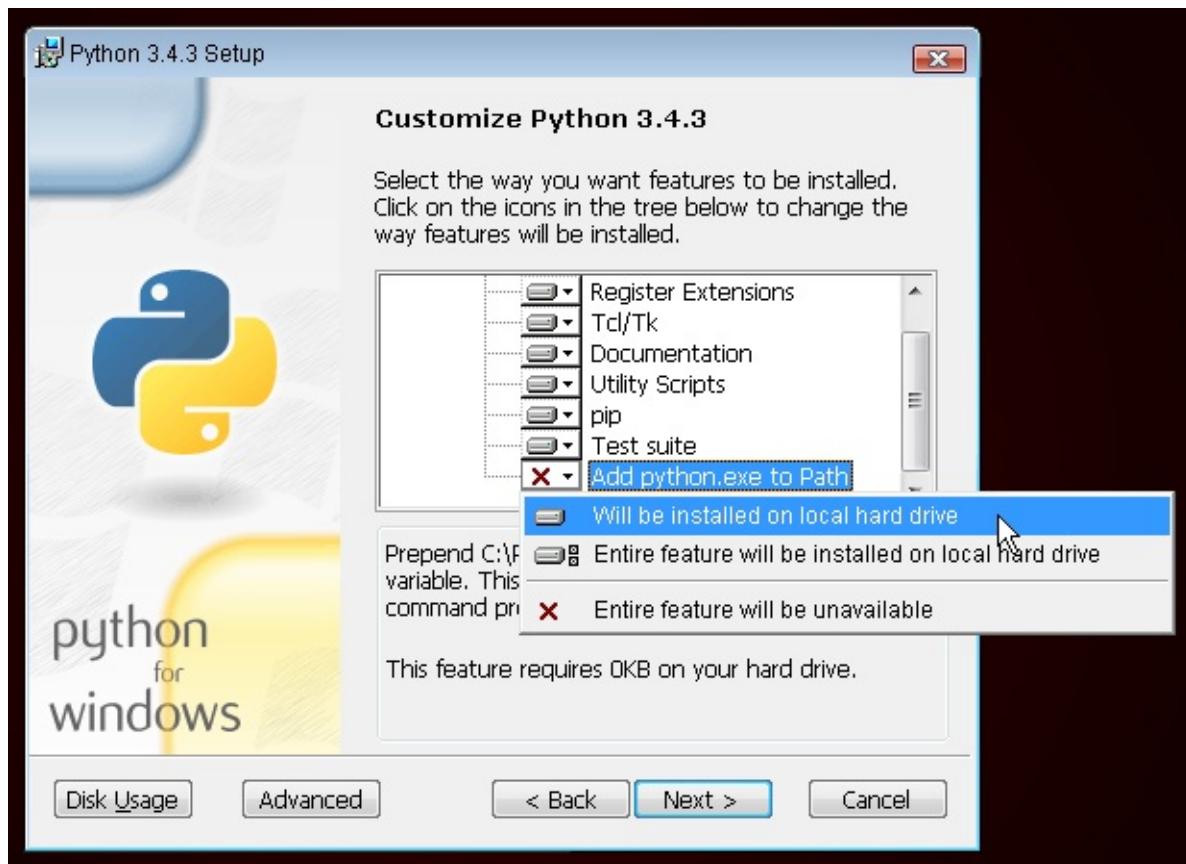
本节基于 Geek Girls Carrots (<https://github.com/ggcarrots/django-carrots>) 的教程

Django 是用 Python 写的。在 Django 中，我们需要使用 Python 语言去做所有的事情。让我们从安装开始！我们希望您能安装 Python 3.4，所以如果你有任何以前的版本，你将需要将其升级。

Windows 系统

您可以从 <https://www.python.org/downloads/release/python-343/> 网站上下载 Windows 版的 Python。下载 *.msi 文件完成之后，你应该运行它（双击它），并按照那里的指示。重要的是记住 Python 的安装位置的路径（目录）。它以后将会被需要！

需要注意一件事情：在安装向导的第二步，选中“Customize”，向下滚动并确保选中“Add python.exe to the Path”选项，如下所示：



Linux系统

很有可能你的系统已经默认安装了Python。要检查你是否安装了（并且它是哪一个版本），打开控制台，并键入以下命令：

```
$ python3 --version  
Python 3.4.3
```

如果你未曾安装过Python或者你想要一个不同的版本，你可以按如下所示安装它：

Debian 或 Ubuntu

在控制台中键入此命令：

```
$ sudo apt-get install python3.4
```

Fedora

在您的控制台中使用此命令：

```
$ sudo dnf install python3.4
```

openSUSE

在您的控制台中使用此命令：

```
$ sudo zypper install python3
```

OS X

你需要去到网站 <https://www.python.org/downloads/release/python-343/> 然后下载 Python 安装程序：

- 下载 Mac OS X 64-bit/32-bit installer 文件，
- 双击 `python-3.4.3-macosx10.6.pkg` 以运行安装程序。

验证安装成功，请打开 终端 应用，运行 `python3` 命令：

```
$ python3 --version
Python 3.4.3
```

如果您有任何疑问，或如果出了错，你根本不知道该怎么办下一步-请询问你的指导！有时事情都不顺利，这时寻求其他更多经验的人的帮助会更好。

配置 `virtualenv` 与安装 Django

本节部分内容基于 Geek Girls Carrots (<https://github.com/ggcarrots/django-carrots>) 教程而来。

本章的部分内容基于 [django-marcador 教程](#)，使用知识共享署名-4.0 国际许可协议许可。Django-marcador 教程的版权归 Markus Zapke-Gründemann 所有。

虚拟环境

在安装 Django 之前，我们会让你安装一个非常实用的工具，它可以让你计算机上的编码环境保持整洁。这个步骤可以跳过，但我们强烈建议你跟着做。从最佳实践设置开始将会在未来为你省去无数的烦恼！

所以，让我们创建一个 虚拟环境（也称为 `virtualenv`）。虚拟环境会以项目为单位将你的 Python/Django 安装隔离开。这意味着对一个网站的修改不会影响到你同时在开发的其他任何一个网站。优雅吧？

你需要做的就是找到您想创建 虚拟环境 的一个目录；比如您的主目录。在 Windows 上，它可能看起来像 `C:\Users\Name`（其中 `Name` 是您的登录名）。

在本教程中我们将使用您的 `home` 目录下的一个新目录 `djangogirls`：

```
mkdir djangogirls
cd djangogirls
```

我们将虚拟环境称为 `myvenv`。一般的命令格式是：

```
python3 -m venv myvenv
```

Windows 系统

若要创建新的 虚拟环境，您需要打开的控制台（我们在前几章里已经告诉过你了，还记得吗？），然后运行 `C:\Python34\python -m venv myvenv`。它看起来会像这样：

```
C:\Users\Name\djangogirls> C:\Python34\python -m venv myvenv
```

`C:\Python34` 是您之前安装Python的目录，`myvenv` 是您 虚拟环境 的名字。你可以使用其他任何名字，但请坚持使用小写，并不要使用空格、重音符号或特殊字符。始终保持名称短小是个好主意——你会大量引用它！

Linux 和 OS X

在 Linux 和 OS X 上创建的 虚拟环境 就和运行 `python3 -m venv myvenv` 一样简单。看起来像这样：

```
~/djangogirls$ python3 -m venv myvenv
```

`myvenv` 是您 `虚拟环境` 的名字。 您可以使用其他任何名称，但请始终使用小写以及不要有空格。 始终保持名称短小是个好主意，因为你会大量引用它！

注意：在Ubuntu 14.04上启动虚拟环境会报以下错误：

```
Error: Command '['/home/eddie/Slask/tmp/venv/bin/python3', '-Im', 'ensurepip', '--upgrade', '--default-pip']'
returned non-zero exit status 1
```

为了解决这个问题，请使用 `virtualenv` 命令。

```
~/djangogirls$ sudo apt-get install python-virtualenv
~/djangogirls$ virtualenv --python=python3.4 myvenv
```

使用虚拟环境

上面的命令将创建一个名为 `myvenv` 目录（或任何你选择的名字），其中包含我们的虚拟环境（基本上是一堆的目录和文件）。

Windows 系统

运行如下命令进入你的虚拟环境：

```
C:\Users\Name\djangogirls> myvenv\Scripts\activate
```

Linux 和 OS X

运行如下命令进入你的虚拟环境：

```
~/djangogirls$ source myvenv/bin/activate
```

记住要将 `myvenv` 替换成你选择的 `虚拟环境` 的名字！

注：有时 `source` 可能不可用。在这些情况下试着做下面这些事情：

```
~/djangogirls$ . myvenv/bin/activate
```

当你看到在您的控制台中有如下提示就知道你已经进入 `虚拟环境`：

```
(myvenv) C:\Users\Name\djangogirls>
```

或：

```
(myvenv) ~/djangogirls$
```

注意 `(myvenv)` 前缀的出现！

当在一个虚拟的环境中工作时，`python` 将自动指向正确的版本，因此您可以使用 `python` 代替 `python3`。

好的现在所有重要的依赖关系已经就位。最后，我们可以安装 Django！

安装 Django

既然你有了 虚拟环境 ，您可以使用 pip 安装 Django 。在控制台中，运行 `pip install django==1.8` （注意我们使用双等号：`==`）。

```
(myvenv) ~$ pip install django==1.8
Downloading/unpacking django==1.8
  Installing collected packages: django
    Successfully installed django
Cleaning up...
```

在 Windows 上

如果你在 Windows 平台上调用 pip 时得到一个错误，请检查是否您的项目的路径名是否包含空格、重音符号或特殊字符（如：`C:\Users\User Name\django\girls`）。若的确如此，请尝试移动它到另外一个没有空格、重音符号或特殊字符的目录，（例如：`C:\django\girls`）。在移动之后，请重试上面的命令。

在 Linux 上

如果你在 Ubuntu 12.04 上得到一个错误，请运行 `pip python -m pip install -U --force-reinstall pip` 来修复虚拟环境中的 pip 安装。

就是这样！你现在（终于）准备好创建一个 Django 应用程序！

安装一个代码编辑器

有很多不同的编辑器，通常根据个人偏好选择。大部分 Python 程序员使用像 PyCharm 这样复杂但是功能强大的 IDE（集成开发环境）。然而，这可能不太适合初学者。我们建议使用同样强大但是更为简单的编辑器。

下面是我们的建议，但是你可以随时咨询你的教练。那样会更容易得到他们的帮助。

Gedit 文本编辑器

Gedit 是开源、免费的编辑器，支持所有操作系统。

[在这里下载](#)

Sublime Text 3 编辑器

Sublime Text 是一个很受欢迎的、免费试用的编辑器。它很容易安装和使用，并且支持所有操作系统。

[在这里下载](#)

Atom 编辑器

Atom 是 GitHub 最新发布的代码编辑器。它开源、免费并且容易安装和使用。它可用于 Windows、OSX 和 Linux。

[在这里下载](#)

我们为什么要安装代码编辑器？

你可能会疑惑为什么我们要安装代码编辑器软件，而不是使用如 Word 或记事本这样的软件。

第一个理由是代码应该是纯文本，如 Word 和 Textedit 的程序实际上不能产生纯文本，它们产生富文本（含有字体和格式），使用自定义的格式，如 [RTF \(Rich Text Format\)](#)。

第二个原因是代码编辑器专长于编辑代码，因此它们可以提供一些有用的功能，比如代码高亮，比如自动闭合引用号。

以上这些我们将在实践中具体体会。很快你将会觉得，似乎过时的老文本编辑器正是最得心应手的工具！:)

安装 Git

Windows 系统

你可以从 git-scm.com 下载Git。你可以在所有的安装步骤中点击"next next next"，除了第5步"Adjusting your PATH environment"，需要选择"Run Git and associated Unix tools from the Windows command-line"(底部的选项)。除此之外，默认值都没有问题。签出时使用 Windows 风格的换行符，提交时使用 Unix 风格的换行符，这样比较好。

MacOS 系统

从git-scm.com下载Git,根据说明操作。

Linux 系统

如果git还没有被安装的话，你可以从你的软件包管理器中下载git,请试试下面命令:

```
sudo apt-get install git  
# or  
sudo yum install git  
# or  
sudo zypper install git
```

创建一个 Github 账户

前往 [GitHub.com](https://github.com) 并注册一个新的账号。

创建一个 PythonAnywhere 帐户

下一步，在PythonAnywhere注册一个“Beginner”账户。

- www.pythonanywhere.com

注意 在这里选择的用户名会出现在博客链接的地址当中 `yourusername.pythonanywhere.com`，因此最好选择你自己的绰号或者是与博客内容相关的名字。

开始阅读

恭喜你，你配置好了全部的软件，可以准备正式开始了！如果你仍然在线下活动之前有一些时间，阅读一些入门章节会有一些帮助：

- [互联网是如何工作的？](#)
- [命令行介绍](#)
- [Python 简介](#)
- [Django 是什么？](#)

互联网是如何工作的？

本章内容衍生自Jessica McKellar的演讲“互联网是怎么工作的”(<http://web.mit.edu/jesstess/www/>)。

我们猜你每天在使用互联网。但是当你在浏览器里输入一个像 [https://django.org](https://.djangoproject.org) 的地址并按 回车键 的时候，你真的知道背后发生了什么吗？

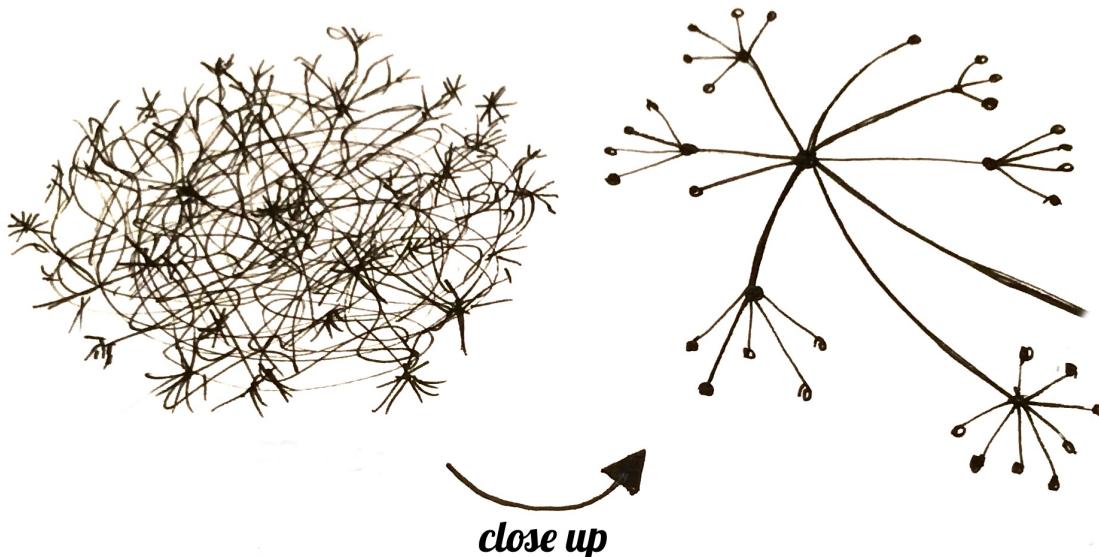
你需要了解的第一件事是一个网站只是一堆保存在硬盘上的文件。就像你的电影、音乐或图片一样。然而，网站的唯一的不同之处是：网站包含一种称为 HTML 的代码。

如果你不熟悉编程，一开始你会很难理解HTML。你的浏览器(如Chrome、Safari、Firefox等)却很喜欢它。Web浏览器懂得这些代码，遵循它的指令并如你所想的那样展示这些文件。

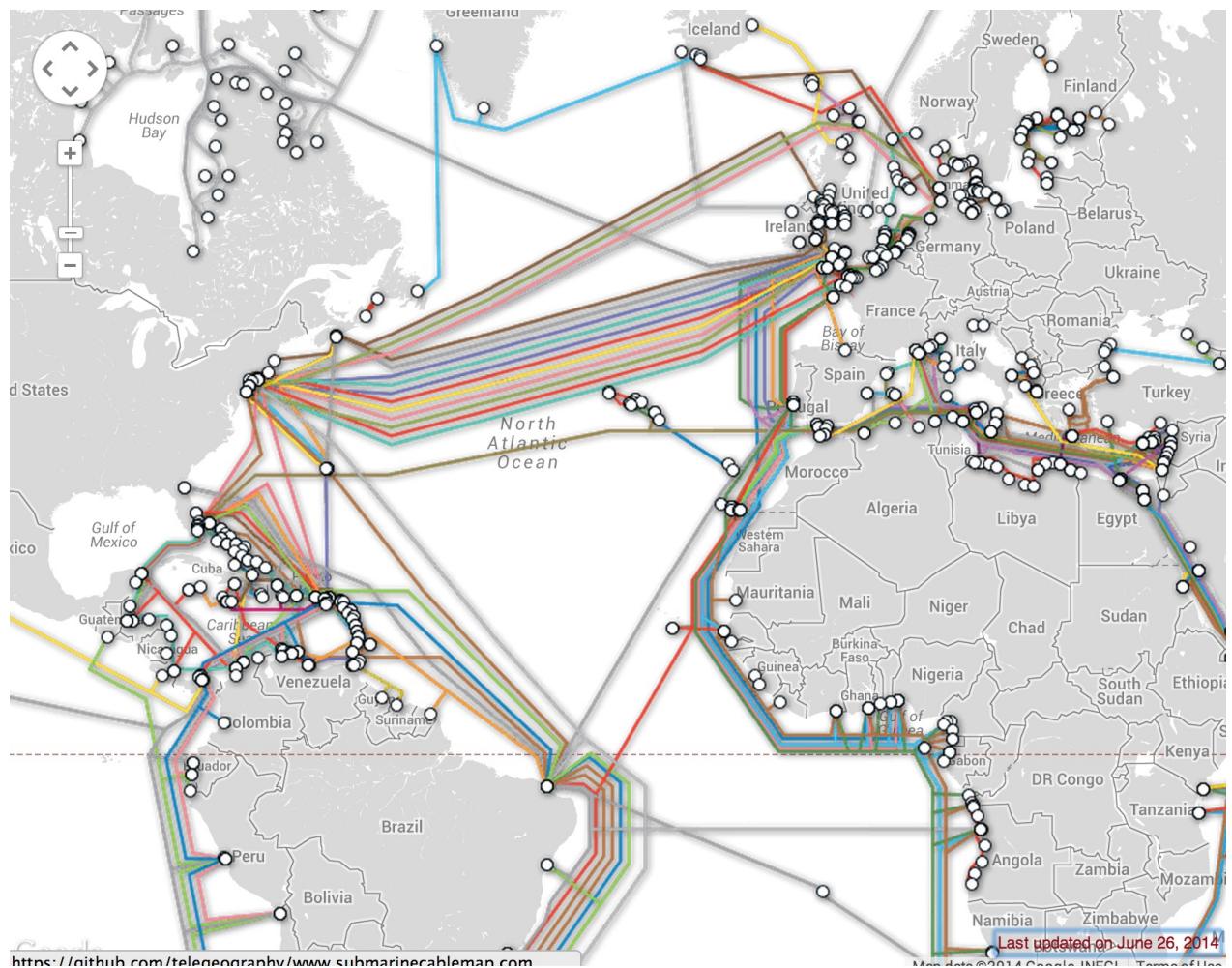
就像我们对待每个电脑文件一样，我们需要把HTML文件存储在硬盘的某个位置。对于互联网，我们使用特定而功能强大的电脑，我们称之为服务器。它们没有屏幕、鼠标或者键盘，因为它们的主要目的是存储数据，并用它来提供服务。这就是为什么它们被称作服务器—因为他们用数据服务你。

好了，但是你想知道互联网看起来是什么样子的，对吗？

我们给你画了一幅画！它看起来像这样：

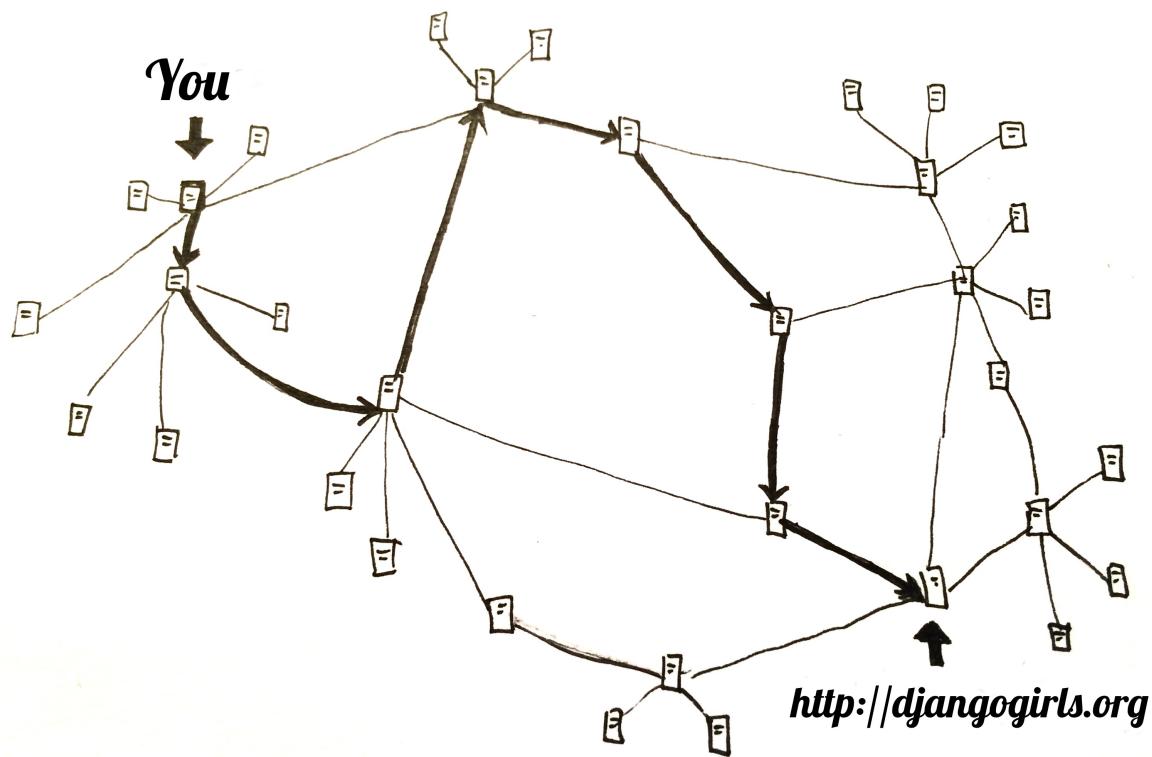


看起来很糟糕，对吗？事实上，它是一个由互相连通的机器(上面提到的服务器)组成的网络。数以十万计的机器！很多，很多数以公里长的电缆分布在全世界！你能访问一个海底电缆地图网站(<http://submarinecablemap.com/>)来看这个网络有多么复杂。这是一个网站上的截图：



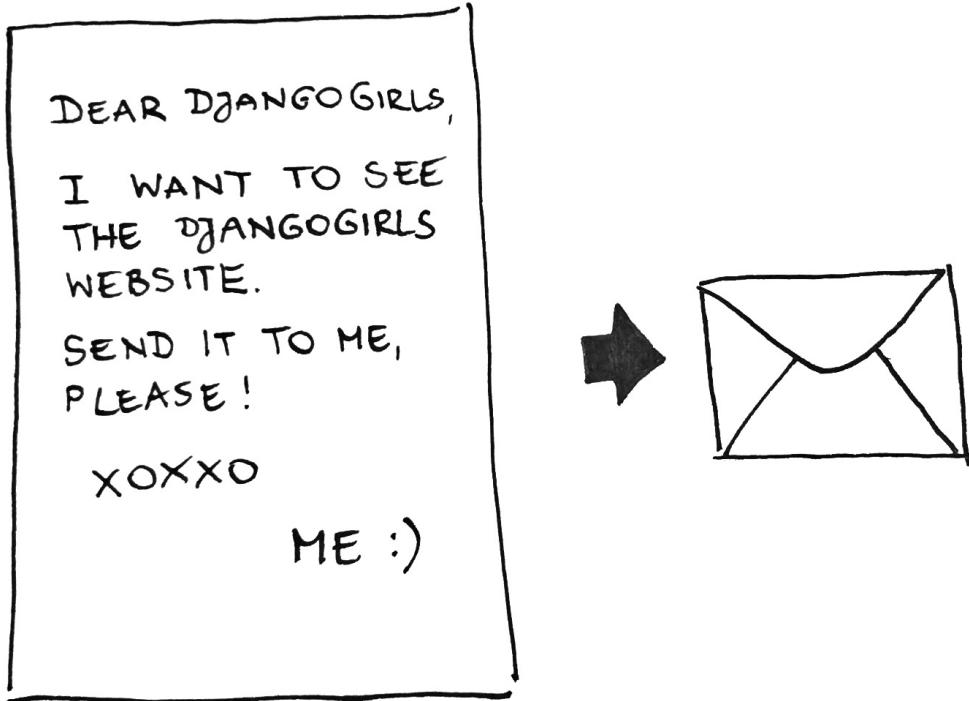
它很迷人，对吧？但是很明显，我们不太可能用电缆去连接任何两台上网的电脑。我们需要将请求通过很多很多不同的机器传递，以便连接一台机器（比如存储<https://djangogirls.org>的机器）。

它看起来像这样：



想象一下当你键入 <https://djangogirls.org>，你会发送一封信说：“亲爱的Django Girls，我想看看djangogirls.org网站，请将它发给我！”

你的信件去了离你最近的邮局。然后它去离你的收件人稍近一点的邮局，然后再去另一个，以此类推地到达它的目的地。唯一的事情是，如果你将许多信件(数据包)发送到同一个地方，他们可以通过完全不同邮政局(路由器)。这取决于每个办公室的分布情况。



是的，就这么简单。你发送信息，然后期望一些回复。当然，你使用的不是纸笔，而是比特数据，但是思想是一样的！

我们使用的地址叫做IP地址，这种地址却不包含街道、城市、邮编和城市。你的电脑首先要求DNS（域名服务器）去解析 djangogirls.org 成为IP地址。他工作起来有点像老式的电话簿，在那里你可以找到你想联系的人的名字，并且找到他们的电话号码和地址。

当你发送一封信时，它需要确切的功能以便正确的传递：地址，邮票等等。你使用一种接收者同样能够理解的语言，对吗？你送到网站的数据包也类似如此。我们使用一个名为 HTTP (Hypertext Transfer Protocol：超文本传输协议) 的协议。

所以，基本上，当你有一个网站你需要有一台服务器用于网站的托管。当服务器接收到一个访问请求（一封信），它把网站内容发送回去（回信）。

既然这是Django教程，你会问Django做什么。当你发送一个响应时，你通常不会发送同样的东西给每一个人。如果你信件的内容是个性化的必然更好，尤其是对于那个刚刚给你写信的人，对吗？Django帮助你创建这些个性化，有趣的信件：）。

废话少说，抓紧时间创造！

命令行界面简介

哈，这是令人兴奋，不是吗？:) 仅几分钟内你会写你第一行代码

让我们把你介绍给你的第一个新朋友：命令行！

以下步骤将显示你如何使用所有的黑客都使用的黑色窗口。它可能刚开始看上去有点吓人，但事实上它只是等待你的命令提示符。

注意 请注意，在本书中我们使用术语“目录”和“文件夹”完全可以互换，它们指的是同一个东西。

什么是命令行？

这个窗口通常被称为 命令行 或 命令行界面，是一个基于文本的用来查看、处理、和操作您的计算机上的文件的应用程序。就像 Windows 资源管理器或 Mac 上的 Finder，但没有图形界面。命令行的其他名字是： cmd，CLI、提示符、控制台 或 终端。

打开命令行界面

为了进行一些实验，我们需要首先打开我们的命令行界面。

Windows 系统

转到开始菜单 → 所有程序 → 附件 → 命令提示符。

Mac OS X 系统

应用程序 → 实用工具 → 终端。

Linux 系统

它可能是根据应用程序 → 附件 → 终端，但这可能取决于您的版本系统。如果它不存在，只需谷歌一下 :)

提示符

你现在应该看到一个白色或黑色的窗口，正等待着你的命令。

如果你是在 Mac 或 Linux 上，您可能看到 \$，就像这样：

```
$
```

在 Windows 上，它是一个 > 标志，像这样：

```
>
```

每个命令将前缀由这个符号和一个空格组成，但您不必键入它。您的计算机将为您完成这个 :)

温馨提示： 你的提示符前可能是一些类似 c:\Users\ola > 或 Ola-MacBook-Air:~ola\$ 这样的符号，这都是正确的。在本教程中我们将只是简化它到最低限度。

你的第一个命令（耶！）

让我们从简单的东西开始。键入以下命令：

```
$ whoami
```

或

```
> whoami
```

然后键入 `enter` 。这是我们的结果：

```
$ whoami  
olasitarska
```

正如你所看到的计算机刚刚打印了你的用户名。棒吧，呵呵?:)

尝试键入每个命令，请不要复制粘贴。你会通过这种方式记得更多东西！

基础知识

每个操作系统都有一组略有不同的适用于相应命令行的命令，所以请务必按照您的操作系统说明来做。我们试试这个，好吗？

当前目录：

知道身在何处让人高兴，对不对？让我们看看。输入命令并键入 `enter`：

```
$ pwd  
/Users/olasitarska
```

如果你在 Windows 操作系统上：

```
> cd  
C:\Users\olasitarska
```

在你的机器上你可能会看见类似的东西。一旦你打开命令行工具你通常开始于用户主目录。

注：'pwd' 代表 '打印工作目录'。

列出文件和目录

那么它是什么？它一定很酷，找出。让我们看看：

```
$ ls  
Applications  
Desktop  
Downloads  
Music  
...
```

Windows 系统

```
> dir  
Directory of C:\Users\olasitarska  
05/08/2014 07:28 PM <DIR> Applications  
05/08/2014 07:28 PM <DIR> Desktop  
05/08/2014 07:28 PM <DIR> Downloads  
05/08/2014 07:28 PM <DIR> Music  
...
```

更改当前目录

现在，让我们去我们桌面目录：

```
$ cd Desktop
```

Windows系统

```
> cd Desktop
```

请检查它是否真的发生改变了：

```
$ pwd  
/Users/olasitarska/Desktop
```

Windows系统

```
> cd  
C:\Users\olasitarska\Desktop
```

它在这儿！

进阶提示：如果你键入 `cd D`，然后在点击 `tab` 键，命令行将自动填充剩下的名称以便你可以更快地导航。如果有多个文件夹以“D”开头，按下 `tab` 按钮两次以获取选项的列表。

创建目录

在你的桌面上创建一个`practice`目录怎么样？你可以这样做：

```
$ mkdir practice
```

Windows系统

```
> mkdir practice
```

这小小的命令将在桌面上为你创建名为 `practice` 的目录。你可以查看桌面文件夹或者通过运行 `ls` 或 `dir` 命令检查它是否存在！试试：)

进阶提示：如果你不想要一遍又一遍地键入相同的命令，试着按 `向上箭头` 和 `向下箭头` 下循环查看最近你使用过的命令。

练习

给你一个小挑战：在您新创建的 `practice` 目录下创建一个名为 `test` 的目录。使用 `cd` 和 `mkdir` 命令。

解决方法

```
$ cd practice  
$ mkdir test  
$ ls  
test
```

Windows系统

```
> cd practice  
> mkdir test  
> dir  
05/08/2014 07:28 PM <DIR>      test
```

恭喜！:)

清理

我们不想留下一个烂摊子，所以让我们删除所有的东西，直到达到目的。

首先，我们需要回到桌面：

```
$ cd ..
```

Windows系统

```
> cd ..
```

使用 `..` 和 `cd` 命令将改变你的当前目录到父目录（包含当前目录的目录）。

检查你在哪里：

```
$ pwd  
/Users/olasitarska/Desktop
```

Windows系统

```
> cd  
C:\Users\olasitarska\Desktop
```

现在删除 `practice` 目录：

注意：删除文件使用 `del`，`rmdir` 或 `rm` 后便不能挽回，意思就是已删除的文件会一去不复返了！因此，要十分小心地使用此命令。

```
$ rm -r practice
```

Windows系统

```
> rmdir /S practice  
practice, Are you sure <Y/N>? Y
```

完成了！为了肯定它真的已经删除了，让我们检查一下：

```
$ ls
```

Windows系统

```
> dir
```

退出

暂时就到这里！你可以安全的关闭命令行。让我们以黑客的方式，好吗？:)

```
$ exit
```

Windows系统

```
> exit
```

很酷吧？：）

摘要

这里是一些有用的命令摘要：

命令 (Windows)	命令 (Mac OS / Linux)	说明：	示例：
exit	exit	关闭窗口	exit
cd	cd	更改目录	cd test
dir	ls	列出的目录文件	dir
copy	cp	复制文件	copy c:\test\test.txt c:\windows\test.txt
move	mv	移动文件	move c:\test\test.txt c:\windows\test.txt
mkdir	mkdir	创建一个新目录	mkdir testdirectory
del	rm	删除文本文件目录	del c:\test\test.txt

这些不过是你在你的命令行中可以运行的少量命令，但你今天不会用到比这里更多的了。

如果你好奇，ss64.com 包含用于所有操作系统命令的完整引用。

准备好了吗？

让我们深入到 Python ！

让我们从学习 Python 开始

我们终于要开始了！

但首先，让我们告诉你什么是 Python。Python 是一种非常流行的编程语言，可以用来创建网站、游戏、科学软件、图形和很多很多其他的东西。

Python 起源于 1980 年代后期，该语言的重要目标是源代码的人类可读性（而不是机器！）。这就是为什么它看起来比其他编程语言简单得多。这使得它易于学习，但不要担心，Python 也是真的很强大！

Python 的安装

需要说明的是，如果你已经熟悉 Python 的安装步骤，那你就无需看这章，直接跳到下一章好了。

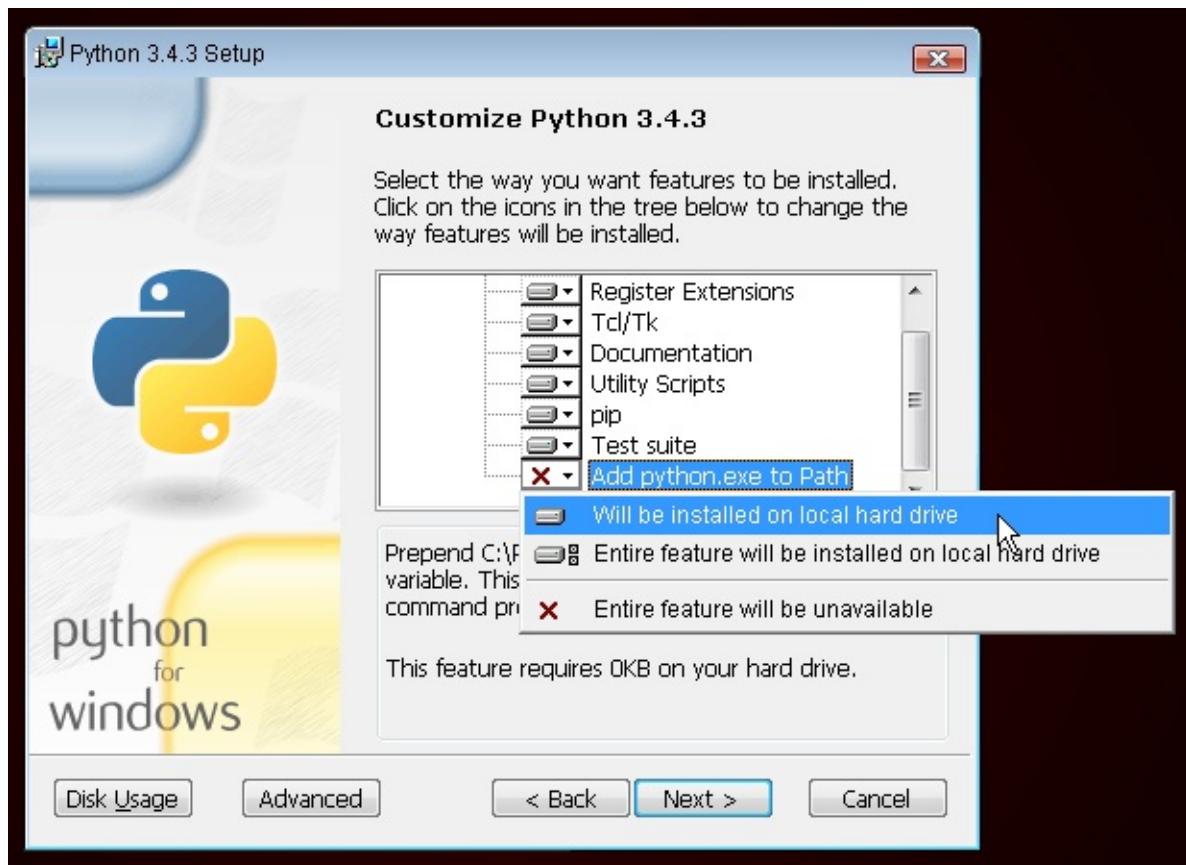
本节基于 Geek Girls Carrots (<https://github.com/ggcarrots/django-carrots>) 的教程

Django 是用 Python 写的。在 Django 中，我们需要使用 Python 语言去做所有的事情。让我们从安装开始！我们希望您能安装 Python 3.4，所以如果你有任何以前的版本，你将需要将其升级。

Windows 系统

您可以从 <https://www.python.org/downloads/release/python-343/> 网站上下载 Windows 版的 Python。下载 *.msi 文件完成之后，你应该运行它（双击它），并按照那里的指示。重要的是记住 Python 的安装位置的路径（目录）。它以后将会被需要！

需要注意一件事情：在安装向导的第二步，选中“Customize”，向下滚动并确保选中“Add python.exe to the Path”选项，如下所示：



Linux系统

很有可能你的系统已经默认安装了Python。要检查你是否安装了（并且它是哪一个版本），打开控制台，并键入以下命令：

```
$ python3 --version  
Python 3.4.3
```

如果你未曾安装过Python或者你想要一个不同的版本，你可以按如下所示安装它：

Debian 或 Ubuntu

在控制台中键入此命令：

```
$ sudo apt-get install python3.4
```

Fedora

在您的控制台中使用此命令：

```
$ sudo dnf install python3.4
```

openSUSE

在您的控制台中使用此命令：

```
$ sudo zypper install python3
```

OS X

你需要去到网站 <https://www.python.org/downloads/release/python-343/> 然后下载 Python 安装程序：

- 下载 *Mac OS X 64-bit/32-bit installer* 文件，
- 双击 *python-3.4.3-macosx10.6.pkg* 以运行安装程序。

验证安装成功，请打开 终端 应用，运行 `python3` 命令：

```
$ python3 --version  
Python 3.4.3
```

如果您有任何疑问，或如果出了错，你根本不知道该怎么办下一步-请询问你的指导！有时事情都不顺利，这时寻求其他更多经验的人的帮助会更好。

代码编辑器

你马上就要写下第一行代码，现在该下载一个代码编辑器了！

注意在之前章节你可能已经完成了这一步，如果那样的话，你可以直接进入下一章节。

有很多不同的编辑器，通常根据个人偏好选择。大部分 Python 程序员使用像 PyCharm 这样复杂但是功能强大的 IDE (集成开发环境)。然而，这可能不太适合初学者。我们建议使用同样强大但是更为简单的编辑器。

下面是我们的建议，但是你可以随时咨询你的教练。那样会更容易得到他们的帮助。

Gedit 文本编辑器

Gedit 是开源、免费的编辑器，支持所有操作系统。

[在这里下载](#)

Sublime Text 3 编辑器

Sublime Text 是一个很受欢迎的、免费试用的编辑器。它很容易安装和使用，并且支持所有操作系统。

[在这里下载](#)

Atom 编辑器

Atom 是 GitHub 最新发布的代码编辑器。它开源、免费并且容易安装和使用。它可用于 Windows、OSX 和 Linux。

[在这里下载](#)

我们为什么要安装代码编辑器？

你可能会疑惑为什么我们要安装代码编辑器软件，而不是使用如 Word 或记事本这样的软件。

第一个理由是代码应该是纯文本，如 Word 和 Textedit 的程序实际上不能产生纯文本，它们产生富文本 (含有字体和格式)，使用自定义的格式，如 [RTF \(Rich Text Format\)](#)。

第二个原因是代码编辑器专长于编辑代码，因此它们可以提供一些有用的功能，比如代码高亮，比如自动闭合引用号。

以上这些我们将在实践中具体体会。很快你将会觉得，似乎过时的老文本编辑器正是最得心应手的工具！:)

Python 简介

本章的部分内容基于 Geek Girls Carrots (<https://github.com/ggcarrots/django-carrots>) 的教程。

让我们现在就开始写代码！

Python 提示符

要玩转 Python，首先需要打开您的计算机上的命令行。这一点，你应该已经掌握了——你在 [命令行入门](#) 这一章已经学习过。

如果你准备好了，那么请按照以下说明进行操作。

在 Windows 下输入 `python` 或在 Mac OS/Linux 上输入 `python3` 并敲下 `回车键`。

```
$ python3
Python 3.4.3 (...)
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

第一条 Python 命令！

完成运行 Python 命令后，提示符变为 `>>>`。这就意味着，现在我们只能使用符合 Python 语法的命令。你不必键入 `>>>` —— Python 会为你做的。

如果你想在任何时候退出 Python 控制台，只需要键入 `exit()` 或者在 Windows 下使用快捷键 `Ctrl+Z`，在 Mac/Linux 下使用 `Ctrl+D` 快捷键。这时候你就不会再看到 `>>>`。

但是现在，我们不需要退出 Python 控制台。我们往下要继续了解 Python。让我们从最简单的开始。例如，试着输入一些数学运算，如 `2 + 3` 并敲下 `回车`。

```
>>> 2 + 3
5
```

好的！看到答案是如何显示出来的吗？Python 懂数学！你可以试试其他命令：`- 4 * 5 - 5 - 1 - 40 / 2`

好好的玩一会儿，然后回到这里 :)

正如你所看到的，Python 是一个不错的计算器。如果你想知道你还能用它做什么……

字符串

比试试你的名字？把你的名字用引号括起来，就像这样：

```
>>> "Ola"
'Ola'
```

现在你创建了第一个字符串！字符串是一个可以由计算机处理的字符序列。该字符串的开始和结束位置必须用同一个字符标志。可以由单引号（'）或双引号（"）来包裹（两者是没有区别的！），让 Python 知道，这中间是一个字符串。

字符串可以用加号连接在一起。像这样：

```
>>> "Hi there " + "Ola"
'Hi there Ola'
```

你也可以将字符串与数字相乘：

```
>>> "Ola" * 3
'OlaOlaOla'
```

如果你的字符串中需要表示一个单引号，你有两种方法：

用双引号来包裹字符串：

```
>>> "Runnin' down the hill"
"Runnin' down the hill"
```

或者使用反斜杠（`）来转义单引号

```
>>> 'Runnin\' down the hill'
"Runnin' down the hill"
```

表现得不错，是吧？如果想把字符串变成大写，只需要输入：

```
>>> "Ola".upper()
'OLA'
```

你刚刚在你的字符串上使用了 `upper` 函数！像 (`upper()`) 这样的函数是告诉 Python 运行某个给定对象如 (`"Ola"`) 上的一些列指令。

如果你想获取你名字里字母的个数，这里也有一个函数！

```
>>> len("Ola")
3
```

不知道为什么有时候你调用函数会在字符串尾部是用 . (就像 `"Ola".upper()`) 而有时候你会在一开始就调用函数，然后将字符串置于括号中吗？好吧，在某些情况下，函数属于对象，就像 `upper()`，它只可以被应用于字符串上。在这种情况下，我们将此函数称为方法 (**method**)。其他时候，函数不隶属于任何一个特定的对象，可以用在不同类型的对象上，例如 `len()`。这就是为什么我们使用 `"Ola"` 作为 `len` 函数的一个参数。

摘要

好的，关于字符串就学到这里。现在你已经学习到了以下这些内容：

- 提示符-键入命令行（代码）到 Python 提示符中，将在 Python 中得到回答。
- 数字和字符串-在 Python 里数字用于数学运算，而字符串被用作文本对象。
- 运算符 - 例如 + 和 *，将对赋值进行结合，并产生新的值。
- 函数-例如 `upper()` 和 `len()`，对对象执行操作。

这些都是学习每种编程语言中最基本的东西。想学点更难的东东了吗？我们敢打赌你想！

错误

让我们试试一些新东西，我们能够像得到名称长度一样的方式来得到一个数字的长度吗？输入 `len(304023)` 然后敲下 回车键：

```
>>> len(304023)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: object of type 'int' has no len()
```

我们得到了第一个错误！它说“int”（整数）的对象类型没有长度。所以我们现在该做什么呢？或许我们能够把我们的数字写成字符串？字符串有长度，对吗？

```
>>> len(str(304023))
6
```

它成功了！我们将 `str` 函数插入到 `len` 函数内。`str` 将所有东西转换成字符串。

- `str` 函数将对象转换成字符串
- `int` 函数将对象转换为整数

重要提示：我们能够将所有的数字都转换成文本，但我们不能将所有的文本转换成数字，否则 `int("hello")` 究竟是什么意思？

变量

变量是编程中的一个重要概念。变量只不过是一个待会儿你可以使用的一个东西的名称。程序员是用这些变量去存储数据，让他们的代码变得更具可读性，所以他们不需要一直记住东西是什么。

假设我们想创建一个新变量，叫做 `name`：

```
>>> name = "Ola"
```

你看到了吗？这很容易！原因很简单：名字等于Ola。

正如你所注意到的，你的程序不会像以前那样返回任何东西。那么我们怎么知道这个变量确实存在？简单键入 `name` 然后按下回车。

```
>>> name
'Ola'
```

耶！你的第一个变量:)！你随时可以改变它指代的内容：

```
>>> name = "Sonja"
>>> name
'Sonja'
```

你还可以这样使用函数：

```
>>> len(name)
5
```

太棒了，对吗？当然，变量可以是任何东西，比如数字！试试这个：

```
>>> a = 4
>>> b = 6
>>> a * b
24
```

但是如果我们将名字写错呢？你能猜出来会发生什么吗？试试吧！

```
>>> city = "Tokyo"
>>> ctiy
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'ctiy' is not defined
```

一个错误！正如你所见，Python有不同种类的错误，这一种称作**NameError**。Python将会在你尝试使用未定义的变量时产生这种错误。如果你以后再次遇到这种错误，检查你的代码，看看你是不是错误输入了任何名称。

玩一会儿，然后看看你能做什么！

打印功能

试试这个：

```
>>> name = 'Maria'  
>>> name  
'Maria'  
>>> print(name)  
Maria
```

当你键入 `name`，Python 解释器会打印变量`'name'`表示的字符串，单引号包裹的：`'Maria'`。当你调用 `print(name)` 的时候，Python 将会“打印”内容到屏幕上，没有引号，更加整洁。

我们稍后会看到的，当我们想打印一些东西，或者想打印一些多行的东西，`print()` 是一个对我们很有用的函数。

列表

除了字符串和整数，Python提供了各种不同类型的对象。现在我们要介绍一个叫做列表的东西。列表和你想的一样：一个容纳着其他对象的列表对象：）

继续，创建一个列表：

```
>>> []  
[]
```

是的，这个列表是空的。并不是十分有用，对吗？让我们创建一个彩票号码的列表。我们不想总是重复我们的工作，所以我们也将它置于一个变量中：

```
>>> lottery = [3, 42, 12, 19, 30, 59]
```

好吧，我们有了一个列表！我们能为它做什么呢？让我们看一看在列表中有多少个彩票数字。你知道什么方法什么函数你可以使用的吗？你已经知道了！

```
>>> len(lottery)  
6
```

是的！`len()` 可以给你列表中对象的个数。很方便，对吗？也许我们可以将它排序：

```
>>> lottery.sort()
```

这不会返回任何东西，它仅仅改变了数字出现在列表里的顺序。让我们再一次把它打印出来，看看发生了什么：

```
>>> print(lottery)  
[3, 12, 19, 30, 42, 59]
```

正如你所看到的，你的列表里的数字现在从最小到最大排序。祝贺！

也许我们想要将顺序倒序呢？让我们开始做吧！

```
>>> lottery.reverse()
>>> print(lottery)
[59, 42, 30, 19, 12, 3]
```

很简单，对吧？如果你想给你的列表里加入些东西，你可以通过简入以下命令：

```
>>> lottery.append(199)
>>> print(lottery)
[59, 42, 30, 19, 12, 3, 199]
```

如果你只想要显示第一个数字，你可以通过使用索引完成。索引就是列表中出现一个项的位置。程序员一般习惯从0开始计数，所以列表中的第一个对象的索引是0，下一个是1，依此类推。试试这个：

```
>>> print(lottery[0])
59
>>> print(lottery[1])
42
```

正如你所见，你可以通过使用列表名和置方括号中的对象索引来访问列表中的不同元素。

如果需要从列表中删除元素，需要使用 `索引` 和上面学到的 `pop()` 语句来完成。我们看个例子，删除列表中的第一个数字，以加强我们之前学到的知识。

```
>>> print(lottery)
[59, 42, 30, 19, 12, 3, 199]
>>> print(lottery[0])
59
>>> lottery.pop(0)
>>> print(lottery)
[42, 30, 19, 12, 3, 199]
```

干得漂亮！

出于额外的乐趣，试试其他的索引：6，7，1000，-1，-6或者-1000。看看你是否能预测出命令尝试的结果。这些结果有意义吗？

你可以找到本章中所有关于列表的方法的Python文档：<https://docs.python.org/3/tutorial/datastructures.html>

字典

字典类似与列表，但是你通过查找键来获取值，而不是通过访问索引。一个键可以是任何字符串或者数字。定义一个空字典的语法是：

```
>>> {}
{}
```

这表明你刚创建了一个空字典。加油！

现在，试着写下下面的命令（试着也替换你自己的信息）：

```
>>> participant = {'name': 'Ola', 'country': 'Poland', 'favorite_numbers': [7, 42, 92]}
```

通过这个命令，你刚创建了一个名为 `participant` 的变量，有三个键值对：

- 键 `name` 指向 '`Ola`' (一个字符串 对象),
- `country` 指向 '`Poland`' (另一个 字符串),
- 和 `favorite_numbers` 指向 `[7, 42, 92]` (一个有三个数字的 列表).

你可以通过下面的语法检查每个键的内容：

```
>>> print(participant['name'])
Ola
```

看，这就和列表很相似了。但是你不需要记住索引-仅仅需要记住名字。

如果我们问Python一个不存在的键呢？能猜到吗？让我们试一试，看看！

```
>>> participant['age']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'age'
```

看，另一个错误！这是一个 **KeyError**。Python 会告诉你键 'age' 并不存在于这个字典里。

那么什么时候使用字典或者列表呢？好吧，这是一个值得思考的点。在查看下一行答案前先在脑子里面设想一个答案。

- 你需要一个有序的元素排列吗？使用列表。
- 你需要将键值相关联，这样你可以在后面更有效的查找他们？使用字典。

字典，就像列表，是可变的，意味着他们在创建之后可以被改变。你可以在字典被创建之后，增加新的键/值对，就像：

```
>>> participant['favorite_language'] = 'Python'
```

像列表，使用 `len()` 方法，返回字典中键值对数目。继续输入命令：

```
>>> len(participant)
4
```

我希望你能觉得到目前为止这些都合情合理。:)准备享受更多字典的乐趣吗？跳到下一行去做一些更有趣的事情。

你可以使用 `pop()` 语句去删除字典里的元素。比如，如果你想删除键 'favorite_numbers' 所对应的项，只需要键入如下命令：

```
>>> participant.pop('favorite_numbers')
>>> participant
{'country': 'Poland', 'favorite_language': 'Python', 'name': 'Ola'}
```

正如你从输出中看到的，键值对所对应的'favorite_numbers'键已经被删除了。

此外，你还可以改变字典中已经存在的键所对应的值。键入：

```
>>> participant['country'] = 'Germany'
>>> participant
{'country': 'Germany', 'favorite_language': 'Python', 'name': 'Ola'}
```

正如你所见，键 'country' 的值已经从 'Poland' 变为了 'Germany' 。:)兴奋吗？哈哈！你刚又学了另一个有趣的东西。

摘要

太棒了！你现在知道很多关于编程的东西了。在最后的部分你学习了：

- **errors** - 你知道如何读并理解错误，如果Python不能理解你所给它的命令。
- **variables** - 对象名称让你代码变得更简单更可读的
- **lists** - 按照特定序列排序的存储着对象的列表
- **dictionaries** - 存储着键值对的对象

为接下来的部分感到兴奋吗？：)

比较事物

编程里经常会比较事物。什么是最容易比较的东西呢？当然是数字。让我们来看一看它是怎么工作的：

```
>>> 5 > 2
True
>>> 3 < 1
False
>>> 5 > 2 * 2
True
>>> 1 == 1
True
>>> 5 != 2
True
```

我们给 Python 一些数字去比较。正如你所见的，Python 不仅可以比较数字，也可以比较函数的返回值。不错，是的吧？

你知道为什么我们在判断相等时要把两个等号 `==` 放在一起吗？我们使用一个等于符号 `=` 来给变量赋值。你总是需要把两个 `==` 放在一起，如果你希望去检查两个东西是不是相同。如果我们认为两个东西是不相等的。我们使用符号 `!=`，如上面的示例中所示。

给 Python 两个更多的任务：

```
>>> 6 >= 12 / 2
True
>>> 3 <= 2
False
```

`>` 和 `<` 很简单，但 `>=` and `<=` 表示什么啊？阅读下面的说明：

- `x > y` 表示：`x` 大于 `y`
- `x < y` 表示：`x` 小于 `y`
- `x <= y` 表示：`x` 小于或等于 `y`
- `x >= y` 表示：`x` 大于或等于 `y`

棒极了！想要做一次吗？试试这个：

```
>>> 6 > 2 and 2 < 3
True
>>> 3 > 2 and 2 < 1
False
>>> 3 > 2 or 2 < 1
True
```

你可以给 Python 任意多的数字来比较，他会给你想要的答案！非常智能，对吗？

- **and** - 如果你使用 `and` 运算符，两个比较值都需要为真（`True`），这样整个命令才能为真
- **or** - 如果你想使用 `or` 运算符，只要有一个比较值是真，那么整个命令就为真

你听说过“驴唇不对马嘴”这种说法吗？让我们试试它的 Python 版：

```
>>> 1 > 'django'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: '>' not supported between instances of 'int' and 'str'
```

在这里你看到就像在这表达式中，Python 是不能比较数字（`int`）和字符串（`str`）。相反，它显示一个 **TypeError**，并告诉我们两个类型不能相互比较。

布尔逻辑

顺带提一句，你刚刚学习了一个 Python 新的类型。它被叫做布尔——而且这可能是最简单的类型。

只有两个布尔类型对象: - True - False

但是为了 Python 能够理解，你需要写成 True (第一个字母大写，其他字母小写)。true, TRUE, tTUE 都不会正常工作 —— 只有 True 是正确的 (同样适用于 False)

布尔也可以是变量！看这里：

```
>>> a = True
>>> a
True
```

或者通过这种方式：

```
>>> a = 2 > 5
>>> a
False
```

练习有趣的布尔值，然后尝试下面的命令：

- True and True
- False and True
- True or 1 == 1
- 1 != 2

祝贺你！布尔值是编程中最酷的功能，你刚刚学会了如何使用它们！

保存它！

到目前为止，我们所写的所有 Python 代码都是在命令行中，这限制了我们每次只能写一行代码。正常的程序是被保存在文件里，并通过对应程序语言的解释器或者编译器处理后执行。目前为止，我们已经试过在 Python 解释器中一行一行的运行我们的程序。我们接下来的几个任务需要不止一行代码，所以我们很快就会需要：

- 退出 Python 解释器
- 打开我们选择的代码编辑器
- 将一些代码保存到一个新的 python 文件
- 运行它！

为了退出我们一直使用的 Python 解释器，只需要键入 exit() 函数：

```
>>> exit()
$
```

这将让你返回到命令提示符。

早些时候，我们在[代码编辑器][2]章节选择了一个代码编辑器。我们现在需要打开一个编辑器，然后写一些代码进入新文件：

[2]: code_editor/README. md

```
python
print('Hello, Django girls!')
```

注意你应该注意到代码编辑器最酷的事情之一：颜色！在 Python 控制台中，一切都是相同的颜色，现在你应该看到 打印 函数和字符串的颜色不同。这就所谓的“语法高亮”，在编写代码时是非常有用的功能。颜色会给你提示，如未闭合的字符串或关键字名称拼写错误（如函数中的 def 关键字，我们会在下面看到）。这是我们使用代码编辑器的原因之一：）

显然，你现在是一个相当熟练的python开发者，所以随便写一些你今天学到的代码吧。

现在我们需要保存文件，然后给它一个描述性的名字。让我们命名这个文件 **python_intro.py** 然后保存它到您的桌面。我们可以随意给文件起名字，但是要确保 **.py** 在文件名结尾。**.py** 扩展名告诉操作系统，这是一个 **python** 可执行文件，Python 可以运行它。

文件保存之后，就到了可以运行它的时候了！使用你在命令行章节学习到的技能，在终端改变目录到桌面。

在 Mac 上命令会看起来像这样：

```
$ cd ~/Desktop
```

在Linux，它会看起来像这样（“桌面”可能会被翻译成您所在的语言）：

```
$ cd ~/Desktop
```

并在 windows 上，它会看起来像这样：

```
> cd %HomePath%\Desktop
```

如果你遇到麻烦了，尽管提问寻求帮助。

现在文件中输入这样的代码并使用 Python 执行：

```
$ python3 python_intro.py
Hello, Django girls!
```

好吧！你刚刚运行了你保存在文件中的第一个 python 程序。感觉棒极了吗？

你可以继续去学习编程中一些重要的工具：

If...elif...else

很多代码只会在提供的条件被满足的时候运行。这就是为什么Python有**if**语句。

用以下代码替换 **python_intro.py** 中的内容：

```
python
if 3 > 2:
```

如果我们保存这个，然后运行它，我们会看到一个类似这样的错误：

```
$ python3 python_intro.py
File "python_intro.py", line 2
    ^
SyntaxError: unexpected EOF while parsing
```

当条件 `3 > 2` 为真（或者 `True`），Python 希望我们给它进一步的指示。我们试着让 Python 打印 “it works!”。更改您在 **python_intro.py** 文件中的代码变为这样：

```
if 3 > 2:
    print('It works!')
```

注意我们已经为下一行缩进了4个空格了吗？这样做是为了让 Python 知道如果条件正确它将运行什么代码。你可以使用一个空格，但是几乎所有的 Python 程序员都是用4个，使事情看上去很整洁。单个 `tab` 也将算作 4 个空格。

保存它，然后再次运行它：

```
$ python3 python_intro.py
It works!
```

如果条件不为真呢？

在之前的例子里，代码只会在条件被满足的时候被运行。但是 Python 同样有 `elif` 和 `else` 语句：

```
if 5 > 2:
    print('5 is indeed greater than 2')
else:
    print('5 is not greater than 2')
```

这段代码在运行后会打印：

```
$ python3 python_intro.py
5 is indeed greater than 2
```

如果2是比5大，那么第二个命令会被执行。简单，对吧？在我们看看 `elif` 是怎么工作的：

```
name = 'Sonja'
if name == 'Ola':
    print('Hey Ola!')
elif name == 'Sonja':
    print('Hey Sonja!')
else:
    print('Hey anonymous!')
```

然后运行：

```
$ python3 python_intro.py
Hey Sonja!
```

看到发生什么了吗？如果前面的条件失败了，`elif` 允许你添加额外条件来判断接下来该怎么运行。

在 `if` 语句之后，你可以添加任意多的 `elif` 语句。例如：

```
volume = 57
if volume < 20:
    print("It's kinda quiet.")
elif 20 <= volume < 40:
    print("It's nice for background music")
elif 40 <= volume < 60:
    print("Perfect, I can hear all the details")
elif 60 <= volume < 80:
    print("Nice for parties")
elif 80 <= volume < 100:
    print("A bit loud!")
else:
    print("My ears are hurting!")
```

Python 遍历判断每个测试的条件，并打印：

```
$ python3 python_intro.py
Perfect, I can hear all the details
```

摘要

在刚才的三个练习，你已经学习到了：

- 比较事物 - 在Python里你可以比较东西通过使用 `>`, `>=`, `==`, `<` 和 `and`, `or` 运算符。
- **Boolean** - 一个只能有两个值中的一个的对象: `True` 或者 `False`
- 保存文件 - 保存代码到文件里, 这样你可以执行更大的程序。
- `if...elif...else` - 让你只执行当某个条件满足后的代码。

现在是本章的最后一个部分了！

你自己的函数！

还记得你在Python里运行的函数 `len()` ? 好消息, 你会学习如何编写自己的函数了!

一个函数就是一些 Python 应该运行的指令集。每个Python函数都是以关键词 `def` 开始的, 我们可以给定一个名字并能指定若干个参数。让我们从一个简单的开始。使用下面的替换`python_intro.py`中的代码：

```
def hi():
    print('Hi there!')
    print('How are you?')

hi()
```

好吧, 我们的第一个函数已经准备好了!

你可以想知道为什么我们把函数的名称写在文件的底部。这是因为Python读了文件, 然后自顶向下的执行它。所以为了使用我们的函数, 我们必须要在底部重写它。

现在让我们运行这个, 看看会发生什么:

```
$ python3 python_intro.py
Hi there!
How are you?
```

那很简单! 让我们建立我们第一个有参数的函数。我们使用前面的例子-一个函数打印“hi to the person running it”

```
def hi(name):
```

正如你所见的, 我们给我们的函数一个叫 `name` 的参数:

```
def hi(name):
    if name == 'Ola':
        print('Hi Ola!')
    elif name == 'Sonja':
        print('Hi Sonja!')
    else:
        print('Hi anonymous!')

hi()
```

记住: `print` 函数是在 `if` 语句的缩进内的。这是因为我们要在仅满足条件时, 才运行函数。让我们看看它是如何工作:

```
$ python3 python_intro.py
Traceback (most recent call last):
File "python_intro.py", line 10, in <module>
    hi()
TypeError: hi() missing 1 required positional argument: 'name'
```

哦! 一个错误。幸运的是, Python给我们一个有用的错误提示信息。告诉我们函数 `hi()` (这是我们定义的) 必须有一个参数(称之为 `name`) , 我们调用函数的时候忘记传递它了。让我们在文件的底部解决它:

```
hi("Ola")
```

然后再次运行它：

```
$ python3 python_intro.py
Hi Ola!
```

那如果我们改变名字了呢？

```
hi("Sonja")
```

然后运行它：

```
$ python3 python_intro.py
Hi Sonja!
```

现在，如果我们写了另一个名字会发生什么？（不是 Ola 或者 Sonja）试试看是不是和你想的是一样的。它应该打印成这样：

```
Hi anonymous!
```

这太棒了，对吗？这样我们不用在每次调用此方法跟不同的人打招呼的时候重复自己。而这正是我们为什么需要（定义）函数 - 你永远不想重复你的代码！

让我们做一些更智能的事情——如果有超过两个或者更多的名字，我们需要为每个都写一个判断，会比较难，对不对？

```
def hi(name):
    print('Hi ' + name + '!')

hi("Rachel")
```

现在让我们调用代码：

```
$ python3 python_intro.py
Hi Rachel!
```

祝贺你！你刚刚学习了如何写函数！:)

循环

这是最后一部分。这真快，对吗？:)

程序员不喜欢重复劳动。编程的核心是自动化，所以我们不希望手动的一行一行去调用打招呼函数，是吧？这时候循环就能派上用场了。

还记得列表吗？让我们做一个女孩的列表：

```
girls = ['Rachel', 'Monica', 'Phoebe', 'Ola', 'You']
```

我们想要根据他们所有人的名字依次打招呼。我们有一个 `hi` 函数去实现这个，所以让我们在一个循环中使用它：

```
for name in girls:
```

`for` 语句的声明和 `if` 语句声明类似，下面代码需要缩进。

这是文件中完整的代码：

```
def hi(name):
    print('Hi ' + name + '!')

girls = ['Rachel', 'Monica', 'Phoebe', 'Ola', 'You']
for name in girls:
    hi(name)
    print('Next girl')
```

然后当我们去运行它：

```
$ python3 python_intro.py
Hi Rachel!
Next girl
Hi Monica!
Next girl
Hi Phoebe!
Next girl
Hi Ola!
Next girl
Hi You!
Next girl
```

正如你所看见的，所有你放在 `for` 中的语句都将会根据列表 `girls` 中的每个元素而重复执行。

你同样可以使用 `for` 来遍历使用 `range` 函数生成的数字：

```
for i in range(1, 6):
    print(i)
```

这将会打印：

```
1
2
3
4
5
```

`range` 函数产生一个列表的数字，一个挨着一个（这些数字是由您提供的参数而产生的）。

请注意第二个参数将不会被包括在Python输出列表中（意味着 `range(1, 6)` 从1到5计数，但是不包括数字6）。这是因为"范围"是半开区间，意思是包含第一个值，但不包括最后一个。

摘要

就是这样，你太厉害啦！这是一个棘手的章节，所以你应该为自己感到骄傲。我们为你取得这么多进展而感到骄傲！

你可以简单做点其他事情——伸个懒腰，走动一会儿，放松下你的眼睛，再接着往下第一章。:)



Django 是什么？

Django (*/dʒæŋgəʊ/jang-goh*) 是用 Python 写的一个自由和开放源码 web 应用程序框架。web 框架是一套组件，能帮助你更快、更容易地开发 web 站点。

当你开始构建一个 web 站点时，你总需要一些相似的组件：处理用户认证（注册、登录、登出）的方式、一个管理站点的面板、表单、上传文件的方式，等等。

幸运的是，其他人很早就注意到 web 开发人员会面临一些共同的问题。所以他们联手创建了 web 框架（Django 是其中一个）来让你使用。

由于框架的存在，你无需重新发明轮子就能建立新的站点。

你为什么需要一个框架？

要理解什么是 Django，我们需要更仔细的看一下服务器。服务器需要知道的第一件事就是你希望它为你的网页做什么。

想象一个用来监控接收邮件（请求）的邮箱（端口）。这就是网站服务器做的事情。网站服务器读这封信，然后将响应发送给网页。但是当你想发送一些东西的时候，你必须要有一些内容。而 Django 就是可以帮助您创建内容的工具。

当有人向您的服务器请求一个网站，会发生什么呢？

当一个请求到达网站服务器，它会被传递到 Django，试图找到实际上什么是被请求的。它首先会拿到一个网页的地址，然后试图去弄清该做什么。这个部分是由 Django 的 **urlresolver**（url 解析器）。注意一个网站的地址被叫做 URL，统一资源定位器，所以 url 解析器是有意义的）。它并不十分聪明，他接受一个模式列表，然后试图去匹配 URL。Django 从顶到底检查模式，如果有匹配上的，那么 Django 会将请求传递给相关的函数（这被称作视图）。

想象一个邮递员拿着一封信。她沿着街区走过去，检查每一个房号与信件地址是否对应。如果匹配上了，她就把信投在那里。这也是 url 解析器的工作方式！

在视图函数里做了很多有趣的事情：我们能在数据库中寻找一些信息。如果用户要求修改数据呢？就像一封信里说，“请修改我的工作描述”，视图将检查是否你允许它这么干，然后更新工作描述并发回一个消息：“做完了”。然后视图生成响应，并且 Django 能够发送给用户的 web 浏览器。

当然，上述的描述是有一些简化的，但是你不需要去知道所有的技术的东西。有一个大概的想法就够了。

所以不要太过深入细节，我们会简单用 Django 创建一些东西，然后我们在学习的路上学到所有重要的部分！

Django安装

注意如果你已完成了安装步骤，可以直接进入下一章。

本节部分内容基于 Geek Girls Carrots (<https://github.com/ggcarrots/django-carrots>) 教程而来。

本章的部分内容基于 [django-marcador](#) 教程，使用知识共享署名-4.0 国际许可协议许可。Django-marcador 教程的版权归 Markus Zapke-Gründemann 所有。

虚拟环境

在安装 Django 之前，我们会让你安装一个非常实用的工具，它可以让你计算机上的编码环境保持整洁。这个步骤可以跳过，但我们强烈建议你跟着做。从最佳实践设置开始将会在未来为你省去无数的烦恼！

所以，让我们创建一个 虚拟环境（也称为 *virtualenv*）。虚拟环境会以项目为单位将你的 Python/Django 安装隔离开。这意味着对一个网站的修改不会影响到你同时在开发的其他任何一个网站。优雅吧？

你需要做的就是找到您想创建 虚拟环境 的一个目录；比如您的主目录。在 Windows 上，它可能看起来像 `C:\Users\Name`（其中 `Name` 是您的登录名）。

在本教程中我们将使用您的 home 目录下的一个新目录 `djangogirls`：

```
mkdir djangogirls
cd djangogirls
```

我们将虚拟环境称为 `myvenv`。一般的命令格式是：

```
python3 -m venv myvenv
```

Windows 系统

若要创建新的 虚拟环境，您需要打开的控制台（我们在前几章里已经告诉过你了，还记得吗？），然后运行

```
C:\Python34\python -m venv myvenv
```

```
C:\Users\Name\djangogirls> C:\Python34\python -m venv myvenv
```

`C:\Python34` 是您之前安装Python的目录，`myvenv` 是您 虚拟环境 的名字。你可以使用其他任何名字，但请坚持使用小写，并不要使用空格、重音符号或特殊字符。始终保持名称短小是个好主意——你会大量引用它！

Linux 和 OS X

在 Linux 和 OS X 上创建的 虚拟环境 就和运行 `python3 -m venv myvenv` 一样简单。看起来像这样：

```
~/djangogirls$ python3 -m venv myvenv
```

`myvenv` 是您 虚拟环境 的名字。您可以使用其他任何名称，但请始终使用小写以及不要有空格。始终保持名称短小是个好主意，因为你会大量引用它！

注意：在Ubuntu 14.04上启动虚拟环境会报以下错误：

```
Error: Command '['/home/eddie/Slask/tmp/venv/bin/python3', '-Im', 'ensurepip', '--upgrade', '--default-pip']'
returned non-zero exit status 1
```

为了解决这个问题，请使用 `virtualenv` 命令。

```
~/djangogirls$ sudo apt-get install python-virtualenv
~/djangogirls$ virtualenv --python=python3.4 myvenv
```

使用虚拟环境

上面的命令将创建一个名为 `myvenv` 目录（或任何你选择的名字），其中包含我们的虚拟环境（基本上是一堆的目录和文件）。

Windows 系统

运行如下命令进入你的虚拟环境：

```
C:\Users\Name\djangogirls> myvenv\Scripts\activate
```

Linux 和 OS X

运行如下命令进入你的虚拟环境：

```
~/djangogirls$ source myvenv/bin/activate
```

记住要将 `myvenv` 替换成你选择的 虚拟环境 的名字！

注：有时 `source` 可能不可用。在这些情况下试着做下面这些事情：

```
~/djangogirls$ . myvenv/bin/activate
```

当你看到在您的控制台中有如下提示就知道你已经进入 虚拟环境：

```
(myvenv) C:\Users\Name\djangogirls>
```

或：

```
(myvenv) ~/djangogirls$
```

注意 `(myvenv)` 前缀的出现！

当在一个虚拟的环境中工作时，`python` 将自动指向正确的版本，因此您可以使用 `python` 代替 `python3`。

好的现在所有重要的依赖关系已经就位。最后，我们可以安装 Django！

安装 Django

既然你有了 虚拟环境，您可以使用 `pip` 安装 Django。在控制台中，运行 `pip install django==1.8`（注意我们使用双等号：`==`）。

```
(myvenv) ~$ pip install django==1.8
Downloading/unpacking django==1.8
  Installing collected packages: django
    Successfully installed django
  Cleaning up...
```

在 Windows 上

如果你在 Windows 平台上调用 pip 时得到一个错误，请检查是否您项目的路径名是否包含空格、重音符号或特殊字符（如：`C:\Users\User Name\django\girls`）。若的确如此，请尝试移动它到另外一个没有空格、重音符号或特殊字符的目录，（例如：`C:\django\girls`）。在移动之后，请重试上面的命令。

在 Linux 上

如果你在 Ubuntu 12.04 上得到一个错误，请运行 `pip python -m pip install -U --force-reinstall pip` 来修复虚拟环境中的 pip 安装。

就是这样！你现在（终于）准备好创建一个 Django 应用程序！

你的第一个Django 项目！

本章的部分内容基于 Geek Girls Carrots (<https://github.com/ggcarrots/django-carrots>) 的教程。

本章的部分是基于知识共享署名-4.0 国际许可协议许可的 [django marcador 教程](#)。 Django marcador 教程的版权由 Markus Zapke-Gründemann 持有。

我们将要创建一个简单的博客！

第一步是创建一个新的 Django 项目。首先，我们需要运行一些由 Django 提供的脚本，为我们即将开始的项目建立主要骨架。它会生成一系列的文件夹和文件，在后面的项目中我们会需要修改和使用到它们。

某些名称的文件和目录在 Django 起着至关重要的作用。你不应该重命名我们将要创建的文件。将它们移动到一个不同的地方也不是一个好主意。Django 需要固定的系统结构，以便 Django 能够找到重要的东西。

记住在虚拟环境中运行的一切。如果您没有看到您的控制台中的前缀 (myvenv)，您需要激活您的虚拟环境。我们在 [Django 安装](#)这一节内的 在虚拟环境下工作 部分中解释过了。在 windows 下面运行命令：
myvenv\Scripts\activate，在苹果或linux环境下运行命令：source myvenv/bin/activate

在苹果或Linux系统下，你需要运行下面的命令，记得不要漏掉命令后面的小点(.)：(myvenv) ~/djangogirls\$ django-admin startproject mysite .

```
(myvenv) ~/djangogirls$ django-admin startproject mysite .
```

在 windows 环境下也不要忘了下面命令最后的小点。

```
(myvenv) C:\Users\Name\djangogirls> django-admin startproject mysite .
```

此时，符号“.”很重要，它将告诉脚本程序自动安装Django到你当前选择的目录中（所以这个“.”是告诉脚本执行时的一种参考点）

注意：当敲入前面命令时，记住你是从 django-admin 或 django-admin.py 开始。而命令串前面的 (myvenv) ~/djangogirls\$ 和 (myvenv) C:\Users\Name\djangogirls> 则是由系统提示用户输入命令的光标位置，这部分不用用户输入。

django-admin.py 是一个脚本，将自动为您创建目录和文件。前面的命令正确的话，你现在就应该有一个目录结构，看起来像下面这样：

```
djangogirls
├── manage.py
└── mysite
    ├── settings.py
    ├── urls.py
    ├── wsgi.py
    └── __init__.py
```

它是一个Django的项目文件夹，为即将开始的项目准备好了必要的资源文件和文件夹

manage.py 是一个帮助管理站点的脚本。在它的帮助下我们将能够在我们的计算机上启动一个 web 服务器，而无需安装任何东西。

settings.py 文件包含您的网站的配置数据。

还记得当我们谈到一名邮差在决定何处交付一封信时的例子吗？urls.py 文件包含 urlresolver 所需的模型的列表。

现在让我们忽略其他文件，现在我们不会改变它们。要记住的唯一一点是不要不小心删除他们！

更改设置

让我们在 `mysite/settings.py` 中进行一些更改。使用您前面安装了的代码编辑器打开文件。

在我们的站点上有正确的时间是非常不错的。访问[wikipedia timezones list](#)复制你所在地区的时区 (TZ). (eg. `Europe/Berlin`)

然后在 `settings.py` 文件中，找到包含`TIME_ZONE</0>字段的这行，并将时区改为你所在地区的时区。即：`</p>`

```
python
TIME_ZONE = 'Europe/Berlin'
'
```

适当的修改"Europe/Berlin" ps:中国大陆地区可修改为 Asia/Shanghai

我们还需要添加(我们会找出在教程后面所提到的静态文件和 CSS文件)静态文件的路径。我们下拉到文件的最底部，就是在 `STATIC_URL` 条目的下面。添加新的一行内容为 `STATIC_ROOT`：

```
STATIC_URL = '/static/'
STATIC_ROOT = os.path.join(BASE_DIR, 'static')
```

设置数据库

有很多不同的数据库软件，可以用来存储你的网站数据。我们将使用默认的那个，`sqlite3`。

这已经在您的 `mysite/settings.py` 文件中设置了：

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}
```

若要为我们的博客创建一个数据库，，让我们运行以下命令在控制台中：`python manage.py migrate` (我们需要 `djangogirls` 目录中包含 `manage.py` 文件)。如果一切顺利，您应该看到这样：

```
(myvenv) ~/djangogirls$ python manage.py migrate
Operations to perform:
  Synchronize unmigrated apps: messages, staticfiles
  Apply all migrations: contenttypes, sessions, admin, auth
Synchronizing apps without migrations:
  Creating tables...
    Running deferred SQL...
  Installing custom SQL...
Running migrations:
  Rendering model states... DONE
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying sessions.0001_initial... OK
```

我们完成了！现在去启动网站服务器，看看是否我们的网站正在工作！

你必须进入包含 `manage.py` 文件的目录 (`djangogirls` 目录)。在控制台中，我们可以通过运行 `python manage.py runserver` 开启 web 服务器：

```
(myvenv) ~/djangogirls$ python manage.py runserver
```

如果你在Windows系统遇到UnicodeDecodeError这个错误，用这个命令来代替

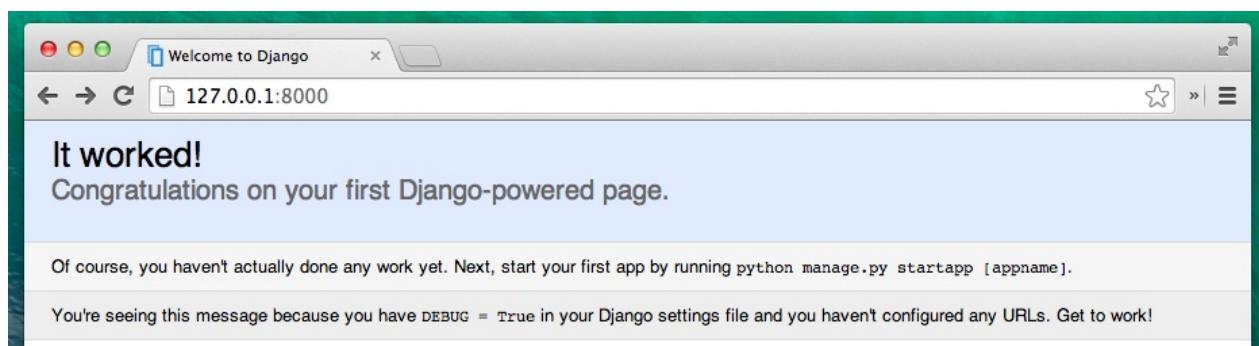
```
(myvenv) ~/djangogirls$ python manage.py runserver 0:8000
```

现在，你需要做的就是检测你的站点的服务器是否已经在运行了。打开你的浏览器（火狐，Chrome，Safari，IE 或者其他你惯用的浏览器）输入这个网址：

```
http://127.0.0.1:8000/
```

Web服务器将接管你的命令行提示符，直到我们停止它。为了尝试更多命令，我们应该同时打开一个新的终端，并激活虚拟环境。想要停止web服务器，我们应该切换到刚才在运行程序的窗口，并且按下 CTRL+C - 同时按下CTRL键和字母C键(如果你的操作系统是windows，那么应当按下 Ctrl+Break)。

祝贺你！你已经创建您的第一个网站，并使用 web 服务器运行它！这超级赞吧？



准备好下一步了吗？现在来创建一些内容！

Django模型

我们现在将要创建的是一个能存储我们博客所有文章的东西。为了达到这个目的，我们将要讲解一下一个被称为 `objects` (对象) 的东西。

对象

在编程中有一个概念叫做 面向对象编程。它的思想是，与其用无聊的一连串的程序指令方式写程序，我们不如为事物建立模型，然后定义他们是怎样互相交互的。

那什么是对象呢？它是一个属性和操作的集合。它听起来很奇怪，但我们会给你一个例子。

如果我们想塑造一只猫的模型，我们会创建一个名为 `cat` 的对象，它含有一些属性例如：`color`, `age`, `mood` (又比如：`good`, `bad`, `sleepy`;)，还有 `owner` (主人) (那是一个 `Person` 对象或者假若是流浪猫，这个属性可以为空)。

然后这个 `cat` 会有一些行为：`purr`, `scratch`, 或者 `feed` (在这其中我们会给这只猫一些 `CatFood`，这个 `CatFood` 可以是单独的一个包含比如`taste`属性的对象)。

```
Cat
-----
color
age
mood
owner
purr()
scratch()
feed(cat_food)

CatFood
-----
taste
```

所以基本思想就是用包含属性的代码来描述真实的东西 (称为 对象属性) 和操作 (称为 方法)。

我们将如何为博客帖子建立模型呢？我们想要建立一个博客，对吗？

我们需要回答一个问题：什么是一篇博客文章？它应该含有什么样的属性？

嗯，肯定我们的博客文章需要一些文本，包括内容与标题，对吗？我们也需要知道是谁写的——所以我们需要一位作者。最后，我们想要知道什么时候该文章创建并发布。

```
Post
-----
title
text
author
created_date
published_date
```

一篇博客文章需要做什么样的事情？应该有一些正确的 方法 来发布文章，对吗？

因此我们需要一个 `publish` 的方法

既然我们已经知道什么是我们想要实现的，让我们开始在Django里面为它建模！

Django模型

知道什么是对象，我们可以为我们的博客文章创建一个 Django 模型。

Django 里的模型是一种特殊的对象——它保存在 `数据库` 中。数据库是数据的集合。这是您存储有关用户、您的博客文章等信息的地方。我们将使用 SQLite 数据库来存储我们的数据。这是默认的 Django 数据库适配器——对于我们现在的需求而言它是足够的。

您可以将数据库中的模型看作是电子表格中的列（字段）和行（数据）。

创建应用程序

为了让一切保持整洁，我们将我们的项目内部创建单独的应用程序。如果一开始就把每一件东西井然有序，那就太好了。为了创建一个应用程序，我们需要在命令行中执行以下命令（从 `manage.py` 文件所在的 `djangogirls` 目录）：

```
(myvenv) ~/djangogirls$ python manage.py startapp blog
```

你会注意到一个新的 `blog` 目录被创建，它现在包含一些文件。我们的目录和我们的项目中的文件现在应该看起来像这样：

```
djangogirls
├── mysite
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
└── manage.py
└── blog
    ├── migrations
    │   ├── __init__.py
    │   ├── __init__.py
    ├── admin.py
    ├── models.py
    ├── tests.py
    └── views.py
```

创建应用程序后，我们还需要告诉 Django 它应该使用它。我们是在 `mysite/settings.py` 文件中这样做的。我们需要找到 `INSTALLED_APPS` 并在它下面添加一行 `'blog'`。所以最终的代码应如下所示：

```
INSTALLED_APPS = (
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'blog',
)
```

创建一个博客文章模型

我们在 `blog/models.py` 文件中，定义所有的 `Models` 对象——我们将在其中都定义我们的博客文章。

让我们打开 `blog/models.py`，从中删除一切并编写这样的代码：

```

from django.db import models
from django.utils import timezone

class Post(models.Model):
    author = models.ForeignKey('auth.User', on_delete=models.CASCADE)
    title = models.CharField(max_length=200)
    text = models.TextField()
    created_date = models.DateTimeField(
        default=timezone.now)
    published_date = models.DateTimeField(
        blank=True, null=True)

    def publish(self):
        self.published_date = timezone.now()
        self.save()

    def __str__(self):
        return self.title

```

当你在 `str` 的两端使用两个下划线字符（`_`）的时候务必三思而后行。这是Python编程里面的一种常见的约定写法，有时我们也叫这个做"dunder"("double-underscore"的缩写)。

这看起来太吓人了，对吧？不过不用担心，我们会解释这几行是什么意思！

所有以 `from` 或 `import` 开始的所有行，都是需要从其他文件中添加一些内容。所以与其复制和粘贴同样的内容，我们可以用 `from..... import.....` 来导入这些文件。

`class Post(models.Model):` - 这行是用来定义我们的模型(这是一个 对象)。

- `class` 是一个特殊的关键字，表明我们在定义一个对象。
- `Post` 是我们模型的一个名字。我们可以给它取另外一个不同的名字(但是我们必须避免使用特殊字符或者空格符)。总是以首字母大写来作为类名。
- `models.Model` 表明Post是一个Django模型，所以Django知道它应该被保存在数据库中。

现在我们定义了我们曾经提及到的那些属性：`title`，`text`，`created_date`，`published_date` 和 `author`。为了做到那样我们需要为我们每个字段定义一个类型(它是文本吗？是数字？是日期？到另一个对象的关联，比如用户吗？)。

- `models.CharField` - 这是你如何用为数有限的字符来定义一个文本。
- `models.TextField` - 这是没有长度限制的长文本。这听起来用在博客文章的内容上挺适合的，对吧？
- `models.DateTimeField` - 这是日期和时间。
- `models.ForeignKey` - 这是指向另一个模型的连接。

我们不会对这里的代码解释得面面俱到因为那会花太多时间了。如果你想了解更多有关模型字段以及如何定义除上面描述以外的东西，那你应该去看看Django的官方文档(<https://docs.djangoproject.com/en/1.8/ref/models/fields/#field-types>)。

`def publish(self):` 又怎样呢？这正是我们之前提到的 `publish` 方法。`def` 表明这是一个函数或者方法，`publish` 是这个方法的名字。如果你喜欢的话你可以改变方法名。命名的规则是使用小写字母以及下划线而非空白符。举个例子，一个计算平均价格的方法可以叫做 `calculate_average_price`。

方法通常会 `return` 一些东西。例如在 `__str__` 方法中就有这个。在这种情况下，当我们调用 `__str__()` 我们将得到文章标题的文本（字符串）。

如果关于模型尚有不清楚的，请随时问你的教练！我们知道它很复杂，特别是当你同时学习对象和函数的时候。但希望它在你看来没有那么神奇！

在你的数据库中为模型创建数据表

在这里的最后一步是将我们新的模型添加到我们的数据库。首先我们必须让Django知道我们在我们的模型(我们刚刚创建的！)有一些变更。输入 `python manage.py makemigrations blog`。它看起来会像这样：

```
(myvenv) ~/djangogirls$ python manage.py makemigrations blog
Migrations for 'blog':
  0001_initial.py:
    - Create model Post
```

Django为我们准备了我们必须应用到我们数据库的迁移文件。输入 `python manage.py migrate blog`，然后对应的输出应该是：

```
(myvenv) ~/djangogirls$ python manage.py migrate blog
Operations to perform:
  Apply all migrations: blog
Running migrations:
  Rendering model states... DONE
  Applying blog.0001_initial... OK
```

万岁！我们的Post模型现在已经在我们的数据库里面了！它看起来很不错，对吧？跳转到下一个章节，看看你博客文章的样子！

Django admin 管理后台

我们将使用 Django admin 添加，编辑和删除我们刚刚创建的帖子。

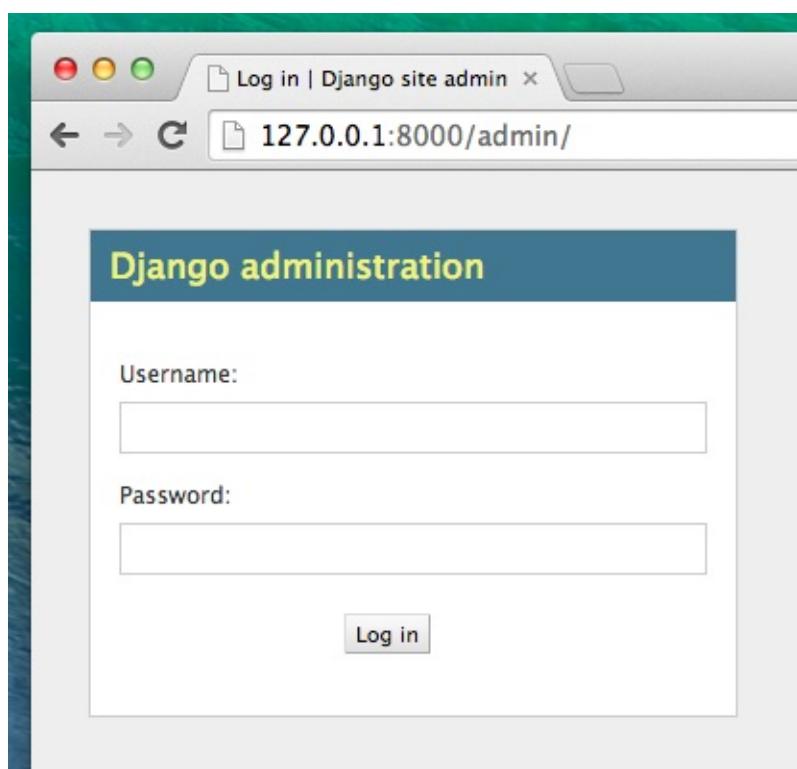
让我们打开 `blog/admin.py` 文件，并替换其中的文件像这样：

```
from django.contrib import admin
from .models import Post

admin.site.register(Post)
```

如你所见，我们导入（包括）了前一章定义的Post模型。为了让我们的模型在admin页面上可见，我们需要使用 `admin.site.register(Post)` 来注册模型。

OK, 现在来看看我们的 Post 模型。记得先在控制台输入 `python manage.py runserver` 启动服务器。然后打开浏览器，输入地址 <http://127.0.0.1:8000/admin/> 你会看到登录界面像这样：



为了登录，你需要创建一个掌控整个网站所有东西的超级用户。回到刚才的命令行，输入 `python manage.py createsuperuser`，按下Enter。然后输入你的用户名(英文小写，不包括空格)，邮箱和密码。你输密码的时候看不见输入？别担心，它就是这样的。你就输入要输得到然后按 Enter 继续就好了。输出应该长得像这样（用户名和邮箱应该是你自己的）：

```
(myenv) ~/djangogirls$ python manage.py createsuperuser
Username: admin
Email address: admin@admin.com
Password:
Password (again):
Superuser created successfully.
```

返回到你的浏览器，用你刚才的超级用户来登录，然后你应该能看到Django admin的管理面板。

The screenshot shows the Django administration interface. At the top, there's a header bar with the title "Site administration | Django" and the URL "127.0.0.1:8000/admin/". Below the header is a dark blue navigation bar with the title "Django administration". Underneath, there's a "Site administration" heading. The main content area is organized into sections:

- Auth**: Contains links for "Groups" (with "Add" and "Change" buttons) and "Users" (with "Add" and "Change" buttons).
- Blog**: Contains a link for "Posts" (with "Add" and "Change" buttons).

到 Posts 页面多试几次，发布5、6条博客文章，不用担心你的文章内容，你可以复制粘贴这个教程的一些文字，这样比较节约时间 :)。

请确保至少有两到三个帖子（但不是全部）具有设置的发布日期。它在以后会有用。

The screenshot shows the "Add post" form in the Django administration interface. The title bar says "Add post | Django site admin" and the URL is "127.0.0.1:8000/admin/blog/post/add/". The page title is "Django administration" and it says "Welcome, olasitarska. Change password / Log out". The breadcrumb navigation shows "Home > Blog > Posts > Add post".

The form fields are:

- Author:** A dropdown menu set to "olasitarska" with an "Add" button.
- Title:** A text input field containing "Cras justo odio, dapibus ac facilisis in, eu".
- Text:** A large text area containing placeholder text about a blog post.
- Created date:** Date: 2014-07-07, Today |
- Published date:** Date: 2014-07-07, Time: 23:07:34

At the bottom right of the form are three buttons: "Save and add another", "Save and continue editing", and a highlighted "Save" button.

如果你想更多地了解Django admin模块，你可以查看Django 的官方文档:<https://docs.djangoproject.com/en/1.8/ref/contrib/admin/>

现在你可以来杯咖啡(或者是茶) 或吃点东西给自己充下电，你刚刚创建了你的第一个Django模型，你应该休息一下。

部署！

注 这一章可以有时有点难打通。坚持并完成它；部署是在网站开发过程的重要组成部分。这一章放在本教程的中部，因此你的指导者可以帮助你在使网站上线中的一些小困难。这意味着如果您花大量的时间，你仍然能独立完成这个教程。

到目前为止您的网站只是在您的计算机上可用，现在您将了解如何部署它！部署是在互联网上发布你的应用程序的一系列过程，因此人们最终可以一起去看看你的应用程序。

正如你所学习的，一个网站必须要放到一个服务器上。在互联网上你可以找到很多的服务器供应商。我们将使用一个相对简单的部署过程：[PythonAnywhere](#)。PythonAnywhere 对于一些没有太多访问者的小应用是免费的，所以它对你来说绝对是足够使用的。

其它我们将使用到的外部服务是[GitHub](#)，它是一个代码托管服务。还有其它的一些服务，但当今几乎所有的程序员都有 GitHub 帐户，并且现在你也会有的！

我们将使用 GitHub 作为基石，以和 PythonAnywhere 互相传输我们的代码。

Git

Git是一个被大量程序员使用的"版本控制系统"。此软件可以跟踪任何时间文件的改变，这样你以后可以随时召回某个特定版本。有点像Microsoft Word 的"跟踪更改"功能，但更强大。

安装Git

注意 如果你已经做过安装步骤了，你可以直接跳过这个步骤开始创建你自己的Git版本库。

Windows系统

你可以从 [git-scm.com](#) 下载Git。你可以在所有的安装步骤中点击"next next next"，除了第5步"Adjusting your PATH environment"，需要选择"Run Git and associated Unix tools from the Windows command-line"(底部的选项)。除此之外，默认值都没有问题。签出时使用 Windows 风格的换行符，提交时使用 Unix 风格的换行符，这样比较好。

MacOS系统

从[git-scm.com](#) 下载Git,根据说明操作。

Linux系统

如果git还没有被安装的话，你可以从你的软件包管理器中下载git，请试试下面命令：

```
sudo apt-get install git
# or
sudo yum install git
# or
sudo zypper install git
```

开始我们自己的Git版本库

Git跟踪一组特定的在代码仓库（或简称“仓库”）中文件的更改。我们开始用git管理自己的项目吧。打开你的终端，进入 `djangogirls` 文件夹运行以下的命令：

注意在初始化仓库之前，请使用 `pwd` 命令 (OSX/Linux) 或者 `cd` (Windows) 命令检查你当前的工作目录。你应该是在 `djangogirls` 文件夹下运行命令。

```
$ git init
Initialized empty Git repository in ~/djangogirls/.git/
$ git config --global user.name "Your Name"
$ git config --global user.email you@example.com
```

每个项目我们只需要初始化一次Git仓库（而且你从此不需要重新输入用户名和邮箱）。

Git会追踪这个目录下所有文件和文件夹的更改，但是有一些文件我们希望Git忽略它。为此，我们可以在系统根目录下创建一个命名为 `.gitignore` 的文件。打开编辑器，创建新文件并写入以下内容：

```
*.pyc
__pycache__
myvenv
db.sqlite3
.DS_Store
```

然后在 `djangogirls` 项目根目录下保存为 `.gitignore` 文件。

注意在文件名最前面的“.”很重要。如果你在创建文件的时候有困难(例如，Mac不能通过Finder创建开头为“.”的文件)，可以在编辑器里点击“另存为”，这个没问题的。

在执行git操作之前，最好使用 `git status` 命令查看一下当前的状态，尤其是在执行 `git add` 或者在你不确定哪些文件被改动的情况下。这有助于阻止各种意外发生，例如错误的文件被添加或提交。`git status` 命令会返回所有未追踪/修改/暂存的文件，还有分支状态等信息。输出会是这样：

```
$ git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    .gitignore
    blog/
    manage.py
    mysite/

nothing added to commit but untracked files present (use "git add" to track)
```

最后保存我们的更改。转到你的控制台并运行这些命令：

```
$ git add --all .
$ git commit -m "My Django Girls app, first commit"
[...]
13 files changed, 200 insertions(+)
create mode 100644 .gitignore
[...]
create mode 100644 mysite/wsgi.py
```

推送我们的代码到 GitHub 上

跳转到 [GitHub.com](#) 网站，注册一个新的免费账号。（如果你在看线下活动之前就已经有账号的话，那就太好了！）

现在，创建一个新的仓库，命名为“my-first-blog”。保持 “initialise with a README” 复选框未选中状态，`.gitignore` 选项为无（我们已经手动创建了），让 License 设置为无。

Owner **Repository name**

 **hjwp** / my-first-blog 

Great repository names are short and memorable. Need inspiration? How about **ducking-octo-tyrion**.

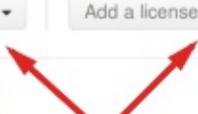
Description (optional)

 **Public**
Anyone can see this repository. You choose who can commit.

 **Private**
You choose who can see and commit to this repository.

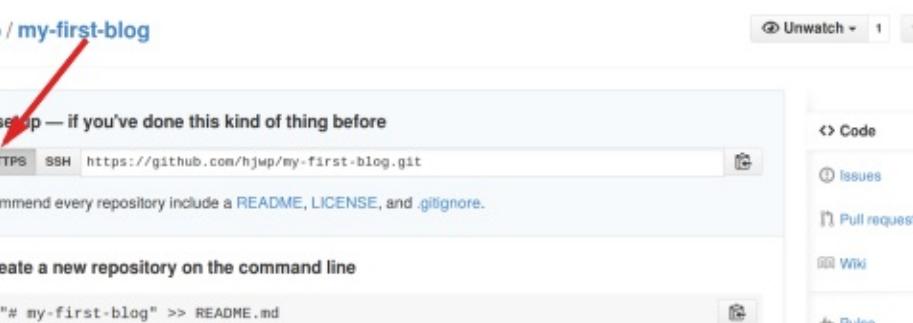
Initialize this repository with a README
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **None**  Add a license: **None**  

注意 `my-first-blog` 这个名字很重要 — 你可以用其它的，但是这个名字会在教程中出现多次，你需要确保每次都替换成它。保持用 `my-first-blog` 这个名字可能更为简单。

在下一屏中，你将看到你的仓库克隆 URL。选择“HTTPS”版本，拷贝地址，我们马上要把它粘贴到终端：



A screenshot of a GitHub repository setup page. The URL in the address bar is <https://github.com/hjwp/my-first-blog>. The page title is "hjwp / my-first-blog". A red arrow points to the "HTTPS" button in the "Quick setup" section, which contains instructions for cloning the repository. To the right, there's a sidebar with links to Code, Issues, Pull requests, Wiki, Pulse, Graphs, and Settings.

Quick setup — if you've done this kind of thing before

or **HTTPS** SSH <https://github.com/hjwp/my-first-blog.git>

We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

...or create a new repository on the command line

```
echo "# my-first-blog" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/hjwp/my-first-blog.git
git push -u origin master
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/hjwp/my-first-blog.git
git push -u origin master
```

现在我们需要把你电脑上的Git仓库和Github上的挂接。

在控制台输入以下内容（替换 <your-github-username> 为你的 github 用户名，不包含尖括号）：

```
$ git remote add origin https://github.com/<your-github-username>/my-first-blog.git  
$ git push -u origin master
```

输入你的Github账号名和密码，然后你会看到这样：

```
Username for 'https://github.com': hjwp
Password for 'https://hjwp@github.com':
Counting objects: 6, done.
Writing objects: 100% (6/6), 200 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/hjwp/my-first-blog.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.
```

你的代码已经在Github上了。快去确认一下吧！你会发现这些好家伙们 — [Django](#), [Django Girls Tutorial](#)，还有很多其它优秀的开放源代码软件项目同样也在Github上 :)

在 PythonAnywhere 设置我们的博客

注意 你可能在之前的安装步骤中已经创建 PythonAnywhere 账户 — 如果是的话，那么无需再来一次。

下一步，在PythonAnywhere注册一个“Beginner”账户。

- www.pythonanywhere.com

注意 在这里选择的用户名会出现在博客链接的地址当中 `yourusername.pythonanywhere.com`，因此最好选择你自己的绰号或者是与博客内容相关的名字。

在 PythonAnywhere 上拉取我们的代码

当然注册完 PythonAnywhere，你讲会转到仪表盘或“控制台”页面。选择启动“Bash”控制台这一选项 — 这是 PythonAnywhere 版的控制台，就像你本地电脑上的一样。

注意 PythonAnywhere 基于 Linux，因此如果你使用 Windows，控制台将会和你本地电脑上的略有不同。

让我们通过创建一个我们仓库的“Clone”以便从 Github 拉取代码到 PythonAnywhere。在 PythonAnywhere 控制台输入以下(不要忘记使用 Github 用户名替换 `<your-github-username>`)：

```
$ git clone https://github.com/<your-github-username>/my-first-blog.git
```

这将会拉取一份你的代码副本到 PythonAnywhere 上。通过键入 `tree my-first-blog` 查阅：

```
$ tree my-first-blog
my-first-blog/
├── blog
│   ├── __init__.py
│   ├── admin.py
│   ├── migrations
│   │   ├── 0001_initial.py
│   │   └── __init__.py
│   ├── models.py
│   ├── tests.py
│   └── views.py
└── manage.py
└── mysite
    ├── __init__.py
    ├── settings.py
    ├── urls.py
    └── wsgi.py
```

在 PythonAnywhere 上创建 virtualenv

如同你在自己电脑上做的，你可以在 PythonAnywhere 上创建 virtualenv 虚拟环境。在 Bash 控制台下，键入：

```
$ cd my-first-blog

$ virtualenv --python=python3.4 myvenv
Running virtualenv with interpreter /usr/bin/python3.4
[...]
Installing setuptools, pip...done.

$ source myvenv/bin/activate

(mvenv) $ pip install django whitenoise
Collecting django
[...]
Successfully installed django-1.8.2 whitenoise-2.0
```

注意 `pip` 安装 步骤可能需要几分钟。耐心，耐心！但是如果超过 5 分钟，就不对劲了。问问你的教练。

收集静态文件。

你可曾好奇，什么是“whitenoise”白噪音？它是用来服务所谓的“static files”静态文件的工具。静态文件是很少改动或者并非可运行的程序代码的那些文件，比如 HTML 或 CSS 文件。在我们的计算机上，它们以不同的方式工作，我们需要比如“whitenoise”这样的工具来为其服务。

在教程后续编辑网站 CSS 章节会介绍更多有关静态文件的内容。

暂且我们只需要在服务器上运行一个额外的命令，就是 `collectstatic`。它告诉 Django 去收集服务器上所有需要的静态文件。就眼下来说主要是使 admin 管理界面看起来更漂亮的文件。

```
(mvenv) $ python manage.py collectstatic

You have requested to collect static files at the destination
location as specified in your settings:

/home/edith/my-first-blog/static

This will overwrite existing files!
Are you sure you want to do this?

Type 'yes' to continue, or 'no' to cancel: yes
```

键入 “yes”，然后它会自行运转！你可喜欢让计算机打印一页一页令人费解的文本？我总会弄点噪音作为伴奏。Brp, brp brp...

```
Copying '/home/edith/my-first-blog/mvenv/lib/python3.4/site-packages/django/contrib/admin/static/admin/js/actions.min.js'
Copying '/home/edith/my-first-blog/mvenv/lib/python3.4/site-packages/django/contrib/admin/static/admin/js/inline.min.js'
[...]
Copying '/home/edith/my-first-blog/mvenv/lib/python3.4/site-packages/django/contrib/admin/static/admin/css/changelist.css'
Copying '/home/edith/my-first-blog/mvenv/lib/python3.4/site-packages/django/contrib/admin/static/admin/css/base.css'
62 static files copied to '/home/edith/my-first-blog/static'.
```

在 PythonAnywhere 上创建数据库

服务器与你自己的计算机不同的另外一点是：它使用不同的数据库。因此用户账户以及文章和你电脑上的可能会有不同。

我们可以像在自己的计算机上一样在服务器上初始化数据库，使用 `migrate` 以及 `createsuperuser`：

```
(mvenv) $ python manage.py migrate
Operations to perform:
[...]
    Applying sessions.0001_initial... OK

(mvenv) $ python manage.py createsuperuser

Context | Request Context
```

将我们的博客发布为一个网络应用程序

现在我们的代码已在PythonAnywhere上，我们的 virtualenv 已经准备好，静态文件已收集，数据库已初始化。我们准备好发布网络应用程序！

通过点击 logo 返回到 PythonAnywhere 仪表盘，然后点击 **Web** 选项卡。最终，点 **Add a new web app.**

在确认你的域名之后，选择对话框中 **manual configuration** (注 不是 "Django" 选项)：下一步选择 **Python 3.4**，然后点击 **Next** 以完成该向导。

注意 确保你选中 "Manual configuration" 选项，而不是 "Django" 那个。我们太牛逼，所以不要用 PythonAnywhere Django 默认设置 ;-)

设置 virtualenv

你将会被带到 PythonAnywhere 上你的Web 应用程序的配置屏，那个页面是每次你想修改服务器上你的应用程序时候要去的页面。

Configuration for [edith.pythonanywhere.com](#)

Actions:

Code:

What your site is running.

Source code:	You can use the Files tab to navigate to your app's source code.
WSGI configuration file:	/var/www/edith_pythonanywhere_com_wsgi.py
Python version:	3.4

Virtualenv:

Use a virtualenv to get different versions of flask, django etc from our default system ones. More info [here](#). You need to Reload your web app to activate it; NB - will do nothing if the virtualenv does not exist.

/home/edith/my-first-blog/myvenv

在“Virtualenv”一节，点击红色文字“Enter the path to a virtualenv”，然后键入：`/home/<your-username>/my-first-blog/myvenv/`。前进之前，先点击有复选框的蓝色框以保存路径。

注意 替换你自己的用户名。如果你犯了错，PythonAnywhere 会显示一个小警告。

配置 WSGI 文件

Django 使用 “WSGI 协议”，它是用来服务 Python 网站的一个标准。PythonAnywhere 支持这个标准。PythonAnywhere 识别我们 Django 博客的方式是通过配置 WSGI 配置文件。

点击 “WSGI configuration file” 链接（在 “Code” 一节，接近页面上方 — 它将被命名为如 `/var/www/<your-username>_pythonanywhere_com_wsgi.py`），然后跳转到一个编辑器。

删除所有的内容并用以下内容替换：

```
import os
import sys

path = '/home/<your-username>/my-first-blog' # use your own username here
if path not in sys.path:
    sys.path.append(path)

os.environ['DJANGO_SETTINGS_MODULE'] = 'mysite.settings'

from django.core.wsgi import get_wsgi_application
from whitenoise.django import DjangoWhiteNoise
application = DjangoWhiteNoise(get_wsgi_application())
```

注意 当看到 `<your-username>` 时，别忘了替换为你的用户名。

这个文件的作用是告诉 PythonAnywhere 我们的 Web 应用程序在什么位置，Django 设置文件的名字是什么。它也设置 “whitenoise” 静态文件工具。

点击 **Save** 然后返回到 **Web** 选项卡。

一切搞定！点击大大的绿色 **Reload** 按钮然后你将会看到你的应用程序。页面的顶部可以看到它的链接。

调试小贴士

如果你在访问你的网站时候看到一个错误，首先要去 **error log** 中找一些调试信息。你可以在 PythonAnywhere **Web 选项卡** 中发现它的链接。检查那里是否有任何错误信息，底部是最新的信息。常见问题包括：

- 忘记我们在控制台中的步骤之一：创建 `virtualenv`，激活它，安装 `Django` 进去，运行 `collectstatic`，迁移数据库。
- 在 **Web** 选项卡中，`virtualenv` 路径设置错误 — 如果真是这样，这通常会是一个红色错误消息。
- `WSGI` 文件设置错误 — 你的 `my-first-blog` 目录地址设置是否正确？
- 你是否为你的 `virtualenv` 选择了同样的 `Python` 版本，如同 **Web** 应用程序里的那样？两个应该都是 `3.4`。
- 有一些常见的调试小贴士在 [general debugging tips on the PythonAnywhere wiki](#) 里。

记住，你的教练会在这里帮你！

你上线了！

你网站的默认页面说 “Welcome to Django”，如同你本地计算机上的一样。试着添加 `/admin/` 到 URL 的末尾，然后你会到达管理者的页面。输入用户名和密码登录，然后你会看到服务器上的 `add new Posts`。

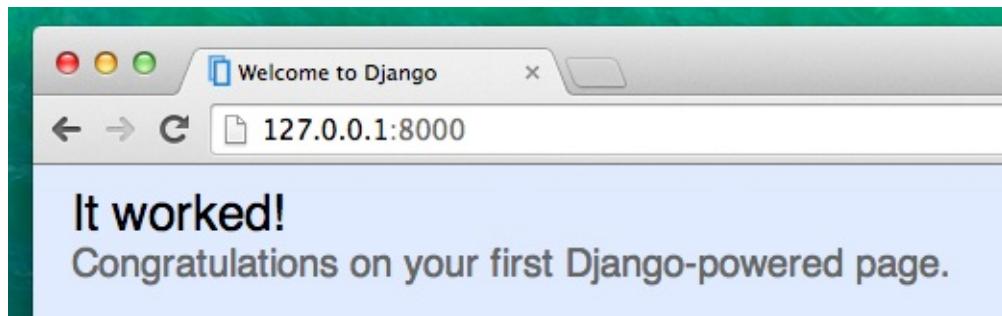
给你自己一个超大的鼓励！服务器部署是 web 开发中最棘手的部分之一，它通常要耗费人们几天时间才能搞定。但你的网站已经上线，运转在真正的互联网上，就是这样！

Django urls

我们将要建立第一个网页：你博客的主页！但是首先，让我们先学习一点 Django 的 url 知识。

什么是 URL？

简单地说，URL 是一个网页地址。每当你访问一个网站时，你都能在浏览器的地址栏里看到一个 URL。（是的！`127.0.0.1:8000` 是一个 URL！同时 `https://djangogirls.org` 也是一个 URL）：



每一个互联网的网页都需要自己的 URL。这样当用户打开一个 URL 时，你的应用程序才知道应该展现什么内容。在 Django 中，我们使用一种叫做 `URLconf`（URL 配置）的机制。`URLconf` 是一套模式，Django 会用它来把 URL 匹配成相对应的 View。

URL 在 Django 中如何工作？

使用你喜欢的编辑器打开 `mysite/urls.py` 就能看到它长什么样子了：

```
from django.conf.urls import include, url
from django.contrib import admin

urlpatterns = [
    # Examples:
    # url(r'^$', 'mysite.views.home', name='home'),
    # url(r'^blog/', include('blog.urls')),

    url(r'^admin/', include(admin.site.urls)),
]
```

正如你所看到的，Django 已经为我们放了一些东西在里面。

以 `#` 开头的行是注释，这些行都不会被 Python 运行。是不是很方便呢？

你在上一章节中学到的 `admin` 的 URL 已经在里面了：

```
url(r'^admin/', include(admin.site.urls)),
```

这表示对于每一个以 `admin` 开头的 URL，Django 都会找到一个相对应的 view。在这行代码中，我们包含了许多 `admin` URL 进来，所以这些 URL 不需要都被打包进这个小文件中。这使得代码更具可读性和简洁性。

正则表达式

你知道 Django 是如何将 URL 匹配到 view 的吗？好吧，这部分很棘手。Django 使用了 `regex`，这是“正则表达式”的缩写。正则表达式有很多（非常多！）规则用来形成一个搜索模式。由于正则表达式是一个比较深入的话题，我们不会太深入讲解它的运行机制。

如果你还是想要了解怎样创建这些模式，下面有一个例子。我们只需要有限的规则的子集，就可以表达出想要的模式，比如：

```
^ 表示文本的开始
$ 表示文本的结束
\d 表示数字
+ 表示前面的元素应该重复至少一次
() 用来捕捉模式中的一部分
```

其他定义在模式中的部分会保留原本的含义。

现在，想象你有一个网站，其中有一个 URL 类似这样：`http://www.mysite.com/post/12345/`。其中 `12345` 是帖子的编号。

为每一个帖子都写一个单独的视图是一件会让人恼火的事情。用正则表达式，我们可以创建一种模式，用来匹配 URL 并提取出帖子编号：`^post/(\d+)/$`。让我们一步一步将它分解，看看里面做了什么：

- `^post/` 告诉 Django 在 URL 的开头匹配 `post/`（后于 `^`）
- `(\d+)` 表示 URL 中会有一个数（一位或者多位数字），并且我们想提取出这个数
- `/` 告诉 Django 后面紧跟着一个 `/` 字符
- `$` 表示 URL 的末尾，即以 `/<1>` 结尾的 URL 才会被匹配到

你的第一个 Django url !

是时候创建第一个 URL 了！我们想用 '`http://127.0.0.1:8000/`' 作为博客的首页，并展示一个帖子列表。

我们也想保持 `mysite/urls.py` 文件简洁，所以我们从 `blog` 应用导出 `urls` 到主 `mysite/urls.py` 文件。来吧，删除被注释掉的那行（以 `#` 开头的行），然后添加一行代码用于把 `blog.urls` 导入到主 `url('')`。

你的 `mysite/urls.py` 文件现在应该看起来像这样：

```
from django.conf.urls import include, url
from django.contrib import admin

urlpatterns = [
    url(r'^admin/', include(admin.site.urls)),
    url(r'', include('blog.urls')),
]
```

现在，Django 会把访问 '`http://127.0.0.1:8000/`' 的请求转到 `blog.urls`，并看看那里面有没有进一步的指示。

写正则表达式时，记得把一个 `r` 放在字符串的前面。这告诉 Python，这个字符串中的特殊字符是为正则表达式准备的，而不是为 Python 自身准备的。

blog.urls

现在我们创建一个新的 `blog/urls.py` 空文件。好了！加入以下两行：

```
from django.conf.urls import url
from . import views
```

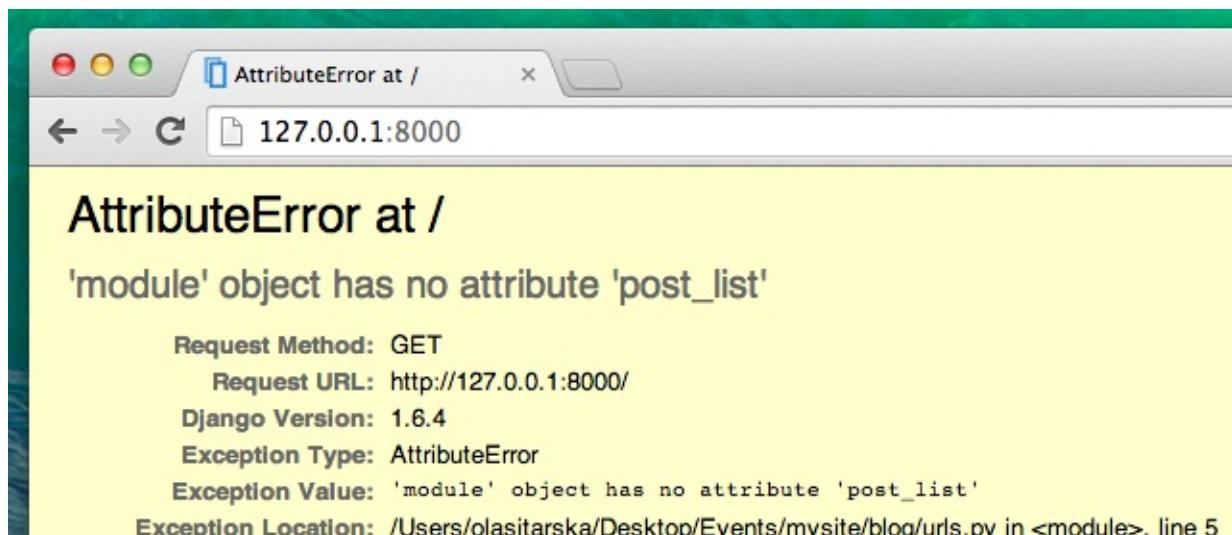
我们仅仅把 Django 的方法以及 `blog` 应用的全部 `views` 导入了进来。（目前什么都没有，但是不超过一分钟就能搞好！）然后，我们可以加入第一个 URL 模式：

```
urlpatterns = [
    url(r'^$', views.post_list, name='post_list'),
]
```

正如你所见，我们现在分配了一个叫作 `post_list` 的 `view` 到 `^$` 的 URL 上。这个正则表达式会匹配 `^`（表示开头）并紧随 `$`（表示结尾），所以只有空字符串会被匹配到。这是正确的，因为在 Django 的 URL 解析器中，`'http://127.0.0.1:8000/'` 并不是 URL 的一部分。（译注：即只有 `'http://127.0.0.1:8000/'` 后面的部分会被解析。如果后面的部分为空，即是空字符串被解析。）这个模式告诉了 Django，如果有人访问 `'http://127.0.0.1:8000'` 地址，那么 `views.post_list` 是这个请求该去到的地方。

最后的部分，`name='post_list'` 是 URL 的名字，用来唯一标识对应的 `view`。它可以跟 `view` 的名字一样，也可以完全不一样。在项目后面的开发中，我们将会使用命名的 URL，所以在应用中为每一个 URL 命名是重要的。我们应该尽量让 URL 的名字保持唯一并容易记住。

一切都搞定了？在浏览器里打开 `http://127.0.0.1:8000/`，看看结果。



"It works" 不见了，啊？不要担心，这只是个错误页面，不要被吓到了。它们实际上是非常有用的：

你会发现有一个 **no attribute 'post_list'**（没有 'post_list' 属性）的错误。`post_list` 提醒你什么东西了吗？这是我们的 `view` 的名字！这表示其他的一切正常，但是我们还没创建这个 `view`。不要担心，我们将会抵达那里。

如果你想了解更多关于 Django URLconf 的知识，去官方文档看一看：<https://docs.djangoproject.com/en/1.8/topics/http/urls/>

Django视图 - 是时候去创建！

是时候去解决我们在上一章所制造的Bug了：）

`view`是存放应用逻辑的地方。它将从你之前创建的 `模型` 中获取数据，并将它传递给 `模板`。我们将在下一章创建 `tempalte` 模板。视图就是Python方法，只不过比我们在Python简介章节中所做的事情稍复杂。

视图都被置放在 `views.py` 文件中。我们将加入我们自己的`views`到 `blog/views.py` 文件。

blog/views.py

好，让我们打开这个文件，看看里面有什么：

```
from django.shortcuts import render

# Create your views here.
```

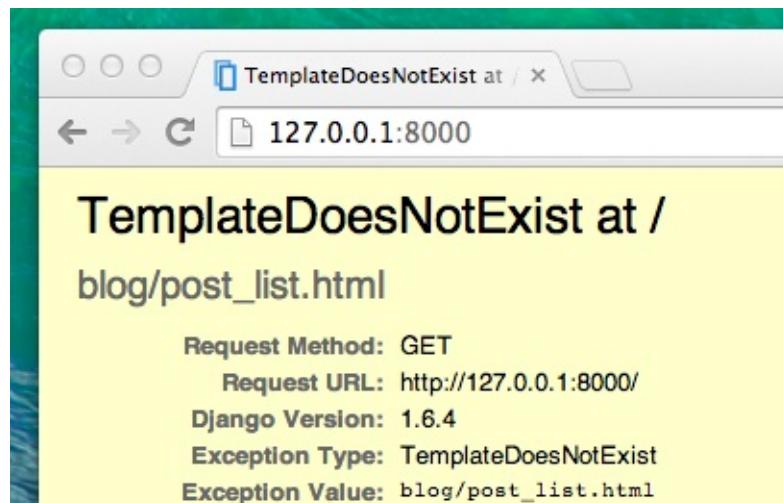
这里没有太多的东西。一个最简单的`view`就长得这个样子。

```
def post_list(request):
    return render(request, 'blog/post_list.html', {})
```

如你所见，我们创建一个方法 (`def`)，命名为 `post_list`，它接受 `request` 参数作为输入，并 `return` (返回) 用 `render` 方法渲染模板 `blog/post_list.html` 而得到的结果。

保存文件，转到 <http://127.0.0.1:8000/> 然后看看我们现在得到什么了。

另一个错误！读读现在发生了什么：



这个错误很直白：`TemplateDoesNotExist`。让我们修复这个bug，然后在下一个章节里创建一个模板！

阅读更多关于 Django views 的内容请参阅官方文档：<https://docs.djangoproject.com/en/1.8/topics/http/views/>

HTML 简介

什么是模板，你可能会问？

模板是一个文件，它可以让我们将一致的格式来展示不同的信息——例如，您可以使用模板来帮助您写一封信，虽然每封信可以包含不同的消息和发送给不同的人，但他们使用相同的格式。

Django模板的格式是由HTML（也就是我们在第一章 互联网是如何工作的 中提到的HTML）语言来描述的。).

HTML 是什么？

HTML 是一种简单的代码，由 Web 浏览器解释——如 Chrome、火狐或 Safari——为用户显示一个网页。

HTML 代表“HyperText Markup Language（超文本标记语言）”。超文本是指它是一种支持网页之间的超链接的文本。标记是指我们将一份文件用代码标记组织起来，然后告诉某个东西（在这种情况下，浏览器中）如何解释网页。HTML 代码是由标记构成的，每一个都是由 < 开始，由结束 >。这些标签表示标记元素。

你的第一个模板！

创建一个模板是指创建一个模板文件。一切都是文件，对吧？你可能已经注意到这点了。

模板保存在 `blog/templates/blog` 目录中。因此，首先在您的 `blog` 目录内创建一个称为 `templates` 的目录。然后创建另一个称为 `blog` 的目录到你的 `templates` 目录：

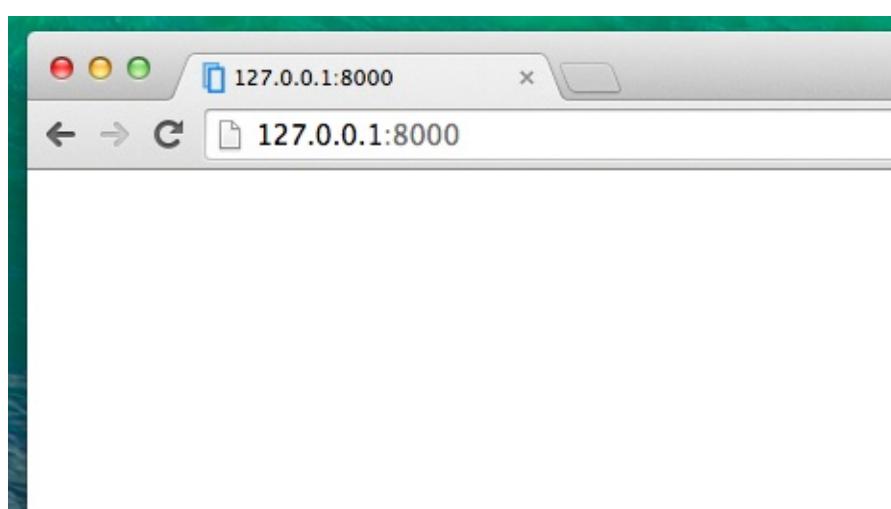
```
blog
└── templates
    └── blog
```

（你可能会疑惑为什么我们需要两个目录都叫的 `blog` — 你以后会发现这是简单有效的命名规则，使生活更容易，当事情开始变得更加复杂的时候。）

现在创建一个叫做 `post_list.html` 的文件（现在是空的，别管它）到 `blog/templates/blog` 目录下。

看看你的网站现在是什么样子：<http://127.0.0.1:8000/>

如果您仍然有错误 `TemplateDoesNotExist`，尝试重新启动您的服务器。转到命令行，按 `Ctrl + C`（同时按 Control 和 C 按钮）停止服务，并通过执行 `python manage.py runserver` 命令再次启动它。

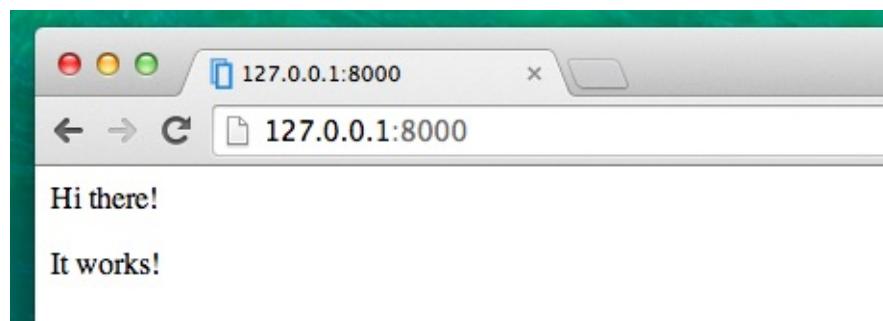


再也没有错误了！祝贺 :) 然而，您的网站实际上并没有展现任何东西除了一个空白页，因为您的模板也是空的。我们需要解决这个问题。

在您的模板文件中添加以下内容：

```
<html>
  <p>Hi there!</p>
  <p>It works!</p>
</html>
```

你的网站现在看这样怎么样？单击去查找：<http://127.0.0.1:8000/>



它工作了！很好地完成了：）

- 最基本的标记，`<html>`，始终是任何网页的开始，`</html>` 始终是最后。正如你所看到的，整个网站的内容是在`<html>` 开始标记和结束标记之间`</html>` 的
- `<p>` 一种用于段落元素标记；`</p>` 关闭每个段落

Head & body

每个 HTML 页面也分为两个元素：**head** 和 **body**。

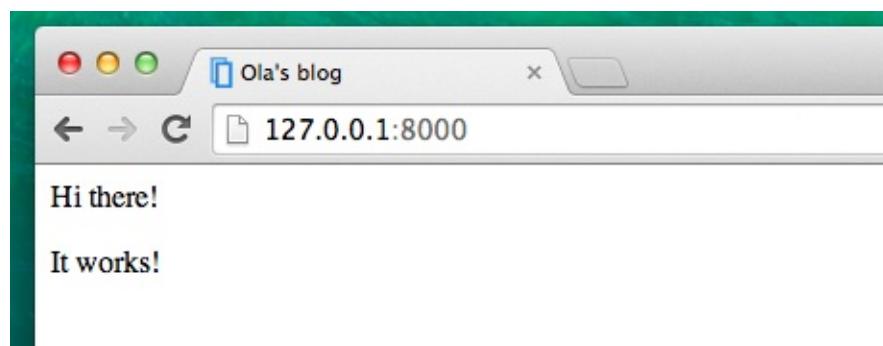
- **head** 是一个包含有关文档且未显示在屏幕上的信息元素。
- **body** 是包含 web 页中所有显示的元素。

我们使用 `<head>` 告诉浏览器这个页面的配置，以及 `<body>` 来告诉它什么实际上是在页面上。

例如，你可以把网页标题元素放到 `<head>` 里，像这样：

```
<html>
  <head>
    <title>Ola's blog</title>
  </head>
  <body>
    <p>Hi there!</p>
    <p>It works!</p>
  </body>
</html>
```

保存该文件并刷新您的网页。



注意浏览器怎么理解 "Ola's blog" 是你的网页的标题的呢？ 它解释了 `<title> Ola's blog </title>` 并将其放置到您的浏览器（它也用于书签，等等）的标题栏中。

可能你也注意到每个开始标记匹配的结束标记，用 `/` 和元素是嵌套（即只有您不能关闭特定标记，直到在里面的所有标记都关闭了）。

这就像把东西放进盒子里。你有一个大箱子里，`<html></html>`；在它里面还有 `<body></body>`，并包含更小盒：`<p></p>`。

你需要遵循这些规则即 关闭 标签和 嵌套 的元素 —— 如果你不这样做，浏览器可能不能正确地解释它们以及您的页面将显示不正确。

自定义您的模板

现在，你可以找点乐子，尝试自定义您的模板！为此这里有几个有用的标签：

- `<h1>A heading</h1>` - 为你最重要的标题
- `<h2>A sub-heading</h2>` 为下一层级的标题！
- `<h3>A sub-sub-heading</h3>` ... 同样，直到 `<h6>`
- `text` 强调您的文本
- `text` 强烈强调您的文本
- `
` 跳转到下一行（你不能放任何东西在 `br` 里面）
- `link` 创建一个链接
- `first itemsecond item` 产生一个列表，就像这样！
- `<div></div>` 定义页面上的一个段

下面是模板的一个完整示例：

```
<html>
  <head>
    <title>Django Girls blog</title>
  </head>
  <body>
    <div>
      <h1><a href="">Django Girls Blog</a></h1>
    </div>

    <div>
      <p>published: 14.06.2014, 12:14</p>
      <h2><a href="">My first post</a></h2>
      <p>Aenean eu leo quam. Pellentesque ornare sem lacinia quam venenatis vestibulum. Donec id elit non mi po
      rta gravida at eget metus. Fusce dapibus, tellus ac cursus commodo, tortor mauris condimentum nibh, ut fermentum mass
      a justo sit amet risus.</p>
    </div>

    <div>
      <p>published: 14.06.2014, 12:14</p>
      <h2><a href="">My second post</a></h2>
      <p>Aenean eu leo quam. Pellentesque ornare sem lacinia quam venenatis vestibulum. Donec id elit non mi po
      rta gravida at eget metus. Fusce dapibus, tellus ac cursus commodo, tortor mauris condimentum nibh, ut f.</p>
    </div>
  </body>
</html>
```

我们已经创建了三个不同的 `div` 部分。

- 第一个 `div` 元素包含我们的博客的标题 —— 这是一个标题和链接
- 另两个 `div` 元素包含我们博客文章发表的日期，`h2` 是可点击文章标题和两个 `p`（段落）的文本、日期和我们的博客。

它给了我们这种效果：



耶！但到目前为止，我们的模板永远只显示完全相同的信息——而我们早些时候谈到模板作为使我们能够以相同的格式显示不同的信息。

我们想要的真正想要做的是的显示实际添加到我们Django admin里面的文章：这是我们接下来要做的。

还有一件事：部署！

我们把这些成果放到网上一定很棒，对吧？让我们再来一次 PythonAnywhere 部署：

提交并推送代码到 GitHub

首先，让我们看看上次部署之后什么文件改变了（运行这些本地命令，不是在 PythonAnywhere 上）：

```
$ git status
```

请确保你在 `djangogirls` 目录中，让我们告诉 `git` 包括此目录内的所有更改：

```
$ git add --all .
```

注 `-A`（简称“全部”）意味着，`git` 也会认识到如果你已经删除的文件（默认情况下，它只能识别新的/已修改的文件）。此外记得（在第 3 章）`.` 意味着当前目录。

在我们上传的所有文件之前，让我们检查 `git` 将上传什么（所有 `git` 将上传的文件现在应以绿色显示）：

```
$ git status
```

我们已经差不多完成了，现在是时候告诉它要在它的历史记录中保存此更改。我们将要给它附上一条“提交消息”用来描述我们做了什么改动。您可以在这个阶段键入任何您想要的东西，但写一些描述性的东西更有用，因为它能在将来使你记起你做了什么。

```
$ git commit -m "Changed the HTML for the site."
```

注意 请确保您使用双引号括提交消息。

做完这些，我们上传（push）改动到 GitHub：

```
git push
```

Pull 把你的新代码拉到 **PythonAnywhere**，然后重新加载你的 **web** 应用程序

- 打开 [PythonAnywhere consoles page](#) 并转到你的 **Bash console**（或启动一个新的）。然后，运行：

```
$ cd ~/my-first-blog
$ source myvenv/bin/activate
(myvenv)$ git pull
[...]
(myvenv)$ python manage.py collectstatic
[...]
```

然后观察你的代码被下载下来。如果你想要检查他是否正确下载，你可以跳转到**Files Tab** 并在PythonAnywhere上查看代码。

- 最后，跳到 [Web tab](#) 并点击对应你的 Web 应用程序的 **Reload**。

你的更新应已上线！刷新你的浏览器，看到更新了吧 :)

Django ORM 和 QuerySets (查询集)

在这一章中，你将学习 Django 如何连接到数据库，并将数据存储在里面。一探究竟吧！

QuerySet 是什么呢？

从本质上说，QuerySet 是给定模型的对象列表（list）。QuerySet 允许您从数据库中读取数据，对其进行筛选以及排序。

用例子来学习最容易的了。让我们试试这个，好吗？

Django shell

打开你本地的终端(不是在Python解析器里面) 然后输入这个命令：

```
(myvenv) ~/djangogirls$ python manage.py shell
```

效果应该像这样：

```
(InteractiveConsole)
>>>
```

你现在在 Django 的交互式控制台中。它是就像 Python 提示符，但有一些额外神奇的 Django 特性：）。你当然也可以在这里使用所有的 Python 命令。

所有对象

首先让我们尝试显示所有我们的文章。你可以用下面的命令：

```
>>> Post.objects.all()
Traceback (most recent call last):
  File "<console>", line 1, in <module>
NameError: name 'Post' is not defined
```

哎呀！出现了一个错误。它告诉我们没有文章。这是正确的——我们忘了首先导入它！

```
>>> from blog.models import Post
```

这很简单：我们从 `blog.models` 导入 `Post` 的模型。让我们试着再一次显示所有的帖子：

```
>>> Post.objects.all()
<QuerySet [<Post: my post title>, <Post: another post title>]>
```

这是我们之前创建的文章的 list 列表！我们通过使用Django admin界面创建了这些文章。但是我们现在想通过Python来创建新的文章，那么我们应该如何做呢？

创建对象

这就是你如何在数据库创建一个新的Post对象的方法：

```
>>> Post.objects.create(author=me, title='Sample title', text='Test')
```

但是我们这里有一个遗漏的要素：`me`。我们需要传递 `User` 模型的实例作为作者。如何做到这一点？

让我们首先导入用户模型：

```
>>> from django.contrib.auth.models import User
```

我们在数据库中有哪些用户？试试这个：

```
>>> User.objects.all()
<QuerySet [<User: ola>]>
```

这是一个我们之前创建的超级用户！让我们现在获取一个用户实例：

```
me = User.objects.get(username='ola')
```

正如你所看到的，我们现在 `get` 一个 `username` 等于 'ola' 的 `User`。简单吧！当然，你必须将其改为你所使用的用户名。

现在我们终于可以创建我们的文章了：

```
>>> Post.objects.create(author=me, title='Sample title', text='Test')
```

哈哈！要检查是否有效吗？

```
>>> Post.objects.all()
<QuerySet [<Post: my post title>, <Post: another post title>, <Post: Sample title>]>
```

就是这样，又一个文章在列表里面！

添加更多文章

你现在可以找点乐子，并添加更多的帖子，看它是如何工作。添加 2-3 个并前进到下一个部分。

筛选对象

`QuerySets` 的很大一部分功能是对它们进行筛选。譬如，我们想要发现所有都由用户 `ola` 编写的文章。我们将使用 `filter`，而不是 `all` 在 `Post.objects.all()`。我们需要在括号中申明哪些条件，以在我们的 `queryset` 结果集中包含一篇博客文章。在我们的情况是 `author`，它等于 `me`。把它写在 `Django` 的方式是：`author = me`。现在我们的代码段如下所示：

```
>>> Post.objects.filter(author=me)
<QuerySet [<Post: Sample title>, <Post: Post number 2>, <Post: My 3rd post!>, <Post: 4th title of post>]>
```

或者，也许我们想看到包含在 `title` 字段标题的所有帖子吗？

```
>>> Post.objects.filter(title__contains='title')
<QuerySet [<Post: Sample title>, <Post: 4th title of post>]>
```

注在 `title` 与 `contains` 之间有两个下划线字符（`_`）。`Django` 的 `ORM` 使用此语法来分隔字段名称（"title"）和操作或筛选器（"contains"）。如果您只使用一个下划线，您将收到类似"FieldError：无法解析关键字 title_contains"的错误。

你也可以获取一个所有已发布文章的列表。我们通过筛选所有含 `published_date` 为过去时间的文章来实现这个目的：

```
from django.utils import timezone
Post.objects.filter(published_date__lte=timezone.now())
```

不幸的是，通过 Python 终端添加的文章还没发布。我们可以改变它！首先获取一个我们想要发布的文章实例：

```
>>> post = Post.objects.get(title="Sample title")
```

然后将它与我们 `publish` 的方法一起发布！

```
>>> post.publish()
```

现在再一次尝试获取已发布的文章（按向上箭头按钮三次，然后按回车）：

```
>>> Post.objects.filter(published_date__lte=timezone.now())
<QuerySet [<Post: Sample title>]>
```

对象排序

`Queryset` 还允许您排序结果集对象的列表。让我们试着让它们按 `created_date` 字段排序：

```
>>> Post.objects.order_by('created_date')
<QuerySet [<Post: Sample title>, <Post: Post number 2>, <Post: My 3rd post!>, <Post: 4th title of post>]>
```

我们也可以在开头添加 `-` 来反向排序：

```
>>> Post.objects.order_by('-created_date')
<QuerySet [<Post: 4th title of post>, <Post: My 3rd post!>, <Post: Post number 2>, <Post: Sample title>]>
```

链式 `QuerySets`

你可以通过链式调用连续组合`QuerySets`

```
>>> Post.objects.filter(published_date__lte=timezone.now()).order_by('published_date')
```

这真是强而有力，而且可以让你写较复杂的查询

酷！现在，你准备下一个部分！若要关闭shell程序，请键入这：

```
>>> exit()
$
```

模板中的动态数据

我们已有的几件：`Post` 模型定义在 `models.py` 中，我们有 `post_list` `views.py` 和添加的模板中。但实际上我们如何使我们的帖子出现在我们的 HTML 模板上呢？因为那是我们所想要的：获取一些内容（保存在数据库中的模型）然后在我们的模板中很漂亮的展示，对吗？

这就是 `views` 应该做的：连接模型和模板。在我们的 `post_list` 视图中我们需要获取我们想要显示的模型，并将它们传递到模板中去。所以基本上在视图中，我们决定什么（模型）将显示在模板中。

好吧，我们将如何实现它呢？

我们需要打开我们的 `blog/views.py`。到目前为止 `post_list` view 看起来像这样：

```
from django.shortcuts import render

def post_list(request):
    return render(request, 'blog/post_list.html', {})
```

还记得当我们谈论过导入在不同文件中编写的代码吗？现在是我们必须导入我们已经写在 `models.py` 里的模型的时候了。我们将添加这行 `from .models import Post`，像这样：

```
from django.shortcuts import render
from .models import Post
```

`from` 后面的点号意味着当前目录或当前的应用程序。因为 `views.py` 和 `models.py` 是在同一目录中，我们只需要使用`.` 和文件的名称（无 `.py`）。然后我们导入模型（`Post`）。

但接下来是什么呢？为了让实际的博客帖子从 `Post` 模型里获取，我们需要一种叫做 `QuerySet` 的东西。

QuerySet 查询集

您应该已经熟悉 `Queryset` 是如何工作的。我们在 [Django ORM \(Queryset\) 章节](#) 谈论过它。

所以现在我们对已经发表并进行由 `published_date` 排序的博客列表感兴趣，对吗？我们已经在 `QuerySets` 查询集一节里这么干过！

```
Post.objects.filter(published_date__lte=timezone.now()).order_by('published_date')
```

现在我们把这段代码插入 `blog/views.py` 文件，添加到函数 `def post_list(request)` 里去：

```
from django.shortcuts import render
from django.utils import timezone
from .models import Post

def post_list(request):
    posts = Post.objects.filter(published_date__lte=timezone.now()).order_by('published_date')
    return render(request, 'blog/post_list.html', {})
```

请注意我们为这里的 `QuerySet` 查询集创建了一个变量：`posts`。将此视为我们的 `QuerySet` 的名字。从现在开始我们可以通过这个名字引用它。

同时，代码中使用了 `timezone.now()` 函数，因此我们需要添加一个导入 `timezone`。

最后还没有完成的部分是传递 `posts` 查询集到模板中（我们将在下一章中介绍如何显示它）。

在 `render` 函数中我们已经有了 请求（因此我们通过互联网从用户接收的一切）和模板文件 `'blog/post_list.html'` 参数。最后一个参数，看起来像这样：`{}`，我们可以在其中添加一些模板要使用的东西。我们需要给它们起名字（我们暂且沿用 `'posts' :)`）。它应该看起来像这样：`{'posts': posts}`。请注意，`:` 之前的部分是字符串；你需要将它用引号包围 `''`。

所以最后我们的 `blog/views.py` 文件应如下所示：

```
from django.shortcuts import render
from django.utils import timezone
from .models import Post

def post_list(request):
    posts = Post.objects.filter(published_date__lte=timezone.now()).order_by('published_date')
    return render(request, 'blog/post_list.html', {'posts': posts})
```

就是它！现在回到我们的模板并显示此QuerySet查询集！

如果你想了解更多关于QuerySet的内容，那么你可在这里得到帮助：<https://docs.djangoproject.com/en/1.8/ref/models/querysets/>

Django模板

是时候把数据展示出来了！Django提供了一个非常有用的内置来实现---模板标签

什么是模板标签呢？

正如你在前面章节中所了解的那样，我们并不能将 Python 代码嵌入到HTML中。因为浏览器不能识别 Python 代码，它只能解析HTML。我们知道，HTML是静态页面，而 Python 则显得更加动态。

Django模板标签允许我们将Python之类的内容翻译成HTML，所以你可以更快更简单的建立动态网站。哈哈！

展现文章列表模板

在之前的章节，我们给我们的模板一系列文章在 `post` 变量里。现在我们将在HTML里展现它。

为了用模板标签在HTML中显示变量，我们会使用两个大括号，并将变量包含在里面，正如这样

```
{{ posts }}
```

在你的 `blog/templates/blog/post_list.html` 文件中进行如下的操作。将所有 `<div> to the third </div>` 中的 `to the third` 用 `{{ posts }}` 代替。并保存文件，刷新页面后去看看我们做的那些改变。



如你所见，我们得到如下：

```
<QuerySet [<Post: My second post>, <Post: My first post>]>
```

这意味着Django视它为对象的列表。还记得在 **Python入门介绍** 里我们怎么展现列表的吗？是的，我们可以使用循环！在 django模板中使用循环去遍历它们。如下所示：

```
{% for post in posts %}
    {{ post }}
{% endfor %}
```

在你的模板里试试这个。

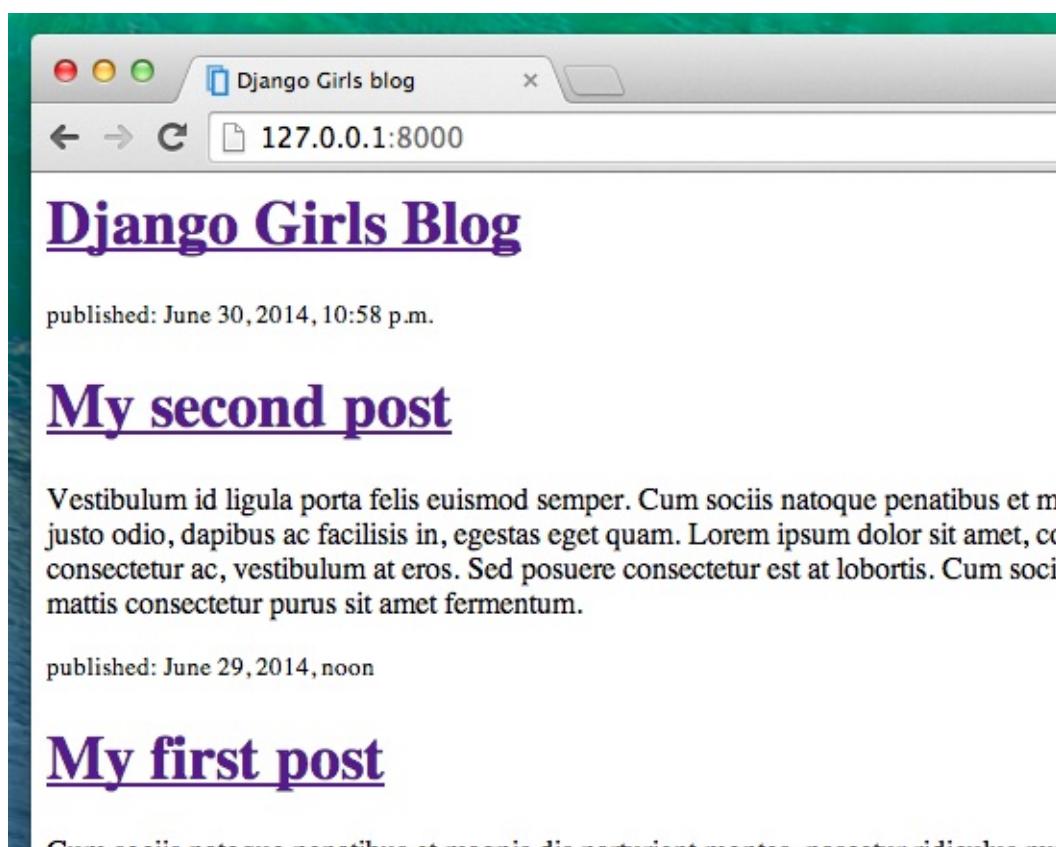


它工作了！但是想让他们展现的像我们之前在HTML介绍章节里创建的静态文章一样 你可以混合HTML和模板标签。我们的 body 将长得像这样：

```
<div>
    <h1><a href="/">Django Girls Blog</a></h1>
</div>

{% for post in posts %}
<div>
    <p>published: {{ post.published_date }}</p>
    <h1><a href="{{ post.title }}>{{ post.title }}</a></h1>
    <p>{{ post.text|linebreaksbr }}</p>
</div>
{% endfor %}
```

所有的在 `{% for %}` 和 `{% endfor %}` 之间的内容将会被Django对象列表中的每个对象所代替。刷新页面去看看：



Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus

你注意到这次我们使用了一个明显不同的标记 `{{ post.title }}` 或 `{{ post.text }}`？我们正在访问定义在 Post 模型中的每一个域。此外，`|linebreaksbr` 通过一个过滤器，使得行间隔编程段落。

还有一件事

如果我们将我们的网站放在互联网上运行，那将是一件很不错的的事情，难道不是吗？让我们试着再次部署到 PythonAnywhere。简单部署步骤如下……

- 首先，我们将我们的代码放到Github

```
$ git status  
[...]  
$ git add --all .  
$ git status  
[...]  
$ git commit -m "Modified templates to display posts from database."  
[...]  
$ git push
```

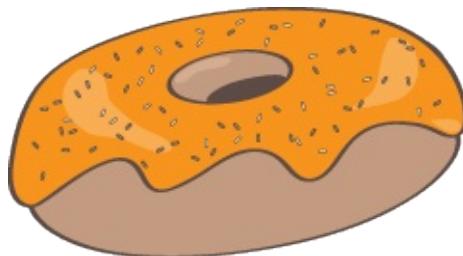
- 然后，重新登陆 [PythonAnywhere](#) 并进入**Bash** 控制台（或重开一个），并运行：

```
$ cd my-first-blog  
$ git pull  
[...]
```

- 最后，我们返回 [Web tab](#) 重新加载我们的应用程序，此时我们应该可以看到更新后的程序运行情况了。

祝贺你！现在往前走，尝试在你的Django管理中添加一篇新文章（记得添加发布日期！），然后刷新你的页面看看是否文章正常显示了。

这一定是个让人沉醉的作品？为此我们应当骄傲，在计算机学科的一点儿进步，都是自我的一次突破 :)。



CSS - 让它更好看！

我们的博客看起来仍然很丑吧？是时候来让它变得更好看了！为此，我们将引入CSS。

什么是 CSS？

层叠样式表(Cascading Style Sheets)是一种语言，用来描述使用标记语言(如HTML)写成的网站的外观和格式。把它当做我们网站的化妆;)。

但我们不想总是从零开始，对吧？我们将会再一次采用程序员们编写并发布到互联网上的免费玩意。重新发明轮子十分无趣，你懂的。

让我们用 Bootstrap 吧！

Bootstrap是最流行的HTML和CSS框架之一，它可以用来开发炫酷的网站:<https://getbootstrap.com/>

起初，它由Twitter的程序员编写，现在由来自世界各地的志愿者写的。

安装 Bootstrap

若要安装Bootstrap，您需要将它添加到你的 `.html` 文件的 `<head>` 中：

`blog/templates/blog/post_list.html`

```
<link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap.min.css">
<link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap-theme.min.css">
```

这不会向你的项目添加任何文件。它只是指向在互联网上存在的文件。只需要继续打开您的网站并刷新页面。你瞧！



已经变得更好看了！

Django 中的静态文件

最后我们来看看这些我们叫做静态文件的东西。静态文件是指你所有的CSS文件和图片文件，这些文件不是动态的，对所有用户都是一样的，不会因为请求内容而发生变化。

Django的静态文件放在哪儿呢？

Django已经知道到何处找到内建“admin”应用的静态文件。现在我们只需要给我们的 `blog` 应用添加一些静态文件。

我们在`blog`应用的目录下创建一个名为 `static` 的文件夹，创建后目录结构如下：

```
djangogirls
├── blog
│   ├── migrations
│   └── static
│       └── templates
└── mysite
```

Django会自动找到你应用文件夹目录下所有名字叫“static”的文件夹，并能够使用其中的静态文件。

你的第一个 CSS 文件！

现在让我们创建一个 CSS 文件，为了在您的 web 页中添加你自己的风格。在 `static` 的目录下创建一个新的目录称为 `css`。然后，在这个 `css` 目录里创建一个新的文件，称为 `blog.css`。准备好了吗？

```
djangogirls
└── blog
    └── static
        └── css
            └── blog.css
```

是时候来写一些CSS了！首先用你的代码编辑器打开 `blog/static/css/blog.css`。

在这里我们不会太深入学习自定义和关于 CSS 的内容，因为这些内容很容易而且您可以在本教程结束后自行学习。这页的最后会推荐一个免费课程来学习更多相关的内容。

但至少要做一点吧。也许我们可以改变我们标题的颜色？计算机使用特殊的编码来了解颜色。它们以 `#` 开始，后面跟着 6 个字母（A 到 F）或数字（0-9）。例如，蓝色的的编码是 `#0000FF` 你在这里可以发现许多颜色的编码：

<http://www.colorpicker.com/>。你也可以使用 预先定义的颜色，如 `red` 和 `green`。

在你的 `blog/static/css/blog.css` 文件中添加下面的代码：

```
h1 a {
    color: #FCA205;
}
```

`h1 a` 是CSS选择器。这意味着我们要将我们的样式应用于在 `h1` 元素的任何 `a` 元素。所以，当我们的代码中有类似 `<h1>link</h1>` 的代码，上述的样式就会被应用。在这种情况下，我们会告诉它要改变其颜色为 `#FCA205`，它是橙色的。当然，你可以把自己的颜色放在这里！

在 CSS 文件中，我们决定HTML文件中元素的样式。我们标识元素的第一种方式是用元素名。你可能把来自于HTML段落的这些元素名当做了标签记忆。例如 `a`，`h1`，`body`，这些都是元素名。我们也用属性去标识元素，如 `class` 或属性 `id`。类和 `id` 是你自己给该元素的命名。类定义元素组，并指向特定元素的 `id`。例如，可以使用标记名称 `a`、类 `external_link` 或 `id` `link_to_wiki_page` 标识以下标签：

```
<a href="https://en.wikipedia.org/wiki/Django" class="external_link" id="link_to_wiki_page">
```

可以在 [CSS Selectors in w3schools](#) 了解更多。

我们还需要告诉我们的 HTML 模板，我们添加了一些 CSS。打开 `blog/templates/blog/post_list.html` 文件并在文件最开始的地方添加以下代码：

```
{% load staticfiles %}
```

我们刚刚加载了静态文件到这里（译者注：这里实际上是为模板引入staticfiles相关的辅助方法）：）。然后，在 `<head>` 和 `</head>` 之间，在Bootstrap的CSS文件的引导之后添加以下行：

```
<link rel="stylesheet" href="{% static 'css/blog.css' %}">
```

浏览器按照给定文件的顺序读取文件，所以我们需要确保代码在正确的位置，否则我们文件中的代码可能会覆盖Bootstrap文件中的代码。我们只是告诉我们的模板我们的 CSS 文件所在的位置。

现在，您的文件应该像这样：

```
{% load staticfiles %}
<html>
  <head>
    <title>Django Girls blog</title>
    <link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap.min.css">
    <link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap-theme.min.css">
    <link rel="stylesheet" href="{% static 'css/blog.css' %}">
  </head>
  <body>
    <div>
      <h1><a href="/">Django Girls Blog</a></h1>
    </div>

    {% for post in posts %}
      <div>
        <p>published: {{ post.published_date }}</p>
        <h1><a href="{{ post.title }}>{{ post.title }}</a></h1>
        <p>{{ post.text|linebreaksbr }}</p>
      </div>
    {% endfor %}
  </body>
</html>
```

好的保存该文件并刷新网站！



将它添加到你的 CSS 代码，保存该文件并查看它如何工作！

Django Girls Blog

published: June 30, 2014, 10:58 p.m.

My second post

Vestibulum id ligula porta felis euismod semper. Cum sociis natoque p

也许我们可以在我们的头中自定义字体吗？粘贴到你的 `< head >` 在 `blog/templates/blog/post_list.html` 文件中：

```
<link href="//fonts.googleapis.com/css?family=Lobster&subset=latin,latin-ext" rel="stylesheet" type="text/css">
```

这行将从谷歌的字体 (<https://www.google.com/fonts>) 中导入称为 龙虾的字体。

现在添加一行 `font-family: 'Lobster';` 到CSS文件 `blog/static/css/blog.css` 的 `h1 a` 声明块中(花括弧 `{` 与 `}` 之间的代码)，然后刷新页面：

```
h1 a {
    color: #FCA205;
    font-family: 'Lobster';
}
```

Django Girls Blog

published: June 30, 2014, 10:58 p.m.

My second post

Vestibulum id ligula porta felis euismod semper. Cum sociis natoque p

太棒了！

如上文所述，CSS 有一个概念叫做类，其中基本上允许您命名的 HTML 代码的一部分并只对这部分应用样式，不会影响其他部分。它是超级有用的如果你有两个 `div`，但他们有些很多不一样（如您的标题和你的帖子）。类可以帮你让它们看起来不同。

继续命名部分HTML 代码。添加一个称为 `page-header` 的类到您的 `div` 中，其中包含您的标题，像这样：

```
<div class="page-header">
    <h1><a href="/">Django Girls Blog</a></h1>
</div>
```

和现在将包含一篇博客文章的类 `post` 添加到您的 `div`。

```
<div class="post">
    <p>published: {{ post.published_date }}</p>
    <h1><a href="{{ post.title }}>{{ post.title }}</a></h1>
    <p>{{ post.text|linebreaksbr }}</p>
</div>
```

现在，我们将向不同的选择器添加声明块。选择器以`.`开始，关联到类。网络上有很多很棒的CSS教程以及相关解释，帮助您理解下面的代码。至于现在，就简单地复制粘贴到你的`blog/static/css/blog.css`文件中吧。

```
.page-header {
    background-color: #ff9400;
    margin-top: 0;
    padding: 20px 20px 20px 40px;
}

.page-header h1, .page-header h1 a, .page-header h1 a:visited, .page-header h1 a:active {
    color: #ffffff;
    font-size: 36pt;
    text-decoration: none;
}

.content {
    margin-left: 40px;
}

h1, h2, h3, h4 {
    font-family: 'Lobster', cursive;
}

.date {
    float: right;
    color: #828282;
}

.save {
    float: right;
}

.post-form textarea, .post-form input {
    width: 100%;
}

.top-menu, .top-menu:hover, .top-menu:visited {
    color: #ffffff;
    float: right;
    font-size: 26pt;
    margin-right: 20px;
}

.post {
    margin-bottom: 70px;
}

.post h1 a, .post h1 a:visited {
    color: #000000;
}
```

然后将文章的HTML代码用类声明包裹起来。替换以下内容：

```
{% for post in posts %}
    <div class="post">
        <p>published: {{ post.published_date }}</p>
        <h1><a href="">{{ post.title }}</a></h1>
        <p>{{ post.text|linebreaksbr }}</p>
    </div>
{% endfor %}
```

在 `blog/templates/blog/post_list.html` 是这样的：

```
<div class="content container">
    <div class="row">
        <div class="col-md-8">
            {% for post in posts %}
                <div class="post">
                    <div class="date">
                        {{ post.published_date }}
                    </div>
                    <h1><a href="">{{ post.title }}</a></h1>
                    <p>{{ post.text|linebreaksbr }}</p>
                </div>
            {% endfor %}
        </div>
    </div>
</div>
```

保存这些文件并刷新您的网站。



呜呼！看起来棒极了，是吧？看看我们刚在HTML粘贴加入的类以及在CSS中使用的代码。如果你想要时间变成绿松石色，你该在哪里做改变呢？

不要害怕摆弄这个 CSS，并试图去改变一些东西。调整CSS可以帮你搞明白不同的东西怎样工作。如果你做错了什么东西，别担心，您总是可以撤消它！

我们非常建议在线免费学习这个 [Codecademy HTML 和 CSS 课程](#)。它可以帮你学到一切你需要知道的有关如何通过CSS使你的网站更漂亮。

准备好下一章了吗?:)

模板扩展

另一个有趣的事情Django已经为你做好了就是模板扩展。这是什么意思呢？它意味着你可以使用你的HTML相同代码为你网站不同的网页共享。

通过这种方法，当你想使用同样的信息或布局，或者你想改变某些模板内容时，你不必在每个文件中都重复着相同的代码。你仅仅只需要改变一个文件，而不是所有的。

创建一个基础模板

一个基础模板是最重要的模板，你扩展到你网站的每一页。

让我们创建一个 `base.html` 文件到 `blog/templates/blog/` :

```
blog
└──templates
    └──blog
        base.html
        post_list.html
```

然后将它打开，从 `post_list.html` 中复制所有东西到 `base.html` 文件，就像这样：

```
% load staticfiles %
<html>
    <head>
        <title>Django Girls blog</title>
        <link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap.min.css">
        <link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap-theme.min.css">
        <link href='//fonts.googleapis.com/css?family=Lobster&subset=latin,latin-ext' rel='stylesheet' type='text/css'>
        <link rel="stylesheet" href="{% static 'css/blog.css' %}">
    </head>
    <body>
        <div class="page-header">
            <h1><a href="/">Django Girls Blog</a></h1>
        </div>

        <div class="content container">
            <div class="row">
                <div class="col-md-8">
                    {% for post in posts %}
                        <div class="post">
                            <div class="date">
                                {{ post.published_date }}
                            </div>
                            <h1><a href="{{ post.title }}>{{ post.title }}</a></h1>
                            <p>{{ post.text|linebreaksbr }}</p>
                        </div>
                    {% endfor %}
                </div>
            </div>
        </body>
    </html>
```

然后在 `base.html` 中，替换你所有的 `<body>` (所有的在 `<body>` 和 `</body>` 之间的内容)像这样：

```
<body>
    <div class="page-header">
        <h1><a href="/">Django Girls Blog</a></h1>
    </div>
    <div class="content container">
        <div class="row">
            <div class="col-md-8">
                {% block content %}
                {% endblock %}
            </div>
        </div>
    </div>
</body>
```

用如下内容替换所有在 `{% for post in posts %}{% endfor %}` 之间的代码：

```
{% block content %}
{% endblock %}
```

这是什么意思呢？你刚刚创建了一个 `block`（块），这个模板标签允许你在其中插入扩展自 `base.html` 的HTML代码。我们一会儿将给你展示这个如何使用。

现在保存它，然后再次打开你的 `blog/templates/blog/post_list.html`。删除一切 `body` 外的代码，然后删除 `<div class="page-header"></div>`，此时文件会看起来像这样：

```
{% for post in posts %}
    <div class="post">
        <div class="date">
            {{ post.published_date }}
        </div>
        <h1><a href="">{{ post.title }}</a></h1>
        <p>{{ post.text|linebreaksbr }}</p>
    </div>
{% endfor %}
```

然后现在将这行加到文件的开始：

```
{% extends 'blog/base.html' %}
```

这意味着我们在 `post_list.html` 模板文件中扩展了 `base.html` 模板的内容。还有一件事：将所有(除了我们刚刚加入的那一行)内容置于 `{% block content %}` 和 `{% endblock %}` 之间。。像这样：

```
{% extends 'blog/base.html' %}

{% block content %}
    {% for post in posts %}
        <div class="post">
            <div class="date">
                {{ post.published_date }}
            </div>
            <h1><a href="">{{ post.title }}</a></h1>
            <p>{{ post.text|linebreaksbr }}</p>
        </div>
    {% endfor %}
{% endblock %}
```

好了，就是它了！检查你的网站还能正常工作：）

如果你有任何错误 `TemplateDoesNotExist` 这意味着没有 `blog/base.html` 文件，你需要 `runserver` 运行在控制台，尝试去关掉它（通过按下 `Ctrl+C` -Control和C按钮一切）然后重新运行 `python manage.py runserver` 命令行。

扩展您的应用

我们已经完成了所有创建网站的各项不同必须的步骤：我们知道如何写一个模型，url，视图和模板。我们同样知道如何让我们网站更漂亮。

现在来练习吧！

我们网站里的第一件事情就是，一个展现一篇博客的页面，对吗？

我们已经有了 Post 模型，所以我们不需要再添加任何内容到 models.py 文件中。.

创建一个模板链接，跳转到博文的内容页

我们将从在 blog/templates/blog/post_list.html 里添加一个链接开始。目前它应该看起来像这样：

```
{% extends 'blog/base.html' %}

{% block content %}
    {% for post in posts %}
        <div class="post">
            <div class="date">
                {{ post.published_date }}
            </div>
            <h1><a href="">{{ post.title }}</a></h1>
            <p>{{ post.text|linebreaksbr }}</p>
        </div>
    {% endfor %}
{% endblock %}
```

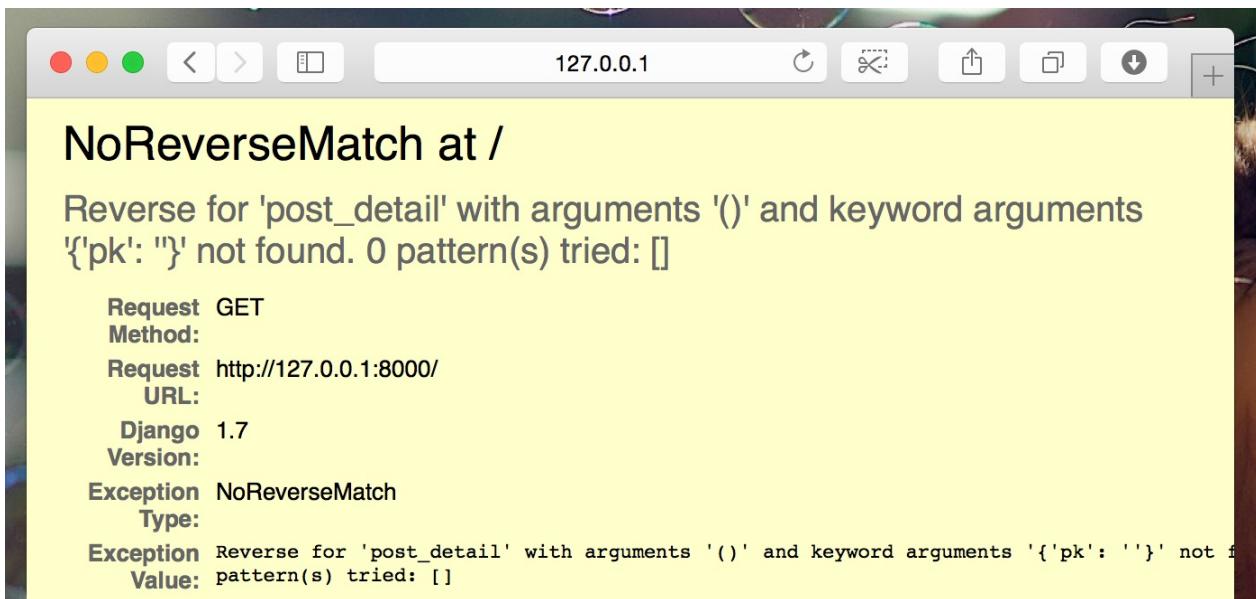
我们在博文列表的博文标题处添加一个链接用于跳转到该博文的详细页面。让我们编辑 `<h1>{{ post.title }}</h1>` 使得它能链接到博文详情页面：。

```
<h1><a href="{% url 'post_detail' pk=post.pk %}">{{ post.title }}</a></h1>
```

是解释神秘的 `{% url 'post_detail' pk=post.pk %}` 时候了。正如你所猜想的，`{% %}` 标记意味着我们正在使用Django模板标签。这一次我们将使用一个能为我们创建URL的！

`blog.views.post_detail` 是我们想创建的 `post_detail view` 的路径。请注意：`blog` 是我们应用的名字 (`blog` 目录)，`views` 是表单 `views.py` 文件的名字同时最后一个部分 - `post_detail` - 是 `view` 的名字。.

现在当我们访问：<http://127.0.0.1:8000/> 我们会得到一个错误（这是预料之中的，因为我们没有名为 `post_detail` 的 URL 或视图）。看起来会像这样：



创建文章详细页面的URL

让我们在 `urls.py` 里为我们的 `post_detail` view 创建一个URL!

我们希望我们的第一条文章详细页面显示在类似这样的URL：<http://127.0.0.1:8000/post/1/>

让我们在 `blog/urls.py` 文件中增加一个 URL 来指引 Django 到名为 `post_detail` 的 view，它将用来显示整篇博客文章。增加这行 `url(r'^post/(?P<pk>[0-9]+)/$', views.post_detail, name='post_detail')`，到 `blog/urls.py` 文件。文件应当如下所示：

```
from django.conf.urls import url
from . import views

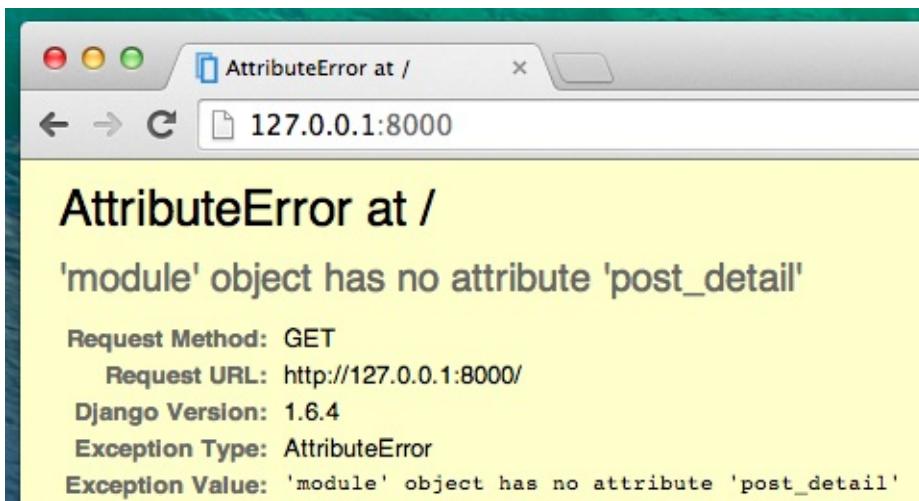
urlpatterns = [
    url(r'^$', views.post_list, name='post_list'),
    url(r'^post/(?P<pk>[0-9]+)/$', views.post_detail, name='post_detail'),
]
```

`^post/(?P<pk>[0-9]+)/$` 部分看上去很骇人，请勿担心—我们来解释给你听：- 还是起始自 `^` — "开头部分"- `post/` 只是表示，紧接着开头，URL 应当包含 `post` 和 `/`。到目前为止，一切都好。- `(?P<pk>[0-9]+)` - 这部分很棘手。这表示 Django 会把所有你放到这里的东西转变成一个称作 `pk` 的变量并传递给视图。`[0-9]` 也告诉我们它只能由数字，而不是字母（所以是取值范围是介于0和9之间）。`+` 意味着需要一个或更多的数字。所以诸如 `http://127.0.0.1:8000/post//` 是无效的，但是像 `http://127.0.0.1:8000/post/1234567890/` 是完全OK的！- `/` - 然后我们再次需要 `/`- `$` - "结尾"！

这意味着如果你键入 `http://127.0.0.1:8000/post/5/` 到你的浏览器里，Django 明白你在寻找一个叫做 `post_detail` 的视图，然后传递 `pk` 等于 `5` 到那个视图。

`pk` 是 primary key (主键) 的缩写。在 Django 项目中常常用到这个名字。但是你可以使用你想要的变量（记住：使用小写以及 `_` 而不是空格！）。比如与其说 `(?P<pk>[0-9]+)` 我们可以有 `post_id`，所以这个就像：`(?P<post_id>[0-9]+)`。

好吧，我们已经向 `blog/urls.py` 添加了一个新的 URL 模式！让我们刷新页面：<http://127.0.0.1:8000/> Duang！还有另一个错误！果然！



你还记得下面应该怎么做吗？当然：添加一个视图！

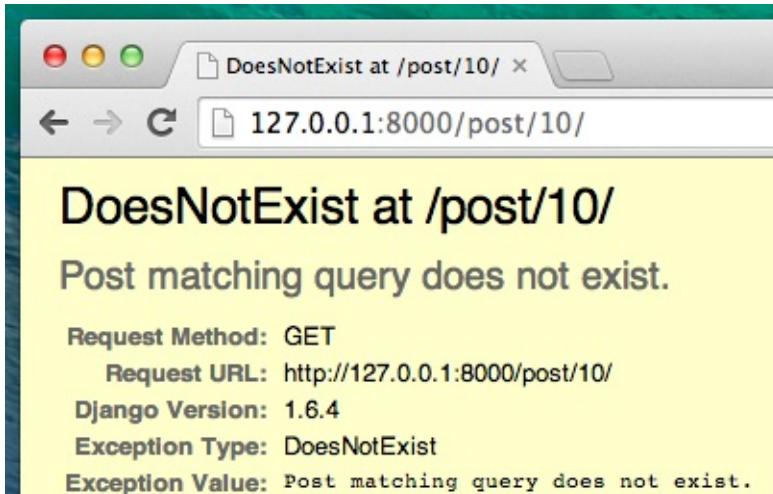
增加文章详细页面的视图

这次我们的视图提供了一个额外的参数 `pk`。我们的视图需要能捕获它，对吗？所以我们将定义我们的函数为 `def post_detail(request, pk):`。注意我们需要使用我们在 `urls` 里指定的 `(pk)`。省略这个变量是不正确的，将会导致一个错误！

现在，我们想要有一个并且只有一个博客帖子。为了做到这一点，我们需要使用下面的请求集合：

```
Post.objects.get(pk=pk)
```

但是这段代码有一个问题。如果没有 `Post` 和给定 主键 `(pk)` 我们将有一个非常丑陋的错误！



我们不希望那样！但是，当然，Django已经为我们处理好了这些：`get_object_or_404`。万一没有 `Post` 和给定的 `pk`，它将展现更多有趣的页面（称作 `Page Not Found 404` 页面）。



好消息是你实际上可以创建你自己 `Page not found` 的页面和使它漂亮如你所愿。但现在它不是超级重要的，所以我们将跳过它。

好吧，是时候将 视图 添加到我们的 `views.py` 文件了！

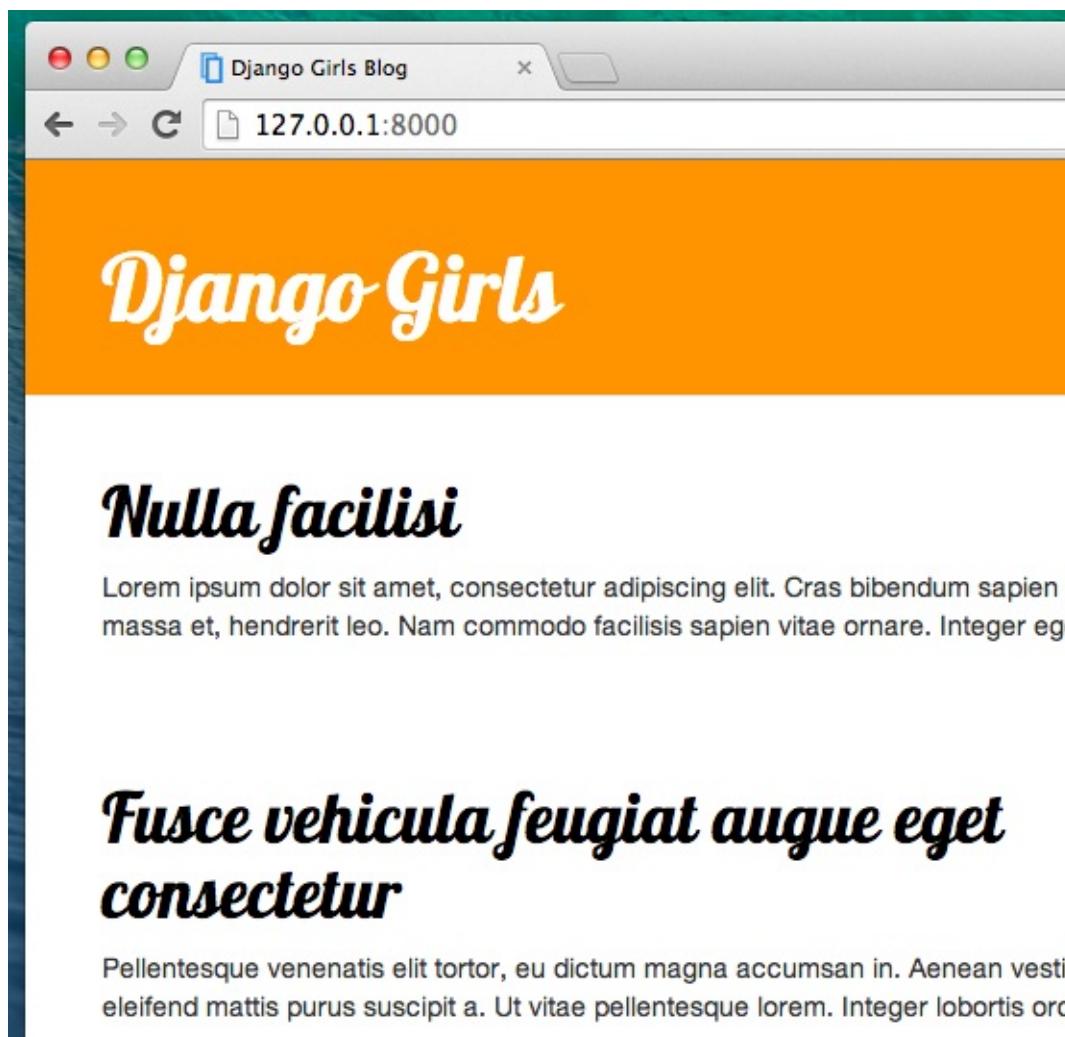
我们应该打开 `blog/views.py` 并添加以下代码：

```
from django.shortcuts import render, get_object_or_404
```

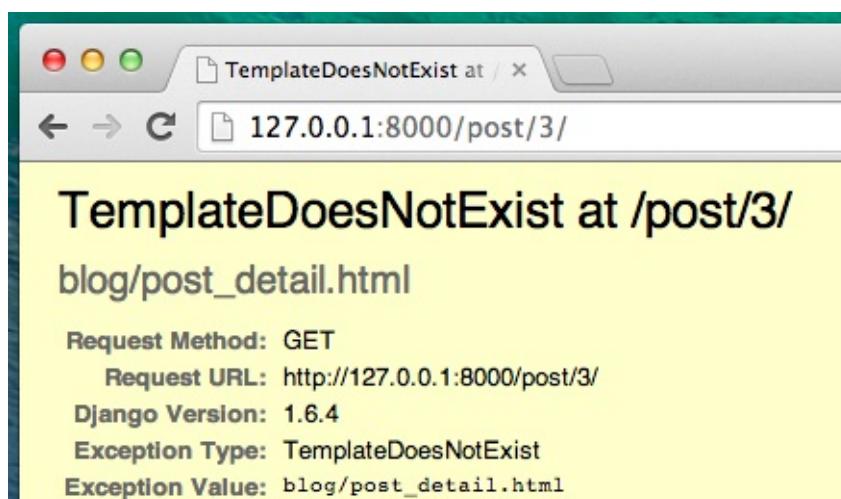
在 `from` 附近行。并在文件的末尾，我们将增加我们的 `view`：

```
def post_detail(request, pk):
    post = get_object_or_404(Post, pk=pk)
    return render(request, 'blog/post_detail.html', {'post': post})
```

好了。是时候刷新页面了：<http://127.0.0.1:8000/>



它工作了！但是，当您单击博客文章标题中的某个链接时，会发生什么呢？



哦不！另一个错误！但我们已经知道如何处理它，对吗？我们需要添加一个模板！

为文章详细页面增加模板

我们将在 `blog/templates/blog` 中创建一个文件，叫做 `post_detail.html`。

它看起来会像这样：

```
{% extends 'blog/base.html' %}

{% block content %}
    <div class="post">
        {% if post.published_date %}
            <div class="date">
                {{ post.published_date }}
            </div>
        {% endif %}
        <h1>{{ post.title }}</h1>
        <p>{{ post.text|linebreaksbr }}</p>
    </div>
{% endblock %}
```

现在，我们要扩展 `base.html`。在 `content` 块中，我们想要显示一篇文章的 `published_date`（如果存在的话），标题和文本。但我们应该讨论一些重要的东西，对不对吗？

`{% if ... %} ... {% endif %}` 是当我们想检查某样东西（还记得Python简介里的 `if ... else ...` 吗？）的时候的一种模板记号。在这个例子中，我们想要检查文章的 `published_date` 不是空的。

好的，我们可以刷新我们的页面并查看是否 `Page not found` 是不是没有了。



耶！它工作了！

还有一件事：部署时刻！

你的网站如果还能在 PythonAnywhere 正常运转就好了，对吧？让我们再次部署。

```
$ git status
$ git add --all .
$ git status
$ git commit -m "Added view and template for detailed blog post as well as CSS for the site."
$ git push
```

- 然后，在一个 [PythonAnywhere](#) 的 Bash 终端里运行：

```
$ cd my-first-blog
$ source myenv/bin/activate
(myenv)$ git pull
[...]
(myenv)$ python manage.py collectstatic
[...]
```

- 最后，跳到 [Web 标签页](#) 并点击重新载入。

就是这样！祝贺你：）

Django表单

我们最后一件关于我们的网站的事情就是创建一个漂亮的方式来增加和编辑博客文章。 Django的管理是很酷，但是它很难去自定义，变得更漂亮。通过 `forms`，我们可以拥有对我们界面绝对的权利—我们能够做几乎我们能想象到的所有事情！

Django表单的一个好处就是我们既可以从零开始自定义，也可以创建 `ModelForm`，它将表单的结果保存到模型里。

这正是我们想做的：我们将为我们自己的 `Post` 模型创建一个表单。

就像所有Django的重要部分一样，表单有他们自己的文件 `forms.py`。

我们需要创建一个文件，把它的名字放在 `blog` 目录下。

```
blog
└── forms.py
```

好吧，让我们打开它，然后键入以下代码：

```
from django import forms

from .models import Post

class PostForm(forms.ModelForm):

    class Meta:
        model = Post
        fields = ('title', 'text',)
```

首先我们需要导入Django表单（`from django import forms`）然后，显然是我们的 `Post` 模型（`from .models import Post`）。></p>

`PostForm`，正如你所猜想的，是我们表单的名字。我们需要告诉Django，这个表单是一个`ModelForm`（所以Django将会为我们变一些魔法）—`forms.ModelForm``对此负责。

下面，我们有 `class Meta`，在这里我们告诉Django哪个模型会被用来创建这个表单（`model=Post`）。

最后，我们可以说哪些字段会在我们的表单里出现。在这个场景里，我们只想要 `title` 和 `text` 显示出来—`author` 应该是当前登录的人（你！）然后 `created_date` 应该是我们创建文章时自动分配的（比如，在代码里），对吗？

就是这样！现在我们所有要做的就是在视图里使用表单，然后展现在在模板里。

所以下次我们将会创建：一个指向页面的链接，一个URL，一个视图和一个模板。

指向页面表单的链接

是时候打开 `blog/templates/blog/base.html` 了。我们将添加一个链接到 `div`，命名为 `page-header`：

```
<a href="{% url 'post_new' %}" class="top-menu"><span class="glyphicon glyphicon-plus"></span></a>
```

请注意我们想要调用我们的新视图 `post_new`。

添加了新的行后，你的html文件现在应该看起来像这样：

```
% load staticfiles %
<html>
  <head>
    <title>Django Girls blog</title>
    <link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap.min.css">
    <link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap-theme.min.css">
    <link href='//fonts.googleapis.com/css?family=Lobster&subset=latin,latin-ext' rel='stylesheet' type='text/css'>
    <link rel="stylesheet" href="{% static 'css/blog.css' %}">
  </head>
  <body>
    <div class="page-header">
      <a href="{% url 'post_new' %}" class="top-menu"><span class="glyphicon glyphicon-plus"></span></a>
      <h1><a href="/">Django Girls Blog</a></h1>
    </div>
    <div class="content container">
      <div class="row">
        <div class="col-md-8">
          {% block content %}
          {% endblock %}
        </div>
      </div>
    </body>
  </html>
```

然后保存，刷新 <http://127.0.0.1:8000> 页面，你可以明显地看到一个熟悉的 `NoReverseMatch` 错误信息，是吧？

URL

我们打开 `blog/urls.py` 然后添加一个新行：

```
url(r'^post/new/$', views.post_new, name='post_new'),
```

最终代码会看起来像这样：

```
from django.conf.urls import url
from . import views

urlpatterns = [
    url(r'^$', views.post_list, name='post_list'),
    url(r'^post/(?P<pk>[0-9]+)/$', views.post_detail, name='post_detail'),
    url(r'^post/new/$', views.post_new, name='post_new'),
]
```

刷新网页后，我们看到一个 `AttributeError`，因为我们没有实现 `post_new` 视图。让我们现在把它加上吧。

post_new视图

现在打开 `blog/views.py` 文件，加入下面的各行到 `from` 行下：

```
from .forms import PostForm
```

还有我们的 `view`：

```
def post_new(request):
    form = PostForm()
    return render(request, 'blog/post_edit.html', {'form': form})
```

为了创建一个新的 Post 表单，我们需要调用 `PostForm()`，然后把它传递给模板。我们会回到这个视图，但是现在，让我们为这个表单快速创建一个模板。

模板

我们需要在 `blog/templates/blog` 目录下创建一个文件 `post_edit.html`。为了创建一个表单，我们需要几件事情：

- 要展示表单，我们只需要很简单地加上 `{{ form.as_p }}`。
- 上面的这行需要被HTML表单标签包裹：`<form method="POST">...</form>`
- 我们需要一个 Save 按钮。我们通过使用一个HTML按钮来完成：`<button type="submit">Save</button>`
- 最后在 `<form ...>` 标签后，我们需要加上 `{% csrf_token %}`。这个非常重要，因为他会让你的表单变得更安全！Django会提醒你，当你试图保存表单而你又恰巧忘了这一点：

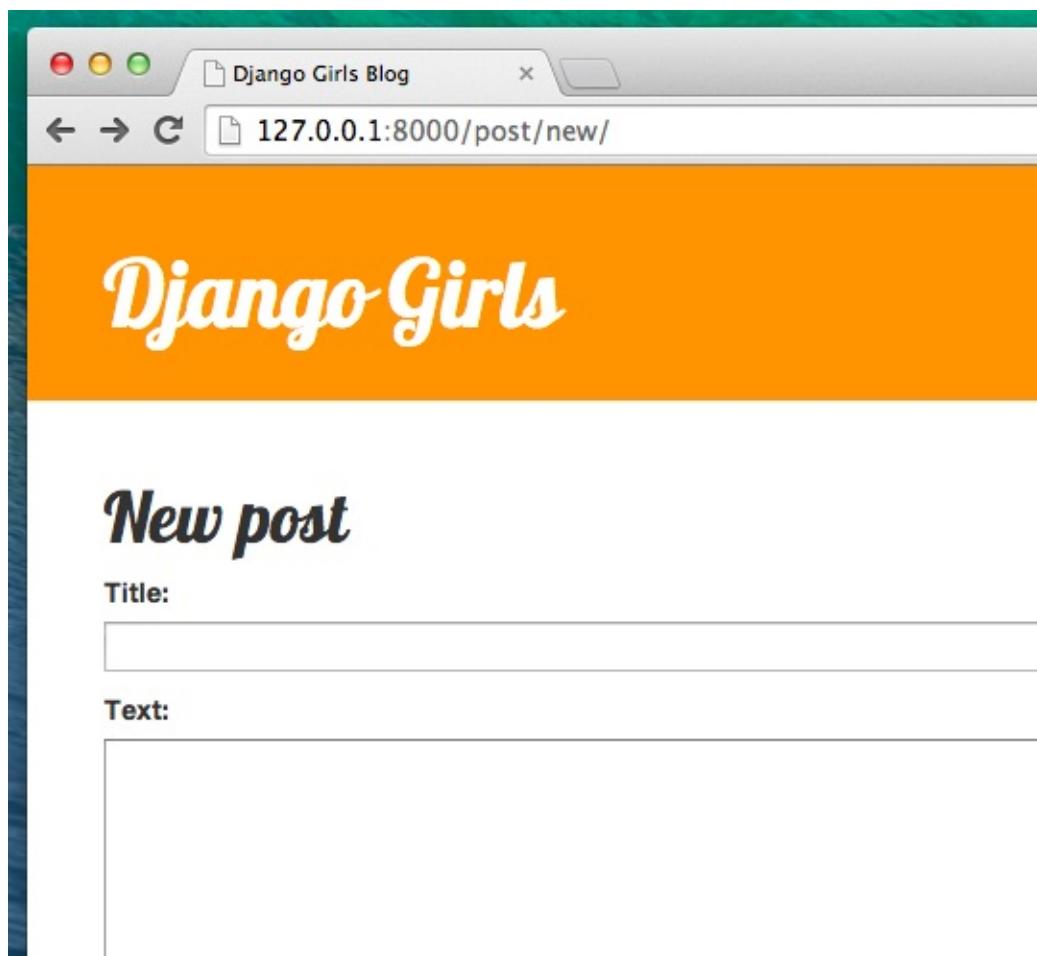


好，让我们看看HTML 在 `post_edit.html` 里应该看起来什么样：

```
{% extends 'blog/base.html' %}

{% block content %}
    <h1>New post</h1>
    <form method="POST" class="post-form">{% csrf_token %}
        {{ form.as_p }}
        <button type="submit" class="save btn btn-default">Save</button>
    </form>
{% endblock %}
```

现在刷新！哇！你的表单显示出来了！



但是，请等一下！当你键入诸如 `title` 和 `text` 字段，然后视图保存它—下面会发生什么？

什么都没有！我们再一次回到了同一个页面，然而我们的文本已经消失了...同时没有新的文章被发布。所以错在哪里了呢？

答案是：没有错误。我们需要在我们的视图里做更多的工作。

保存表单

再一次打开 `blog/views.py`。我们在看到 `post_new` 中的视图内容是：

```
def post_new(request):
    form = PostForm()
    return render(request, 'blog/post_edit.html', {'form': form})
```

当我们提交表单，我们都回到相同的视图，但是这个时候我们有一些更多的数据在 `request`，更具体地说在 `request.POST`（命名和博客后缀“post”无关，它只是用来帮我们“上传”数据）。还记得在HTML文件里，我们的 `<form>` 定义有一个方法 `method="POST"`？现在所有从表单来的东西都在 `request.POST`。你不应该重命名 `POST` 为其他任何东西（其他唯一有效的 `method` 值是 `GET`，但是我们没有时间去解释它们两者区别是什么）。

所以在我们的视图里，我们有了两种不同的情况去处理。首先：当我们首次访问一个页面，我们想要得到一个空白的页面。第二：当我们回到视图，要有我们所有我们刚刚键入的数据。所以我们需要添加一个条件判断（我们为此使用 `if`）。

```
if request.method == "POST":
    [...]
else:
    form = PostForm()
```

现在去填写 [...]。如果 `method` 是 `POST`，那么我们要用表单里的数据构建 `PostForm`，对吗？我们会这样做：

```
form = PostForm(request.POST)
```

很容易吧！下一件事情就是去检查表单是否正确（所有必填字段都要被设置并且不会保存任何不正确的值）。我们将使用 `form.is_valid()` 来实现。

我们检查表单是否正确，如果是我们就保存它！

```
if form.is_valid():
    post = form.save(commit=False)
    post.author = request.user
    post.published_date = timezone.now()
    post.save()
```

基本上，我们这里有两件事情：我们使用 `form.save` 保存表单，我们添加一个作者（因为 `PostForm` 中没有 `author` 字段，然而这个字段是必须的！）。`commit=False` 意味着我们还不想保存 `Post` 模型—我们想首先添加作者。大多数情况下，当你使用 `form.save()` 时，不会使用 `commit=False`，但是在这种情况下，我们需要这样做。`post.save()` 会保留更改（添加作者），并创建新的博客文章！

最后，如果我们能够立即去 `post_detail` 页面创建新的博客内容，那将很酷，对吗？为了做到这点，我们需要再导入一个：

```
from django.shortcuts import redirect
```

把它添加到你文件的最开始处。现在我们可以说：创建完新帖子我们就转去 `post_detail` 页面。

```
return redirect('post_detail', pk=post.pk)
```

`post_detail` 是我们想去的视图的名字。还记得这个视图需得具有一个 `pk` 变量吗？为了把它传递给视图我们使用 `pk=post.pk`，其中 `post` 就是我们刚刚创立的博客帖子！

好吧，我们已经说了很多了，但可能我们想看到整个视图现在看起来什么样，对吗？

```
def post_new(request):
    if request.method == "POST":
        form = PostForm(request.POST)
        if form.is_valid():
            post = form.save(commit=False)
            post.author = request.user
            post.published_date = timezone.now()
            post.save()
            return redirect('post_detail', pk=post.pk)
    else:
        form = PostForm()
    return render(request, 'blog/post_edit.html', {'form': form})
```

让我们看看它是否正常工作。转到页 <http://127.0.0.1:8000//post/new/>，添加 `title` 和 `text`，将它保存... 看！新博客文章已经加进来了，我们被重定向到 `post_detail` 页面！

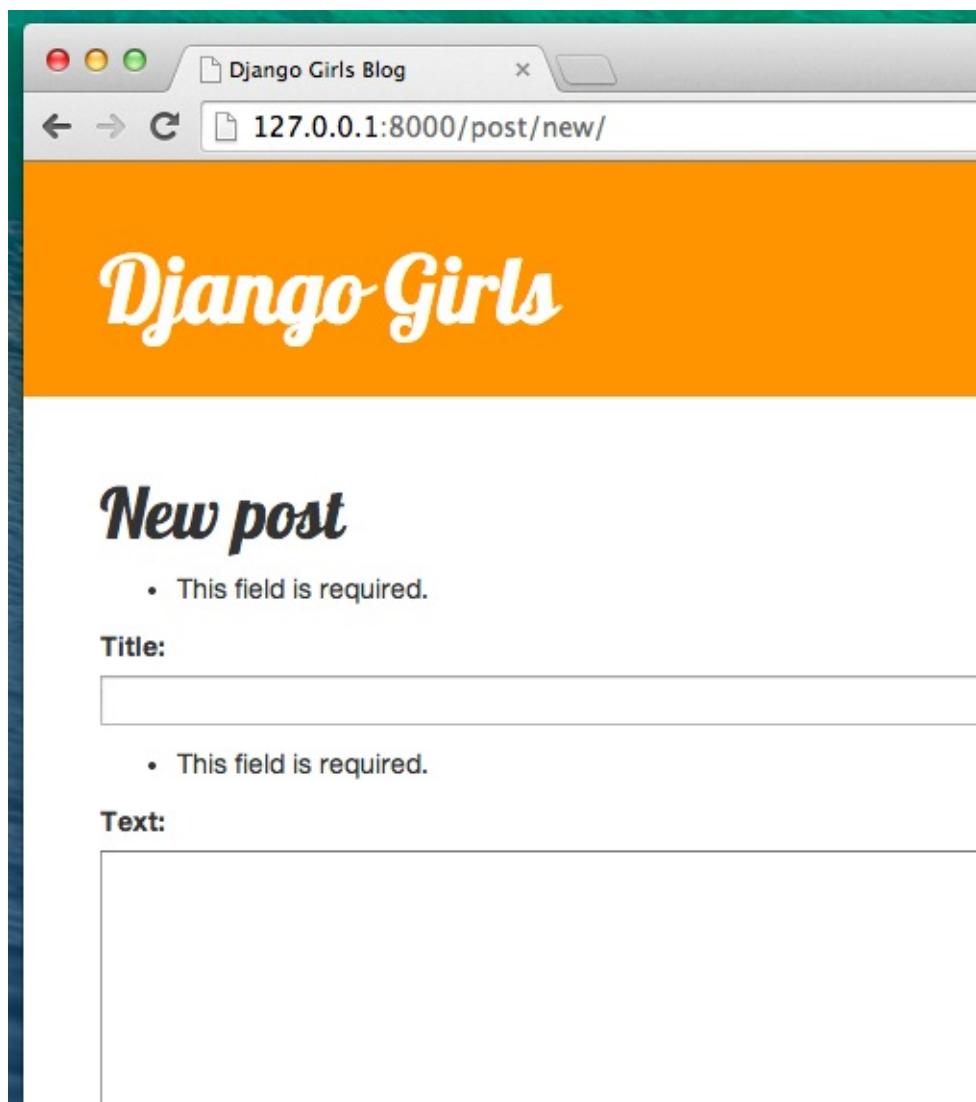
你可能已经注意到在保存博客文章之前我们设置发布日期。稍后，我们讲介绍一个在 **Django Girls** 教程：扩展中介绍 `publish button`。

太棒了！

表单验证

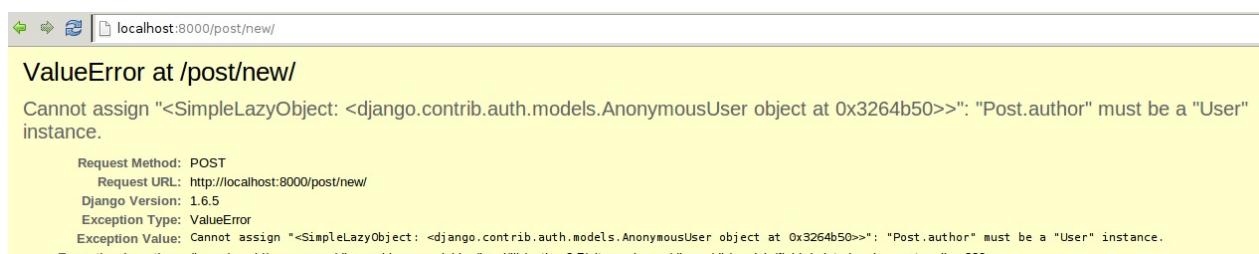
现在，我们将给你展现Django表单是多么酷。一篇博客文章需要有 `title` 和 `text` 字段。在我们的 `Post` 模型中我们并没有说（和 `发布日期` 恰恰相反）这些字段不是必须的，所以Django，默认期望他们是有存储数据的。

尝试不带 `title` 和 `text` 内容保存表单。猜猜，会发生什么！



Django会处理验证我们表单里的所有字段都是正确的。这不是很棒？

因为我们最近使用过Django管理界面，系统目前认为我们已经登录了。有几种情况可能导致我们被登出（关闭浏览器，重新启动数据库等等）。如果你发现当你创建一个文章时得到了一个指向未登录用户错误的时候，前往管理页面 <http://127.0.0.1:8000/admin>，再登录。这会暂时解决问题。有一个一劳永逸的方法在等着你，可以看看只要教程后的Homework: add security to your website! 章节。



编辑表单

现在我们知道如何添加一个新的表单。但是如果我们要编辑一个现有的呢？这和我们刚才做的非常相似。让我们快速创建一些重要的东西（如果你还不清楚他们，你应该问问你的教练或者看看前面的章节，因为我们已经覆盖了所有的这些步骤）。

打开 `blog/templates/blog/post_detail.html` 并添加以下行：

```
<a class="btn btn-default" href="{% url 'post_edit' pk=post.pk %}"><span class="glyphicon glyphicon-pencil"></span>
</a>
```

所以模板看起来像这样：

```
{% extends 'blog/base.html' %}

{% block content %}
<div class="post">
    {% if post.published_date %}
        <div class="date">
            {{ post.published_date }}
        </div>
    {% endif %}
        <a class="btn btn-default" href="{% url 'post_edit' pk=post.pk %}"><span class="glyphicon glyphicon-pencil"></span>
    <h1>{{ post.title }}</h1>
    <p>{{ post.text|linebreaksbr }}</p>
</div>
{% endblock %}
```

在 `blog/urls.py` 里我们添加这行：

```
url(r'^post/(?P<pk>[0-9]+)/edit/$', views.post_edit, name='post_edit'),
```

我们将复用模板 `blog/templates/blog/post_edit.html`，所以最后缺失的东西就是 `view`.

让我们打开 `blog/views.py`，并在文件的最后加入：

```
def post_edit(request, pk):
    post = get_object_or_404(Post, pk=pk)
    if request.method == "POST":
        form = PostForm(request.POST, instance=post)
        if form.is_valid():
            post = form.save(commit=False)
            post.author = request.user
            post.published_date = timezone.now()
            post.save()
            return redirect('post_detail', pk=post.pk)
    else:
        form = PostForm(instance=post)
    return render(request, 'blog/post_edit.html', {'form': form})
```

这看起来几乎完全和我们的 `post_new` 视图一样，对吗？但是不完全是。第一件事：我们从 `urls` 里传递了一个额外的 `pk` 参数。然后：我们得到了 `Post` 模型，我们想编辑 `get_object_or_404(Post, pk=pk)`，然后当我们创建了一个表单我们用一个 实例 来传递这篇文章，当我们想保存它：

```
form = PostForm(request.POST, instance=post)
```

当我们只是打开这篇文章的表单来编辑时：

```
form = PostForm(instance=post)
```

好，让我们来试试它是否可以工作！让我们先去 `post_detail` 页面。在右上角应该有一个编辑按钮：



当你点击它的的时候，你会看到我们博客文章的表单。

A screenshot of a web browser window titled "Django Girls Blog". The URL in the address bar is "127.0.0.1:8000/post/3/edit/". The page features a large orange header with the "Django Girls" logo. Below the header, the title of the post is "New post". The form has two fields: "Title:" with the value "Nulla facilisi" and "Text:" with a large text area containing placeholder text: "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras bibendum sapien interdum, posuere massa et, hendrerit leo. Nam commodo facilisis sapien vitae ornare. Integer eget purus posuere, vestibulum arcu at, pulvinar elit. Integer eleifend, nisl a molestie auctor, dui arcu ultricies erat, sed dapibus ante tellus id est. Aliquam erat volutpat. Vestibulum tellus est, ultrices nec iaculis sit amet, hendrerit in mauris. Suspendisse lacinia mi in magna tincidunt, sed convallis ipsum semper. Clas".

随意修改标题和内容，然后保存更改！

祝贺你！你的应用程序正在变得越来越完整！

如果你需要更多关于Django表单的信息，你应该阅读文档：<https://docs.djangoproject.com/en/1.8/topics/forms/>

安全性

能够通过点击一条连接进行发布确实不错。但是现在，任何访问你网站的人都能够发布一条新博客日志，这可能不是你想要的。那让我们来让这个发布按钮只显示给你，对其他人则不显示。

在 `blog/templates/blog/base.html` 中，找到我们 `page-header` `div` 和你早些时候在放那里锚点标记。看起来应该像这样：

```
<a href="{% url 'post_new' %}" class="top-menu"><span class="glyphicon glyphicon-plus"></span></a>
```

我们要将另一个 `{% if %}` 标记到这，这会使链接仅在以管理者身份登录的用户访问时显示。现在来说，管理员就是你！像这样修改 `<a>` 标记：

```
{% if user.is_authenticated %}
    <a href="{% url 'post_new' %}" class="top-menu"><span class="glyphicon glyphicon-plus"></span></a>
{% endif %}
```

这个 `{% if %}` 会使得链接仅仅发送到哪些已经登陆的用户的浏览器。这并不能完全保护发布新文章，不过这是很好的第一步。我们将在扩展课程中包含更多安全部分。

你应已登录，如果你刷新页面，你不会看到有什么不同。不过，在新的浏览器或隐身窗口中加载页，你会看不到这个链接！

还有一件事：部署时刻！

我们来看看这一切能否运行在 [PythonAnywhere](#) 上。另一次部署的时间到了！

- 首先，提交你的新代码，然后将它推送到 [Github](#) 上

```
$ git status
$ git add --all .
$ git status
$ git commit -m "Added views to create/edit blog post inside the site."
$ git push
```

- 然后，在一个 [PythonAnywhere](#) 的 Bash 终端里运行：

```
$ cd my-first-blog
$ source myenv/bin/activate
(myenv)$ git pull
[...]
(myenv)$ python manage.py collectstatic
[...]
```

- 最后，跳到 [Web 标签页](#) 并点击重新载入。

就是这样！祝贺你：）

接下来呢？

祝贺你！你简直太棒了。我们很自豪！<3

现在要做什么？

休息一下，放松。你刚做完非常多的事。

之后，请确保：

- 在[Facebook](#)或者[推特](#)来关注[Django Girls](#)以了解最新情况

你能推荐更进一步的资源吗？

是啊！首先，继续尝试其他书籍，称作[Django Girls教程：扩展](#).

稍后，你可以试试利用以下的资源。他们非常值得推荐啊！

- [Django's official tutorial](#)
- [New Coder tutorials](#)
- [Code Academy Python course](#)
- [Code Academy HTML & CSS course](#)
- [Django Carrots tutorial](#)
- [Learn Python The Hard Way book](#)
- [Getting Started With Django video lessons](#)
- [Two Scoops of Django: Best Practices for Django 1.8 book](#)