

目次

| | |
|---------------------------|------|
| イントロダクション | 1.1 |
| インターネットはどうやって動いているの？ | 1.2 |
| コマンドラインを使ってみよう | 1.3 |
| Pythonをインストールしよう | 1.4 |
| コードエディタ | 1.5 |
| Pythonをはじめよう | 1.6 |
| Djangoってなあに？ | 1.7 |
| Djangoをインストールしよう | 1.8 |
| プロジェクトを作成しよう | 1.9 |
| Djangoモデル | 1.10 |
| ログインページを作ろう（Django admin） | 1.11 |
| デプロイ！ | 1.12 |
| Django urlsってなに？ | 1.13 |
| Djangoビューってなに？ | 1.14 |
| HTMLをやってみよう | 1.15 |
| クエリセット 1 | 1.16 |
| クエリセット 2 | 1.17 |
| テンプレートに表示しよう | 1.18 |
| CSSでカワイくしよう | 1.19 |
| テンプレートを拡張しよう | 1.20 |
| アプリケーションを拡張しよう | 1.21 |
| フォームを作ろう | 1.22 |
| ドメインを設定しよう | 1.23 |
| 次のステップは？ | 1.24 |

Django Girls Tutorial

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

これは、Creative Commons Attribution-ShareAlike 4.0 International License のライセンスの下で提供しています。ライセンスについてはこちらをご確認ください。

<http://creativecommons.org/licenses/by-sa/4.0/>

Introduction

テクノロジーが日々進化して、次々とでてくる新しいものに驚いていませんか？ウェブサイトはどうやって動いているのだろうと興味をもちつつ、先延ばしにしていませんか？難しそうと思って、1人で勉強するのを辞めちゃったことはありませんか？

そんなあなたに朗報です！プログラミングはそれほど難しくありませんよ。楽しみかたをお教えします！！

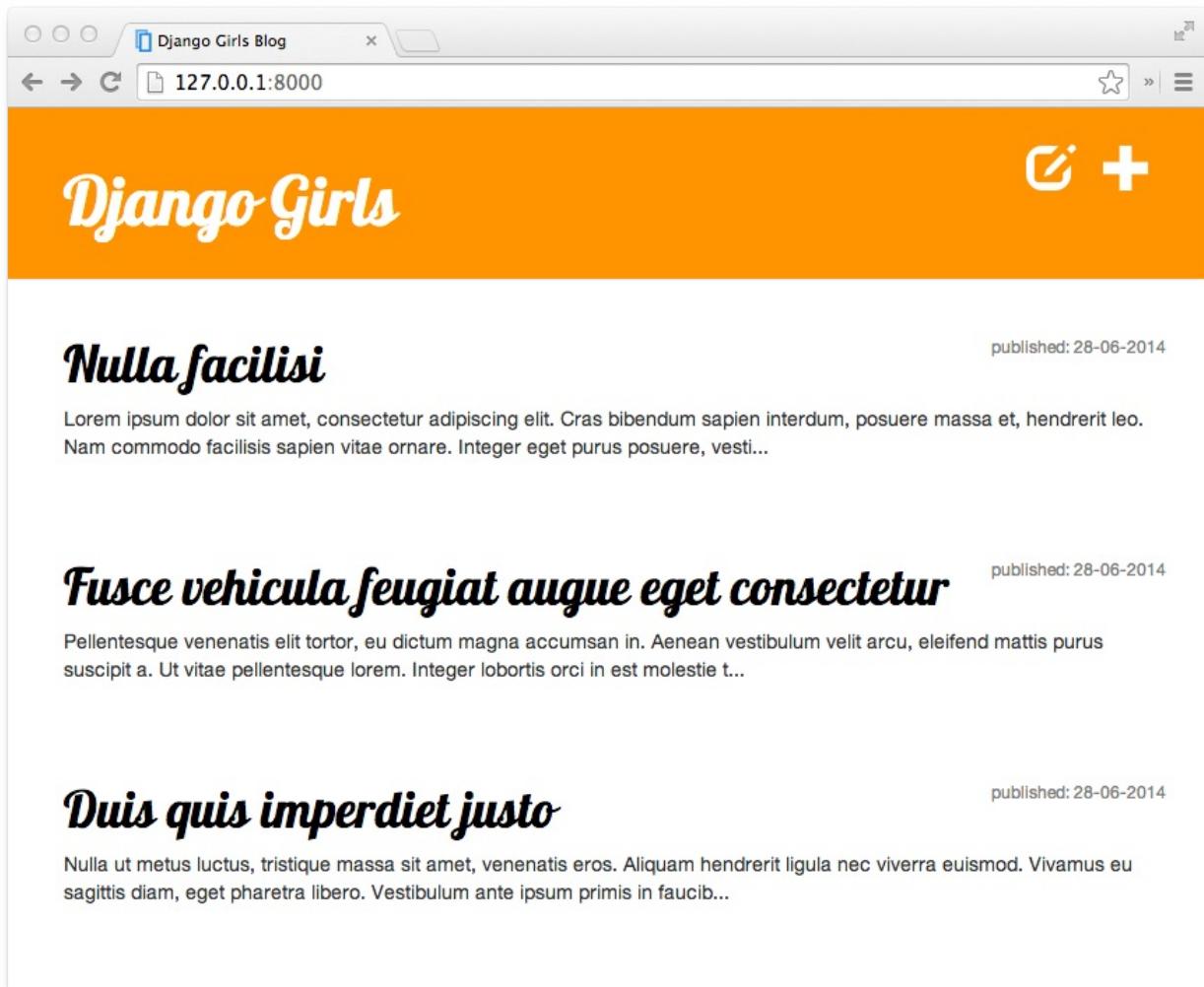
このチュートリアルは、魔法のようにあなたをプログラマーに変身させるものではありません。上手くなりたかったら、何ヶ月あるいは何年もの勉強と練習を積まなければいけません。しかし、このチュートリアルをとおして、プログラミングやウェブサイトを作成することはあなたが思っているほど複雑ではないことを分かってもらえればと思います。できるだけ細かく説明していきますね。コワイと思わず、新しい世界に一歩を踏み出してみてください！

あなたがテクノロジーやプログラミングを楽しんでくれると嬉しいです！！

What will you learn during the tutorial?

このチュートリアルを終える頃には、シンプルなウェブアプリケーションができているでしょう。あなたのブログです。オンラインへ載せる方法もお教えするので、お友達にも見せることができますよ！

このようなサイトが出来上がります！：



もしあなたがこのチュートリアルを1人ですすめていて、質問ができるコーチが周りにいないう時は、ここにチャットを用意しています。: [gitter](#) [join chat](#) これまでにワークショップに参加したことがある方やコーチの皆さんのが、このチャットで助けてくれることでしょう。心配せずに、質問をなげかけてみてくださいね！

OK, では、早速はじめていきましょう...

About and contributing

This tutorial is maintained by [DjangoGirls](#). If you find any mistakes or want to update the tutorial please [follow the contributing guidelines](#).

Translation into Japanese

多くの有志により、チュートリアルの日本語翻訳やアップデートがおこなわれています。

How the Internet works

This chapter is inspired by a talk "How the Internet works" by Jessica McKellar (<http://web.mit.edu/jesstess/www/>).

We bet you use the Internet every day. But do you actually know what happens when you type an address like <http://djangogirls.org> into your browser and press 'Enter'?

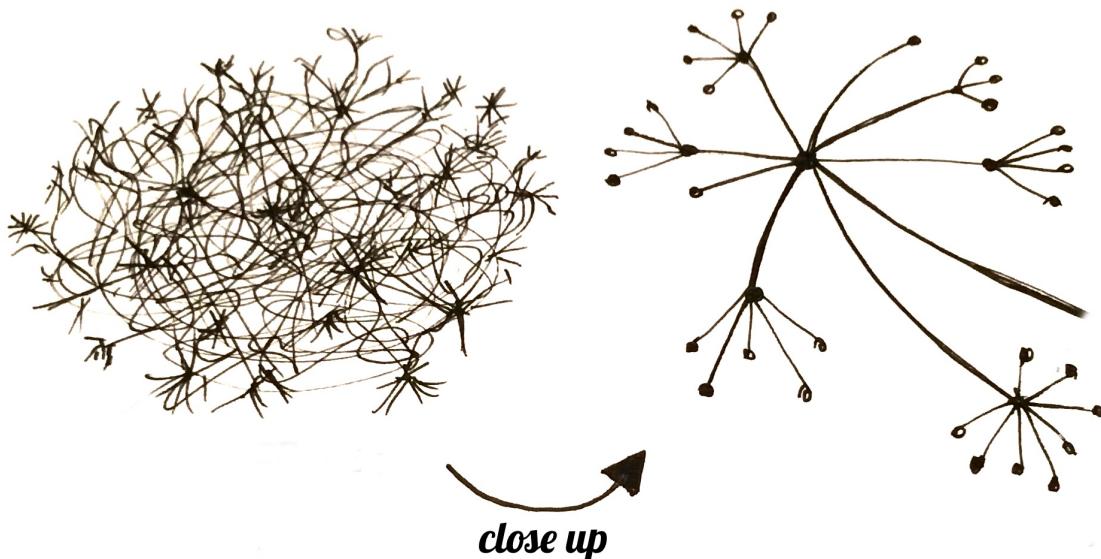
The first thing you need to understand is that a website is just a bunch of files saved on a hard disk. Just like your movies, music or pictures. However, there is one part that is unique for websites: they include computer code called HTML.

If you're not familiar with programming, it can be hard to grasp HTML at first, but your web browsers (like Chrome, Safari, Firefox, etc.) love it. Web browsers are designed to understand this code, follow its instructions and present all these files that your website is made of exactly the way you want them to be presented.

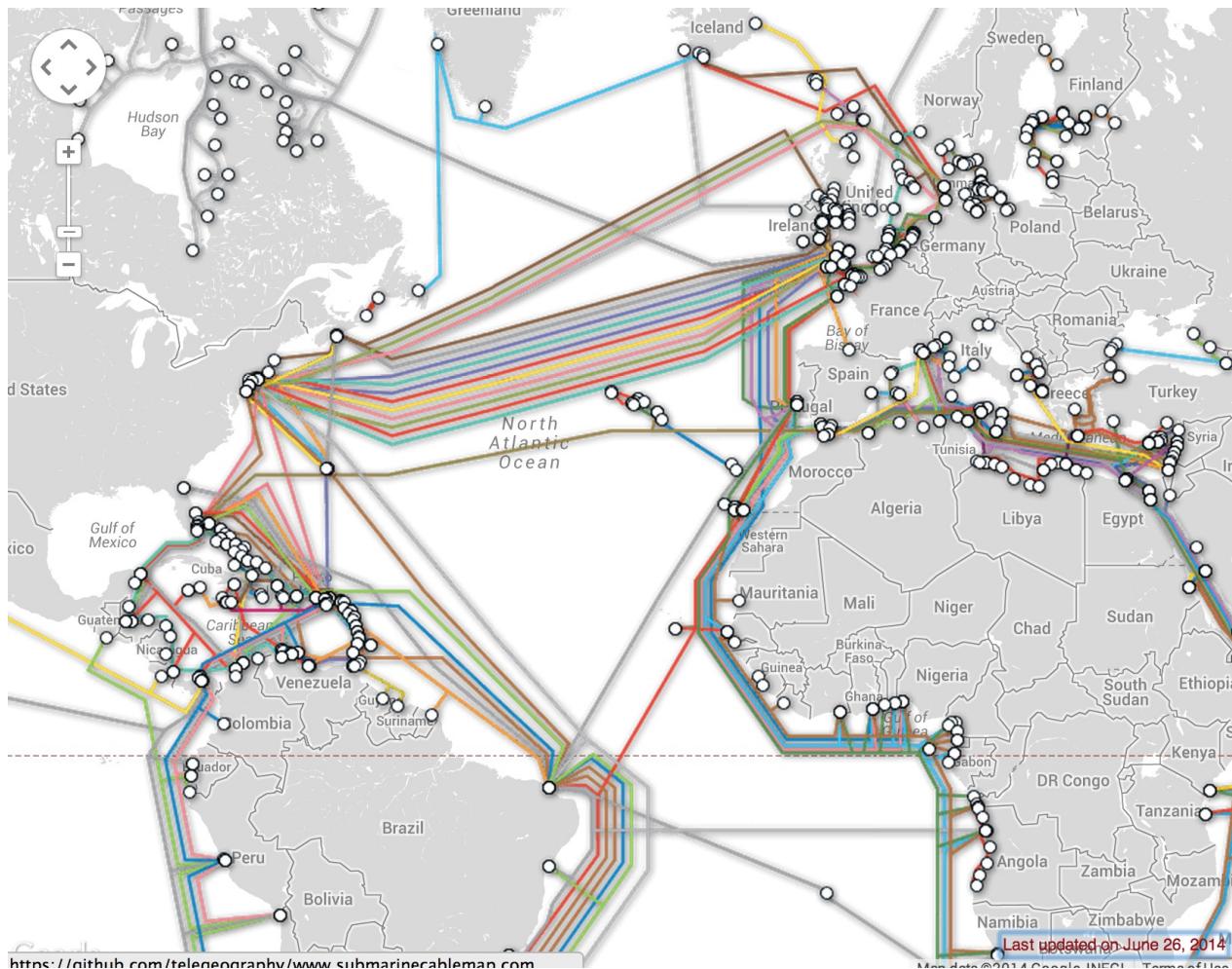
As with every file, we need to store HTML files somewhere on a hard disk. For the Internet, we use special, powerful computers called *servers*. They don't have a screen, mouse or a keyboard, because their main purpose is to store data and serve it. That's why they're called *servers* -- because they *serve* you data.

OK, but you want to know how the Internet looks like, right?

We drew you a picture! It looks like this:

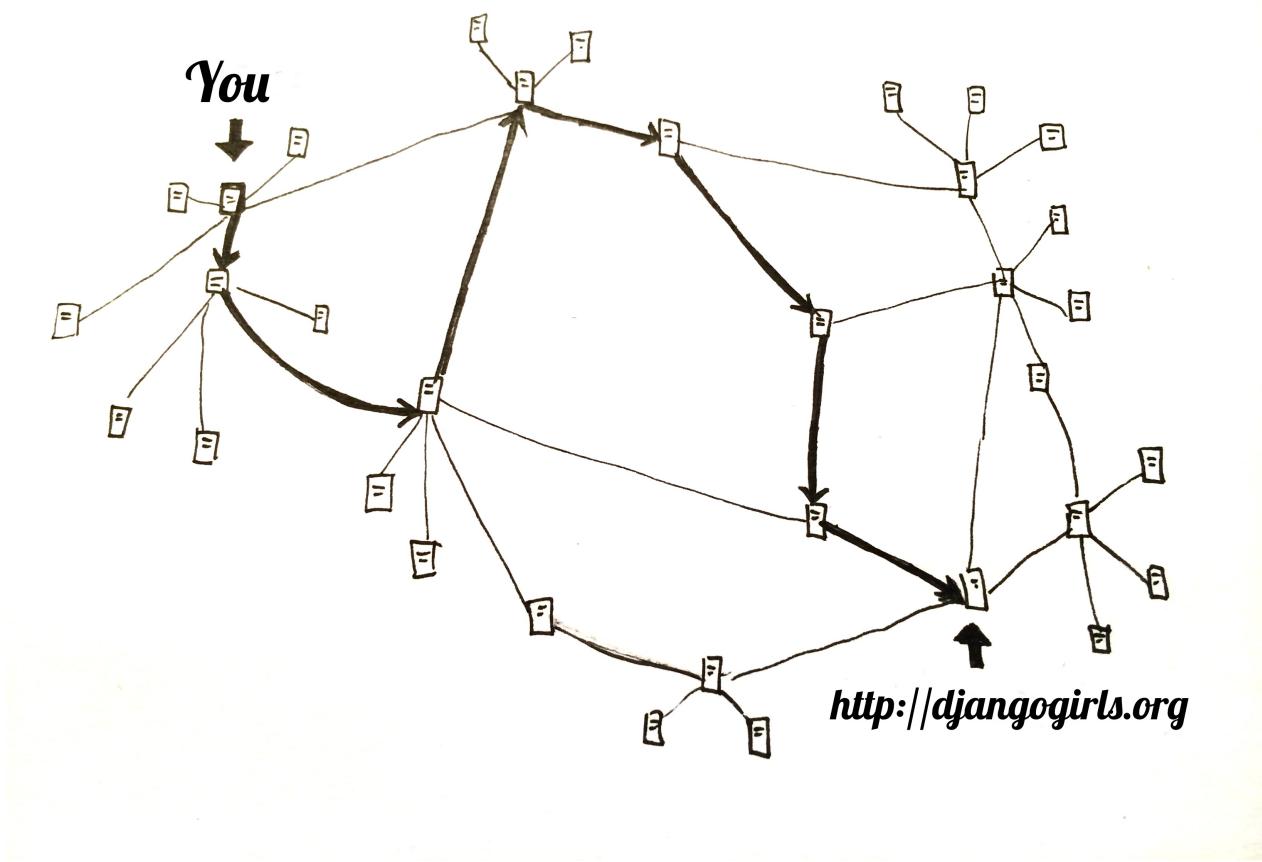


Looks like a mess, right? In fact it is a network of connected machines (the above mentioned servers). Hundreds of thousands of machines! Many, many kilometers of cables around the world! You can visit a Submarine Cable Map website (<http://submarinecablemap.com/>) to see how complicated the net is. Here is a screenshot from the website:



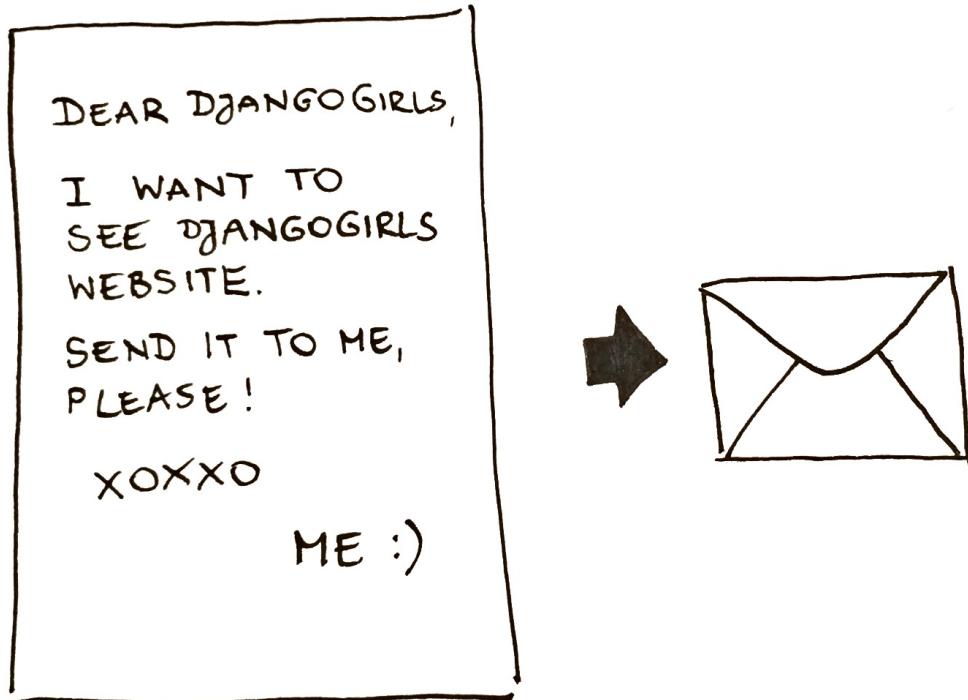
It is fascinating, isn't it? But obviously, it is not possible to have a wire between every machine connected to the Internet. So, to reach a machine (for example the one where <http://djangogirls.org> is saved) we need to pass a request through many, many different machines.

It looks like this:



Imagine that when you type <http://djangogirls.org>, you send a letter that says: "Dear Django Girls, I want to see the djangogirls.org website. Send it to me, please!"

Your letter goes to the post office closest to you. Then it goes to another that is a bit nearer to your addressee, then to another and another till it is delivered at its destination. The only unique thing is that if you send letters (*data packets*) frequently to the same place, each letter might go through totally different post offices (*routers*), depending on how they are distributed in each office.



Yes, it is as simple as that. You send messages and you expect some response. Of course, instead of paper and pen you use bytes of data, but the idea is the same!

Instead of addresses with a street name, city, zip code and country name, we use IP addresses. Your computer first asks the DNS (Domain Name System) to translate djangogirls.org into an IP address. It works a little bit like old-fashioned phonebooks where you could look for the name of the person you want to contact and find their phone number and address.

When you send a letter, it needs to have certain features to be delivered correctly: an address, stamp etc. You also use a language that the receiver understands, right? The same is with *data packets* you send in order to see a website: you use a protocol called HTTP (Hypertext Transfer Protocol).

So, basically, when you have a website you need to have a *server* (machine) where it lives. The *server* is waiting for any incoming *requests* (letters that ask the server to send your website) and it sends back your website (in another letter).

Since this is a Django tutorial, you will ask what Django does. When you send a response, you don't always want to send the same thing to everybody. It is so much better if your letters are personalized, especially for the person that has just written to you, right? Django helps you with creating these personalized, interesting letters :).

Enough talk, time to create!

インターネットはどうやって動いているの？

Introduction to the command-line interface

さあ、これから最初のコードを書いていきますよ。楽しんでいきましょう！(:)

最初にお友達になるのはコレです。: コマンドライン！

プログラマーが黒い画面に向かっている光景を見たことがありますか？ここからは、その黒い画面を触ってみます。最初はちょっとコワイと思うかもしれません、そんなことはありません。プロンプトと呼ばれるものがあなたの命令（コマンド）を待っています。

What is the command line?

さて、コマンドラインあるいはコマンドラインインターフェイスと呼ばれるこの画面は、キーボードで入力したテキストで命令を出してコンピューターと直接対話するように、ファイルを見たり、変更したり（グラフィカル・インターフェースじゃないだけで、WindowsのエクスプローラやMacのファインダーと同じ役割です）するものです。このコマンドラインは、*cmd*, *CLI*, プロンプト, コンソール or ターミナルと呼ばれることもあります。

Open the command-line interface

では、実際にコマンドラインを開いて、触ってみるとしましょう。

Windows

[スタート] メニューから [すべてのプログラム] — [アクセサリ] — [コマンドプロンプト] を選択してください。

Mac OS X

[アプリケーション] — [ユーティリティ] — [ターミナル] を選択して下さい。

Linux

おそらく [アプリケーション] — [アクセサリ] — [ターミナル] と選択し起動できるでしょう。あなたのシステムによってはこの通りではないことがあります。見つからないときは、Google先生にきいてみましょう。(:)

Prompt

おそらく今、真っ白または真っ黒な画面が開かれていることでしょう。この画面はあなたの命令を待っています。

MacあるいはLinuxの方は、次のように `$` と表示されているのがわかりますか?:

```
$
```

Windowsの方は、`>` という記号が表示されていることでしょう。:

```
>
```

各コマンドの先頭には、この記号とスペースがつきます。あなたのコンピューターが表示してくれるので、自分で入力する必要はありません。:)

ちょっと補足です。プロンプト記号の前に `C:\Users\ola>` or `Olas-MacBook-Air:~ ola$` のような表示がありますね。これは間違いではありません。100%正解です。このチュートリアルでは、シンプルにわかりやすくするために、その部分を省略して記述します。

Your first command (YAY!)

では最初は簡単なコマンドを実行してみましょう。次のように入力してみてください。:

```
$ whoami
```

または

```
> whoami
```

そして最後にEnterキーを押して下さい。このような結果が返ってきます:

```
$ whoami  
olasitarska
```

ご覧のとおり、コンピューターがあなたのユーザーネームを表示してくれましたね。面白いでしょ?:)

コピー&ペーストではなく、コマンドを入力して試してみてください。そのうち自然と覚えられるようになりますからね！

Basics

OSによってコマンドが若干違います。あなたのコンピューターのOSの方法に従って、以下は進めていってくださいね。次にいってみましょう。

Current directory

今どこのディレクトリにいるか（どのフォルダで作業をしているか）、知りたいですね？では、このようにキーボードで入力して、Enterキーをおしてください。：

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

Windowsの方は次のように：

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

おそらく、似たようなものがあなたの画面に表示されたのではないでしょうか。コマンドラインを起動した最初は、通常ユーザーのホームディレクトリが表示されます。

補足: 'pwd' は'print working directory'を意味しており、現在いる作業ディレクトリを取得することです。

List files and directories

では、その中には何があるのでしょうか？表示させてみましょう。：

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

Windows:

```
> 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  
...
```

Change current directory

次に、デスクトップのディレクトリに移動してみましょう。

```
$ cd Desktop
```

Windows:

```
> cd Desktop
```

本当にディレクトリを移動できたか、確認してみましょう。:

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

Windows:

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

できていますね！

PRO tip: `cd D` と入力して、キーボードの `tab` ボタンを押してください。すると、Dに続く残りの部分が自動的に補完されて入力されます。もし、Dから始まるディレクトリ名が他にもあれば、`tab` ボタンを繰り返し押すと候補が次々と表示されます。入力が楽になりますね。

Create directory

それでは、Django Girlsのディレクトリをデスクトップに新規作成してみましょう。:

```
$ mkdir djangogirls
```

Windows:

```
> mkdir djangogirls
```

この短いコマンドで、デスクトップに `djangogirls` という名前の新しいフォルダが作成されました。あなたのデスクトップを見てフォルダが作成されていることを確認してみましょう。あるいは、先ほど学んだコマンド `ls / dir` を使って確認しましょう。やってみてください。:)

PRO tip: 同じコマンドを何度もなんども入力したくない時は、上下矢印キー↑, ↓を押せば、先ほどキーボードから入力したものが現れます。内容を修正したい場合には、左右矢印キー←, →を利用して修正したい位置にカーソルを移動させて、修正することができますよ。

Exercise!

練習をしてみましょう。先ほど作成した `djangogirls` ディレクトリの中に、新たに `test` という名前のディレクトリを作成してください。使うコマンドは、`cd` と `mkdir` ですよ。

Solution:

```
$ cd djangogirls
$ mkdir test
$ ls
test
```

Windows:

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

おめでとうございます！よくできました！:)

Clean up

練習がおわったら、それをそのままに置いておくと邪魔になりますね。削除しておきましょう。

はじめに、作業するディレクトリをデスクトップに戻しましょう。:

```
$ cd ..
```

Windows:

```
> cd ..
```

`cd` の後にある `..` で、現在の親ディレクトリに移動します。（今作業しているフォルダのひとつ上のフォルダに移動するということですね。）

現在の作業ディレクトリを確認しておきましょう。:

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

Windows:

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

では、`djangogirls` ディレクトリを削除しましょう。

注意！: `del` , `rmdir` や `rm` のコマンドを使って削除したファイルは、復活できません。完全に消えてしまいます。このコマンドを使う時は、よく気をつけてくださいね。

```
$ rm -r djangogirls
```

Windows:

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

できました！本当に削除されたか、確認してみましょう。:

```
$ ls
```

Windows:

```
> dir
```

Exit

ここまでです。それではコマンドラインを終了しましょう。かっこいいやり方で終わりたいですよね?:)

```
$ exit
```

Windows:

```
> exit
```

かっこいいですね!:)

Summary

ここに学んだコマンドをまとめておきます。:

| コマンド (Windows) | コマンド (Mac OS / Linux) | 説明 | 実行例 |
|-------------------|--------------------------|-------------------|--|
| exit | exit | ウィンドウを閉じる | exit |
| cd | cd | ディレクトリを移動する | cd test |
| dir | ls | ディレクトリとファイルの一覧を表示 | dir |
| copy | cp | ファイルをコピーする | copy c:\test\test.txt c:\windows\test.txt |
| move | mv | ファイルを移動させる | move c:\test\test.txt c:\windows\test.txt |
| mkdir | mkdir | ディレクトリを新規作成する | mkdir testdirectory |
| del | rm | ディレクトリやファイルを削除する | del c:\test\test.txt |

ここで勉強したのはコマンドのほんの一部でしたが、このワークショップで使うコマンドはこれだけです。

もっと勉強したい方は、ss64.com に各OSのコマンド一覧があります。ご参考までに。

Ready?

よし、次はPythonを勉強していきましょう！

Let's start with Python

ついにここまできました！

まずは、最初にPythonとは何かお話させて下さいね。Pythonはとても人気のあるプログラミング言語です。Webサイトや、ゲーム、サイエンス、グラフィックス、などなど、たくさんの場面で使われています。

Pythonは1980年台の終わりに、人間が読みやすい（機械だけでなく）言語を目的に開発されました。だから、他の言語に比べて、Pythonはとてもシンプルで、勉強しやすいのです。でもご心配なく、Pythonはとってもパワフルな言語でもありますから！

Python installation

このセクションは、Geek Girls Carrots (<http://django.carrots.pl/>) のチュートリアルをもとに作成されています。

Django は、Pythonで開発されています。なにをするにせよ、まずはPythonが必要です。インストールしましょう！ Python 3.5 をインストールします。3.5以前のバージョンをインストール済みの場合は、アップグレードしてください。

Windows

Windowsをお使いのかたは、次のリンクからダウンロードすることができます。

[https://www.python.org/downloads/release/python-352/.*.msi](https://www.python.org/downloads/release/python-352/) ファイルをダウンロードしたら、ダブルクリックして実行して、指示のとおりインストールしてください。Pythonをどのディレクトリにインストールしたか確認して、おぼえておいてくださいね。後ほど必要になってきます。

Linux

おそらく殆どの場合、Pythonはすでにインストール済みでしょう。インストールされているか確認するためには（バージョンを確認するためにも）、コンソールを起動して次のコマンドを打ってください。

```
$ python3 --version  
Python 3.5.2
```

もし、Pythonがインストールされていない場合、あるいはバージョンが古い場合は、次の指示に従ってインストールしてください。

Ubuntu

次のコマンドをコンソールに打って下さい。

```
sudo apt-get install python3.5 python3.5-venv
```

Fedora

次のコマンドをコンソールに打って下さい。

```
sudo yum install python3.5
```

OS X

Webサイトからダウンロードしてインストールしましょう。

<https://www.python.org/downloads/release/>

- *Mac OS X 64-bit/32-bit installer DMG* ファイルをダウンロードして下さい。
- ダブルクリックで開いてください。
- *Python.mpkg* をダブルクリックして、インストーラーを実行してください。

インストールが正しく行われたか確認するために、ターミナルを開いて、`python3` コマンド次のようにタイプしてみましょう。

```
$ python3 --version
Python 3.5.2
```

分からない時や、質問がある時は、コーチに質問してくださいね。ときどき上手くいかないこともあります。そんな時は、経験豊富な人に聞くといいですよ。

Code editor

はじめてのコードをこれから書いていくわけですから、コードエディタをダウンロードしましょう！

たくさんのエディタがありますが、どれを使うかは、個人の好みということに尽きます。多くのPythonプログラマーは、例えばPyCharmといった複雑ですがとても強力なIDEs (Integrated Development Environments)を使っています。しかし、初心者である我々には、強力でありながらももっとシンプルなエディタの方が適しているかもしれません。。

オススメのエディタは下記に挙げますが、気軽にコーチに質問して好みや特徴をきいてみてください。

Gedit

Geditはオープンソースの無料エディタです。全てのOSで使用できます。

[ダウンロード](#)

Sublime Text 2

Sublime Textは、とても人気のエディタです。無料の評価版があり、インストールも使い方も簡単です。こちらも全てのOSで利用できます。

[ダウンロード](#)

Atom

Atomは、GitHubによる新しいエディタです。無料で、オープンソースで、インストールも使い方も簡単です。Windows、OSX、Linuxで利用可能です。

[ダウンロード](#)

Introduction to Python

このチャプターの一部はGeek Girls Carrotsのチュートリアルをもとにしています。
(<http://django.carrots.pl/>).

さあ、コードを書いてみましょう！

Python prompt

Pythonで遊ぶために、コマンドラインを開きましょう。やり方は、チャプター [Intro to Command Line](#) で学びましたね。

準備ができたら、次の指示に従ってやってみましょう。

Pythonコンソールを開きましょう。Windowsなら `python` 、Mac OSやLinuxなら `python3` とタイプして Enterキーをおしてください。.

```
$ python3
Python 3.5.2 (...)

Type "copyright", "credits" or "license" for more information.
>>>
```

Your first Python command!

Pythonのコマンドが走ると、プロンプト記号が `>>>` に変わりました。これは、今Pythonの言語を実行できますという意味です。 `>>>` はタイプしなくていいですよ。 - Pythonがあなたの代わりにやってくれます。

Pythonコンソールを終える時は、 `exit()` とタイプするか、ショートカット `ctrl + z` (Windows) 、 `ctrl + d` (Mac/Linux) で終了です。 `>>>` は現れなくなりました。

けど、今はまだコンソールを終了しないで、もっと動かして学びましょう。最初はとてもシンプルなものからはじめましょう。例えば、簡単な計算をしてみましょう。 `2 + 3` とタイプして、Enterキーを押してください。

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

できました！答えがでてきましたね。Pythonは計算ができます。他にも、次のようなコマンドを試してみましょう：

- 4 * 5
- 5 - 1
- 40 / 2

ちょっとの間楽しんであそんでみたら、またココに戻ってきてくださいね。:)

お分かりのとおり、Pythonはステキな計算機ですね。他になにができるんだろう…と思ったら、次にいってみましょう。

Strings

あなたのお名前を次のようにクオーテーションをつけてタイプしてください。

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

はじめてのString（文字列）が完成です！Stringとは、文字の集合のことです。シングルクオーテーション(')あるいは、ダブルクオーテーション(")で囲います。最初と最後は同じ記号にしてください。- クオーテーションの中が文字列であることを意味しています。

複数の文字列を結合することもできます。次のように試してみましょう。:

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

文字列を繰り返すためには、演算子を使って繰り返し回数を指定することもできます。:

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

アポストロフィーを文字列の中に含めたい場合は、2通りの方法があります。

まずは、ダブルクオーテーションを使う方法です。

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

あるいは、バックスラッシュ(\)を使う方法もあります。:

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

できましたか？次に、あなたの名前を大文字に変えてみましょう。次のように記述してください。

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

ここで `upper` 関数 (**function**) を使うことができましたね！関数 (`upper()` など) は、呼び出したオブジェクト ("Ola" のことです) に対してどのような手順でどのような処理をするかをひとまとめにしたものです。

あなたの名前の文字数を知りたいときは、その関数もあります！

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

どうして、文字列の後に `.` をつけて関数を呼び出したり (`"Ola".upper()` のように)、あるいは、先に関数を呼び出してからこの中に文字列をいれているのか、と疑問に思ったかもしれません。そうですね。時に、オブジェクトに結びついた関数というのがあります。例えば、`upper()` は、文字列にのみ実行される関数です。私たちはこれをメソッド (**method**) と呼びます。それとは別に、特定のオブジェクトに関連せず、異なるタイプのオブジェクトに対して実行できる関数があります。例えば `len()` ですね。`len` 関数の引数として `"Ola"` をからこの中にいれているのです。

Summary

文字列はだいじょうぶですね。ここまでに学んだことをまとめましょう。

- プロンプト - Pythonプロンプトにコマンド（コード）を入力すると、答えがかえってきます。
- 数値と文字列 - 数値は計算に、文字列はテキストに使われます。
- 演算子 - 例えば `+` や `*` のように、値を計算して新しい値を返します。
- 関数 - `upper()` や `len()` のようにオブジェクトに対して行う機能のことです。

すべてのプログラミング言語に共通する基礎になります。もう少し難易度の高いものに挑戦してみましょう。準備はいいですか？

Errors

さて、新しいことをやってみましょう。あなたの名前の文字数を数えたように、数字の文字列は数えれるでしょうか？ `len(304023)` と記述して、Enterキーを押してみましょう。

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

はじめてのエラーがでました！オブジェクトタイプ"int" (**integers, 数値**) は文字数がありませんと言っています。では、どうすればよいでしょうか？この数字を文字列として扱えれば、文字数を数えるはずですよね？

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

できました！`str` 関数を `len` の中に記述しました。`str()` は、その中身を文字列に変換します。

- `str` 関数は、文字列に変換します。
- `int` 関数は、文字列や数値を整数に変換します。

重要！: 数字は文字列にすることはできますが、全ての文字が数字に変換できるわけではありません。例えば `int('hello')` は数字にはなりませんよね？

Variables

変数（variables）は、プログラミングの重要なコンセプトです。後で使うためにつける單なる名札ではありません。プログラマーは変数を使ってデータを保管したり、コードを読みやすくして、後でそれが何だったか覚えておかなくてもいいようにします。

変数 `name` を新しくつくってみましょう。

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

できましたか？簡単ですね。単純に `name` イコール (=) `Ola`と記述するだけです。.

見てのとおり、プログラムは、なにも返してくれませんね。では、変数がきちんとあるか、どうやって確かめたらいいのでしょうか？`name` とタイプして、Enterキーをおしてください。

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

やりました！あなたのはじめての変数ができましたね！代入する値を変えることもできます。

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

関数にも使えます。

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

素晴らしいですね！変数は、もちろん数値にも使えますよ。

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

もしも、間違えた変数名を使ってしまったら、どうなるでしょうか？予想できますか？やってみましょう！

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

エラーになりました！前回とは違うエラータイプです。**NameError**という、初めて見るエラータイプですね。作成されていない変数を使った時は、Pythonがエラーを教えてくれます。もし、このエラーに出くわしたら、記述したコードにタイプミスがないか確認してください。

ちょっと遊んで、何ができるか試してみてくださいね！

The print function

次に挑戦してみましょう。

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

単に `name` とタイプした時は、Pythonインタプリタが、変数`'name'`の *representation* を返します。ここでは、`M-a-r-i-a`という单なる文字の集まりで、シングルクォーテーション (`"`) に囲われています。しかし、`print(name)` と記述した時は、Pythonは変数の中身を出力します。クォーテーションはありません。

これからさらに詳しくみていきますが、`print()` は、関数から出力をする時や、複数行の出力をを行うときにも便利です。

Lists

数値と文字列の他にも、すべてのオブジェクトタイプを勉強しておきましょう。リスト(**list**)というものがあります。リストは、その名のとおり、オブジェクトの並びをもつものですね。:)

まずはリストを作りましょう：

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

はい、このリストは空っぽです。使いにくいですよね。では、くじ引きの番号のリストを作りましょう。この番号を何度も繰り返し書きたくないから、同時に変数に代入してしまいましょう。

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

よし、これでリストができました！このリストで何をしましようか？では、くじ引きの番号がいくつあるか、数えてみましょう。何の関数を使えばいいか、予想できますか？すでに知っていますよね！

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

そうです！`len()` がリストにあるオブジェクトの数を取得できます。便利ですね。では、くじ引きの番号をソートしてみましょう。

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

これは何も返していません。これはリストに表示される番号を、順番に並べ替えただけです。再度出力して、確かめてみましょう。:

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

ご覧のとおり、小さい順に並び替えられましたね。おめでとう！

逆順に並び替えてみたくなりましたか？やってみましょう。

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

簡単でしたね？リストに何かを追加したいときは、次のようにコマンドを記述してください。

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

最初の数字だけを出力したいときは、インデックス (**index**)を使って指定することができます。インデックスは、リストの先頭の要素から順に「0」、次に「1」と割り当てられています。次のとおり試してみてください。：

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

このように、リスト名と要素のインデックスを[]に記述することで、指定した要素を取り出すことができます。

他のインデックスも試して遊んでみてください。例えば、6, 7, 1000, -1, -6, -1000などをインデックスに指定するとどうなるでしょうか。コマンドを実行する前に予測してみましょう。結果はどうですか？

ご参考に、こちらのドキュメントにリストメソッドがすべて記されています。

<https://docs.python.org/3/tutorial/datastructures.html>

Dictionaries

辞書(ディクショナリ)について確認しましょう。リストに似ていますが、インデックスのかわりにキーと呼ばれる識別子で値を参照します。キーは文字列も数値も使えます。ディクショナリは次のように {} 括弧で囲んで作成します。

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

これで中身が空っぽのディクショナリができましたね。やったね！

では、つぎのコマンドを記述してみましょう。（あなた自身の情報に値をおきかえてみてもいいですよ）

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

このコマンドで、`participant` という名前の変数をつくって、3つのキーと値をもつ要素を作成しました。

- キーが `name` で、値が `'Ola'` の要素です。（`string`），
- キー `country` は、値 `'Poland'`（`string`），
- キー `favorite_numbers` は リスト `[7, 42, 92]`。（数字を3つ持つ `list`）.

次の構文で各キーの値を確認できます。

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

リストに似ていますね。しかし、ディクショナリーでは、インデックスを記憶する必要がなく、名前でいいのです

もし存在しないキーを参照しようとすると、どうなるでしょうか？予想できますか？実際にやってみましょう！

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

またエラーです。今回は **KeyError** というエラーが出ました。Pythonは、このディクショナリにキー `'age'` は存在しませんよ、と教えてくれています。

ディクショナリとリストはどう使い分ければよいのでしょうか？そうですね、これはゆっくり考えてみるべきポイントですね！この後の解答を読むまえに、考えてみてください。

- 必要なのは、順序付けられた一連のアイテムですか？リストを使いましょう。
- キーに対応する値が必要？キーから値を参照する？ディクショナリを使いましょう。

ディクショナリやリストは、生成後に値を変更できるオブジェクトです。これを *mutable* と呼びます。次のように、ディクショナリを作ったあとで、新しいキーと値を追加することができます。:

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

リストと同様に、`len()` 関数をディクショナリに使ってみましょう。ディクショナリでは、キーと値のペアの数を返します。コマンドを入力してやってみましょう。:

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

お分かり頂けたでしょうか。:) では、ディクショナリを使ってもう少し練習してみましょう。準備ができたら、次の行にいってみましょう。

ディクショナリの要素を削除する時は、`del` コマンドを使います。例えば、キー 'favorite_numbers' の要素を削除するには、次のように記述してください。

```
>>> del participant['favorite_numbers']  
>>> participant  
{'country': 'Poland', 'favorite_language': 'Python', 'name': 'Ola'}
```

このように、'favorite_numbers'のキーと値が削除されます。

同様に、次のように記述することで、すでにあるキーの値を変更することができます。:

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

これで、キー 'country' の値は、'Poland' から 'Germany' に変わりました。:) 面白くなつてきましたか？その調子です！

Summary

素晴らしいです！これで、あなたはプログラミングについて沢山のことを学びました。ここまでのことろをまとめましょう。

- エラー - あなたのコマンドをPythonが理解できない時にエラーが表示されます。
- 変数 - コードを簡単にまた読みやすくするために、文字や数値などのオブジェクトにつける名札。
- リスト - 複数の値（要素）が順に並んでいるもの。

- ディクショナリ - キーと値のペアの集合です。

次に進む準備はいいですか？ :)

Compare things

比較することは、プログラミングの醍醐味の1つです。簡単に比較できるものといえば、何でしょうか？ そうです、数字ですね。さっそくやってみましょう。

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

Pythonにいくつか比較する数字をあたえてみました。数字を比較するだけでなく、演算式の答えも比較することができます。便利でしょ？

2つの数字がイコールであるかどうかを比べる時に、イコールの記号が2つ `==` 並んでいます。疑問に思いましたか？ Pythonを記述する時、イコール1つ `=` は、変数に値を代入するときに使います。ですので、値同士が等しいかどうか比較するときは、必ず必ずイコール記号2つ `==` を記述してください。等しくないとするときは、上記の例のように `!=` と記述します。

次の2つはどうでしょうか

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

`>` と `<` は簡単でしたね。`>=` と `<=` はどうでしょうか？ それぞれの意味は、次のとおりです。

- `x > y`: x は y より大きい
- `x < y`: x は y より小さい
- `x <= y`: x は y 以下
- `x >= y`: x は y 以上

すばらしい! もう少しやってみましょう。

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

条件式が複数あって複雑になっても、その答えを出してくれます。とても賢いですね。

- **and** - `and` の左辺と右辺が共にTrueの場合、True。
- **or** - `or` の左辺あるいは右辺の少なくとも1つがTrueの時、True。

"comparing apples to oranges"という英語の表現を聞いたことはありますか? 文字通り訳すと「リンゴとオレンジを比較する」となり、「比較にならないものを比較する」という意味です。Pythonでも同じようなことをやってみましょう。:

```
>>> 1 > 'django'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unorderable types: int() > str()
```

Pythonは、数値(`int`)と文字列(`str`)の比較はできません。**TypeError**とエラーが表示され、2つのオブジェクトタイプが比較できないことを教えてくれています。

Boolean

偶然にも、ブール型 (**Boolean**) というあたらしいオブジェクトタイプを学びました。-- おそらく、ブール型は一番簡単なオブジェクトタイプです。

ブール型は、たった2つの値を持ちます。

- `True`
- `False`

Pythonを記述するときは、`True`の最初は大文字のT、残りは小文字です。`true`, `TRUE`, `tRUE` は間違いです。-- `True` と記述してください(もちろん `False` についても同様です。)

ブール型は、次のように変数に代入することもできます。:

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

このようなこともできます。

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

ブール型を使って、練習して遊んでみましょう。次のコマンドを試してみてください。

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

おめでとうございます！ブール型を理解することは、プログラミングでとても大事です。ここまでできましたね！

Save it!

ここまでインタプリタでPythonのコードを書いてきました。つまり、コードを1行づつしか書くことができませんでした。普通のプログラムはファイルに保存され、インタプリタあるいはコンパイラでプログラミング言語を処理して実行します。ここまで、私たちはプログラムを1行ごとにPython インタプリタで実行してきました。ここからは、1行以上のコードを実行していきましょう。次のような流れになります。

- Python インタプリタを終了します。
- お好きなエディタを起動します。
- Python ファイルとしてコードを保存します。
- 実行します！

これまで使っていたPython インタプリタを終了しましょう。`exit()` ファンクションを記述してください。

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

これで、コマンドプロンプトに戻りました。

前のチャプター [code editor](#) で、エディタを紹介しました。エディタを起動して、新しいファイルにコードを書いてみましょう。

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

あなたは、すでにベテランのpython開発者です。今日学んだコードを自由に書いてみてください。

コードを書いたら、わかりやすい名前をつけて保存しましょう。**python_intro.py**と名前をつけて、デスクトップに保存してください。ファイル名は何でもかまいません。ここで重要なことは、拡張子を**.py**とすることです。コンピュータにこのファイルは**python**で実行するファイルですとおしえます。

ファイルを保存したら、実行してみましょう！コマンドラインのセクションで学んだことを思い出して、ターミナルのディレクトリを変更して、デスクトップにしましょう。

Macでは、コマンドは次のようにになります。

```
cd /Users/<your_name>/Desktop
```

Linuxでは、次のようにになります。（"Desktop"のところは"デスクトップ"と表示されているかも知れません）：

```
cd /home/<your_name>/Desktop
```

Windowsでは、次のようにになります。

```
cd C:\Users\<your_name>\Desktop
```

うまくできない時は、質問してください。

次に、ファイルのコードを実行します。

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

できました！これで、あなたはファイルに保存されたPythonプログラムを実行できましたね。いい気分ですね。

では、ここからプログラミングに不可欠のツールを学んでいきましょう

If...elif...else

ある条件が成立するときに処理を行いたいという時に用いるのが、**if** 条件式です。

では、**python_intro.py** ファイルのコードを次のように書き換えてください。

```
if 3 > 2:
```

これを保存して実行すると、次のようなエラーがでます。

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

条件式 `3 > 2` がTrueの時、どのように処理をすべきかが記述されていませんね。では、Python に “It works!” と出力してもらいましょう。**python_intro.py** ファイルの中身を、次のとおりに書き換えてください。

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

2行目をスペース4つでインデントしていることに気が付きましたか？if文がTrueの時、どのコードを実行するかPythonに知らせる必要があります。スペース1つでもできますが、ほぼ全員のPythonプログラマーはスペース4つとしています。タブ1つも、スペース4つと同じです。

保存して、もう一度実行してみましょう。

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

What if not?

前述の例では、if文の条件式がTrueの時だけ、コードが実行されました。Pythonは、`elif` や `else` といった記述もできます。

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

これを実行した場合、次のように出力されます。

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

2が5より大きい場合、2行目のコマンドが実行されます。では、`elif` はどうなるのでしょうか？

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

それを実行すると…

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

どうなったか、わかりましたか？

Summary

3つのエクササイズを通して、これまでに学んだことは、、、：

- 比較 - 比較に用いる `>`, `>=`, `==`, `<=`, `<` そして `and`, `or` といった演算子があります。
- ブール型 - `True` と `False` 2つの値のみを持ちます。
- ファイルの保存 - コードはファイルに保存することで、大きなプログラムも実行できます。
- `if...elif...else` - 条件分岐することで、特定の条件によって処理を分けて実行することができます。

では、このチャプターの最後のパートに挑戦していきましょう！

Your own functions!

Pythonには `len()` のように関数があったのを覚えていますか？ ここでは、自分で関数を作る方法を学びます。

実行する機能をひとまとめにしたものを作ります。Pythonでは、functionは `def` というキーワードからはじまり、引数を含むことができます。簡単なものからはじめてみましょう。`python_intro.py` の中身を書きのコードに置き換えてください。：

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

hi()
```

あなたの最初の関数を実行する準備ができましたね！

ここであなたは、最後の行になぜ関数の名前を書いたのだろう、と疑問に感じたかもしれません。これは、Pythonがファイルを読み、上から下へ実行していくからです。関数を定義したあとに、もう一度その関数を書いて呼び出します。

では実行して、どうなるか見てみましょう：

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

簡単にできましたね！次に引数をつかった関数を作ってみましょう。先ほどの例を使います。`'hi'`という挨拶をする関数に、挨拶をする人の名前をいれてみます。

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

このとおり、関数に `name` という引数を足します。：

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

hi()
```

上記のように、`print` 関数の前に、インデントを2ついれる必要があります。`if` の条件式が真の時に、なにをすべきかという処理はインデントの後に記述します。実行して、どのように動くか見てみましょう。：

```
$ 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'
```

おっと、エラーがでてしまいました。Pythonがエラーメッセージを表示してくれています。定義した関数 `hi()` は、`name` という引数が必要ですが、関数を呼び出す時に引数を忘れてしまっています。最後の行を修正しましょう。:

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

実行してください。:

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

では、名前を変えてみたらどうなりますか？

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

再度実行してください。:

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

では、`Ola`や`Sonja`以外の名前を入れた時、どうなるかわかりますか？やってみて、予測が正しいか確認して下さい。このように出力されましたか。:

```
Hi anonymous!
```

すばらしいですね。挨拶をする人の名前を毎回何度も繰り返して書く必要がなくなりました。これが関数を作る理由です。何度も繰り返してコードを書く必要はありません！

もっとスマートなやり方を試してみましょう。--2人以上の名前があり、それぞれに対して条件をつけるのは大変ですよね。

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

では、実行してみましょう。：

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

おめでとうございます！Functionsの書き方を学びましたね。:)!

Loops

さあ、もう最後のパートですよ。あつという間ですね。:)

先ほどお話したとおり、プログラマーはめんどくさがりで、同じことを繰り返すことは好きではありません。プログラミングはすべてを自動的に処理したい。私たちはすべての人の名前ひとつひとつに対して挨拶をしたくないですよね？こういう時にループが便利です。

リストを覚えていますか？女の子の名前をリストにしてみましょう。:

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

名前を呼んで、全員にあいさつをしてみましょう。`hi` 関数が使えますね。ループの中でつかいましょう。:

```
for name in girls:
```

この `for` は `if` に似ています。この次に続くコードは、4つスペースを入れる必要があります。

ファイルに書かれるコードはこのようになります。:

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

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

実行してみましょう。:

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

ご覧のとおり、`girls` リストのすべての要素に対して、`for` の中にインデントして書かれたことが繰り返されています。

`for` 文では、`range` 関数をつかって指定した回数だけ繰り返すこともできます。:

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

これを実行すると、次のように出力されます:

```
1
2
3
4
5
```

`range` 関数は、引数に指定した開始と終了の数値から連続する数値の値を要素として持つリスト型のオブジェクトを作成します。

2つ目の引数（終了の数値）は、リストに含まれないことに注意してください。つまり、`range(1, 6)` は、1から5のことであり、6は含まれません。

Summary

以上です！おめでとう！頑張りました！これは簡単ではなかったと思います。自分を褒めてあげてくださいね。ここまで進めることができたのは、本当に素晴らしいことです！

次のチャプターにうつるまえに、少し気晴らしに、ストレッチやお散歩をして、目や身体を休ませてあげてくださいね。:)



What is Django?

Django (/dʒæŋgou/jang-goh) は、無料でオープンソースとして公開されているPythonを使用したWebアプリケーションフレームワークです。Webフレームワーク、つまり、Web対応のアプリケーション構築を早く簡単に開発する枠組みです。

ほら、Webサイトを構築する時、同じような構造が毎回必要になってきますよね。ユーザー認証（サインアップ、サインイン、サインアウト）、管理者用の画面、フォーム、ファイルのアップロードなど。

開発者たちはサイトを構築する度に同じ問題を抱えたため、みんなで力を合わせてフレームワークを開発しました。（Djangoはフレームワークのひとつです。）幸運なことに、私たちは、開発に必要な要素がすでに含まれているフレームワークを使って開発することができます。

フレームワークを使うことで、私たちは開発を土台から作り直すことを避けられます。また、新しいサイトを構築する際にかかる最初の準備に必要なコストを軽減します。

Why do you need a framework?

Djangoを本当に理解するために、サーバーの役割についてもう少し考えてみましょう。サーバーにWebページを配信してもらうようにするには、サーバーにそのような設定をする必要があります。

手紙が届くポストを想像してください。手紙はユーザからWebサーバーに送られるリクエストで、ポストはWebサーバーのポートのことです。Webサーバーはこのポストを監視して、手紙が届くとそれを読み、Webページから返事を送ります。送ろうとする時、コンテンツが必要ですね。Djangoは、あなたがそのコンテンツを作る手助けをするものです。

What happens when someone requests a website from your server?

Webサーバーにリクエストがあると、Djangoに伝えられ、リクエストの内容を把握しようとします。まずWebページのアドレスを調べ、リクエストに対して何をするか決めます。これは、Djangoの**urlresolver**が行います。（WebサイトのアドレスはURLと呼ばれます。Uniform Resource Locator の略です。-resolverとは「解決するもの」という意味ですの

で、*urlresolver* というのはうなずけますよね。)。あまり賢いとはいえません。Djangoは上から下にURLパターンを順に調べていきます。そこで何かがマッチすると、Djangoはビューと呼ばれる関数にリクエストを送ります。

郵便配達員を思い浮かべてください。配達員は、通りを歩き、ひとつひとつの家の番地と、手紙に書かれている番地を見比べて行きます。マッチする番地があったら、手紙をそこに置いていきます。*urlresolver*も同じ仕組みです。

ビュー関数では、面白いことが行われます。私たちは、データベースに情報を探しにいきます。時に、ユーザーがデータを変更するよう求めてきますよね？例えば、「私の仕事内容を変えて下さい」といった手紙のように。ビューは、まずあなたにその権限があるか確認します。次に、仕事内容を書き換えて、「完了しました！」というメッセージをあなたに送り返します。そして、ビューが反応を返して、DjangoがユーザーのWebブラウザに情報を送ります。

もちろん、上記の説明は、多少簡略化して説明しています。しかし、今ここでは、技術的なことを完璧に理解する必要はありません。概念が分かれば十分です。

これ以上詳細について深く説明するより、きっと、Djangoを使って実際に手を動かして作ってみる方がいいでしょう。重要な事はすべてその過程で学べますよ！

Djangoのインストール

このチャプターの一部はGeek Girls Carrots (<http://django.carrots.pl/>)のチュートリアルに基づいています。

このチャプターの一部はCreative Commons Attribution-ShareAlike 4.0 International License のライセンスによる [django-marcador tutorial](#)に基づいています。このdjango-marcador tutorialはMarkus Zapke-Gründemann et al.が著作権を保有しています。

仮想環境

Djangoをインストールする前に、まずはあなたのコンピュータの中のコーディング環境をきれいにしておくのに役立つとても便利な道具をインストールしてもらいます。このステップをとばすこともできますが、しかし、このステップをとばすことは全くお勧めしません。可能な限りベストなセットアップで始めることは将来のたくさんのトラブルからあなたを救うはずです！

さあ、**virtual environment** (`virtualenv`とも呼ばれています)を作りましょう。**virtual environment**はプロジェクト単位であなたのPython/Djangoセットアップを他から隔離します。つまり、あなたがひとつのウェブサイトにおこなったどんな変更もあなたが開発中の他のサイトに影響を及ぼさないということです。わかりましたか？

あなたがしなければならないのは、あなたが'virtualenv'を作成したいディレクトリを見つけることです（たとえばホームディレクトリなどです）。Windowsでは、ホームディレクトリは `C:\Users\Name\` と書かれているかもしれません（`Name` はあなたのログインネームです）。

このチュートリアルのために、ホームディレクトリに新しいディレクトリ'`djangogirls`'を作成します。

```
mkdir djangogirls  
cd djangogirls
```

`myvenv` という `virtualenv`を作成します。一般的なコマンドは以下のようになります：

```
python3 -m venv myvenv
```

Windows

新しい `virtualenv` を作成するために、コンソールを開き（コンソールについては何章か前にお話ししましたね。覚えてますか？）、`c:\Python35\python -m venv myvenv` を実行して下さい。たとえばこのように入力します：

```
C:\Users\Name\.djangogirls> C:\Python35\python -m venv myvenv
```

`C:\Python35\python` はあなたがPythonをインストールしたディレクトリ、`myvenv` はあなたの'virtualenv'の名前です。どんな名前でも使うことができますが、必ず小文字で表記し、スペース・アクセント記号・特殊文字は入れないでください。短い名前にしておくのもいいアイデアです—あなたはこの名前を何度も参照しますから！

Linux and OS X

LinuxやOS Xで'virtualenv'を作るときは、`python3 -m venv myvenv` と実行するだけです。

たとえばこんな感じです：

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

`myvenv` はあなたの `virtualenv` の名前です。どんな名前でも使うことができますが、必ず小文字で表記し、スペースは入れないでください。短い名前にしておくのもいいアイデアです—あなたはこの名前を何度も参照しますから！

virtualenvを使う

上に示したコマンドは仮想環境（基本的には一連のディレクトリとファイル）を含む `myvenv` という名前（あるいはあなたが選んだ名前）のディレクトリを生成します。次に我々がしたいのは、これを実行し、開始することです。

Windowsではこのように入力してください：

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

あるいはOS X、Linuxでは以下のように入力して下さい：

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

`myvenv` のところをあなたが選んだ `virtualenv` 名に置き換えることを忘れないで下さいね！

備考: `source` ではできない場合もあります。その場合は、代わりに以下のように入力してみてください：

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

あなたのコンソールのプロンプトが以下のように表示されるのを見て、あなたは `virtualenv` が起動したことに気がつくでしょう。

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

あるいはこういう表示かもしれません：

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

行頭に `(myvenv)` が現れたのに注目して下さい！

`virtual environment`(仮想環境)の中で作業しているとき、`python` は自動的に正しいバージョンのPythonを参照しますので、「`python3`」の代わりに「`python`」を使うことができます。

OK、これでDjangoのインストール前に入れておきたい依存関係の準備がすべて整いました。いよいよDjangoのインストールです！

Djangoのインストール

いまあなたの `virtualenv` は起動しているので、`pip` を使ってDjangoをインストールすることができます。コンソールの中で、`pip install django==1.11` (ここではダブルイコールサイン `==` を使います)と実行して下さい。

```
(myvenv) ~$ pip install django==1.11
Downloading/unpacking django==1.11
Installing collected packages: django
Successfully installed django
Cleaning up...
```

Windowsの場合

Windowsでpipを呼んだときにエラーが起きた場合は、あなたのプロジェクトのパス名がスペース・アクセント・特殊文字を含んでいないか確認してみて下さい（例 `C:\Users\User Name\.djangogirls`）。もし含まれている場合は、そのディレクトリを他のスペース・アクセント・特殊文字が含まれていない場所（`c:\djangogirls`など）に移動することを検討してみてください。移動したあとに上記のコマンドを試してみてください。

Linuxの場合

Ubuntu 16.04でpipを呼んだときにエラーが起きた場合は、virtualenv内でpipインストールをフィックスするために `python -m pip install -U --force-reinstall pip` を実行して下さい。

以上です！あなたは（ついに）Djangoアプリケーションを作成する準備が整いました！

Your first Django project!

このチャプターの一部はGeek Girls Carrots (<http://django.carrots.pl/>) のチュートリアルに基づいています。

このチャプターの一部はCreative Commons Attribution-ShareAlike 4.0 International License のライセンスによる [django-marcador tutorial](#) に基づいています。このdjango-marcador tutorialはMarkus Zapke-Gründemannらが著作権を保有しています。

ここからは、シンプルなブログを作っていきますよ！

最初のステップは、Djangoのプロジェクトを新しく作成します。つまり、Djangoのスクリプトを実行し、これから使う沢山のファイルやディレクトリを自動生成して、プロジェクトの骨格を作ります。

Djangoでは、ファイルやディレクトリの名前がとても重要です。ここで作成するファイルの名前は、後から変えるべきではありません。ファイルを移動させるのもいいアイディアとはいません。Djangoでは、重要なファイルを決められたファイル構成で作成しておくことが必要です。

virtualenv（仮想環境）を実行しているでしょうか。コンソールに (myvenv) という括弧が表示されていなければ、virtualenvを実行してください。チャプター **Django installation** の **Working with virtualenv** で、仮想環境を実行する方法を説明しました。覚えていますか？次のコマンドを入力ですよ。: Windowsでは、 myvenv\Scripts\activate , Mac OS / Linuxでは、 myvenv/bin/activate でしたね。

プロジェクトは、コンソール上で次のコマンドで作成します。（いいですか？ (myvenv) ~/djangogirls\$ は入力の必要がありませんよ。）：

補足 コマンドのコマンドの最後にピリオド (.) があることを確認してくださいね。これば、現在の作業ディレクトリに Django をインストールするということを示すので、とても重要なのです。

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

On Windows:

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

~~補足 コマンドの最後にピリオド(.)があることを確認してくださいね。これば、現在の作業ディレクトリに Django をインストールするということを示すので、とても重要なです。~~

`django-admin.py` は、必要なディレクトリとファイルを作成するスクリプトです。次のようなファイル構造が作成されましたね。:

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

`manage.py` はサイト管理用のスクリプトです。これで、他のものを一切インストールすることなく、コンピューター上で Web サーバーを動かすことができます。

`settings.py` は、サイトの設定ファイルです。

どこに手紙を配達するか番地を確認する郵便配達員の話を覚えていますか? `urls.py` ファイルは、`urlresolver` をつかった URL のパターンのリストを含んでいます。

今は他のファイルについては無視しておきましょう。触ることはありません。間違って削除してしまわないようにさえしておけば大丈夫です!

Changing settings

次に、`mysite/settings.py` の中を少し変更していきましょう。エディタでファイルを開いて下さい。

サイトのタイムゾーンを修正しておきましょう。[ウィキペディアの `timezones list`] (http://en.wikipedia.org/wiki/List_of_tz_database_time_zones) に一覧がありますので、あなたがいるタイムゾーン(TZ)を探してください。(例 `Europe/Berlin` `Asia/Tokyo`)

開いたファイルの中に、`USE_TZ` や `TIME_ZONE` と書かれた行をみつけてください。タイムゾーンを次のように変更しましょう。:

```
LANGUAGE_CODE = 'ja-JP'

TIME_ZONE = 'Asia/Tokyo'

USE_TZ = False
```

Setup a database

世の中には沢山のデータベースソフトウェアがありますが、今日は `sqlite3` というデータベースを使います。

これは、`mysite/settings.py` ファイルの次の箇所で、すでにセットアップされています。:

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

次のコマンドをコンソールで実行して、データベースをこのブログに作成しましょう。: `python manage.py migrate` (`manage.py` ファイルが含まれている `djangogirls` ディレクトリでコマンドを実行してください。) 次のような画面になれば、実行は上手くいっています。:

```
(myvenv) ~/djangogirls$ python manage.py migrate  
Operations to perform:  
  Apply all migrations: admin, contenttypes, auth, sessions  
Running migrations:  
  Applying contenttypes.0001_initial... OK  
  Applying auth.0001_initial... OK  
  Applying admin.0001_initial... OK  
  Applying sessions.0001_initial... OK
```

これでデータベースができました。サーバーを動かして、Webサイトがうまく動いているか確認しましょう！

`manage.py` ファイルを含むディレクトリ（`djangogirls` ディレクトリ）にいることを確認してください。コンソールで、`python manage.py runserver` とコマンドを入力してサーバーを動かします。:

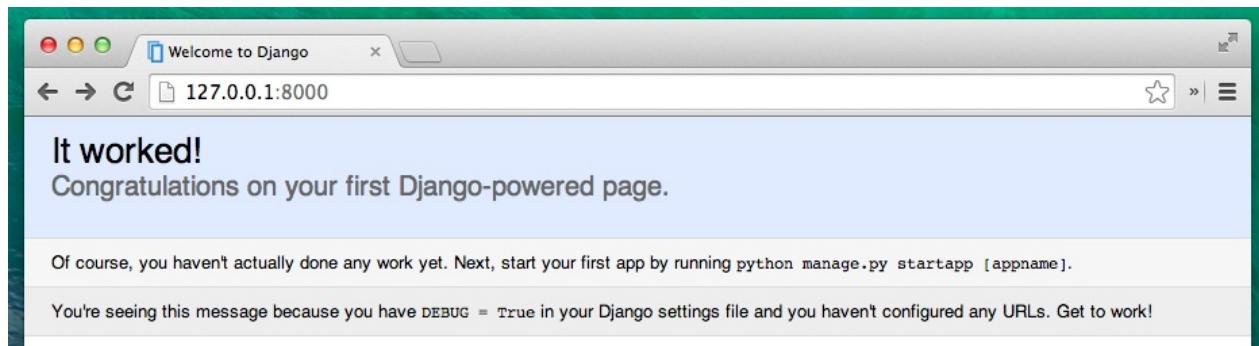
```
(myvenv) ~/djangogirls$ python manage.py runserver
```

サイトがきちんと動いているか確認してみましょう。ブラウザを開いてください。(Firefox, Chrome, Safari, Internet Explorerなど、いつも使っているブラウザでいいですよ) 次のアドレスを入れてください。:

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

ウェブサーバーは、あなたがコマンドプロンプトで実行を停止するまで動き続けています。さらにコマンドを実行する場合は、新しいターミナルウインドウを開くか（その時、virtualenvをまた実行することを忘れないで下さい）、あるいは、実行中のウェブサーバーを停止してください。停止する時は、CTRL+C（コントロールとCボタンを同時に押す）です。（Windowsの方は、Ctrl+Breakで停止することができます。）

おめでとうございます！これで、はじめてのWebサイトを作成して、サーバーを使って動かすことができました。すごいですね！



次のステップに進みましょう！いよいよ、コンテンツを作成していきます！

Django models

さて、ブログの中のポストを格納するものが欲しいですよね。そのためには オブジェクト についてちょっとお話しします。

オブジェクト

プログラミングには オブジェクト指向プログラミング という概念があります。それは退屈なプログラムを繰り返し書く代わりにモデルになるものを作つて、それが他とどう作用するかを定義するという考え方です。

じゃあオブジェクトって何なの？って思いますよね。オブジェクトは状態（プロパティ）と命令（アクション）の塊です。ピンと来ないでしようから例を挙げましょう。

猫をモデルにしたいときは、 猫 (cat) オブジェクトを作ります。そのプロパティは、 色 (color) 、 年齢 (age) 、 機嫌 (mood) (いい、悪い、眠い) 、 飼い主 (owner) (人 (person) オブジェクトですね、捨て猫ならそのプロパティは空白) です。

猫 のアクションは、 喉を鳴らす (purr) 、 引っ搔く (scratch) 、 餌を食べる (feed) (キャットフード (CatFood) などで、それはまた 味 (taste) というプロパティを持つ別のオブジェクトになるでしょう。)

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

```
CatFood
-----
taste
```

つまり、オブジェクト指向とは実際の物をプロパティ（ オブジェクト・プロパティ と呼びます）を持つコードと命令（ メソッド と呼びます）で表現するという考え方です。

ではブログポストはどういうモデルになるでしょうか。ブログが作りたいんですね？

それにはブログポストとは何か、それはどんなプロパティがあるかという問い合わせに答えなければなりません。

まず確実なのはブログポストにはコンテンツと表題が必要ですね。それからそれを書いた人が分かるといいでしょう。最後にポストの作成公開した時間も分かるといいですね。

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

ではブログポストがどうなればいいですか？ポストをが公開されるといいですね？なので `publish` メソッドが必要です。

達成したいことが分かったので、Djangoでモデリングの開始です！

Django model

オブジェクトが何か分かったので、ブログポストのDjangoモデルを作りましょう。

Djangoのモデルは特別なオブジェクトで、データベースに格納されます。データベースはデータの集まりです。ここにユーザーやブログポストの情報を格納します。データを格納するのにSQLiteデータベースを使います。これはDjangoのデフォルトのデータベースで、今はこれで十分です。

データベースの中のモデルは、列（フィールド）と行（データ）があるスプレッドシートと思ってもらっても結構です。

Creating an application

全部をきちんと整理しておくため、プロジェクトの中に別のアプリケーションを作ります。初めから全てを整理しておくのはとっても良いことです。アプリケーションを作るには、次のコマンドをコンソールの中で走らせなければなりません。（`manage.py` ファイルがある `djangogirls` ディレクトリから）：

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

新しいブログディレクトリが作られて、今沢山のファイルがそこに入っているのに気がついたでしょう。ディレクトリとファイルはこんな風に見えるはずです：

```
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
```

アプリケーションを作ったら、Djangoにそれを使うように伝えないと困ります。それは `mysite/settings.py` ファイルの中でやります。まず `INSTALLED_APPS` をみつけて) の上に `'blog'` という一行を追加します。

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

Creating a blog post model

`blog/models.py` ファイルで `Models` と呼ばれる全てのオブジェクトを定義します。これがブログポストを定義する場所です。

`blog/models.py` を開いて全部削除し、下のコードを書きます：

```

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

class Post(models.Model):
    author = models.ForeignKey('auth.User')
    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

```

`str` の両側に2つのアンダースコア（`_`）がちゃんと入っているか確認しましょう。
これはPythonでよく使われて"ダッパー"(ダブルアンダースコア)と呼んでいます。

難しそうでしょ？でも大丈夫！ちゃんと説明しますから。

`from` とか `import` で始まる行は全部他のファイルから何かをちょこつとずつ追加する行です。なので色々なファイルから必要な部分をコピペする代わりに `from ... import ...` で必要部分を入れられるんです。

`class Post(models.Model):` - この行が今回のモデルを定義します(これがオブジェクトです)。

- `class` はオブジェクトを定義しますよ、ということを示すキーワードです。
- `Post` はモデルの名前で、他の名前をつけることもできます(が、特殊文字と空白は避けなければいけません)。クラス名は大文字で始めます。
- `models.Model` はポストがDjango Modelだという意味で、Djangoが、これはデータベースに保存すべきものだと分かるようにしています。

さて今度はプロパティを定義しましょ

う：`title`、`text`、`created_date`、`published_date`、それに `author` ですね。それにはまずフィールドのタイプを決めなければいけません。(テキスト？数字？日付？他のオブジェクト、例えばユーザーとの関係は？)

- `models.CharField` - テキスト数を定義するフィールド
- `models.TextField` - これは制限無しの長いテキスト用で、ブログポストのコンテンツに理想的なフィールドでしょ？
- `models.DateTimeField` - 日付と時間のフィールド
- `models.ForeignKey` - これは他のモデルへのリンク

コードの細かいところまでは説明し出すと時間がかかるので、ここではしませんが、モデルのフィールドや上記以外の定義のやり方について知りたい方は是非Djangoドキュメントを見てみて下さい。(<https://docs.djangoproject.com/ja/1.11/ref/models/fields/#field-types>)

`def publish(self):` は何かと言うと、これこそが先程お話したブログを公開するメソッドそのものです。`def` は、これはファンクション（関数）/メソッドといいう意味です。`publish` というのはこのメソッドの名前で、変えることもできます。メソッドの名前に使っていいのは、英小文字とアンダースコアで、アンダースコアはスペースの代わりに使います（例えば、平均価格を計算するメソッドは `calculate_average_price` っていう名前にします）

メソッドは通常何かを `return` します。一つの例が `__str__` メソッドにあります。このシナリオでは、`__str__()` を呼ぶと、ポストの表題のテキスト(**string**)が返ってきます。

もしモデルがまだはつきりつかめないようだったら、気軽にコーチに聞いて下さい！特にオブジェクトとファンクションを同時に習ったときはとても複雑なのはよく分かってますから。でも前ほど魔法みたいじゃないといいですけど！

データベースにモデル用のテーブルを作る

最後のステップは新しいモデルをデータベースに追加することです。まず、（今作った）モデルの中で少し変更があったことをDjangoに知らせなければなりませんから、`python manage.py makemigrations blog` とタイプします。こんな感じです。

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

Djangoがデータベースに入れる爲の移行ファイルを作ってくれているので、`python manage.py migrate blog` とタイプするとこうなるでしょう。

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

やった～！ポストモデルがデータベースに入りました。どうなったか見たいでしょ？次へ進みましょう！

Django admin

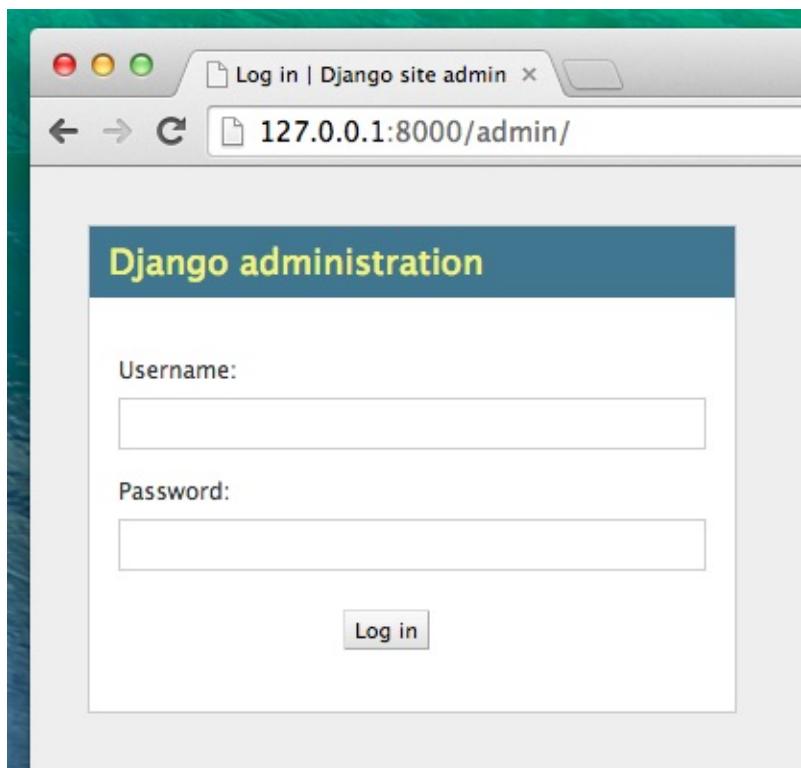
今作成したポストを追加、編集、削除するのにDjango adminを使います。
まず `blog/admin.py` ファイルを開いて、コンテンツをこれに置き換えましょう。

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

admin.site.register(Post)
```

見て分かる通り、前回定義したPostモデルをimportしています。モデルをadminページで見れるようにするには、モデルを `admin.site.register(Post)` で登録する必要があります。

ではPostモデルを見てみましょう。Webサーバーを走らせるにはConsoleの中で `python manage.py runserver` を走らせることを覚えておいて下さい。ブラウザに行って <http://127.0.0.1:8000/admin/> とアドレスバーにタイプします。こんなログインページが出ますね。

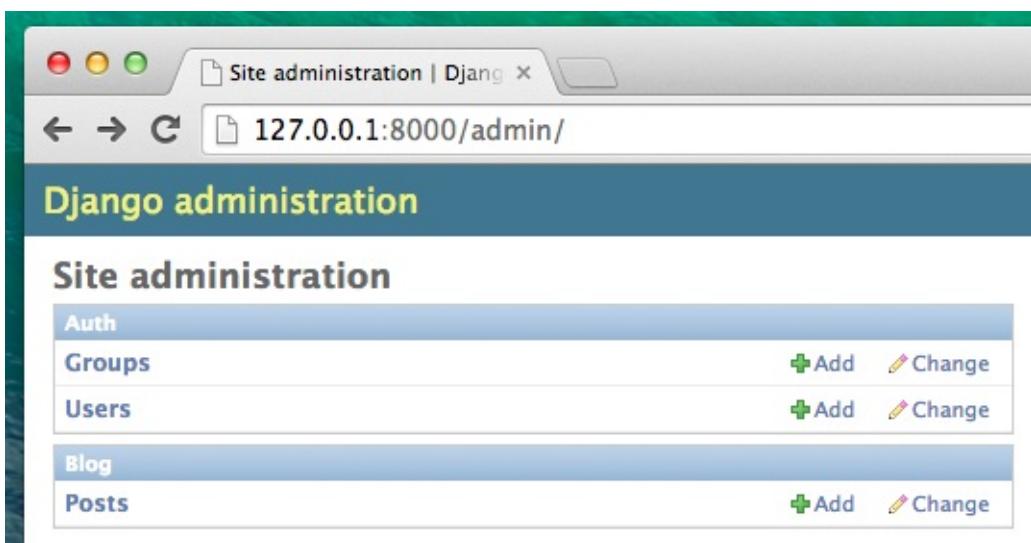


ログインするには、`superuser`（サイトの全てを管理するユーザー）を作る必要があります。コマンドラインに戻って、`python manage.py createsuperuser` とタイプし `enter`、それからユーザー名（小文字スペース無し）、聞かれたらメールアドレスとパスワードを入

れます。タイプしてある間パスワードは見えなくても大丈夫、それが正常です。タイプして Enter を押して続けましょう。そうすればこのように見えるはずです。（ユーザー名とパスワードは今あなたがタイプしたものです。）

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

ブラウザに戻って superuser でログインすると、Django admin ダッシュボードが見えるでしょう。



Posts に行って少し試してみてください。5~6のブログポストを入れてみましょう。コンテンツは心配しなくて大丈夫。今はとりあえずこのチュートリアルからテキストをいくつかコピペするだけでいいです。

少なくとも2~3のポスト（全部じゃなくても）がポスト時刻を表示しているかだけ確認してください。これが後で役立ちます。

Django administration

Welcome, olasitarska. Change password / Log out

Home > Blog > Posts > Add post

Add post

Author: olasitarska

Title: Cras justo odio, dapibus ac facilisis in, eq

Text:

Cras mattis consectetur purus sit amet fermentum. Donec id elit non mi porta gravida at eget metus. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam quis risus eget urna mollis ornare vel eu leo. Curabitur blandit tempus porttitor. Cras mattis consectetur purus sit amet fermentum.

Fusce dapibus, tellus ac cursus commodo, tortor mauris condimentum nibh, ut fermentum massa justo sit amet risus. Curabitur blandit tempus porttitor. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec sed odio dui. Aenean eu leo quam. Pellentesque ornare sem lacinia quam venenatis vestibulum.

Created date: Date: 2014-07-07 Today |

Time: 23:07:34

Published date: Date: 2014-07-07

Time: 23:07:34

もし Django adminについてもっと知りたければ、Django's documentationを見て下さい。

<https://docs.djangoproject.com/ja/1.11/ref/contrib/admin/>

さてコーヒー飲むか何かつまんでください。初Django modelを作ったんですから休憩しましょう！

Deploy!

補足: このチャプターはちょっと難しいことが沢山書かれています。頑張って最後までやりきってください。デプロイはウェブサイトを開発するプロセスの上で、とても重要な部分ですが、躊躇やすいポイントも多く含まれています。チュートリアルの途中にこのチャプターを入れています。そういういた躊躇やすい箇所はメンターに質問して、あなたが作っているウェブサイトをオンラインでみれるようにしてください。言い換えれば、もし時間切れでワークショップ内でチュートリアルを終わらせることができなかったとしても、この後のチュートリアルはきっと自分で終わらせることができるでしょう。

今、あなたのウェブサイトはあなたのコンピューターでのみ見ることができますね。では、これをデプロイする方法を学んでいきましょう。デプロイとは、あなたが作っているアプリケーションをインターネットで公開することです。あなた以外の人もウェブサイトを見るができるようになりますよ。:)

これまでに学んだとおり、ウェブサイトはサーバーに置かれています。様々なサーバーがありますが、ここでは最もシンプルなやり方でデプロイすることができるものを使いましょう。[Heroku](https://devcenter.heroku.com/articles/getting-started-with-django)です。Herokuは、多くの人がアクセスするものではない小さいアプリケーションは無料で公開できます。今回には最適でしょう。

Heroku のチュートリアル (<https://devcenter.heroku.com/articles/getting-started-with-django>) に従ってすすめていきます。以下に同内容を記していますので、このまま進めていってください。

The requirements.txt file

最初に、`requirements.txt` ファイルを作成します。あなたのサーバーにどんなPythonパッケージがインストールされる必要があるか、Herokuに伝えるものです。

その前に、Herokuを使うために必要ないくつかのパッケージをインストールしておきましょう。コンソールを開いて、`virtualenv` を実行し、次のように入力して下さい:

```
(myvenv) $ pip install dj-database-url gunicorn whitenoise
```

インストールが終わったら、`djangogirls` ディレクトリに行き、次のコマンドを実行します:

```
(myvenv) $ pip freeze > requirements.txt
```

これで、`requirements.txt` とよばれるファイルが作成されます。必要なパッケージのリストが書かれています。どんなPythonライブラリを使っているかといった情報です。（例えばDjangoとか）:)).

補足: `pip freeze` は、あなたのvirtualenvにインストール済みの全てのPythonライブラリを一覧にして出力します。その `pip freeze` した出力先を、`>` の後に示しファイルに保存します。`> requirements.txt` を含まずに `pip freeze` だけで実行してみて、何が起こるか試してみるとよいでしょう。

ファイルを開いて、最終行に次の1行を追加しましょう:

```
psycopg2==2.5.4
```

これは、あなたのアプリケーションをHerokuで動かすために必要な1行です。

Procfile

次に必要なものは、`Procfile`です。このファイルが、どのコマンドを実行してウェブサイトをスタートするかHerokuに伝えます。エディタを開いて、`djangogirls` ディレクトリに `Procfile` という名前のファイルを作成して下さい。ファイルに次のとおり入力しましょう:

```
web: gunicorn mysite.wsgi
```

この1行が何を意味しているのでしょうか。私たちは `web` アプリケーションをデプロイしようとしていること、そして `gunicorn mysite.wsgi` というコマンドを実行することでデプロイすることを意味しています。（`gunicorn` は、Djangoの `runserver` コマンドのもっとパワフルなものと考えて下さい。）

これで完了です！保存しましょう。

The `runtime.txt` file

Herokuに使っているPythonのバージョンを伝えなければいけません。`djangogirls` ディレクトリに `runtime.txt` ファイルを作ってその中に書きます。エディタで新規ファイルを作成して、次のように書いてください:

```
python-3.5.2
```

mysite/local_settings.py

コンピューター上のローカルとサーバーでは設定に違いがあります。Herokuとコンピューターでは別のデータベースをそれにつかっています。そこで、別途ファイルを作成し、ローカル環境で動かすための設定を保存しておく必要があります。

では、ファイルを作成しましょう。`mysite/local_settings.py` というファイルです。その中には、`mysite/settings.py` ファイルからの `DATABASE` の設定が必要です。次のように記述してください：

```
import os
BASE_DIR = os.path.dirname(os.path.dirname(__file__))

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

DEBUG = True
```

保存しておきましょう：)

mysite/settings.py

すべき事がまだあります。ウェブサイトの `settings.py` ファイルに変更を加えておきましょう。エディタで、`mysite/settings.py` ファイルを開いて下さい。最終行に、次のとおり追加しましょう：

```
import dj_database_url
DATABASES['default'] = dj_database_url.config()

SECURE_PROXY_SSL_HEADER = ('HTTP_X_FORWARDED_PROTO', 'https')

ALLOWED_HOSTS = ['*']

STATIC_ROOT = 'staticfiles'

DEBUG = False

try:
    from .local_settings import *
except ImportError:
    pass
```

このファイルは、Herokuに必要な構成だけでなく、`mysite/local_settings.py` ファイルがある時にはローカルの設定にも重要な役割となります。

保存してください。

mysite/wsgi.py

次に `mysite/wsgi.py` ファイルを開き、最終行に次のとおり追加してください:

```
from whitenoise.django import DjangoWhiteNoise  
application = DjangoWhiteNoise(application)
```

よし、できましたね！

Heroku account

では、Heroku toolbeltをインストールしましょう。Herokuをコマンドライン操作を行うためのアプリケーションです。（すでに設定でインストール済みでしたら飛ばしてください。）インストーラーはここから入手できます: <https://toolbelt.heroku.com/>

WindowsでHeroku toolbeltをインストールする方は、インストールするコンポーネントを選択する際に"Custom Installation"を選択してください。コンポーネントを選択すると、さらにチェックボックスがありますので、"Git and SSH"にチェックを入れて下さい。

Windowsの方は、コマンドプロンプトの `PATH` にGitとSSHを追加するために、次のように入力し実行してください: `setx PATH "%PATH%;C:\Program Files\Git\bin"`。コマンドプロンプトを一旦閉じて、再度起動しましょう。すると変更が適用されます。

コマンドプロンプトを再起動したら、`djangogirls` フォルダに移動し、virtualenvを実行することを忘れないでくださいね！(ヒント: [Check the Django installation chapter](#))

Herokuの無料アカウントを作成してください: <https://id.heroku.com/signup/www-home-top>

コマンドプロンプトでHerokuにログインします。コマンドプロンプトを起動して、次のコマンドを入力しましょう:

```
$ heroku login
```

SSHキーがない場合はここで自動的に作成されます。SSHキーは、コードをHerokuにpushするためには必要なものです。

Git

Gitとは、バージョン管理システムです。多くのプログラマーがつかっています。ファイルの変更履歴を記録・追跡するシステムです。そのため、一度編集したファイルを特定の変更前の状態に戻したり、編集箇所の差分を表示したりすることができます。Herokuでは、Gitのリポジトリを使って、プロジェクトのファイルを管理します。

djangogirls ディレクトリに、`.gitignore` という名前のファイルを作成しましょう。次のように記述してください:

```
myvenv  
__pycache__  
staticfiles  
local_settings.py  
db.sqlite3  
*.py[co]
```

Git に `local_settings.py` は無視してアップロードしないように伝えています。あなたのコンピューター上（ローカル環境）でのみ動作します。それでは保存しておきましょう。ファイル名の最初にあるドットはとても重要ですよ。

次に、Gitのリポジトリを作成し、これまでの変更を記録しておきましょう。

補足: リポジトリを作成する前に、現在作業しているディレクトリを確認しておきましょう。`pwd` コマンドでしたね。`djangogirls` フォルダになっていればOKです。

コンソールを開き、次のコマンドを実行しましょう:

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

Gitリポジトリを作成するのは、プロジェクトにつき最初の1度だけです。

間違ったファイルを追加したりコミットするがないように、`git add` コマンドを実行する前や、どのような変更を加えたか定かでない時は、`git status` コマンドを実行しておくとよいでしょう。`git status` コマンドを実行すると、ステージされた変更内容、されていない変更内容、Gitによる追跡の対象外となっているファイルやブランチのステータス等が表示されます。次のような情報が出力されることでしょう:

```
$ git status
On branch master

Initial commit

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

.gitignore
Procfile
mysite/__init__.py
mysite/settings.py
mysite/urls.py
mysite/wsgi.py
manage.py
requirements.txt
runtime.txt

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

この変更を保存しましょう。コンソールで次のコマンドを実行してください:

```
$ git add -A .
$ git commit -m "My Django Girls app"
[master (root-commit) 2943412] My Django Girls
 7 files changed, 230 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 Procfile
 create mode 100644 mysite/__init__.py
 create mode 100644 mysite/settings.py
 create mode 100644 mysite/urls.py
 create mode 100644 mysite/wsgi.py
 create mode 100644 manage.py
 create mode 100644 requirements.txt
 create mode 100644 runtime.txt
```

Pick an application name

アプリケーションの名前を考えましょう。オンライン上にブログを作る際に、そのURLは [your blog's name].herokuapp.com となります。[your blog's name] には、だれもつけていない名前をつける必要があります。この名前は、Django blog app や mysite といったこれまでに作成したプロジェクト名等と関連している必要はありません。あなたの好きな名前をつけていいですよ。ただし、小文字の英数字とダッシュ(-)の組み合わせにしてください。大文字や記号はつかえません。

アプリケーション名が決まれば（おそらくあなたの名前やニックネームが入ったものでしょう。）、次のコマンドを実行してください。ここでは、アプリケーションの名前を `djangogirlsblog` としていますので、あなたのアプリケーション名に置き換えてください：

```
$ heroku create djangogirlsblog
```

補足： `djangogirlsblog` を、Herokuで使うあなたのアプリケーション名に置き換えることを忘れないでください！

もし名前が自分で思いつかない時は、かわりに次のコマンドを実行してください。

```
$ heroku create
```

Herokuが代わって名前をつけてくれます。（おそらく `enigmatic-cove-2527` といった感じに。）

もし、あとでHerokuのアプリケーション名を変更したくなれば、次のコマンドでいつでも変更することができます。（`the-new-name` を変更する新しい名前におきかえてください。）

```
$ heroku apps:rename the-new-name
```

補足： アプリケーション名を変更したら、あなたのウェブサイトのURLは `[the-new-name].herokuapp.com` に変わります。

Deploy to Heroku!

ここまで沢山のインストールや設定がありましたね。けど、次がこのチャプターの最後です。いよいよデプロイします！

先ほど `heroku create` を実行した時に、自動的に `heroku` という名前でリモートリポジトリが作成されています。アプリケーションをデプロイするには、これをGitでプッシュするだけです：

```
$ git push heroku master
```

補足： これを最初の実行後は、Heroku が `psycopg` をコンパイルしてインストールするのにともない沢山の出力が表示されることでしょう。暫く待って、最後の方に `https://yourapplicationname.herokuapp.com/ deployed to Heroku` というような出力があれば、成功した印です。

Visit your application

これであなたのコードを Heroku にデプロイすることができました。そして、プロセスは `Procfile` と指定されています。(先ほど、プロセスタイプを `web` と選びましたね。) 最後に、Heroku のウェブプロセスを起動します。

次のコマンドを実行してください:

```
$ heroku ps:scale web=1
```

ウェブプロセスが 1 つ起動しました。このブログアプリケーションはとてもシンプルなものなので、沢山のパワーは必要ありません。1 つのウェブプロセスで十分です。Heroku でそれ以上のウェブプロセスを起動することは可能ですが、今は無料ではないようです。(ところで、Heroku ではこのプロセスを "Dynos" と呼んでいます。この言葉が出てきてもびっくりしないでくださいね。)

さて、ブラウザでアプリケーションを開いてみましょう。`heroku open` コマンドです。

```
$ heroku open
```

補足: エラーページが表示されますね。これについては、すぐ説明するので心配しなくていいですよ。

コマンドで、<https://djangogirlsblog.herokuapp.com/> といったあなたのアプリケーションの URL がブラウザで開かれます。おそらくエラーページが表示されることでしょう。これは、私たちがまだ `admin view` しか作成していないからです。URL の後ろに `admin/` をつけてアプリケーションが動いているのをみてみましょう。(例

<https://djangogirlsblog.herokuapp.com/admin/>)

エラーが表示されたのは、Heroku にデプロイする時に新しいデータベースを作成したため、まだデータベースが空っぽの状態だからです。そこで、プロジェクトの最初にローカルにデータベースをセットアップした時のように、再度 `migrate` コマンドを実行します:

```
$ heroku run python manage.py migrate
$ heroku run python manage.py createsuperuser
```

コマンドプロンプトは、ユーザー名とパスワードを選ぶよう再度きいてきます。これは、オンライン上のウェブサイトのログインに必要となるログイン情報です。ブラウザを再読み込みして、完了です！

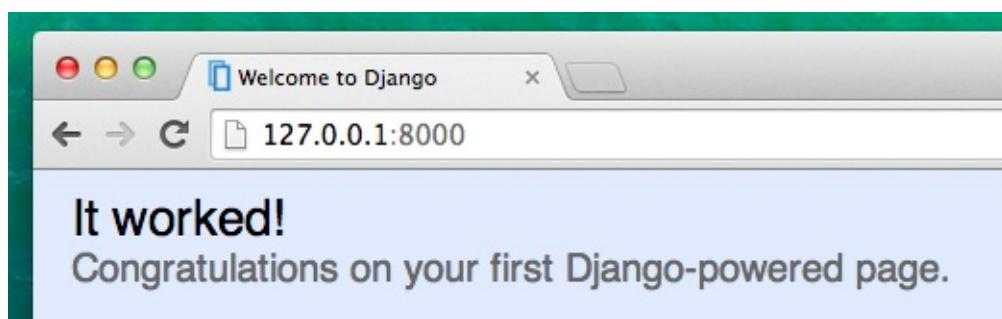
これで、あなたのウェブサイトがブラウザでみられるようになりました！おめでとう！！：)

Django urls

最初のウェブページを立てましょう、あなたのブログです。始めに、djangoのURLについて少し学びましょう。

What is a URL?

URLは簡単に言えばWEB上のアドレスです。サイトのURLは、ブラウザのアドレスバーで見ることができます。（そう、`127.0.0.1:8000` や <http://djangogirls.com> がURLです。）



インターネットの全てのページはURLを持っています。それによって、これから作るアプリケーションが、URLを指定してアクセスしてきたユーザに、何を見せたらいいのかわかるのです。Djangoでは `URL_conf`（URL設定）と呼ばれるものを使います。これは、指定されたURLに合わせてDjangoがどのviewを返したらいいか判断する仕組みのことです。

How do URLs work in Django?

`mysite/urls.py` を開いて、中身を見てみると：

```
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)),
]
```

ご覧のとおり、Djangoは既にこのようなものを置いてくれています。

で始まっている行はコメント行です。これはPythonによって実行されない行です。とっても便利でしょう？

前の章で訪れたadminのURLについてはすでに書いてありますね。

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

admin/ で始まる全てのURLについて、Djangoが返すべきviewをこの行で指定しています。今回の場合、adminで始まるURLをたくさん作ることになりますが、その全てをこの小さいファイルに書くようなことはしません。この方がきれいで読みやすいです。

Regex

どのやつて Django はビューに URL をマッチするのかと思うかもしれません。そうです、この部分はひとひねりしています。Django は regex 、正規表現を使います。Regex は多くの（本当に多くの）検索パターンのルールを持っています。理解するのは簡単では無いですが、今は心配しないで下さい。将来、それらを正確に理解できるでしょう。今回は必要なものだけ使っていきます。

あまり悩まないで簡単な例で説明しますね。

`http://www.mysite.com/post/12345/` のようなアドレスを持つウェブサイトがあるとします。ここで、`12345` はブログポストの番号です。ブログポストの番号ごとに、それぞれ別々の view を書くなんて絶対無理ですよね。Django では `http://www.mysite.com/post/<a number>/` と書くだけで、後は全部お任せできるんです！

Your first Django url!

さあ最初のURLを作りましょう！`http://127.0.0.1:8000/` はブログの入口ページなので、投稿したブログポストのリストを表示したいです。

`mysite/urls.py` ファイルは簡潔なままにしておきたいので、`mysite/urls.py` では `blog` アプリから URL をインポートするだけにしましょう。

コメントされた行（# で始まる行）を消して、入口ページの URL ('') には `blog.urls`` をインポートするように書いてください。

`mysite/urls.py` ファイルはこのようになります：

```
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')),
]
```

これでDjangoは '<http://127.0.0.1:8000/>' に来たリクエストは `blog.urls` ヘリダイレクトするようになり、それ以降はそちらを参照するようになります。

blog.urls

新しく `blogs/urls.py` という空のファイルを作つて下さい。そして最初の2行を以下のように書きます：

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

これはDjangoのメソッドと、 `blog` アプリの全てのビュー（といっても、今は一つもありません。すぐに作りますけど！）をインポートするという意味です。

その後、最初のURLパターンを追加します。

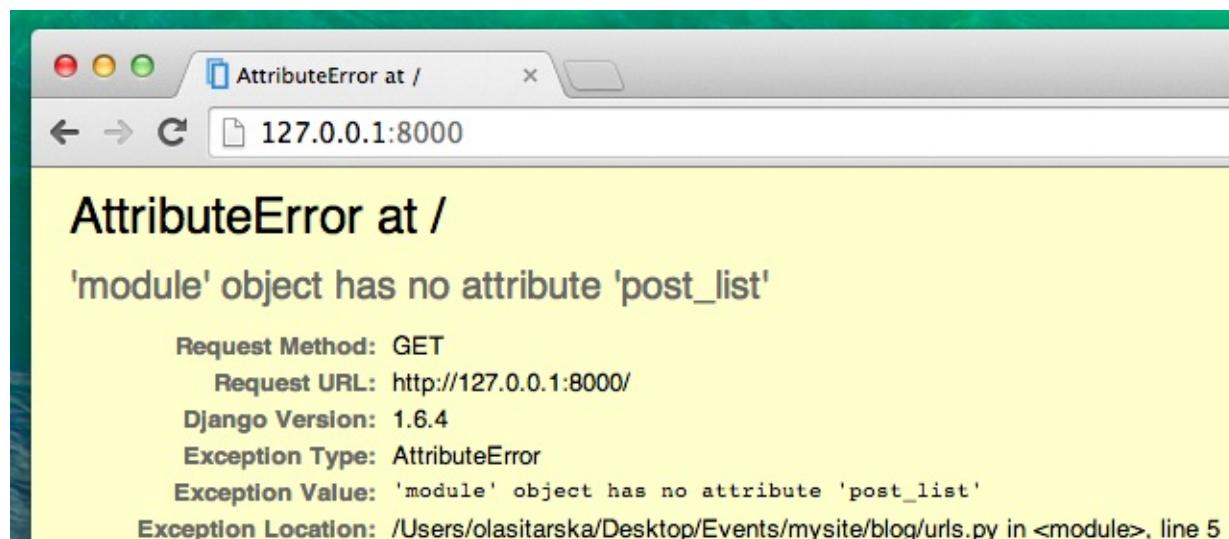
```
urlpatterns = [
    url(r'^$', views.post_list),
]
```

これは `^$` というパターンのURLを `post_list` というビューに割り当てたことを意味します。`^$` とは何を意味しているのでしょうか。それは正規表現のマジックです:) 分解してみましょう：

- 正規表現での `^` は「文字列の開始」を意味します。ここからパターンマッチを始めます。
- `$` は「文字列の終端」を意味していて、ここでパターンマッチを終わります。

この2つの記号を並べたパターンがマッチするのは、そう、空の文字列です。といつても、 DjangoのURL名前解決では '<http://127.0.0.1:8000/>' は除いてパターンマッチするので、このパターンは '<http://127.0.0.1:8000/>' 自体を意味します。つまり、「<http://127.0.0.1:8000/>」というURLにアクセスしてきたユーザに対して `views.post_list` を返すように指定していることになります。

いいですか？ <http://127.0.0.1:8000/> を開いて結果を見てみましょう。



どう見ても「うまく動いた」感じはしませんね。でも心配しないで、これはただのエラーページで、怖がるものではありません。むしろ、結構便利なものなんですよ：

'post_list'というアトリビュートがない、と書いてあるのが見えますね。`post_list`と聞いて、何か思い出しませんか？そう、さっきこのURLに、ビューの `post_list` を割り当てたのでした。でも、ビューをまだ作っていないですから、このエラーが出るのは当然ですね。なにもおかしいことはしていません。次の章ではビューを作りましょう。

Django URLconfについて知りたい場合は、公式のドキュメントを見て下さい：

<https://docs.djangoproject.com/ja/1.11/topics/http/urls/>

Django views - time to create!

それでは前の章の続きをやりましょう。確かにビューの作成がまだだったので、エラーになっていましたね。

ビューはアプリのロジックを担当しています。ビューは要求に応じて `model` から情報を取得し `template` に渡します。モデルはついさっき作りましたし、テンプレートもすぐ次の章で作ります。ビューは、Pythonで記述されているだけです。**Introduction to Python** の章でやったことよりも、ちょっと複雑なだけですよ。

ビューは、`views.py` に記述します。私達の場合 `blog/views.py` に書くことになります。

blog/views.py

では、早速 `blog/views.py` を開いてみましょう。

```
from django.shortcuts import render

# Create your views here.
```

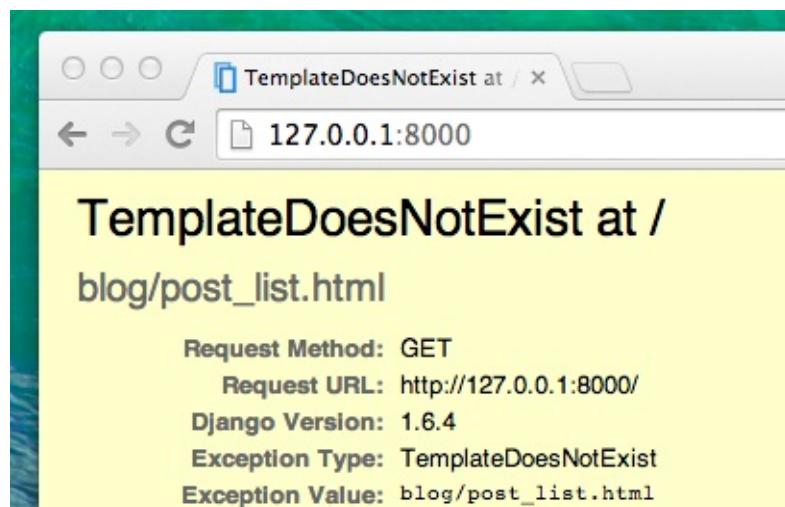
まだ何もないですね。とりあえず、次のような、ちょっとしたビューを作ってみましょう。

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

よく見てみましょうか。まず `post_list` というメソッド(`def` から始まる部分のことです)を、記述しています。この `post_list` は `request` を引数に取り、`render` メソッドを `return` しています。`render` メソッドは `blog/post_list.html` というテンプレートファイルを使って、引数で受け取った `request` の内容を出力しています。

ファイルを保存したら、どんな風に表示されるか、ブラウザで <http://127.0.0.1:8000/> を確認してみましょう。

今度は別のエラーになりましたね。なんと書いてあるでしょうか。



この *TemplatedoesNotExist* エラーの解決は、簡単です。テンプレートファイルがないだけなので、それでは次の章でテンプレートを作ってみましょう！

Djangoのビューについてもっと知りたいのなら、英語で書かれていますが、オフィシャルドキュメントを是非読んでみてください
い : <https://docs.djangoproject.com/ja/1.11/topics/http/views/>

Introduction to HTML

テンプレートって何?とあなたは思っているかもしれませんね。

テンプレートは、あらかじめ用意された共通したフォーマットのなかで異なる情報をみせるためのファイルです。- 例えば、手紙を書く時に、中身は異なるメッセージや宛名や住所を書きますが、共通のフォーマットを使いますよね。それがテンプレートです。

Djangoのテンプレートは、HTMLという言語でできています。(はじめのチャプター **How the Internet works** で、HTMLについては少しお話しましたね。)

What is HTML?

HTMLはシンプルなプログラムで、ChromeやFireFox、SafariといったWebブラウザが解釈して、ユーザーが読めるようにしてディスプレイに表示します。

HTMLは "HyperText Markup Language" の略です。**HyperText**とは、テキストなどをクリックすると別のページに繋がるハイパーアリンクが使えるということです。**Markup**は、目印をつけるという意味です。文章の各部分がどのような役割を果たしているかブラウザに伝えます。 <と> に囲まれたタグによってマークアップされたものは、要素と呼ばれます。

Your first template!

テンプレートを作成するということは、テンプレートファイルを作成するという意味です。すべてはファイルですよね?あなたは、このことにすでに気がついていることと思います。

テンプレートは `blog/templates/blog` ディレクトリに保存します。では、まず `blog` ディレクトリに `templates` という新しいディレクトリを作成しましょう。次に、`templates` ディレクトリの中に、`blog` というディレクトリを作成します。:

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

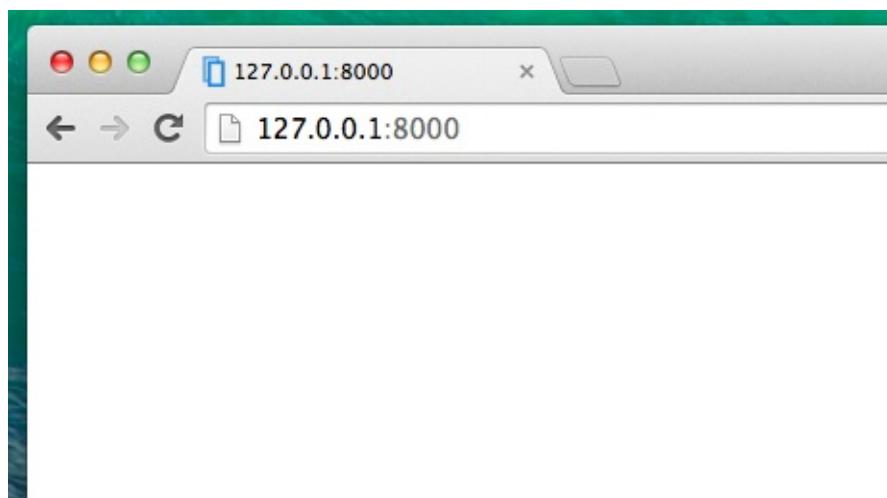
(なぜ `blog` という名前のディレクトリが2つもあるのだろうと、疑問に思ったかもしれませんね。- 後にわかってくると思いますが、これは複雑になりやすい物事を楽にするための、わかりやすい命名規則なのです。)

さて、`blog/templates/blog` ディレクトリの中に、`post_list.html` というファイルを作成しましょう。(今のところは、空っぽのファイルにしておいてください。)

ここで、あなたのWebサイトがどのように見えるか確認してみましょう。:

<http://127.0.0.1:8000>

もし、まだ `TemplateDoesNotExist` というエラーが出るようでしたら、サーバーをリストアートしてみてください。コマンドラインで、`Ctrl+C` (Control と C ボタンを同時に押す)を押すと、サーバーが止まります。`python manage.py runserver` とコマンドを打つて、再度スタートしてください。



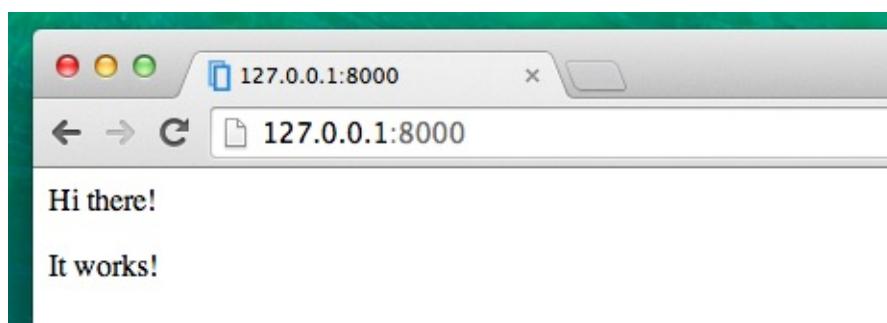
エラーが出なくなりました！おめでとう！！:) しかし、あなたのWebサイトはまだ何も表示されていませんね。あなたのテンプレートにまだ何もないからです。これを修正していきましょう。

テンプレートファイルに、次のとおりにコードを追加して下さい。:

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

さて、あなたのWebサイトは今度はどのように見えるでしょうか？確認してみましょう。:

<http://127.0.0.1:8000>



表示されましたね！よくできました！ :)

- `<html>` 、Webページのはじまりにつけるタグです。そして `</html>` はページの最後につきます。先ほどの例でお分かりのように、最初の行の `<html>` と、最終行の `</html>` の間に、Webサイトの全てのコンテンツが書かれています。
- `<p>` はパラグラフのタグです。 `</p>` を各パラグラフの終わりにつけます。

Head & body

すべてのHTMLページは、2つの要素が必要です。**head** と **body** です。

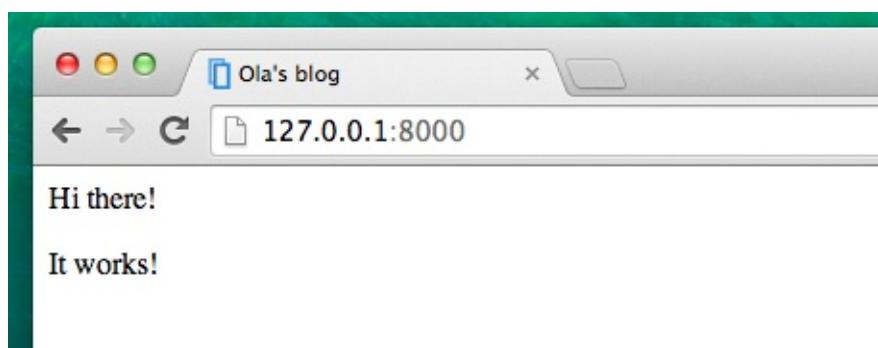
- **head**要素 これには、ドキュメントに必要な情報を含んでいます。ディスプレイには表示されません。
- **body**要素 これには、そのほかのWebページに表示される内容が含まれます。

`<head>` は、ページの構成などをブラウザに伝えます。そして、`<body>` のタグ内の内容が、実際にページに表示されます。

例えば、次の例のように、Webページのタイトルを `<head>` タグの中に含めることができます。:

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

ファイルを保存して、再読み込みしてみましょう。



あなたのページのタイトルが "Ola's blog" だと、ブラウザが認識してくれましたね？ タグ内に書かれた `<title>Ola's blog</title>` をブラウザは解釈して、ブラウザのタイトルバーの文字が反映されました。(ブックマークのタイトルなどにも使われます)。

おそらく既にお気づきでしょう。タグは開始タグと終了タグのペアになっていて、終了タグには / がつきます。そして要素の中にはほかの要素が次々と入れ子になっています(入れ子なので、あるタグを閉じるときには、その中のタグもすべて閉じていなくてはなりません)。

これは箱に入れるようなイメージです。まず最初に、`<html></html>` という大きな箱を1つもっています。その中に、`<body></body>` という箱があり、また更にその中に `<p></p>` という小さい箱が入っています。

タグを閉じることと、入れ子にすることは、守らなければいけないルールです。タグを閉じていなかつたり、入れ子になってしまないと、ブラウザは正しく解釈ができないため、あなたのページは正しく表示されません。

Customize your template

テンプレートをつかって、カスタマイズしてちょっと遊んでみてください。よく使われるタグをいくつかご紹介しておきますね。：

- `<h1>見出し 1</h1>` - 一番大きな見出し
- `<h2>見出し 2</h2>` 次に大きな見出し
- `<h3>見出し 3</h3>` ... 見出しのレベルは1～6まであり、`<h6>` が一番小さい見出しどなります。
- `text` 要素に囲まれたテキストを強調します。文章の意味合いとして強調すべき語句やフレーズを示します。
- `text` 囲まれたテキストを強く強調します。文中の特定の語句が重要であることを示します。
- `
` `br` は Break(改行)の略です。テキストをこの位置で改行します。`(br)` 中には、何も書いてはいけません)
- `link` リンクをつくります。
- `first itemsecond item` リストをつくります。
- `<div></div>` ブロック要素として囲みます。

以下をテンプレートとして使って下さい。：

```
<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 porta gravida at eget metus. Fusce dapibus, tellus ac cursus commodo, tortor mauris condimentum nibh, ut fermentum massa 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 porta gravida at eget metus. Fusce dapibus, tellus ac cursus commodo, tortor mauris condimentum nibh, ut f.</p>
    </div>
  </body>
</html>
```

ここで、3つの `div` ブロックを作りました。

- 最初の `div` 要素は、ブログのタイトルです。見出しとリンクが含まれていますね。
- その他の2つの `div` 要素は、ブログ記事を投稿日時とを囲んでいます。`h2` の見出しへは、リンクタグがついた記事のタイトルです。そして、2つの `p` パラグラフがありますね。投稿日時とブログ記事の内容です。

ブラウザで見るとこのようになります。:



わーい！うまく表示されましたね！でも、まだこのテンプレートは同じ情報を表示するだけですね。- 先ほどお話したようにテンプレートとは、同じフォーマットで、異なる情報を表示するものです。

私たちが本来やりたいことというのは、Djangoの管理画面に追加された記事の情報を、同じフォーマットで表示することです。では、次のレベルにいってみましょう。

One more thing

このWebサイトがローカルだけでなく、同じようにHerokuでも動いているのを見たいですね。再度デプロイしてみましょう。:

```
$ git status
```

次に、作業ディレクトリ内のすべての変更を `git` に追加します。:

```
$ git add -A .
```

補足 `-A` ("all"の意味) とすると、削除されたファイルも `git` は認識します。（デフォルトでは、新規ファイルと変更のあったファイルだけを認識します）そして、チャプター3で説明があったように、`.` は、カレントディレクトリを示していますので、忘れないようにしましょう。

すべてのファイルをアップロードする前に、`git` が何をアップロードしようとしているのか、確認しておきましょう。（`git` がアップロードしようとしている全てのファイルが緑色で表示されます）：

```
$ git status
```

あとちょっとです！がんばりましょう。`git`に変更を保存しておきましょう。どのような変更を加えたかわかるように、"コミットメッセージ"をつけておきます。メッセージはなんでも構いませんが、後でなにをしたかわかるような説明をしておくとよいでしょう。

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

補足 コミットメッセージは、ダブルクオーテーションで囲みましょう。

この次に、やっと変更をHerokuのWebサイトにアップロード（プッシュ）することができます。：

```
git push heroku master
```

アップロードの作業は以上です。Herokuでの処理がおわれば、ブラウザ上で変更を加えられたサイトが見れるでしょう。ブラウザを再読み込みして確認してください。変更されていますね！

DjangoのORMとクエリセット

この章では、Djangoのデータベース接続方法と、データストアについて学びます。やってみましょう！

クエリセットとは？

クエリセットが何かと言うと、モデルが提供しているオブジェクトのリストのことです。クエリセットは、データベースからデータを読み込んだり、抽出したり、言われた通りにやってくれます。

実際に動かしてみるのが一番わかりやすいので、試してみましょう。

Django shell

コンソール画面を開いて、次のコマンドを入力してみましょう。

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

次のような表示に切り替わるでしょう。

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

今、Djangoのインタラクティブコンソールが起動しています。Pythonプロンプトしかないよう見えますが、ちゃんとDjangoも動いています。勿論このコンソール画面では、Pythonのコマンドは何でも使えます。

All objects

最初に、ポストデータを全部表示させてみましょう。次のコマンドで、ポストのデータを全部表示させることができます。

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

ごめんなさい、エラーになってしまいましたね。Postがないというエラーです。その通りなんですね。最初にインポートをしなくてはならないのに、忘れていました。

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

こんな風に書くだけで、blog.modelsからPostモデルをインポート出来ます。それでは、もう一度試してみましょう。

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

さっきDjangoの管理画面から作ったポストのリストが表示されました。だけど、次はこのコンソール画面から、新たにポストを作つてみたいですよね。それはどうすればいいのでしょうか。

Create object

データベースに、新しいPostを作成するには、次のようにします。

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

いい感じなのですが、1つだけマズイことをしているんです。authorにmeを渡していますが、これはUserモデルのインスタンスでないといけませんよね。それは、どうやればいいと思います？、

そうです、さっきと同じです。Userモデルも先にインポートしておきましょう。

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

どんなユーザが、データベースに登録されましたっけ？覗いてみましょうか。

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

作成しておいたスーパーユーザがいますね。このユーザを使ってみましょう。

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

olaというユーザ名のUserモデルのインスタンスを、取り出せたでしょう？よかったです！勿論、他のユーザ名のユーザを取り出しても構いません。

さあ、遂にコンソール画面から、最初のポストを作成出来ますね。

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

どうでしょうか？ちゃんと出来ているか、確認しておきましょうね。

```
>>> Post.objects.all()
[<Post: Sample title>]
```

Add more posts

この先の楽しみの爲にも、もう2~3個、記事を作っておきましょう。そうすれば、もっとよくこの章を理解出来ると思います。

Filter objects

クエリセットの大部分は、フィルタ機能だと言えるでしょう。ユーザ `ola` さんのポストを全部確認してみましょうか。全部のポストを取り出すのではなく、`ola` さんのポストだけを取り出したい場合は、`Post.objects.all()` の `all` を `filter` に変更します。クエリセットの結果として、カッコの中に、`blog` の内容が一覧で表示されました。以下の例だと、`author` が `me` と等しいものだけが抽出されています。

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

もしかすると `title` フィールドに `title` という単語が含まれているポストだけを取り出したくなるかもしれませんね。

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

Note `title` と `contains` の間に、アンダーバーが2個続いていますが、これは Django のORMの構文です。フィールド名の`title`と、照合タイプの`contains`を、2つのアンダーバーで連結させています。もしアンダーバーが1個だけだと、`title_contains` というフィールド名だと判断されてしまい、エラーになります。("FieldError: Cannot resolve keyword `title_contains`")

また、公開済みの全ポストを取り出すことも出来ます。全てのポストの中から、既に公開済みのポストを取り出してみましょう。それには、`published_date` を指定します。

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

そうでした、残念なことに、まだどのポストも公開されていませんね。じゃあ、ポストを公開してみるとしましょう。まず公開するポストを決めましょう。

```
>>> post = Post.objects.get(id=1)
```

そして、`publish` メソッドを呼び出します。

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

じゃあ、もう一度公開済みのポストを取り出しましょう。(上方向キーを3回押せば、さつきのコマンドを呼び出せるでしょう。コマンドを表示出来たら、Enterキーを押してみましょう)

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

Ordering objects

クエリーセットは、オブジェクトのリストの並べ替えもやってくれます。試しに`created_date` フィールドで並べ替えてみましょう。

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

逆順、つまり新しく追加した順に並べ替えることもできます。それには、`-` ハイフンを使います。

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

どうです？次の章への準備は万端ですね！このプロンプトを閉じるには、以下のようにします。

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

ジャンゴ クエリセット

ポスト内容を保存する為の `Post` モデルは、`models.py` に定義しました。ポストの一覧を表示する `post_list` は `views.py` にあり、そこにテンプレートも加わりました。これらを準備しましたが、実際のところ、ポストをどうやってHTMLファイルに出力すればいいのでしょうか？大まかなイメージとしては、データベースに保存された記事を取り出して、テンプレートのHTMLファイルの中に行儀よく並べるだけのことですけど。

正確には、ビューがモデルとテンプレートの橋渡しをしてくれます。私達が作業している `post_list` ビューの場合、表示したいデータを取り出して、テンプレートファイルに渡すことになります。基本的に、どのモデルのデータを、どのテンプレートに表示させるかは、ビューに記述します。

それでは、実際にやってみましょう。

まず `blog/views.py` を開きます。今のところ `post_list` ビューは、以下のようになっているでしょう。

```
from django.shortcuts import render

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

少し前に、別のファイルに用意したコードをどうやってインクルードするか説明したのですけれど、覚えていませんか？それでは `models.py` のモデルを、インクルードしてみましょう。`from .models import Post` という行を追加してみます。

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

`from` の後のドットは `カレントディレクトリ`、もしくは `カレントアプリケーション`のことです。`views.py` と `models.py` は、同じディレクトリに置いてありますから、こんな風にドットとファイル名だけを使って、簡単に記述することが出来ます。ただし、ファイル名に拡張子 `.py` は必要ないですけど。そして、モデルの名前を指定してインポートします(この場合のモデルは `Post` ですね)

さて、次にすべきことは、実際に `Post` モデルからブログの記事を取り出すことですが、それには `クエリセット` が必要です。

クエリセット

もう、クエリセットの働きについては、知っていますよね。Django ORM (QuerySets) chapter で勉強しましたから。

ブログのポストをリストで取得するやり方については、どうでしょう？既に公開済みのもので、`published_date` で並べ替えをしたリストですよ。これも、QuerySetsの章でやりましたから、大丈夫でしょう？

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

そう、このコードですよね。これを `blog/views.py` の `post_list(request)` 関数 の中に加えてみましょう。

```
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', {})
```

作成したクエリセットは、変数 `posts` で参照できることに、注意しましょう。この変数 `posts` を使って、クエリセットのデータにアクセスします。これから先 `posts` というと、このクエリセットのことです。

最後に一点、クエリセットを参照している変数 `posts` をテンプレートに渡すという作業が、出来ていませんが、これは次の章でやりましょう。

`render` 関数では、既にパラメータとして `request` とテンプレートファイル `blog/post_list.html` が渡されています。リクエストというのは、インターネットを介してユーザから受け取った全ての情報が詰まったものです。最後のパラメータに注目してください。`{}` こんな風に書かれていますね。この中に指定した情報を、テンプレートが表示してくれます。`{}` の中に引数を記述する時は、名前と値をセットにしなくてはなりません。表示させたいのはクエリセットのデータなので、`posts` を指定しましょう。こんな風に、記述することになります。`{'posts': posts}` 注意して欲しいのは、シングルクオートです。コロンで区切られた、前の方の `posts` は、シングルクオートで囲まれて、`'posts'` になっていますよね。こちらが名前で、後ろの方の `posts` は値、クエリセットのことです。

最終的に `blog/views.py` は、以下の様になるはずです。

```
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})
```

どうでしたか？次は、このクエリセットをテンプレートで表示させるところを、やってみましょう。

Djangoのクエリセットについて、もっと知りたければ、英語のドキュメントですが、こちらも読んでみてくださいね。 <https://docs.djangoproject.com/ja/1.11/ref/models/querysets/>

Djangoテンプレート

何かデータを表示しましょう！Djangoはそれをビルトインのテンプレートタグで実現できます。

テンプレートタグとは？

HTMLではブラウザがpythonを認識できないのでpythonのコードは書けません。HTMLはより静的でpythonは動的です。

DjangoテンプレートタグはHTMLにPythonのようなコードを埋め込むことができて、動的なウェブサイトがより早く簡単に作れます！

ブログ一覧テンプレート

前の章で、`posts`変数でテンプレートに記事のリストを渡しました。今からHTMLで表示をしてみましょう。

Djangoテンプレートで変数を表示する爲には、変数の名前を二重括弧で括ります：

```
 {{ posts }} 
```

これを `blog/templates/blog/post_list.html` に書いてみて下さい。（2つめと3つ目の `<div>` `</div>` タグをまるごと `{{posts}}` に置き換えて下さい。）ファイルを保存してページをリロードしますと：



見たとおり、このようになります。

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

Djangoはオブジェクトのリストと認識します。**Introduction to Python**を思い出して下さい。ループを使ってリストを表示しましたよね。Djangoテンプレートではこう書きます:

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

これをブログのテンプレートで使ってみましょう。

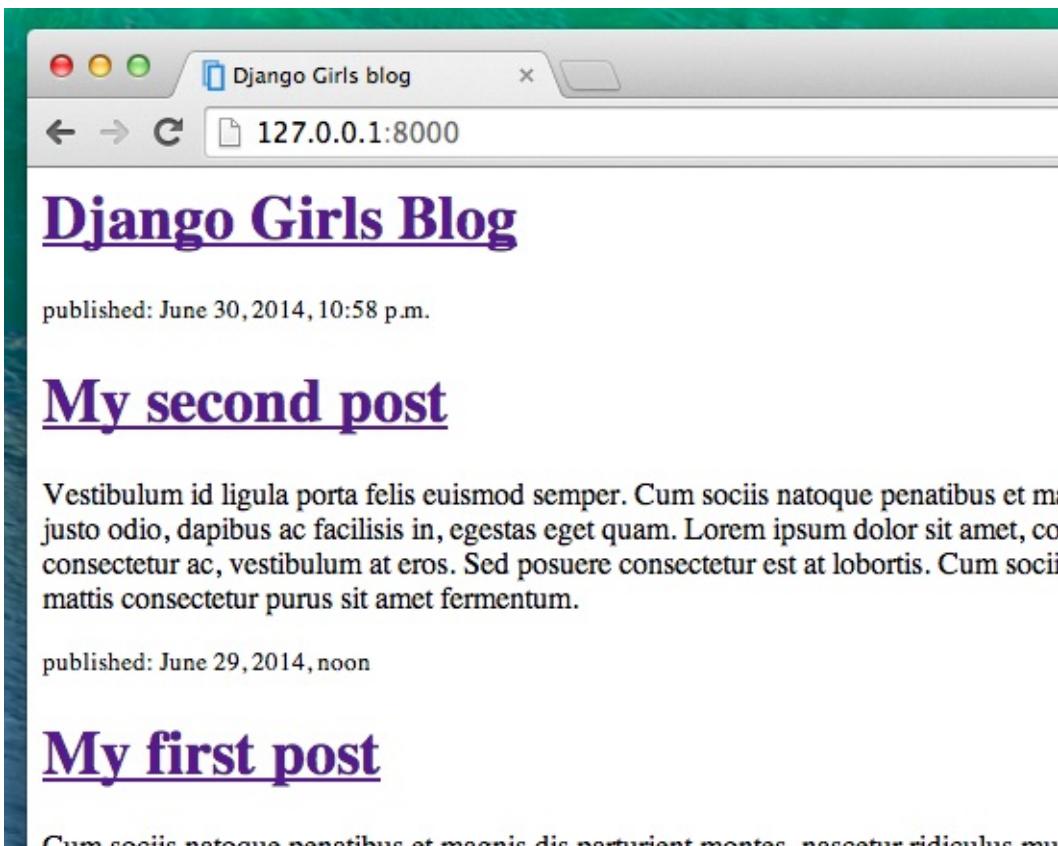


動きましたね。しかし、**Introduction to HTML**で作った静的な記事のような表示です。HTMLとテンプレートタグを混ぜてみましょう。**body**タグの中を次のように書いてください:

```
<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 }}</a></h1>
        <p>{{ post.text|linebreaks }}</p>
    </div>
{% endfor %}
```

{% for %} と {% endfor %} の間にリストの中のオブジェクトごとに表示したい内容を書くとオブジェクトの数だけ繰り返し書かれます。ページをリロードしてみましょう。



post 変数がさつきと違って、`{{ post.title }}` や `{{ post.text }}` になっていることに気づきましたか？ `Post` モデルで定義したそれぞれのフィールドにアクセスしています。`|linebreaks` はPostのテキスト中の改行を段落に変換するフィルタに通すという意味です。

もう一つ…

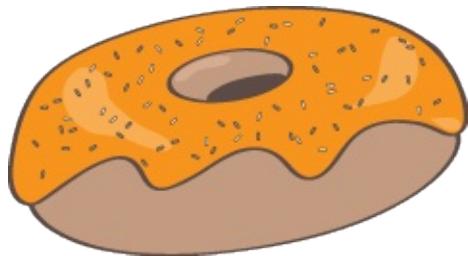
Herokuにデプロイして、インターネットでウェブサイトを公開できます。おさらいしましょう。

```
$ git status  
[...]  
$ git add -A .  
$ git status  
[...]  
$ git commit -m "Added views to create/edit blog post inside the site."  
[...]  
$ git push # heroku master
```

最後にブラウザのタブを開いてアプリをリロードします。更新が反映されています！

おめでとう！これができたら、Django adminとして新しい投稿を追加しましょう。
(`published_date`を忘れないで！) それから、投稿したものがそこに見えるか、リロードしましょう。

動くのが楽しくなってきたでしょう？少しパソコンから離れて、休憩しましょう：）



CSS - make it pretty!

ブログはつくりましたが、このままだとカッコ良くないですよね。カッコよくしていきましょう。ここからはCSSを使っていきます。

What is CSS?

Cascading Style Sheets (CSS)とは、HTMLなどのマークアップ言語で書かれたWebサイトの見た目や書式を記述するための言語です。私達のWebページをマイクアップするものとして使います。

でも、またゼロから作りたくないですよね。プログラマーたちがすでに作って無料で公開しているツールを使いましょう。わざわざイチから作り直す必要はないですからね。

Let's use Bootstrap!

Bootstrapは美しいWebサイトを開発するためのHTMLとCSSのフレームワークとしてとても有名です: <http://getbootstrap.com/>

これは、もともとTwitterのプログラマーが作成したもので、今は、世界中の有志のボランティアで開発されています。

Install Bootstrap

Bootstrapをインストールするには、`.html` ファイル
(`blog/templates/blog/post_list.html`) の `<head>` の中に、次のとおり書き加えます：

```
<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">
```

これは、あなたのプロジェクトにファイルを追加しているわけではありません。インターネット上にあるファイルを指しているだけです。では、Webサイトを開いてページを再読み込みしてください。



これだけで見た目がずいぶん良くなりましたね！

Static files in Django

もう1つ、今日ここで学ぶことは、静的ファイルです。静的ファイルとは、CSSファイルや画像ファイルといった、動的な変更が発生しないファイルのことです。そのため、これらのファイルはリクエストに依存せず、どのユーザに対しても中身は同じになります。

CSSは静的ファイルです。そのためCSSをカスタマイズするためには、最初にDjangoの中で設定を行う必要があります。この設定を行うのは1回だけです。さあはじめましょう：

Configure static files in Django

始めに、静的ファイルを保存するためのディレクトリを作ります。 `djangogirls` ディレクトリの中に、ディレクトリを作成して `static` と名前をつけてください。

```
djangogirls
├── static
└── manage.py
```

`mysite/settings.py` ファイルを開き、下にスクロールして次の行を追加して下さい。

```
STATICFILES_DIRS = (
    os.path.join(BASE_DIR, "static"),
)
```

これで、あなたの静的ファイルの保存場所をDjangoが分かってくれます。

Your first CSS file!

CSSファイルを作つて、Webサイトにあなたのスタイル設定していきましょう。 static ディレクトリの中に、 css という名前の新しいディレクトリを作成してください。その css ディレクトリの中に、 blog.css という名前の新しいファイルを作成してください。準備はいいですか？

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

CSSを書く時間がやつてきました！エディタの中で、 static/css/blog.css ファイルを開いて下さい。

ここでは、CSSのカスタマイズや学習について深くは踏み込みません。それらについて学ぶことは非常に簡単ですので、興味のある方はこのワークショップが終了した後にご自分でチャレンジしてみて下さい！より良いCSSを使ったWebサイト作成について学びたい方には、 [Codecademy HTML & CSS course](#) をオススメします。（※英語サイトです）

少しだけ触つてみましょう。ヘッダーの色を変更することは出来るでしょうか？色を理解するために、コンピュータでは特殊なコードを利用しています。コードは、 # で始まり、 6種類のアルファベット (A-F) や数字 (0-9) が続きます。色コードのサンプルはこのサイトで確認することができます：<http://www.colorpicker.com/>

また、 red や green といった定義済みの色を利用するすることもできます。

static/css/blog.css ファイルに、次のコードを記述しましょう。

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

h1 a はCSSセレクタです。 h1 要素の中にある a 要素（例：このようなコードのこと <h1>link</h1> ）にスタイルを適用します、という意味になります。この場合、テキストの色を #FCA205 、オレンジ色にする、という意味です。もちろん、あなたの好きな色に変更してもいいですよ。

CSSファイルで、HTMLの各要素のスタイルを指定していきます。要素は要素名 (i.e. a , h1 , body) や class 属性、 id 属性で識別されます。 class 名と id 名はあなた自身が指定するものです。 class は要素のグループを定義し、 id は具体的な要素を指します。例えば、以下のタグはCSSによってタグ名が a 、 class 名が external_link 、 id 名が link_to_wiki_page と識別されます：

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

より詳細は情報についてはこちらのページを御覧ください（※英語サイトです）：[CSS Selectors in w3schools.](#)

そして、CSSを追加したことでもHTMLテンプレートに教える必要があります。 `blog/templates/blog/post_list.html` のファイルを開き、先頭に次の行を追加します：

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

私達は丁度ここに静的ファイルを読み込んでいます^^。次に、Bootstrap CSSファイルへのリンクの後ろにある `<head>` と `</head>` の間に次の行を追加します（ブラウザは与えられた順にファイルを読み込むため、私達のコードはBootstrapファイル中のコードで上書きされています）：

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

これで、テンプレートにCSSファイルがある場所を教えたわけです。

あなたのファイルは、このようになっていますか：

```
{% 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|linebreaks }}</p>
            </div>
        {% endfor %}
    </body>
</html>
```

OK、ファイルを保存してサイトを更新してみましょう！

The screenshot shows a web browser window titled "Django Girls blog" with the URL "127.0.0.1:8000". The page content includes the title "Django Girls Blog", a timestamp "published: June 30, 2014, 10:58 p.m.", the post title "My second post", and the post content "Vestibulum id ligula porta felis euismod semper. Cum sociis natoque".

お疲れ様です！おそらく表示されたウェブサイトは非常に読みにくいため、左側にもう少しスペースが欲しくなります。試してみましょう！

```
body {  
    padding-left: 15px;  
}
```

これをあなたのCSSをに追加し、ファイルを保存してどのように動くのか確認してみましょう！

The screenshot shows the same web browser window after applying the CSS rule. The left margin of the main content area has been increased, creating more space on the left side of the page.

見出しのフォントはカスタマイズできるでしょうか？これを `blog/templates/blog/post_list.html` ファイルの `<head>` 中に貼り付けて下さい：

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

この行ではGoogle Fonts (<https://www.google.com/fonts>) から *Lobster* と呼ばれるフォントを読み込んでいます。

`static/css/blog.css` のファイルを開いて、`h1 a` というのブロック中に `font-family: 'Lobster';` という行を追加してみましょう（コードは {} で囲まれています）。そしてページを更新します：

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



素晴らしい！

上に示したように、CSSはHTMLコードの一部に名前をつけて、他に影響を与えずにこの部分にだけスタイルを適用するといった、クラスの概念を持っています。あなたが2つのdiv要素を持っていた場合これは非常に便利でしょう。しかし、見出しと投稿のように、これらの要素は通常全く違う事を行います。そのため、あなたはこれらを区別して扱いたくなるでしょう。

先に進んで、HTMLコードの一部に名前をつけましょう。ヘッダーに含まれる `div` 要素に、`page-header` というクラス名をつけましょう：

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

さらにブログ投稿を含む `div` 要素に `post` というクラス名をつけましょう。

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

そして、別のセレクタに宣言ブロックを追加します。`.` で始まるセレクタはクラスに関連します。Web上にはCSSに関する非常に多くのチュートリアルが存在し、それらは以下に示すコードを理解する手助けになるはずです。今のところは、`djangogirls/static/css/blog.css` のファイルに以下の内容をコピー&ペーストしましょう：

```
.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;  
}
```

そして、これをクラス宣言で投稿を表示しているHTMLコードで囲みます。

blog/templates/blog/post_list.html 中のこの部分を

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

これで置き換えて下さい：

```
<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|linebreaks }}</p>
        </div>
      {% endfor %}
    </div>
  </div>
</div>
```

それらのファイルを保存してWebサイトを更新してみましょう。



やったー！ほら凄いでしょ？この貼り付けたコードを理解するのはそんなに難しいことじやありません。実際に読んでみることで、そのほとんどを理解することができるでしょう。

CSSをいじることを恐れないで下さい！たとえ何かを壊してしまっても、すぐに元に戻すことができます。

美しいWebサイトを作るために必要な全てのことを学ぶために、この無料のオンライン講座を受講することをおすすめします：[Codecademy HTML & CSS course](#)（※英語サイトです）

さて、次の章にいく準備はできましたか？^皿^

テンプレートの拡張

Djangoの便利な機能の一つに、テンプレートの拡張機能があります。ウェブサイトを作る時、どのページでも同じHTMLタグを使うことがありますよね。たとえば、ヘッダーやフッター部分は、どのページでも同じものを使いませんか？そういう時に、便利な機能です。

この機能を使えば、ウェブサイトの共通部分(ヘッダーやフッター、メニューバーなど)に変更を加えたい時、全てのHTMLファイルに対して、同じ作業を繰り返さなくてすみます。一箇所だけ変更すれば、その変更は全てのページに適用されます。

ベースとなる骨組みのテンプレートファイルの作成

ベーステンプレートは、ウェブサイトの全てのページの骨組みともいるべきテンプレートです。

`blog/templates/blog/` に、`base.html` というファイル名で、ベーステンプレートファイルを作ってみましょう。

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

`base.html` ファイルを開いて `post_list.html` の内容を全部 `base.html` にコピーしたら、以下の様になるでしょう。

```
{% 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 }}</a></h1>
                            <p>{{ post.text|linebreaks }}</p>
                        </div>
                    {% endfor %}
                </div>
            </div>
        </body>
    </html>
```

コピーしたら `BODYタグ` 部分を、以下の内容に書き換えましょう。`BODYタグ` は、わかりますよね？ `<body>` から `</body>` までの部分のことですよ。

```
<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>
```

基本的に、`{% for post in posts %}{% endfor %}` の間を、以下の様に書き換えただけですね：

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

どうしてこんな風に書くのか、考えてみましょう。`base.html` に `block` を記述しました。そうすると、別のテンプレートの内容を、この `block` の部分に挿入することが出来るようになります。これが、テンプレートの拡張です。それでは、この `block` に挿入する `post_list.html` も修正してみましょう。

`base.html` を保存して `blog/templates/blog/post_list.html` を開きましょう。まず、BODY タグの中身以外は全て削除してしまいます。`<div class="page-header"></div>` も削除してしまって、最終的には以下の様にします。

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

修正出来たら、次の行をファイルの先頭に書き加えましょう。

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

こんな風にすると `base.html` の `block` 部分に `post_list.html` を組み込んで使うことが出来るという訳です。実際に `block` 部分に組み込むには、以下の様に `{% block content %}` と `{% endblock content %}` で囲わなくてはなりません。

```
{% 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|linebreaks }}</p>
        </div>
    {% endfor %}
{% endblock content %}
```

うまくテンプレート機能が動作しているか、ウェブサイトにを開いて確認してみましょう。

確認するとき `blog/base.html` が見当たらないという、`TemplateDoesNotExist` のエラーが出るかもしれません。その場合は、コンソールで動いている `runserver` を一旦止めてみましょう。停止させるのは `Ctrl-C` (コントロールキーを押しながら'C'キーを押す) で、もう一度起動させるには `python manage.py runserver` コマンドを使います。

Extend your application

もう、ウェブサイトを作るのに必要な全ての章は終わりました。どのようにモデル、URL、ビュー、テンプレートを書いたら良いかわかっていますし、またウェブサイトの作り方もわかります。

さあ練習しましょう！

ブログに最初に必要なものはおそらく、記事を表示するページですよね。

もう `Post` モデルが入っていますから、`models.py` は追加する必要はありません。

Create a link in the template

`blog/templates/blog/post_list.html` ファイルに次のようにリンクを追加しましょう：

```
% 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|linebreaks }}</p>
        </div>
    {% endfor %}
{% endblock content %}
```

投稿のタイトルから詳細記事へリンクさせたい時は、`<h1>{{ post.title }}</h1>` の箇所をリンク先に変えてみましょう。

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

`{% url 'post_detail' pk=post.pk %}` という不思議なものについて説明しましょう。うすうすわかっているかもしれません、`{% %}` という表記はDjangoのテンプレートタグを使っているということを意味します。今回書いたこれは、URLに変換されます。

`post_detail` は `post_detail` ビューへのパスです。注：blogはアプリケーションの名前です。ビューは `views.py` ファイルの名前で `post_detail` はビューの名前です。

<http://127.0.0.1:8000/> にアクセスすると、エラーとなります。これは想定通り。URL や post_detail のビューを作っていないからです。このようになります：



なので、urls.py に post_detail ビューを作りましょう。

URL: <http://127.0.0.1:8000/post/1/>

Django に post_detail が呼び出すビューを作りましょう。このビューでブログの投稿を表示します。 blog/urls.py ファイルに url(r'^post/(?P<pk>[0-9]+)/\$', views.post_detail), を追加して下さい：

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

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

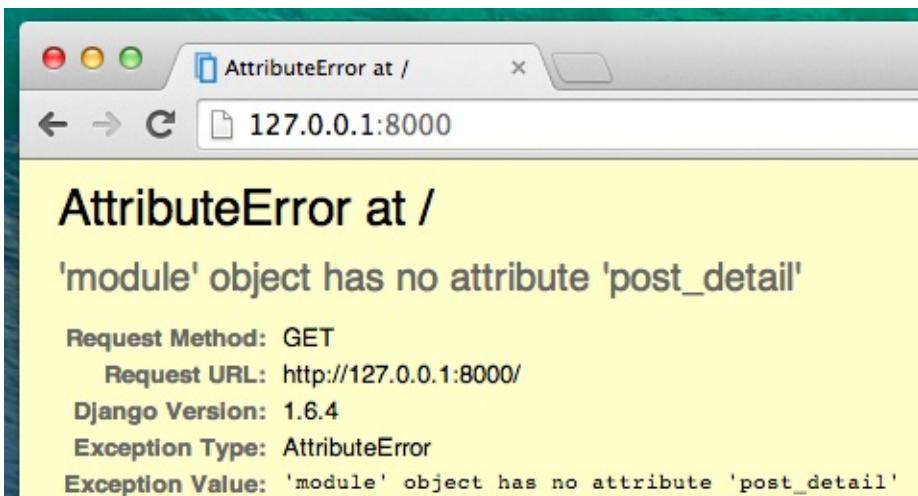
何だか変に見えますが、心配しないで下さい、説明しますと、

- ^ で始まるのは「文字列の開始」です。
- post/ は URL に **post** と / を含む
- (?P<pk>[0-9]+) これはトリッキーな部分です。Django はここに書いた全てを受けてそれを **pk** という変数としてビューへ渡します。 [0-9] は (0から9の間の) 数字のみを意味し、 + はそれが一桁以上続くことを意味しています。 <http://127.0.0.1:8000/post//> はダメで <http://127.0.0.1:8000/post/1234567890/> は OK です。
- / - もう一度 / が必要です。
- \$ は「文字列の終了」を意味します。

ブラウザで `http://127.0.0.1:8000/post/5/` を表示した時、Djangoはビューが `post_detail` を呼び出すということを理解します。そして、`pk` の部分は5とビューへ渡します。

`pk` とはprimary keyの省略です。この名前はDjangoプロジェクトでよく使われますが、好きに名前を付けることもできます（でも、英小文字と `_` だけで、空白などは入れないよう）。`(?P<pk>[0-9]+)` の代わりに、`post_id` と名付けるとすれば `(?P<post_id>[0-9]+)` となります。

OK、ページ <http://127.0.0.1:8000> をリロードしてみましょう。けどまた違うエラーが出てきました！



次のステップを思い出しました？ そうです、ビューの追加です。

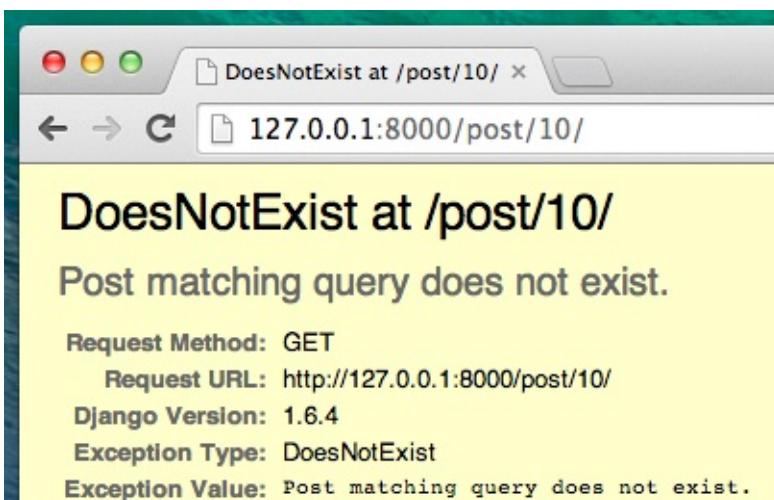
post_detail view

ここでは、ビューが特別なパラメータ `pk` に渡され、ビューはそれを受け取る必要があります。URLで決めた名前 (`pk`) を正確に使う必要があります。`def post_detail(request, pk):` と関数を定義しましょう。この変数を省いてしまうとエラーになります。

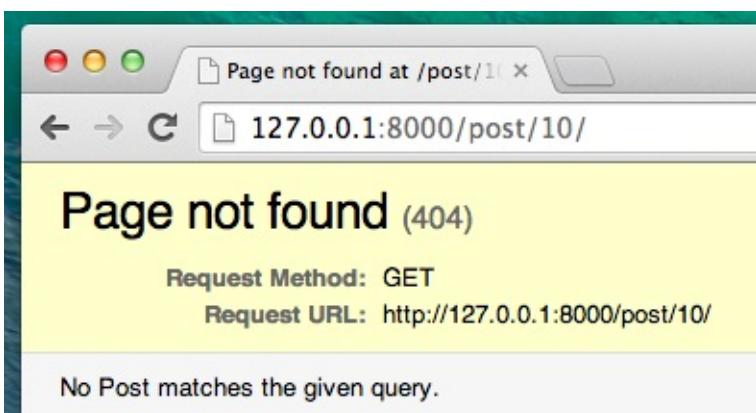
`pk` で指定されたブログの記事を取得するにはクエリセットを使います：

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

しかしこのコードには問題があります。指定された `pk` を持つブログ記事がないと、なんだかとてもダサいエラーが出てきます。



表示してほしいのはこれじゃないです。でももちろん、Django にはこれをうまく処理してくれるものがあります。それが `get_object_or_404` です。`pk` のブログ記事がないときは「404、ページがありません」とさっきよりずいぶんマシな表示をしてくれるのです。



実際に「ページがありません」というページを作れますし、したいようにできますが、重要でないので、飛ばしましょう。

OK, ビューを `views.py` ファイルに追加しましょう。

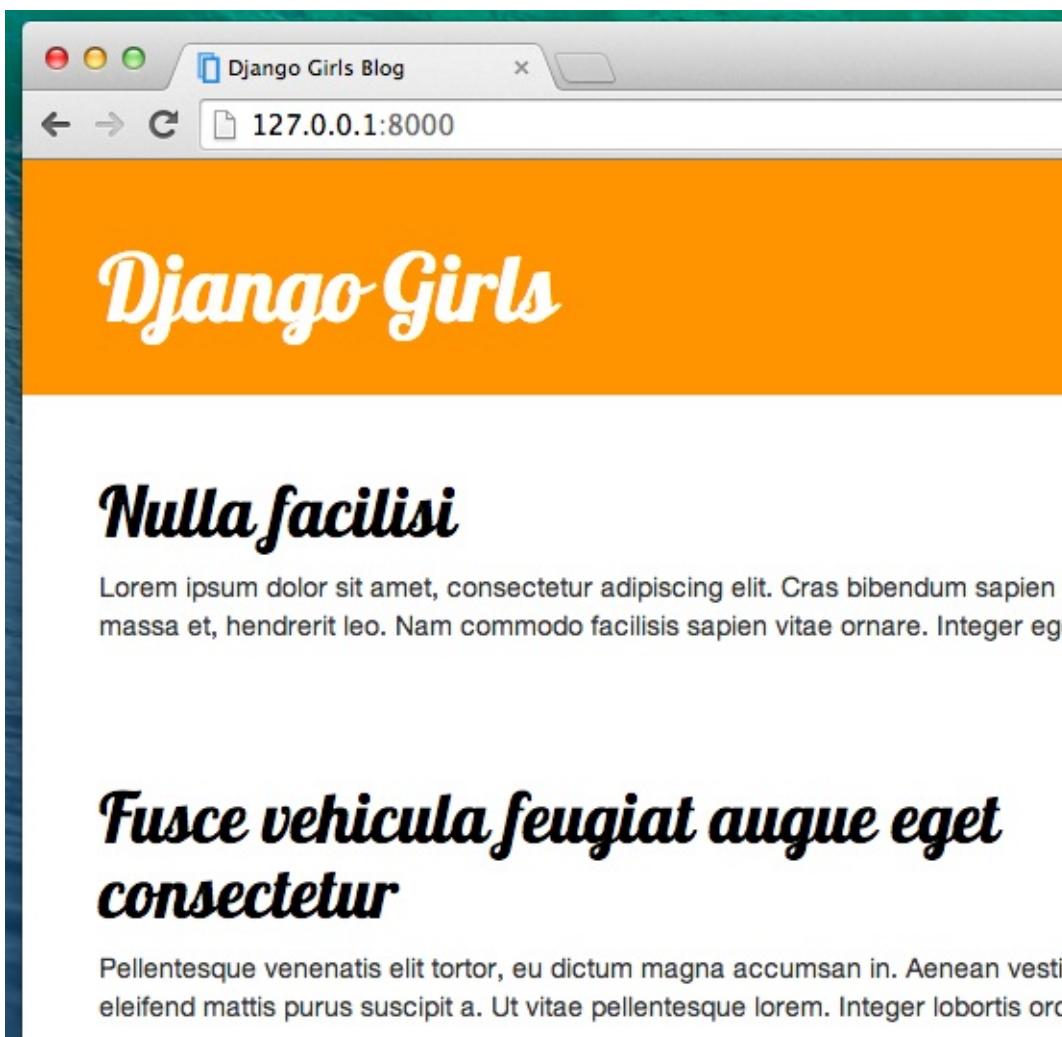
`blog/views.py` ファイルを開いて、今ある `from` で始まる行の次の行に、このコードを追加しましょう：

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

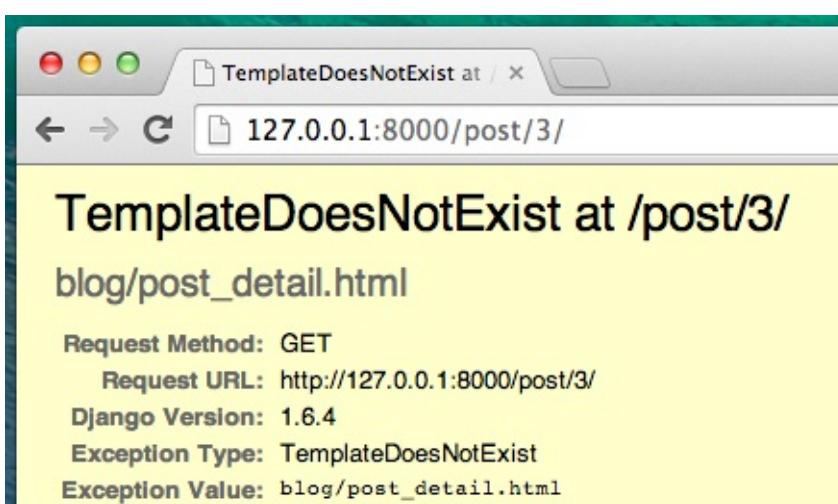
そして、ファイルの最後にビューを追加します。

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

それでは、もう一度 <http://127.0.0.1:8000/> をリロードしてみましょう。



動きました！しかしブログ記事のタイトルのリンクをクリックしたらどうなりますか？



そんな！別のエラーです。しかしこの対処方法を知っていますよね。テンプレートを追加しましょう。

blog/templates/blog に `post_detail.html` を作りましょう。

このように書きましょう：

```

{% 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|linebreaks }}</p>
</div>
{% endblock %}

```

前と同じで `base.html` を拡張します。`content` のブロックで、もしあれば記事の投稿日(`publish_date`)、タイトル(`title`)と本文(`text`)を表示します。ここで重要なポイントについて見てみます。

`{% if ... %} ... {% endif %}` は何かをチェックするときに使うテンプレートタグです。
(**Introduction to Python** の章にあるif~else文を思い出して下さい) この場合
は `published_date` が空欄かどうかを確認しています。

OK, ページをリロードして、もう「ページがありません」とは言われなくなっているか見て
みましょう。



やった！動きました！

One more thing: deploy time!

もしウェブサイトがHerokuで動いているなら、確認するのは良いことです。もう一度deployしてみましょう。どのようにやるか忘れていたら、[Deploy](#)の章を確認しましょう。

```
$ git status  
...  
$ git add -A .  
$ git status  
...  
$ git commit -m "Added more views to the website."  
...  
$ git push heroku master
```

できました！おめでとう：)

Django フォーム

最後に、ブログ記事の追加と編集を、ウェブサイト上でできるようにしましょう。Django の管理画面はクールですが、カスタマイズして綺麗に作るのはとても難しいです。`フォーム` はインターフェースを介してパワフルな機能を提供し、私たちが想像できるほとんどの操作を行うことができます。

Djangoのフォームの良いところは、ゼロからスクラッチもできるし、モデルにフォームの結果を保存するための `ModelForm` を作成することもできます。

それを利用して、これから `Post` モデル用のフォームを作成しましょう。

Djangoの重要な他のパートと同じように、フォームのためのファイル `forms.py` を作成します。

`forms.py` というファイルを `blog` ディレクトリの配下に作成します。

```
blog
└── forms.py
```

いいですね、では `blog/forms.py` を開いて次のコードを入力します：

```
from django import forms

from .models import Post

class PostForm(forms.ModelForm):

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

まずDjangoのフォームモジュールをインポートします(`from django import forms`)。そして私たちが定義した`models.py`から `Post` モデルをインポートします(`from .models import Post`)。

予想しているでしようが、このフォームの名前は `PostForm` にします。そしてこのフォームを `ModelForm` として使用できるように `forms.ModelForm` を継承します(Djangoの機能をうまく使うためのおまじないと思ってください)。

次に、インナークラスとして `class Meta` を定義し、どのモデルからこのフォームを作成するべきかを指定します(`model = Post`)。

最後に、どのフィールドをこのフォームで使用するかを指定します。ここでは `title` と `text` のみ指定します。`author` は現在ログインしている人（あなたのことです）、`created_date` は投稿日が自動で入るようにしましょう。

ひとまずこつちはこれでおしまいです！次は `view` とテンプレートでフォームを表示するようにします。

そのためもう一度、ページへのリンクとURLとビューとテンプレートを作成します。

フォームを含むページへのリンク

`blog/templates/blog/base.html` を開き、`div class="page-header"` タグ内にリンクを追加します：

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

新しいビュー `post_new` を呼び出していることに注意してください。

先ほどのリンクを追加した後のテンプレートファイルは次のようになります：

```
{% 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>
        </div>
    </body>
</html>
```

これを保存して `http://127.0.0.1:8000` を更新してください。するとお馴染みの `NoReverseMatch` エラーが発生するはずです。発生しましたか？

URL

それではURLを追加します。`blog/urls.py` を開き次の行を追加します。

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

追加した後の `blog/urls.py` は次のようにになります。

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

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

サイトを更新すると `AttributeError` が発生します。これは `post_view` を実装していないためです。それでは `post_view` を追加しましょう。

post_new ビュー

`post_view` を実装していきます。`blog/views.py` に次の行を追加します。まずは `from` の行から:

```
from .forms import PostForm
```

`view` はこのようになります:

```
def post_new(request):
    form = PostForm()
    return render(request, 'blog/post_edit.html', {'form': form})
```

新しい `Post` フォームを作成するために、`PostForm()` で `PostForm` をインスタンス化してテンプレートに渡します。また後でこの `view` の実装を変更しますが、そのまえにテンプレートを作りましょう。

テンプレート

`blog/templates/blog` ディレクトリ配下に `post_edit.html` を作成します。フォームを動作させるためにはいくつかやることがあります。

- フォームを表示する必要があります。簡単な例としては `` と記述することです。
- 上記の行は `form` タグでラップする必要があります: `<form method="POST">...</form>`
- `Save` ボタンが必要です。HTMLではボタンは次のように書きます: `<button type="submit">Save</button>`
- 最後に `form` タグ内(開 `form` タグ直後)に `{% csrf_token %}` を追加します。これによりこのフォームがセキュアになるので、とても大切です。もしこれを記述せずにフォーム

を保存しようとするとDjangoは文句を言います。

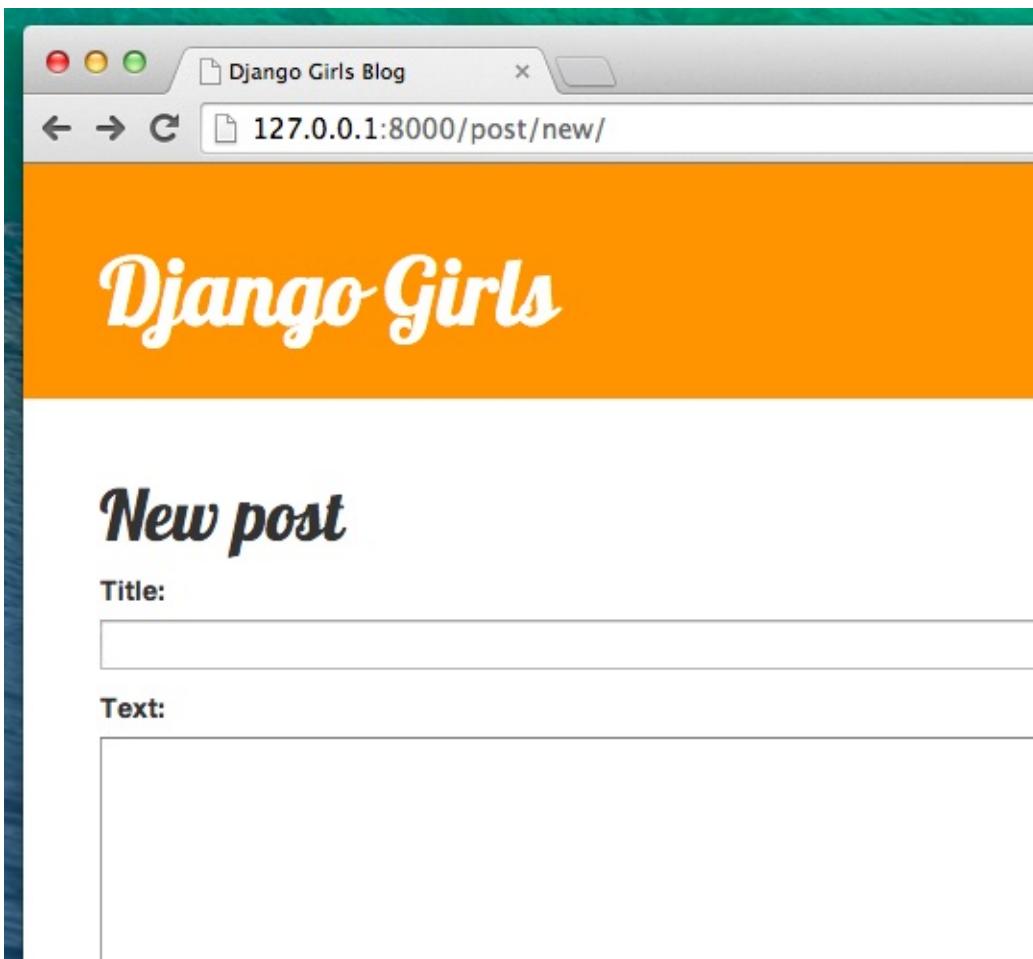


それでは `post_edit.html` 内のHTMLがどのように見えるべきかを見てみましょう:

```
{% 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 %}
```

更新します。わーい! フォームが表示されました!



しかし、ちょっと待ってください! `title` と `text` フィールドを入力して保存しようとすると何が起こるのでしょうか?

何も起こりません! 同じページが再度表示され、私たちのテキストがなくなってしま...そして新しい投稿が追加されていません。何が間違っているのでしょうか?

その答えは...。間違えてはいませんが `view` に対してもう少し作業を行う必要があります。

form を save する

`blog/views.py` を再度開きます。すでに `post_new` ビューがあります:

```
def post_new(request):
    form = PostForm()
    return render(request, 'blog/post_edit.html', {'form': form})
```

フォームを `submit` するとき、今は同じビューに戻りますが、このとき `request` 内(より具体的には `request.POST` 内)にフォームのデータが保持されています。HTMLファイルで定義された `<form>` タグに `method="POST"` が指定されていたことを覚えていましたか? フォームの

すべてのフィールドが `request.POST` 内にあります。`POST` の名前を他の名前に変更することはできません（それを変更する唯一の方法は `method` に `GET` を指定することですが、それがなぜ間違いであるかを話す時間がありません）

そのため `view` では2つの状況をハンドルするようにします。1つ目は初回アクセス時で空のフォームが欲しい時です。2つ目はフォームの入力を終えて全てのフォームのデータとともに `view` に戻る時です。そこで条件を追加します（そのために `if` を使用します）。

```
if request.method == "POST":  
    [...]  
else:  
    form = PostForm()
```

それでは [...] の部分を埋めていきます。`method` が `POST` の場合、フォームから送られたデータを用いて `PostForm` を作成するために次のようにします：

```
form = PostForm(request.POST)
```

簡単ですね！次にフォームの値が正しいかどうかをチェックします（すべての必須フィールドが設定され、全く誤った値が保存されていないことを）。`form.is_valid()` を使うことでチェックできます。

フォームをチェックして、フォームの値が有効であれば保存できます。

```
if form.is_valid():  
    post = form.save(commit=False)  
    post.author = request.user  
    post.save()
```

基本的にここでは2つのことを行います。まず `form.save` でフォームを保存することと `author` を追加することです（まだ `PostForm` 内に `author` フィールドがありませんし、このフィールドは必須です）。`commit=False` は `Post` モデルをまだセーブしません。では `author` を追加します。`commit=False` を指定せず `form.save()` を実行します。そしてこのケースではそれが必要です。`post.save()` は変更(`author`の追加)を保存し、新しいブログ記事を作成します。

最後に、新しく作成された記事の `post_detail` ページを表示できれば良いですね？そのために次のインポートを追加します：

```
from django.shortcuts import redirect
```

それをファイルの先頭に追加します。これでようやく、新しく作成されたポストのための `post_detail` ページに移動する処理を書けます。

```
return redirect('post_detail', pk=post.pk)
```

`post_detail` は新しく作成されたポストのために `post_detail` ページに移動するためのビューです。この `view` では `pk` 変数が必須であることを覚えていませんか? `post` では新しいブログ記事が作成されます。

OK, たくさんのことと説明しました。全体の `view` は以下のようになります。

```
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.save()
            return redirect('post_detail', pk=post.pk)
    else:
        form = PostForm()
    return render(request, 'blog/post_edit.html', {'form': form})
```

では動作確認してみます。<http://127.0.0.1:8000/post/new/> に行き、`title` と `text` を追加し、保存します...。できあがり! 新しいブログ記事が追加され、`post_detail` にリダイレクトされます!

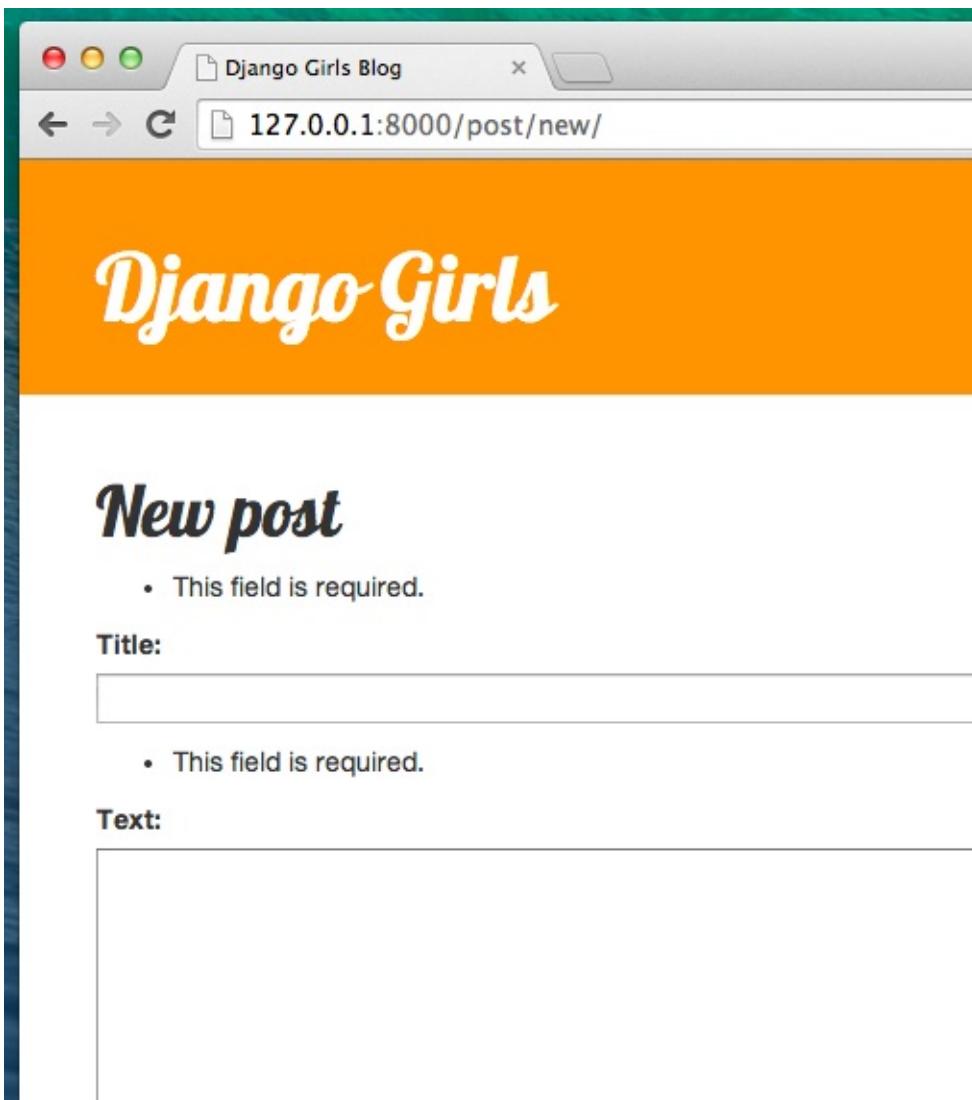
おそらくあなたは日付が設定されていないことに気づいたことでしょう。それについては **Django Girls Tutorial: Extensions** 内の `publish button` をみてください。

素晴らしい!

フォームのバリデーション(検証)

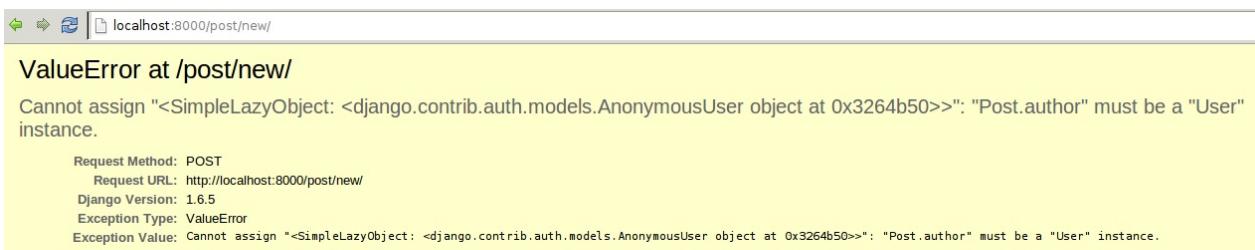
ここではDjangoのフォームのクールなところを紹介します。ブログのポストは `title` と `text` のフィールドが必要です。`Post` モデルでは、これらのフィールドがなくてもよいとは書いておらず(デフォルトの値が設定されている `published_date` とは対照的に)、 Djangoではその場合、それらのフィールドには何らかの値が設定されないとエラーが起こるようになっています。

`title` と `text` を入力せずに保存してみましょう。何が起こるでしょうか?



Djangoはフォームのすべてのフィールドが正しいことを検証してくれます。気が利くでしょう？

ここでは現在、Djangoの管理画面と同様に、ログイン状態で操作しています。いくつかの状況ではログアウト状態になることがあります(ブラウザを閉じる、DBを再起動するなど..)。ポストの作成時にログインユーザがわからないことでエラーが発生した場合、管理画面に移動し再度ログインすることで、その問題は一時的に解決します。メインチュートリアルの後 **Homework: add security to your website!** の章に恒久的な対策がありますので宿題として取り組んでみてください。



フォームの編集

今、私たちは新しいフォームを追加する方法を知っています。しかし既存のデータを編集するためはどうすれば良いのでしょうか?それは先ほど行ったことと非常に似ています。すぐにいくつかの重要なものを作成してみましょう。(もしわからぬ場合、コーチに尋ねるか、もしくはすでに手順をカバーしているので、前の章を見てください)

`blog/templates/blog/post_detail.html` を開いて次の行を追加します:

```
<a class="btn btn-default" href="{% url 'post_edit' pk=post.pk %}"><span class="glyphicon glyphicon-pencil"></span></a>
```

テンプレートは次のようになります:

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

{% block content %}
    <div class="date">
        {% if post.published_date %}
            {{ post.published_date }}
        {% endif %}
        <a class="btn btn-default" href="{% url 'post_edit' pk=post.pk %}"><span class="glyphicon glyphicon-pencil"></span></a>
    </div>
    <h1>{{ post.title }}</h1>
    <p>{{ post.text|linebreaks }}</p>
{% endblock %}
```

`blog/urls.py` には次の行を追加します:

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

テンプレート `blog/templates/blog/post_edit.html` を再利用します。そしてviewを追加します。

`blog/views.py` を開いて次をファイルの最後に追加します:

```

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.save()
            return redirect('post_detail', pk=post.pk)
    else:
        form = PostForm(instance=post)
    return render(request, 'blog/post_edit.html', {'form': form})

```

`post_view` とほとんど同じに見えますか？しかし完全に同じではありません。まずURLから `pk` パラメータを渡します。次に `Post` モデルを `get_object_or_404(Post, pk=pk)` で取得します。その後フォームを保存する際、この記事をインスタンスとしてフォームを作成します。

```
form = PostForm(request.POST, instance=post)
```

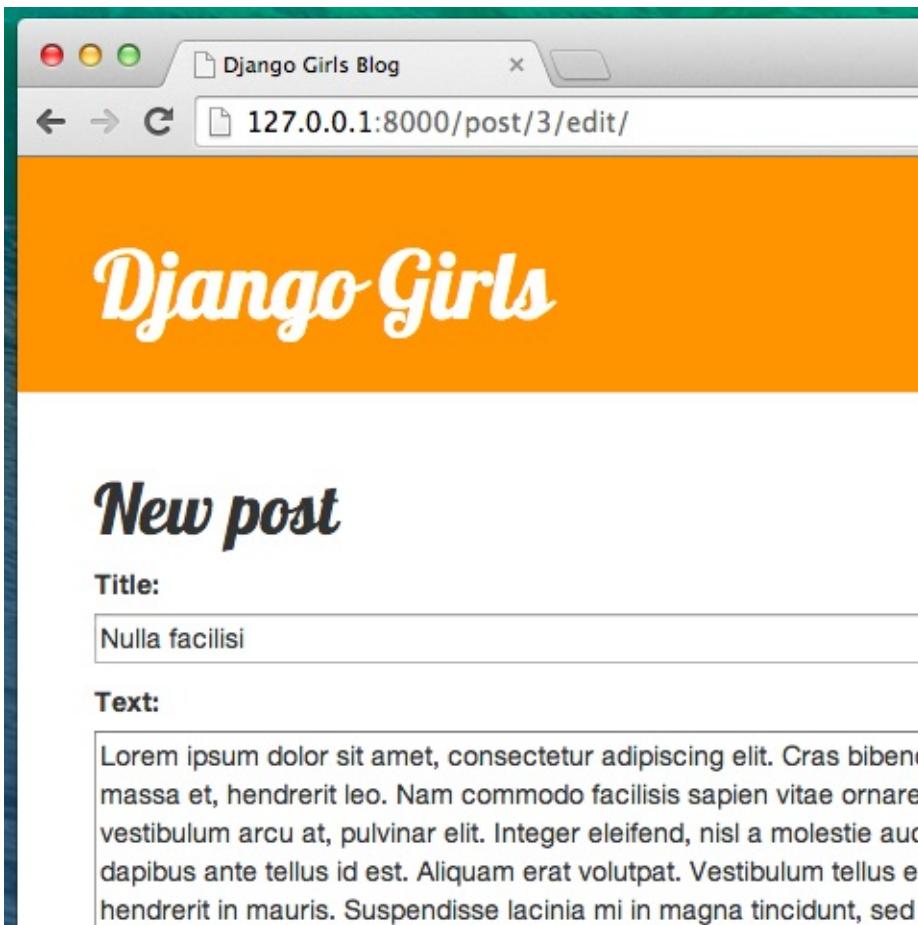
そしてこの記事でフォームを開き編集します。

```
form = PostForm(instance=post)
```

OK, 動作確認しましょう。`post_detail` ページにいきます。そこの右上に [編集] ボタンがあるはずです：



クリックするとブログの記事にフォームが表示されます:



あとはお気軽にタイトルやテキストを変更して保存してください。

おめでとう！アプリケーションが完成しました。

Djangoのフォームについての詳細を知りたい場合、Django Projectのドキュメントを読んでください: <https://docs.djangoproject.com/ja/1.11/topics/forms/>

もう一つ: deployの時間です！

ではHeroku上で動作するかを確認しましょう。再度デプロイします。なおデプロイ方法を忘れてしまった場合は章の最後 Deploy をチェックしてください:

```
$ git status  
...  
$ git add -A .  
$ git status  
...  
$ git commit -m "Added views to create/edit blog post inside the site."  
...  
$ git push heroku master
```

フォームを作ろう

そしてdeployします! おめでとうございます:)

Domain

Heroku gave you a domain, but it's long, hard to remember, and ugly. It'd be awesome to have a domain name that is short and easy to remember, right?

In this chapter we will teach you how to buy a domain and direct it to Heroku!

Where to register a domain?

A typical domain costs around \$15 a year. There are cheaper and more expensive options, depending on the provider. There are a lot of companies that you can buy a domain from: a simple [google search](#) will give hundreds of options.

Our favourite one is [I want my name](#). They advertise as "painless domain management" and it really is painless.

How to register domain in IWantMyName?

Go to [iwanntmyname](#) and type a domain you want to have in the search box.

From here to your domain in 180 seconds:



You should now see a list of all available domains with the term you put in the search box. As you can see, a smiley face indicates that the domain is available for you to buy, and a sad face that it is already taken.

| | | |
|--|--------------------|---------------------|
| | djangogirls.com | |
| | djangogirls.co | \$29.99 now \$15.50 |
| | djangogirls.net | only \$14.90 |
| | djangogirls.org | |
| | djangogirls.me | \$24.99 now \$19.90 |
| | djangogirls.club | only \$19.90 |
| | djangogirls.expert | only \$59.00 |
| | djangogirls.link | only \$15.50 |

We've decided to buy `djangogirls.in` :

IN YOUR CART:

djangogirls.in
\$24.90 USD/YEAR

\$24.90 USD **Checkout**

Go to checkout. You should now sign up for `iwantmyname` if you don't have an account yet. After that, provide your credit card info and buy a domain!

After that, click `Domains` in the menu and choose your newly purchased domain. Then locate and click on the `manage DNS records` link:

| | | |
|--------------------|--|--|
| Nameservers | ns1.iwantmyname.net ns2.iwantmyname.net ns3.iwantmyname.net ns4.iwantmyname.net | update nameservers manage DNS records |
|--------------------|--|--|

Now you need to locate this form:

| Hostname ? | Type ? | Value ? | TTL ? |
|-------------------------------|--------|------------------------------|------------|
| e.g. www, blog or leave empty | A | e.g. 72.32.231.8, web.me.com | 3600 |
| | | | add |

And fill it in with the following details:

- Hostname: www
- Type: CNAME
- Value: your domain from Heroku (for example djangogirls.herokuapp.com)
- TTL: 3600

| Hostname ? | Type ? | Value ? | TTL ? | |
|------------|--------|---------------------------|-------|------------------------------------|
| www | CNAME | djangogirls.herokuapp.com | 3600 | <input type="button" value="add"/> |

Click the Add button and Save changes at the bottom.

It can take up to a couple of hours for your domain to start working, so be patient!

Configure domain in Heroku

You also need to tell Heroku that you want to use your custom domain.

Go to the [Heroku Dashboard](#), login to your Heroku account and choose your app. Then go into app Settings and add your domain in the `Domains` section and save your changes.

That's it!

What's next?

おめでとうございます。素晴らしいです。よくやりましたね。

What to do now?

まずは少し休んでください。あなたはやり遂げました。

休んだ後は、こちらも確認してください：

- Django Girls の [Facebook](#) と [Twitter](#) です。

Can you recommend any further resources?

このチュートリアルの拡張版である、[Django Girls Tutorial: Extensions](#)を試してみてください。

他にオススメの資料のリストを下記に記述します。とてもいいですよ。

- [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 book](#)