

Django Girls Tutorial

ジャンゴガールズチュートリアル

Django Girls Tokyo 著

2018-10-08 版 発行

目次

第 1 章	Django Girls のチュートリアル	1
1.1	ウェルカム！	1
1.2	はじめに	1
1.3	チュートリアルでは何を知ることができる？	2
1.4	自分の家でチュートリアルを進める	3
1.5	コントリビューションについて	4
1.6	チュートリアルの翻訳をしたい方へ	4
第 2 章	Chromebook のセットアップ	5
Cloud 9		5
仮想環境		6
GitHub		7
PythonAnywhere		7
第 3 章	インターネットのしくみ	9
第 4 章	コマンドラインを使ってみよう	15
4.1	コマンドラインって何？	15
4.2	コマンドラインインタフェースを開く	16
4.3	プロンプト	17
4.4	最初のコマンド（イエイ！）	18
4.5	基本	20
	カレントディレクトリ（現在のディレクトリ）	20
	ファイルとディレクトリの一覧	21
	カレントディレクトリの変更	22
	ディレクトリの作成	24
	エクササイズ！	26

目次

クリーンアップ	27
終了	31
4.6 まとめ	32
4.7 準備 OK?	33
第 5 章 Python をはじめよう	35
第 6 章 Python のインストール	37
第 7 章 コードエディタ	45
7.1 Gedit	45
7.2 Sublime Text 3	46
7.3 Atom	46
7.4 なんでコードエディタをインストールするの?	46
第 8 章 Python 入門	47
8.1 Python プロンプト	47
8.2 最初の Python コマンド!	48
8.3 文字列	49
8.4 まとめ	51
8.4 エラー	51
8.5 変数	52
8.6 print 関数	54
8.7 リスト	55
8.8 辞書（ディクショナリ）	57
8.9 まとめ	60
8.9 比較	61
8.10 ブール型（Boolean）	63
第 9 章 保存しよう！	65
9.1 If … elif … else	69
条件が True じゃないときは?	70
9.2 コメント	72
9.2 まとめ	73
9.3 自作の関数！	73
9.4 ループ	77
9.5 まとめ	79

目次

第 10 章	Django ってなに？	81
10.1	なぜフレームワークを必要とするか。	81
10.2	誰かがあなたの WEB サイトにリクエストを要求したときどうなりますか。	82
第 11 章	Django のインストール	83
11.1	仮想環境	83
11.2	仮想環境の操作	87
11.3	Django のインストール	89
	Requirements ファイルによってパッケージをインストールする	89
第 12 章	プロジェクトを作成しよう！	93
12.1	設定変更	96
12.2	データベースをセットアップする	98
12.3	ウェブサーバを起動する	99
第 13 章	Django モデル	103
13.1	オブジェクト	103
13.2	Django モデル	105
	新しいアプリケーションの作成	105
	ブログポストモデルの作成	107
	データベースにモデルのためのテーブルを作成する	109
第 14 章	Django admin	111
第 15 章	デプロイ！	115
第 16 章	Git	117
16.1	Git のインストール	117
16.2	Git リポジトリを始める	120
16.3	GitHub にコードをプッシュする	122
第 17 章	PythonAnywhere でブログを設定する	127
17.1	PythonAnywhere アカウントにサインアップする	127
17.2	PythonAnywhere API トークンの作成	128
17.3	PythonAnywhere でサイトを設定する	129
17.4	動いています！	132

目次

17.5	デバッグのヒント	133
第 18 章	あなたのサイトをチェック！	135
第 19 章	Django URL	137
19.1	URL とは？	137
19.2	Django で URL はどのように機能する？	138
19.3	あなたの初めて Django URL!	139
19.4	blog の URL	139
第 20 章	Django ビュー - 今こそ作りましょう！	143
20.1	blog/views.py	143
第 21 章	HTML 入門	147
21.1	HTML とは？	147
21.2	最初のテンプレート	147
21.3	Head と Body	150
21.4	テンプレートのカスタマイズ	152
21.5	もう一つ: デプロイしましょう！	154
	Github に自分のコードを Push してみよう	154
	新しいコードを PythonAnywhere に pull して、自分のウェブ	
	アプリを再度実行させる	156
第 22 章	Django の ORM とクエリセット	157
22.1	クエリセットとは？	157
22.2	Django shell	157
	すべてのオブジェクト	158
	オブジェクトの作成	159
	さらに投稿を追加しましょう	160
	オブジェクトの抽出	160
	オブジェクトの並び替え	162
	クエリセットをつなげる	163
第 23 章	テンプレート内の動的データ	165
23.1	クエリセット	166
第 24 章	Django テンプレート	169

目次

24.1	テンプレートタグとは？	169
24.2	ブログ一覧テンプレートの表示	169
24.3	もう一つ…	172
第 25 章	CSS でカワイくしよう！	175
25.1	CSS とは？	175
25.2	Bootstrap を使いましょう！	175
25.3	Bootstrap のインストール	175
25.4	Django の静的ファイル	176
	静的ファイルはプロジェクトのどこに置けばいいの？	177
25.5	最初の CSS ファイル！	177
第 26 章	テンプレートを拡張しよう	187
26.1	基本テンプレートを作成する	187
第 27 章	アプリケーションを拡張しよう	193
27.1	投稿の詳細へのテンプレートリンクを作成する	193
27.2	投稿の詳細への URL を作成する	195
27.3	投稿の詳細ビューを追加する	196
27.4	投稿の詳細へのテンプレートリンクを作成する	200
第 28 章	デプロイの時間です！	203
28.1	サーバー上の静的ファイルの更新	204
第 29 章	Django フォーム	205
29.1	フォームにおけるページへのリンク	206
29.2	URL	207
29.3	post_new ビュー	208
29.4	テンプレート	209
29.5	フォームを保存する	211
29.6	フォームのバリデーション（検証）	215
29.7	フォームの編集	216
29.8	セキュリティ	220
29.9	もう一つ：デプロイの時間です！	222
第 30 章	次のステップは？	225
	今何をする？	225

目次

さらにオススメの資料はありますか？ 225

第 1 章

Django Girls のチュートリアル

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

1.1 ウェルカム！

Django Girls のチュートリアルにようこと！ お会いできて嬉しいです :) このチュートリアルでは、ウェブテクノロジーの中身を見て回る旅へあなたをお連れします。私たちが知っているようにウェブを動かすのに必要なものすべてを垣間見ることができるでしょう。

知らないことを学ぶことは冒険のようなものです - でも心配はいりません。あなたはすでに勇気を出してここにいらっしゃるのですから、きっとうまくいくでしょう :)

1.2 はじめに

世界のますます色々なところに自分が（まだ）知らないテクノロジーが使われていると感じたことはありませんか？ ウェブサイトはどうやって作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？ ソフトウェアの世界は複雑すぎて一人でなにかに取り組むことはできないと考えたことはありませんか？

第1章 Django Girls のチュートリアルでは何を知ることができる？

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、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、このチュートリアルをおして示したいと思います。 テクノロジーをコワイと感じないように、できるだけ細かく説明していきますね。

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！！

1.3 チュートリアルでは何を知ることができる？

このチュートリアルを終わらせれば、実際に動作する、自分の小さなウェブアプリを1つ動かせるようになります。 私たちは、アプリをインターネット上で動かす方法を教えます。 インターネット上で動くようになれば、あなたが作ったアプリを世界中の誰でも見られるようになります！

このようなサイトができあがります！：

第 1 章 Django Girls のチュートリアル 1.4 自分の家でチュートリアルを進める

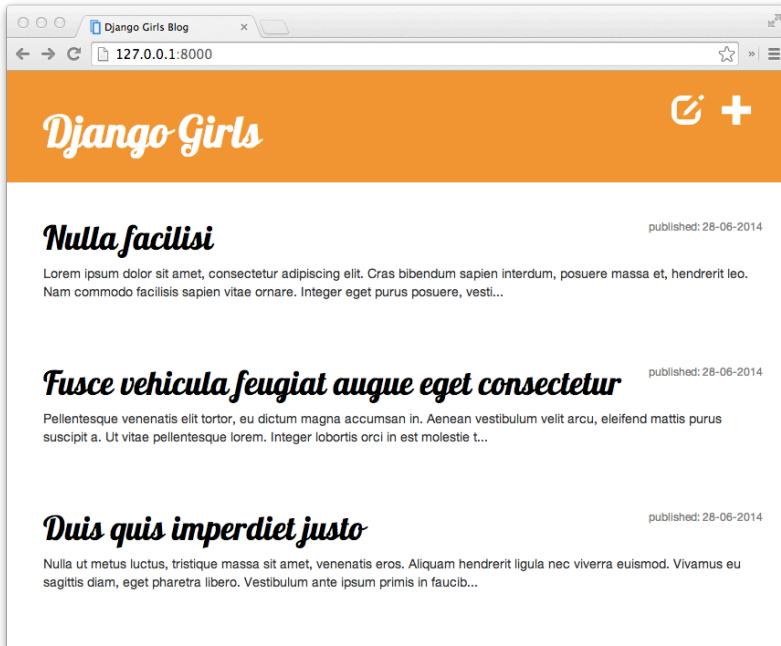


図 1.1: 図 0.1

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、質問できるコーチが周りにいない時は、ここにチャットを用意しています：

@<href>{<https://gitter.im/DjangoGirls/tutorial>, //image[tutorial][Gitter]{//\}}
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か、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！

それでは、??

1.4 自分の家でチュートリアルを進める

Django Girls のワークショップに参加することはすばらしいことです。でも、いつも参加できるとは限らないですよね。 したら、ご自宅でこのチュートリアルに取り組むことをおすすめします。 ご自宅で取り組まれる方向けに、チュー

トリアルを一人で進めやすくする動画の準備を進めています。動画の準備は進行中ですが、YouTube チャンネル Coding is for girls ですぐにますます多くのことがカバーされるでしょう。

動画がすでに準備された章には、動画へのリンクがあります。

1.5 コントリビューションについて

このチュートリアルは、DjangoGirls がメンテナンスしています。ミスを見つけたりチュートリアルを更新したくなった時は、コントリビューション・ガイドラインを読んでください。

1.6 チュートリアルの翻訳をしたい方へ

現在、翻訳は crowdin.com の次のプロジェクトで管理されています。

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

あなたが翻訳したい言語が crowdin ない時は、新しい issue を立てて、追加したい言語を私たちに教えてください。

第 2 章

Chromebook のセットアップ[°]

注意既にインストール手順を実行している場合は、これをやり直す必要はありません。すぐにスキップして??へ進んでください。

Chromebookをお持ちでない場合は、このセクションをスキップできます。もし利用している場合は、普通のインストールの作業とは少し異なります。インストール手順の残りの部分は無視できます。

Cloud 9

Cloud 9 はコードエディタや、インストール、書き込み、およびソフトウェアを実行できるインターネット上で動いているコンピューターへのアクセスを与えるツールです。チュートリアルをやっている間、Cloud 9 はあなたのローカルマシンのように動きます。クラスメートが OS X や Ubuntu、Windows でやっているのと同じようにターミナル上でコマンドを実行していても、あなたのターミナルは Cloud 9 がセットアップしたどこかのコンピューターにつながっています。

1. [Chrome ウェブストア](#)から Cloud 9 をインストールする
2. [C9.io](#) に行く
3. アカウントにサインアップします
4. 新しいワークスペースを作成をクリック
5. 名前は **django-girls** とします
6. **Blank** を選択 (下の行の右側から 2 番目のオレンジ色のロゴ)

インターフェイスには、サイドバー、テキストが書かれた大きなメインウィンドウ、下部に小さなウィンドウがあり、次の表示が見えると思います。

```
{% filename %}Cloud 9{% endfilename %}
```

```
yourusername:~/workspace $
```

この下部のエリアはあなたのターミナルで、Cloud 9 があなたの指示を待っています。ウィンドウサイズを少し大きくすることもできます。

仮想環境

仮想環境 (virtualenv とも呼ばれます) は、取り組んでいるプロジェクト用に、便利なコンピューターコードを詰め込んでおけるプライベートボックスのようなものです。様々なプロジェクトの様々なコードがプロジェクト間で混ざってしまわないようにこれをを使います。

Cloud 9 のインターフェイスのうち、下部のターミナルで次を実行します。

```
{% filename %}Cloud 9{% endfilename %}
```

```
sudo apt update  
sudo apt install python3.6-venv
```

それでも問題が解決しない場合は、コーチに相談してください。

次に、以下を実行します：

```
{% filename %}Cloud 9{% endfilename %}
```

```
mkdir djangogirls  
cd djangogirls  
python3.6 -mvenv myvenv  
source myvenv/bin/activate  
pip install django~={{ book.djangoproject_version }}
```

(最後の行はチルダに続けてイコール記号をつけることに気をつけてください:
~=)

GitHub

GitHub のアカウントを作成しましょう。

PythonAnywhere

Django Girls チュートリアルには、デプロイメントと呼ばれるものに関するセクションが含まれています。これはあなたの新しい Web アプリケーションの原動力となるコードを取得して、それを公にアクセス可能なコンピューター（サーバーと呼ばれます）に移動するプロセスです。これにより、あなたのやったことを他の人が見ることができます。

Chromebook でチュートリアルを行うとき、すでにインターネットに接続されているコンピュータ（ラップトップとは対照的に）を使用しているので、この作業は少し変に思えます。しかし、Cloud 9 のワークスペースを「開発中」の場所、PythonAnywhere をより完成したものを披露する場所として考えると役立ちます。

したがって、新しい PythonAnywhere アカウントにサインアップしてください。www.pythonanywhere.com

第3章

インターネットのしくみ

家で1人でこのチャプターに挑戦している方へ：このチャプターは、動画（英語）もあるので参考にしてください。 [How the Internet Works](#)
この章は、ジェシカ・マッケラー (<http://web.mit.edu/jesstess/www/>)
による「インターネットの仕組み」を参照しています。

私達は、毎日インターネットを使っています。でも、ブラウザのアドレス欄に [https://djangogirls.org](https://.djangoproject.org) のようなアドレスを入力して Enter キーを押すと何が起こるか、あなたは実際に知っていますか？

まず最初に理解する必要があるのは、Web サイトはハードディスクに保存された沢山のファイルであるということです。あなたが自分のパソコンのハードディスクに動画や音楽や画像を保存しているのと同じということです。しかし、Web サイトは動画や音楽、写真のようなデータとは違って、HTML というコンピュータのコードを持っているのです。

もし、あなたがプログラミングに精通していなかったら、最初は HTML も難しく感じるでしょう。でも、あなたがよく使う Web ブラウザ (Chrome や Safari や Firefox 等々) は HTML のコードが大好きです。Web ブラウザはこのコードがわかるようになっていて、コードの指示に従います。そして、あなたのウェブサイトのファイルをあなたが望む方法で表示するのです。

あなたのパソコンへファイルを保存するのと同じで、私達は HTML をハードディスクに保存する必要があります。インターネットの場合、サーバーと呼ばれるパワフルなコンピュータを使い、そのハードディスクに保存します。サーバーの主な目的は、データを保存したり、データを供給したりすることなので、サーバーは画面やマウス、キーボードを持っていません。サーバーはデータを供給する（サーブする）役割を持っているので、サーバーと呼ばれるのです。

はい、では、どのようにインターネットが見えるかを知りたいですよね？
私たちは絵を描いてみました。

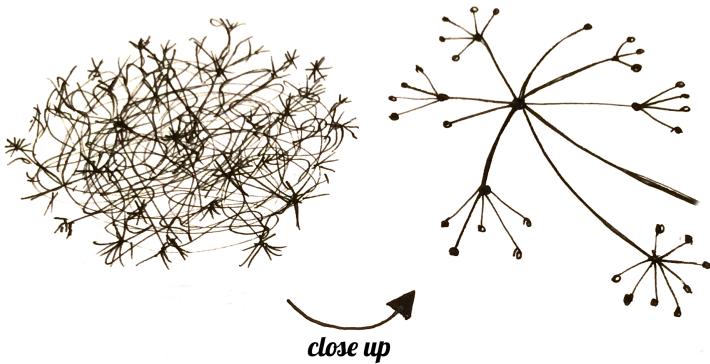


図 3.1: 図 1.1

上記の絵は混乱しているように見えますよね？ 接続されたマシン（サーバー）のネットワークは実際こんな感じです。 数十万台のマシン！ 世界中はりめぐらされたケーブル！ Submarine Cable Map の Web サイト (<http://submarinecablemap.com>) にアクセスすれば、ネットの複雑さを知ることができます。 ここにウェブサイトからのスクリーンショットがあります：

第3章 インターネットのしくみ

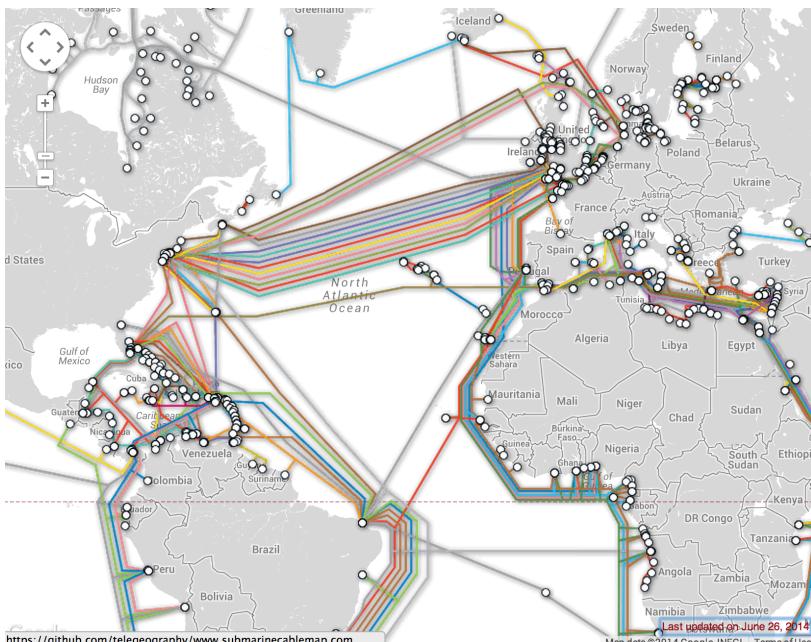


図 3.2: 図 1.2

すばらしいですね。でも、インターネットに接続されているすべてのマシンとマシンの間にワイヤを置くことは不可能です。したがって、マシン（例えば <http://djangogirls.org> が保存されているマシン）に到達するためには、多くの異なるマシンを介してリクエストを渡す必要があります。

こんな感じですね。

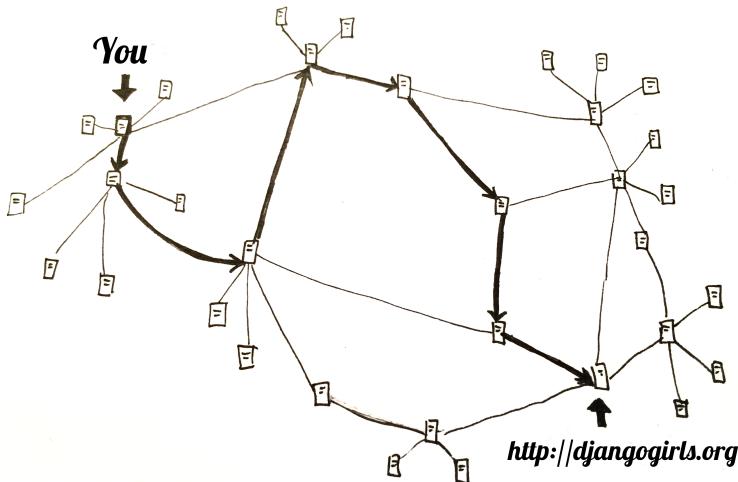


図 3.3: 図 1.3

あなたが <http://djangogirls.org> と入力すると、「親愛なる Django Girls へ。私は djangogirls.org の Web サイトが見たいです。それを私に送ってください」とリクエスト（手紙）を送ることになります。

あなたの手紙（リクエスト）は、まずあなたの一番近くの郵便局にいきますよね。そしてそこから、もう少し宛先に近い別の郵便局に行き、またそこからもう少し近い郵便局に行って・・そしてあなたの目的地まで届きます。特別なことがあります。同じ宛先に多くの手紙（データパケット）を送ると、まったく別の郵便局（ルーター）を通過して届く可能性があるということです。届くまでの道順は、郵便局ごとの配送方法次第です。

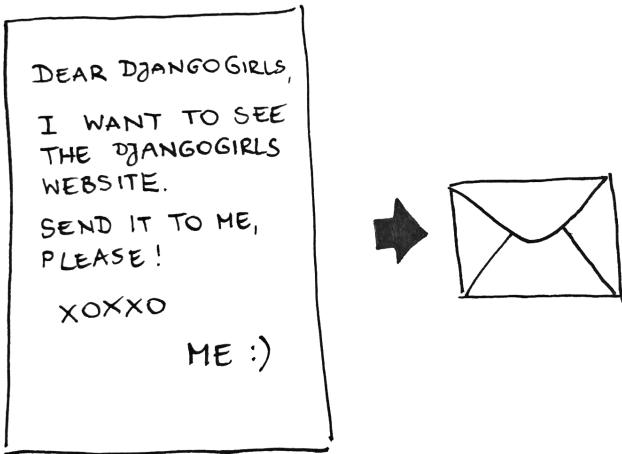


図 3.4: 図 1.4

このように動作しています。あなたはメッセージを送信し、何らかの応答を期待します。紙とペンではなく、データのバイトを使用しますが、アイデアは同じです！

市町村名、郵便番号、国名の住所の代わりに、IP アドレスを使用します。お使いのコンピュータは、まず djangogirls.org を IP アドレスに変換するように DNS (Domain Name System) に依頼します。あなたが連絡したい人の名前を探し、電話番号と住所を見つけることができる昔ながらの電話帳のようなものです。

手紙を送るときには、住所、切手など、正しく配達される特定の機能が必要ですよね。また、受信者が理解できる言語も使用している必要がありますよね？Web サイトを表示するために送信する データパケット についても同様です。HTTP (Hypertext Transfer Protocol) というプロトコルを使用します。(プロトコルとは正しく配達されるための機能や、使う言語などについての取り決めのことです)

だから、基本的に、あなたが Web サイトを持つなら、サーバー（マシン）が必要です。サーバーは リクエスト を（手紙で）受け取ると、Web サイトを（別の手紙で）返します。

これは Django チュートリアルですから、Django が何をしているのか知りたいでしょ？ あなたが返事を返す時、いつもみんなに同じ返事を返したいわけではなく、リクエストを送った人それぞれにパーソナライズされた返事を返した方がよいこともありますよね？ Django はパーソナライズされた面白い手紙を作るのに役立ちます。 :)

インターネットの話は以上です！ さあ、いよいよあなたのブログサイトを作成する時間です！

第 4 章

コマンドラインを使ってみよう

家で 1 人でこのチャプターに挑戦している方へ：このチャプターは、動画（英語）もあるので参考にしてください。 **Your new friend: Command Line**

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）待っています。

備考： このチュートリアルでは、”ディレクトリ” や”フォルダ”という用語が出てきますが、同じ意味です。

4.1 コマンドラインって何？

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、変更したりするものです。 グラフィカル・インターフェイスではないだけで、Windows のエクスプローラや Mac の Finder と同じ役割です。 このコマンドラインは、 cmd や CLI、プロンプト、コンソール、ターミナルと呼ばれることもあります。

4.2 コマンドラインインタフェースを開く

では、実際にコマンドラインを開いて、触ってみることとしましょう。

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
<!--sec data-title="Opening: Windows" data-id="windows_prompt" data-collapse=true ces-->
```

[スタート] → [Windows システムツール] → [コマンドプロンプト] を選択しましょう

Windows の古いバージョンの場合、[スタート] → [アクセサリ] → [コマンドプロンプト] です。

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
<!--endsec-->
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
<!--sec data-title="Opening: OS X" data-id="OSX_prompt" data-collapse=true ces-->
```

[アプリケーション] → [ユーティリティ] → [ターミナル] を選択しましょう。

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
<!--endsec-->
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
<!--sec data-title="Opening: Linux" data-id="linux_prompt" data-collapse=true ces-->
```

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。:)

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT  
<!--endsec-->
```

4.3 プロンプト

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。

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT  
<!--sec data-title="Prompt: OS X and Linux" data-id="OSX_Linux_prompt" data-collapse=true-->
```

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？

```
{% filename %}command-line{% endfilename %}
```

```
$
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT  
<!--endsec-->
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT  
<!--sec data-title="Prompt: Windows" data-id="windows_prompt2" data-collapse=true ces-->
```

Windowsの方は、>という記号が表示されていることでしょう。

```
{% filename %}command-line{% endfilename %}
```

```
>
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT  
<!--endsec-->
```

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> や Olas-MacBook-Air:~ ola\$ のような表示がありますね。これは間違いでありません。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。

\$や>と書かれているところまでを、コマンドラインプロンプトあるいは略してプロンプトと呼ばれます。プロンプトは、あなたがここに何かを入力することを促しています。

このチュートリアルでは、コマンドを入力してほしい時は、\$や>を含めて示しています。\$や>は無視して、プロンプト以降のコマンドを入力してください。

4.4 最初のコマンド（イエイ！）

次のようにコマンドを入力してみましょう。:

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT  
<!--sec data-title="Your first command: OS X and Linux" data-id="OSX_Linux_whoami" data-co
```

```
{% filename %}command-line{% endfilename %}
```

```
$ whoami
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT  
<!--endsec-->
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT  
<!--sec data-title="Your first command: Windows" data-id="windows_whoami" data-collapse=true
```

```
{% filename %}command-line{% endfilename %}
```

```
> whoami
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT  
<!--endsec-->
```

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
{% filename %}command-line{% endfilename %}

```
$ whoami  
olasitarska
```

ご覧のとおり、コンピューターがあなたのユーザー名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！

4.5 基本

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。

カレントディレクトリ（現在のディレクトリ）

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）、知りたいですよね？では、このようにキーボードで入力して、Enterキーをおすください。

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
<!--sec data-title="Current directory: OS X and Linux" data-id="OSX_Linux_pwd" data-collapse=true ce-->
```

{% filename %}command-line{% endfilename %}

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

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。

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
<!--endsec-->
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
<!--sec data-title="Current directory: Windows" data-id="windows_cd" data-collapse=true ce-->
```

```
{% filename %}command-line{% endfilename %}
```

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

補足: 'cd' は、'change directory' を意味しています。Powershell を使うと、Linux や Mac OS X のように pwd コマンドを使えます。

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT  
<!--endsec-->
```

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。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。

ファイルとディレクトリの一覧

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。

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT  
<!--sec data-title="List files and directories: OS X and Linux" data-id="OSX_Linux_ls" data-
```

```
{% filename %}command-line{% endfilename %}
```

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

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT  
<!--endsec-->
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT  
<!--sec data-title="List files and directories: Windows" data-id="windows_dir" data-collapse="1"-->
```

{% filename %}command-line{% endfilename %}

```
> 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  
...
```

補足：Powershell では、ls コマンドを Linux や Mac OS X 同様に使えます。 <!--endsec-->

カレントディレクトリの変更

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。

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT  
<!--sec data-title="Change current directory: OS X and Linux" data-id="OSX_Linux_move_to"-->
```

{% filename %}command-line{% endfilename %}

```
$ cd Desktop
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT  
<!--endsec-->
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT  
<!--sec data-title="Change current directory: Windows" data-id="windows_move_to" data-collap
```

{% filename %}command-line{% endfilename %}

```
> cd Desktop
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT  
<!--endsec-->
```

本当に変更されたかどうか確認してみてください：

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT  
<!--sec data-title="Check if changed: OS X and Linux" data-id="OSX_Linux_pwd2" data-collap
```

{% filename %}command-line{% endfilename %}

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

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT  
<!--endsec-->
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT  
<!--sec data-title="Check if changed: Windows" data-id="windows_cd2" data-collapse=true ce
```

{% filename %}command-line{% endfilename %}

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

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT  
<!--endsec-->
```

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。

ディレクトリの作成

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。

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT  
<!--sec data-title="Create directory: OS X and Linux" data-id="OSX_Linux_mkdir" data-collapse=true ce
```

{% filename %}command-line{% endfilename %}

```
$ mkdir practice
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
<!--endsec-->
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
<!--sec data-title="Create directory: Windows" data-id="windows_mkdir" data-collapse=true
```

{% filename %}command-line{% endfilename %}

```
> mkdir practice
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
<!--endsec-->
```

この短いコマンドで、デスクトップに practice 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` や `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。

エクササイズ！

練習をしてみましょう。先ほど作成した `practice` 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。)

解答：

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
<!--sec data-title="Exercise solution: OS X and Linux" data-id="OSX_Linux_test_dir" data-c-->
```

{% filename %}command-line{% endfilename %}

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

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
<!--endsec-->
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
<!--sec data-title="Exercise solution: Windows" data-id="windows_test_dir" data-collapse=t-->
```

{% filename %}command-line{% endfilename %}

```
> cd practice
```

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

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT  
<!--endsec-->
```

おめでとうございます！ よくできました！

クリーンアップ[°]

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。

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT  
<!--sec data-title="Clean up: OS X and Linux" data-id="OSX_Linux_back" data-collapse=true
```

```
{% filename %}command-line{% endfilename %}
```

```
$ cd ..
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT  
<!--endsec-->
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT  
<!--sec data-title="Clean up: Windows" data-id="windows_back" data-collapse=true ces-->
```

```
{% filename %}command-line{% endfilename %}
```

```
> cd ..
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT  
<!--endsec-->
```

`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。

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT  
<!--sec data-title="Check location: OS X and Linux" data-id="OSX_Linux_pwd3" data-collapse=
```

```
{% filename %}command-line{% endfilename %}
```

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

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT  
<!--endsec-->
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
<!--sec data-title="Check location: Windows" data-id="windows_cd3" data-collapse=true ces-
```

{% filename %}command-line{% endfilename %}

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

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
<!--endsec-->
```

では、practice ディレクトリを削除しましょう。

注意！： del や rmdir、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。

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
<!--sec data-title="Delete directory: Windows Powershell, OS X and Linux" data-id="OSX_Lin-->
```

{% filename %}command-line{% endfilename %}

```
$ rm -r practice
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
```

```
<!--endsec-->
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
```

```
<!--sec data-title="Delete directory: Windows Command Prompt" data-id="windows_rmdir" data-
```

```
{% filename %}command-line{% endfilename %}
```

```
> rmdir /S practice
```

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

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
```

```
<!--endsec-->
```

できました! 本当に削除されたか、確認してみましょう。:

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
```

```
<!--sec data-title="Check deletion: OS X and Linux" data-id="OSX_Linux_ls2" data-collapse=
```

```
{% filename %}command-line{% endfilename %}
```

```
$ ls
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
```

```
<!--endsec-->
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT  
<!--sec data-title="Check deletion: Windows" data-id="windows_dir2" data-collapse=true ces-
```

```
{% filename %}command-line{% endfilename %}
```

```
> dir
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT  
<!--endsec-->
```

終了

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? :)

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
```

```
<!--sec data-title="Exit: OS X and Linux" data-id="OSX_Linux_exit" data-collapse=true ces-
```

```
{% filename %}command-line{% endfilename %}
```

```
$ exit
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
<!--endsec-->
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
<!--sec data-title="Exit: Windows" data-id="windows_exit" data-collapse=true ces-->
```

{% filename %}command-line{% endfilename %}

```
> exit
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
<!--endsec-->
```

かっこいいですね? :)

4.6 まとめ

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	例
exit	exit	ウインドウを閉じる	exit
cd	cd	ディレクトリを変更	cd test
cd	pwd	現在のディレクトリを表示	cd (Windows)
dir	ls	ディレクトリ/ファイルの一覧を表示	dir
copy	cp	ファイルのコピー	copy c:\test\
move	mv	ファイルを移動	move c:\tes
mkdir	mkdir	新しいディレクトリを作成	mkdir testd
rmdir (or del)	rm	ファイルを削除	del c:\test\
rmdir /S	rm -r	ディレクトリを削除	rm -r testdi

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。

4.7 準備 OK?

よし、次は Python を勉強していきましょう！

第 5 章

Python をはじめよう

ついにここまで来ましたね！

まずは、最初に Python とは何かお話をさせて下さい。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すから！

第 6 章

Python のインストール

注意：Chromebook をお使いの場合、このチャプターは飛ばして、??の章をすすめてください。

注意：すでにインストール手順を実行している場合は、これをもう一度行う必要はありません。次の章に進んでください。

家で 1 人でこのチャプターに挑戦している方へ：この章は、動画（英語）もあるので参考にしてください。 [Installing Python & Code Editor](#)

このセクションは Geek Girls Carrots (<https://github.com/ggcarrots/django-carrots>) のチュートリアルをベースに作成しました。

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 3.6 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。

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
<!--sec data-title="Install Python: Windows" data-id="python_windows" data-collapse=true c
```

Windows をお使いのかたは、まずシステム情報を開き、システムの種類が 32-bit バージョンか 64-bit バージョンかを確認します（システム情報の開き方：Windows キー + Pause/Break キー もしくは コントロールパネル>システムとセキュリティ>システムを開く）。Python for Windows は、<https://www.python.org/downloads/windows/> からダウンロードできます。

第 6 章 Python のインストール

「Latest Python 3 Release - Python x.x.x」のリンクをクリックしてください。お使いのコンピュータが **64 ビット** 版の Windows を実行している場合は、**Windows x86-64 executable installer** をダウンロードしてください。32 ビット版の場合は、**Windows x86 executable installer** をダウンロードします。インストーラをダウンロードしたら、それを実行して（ダブルクリックして）インストーラの指示に従ってください。

インストール時に必ず「**Add Python 3.6 to PATH**」にチェックをいれましょう：



図 6.1: Python のパスを通すのを忘れないようにしてください。

次のステップでは、Windows コマンドラインを使用します（コマンドラインについて詳しくは後ほど詳しく教えます）。現時点では、コマンドを入力する必用がある場合、スタートメニューから検索フィールドに「コマンドプロンプト」入力してください。（Windwos のバージョンが古い場合、「スタートメニュー」をクリックして、表示されるアプリ一覧から「Windows システムツール」を選んで、「コマンドプロンプト」をクリックしてください。）あるいは、「Windows キー」を押しながら「R」キーを押すと、「ファイル名を指定して実行」が現れます。コマンドプロンプトを開くには、そこに "cmd" と入力して enter キーを押します。

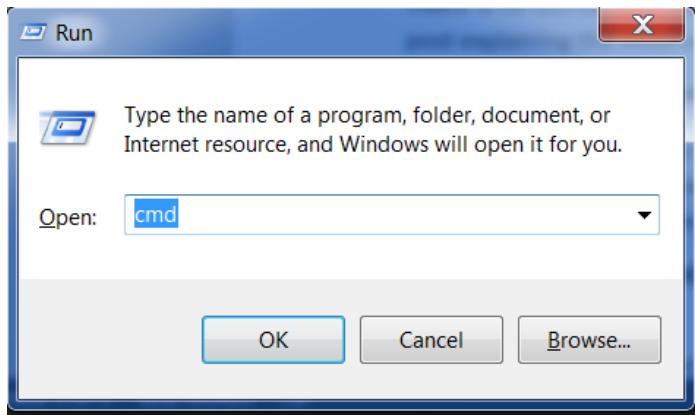


図 6.2: "ファイル名を指定して実行"で、"cmd"と入力してください

注意：古いバージョンの Windows (7、Vista、またはそれ以前のバージョン) を使用していて、Python 3.6.x インストーラがエラーで失敗した場合、次のいずれかを試みることができます：

1. すべての Windows アップデートをインストールして、Python 3.6 を再インストールしてみる。
2. 古いバージョンの Python をインストールしてみる。例えば [3.4.6](#)。

古いバージョンの Python をインストールする場合、インストール画面は上記のものとは多少異なる場合があります。下にスクロールして「Add python.exe to Path」のところを左クリックして「Will be installed on local hard drive (ローカルハードドライブにインストールされます)」を選択してください：

第 6 章 Python のインストール

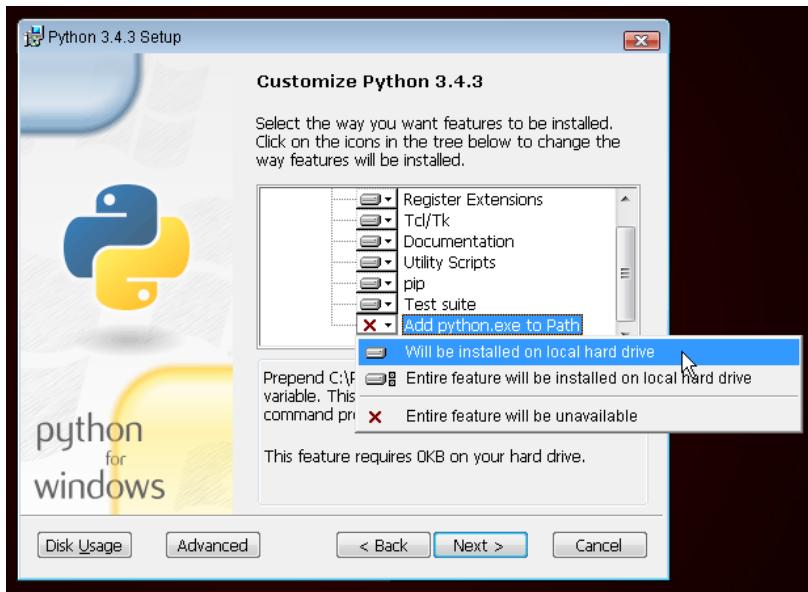


図 6.3: 古いバージョンのパスを追加してください

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
<!--endsec-->
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
<!--sec data-title="Install Python: OS X" data-id="python OSX"
data-collapse=true ces-->
```

注意：OS X に Python をインストールする前に、Mac の設定で App Store 以外のパッケージをインストールできるようにする必要があります。「システム環境設定」（「アプリケーション」フォルダ内）に移動し、「セキュリティとプライバシー」、「一般」タブの順にクリックします。「ダウンロードしたアプリを許可する」が「Mac App Store」に設定されてい

第6章 Python のインストール

る場合は、「Mac App Store と識別された開発者」に変更します。

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/>

- **macOS 64-bit/32-bit installer** ファイルをダウンロードしてください。
- **python-3.6.5-macosx10.6.pkg** 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。

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
<!--endsec-->
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
<!--sec data-title="Install Python: Linux" data-id="python_linux"
data-collapse=true ces-->
```

おそらくほとんどの場合、すぐに使える Python がすでにインストールされているでしょう。インストールされているか（とそれがどのバージョンか）を確認するため、コンソールを起動して次のコマンドを打ってください。

```
{% filename %}command-line{% endfilename %}
```

```
$ python3 --version
Python 3.6.5
```

このとき、バージョンの数字の最後が違っていたとしても（例えば、3.6.0など）、アップグレードする必要はありません。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。

第6章 Python のインストール

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT  
<!--endsec-->
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT  
<!--sec data-title="Install Python: Debian or Ubuntu" data-id="python_debian" data-collapse=true ces-->
```

次のコマンドをコンソールに打って下さい。

```
{% filename %}command-line{% endfilename %}
```

```
$ sudo apt install python3.6
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT  
<!--endsec-->
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT  
<!--sec data-title="Install Python: Fedora" data-id="python_fedora" data-collapse=true ces-->
```

次のコマンドをコンソールに打って下さい。

```
{% filename %}command-line{% endfilename %}
```

```
$ sudo dnf install python3
```

古いバージョンの Fedora の場合は、コマンド `dnf` が見つからないというエ

第6章 Python のインストール

ラーが表示されます。その場合は、代わりに yum を使用してください。

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT  
<!--endsec-->
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT  
<!--sec data-title="Install Python: openSUSE" data-id="python_openSUSE"  
data-collapse=true ces-->
```

次のコマンドをコンソールに打って下さい。

```
{% filename %}command-line{% endfilename %}
```

```
$ sudo zypper install python3
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT  
<!--endsec-->
```

インストールされているか確認するために、コマンドプロンプトを開いて次のように Python3 コマンドを入力してください。

```
{% filename %}command-line{% endfilename %}
```

```
$ python3 --version  
Python 3.6.1
```

注意： Windows をお使いで python3 が見つからないエラーが出る場合、代わりに python と入力して、バージョンが Python3.6 と出てくるか確認してください。(3を抜いてみましょう)

第 6 章 Python のインストール

分からぬ時や、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。

第 7 章

コードエディタ

家で 1 人でこのチャプターに挑戦している方へ：このチャプターは、動画（英語）もあるので参考にしてください。 [Your new friend: Command Line](#)

まさにこれから、はじめてのコードを書いていきます。コードエディタをダウンロードしましょう！

補足： Choromebook を使っている方は、このチャプターは飛ばして、 ?? の説明に従ってセットアップしてください。

注意 インストールの章で、コードエディタのインストールを前もって済ませた人もいるかもしれません。もしそうなら、飛ばして次の章に進んでください！

本当にたくさんのエディタがありますが、どれを使うかは、結局は、ほとんど個人好みの問題です。ほとんどの Python プログラマは、PyCharm など、複雑ですが非常に強力な IDE（統合開発環境）というものを使っています。しかし、初心者にとって IDE はそれほど適していないかもしれませんので、同じくらい強力で、もっとシンプルなものをオススメします。

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。

7.1 Gedit

Gedit はオープン ソースの無料エディタで、すべての OS で利用可能です。
[ダウンロード](#)

7.2 Sublime Text 3

Sublime Text は非常に人気のあるエディタで、無料の試用期間があります。簡単にインストールして使用することができ、すべての OS に対応しています。

[ダウンロード](#)

7.3 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](#)

7.4 なんでコードエディタをインストールするの？

なんで Word や Notepad のようなものを使わず、特別なコードエディタをインストールするのかなと思うかもしれません。

まず、コードはプレーンテキストという、何の装飾もない生の文字列でないといけませんが、問題は、Word や Textedit のようなプログラムが生成するのは、実は、プレーンテキストではないということなんです。Word や Textedit は、RTF（リッチテキストフォーマット）などのような、改造された形式を使って、フォントや書式を持ったリッチテキストを生成します。

2つ目の理由は、コードエディタは、コードを編集することに特化しているので、コードを意味によって色づけして強調したり、引用符を自動的に閉じたりするような、便利な機能を提供できます。

あとで、こういった動作が全てわかるようになります。すぐに、信頼できる使い古したコードエディタを、お気に入りのツールの一つだと思うようになると思いますよ^^

```
{% set warning_icon = '<span class="glyphicon glyphicon-exclamation-sign" style="color: red;" aria-hidden="true" data-toggle="tooltip" title="An error is expected when you run this command!" ></span>' %}
```

第 8 章

Python 入門

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。<https://github.com/ggcarrots/django-carrots>

さあ、コードを書いてみましょう！

8.1 Python プロンプト

家で 1 人でこのパートに挑戦している方へ：このパートと続くパートは、動画（英語）もあるので参考にしてください。 [Python Basics: Integers, Strings, Lists, Variables and Errors](#)

Python であそぶために、コマンドラインを開きましょう。やり方は、チャプター ?? 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` キーを押してください。

{% filename %}command-line{% endfilename %}

```
$ python3
Python 3.6.1 (...)

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

8.2 最初の Python コマンド！

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 コンソールを終わる時は、`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。そうするともう `>>>` は出なくなります。

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。簡単な計算からはじめましょう。`2 + 3` とタイプして、`Enter` キーを押してください。

```
{% filename %}command-line{% endfilename %}
```

```
>>> 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`

`2 の 3 乗`のような指数の計算は、次のようにタイプします。

```
{% filename %}command-line{% endfilename %}
```

```
>>> 2 ** 3  
8
```

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。

8.3 文字列

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。

```
{% filename %}command-line{% endfilename %}
```

```
>>> "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。

```
{% filename %}command-line{% endfilename %}
```

```
>>> "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。

```
{% filename %}command-line{% endfilename %}
```

```
>>> "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。

```
{% filename %}command-line{% endfilename %}
```

```
>>> "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。

```
{% filename %}command-line{% endfilename %}
```

```
>>> '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。

```
{% filename %}command-line{% endfilename %}
```

```
>>> "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 (`function`) もあります！

```
{% filename %}command-line{% endfilename %}
```

```
>>> 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ことがあります。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 "0la" をかっこの中にいれているのです。

まとめ

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、答えるのがかえってきます。
- 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？

8.4 エラー

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は、数字の文字数は数えられるでしょうか？`len(304023)` と記述して、Enterキーを押してみましょう。

```
{% filename %}{{ warning_icon }} command-line{% endfilename %}
```

```
>>> 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！ {{ warning_icon }} アイコンのついたコードは思ったように動かないコードです。（今回はチュートリアルで用意されていましたが）思ったように動かないことは学ぶ上で大事です！

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、文字数を数えられるはずですよね？

```
{% filename %}command-line{% endfilename %}
```

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

うまく行きました！ `str` 関数を `len` のかっこ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？

8.5 変数

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単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。

```
{% filename %}command-line{% endfilename %}
```

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

`name` イコール (=) "Ola" とタイプします。

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` キーを押してください。

```
{% filename %}command-line{% endfilename %}
```

```
>>> 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。

```
{% filename %}command-line{% endfilename %}
```

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

変数には関数も使えます。

```
{% filename %}command-line{% endfilename %}
```

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

素晴らしいですね！ 変数は、数値にも使えますよ。

```
{% filename %}command-line{% endfilename %}
```

```
>>> 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！

```
{% filename %}{warning_icon} command-line{% endfilename %}
```

```
>>> 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！

8.6 print 関数

次に挑戦してみましょう。

```
{% filename %}command-line{% endfilename %}
```

```
>>> 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'`の文字列表現 (**representation**) を返します。ここでは、シングルクォーテーション（'）に囲まれた M-a-r-i-a という文字の集まりです。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。

8.7 リスト

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。今から **list** というものを紹介していきます。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。

```
{% filename %}command-line{% endfilename %}
```

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

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。

```
{% filename %}command-line{% endfilename %}
```

```
>>> 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ょうか？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！

```
{% filename %}command-line{% endfilename %}
```

```
>>> 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。

```
{% filename %}command-line{% endfilename %}
```

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

これは何も返してきません。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。

```
{% filename %}command-line{% endfilename %}
```

```
>>> 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。

```
{% filename %}command-line{% endfilename %}
```

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

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。

```
{% filename %}command-line{% endfilename %}
```

```
>>> 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、アイテムがリストのどこにあるかを指す番号です。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。

```
{% filename %}command-line{% endfilename %}
```

```
>>> 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。

リストから要素を消すには、これまで学んできたインデックスと `pop()` メソッドを使います。例で試してみましょう。リストの最初の要素を削除しています。

```
{% filename %}command-line{% endfilename %}
```

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

お見事！

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>

8.8 辞書（ディクショナリ）

家で1人でこのパートに挑戦している方へ：このパートは、動画（英語）もあるので参考にしてください。 [Python Basics: 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。

```
{% filename %}command-line{% endfilename %}
```

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

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）

```
{% filename %}command-line{% endfilename %}
```

```
>>> 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 `name` が指す値は `'Ola'` (`string` オブジェクト)
- キー `country` が指す値は `'Poland'` (`string` オブジェクト)
- キー `favorite_numbers` が指す値はリスト `[7, 42, 92]` (数字を3つ持つ `list`)

次のように書くと各キーの値を確認できます。

```
{% filename %}command-line{% endfilename %}
```

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

リストに似ていますね。しかし、ディクショナリでは、インデックスを覚えておく必要がなく、キーの名前でいいのです。

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！

```
{% filename %}{{ warning_icon }} command-line{% endfilename %}
```

```
>>> 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！ この後の行を読むまえに、答えを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、作ったあとに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。

```
{% filename %}command-line{% endfilename %}
```

```
>>> 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。

```
{% filename %}command-line{% endfilename %}
```

```
>>> 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、`pop()` メソッドを使います。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。

```
{% filename %}command-line{% endfilename %}
```

```
>>> participant.pop('favorite_numbers')
[7, 42, 92]
>>> 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。

```
{% filename %}command-line{% endfilename %}
```

```
>>> 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。面白くなってきましたか？ その調子です！

まとめ

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？ :)

8.9 比較

家で1人でこのパートに挑戦している方へ：このパートは、動画（英語）もあるので参考にしてください。Python Basics: Comparisons

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。

```
{% filename %}command-line{% endfilename %}
```

```
>>> 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。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は使います。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。等しくないことを比較するときは、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。

```
{% filename %}command-line{% endfilename %}
```

```
>>> 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。

```
{% filename %}command-line{% endfilename %}
```

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

複数の数値を比較して複雑になっても、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 の場合のみ、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。

```
{% filename %}{{ warning_icon }} command-line{% endfilename %}
```

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

8.10 ブール型 (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。

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 は間違います。 - True と記述してください (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。

```
{% filename %}command-line{% endfilename %}
```

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

このようなこともできます。

```
{% filename %}command-line{% endfilename %}
```

```
>>> 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！

第 9 章

保存しよう！

家で 1 人でこのパートに挑戦している方へ：このパートと続くパートは、動画（英語）もあるので参考にしてください。Python Basics: Saving files and "If" statement

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 行ず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す。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() 関数を記述してください。

```
{% filename %}command-line{% endfilename %}
```

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

これで、コマンドプロンプトに戻りました。

第9章 保存しよう！

前のチャプター ?? 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。

```
{% filename %}editor{% endfilename %}
```

```
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 おしゃえます。

メモ コードエディタでは色に注目しましょう！これはとてもクールです。Python コンソールでは、すべての文字は同じ色です。エディタでは、`print` 関数は文字列とは違う色がつきます。これは「シンタックスハイライト」と呼ばれています。エディタは構文（シンタックス）を強調（ハイライト）します。コードを書くとき、これはとても役に立ちます。色のおかげで、文字列の最後のクォーテーションの書き忘れや、キーワードの名前（この後学ぶ関数の `def` など）のタイプに気づくことができます。これが私たちがコードエディタを使う理由の 1 つです。:)

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。

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
<!--sec data-title="Change directory: OS X" data-id="python OSX"
data-collapse=true ces-->
```

Mac では、コマンドは次のようになります。

```
{% filename %}command-line{% endfilename %}
```

第9章 保存しよう！

```
$ cd ~/Desktop
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT  
<!--endsec-->
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT  
<!--sec data-title="Change directory: Linux" data-id="python_linux"  
data-collapse=true ces-->
```

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）

```
{% filename %}command-line{% endfilename %}
```

```
$ cd ~/Desktop
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT  
<!--endsec-->
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT  
<!--sec data-title="Change directory: Windows Command Prompt" data-id="python_windows" dat
```

Windowsのコマンドプロンプトでは、次のようにになります。

```
{% filename %}command-line{% endfilename %}
```

第9章 保存しよう！

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

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT  
<!--endsec-->
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT  
<!--sec data-title="Change directory: Windows Powershell" data-id="python_windowsPSH" data-
```

Windows の PowerShell では、次のようにになります。

```
{% filename %}command-line{% endfilename %}
```

```
> cd $Home\Desktop
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT  
<!--endsec-->
```

うまくできない時は、質問してください。まさにそのためにコーチがここにいます！

次に、ファイルのコードを実行します。

```
{% filename %}command-line{% endfilename %}
```

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

メモ：Windows でファイルのコードを実行するときは、'python3' の代わりに 'python' とタイプしましょう。

```
{% filename %}command-line{% endfilename %}
```

```
> python python_intro.py
```

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。

9.1 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。

```
{% filename %}python_intro.py{% endfilename %}
```

```
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。

```
{% filename %}{{ warning_icon }} command-line{% endfilename %}
```

```
$ 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。

```
{% filename %}python_intro.py{% endfilename %}
```

```
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？ 条件式が 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。タブかスペースか決めたら、変えないようにしましょう。例えばスペース4つでインデントにしたら、この後もスペース4つでインデントするようにしましょう。インデントにスペースとタブを混ぜてしまうと問題が発生してしまうことがあります。

保存して、もう一度実行してみましょう。

```
{% filename %}command-line{% endfilename %}
```

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

メモ：Windowsでは'python3'とタイプしてもうまくいかないことを思い出してください。この後ファイルを実行するときは'python3'の代わりに'python'とタイプしてくださいね。

条件が **True** じゃないときは？

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。

```
{% filename %}python_intro.py{% endfilename %}
```

```
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。

```
{% filename %}command-line{% endfilename %}
```

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

もし 2 が 5 より大きかったら、4 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？

```
{% filename %}python_intro.py{% endfilename %}
```

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

実行すると…

```
{% filename %}command-line{% endfilename %}
```

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

どうなったかわかりましたか？ `elif` を追加する事で、上記の条件（`name == 'Ola'`）が `True` でない場合に実行する条件を追加することができます。

最初の `if` の条件分岐の後に、好きなだけ `elif` を追加することができます。例えば…

```
{% filename %}python_intro.py{% endfilename %}
```

```
volume = 57  
if volume < 20:  
    print("It's kinda quiet.")
```

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

Python は上から順番に各条件をテスト、実行し、出力します。

```
{% filename %}command-line{% endfilename %}
```

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

9.2 コメント

コメントは # で始まる行です。# の後にはなんでも書くことができ、Python はそれを無視します。コメントを書いたコードは、ほかの人にとってもわかりやすくなります。

コメントを書いてみましょう。

```
{% filename %}python_intro.py{% endfilename %}
```

```
# ボリュームが大きすぎたり小さすぎたりしたら変更する  
if volume < 20 or volume > 80:  
    volume = 50  
    print("That's better!")
```

コードのすべての行にコメントを書く必要はありません。コメントには、コードの中である処理をする理由や、複雑なコードの動きのまとめを書くと役に立ちます。

まとめ

直近のエクササイズを通して、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。
- コメント - あなたがコードについて記述できる行。 Python は実行しません。

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！

9.3 自作の関数！

家で1人でこのパートに挑戦している方へ：このパートは、動画（英語）もあるので参考にしてください。[Python Basics: Functions](#)

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

実行する処理をひとまとめにしたものを作ったことがあります。Python では、関数は `def` というキーワードからはじめり、引数（ひきすう）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` の中身を下記のコードに書き換えてください。

```
{% filename %}python_intro.py{% endfilename %}
```

```
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！

ここではあなたは、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。

```
{% filename %}command-line{% endfilename %}
```

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

メモ：思ったように動かなくても慌てないで！ 画面の出力は動かない理由をつかむのに役立ちます。

- `NameError` が出ている場合、おそらく何かミスタイプがあります。同じ名前を使っているかチェックしましょう。関数を定義するときは `def hi():` としていますか？ 関数を実行するときは `hi()` としていますか？
- `IndentationError` が出ている場合、`print` 関数の 2 行が同じ数のスペースでインデントされているかチェックしましょう。関数の中のコードは同じ数のスペースでインデントされていると Python は考えます。
- 画面に何も表示されていない場合、最後の `hi()` がインデントされていないかチェックしましょう - もしインデントされていたら、関数の一部になってしまっています。関数が呼び出されません。

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。

```
{% filename %}python_intro.py{% endfilename %}
```

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

このとおり、関数に `name` という引数を足します。

```
{% filename %}python_intro.py{% endfilename %}
```

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

hi()
```

if 文では print 関数は 4 つのスペースでインデントしていましたね。条件式が True のときに print 関数が実行されました。関数の中の if 文で実行する print 関数の前には、上記のように、スペースを 8 つ入れて 2 回分インデントします。実行して、どのように動くか見てみましょう。

```
{% filename %}{% warning_icon %} command-line{% endfilename %}
```

```
$ 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。

```
{% filename %}python_intro.py{% endfilename %}
```

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

実行してください。

```
{% filename %}command-line{% endfilename %}
```

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

では、名前を変えてみたらどうなりますか？

```
{% filename %}python_intro.py{% endfilename %}
```

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

再度実行してください。

```
{% filename %}command-line{% endfilename %}
```

```
$ 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？ やつてみて、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。

```
{% filename %}command-line{% endfilename %}
```

```
Hi anonymous!
```

すばらしいですね。挨拶をする人の名前を変えるたびに繰り返しコードを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付けるのは大変ですよね。

```
{% filename %}python_intro.py{% endfilename %}
```

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

hi("Rachel")
```

では、実行してみましょう。

```
{% filename %}command-line{% endfilename %}
```

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

おめでとうございます！ 関数の書き方を学びましたね！ :)

9.4 ループ

家で1人でこのパートに挑戦している方へ：このパートは、動画（英語）もあるので参考にしてください。Python Basics: For Loop

さあ、もう最後のパートですよ。あっという間ですね。 :)

先ほどお話ししたとおり、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。

```
{% filename %}python_intro.py{% endfilename %}
```

```
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。ループの中で使いましょう。

```
{% filename %}python_intro.py{% endfilename %}
```

```
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。

```
{% filename %}python_intro.py{% endfilename %}
```

```
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。

```
{% filename %}command-line{% endfilename %}
```

```
$ 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いたことが繰り返されています。

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

```
{% filename %}python_intro.py{% endfilename %}
```

```
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。

```
{% filename %}command-line{% endfilename %}
```

```
1
2
3
4
5
```

`range` 関数は、連続する数値を要素とするリストを作ります。引数に指定した開始の数値から終了の数値までのリストです。

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。

9.5 まとめ

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せん。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！

ご参考に、公式の完全な Python チュートリアルは <https://docs.python.org/3/tutorial/> にあります。挑戦すると Python の理解をもっと深められるでしょう。ここまで頑張りました！

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。



図 9.1: カップケーキ

第 10 章

Django ってなに？

Django (/ˈdʒæŋgəʊ / **jang-goh** ジャンゴと読みます) 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 フレームワークは、素早く、簡単にウェブサイトを開発するのに役立つコンポーネントの一式が含まれています。

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、私たちは開発をイチから作り直して車輪の再発明をする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。

10.1 なぜフレームワークを必要とするか。

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のことで、ポストは Web サーバーのポートのことです。Web サーバーが行っています。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。返事を送ろうとする時、コン

10.2 誰かがあなたの WEB サイトにリクエストを要求したときどうなりますか。

第 10 章 Django ってなに？

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。

10.2 誰かがあなたの WEB サイトにリクエストを要求したときどうなりますか。

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。)。 パターンのリストを受け取って、URL に一致するものを探します。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。

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！

第 11 章

Django のインストール

補足： Chromebook を使っている方は、このチャプターは飛ばして、
?? の説明に従ってセットアップしてください。

補足：インストールのチャプターで既にインストール済みの方は、この
チャプターは飛ばして次に進みましょう。

このチャプターの一部は Geek Girls Carrots
(<https://github.com/ggcarrots/django-carrots>) 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。

11.1 仮想環境

Django をインストールする前に、あなたのコーディング環境を、きれいにしておく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から！

さあ、仮想環境 (**virtual environment**) (`virtualenv` とも呼ばれています) を作成してみましょう。仮想環境 (virtual environment) ではプロジェクト単位であなたの Python/Django のセットアップを他から隔離します。これは、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、あなたが開発している

他のサイトに影響を及ぼさないということです。便利でしょ？

あなたがしなければならないのは、あなたが仮想環境 (virtual environment) 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)。

補足： Windows の方は、ディレクトリ名に特殊文字やアクセント記号を含まないよう気をつけてください。もし、ユーザー名が特殊文字を含む場合は、C:\djangogirls のようなディレクトリを作成してください。

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。

```
{% filename %}command-line{% endfilename %}
```

```
$ mkdir djangogirls  
$ cd djangogirls
```

myvenv という仮想環境 (virtual environment) 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：

```
{% filename %}command-line{% endfilename %}
```

```
$ python3 -m venv myvenv
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT  
<!--sec data-title="Virtual environment: Windows" data-id="virtualenv_installation_windows"  
data-collapse=true ces-->
```

新しい virtualenv を作成するために、コマンドプロンプトを開き（コマンドプロンプトについては何章か前にお話ししましたね。覚えてますか？）、`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：

{% filename %}command-line{% endfilename %}

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

`myvenv` というところが、あなたの `virtualenv`（仮想環境）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ーあなたはこの名前を何度も参照しますから！

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
<!--endsec-->
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
<!--sec data-title="Virtual environment: Linux and OS X" data-id="virtualenv_installation_
data-collapse=true ces-->
```

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：

{% filename %}command-line{% endfilename %}

```
$ python3 -m venv myvenv
```

`myvenv` は、あなたの 仮想環境 (`virtualenvironment`) 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ーあなたはこの名前を何度も参照しますから！

補足：Debian や Ubuntu のバージョンによっては、以下のエラーが出ることがあります。

```
{% filename %}command-line{% endfilename %}  
//emlist{The virtual environment was not created successfully because  
ensurepip is not available. On Debian/Ubuntu systems, you need to  
install the python3-venv package using the following command. apt  
install python3-venvYou may need to use sudo with that command.  
After installing the python3-venv package, recreate your virtual envi-  
ronment.
```

この場合、エラー内の指示にしたがって、`python3-venv` のパッケージをインストールしましょう。

```
$ sudo apt install python3-venv
```

補足：Debian や Ubuntu のバージョンによっては、この仮想環境を実行すると、次のようなエラーがでます。

```
{% filename %}command-line{% endfilename %}
```

```
Error: Command '['/home/eddie/Slask/tmp/venv/bin/python3', '-Im', 'ensurepip', '--upgrade'
```

このエラーを回避するために、代わりに `virtualenv` コマンドを使います。

```
{% filename %}command-line{% endfilename %}
```

```
$ sudo apt install python-virtualenv  
$ virtualenv --python=python3.6 myvenv
```

補足：もし以下ののようなエラーがでたら、

```
{% filename %}command-line{% endfilename %}
```

```
E: Unable to locate package python3-venv
```

代わりに次のコマンドを実行してください。

```
{% filename %}command-line{% endfilename %}
```

```
sudo apt install python3.6-venv
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT  
<!--endsec-->
```

11.2 仮想環境の操作

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。

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT  
<!--sec data-title="Working with virtualenv: Windows" data-id="virtualenv_windows"  
data-collapse=true ces-->
```

実行して、仮想環境を起動します。

```
{% filename %}command-line{% endfilename %}
```

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

補足：Windows 10 では、`execution of scripts is disabled on this system` というエラーが Windows PowerShell に出ることがあります。その場合は、Windows PowerShell を「管理者として開く」で、管理者権限で新しくウィンドウを開いてください。そして、仮想環境を開始する前に、以下のコマンドを入力してください。

```
{% filename %}command-line{% endfilename %}
//emlist{C:\WINDOWS\system32>      Set-ExecutionPolicy      -
ExecutionPolicy RemoteSigned Execution Policy Change The
execution policy helps protect you from scripts that you do not
trust. Changing the execution policy might expose you to the
security risks described in the about_Execution_Policies help topic
at http://go.microsoft.com/fwlink/?LinkId=135170. Do you want to
change the execution policy? [Y] Yes [A] Yes to All [N] No [L] No to
All [S] Suspend [?] Help (default is "N"): A
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
<!--endsec-->
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
<!--sec data-title="Working with virtualenv: Linux and OS X" data-id="virtualenv_linuxosx"
data-collapse=true ces-->
```

実行して、仮想環境を起動します。

```
{% filename %}command-line{% endfilename %}
```

```
$ source myvenv/bin/activate
```

myvenv のところをあなたが選んだ仮想環境 (virtualenvironment) 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：

```
{% filename %}command-line{% endfilename %}
//emlist{$ . myvenv/bin/activate
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT  
<!--endsec-->
```

`virtualenv` が起動すると、プロンプトの行頭に (`myvenv`) が現れます。

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！

11.3 Django のインストール

今度はあなたの `virtualenv` を起動したので、Django をインストールすることができます。

これを行う前に、Django のインストールに使用する最新バージョンの `pip` がインストールされている必要があります。

```
{% filename %}command-line{% endfilename %}
```

```
(myvenv) ~$ python3 -m pip install --upgrade pip
```

Requirements ファイルによってパッケージをインストールする

Requirements ファイルは `pip install` でインストールするためのパッケージリストが記載されているファイルです:

前にインストールしたコードエディタを使用して、最初に `requirements.txt` ファイルを `djangogirls/` フォルダーの中に作ります:

```
djangogirls
└── requirements.txt
```

`djangogirls/requirements.txt` ファイル中に以下のテキストを追加します:

```
{% filename %}djangogirls/requirements.txt{% endfilename %}
```

```
Django~={{ book.djangoproject_version }}
```

そして、`pip install -r requirements.txt` を実行して Django をインストールします。

```
{% filename %}command-line{% endfilename %}
```

```
(myenv) ~$ pip install -r requirements.txt
Collecting Django~={{ book.djangoproject_version }} (from -r requirements.txt (line 1))
  Downloading Django-{{ book.djangoproject_version }}-py3-none-any.whl (7.1MB)
    Installing collected packages: Django
      Successfully installed Django-{{ book.djangoproject_version }}
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
<!--sec data-title="Installing Django: Windows" data-id="django_err_windows"
data-collapse=true ces-->
```

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、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）で作成することを検討してみてください。新しいディレクトリに新しい仮想環境を作成してから、古いディレクトリを削除して、上記のコマンドを試してください。（仮想環境のディレクトリは、絶対パスが使われているので、移動させてもうございません。）

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
<!--endsec-->
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
<!--sec data-title="Installing Django: Windows 8 and Windows 10" data-id="django_err_windows"
data-collapse=true ces-->
```

Django をインストールしようとしてコマンドラインがフリーズして動かなくなってしまうことがあります。その時は、以下のコマンドを代わりに入力してみてください。

```
{% filename %}command-line{% endfilename %}
//emlist{C:\Users\Name\.djangogirls> python -m pip install -r requirements.txt
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
<!--endsec-->
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
<!--sec data-title="Installing Django: Linux" data-id="django_err_linux"
data-collapse=true ces-->
```

Ubuntu 12.04 で pip を呼んだときにエラーが起きた場合は、仮想環境 (virtualenv) 内に pip を再インストールするために `python -m pip install -U --force-reinstall pip` を実行して下さい。

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
<!--endsec-->
```

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！

第 12 章

プロジェクトを作成しよう！

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。(<https://github.com/ggcarrots/django-carrots>)

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](#) をもとにしています。この django-marcador tutorial は Markus Zapke-Gründemann たちが著作権を保有しています。

ここからは、小さ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。基本的に、Django のスクリプトを実行し Django プロジェクトの骨格を作ります。スクリプトは、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。作成されたファイルの名前は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えません。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。もしコンソールのプロンプトの前に (myvenv) という文字が表示されていない時は、virtualenv が実行されていないので、有効にする必要があります。Django のインストールのチャプターの 仮想環境の操作 のパートで、仮想環境を実行する方法について説明しました。Windows では、`myenv\Scripts\activate`、MacOS や Linux では、`source myenv/bin/activate` というコマンドを入力すると有効にできます。

第12章 プロジェクトを作成しよう！

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
<!--sec data-title="Create project: OS X or Linux" data-id="django_start_project OSX_Linux-->
```

MacOS や Linux の場合は、コンソールで以下のコマンドを実行します。最後のピリオド(ドット)。を忘れないようにしてください！

{% filename %}コマンドライン{% endfilename %}

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

コマンドの最後にピリオド。を入力したことを確認してくださいね。このピリオドは、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しています(ピリオド。は、現在のディレクトリを表す省略表記です)。

メモ：上記のコマンドを入力するときは、`django-admin` で始まる部分のみを入力することを忘れないでください。ここに書いた(`myvenv`)
`~/djangogirls$` の部分は、コマンドライン上で入力を受け付けることを示しているプロンプトの一例なので、人によって違うかもしれません。

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
<!--endsec-->
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
<!--sec data-title="Create project: Windows" data-id="django_start_project_windows" data-c-->
```

Windows の場合は、以下のコマンドを実行しないといけません。(最後にピリオド(ドット)@<tt>{.\}</tt>を書いてください)

{% filename %}コマンドライン{% endfilename %}

第12章 プロジェクトを作成しよう！

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

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。(ピリオドは簡略表記です)。

メモ：上記のコマンドを入力するときは、django-admin で始まる部分のみを入力することを忘れないでください。ここに書いた (myvenv) ~/djangogirls\$ の部分は、コマンドライン上で入力を受け付けることを示しているプロンプトの一例なので、人によって違うかもしれません。

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
<!--endsec-->
```

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
│   ├── settings.py
│   ├── urls.py
│   ├── wsgi.py
│   └── __init__.py
└── requirements.txt
```

注：ディレクトリ構造には、以前作成した venv ディレクトリもあります。

manage.py はサイトの管理に役立つスクリプトです。それを使用して、他のものをインストールすることなく、私たちのコンピュータ上で Web サーバーを起動することができます。

settings.py ファイルには、ウェブサイトの設定が含まれています。

手紙を送付する場所を確認する郵便業者について話した事を覚えていますか？

`urls.py` ファイルには、`urlresolver` で使用されるパターンのリストが含まれています。

私たちが変更しない他のファイルを今は無視しましょう。覚えておくべき唯一の事は、間違えてそれらを削除しないことです！

12.1 設定変更

`mysite/settings.py` にいくつか変更を加えましょう。前にインストールしたコードエディタを使用してファイルを開きます。

注：`settings.py` は他のものと同じように通常のファイルであることに注意してください。「ファイルを開く」メニューを使用して、コードエディタ内から開くことができます。これにより、`settings.py` ファイルに移動して選択できる通常のウインドウが表示されます。あるいは、デスクトップの djangogirls フォルダに移動して右クリックしてファイルを開くこともできます。次に、リストからコードエディタを選択します。インストールされた他のプログラムでは、ファイルを開けても編集できないかもしれないのに、エディタの選択は重要です。

私たちの Web サイトが正しい時間で動くといいでしょう。[Wikipedia のタイムゾーンのリスト](#)に移動して、関連するタイムゾーン (TZ) をコピーします（例：`Asia/Tokyo`）。

`settings.py` の中から `TIME_ZONE` と書かれた行を探してください。この行はタイムゾーンを表しているので、自分が住んでいるタイムゾーンに合わせて修正しましょう。たとえば、次のように書きます。

```
{% filename %}mysite/settings.py{% endfilename %}
```

```
TIME_ZONE = 'Asia/Tokyo'
```

言語コードは、あなたの利用する言語を設定する必要があります。英語の場合は `en`、ドイツ語の場合は `de`、国コードの場合は `de` はドイツ、`ch` はスイスです。英語があなたの母国語でない場合、これを追加して Django のデフォルトのボタンや通知をあなたの言語に変更することができます。ですのでたとえば「Cancel」ボタンがここで定義した言語に翻訳されます。[Django には多くの言語が付属しています](#)。

別の言語を使用する場合は、次の行を変更して言語コードを変更します。

```
{% filename %}mysite/settings.py{% endfilename %}
```

```
LANGUAGE_CODE = 'ja'
```

静的ファイルのパスも追加する必要があります。（静的ファイルと CSS についてでは、後ほどチュートリアルで説明します）。ファイルの一番下に移動し、STATIC_URL の下に STATIC_ROOT を追加します。：

```
{% filename %}mysite/settings.py{% endfilename %}
```

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

DEBUG が True に設定されていて、ALLOWED_HOSTS が空のリストの時は、自動的に ['localhost', '127.0.0.1', '[::1]'] という 3 つのホストに対してチェックが行われます。このままの設定では、これから私たちがデプロイして使う PythonAnywhere のホストネームが含まれていません。ですから、次のように設定を変更します。

```
{% filename %}mysite/settings.py{% endfilename %}
```

```
ALLOWED_HOSTS = ['127.0.0.1', '.pythonanywhere.com']
```

メモ： Chromebook を使っている人は、次の 1 行を settings.py ファイルの最後に追加してください。 MESSAGE_STORAGE = 'django.contrib.messages.storage.session.SessionStorage' cloud9 のサービスを使っている人は、.c9users.io も ALLOWED_HOSTS に追加してください。

12.2 データベースをセットアップする

あなたのサイトのデータを保管することができるデータベース・ソフトウェアには、たくさんの種類があります。今は、Django がデフォルトで使う `sqlite3` というデータベースを使うことにします。

この設定はすでに `mysite/settings.py` ファイルの中に次のように書かれています。

```
{% filename %}mysite/settings.py{% endfilename %}
```

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

ブログのデータベースを作成するには、コンソールで次のコードを実行してみましょう： `python manage.py migrate` (`manage.py` ファイルのある `djangogirls` ディレクトリにいる必要があります)。 うまくいったら次のように表示されるでしょう：

```
{% filename %}command-line{% endfilename %}
```

```
(myvenv) ~/djangogirls$ python manage.py migrate
Operations to perform:
  Apply all migrations: auth, admin, contenttypes, sessions
Running migrations:
  Rendering model states... DONE
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
```

```
Applying auth.0006_require_contenttypes_0002... OK
Applying auth.0007_alter_validators_add_error_messages... OK
Applying sessions.0001_initial... OK
```

終わったら、Webサーバーを起動し、私たちのWebサイトが動作しているかどうかを確認する時間です。

12.3 ウェブサーバを起動する

コマンドラインやコマンドプロンプトで `manage.py` ファイルを含むディレクトリ (`djangogirls` ディレクトリ) に移動してください。`python manage.py runserver` を実行して Web サーバーを起動できます。

```
{% filename %}command-line{% endfilename %}
```

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

Chromebookを使用している場合は、代わりに次のコマンドを使用します。

```
{% filename %}Cloud 9{% endfilename %}
```

```
(myvenv) ~/djangogirls$ python manage.py runserver 0.0.0.0:8080
```

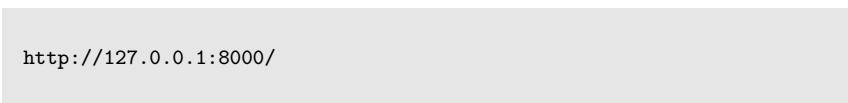
Windows上で、`UnicodeDecodeError`で失敗した場合は、代わりに次のコマンドを使用します。

```
{% filename %}command-line{% endfilename %}
```

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

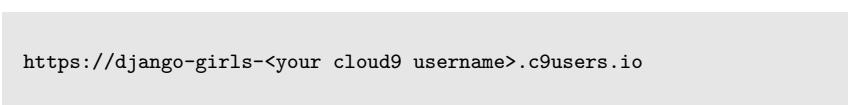
これであなたのウェブサイトが稼働していることを確認するだけです。ブラウザ (Firefox、Chrome、Safari、Internet Explorerなど) を開き、次のアドレスを入力します。

```
{% filename %} ブラウザ{% endfilename %}
```



Chromebook を使用している場合は、次の URL からテストサーバーにアクセスします。

```
{% filename %} ブラウザ{% endfilename %}
```



おめでとう！ たった今、あなたは最初のウェブサイトを作って、それをウェブサーバーの上で起動しました！ 素晴らしいですね！

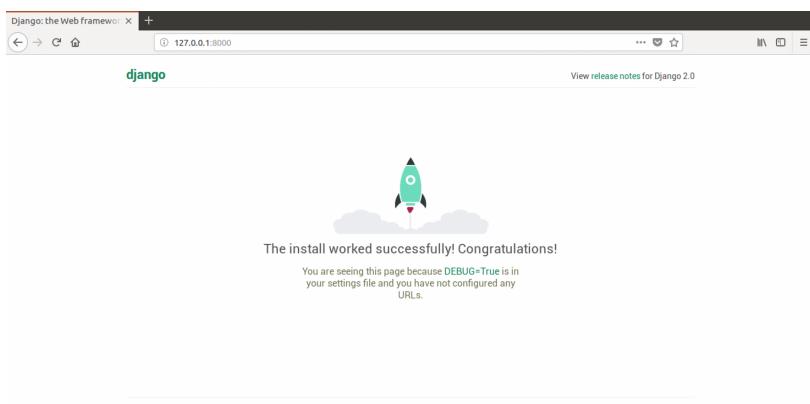


図 12.1: インストールできました！

Web サーバーが稼働している間は、追加のコマンドを入力するための新しいコマンドラインプロンプトは表示されません。新しいテキストを受け入れますが、新しいコマンドは実行しません。これは、Web サーバーが動作している間はずっとリクエストを待つためです。

Web サーバーの仕組みについては、「インターネットの仕組み」の章を参考してください。

Web サーバーの実行中に追加のコマンドを入力するには、新しいターミナル ウィンドウを開き、virtualenv をアクティブにします。 Web サーバーを停止するには、実行中のウィンドウに戻り、CTRL + C - Control キーと C キーを同時に押します (Windows では Ctrl + Break キーを押す必要があります)。

次のステップに進む準備はできましたか？ 今度は実際にコンテンツを作り始めましょう！

第 13 章

Django モデル

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 オブジェクトについてちょっとお話しします。

13.1 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って、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。ピンと来ないでど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
purr()
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）と命令（メソッドと呼びます）を持つコード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タイトルが必要ですね。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、ポストを一つ作成、公開したか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！

13.2 Django モデル

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。

新しいアプリケーションの作成

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ために、次のコマンドをコンソールの中で走らせましょう。`(manage.py` ファイルがある `djangogirls` ディレクトリでコマンドをタイプしてくださいね)

```
{% filename %}Mac OS X and Linux:{% endfilename %}
```

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

```
{% filename %}Windows:{% endfilename %}
```

```
(myenv) C:\Users\Name\djangogirls> python manage.py startapp blog
```

新しく `blog` 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
├── blog
│   ├── __init__.py
│   ├── admin.py
│   ├── apps.py
│   ├── migrations
│   │   └── __init__.py
│   ├── models.py
│   ├── tests.py
│   └── views.py
├── db.sqlite3
├── manage.py
└── mysite
    ├── __init__.py
    ├── settings.py
    ├── urls.py
    └── wsgi.py
└── requirements.txt
```

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いけません。それは `mysite/settings.py` です。エディタでこれを開いてください。まず `INSTALLED_APPS` を見つけて] の上に '`blog`' という一行を追加します。そうすると、最終的には以下のようになりますね。

```
{% filename %}mysite/settings.py{% endfilename %}
```

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'blog',
]
```

ブログポストモデルの作成

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

blog/models.py をエディタで開いて全部削除し、下のコードを書きましょう。

```
{% filename %}blog/models.py{% endfilename %}
```

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

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

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

    def __str__(self):
        return self.title
```

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、"ダンダー"（ダブルアンダースコア）と呼ばれます。

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 ... で必要部分を入れることができます。

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）。モデルの名前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？ 数値？ 日付？ 他のオブジェクト、例えばユーザーとの関係？）

- `models.CharField` - 文字数が制限されたテキスト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す。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う？
- `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/2.0/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）/メソッドという意味です。`publish` はメソッドの名前で、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、ポストのタイトルのテキスト（`string`）が返ってきます。

`def publish(self):` と `def __str__(self):` の両方が `class` キーワードに続く行でインデントされているのに気づきましたか？ Python にモデルのメソッドだと伝えるために、`class` キーワードに続く行ではメソッドをインデントしましょう。そうしないと、メソッドはモデルのものではなくなり、思ってもない振る舞いをするでしょう。

もしモデルがまだはっ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のためのテーブルを作成する

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、モデルに少し変更があったこと（今作ったこと）を Django に知らせましょう。コソールで `python manage.py makemigrations blog` とタイプします。こんな感じですね。

```
{% filename %}command-line{% endfilename %}
```

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

メモ：編集したファイルを忘れずに保存してくださいね。保存しないと、コンピュータが以前のバージョンのファイルを実行してしまい、思ってもみないエラーメッセージに出くわすかもしれません。

Django が作ってくれた移行ファイルを私たちがデータベースに追加すれば完了です。`python manage.py migrate blog` とタイプするとこうなるでしょう。

```
{% filename %}command-line{% endfilename %}
```

```
(myenv) ~/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！

第 14 章

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。

blog/admin.py ファイルをエディタで開いて、内容をこのように変えて下さい：

```
{% filename %}blog/admin.py{% endfilename %}
```

```
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（管理画面）上で見えるようにするため、`admin.site.register(Post)` でモデルを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 サーバーを実行するコンソールで `python manage.py runserver` を実行してください。 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。

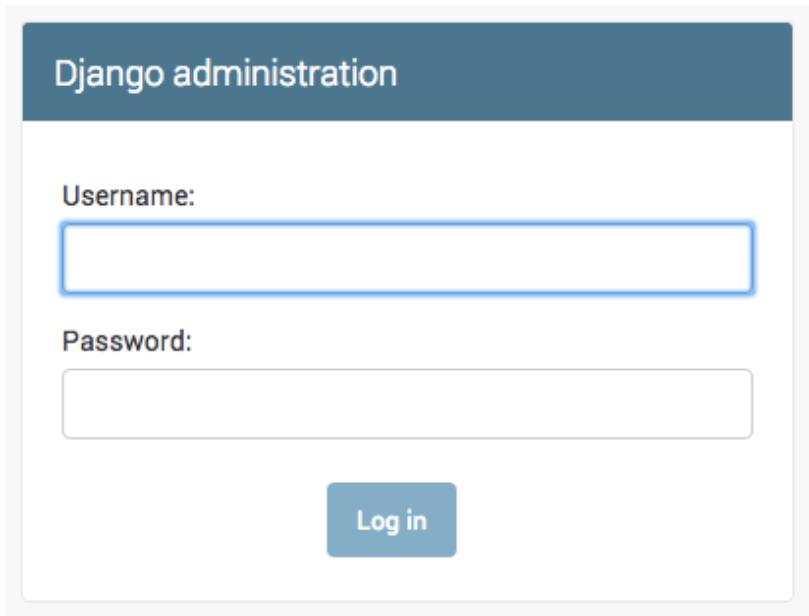


図 14.1: ログインページ

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り、`python manage.py createsuperuser`と入力し、enter キーを押します。

Web サーバーを実行しているときに新しいコマンドを入力したい場合は、新しいターミナルウィンドウを開き、仮想環境（virtualenv）を有効にすることを思い出してください。プロジェクトを作成しよう！のウェブサーバを起動するセクションでどうやって新しいコマンドを書くかというのを見直しました。

{% filename %} Mac OS X または Linux:{% endfilename %}

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

第 14 章 Django admin

```
{% filename %}Windows:{% endfilename %}
```

```
(myenv) C:\Users\Name\djangogirls> python manage.py createsuperuser
```

プロンプトが表示されたら、ユーザー名（小文字、スペースなし）、電子メール アドレス、およびパスワードを入力します。 タイプして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。）

```
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。

The screenshot shows the Django admin dashboard. At the top, there's a dark blue header bar with the text "Django administration". Below it, a light blue navigation bar has the text "Site administration". Underneath, there are two main sections:

- AUTHENTICATION AND AUTHORIZATION**: This section contains two entries: "Groups" and "Users". Each entry has a green "Add" button and a yellow "Change" button.
- BLOG**: This section contains one entry: "Posts". It also has a green "Add" button and a yellow "Change" button.

図 14.2: Django admin

第 14 章 Django admin

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。（ブログ機能を確認するためなので、内容は何でも大丈夫です。自分で適当な文字列を打ち込んでももちろん OK です。）

少なくとも 2 つまたは 3 つの記事（すべてではない）は公開日がセットされているようにしてください。後ほど役に立ちます。

Django administration

WELCOME, KOJO. VIEW SITE / CHANGE PASSWORD / LOG OUT

Home › Blog › Posts › Add post

Add post

Author: kajo

Title:

Text:

Created date:

Date: Today

Time: Now

Published date:

Date: Today

Time: Now

図 14.3: Django admin

Django admin についてもっと知りたいときは、Django のドキュメントを見るとよいでしょう。
<https://docs.djangoproject.com/ja/2.0/ref/contrib/admin/>

ここでそろそろコーヒー（または紅茶）か何か食べるものを摂って自分を元気づけるのにいいタイミングでしょう。最初の Django モデルを作ったのだから、少し休みをとっていいところです！

第 15 章

デプロイ！

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、つまずき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。 そういうたつまずき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つたとしても、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。

今のところ、あなたが作ったサイトは、あなたのコンピューターでしかみることができません。 ここでは、デプロイの方法を学びましょう！ 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能够になりますよ。 :)

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。 インターネットで利用できる多くのサーバー プロバイダーがありますが、私達は [PythonAnywhere](#) を使用します。 PythonAnywhere 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を無料で公開できますので今のあなたには最適でしょう。

使用するその他の外部サービスは [GitHub](#)、コードのホスティング サービスです。 他にも色々ありますが、ほとんどのプログラマは GitHub のアカウントを持っています。 そしてあなたも今そなります。

これら 3 つの場所が重要になります。 ローカルコンピューターは、開発およびテストを行う場所になります。 変更に満足したら、GitHub 上にプログラムの

第 15 章 デプロイ！

コピーを配置します。あなたのウェブサイトは PythonAnywhere で公開され、GitHub からコードの新しいコピーを取得することによって更新されます。

第 16 章

Git

注：もし、すでにインストールしていた場合は再度行う必要はありません。
次のセクションに進んであなたの Git リポジトリを作り始められます。

Git はたくさんのプログラマが利用する「バージョン管理システム」です。このソフトウェアは、特定のバージョンを後で呼び出すことができるよう、時間の経過とともにファイルへの変更を追跡することができます。 Microsoft Word の”トラックの変更 (track changes)”のようですが、はるかに強力です。

16.1 Git のインストール

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
<!--sec data-title="Installing Git: Windows" data-id="git_install_windows"
data-collapse=true ces-->
```

git-scm.com から Git をダウンロードすることができます。1つのステップを除いて「次へ」を押して進んで大丈夫です。「PATH 環境を調整する (Adjusting your PATH environment)」というステップでは、「Windows コマンドプロンプトから Git とオプションの Unix ツールを使用する (Use Git and optional Unix tools from the Windows Command Prompt)」(一番下の選択肢) を選択します。それ以外はデフォルトの設定値で構いません。改行コードの変換 (Configuring the line ending conversions) については、「Checkout Windows-style, commit Unix-style line endings」の選択で大丈夫です。

インストールが正常に終了した後、コマンドプロンプトまたは PowerShell を

再起動することを忘れないでください。<!--endsec-->

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
<!--sec data-title="Installing Git: OS X" data-id="git_install OSX"
data-collapse=true ces-->
```

git-scm.com から Git をダウンロードし、指示に従ってください。

注 OS X 10.6,10.7、または 10.8 を実行している場合は、ここから git のバージョンをインストールする必要があります: [Git installer for OS X Snow Leopard](#)

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
<!--endsec-->
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
<!--sec data-title="Installing Git: Debian or Ubuntu" data-id="git_install_debian_ubuntu"
data-collapse=true ces-->
```

{% filename %}command-line{% endfilename %}

```
$ sudo apt install git
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
<!--endsec-->
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
<!--sec data-title="Installing Git: Fedora" data-id="git_install_fedora"
data-collapse=true ces-->
```

{% filename %}command-line{% endfilename %}

```
$ sudo dnf install git
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
<!--endsec-->
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
<!--sec data-title="Installing Git: openSUSE" data-id="git_install_opensuse"
data-collapse=true ces-->
```

{% filename %}command-line{% endfilename %}

```
$ sudo zypper install git
```

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT
<!--endsec-->
```

16.2 Git リポジトリを始める

Git はコードリポジトリ（または略して「リポジトリ」）というものの中に置かれる特定のファイルへの変更を追跡します。私たちのプロジェクトを開始しましょう。あなたのコンソールを開き、`djangogirls` ディレクトリでこれらのコマンドを実行します。

備考：リポジトリを初期化する前に `pwd` (OSX/Linux) または `cd` (Windows) コマンドで現在の作業ディレクトリを確認してください。`djangogirls` フォルダー内にいる必要があります。

```
{% filename %}command-line{% endfilename %}
```

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

git リポジトリを初期化することは、プロジェクトごとに 1 回だけ行う必要があります（ユーザー名と電子メールをもう一度入力する必要はありません）。

Git はこのディレクトリ内のすべてのファイルとフォルダの変更を追跡しますが、無視してほしいいくつかのファイルがあります。ベースディレクトリ内で `.gitignore` という名前のファイルを作成することによってこれを行います。あなたのエディターを開き、次の内容で新しいファイルを作成します。

```
{% filename %}.gitignore{% endfilename %}
```

```
*.pyc
*~
__pycache__
myvenv
db.sqlite3
/static
.DS_Store
```

これを "djangogirls" フォルダ内に `.gitignore` という名前で保存します。

備考：ファイル名の先頭のドットは重要です！もしそのファイルを作るのが難しいなら、（Macをお使いの方はFinderからドット（.）で始まるファイルを作れません。）そういう時はエディタでSave Asから作成すれば問題ありません。

備考：`.gitignore` ファイルで指定したファイルの1つが `db.sqlite3` です。そのファイルはローカルデータベースで、すべての投稿が保存されます。PythonAnywhere上のあなたのウェブサイトは別のデータベースを使うことになるので、これをあなたのリポジトリには追加したくありません。データベースは開発マシンのように SQLite にすることができますが、通常は SQLite よりも多くのサイト訪問者に対処できる MySQL と呼ばれるものを使用します。どちらの方法でも、GitHubのコードのコピーでは SQLite データベースを無視することで、これまでに作成したすべての投稿はそのままローカルで利用できますが、本番環境（ブログを開ける PythonAnywhereのことです）ではそれらの投稿を再び追加する必要があります。ローカルデータベースは本当のブログ投稿をブログから削除してしまうことを心配せずに、さまざまなことをテストできるよい遊び場として考えるといいでしょう。

`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` コマンドを使用する事をおすすめします。これは間違ったファイルを追加またはコミットなど思いもかけない事を止めるために役立ちます。`git status` コマンドは、あらゆる追跡されていない/変更されている/ステージされている（untracked/modified/staged）ファイルや、ブランチの状態などさまざまな情報を返します。出力は次のようになります。

```
{% filename %}command-line{% endfilename %}
```

```
$ git status
On branch master

Initial commit

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

    .gitignore
    blog/
    manage.py
```

```
mysite/  
requirements.txt  
  
nothing added to commit but untracked files present (use "git add" to track)
```

最後に、変更内容を保存します。コンソールに移動し、これらのコマンドを実行します。

```
{% filename %}command-line{% endfilename %}
```

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

16.3 GitHub にコードをプッシュする

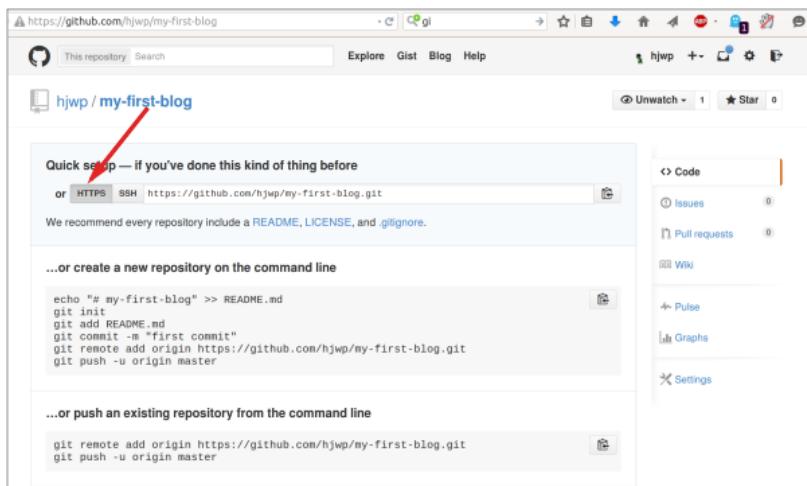
[GitHub.com](#) にアクセスし、新しい無料のユーザーアカウントを登録します。
(もしあなたがすでにワークショップの準備でそれをしていたら、それは素晴らしいことです！)

そして、新しいリポジトリに "my-first-blog" の名前で新しいリポジトリを作成します。 "README" で初期化する"チェックボックスをオフのままにし、.gitignore オプションを空白にして（手動で行っています）、ライセンスを None のままにしておきます。

The screenshot shows the GitHub 'Create repository' interface. At the top, it asks for the 'Owner' (set to 'hjwp') and 'Repository name' ('my-first-blog'). Below that is a note about repository names being short and memorable, with a suggestion like 'ducking-octo-tyrion'. A 'Description (optional)' field is present. The 'Visibility' section shows 'Public' (selected) and 'Private'. A red arrow points from the 'Public' label to the 'Private' label. Below this is a section for initializing the repository with a README, which is currently unchecked. Another red arrow points from the 'Initialize this repository with a README' label to the 'None' dropdown. At the bottom are two dropdowns: 'Add .gitignore: None' and 'Add a license: None', followed by a 'Create repository' button.

注 `my-first-blog` という名前は重要です。何か他のものを選択することもできますが、以下の手順では何度も繰り返す必要があります。他の名前を選択した場合は、毎回それを置き換えてください。できれば、`my-first-blog` の名前にしておきましょう。

次の画面では、リポジトリをクローンするための URL が表示されます。これはこの後のコマンドで利用します。



そして自分のコンピューター上の Git リポジトリを GitHub 上の Git リポジトリに結びつけてあげる必要があります。

コンソールに次のように入力します（<your-github-username>を GitHub アカウントの作成時に入力したユーザー名に置き換えます。山カッコ<>を残さないでください。この URL はさっき見たクローン URL と一致する必要があります）。

```
{% filename %}command-line{% endfilename %}
```

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

あなたの GitHub のユーザー名とパスワードを入力すると、次のように表示されます：

```
{% filename %}command-line{% endfilename %}
```

```
Username for 'https://github.com': ola
Password for 'https://ola@github.com':
```

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

```
XXX: BLOCK_HTML: YOU SHOULD REWRITE IT  
<!--TODO: maybe do ssh keys installs in install party, and point ppl who dont have it to a
```

あなたのコードは今 GitHub 上にあります。見に行きましょう！ [Django](#) や [Django Girls Tutorial](#)、その他たくさんの素晴らしいオープンソースソフトウェアプロジェクトも GitHub でコードをホストしています。 :)

第 17 章

PythonAnywhere でブログを設定する

17.1 PythonAnywhere アカウントにサインアップする

備考：あなたがすでに PythonAnywhere のアカウントを以前に作成しインストールの手順をふんでいたら、再びそれを行う必要はありません。

PythonAnywhere はクラウド上のサーバーで Python のコードを走らせるためのサービスです。私たちはこれを私たちのサイトをホスティングして、インターネット上で動かし続けるために使います。

PythonAnywhere で「Beginner」アカウントにサインアップしましょう（クレジットカードのいらない無料利用枠で大丈夫です）。

- <https://www.pythonanywhere.com>

Plans and pricing

Beginner: Free!

A limited account with one web app at your-username.pythonanywhere.com, restricted outbound internet access from your apps, low CPU/bandwidth, no Python/Jupyter notebook support.
It works and it's a great way to get started!

Create a Beginner account

図 17.1: PythonAnywhere サインアップページが表示する無料の「Beginner」アカウント作成ボタン

注 ユーザー名を選ぶとき、あなたのブログの URL があなたのユーザー名.pythonanywhere.com になることを覚えておいて下さい。ですから、自分のニックネームかあなたのブログが何についてのブログか分かるような名前を選んでみて下さい。

17.2 PythonAnywhere API トークンの作成

これはあなたが一度だけ行う必要があるものです。 PythonAnywhere にサインアップしたとき、ダッシュボードページが表示されます。 右上にある「Account」ページへのリンクをクリックし、それから「API Token」タブを選択し、「Create new API token」ボタンをクリックします。

Upgrade/Downgrade Account Email and security Teacher API Token

Your API token

You do not have an API token yet.

Create a new API token

By clicking this button you agree that you understand that this API is new and

図 17.2: アカウントページの API トークンタブ

17.3 PythonAnywhere でサイトを設定する

ロゴをクリックしてメインの PythonAnywhere Dashboard に戻り、「Bash」コンソールを起動するボタンをクリックします。これは PythonAnywhere バージョンのコマンドラインで、ちょうどあなたのコンピューターのコマンドラインと同じようなものです。

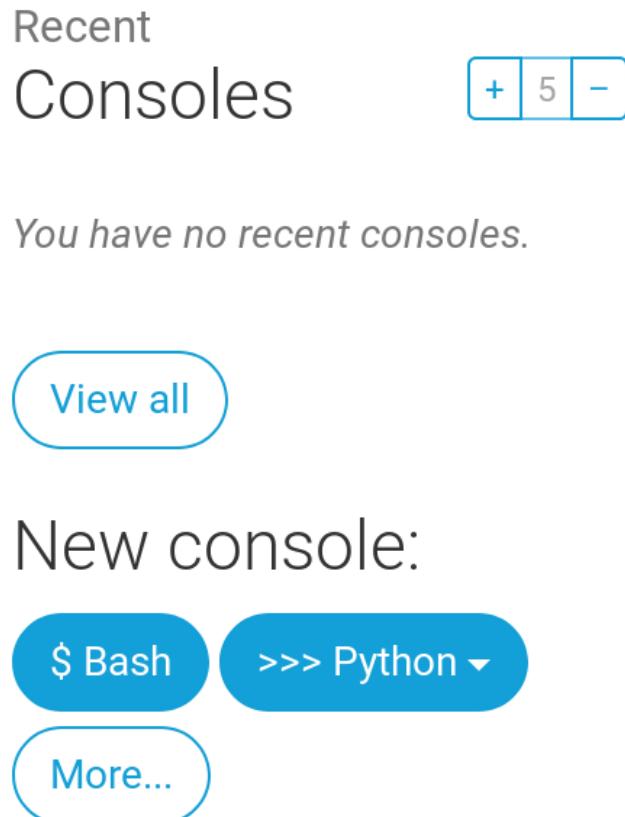


図 17.3: PythonAnywhere のウェブインターフェースの「New Console」、「bash」ボタン

備考：PythonAnywhere は Linux ベースなので、Windows を使っている場合は、コンソールがあなたのものと少し違って見えるでしょう。

PythonAnywhere に Web アプリケーションをデプロイするには、コードを GitHub からプルし、PythonAnywhere がそれを認識して Web アプリケーショ

第17章 PythonAnywhereでプロジェクトを始める

PythonAnywhereでサイトを設定する

ンのサーバを動かし始めるように設定する必要があります。それを手動で行う方法もありますが、PythonAnywhereはそれをすべて行うヘルパツールを提供しています。まず、インストールしてみましょう。

```
{% filename %}PythonAnywhere command-line{% endfilename %}
```

```
$ pip3.6 install --user pythonanywhere
```

Collecting pythonanywhere のようなメッセージがいくつか出力され、最終的に Successfully installed (...) pythonanywhere- (...) という行で終わると思います。

GitHubからアプリを自動的に構成するためのヘルパーを実行します。PythonAnywhereのコンソールに次のように入力します（GitHubからクローリしたときのURLと一致するように、<your-github-username>の代わりにGitHubユーザー名を使用することを忘れないでください）：

```
{% filename %}PythonAnywhere command-line{% endfilename %}
```

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

実行しているところを見れば、何をしているのかわかるでしょう。

- GitHubからコードをダウンロードする
- PythonAnywhere上にvirtualenvを作成する。ちょうどあなたのPC上のように
- 一部のデプロイメント設定で設定ファイルを更新する
- manage.py migrateコマンドを使ってPythonAnywhere上のデータベースをセットアップする
- 静的ファイルの設定（これについては後で学習します）
- APIを通じてPythonAnywhereがあなたのWebアプリケーションを提供するように設定する

PythonAnywhereではこれらすべてのステップは自動化されていますが、他のサーバープロバイダーでは同じ手順を経なければなりません。今注目すべき重要な点は、PythonAnywhere上のデータベースが、自分のPC上のデータベー

スとはまったく別物であることです。つまり、異なる投稿と管理者アカウントを持つことができます。

その結果、自分のコンピュータで行ったように、`createsuperuser`で管理者アカウントを初期化する必要があります。PythonAnywhereがあなたのためにあなたの `virtualenv` を自動的に起動したので、あなたがする必要があるのは以下の通りです：

```
{% filename %}PythonAnywhere command-line{% endfilename %}
```

```
(ola.pythonanywhere.com) $ python manage.py createsuperuser
```

管理者の詳細を入力します。PythonAnywhere上のパスワードをより安全にしたい場合を除き、混乱を避けるために自分のコンピュータで使用しているのと同じものを使用することをお勧めします。

PythonAnywhereのコードを`ls`を使って見てみることもできます：

```
{% filename %}PythonAnywhere command-line{% endfilename %}
```

```
(ola.pythonanywhere.com) $ ls
blog db.sqlite3 manage.py mysite requirements.txt static
(ola.pythonanywhere.com) $ ls blog/
__init__.py __pycache__ admin.py forms.py migrations models.py static
templates tests.py urls.py views.py
```

また、「ファイル」ページに移動し、PythonAnywhereに組み込まれているファイルブラウザを使用して閲覧することもできます。(Consoleページから他のPythonAnywhereページには右上のメニューボタンからいけます。一度いずれかのページに移動したら、他ページへのリンクはトップのあたりにあります。)

17.4 動いています！

あなたのサイトは現在、インターネット上で動作しているはずです！PythonAnywhereの「Web」ページをクリックしてリンクを取得します。あなたはあなたが望む誰とでもこれを共有することができます：)

注 これは初心者向けのチュートリアルです。このサイトをデプロイする

際にはセキュリティの観点からは理想的ではない、いくつかのショートカットをしました。もしこのプロジェクトを利用すると決めたり、新しいプロジェクトを開始する場合は、あなたのサイトを保護するいくつかのヒントのために、[Django デプロイメントチェックリスト](#)を見直してみてください。

17.5 デバッグのヒント

`pa_autoconfigure_django.py` スクリプトの実行中にエラーが表示された場合は、次のような原因が考えられます。

- PythonAnywhere API トークンの作成を忘れている
- あなたの GitHub の URL を間違えている
- **Could not find your settings.py** というエラーが表示された場合は、おそらく Git にすべてのファイルを追加できていなかったか、GitHub にうまくプッシュできていなかった。この場合は Git セクションをもう一度見てください

サイトにアクセスしようとするとエラーが表示された場合、最初にデバッグ情報を探す場所はエラーログです。PythonAnywhere の [Web ページ](#)には、このリンクがあります。そこにエラーメッセージがあるかどうかを確認してください。最新のものは一番下にあります。

[PythonAnywhere ヘルプサイトの一般的なデバッグのヒント](#)もあります。

つまづいた時は、コーチに助けを求めましょう。

第 18 章

あなたのサイトをチェック！

サイトのデフォルトページでは、ローカルコンピュータと同じように「It worked！」と表示されます。 URL の最後に /admin/ を追加すると、管理サイトに移動します。 ユーザー名とパスワードでログインすると、新しい投稿をサーバーに追加することができます。

いくつかの投稿を作成したら、ローカル環境（PythonAnywhere ではなく）に戻ることができます。 ここから、変更を加えるためにはあなたのローカル環境で作業する必要があります。 これが Web 開発の一般的なワークフローです。 ローカルで変更し、それらの変更を GitHub にプッシュし、それからその変更を公開している Web サーバーにプルしてきます。 これにより、公開している Web サイトを壊すことなく作業したり試したりできます。 とってもクールでしょ？

自分をすっごく褒めてあげてください！ サーバーのデプロイは Web 開発の最も難しい部分の 1 つで、ちゃんと動くようになるまで数日かかることもあります。 しかし、あなたは実際のインターネット上で、あなたのサイトを動かす事ができました！

第 19 章

Django URL

もうすぐ最初の Web ページ、あなたのブログのホームページを作るところです！ でも最初に、ちょっとだけ Django の URL について学びましょう。

19.1 URL とは？

URL は 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。）

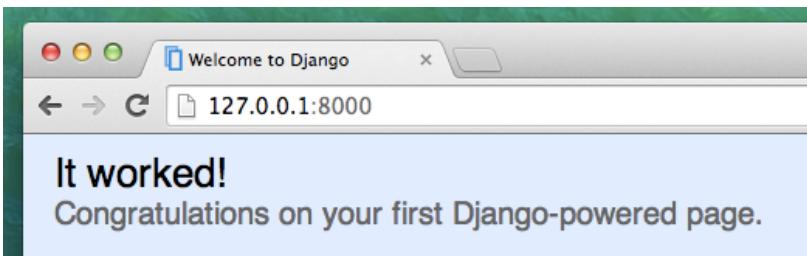


図 19.1: URL

インターネット上のすべてのページには、独自の URL が必要です。 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 URLconf (URL 設定) と呼ばれるものを使います。 URLconf はパターンの集まりで、適切なビューを見つけるために、Django がリクエストされた URL と照合するもの

です。

19.2 Django で URL はどのように機能する？

mysite/urls.py を開いて、中身をみてみると：

```
{% filename %}mysite/urls.py{% endfilename %}
```

```
"""mysite URL Configuration

[...]
"""

from django.urls import path, include
from django.contrib import admin

urlpatterns = [
    path('admin/', 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。

三重クオート（"、や """）で囲まれた行は、docstring とよばれるコメント行です。ファイル、クラス、またはメソッドの先頭に記述して、それが何をするかを説明するのに用います。これは Python によって実行されない行です。

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。

```
{% filename %}mysite/urls.py{% endfilename %}
```

```
path('admin/', admin.site.urls),
```

admin/ で始まる全ての URL について、Django が返すべきビュー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し。

19.3 あなたの初めて Django URL!

さあ最初の URL を作りましょう！'http://127.0.0.1:8000/' をブログの入口ページにして、投稿したブログポストのリストを表示するようにしたいと思います。

`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。

まず、`blog.urls` をインポートする行を追加しましょう。また、ここで `include` 関数を使うので、最初の行を変更し、そのインポートも追加する必要があります。

`mysite/urls.py` ファイルはこのようになります：

```
{% filename %}mysite/urls.py{% endfilename %}
```

```
from django.urls import path, include
from django.contrib import admin

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', 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` へ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。

19.4 blog の URL

`blog` ディレクトリの下に、新しく `urls.py` という空のファイルを作って、コードエディタで開いて下さい。そして最初の 2 行を以下のように書きます：

```
{% filename %}blog/urls.py{% endfilename %}
```

```
from django.urls import path
from . import views
```

これは Django の `path` 関数と、`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。

```
{% filename %}blog/urls.py{% endfilename %}
```

```
urlpatterns = [
    path('', views.post_list, name='post_list'),
]
```

見てのとおり、`post_list` という名前の ビュー をルート URL に割り当てています。この URL パターンは空の文字列に一致し、Django はビューを見つけるとき、URL のフルパスの前半にくっつくドメイン名（つまり、`http://127.0.0.1:8000/` の部分）を無視します。このパターンは誰かがあなたの Web サイトの '`http://127.0.0.1:8000/`' というアドレスにアクセスしてから `views.post_list` が正しい行き先だということを Django に伝えます。

最後の `name='post_list'` は、ビューを識別するために使われる URL の名前です。これはビューと同じ名前にすることもできますが、全然別の名前にすることもできます。プロジェクトでは名前づけされた URL を後で使うことになるので、アプリのそれぞれの URL に名前をつけておくのは重要です。また、URL の名前はユニークで覚えやすいものにしておきましょう。

もし今 `http://127.0.0.1:8000/` にアクセスしたら、'web page not available' のようなメッセージが出るでしょう。これはサーバー (`runserver` ってタイプしたのを覚えてますか？) が動いていないからです。なぜこうなったのかを知るためにサーバーのコンソール画面を見てみましょう。

```
    return _bootstrap._gcd_import(name[level:], package, level)
File "<frozen importlib._bootstrap>", line 2254, in _gcd_import
File "<frozen importlib._bootstrap>", line 2237, in _find_and_load
File "<frozen importlib._bootstrap>", line 2226, in _find_and_load_unlocked
File "<frozen importlib._bootstrap>", line 1200, in _load_unlocked
File "<frozen importlib._bootstrap>", line 1129, in _exec
File "<frozen importlib._bootstrap>", line 1471, in exec_module
File "<frozen importlib._bootstrap>", line 321, in _call_with_frames_removed
File "/Users/dana/Dana-Files/Codes/djangogirls/blog/urls.py", line 5, in <module>
    url(r'^$', views.post_list, name='post_list'),
AttributeError: 'module' object has no attribute 'post_list'
```

図 19.2: エラー

エラーが表示されていますね。でも心配しないで。これはむしろ、結構便利なものなんですよ：ここでは、'post_list' という属性 (attribute) がないことを知らせてくれています。これは ビュー の名前で、Django が探しで使おうとしましたが、私たちはこれをまだ作っていませんでした。現時点では、/admin/ も動作していないと思います。心配しなくて大丈夫です。ちゃんとできますから。別のエラーメッセージが表示された場合は、Web サーバーを再起動してみてください。Web サーバーが動いているコンソール画面で、Ctrl+C (Control と C のキーを同時に) を押してサーバーを止め、`python manage.py runserver` コマンドを実行して再起動します。

Django URLconfについてもっと知りたい場合は、公式のドキュメントを見て下さい：<https://docs.djangoproject.com/ja/2.0/topics/http/urls/>

第 20 章

Django ビュー - 今こそ作り ましょう！

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書いていくところです。ビューは、以前あなたが作ったモデルに情報を要求し、それをテンプレートに渡します。テンプレートは、次の章で作ります。ビューはただのPythonの関数です。**Python入門**の章で書いたもの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。私たちの場合 blog/views.pyに書くことになります。

20.1 blog/views.py

では、早速 blog/views.py をコードエディタで開いてみましょう：

```
{% filename %}blog/views.py{% endfilename %}
```

```
from django.shortcuts import render

# Create your views here.
```

まだ何もないですね。

#で始まる行は、コメントです。この行に書いたものはPythonは無視します。
それでは、次のようなちょっとしたビューを作ってみましょう。

```
{% filename %}blog/views.py{% endfilename %}
```

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

みてのとおり、`post_list` という関数 (`def` から始まる部分のことです) を作りました。これは `request` を引数に取り、`blog/post_list.html` テンプレートを表示する（組み立てる）`render` 関数を `return` 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。

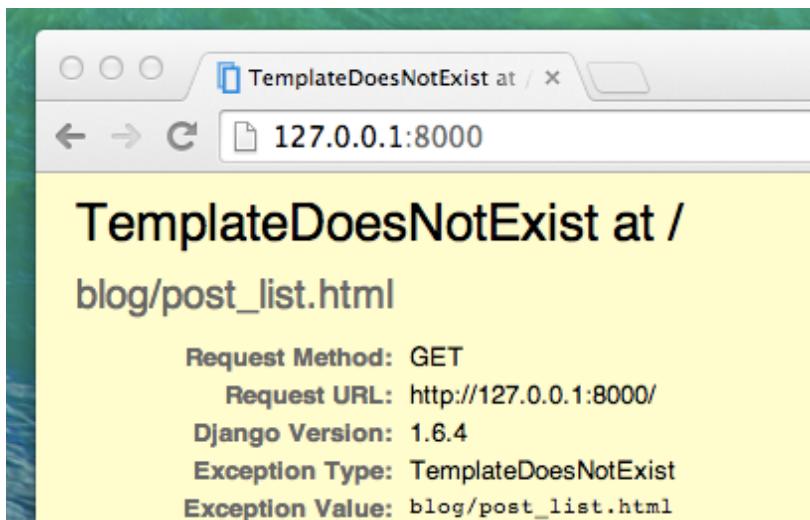


図 20.1: エラー

サーバーは実行されていることはわかるのですが、正しく表示されないのはなぜでしょう？ 心配しないで！ ただのエラーページです！ コンソールでのエラーメッセージと同じように、これは実際にかなり便利です。`TemplateDoesNotExist` と書いてありますね。 それでは次の章でテンプレートを作つて、エラーを解決しましょう！

Django のビューについてもっと知りたいのなら、オフィシャルドキュメントを是非読んでみてください。
<https://docs.djangoproject.com/ja/2.0/topics/http/views/>

第 21 章

HTML 入門

テンプレートとは何でしょうか？

テンプレートは、異なる情報を統一された形式で示すために繰り返し使われるファイルです。例えば、テンプレートは手紙を書く際に役立ちます。それぞれの手紙のメッセージは様々で、宛先も別々かもしれません、どの手紙も同じフォーマットを共有できるのです。

Django のテンプレートのフォーマットは、HTML と呼ばれる言語（最初のチャプター「インターネットのしくみ」で触れた HTML のことです）で書かれています。

21.1 HTML とは？

HTML は、Chrome や Firefox、Safari などのウェブブラウザで解読され、利用者にウェブページを表示するためのコードです。

HTML は、「HyperText Markup Language」の頭文字を取ったものです。**HyperText** は Web ページ間を結びつけるハイパーリンクをサポートするテキスト形式という意味です。**Markup** とは、一つの書類について、コードで修飾を付けて、何かに（この場合、ブラウザに）どう解釈するかを伝えることを意味します。HTML コードは、<で始まり、>で終わるタグで構成されています。これらのタグが、markup 修飾の要素なのです。

21.2 最初のテンプレート

テンプレートを作ることは、テンプレートのファイルを作ることです。すべてはファイルですよね。皆さんは、たぶん、このことに、もう気づいていると思い

ます。

テンプレートは、`blog/templates/blog` ディレクトリに保存されています。それでは、最初に、自分の `blog` ディレクトリの中に `templates` という名前のディレクトリを作成してください。次に、自分の `templates` ディレクトリの中に `blog` という名前のディレクトリを作ります。

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

(なぜ、両方とも `blog` という名前の付いたディレクトリを 2 つ作成する必要があるのか不思議に思う人もいるかもしれません。あとで分かると思いますが、簡単に言うと、これは、もっと複雑なことをやろうとした時に、それが楽にできるようにしてくれる便利な命名法なのです。)

それでは、`blog/templates/blog` ディレクトリの中に、`post_list.html` ファイル (とりあえず何も書かれていないファイルにしておきます) を作成しましょう。

あなたのウェブサイトを見てみてください: `http://127.0.0.1:8000/`

もし、`TemplateDoesNotExist` が引き続き表示されるようなら、自分のサーバーを再起動してみてください。コマンドラインから、`Ctrl+C`(Control と C のキーを同時に) を押してサーバーを止め、`python manage.py runserver` コマンドを動かして再度サーバーを動かします。

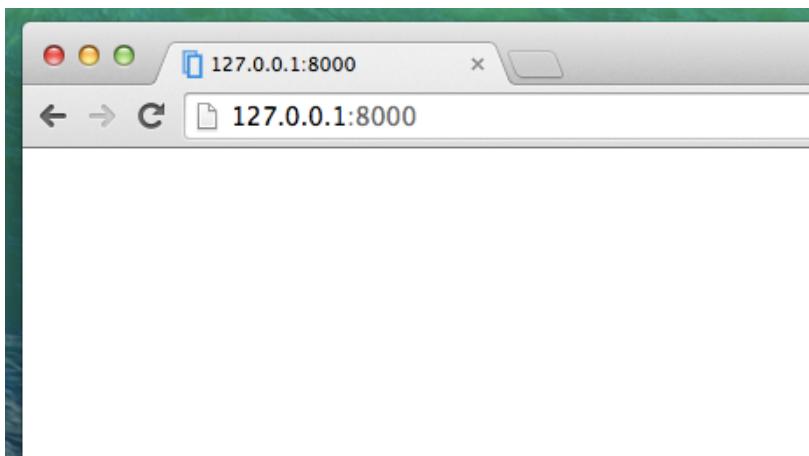


図 21.1: 図 11.1

もうエラーはありませんか！おめでとうございます:)。しかし、あなたのウェブサイトは実際には空白のページ以外は何も表示していないでしょう。テンプレートも空白だからです。それを直していく必要があります。

この新しく作ったファイルをコードエディタで開いて、次の内容を書き加えます。

```
{% filename %}blog/templates/blog/post_list.html{% endfilename %}
```

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

さあ、あなたのウェブサイトはどう見えるでしょうか？以下を開いて確認してみましょう：<http://127.0.0.1:8000/>

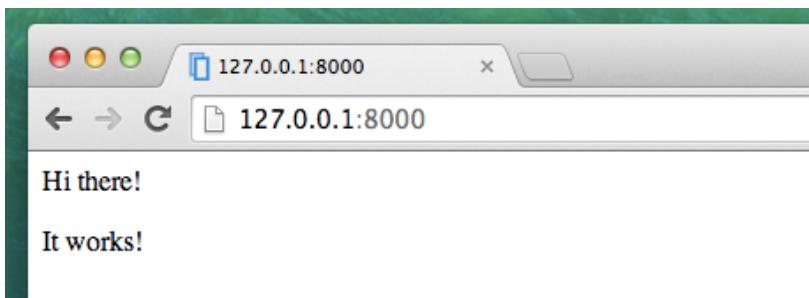


図 21.2: 図 11.2

うまく動いています！ よくできました：）

- どんなウェブページでも、最も基本的なタグである<html>から始まり、そして常に、</html>で終わります。みなさん見てとれるように、ウェブサイトの全てのコンテンツは、開始タグの<html>と閉じタグ</html>の間にあります。
- <p>は、段落要素のためのタグです；</p> でそれぞれの段落を閉じます。

21.3 Head と Body

それぞれの HTML ページは **head** と **body** という要素によって 2 つにわけられています。

- head** は文書についての情報を含む要素で、画面には表示されません。
- body** は Web ページの一部として表示されるすべてを含む要素です。

<head>でページの設定をブラウザに伝え、<body>でページの内容を伝えます。例えば、ウェブページのタイトル要素は<head>の中に書きます。こんな感じですね。

```
{% filename %}blog/templates/blog/post_list.html{% endfilename %}
```

```
<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、ページを更新してください。

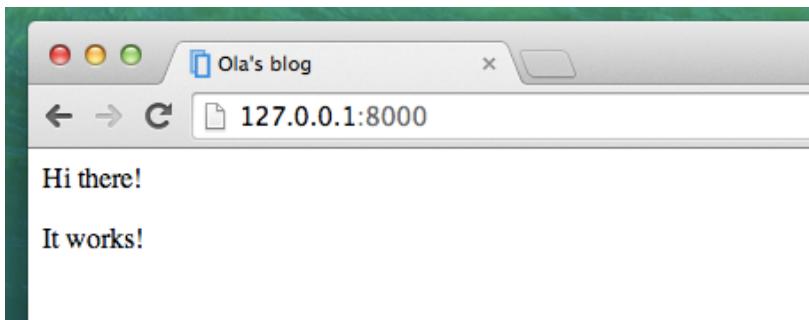


図 21.3: 図 11.3

ブラウザは、どうやって、"Ola's blog"があなたのウェブサイトのタイトルだと理解したのか分かりましたか？ ブラウザは、`<title>Ola's blog</title>`の意味を解釈して、あなたのブラウザのタイトルバーに文を配置したのです（この文はブックマークなどにも利用されます）。

各々の開始タグは閉じタグとセットで、閉じタグには/が付いていて、タグの要素は入れ子になっていることに気がついたことと思います（つまり、ある特定のタグを閉じるには、その中にある全てのタグも閉じられていないとだめなのです）。

箱の中に入れるのと同じですね。大きな箱、`<html></html>`があります；その中に`<body></body>`があり、さらにもっと小さな`<p></p>`が入っています。

こうした閉じタグと、入れ子のルールを守らなくてはいけません - そうしないと、ブラウザはタグを適切に解釈することができず、あなたのウェブページが正しく表示されなくなるのです。

21.4 テンプレートのカスタマイズ

それでは、ちょっと楽しくテンプレートを作り変えてみましょう！ 次のようにいくつか便利なタグがあります。

- <h1>ヘッダー</h1> 最も重要性の高い見出し
- <h2>サブのヘッダー</h2> その次のレベルの見出し
- <h3>サブのサブのヘッダー</h3>… など<h6>まで
- <p>文章の段落</p>
- 文章で文章を強調する
- 文章でさらに文章を強調する
-
は改行 (br タグの中には何も書いてはいけません。閉じタグも無しです)
- リンク はリンクを生成します
- 第 1 の項目第 2 の項目 でリストを作成する、こんな感じに！
- <div></div>はページ内のセクションを定義

いろんな要素をまとめたテンプレートの例がこれです。コピーして blog/templates/blog/post_list.html に貼り付けてみましょう:

```
{% filename %}blog/templates/blog/post_list.html{% endfilename %}
```

```
<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. こんにちは！ よろしくお願ひします！
    </p>
    </div>
  </body>
</html>
```

```
<div>
    <p>公開日: 2014/06/14, 12:14</p>
    <h2><a href="">2 番目の投稿</a></h2>
    <p> こんにちは！ よろしくお願ひします！ </p>
</div>
</body>
</html>
```

ここで 3 つの div セクションを作成しました。

- 最初の div 要素には、私たちのブログのタイトルが含まれています。見出しとリンクです。
- その他の 2 つの div 要素には、このブログにポストされた記事が公開日とともに記載されています。h2 はクリック可能な記事のタイトルです。2 つの p(段落) は、1 つが日付で、1 つがブログにポストされた記事です。

その結果、次のような結果が得られます。



図 21.4: 図 11.4

や～っ。 ところが今のところ、私たちのテンプレートは、常に完全に同じ情報だけしか表示できません。一方で、以前は、テンプレートを使えば、異なる情報を同じ形式で表示できるようになるとお伝えしていたのですが。

本当にやりたいことは、Django の admin に追加された本物の記事を表示することです。そして、それが次にやるべきことなのです。

21.5 もう一つ: デプロイしましょう!

ここまでやったことを公開して、インターネットで動かしてみると楽しいですよね。PythonAnywhere でデプロイしてみましょう。

Github に自分のコードを Push してみよう

まずは、最後に実行したときから、どのファイルを変更したか見てみましょう(以下のコマンドを、PythonAnywhere ではなく、自分のパソコンで実行してください。)。

{% filename %}command-line{% endfilename %}

```
$ git status
```

djangogirls ディレクトリにいることを確認して、git に対してこのディレクトリ内の変更を全て反映させるよう指示してください:

{% filename %}command-line{% endfilename %}

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

ノート --all をつけると、git は、ファイルを削除したかどうかも判定します（これが初期設定の状態では、新しいファイルと変更されたファイルしか認識しません）。. が、今いるディレクトリを表すということも思い出してくださいね（第3章にありました）。

全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するのかチェックしておきましょう（git がアップロードする全ファイルは緑で表示されます）：

{% filename %}command-line{% endfilename %}

```
$ git status
```

ほぼ完了です。どこを変更したかを履歴に保存するよう指示してみましょう。何を変更したのか説明するコミットメッセージを残しましょう。この時、どんな内容を好みで打ち込んでも構いませんが、何をしたかを具体的に書き込んでおけば、将来、作業内容を思い出す助けになるでしょう。

{% filename %}command-line{% endfilename %}

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

ノート コミットメッセージは二重クオート記号で囲みましょう。

ここまで終われば、GitHub に変更部分をアップロード (push) しましょう。

```
{% filename %}command-line{% endfilename %}
```

```
$ git push
```

新しいコードを **PythonAnywhere** に **pull** して、自分のウェブアプリを再度実行させる

- **PythonAnywhere のコンソールページ**を開き、**Bash** コンソールに移動してください（または新たな Bash コンソールを開始してください）。それから次を実行してください。

```
{% filename %}PythonAnywhere command-line{% endfilename %}
```

```
$ cd ~/<your-pythonanywhere-username>.pythonanywhere.com  
$ git pull  
[...]
```

(<your-pythonanywhere-username>の部分を、自分の実際の PythonAnywhere のユーザー名に角カッコをはずして置き換えることを忘れずに)

そのうえで、自分のコードがダウンロードされたことを確認しましょう。それをチェックしたい場合は、「**Files**」ページを押して、自分のコードを PythonAnywhere の中で見てみましょう（コンソールページのメニューボタンから他の PythonAnywhere ページにアクセスできます）。

- 最後に、「**Web**」ページを押して、自分のアプリの **Reload** を押します。

あなたのアプリが更新され、動いています！ ウェブサイトを開いて、再読み込んでみましょう。どう変わったのか、見えるはずです。:)

第 22 章

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、データの格納について学びます。やってみましょう！

22.1 クエリセットとは？

クエリセットが何かと言うと、モデルのオブジェクトのリストのことです。クエリセットを使って、データベースからデータを読み込んだり、抽出したり、並べ替えたりできます。

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。

22.2 Django shell

コンソール画面を開いて（PythonAnywhere のコンソールではないですよ）、次のコマンドを入力してみましょう。

```
{% filename %}command-line{% endfilename %}
```

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

次のような表示に切り替わるでしょう。

```
{% filename %}command-line{% endfilename %}
```

```
(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。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。

すべてのオブジェクト

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。

```
{% filename %}command-line{% endfilename %}
```

```
>>> 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。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。

```
{% filename %}command-line{% endfilename %}
```

```
>>> 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。

```
{% filename %}command-line{% endfilename %}
```

```
>>> Post.objects.all()
<QuerySet [<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。

オブジェクトの作成

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。

```
{% filename %}command-line{% endfilename %}
```

```
>>> 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。

```
{% filename %}command-line{% endfilename %}
```

```
>>> 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てましたっけ？覗いてみましょうか。

```
{% filename %}command-line{% endfilename %}
```

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

作成しておいたスーパーユーザがいますね。このユーザを取り出してみましょう（この行をご自分で作ったスーパーユーザのユーザ名に変更して取り出してください）

ださいね)。

```
{% filename %}command-line{% endfilename %}
```

```
>>> 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！

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。

```
{% filename %}command-line{% endfilename %}
```

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

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。

```
{% filename %}command-line{% endfilename %}
```

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

出来ていますね！ リストにはポストが 1 つ以上あります。

さらに投稿を追加しましょう

楽しくなってきたでしょう？ 理解を深めるためにもう少しポストを作ってお
きましょう。2~3 個記事を追加したら、次に進みましょう。

オブジェクトの抽出

クエリセットの大部分は、抽出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。 取り出されるブログポストが満たす条件を、カッコ ()

の中に指定します。今回の例では、`author` が `me` と等しいという条件です。Django での表し方は、`author=me` となります。このようになりますね。

```
{% filename %}command-line{% endfilename %}
```

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

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。

```
{% filename %}command-line{% endfilename %}
```

```
>>> Post.objects.filter(title__contains='title')
<QuerySet [<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、`titlecontains` 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 `titlecontains`")

また、公開済みの全ポストを取り出すことも出来ます。それには、`published_date` が現在以前の全ポストを取り出します。

```
{% filename %}command-line{% endfilename %}
```

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

そうでした、残念なことに、コンソールから追加したポストはまだ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。

{% filename %}command-line{% endfilename %}

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

そして、`publish` メソッドを呼び出します。

{% filename %}command-line{% endfilename %}

```
>>> 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。(上方キーを3回押せば、さっ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)

{% filename %}command-line{% endfilename %}

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

オブジェクトの並び替え

クエリセットは、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。

{% filename %}command-line{% endfilename %}

```
>>> Post.objects.order_by('created_date')
<QuerySet [<Post: Sample title>, <Post: Post number 2>, <Post: My 3rd post!>, <Post: 4th t
```

逆順、つまり新しく追加した順に並べ替えることも出来ます。それには、`-`(ハイフン) を使います。

{% filename %}command-line{% endfilename %}

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

クエリセットをつなげる

QuerySets をつなげて組み合わせることも出来ます。

```
>>> Post.objects.filter(published_date__lte=timezone.now()).order_by('published_date')
<QuerySet [<Post: Post number 2>, <Post: My 3rd post!>, <Post: 4th title of post>, <Post:
```

クエリセットをつなげるのは本当に強力です。つなげていくことでとても複雑なクエリも書くことが出来ます。

いいですね！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。

```
{% filename %}command-line{% endfilename %}
```

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


第 23 章

テンプレート内の動的データ

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れば良さそうですね。

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。 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を、 *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 をコードエディタで開きます。 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。

```
{% filename %}blog/views.py{% endfilename %}
```

```
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、覚えてていますか？ 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。

```
{% filename %}blog/views.py{% endfilename %}
```

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

`models` の前にあるドットは 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。だから、こんな風に、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。(ファイル名の拡張子`.py` は必要ないです) 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。

23.1 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。?? チャプターで勉強しました。

公開したブログ記事を `published_date` で並べ替えをしたいですね。これも、クエリセットの章でやったので、大丈夫ですね？

```
{% filename %}blog/views.py{% endfilename %}
```

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

それでは `blog/views.py` をコードエディタで開いて、このコードを `def post_list(request)` 関数の中に加えましょう。`from django.utils import timezone` を最初に追加するのを忘れないでくださいね。

```
{% filename %}blog/views.py{% endfilename %}
```

```
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', {})
```

最後に残っているのは、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作業です。テンプレートでの表示については次の章でやりましょう。

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。

`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。`: {'posts': posts}`という具合に、記述します。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。

```
{% filename %}blog/views.py{% endfilename %}
```

```
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こちらも読んでみてくださいね。 <https://docs.djangoproject.com/ja/2.0/ref/models/querysets/>

第 24 章

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。

24.1 テンプレートタグとは？

HTML 中で本当は Python のコードを書くことはできません。なぜならブラウザが理解できないからです。ブラウザは HTML だけ分かれます。HTML はどちらかというと静的で、それに対して Python はもっとずっと動的なことを私たちには知っています。

Django テンプレートタグ は HTML に Pyhton 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！

24.2 ブロガー覧テンプレートの表示

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ためには、次のように変数の名前を二重中括弧で括ります。

```
{% filename %}blog/templates/blog/post_list.html{% endfilename %}
```

```
 {{ posts }} 
```

これを `blog/templates/blog/post_list.html` テンプレートでやってみま

しょう。エディタでこのファイルを開き、2つめと3つめの `<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：



図 24.1: 図 13.1

見たとおり、このようになります。

```
{% filename %}blog/templates/blog/post_list.html{% endfilename %}
```

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

Django は `posts` 変数をオブジェクトのリストと認識します。Python 入門でどうやってリストを表示できたか覚えていませんか？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：

```
{% filename %}blog/templates/blog/post_list.html{% endfilename %}
```

```
{% 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。



図 24.2: 図 13.2

動きましたね。しかし、本当は **HTML** 入門で作った静的な記事のように表示してほしいところです。そこで、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：

```
{% filename %}blog/templates/blog/post_list.html{% endfilename %}
```

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

`{% raw %}{% for %}` と `{% endfor %}` の間に書いたものはリスト中の各オブジェクトの分だけ繰り返されます。ページをリロードしてみましょう。`{% endraw %}`

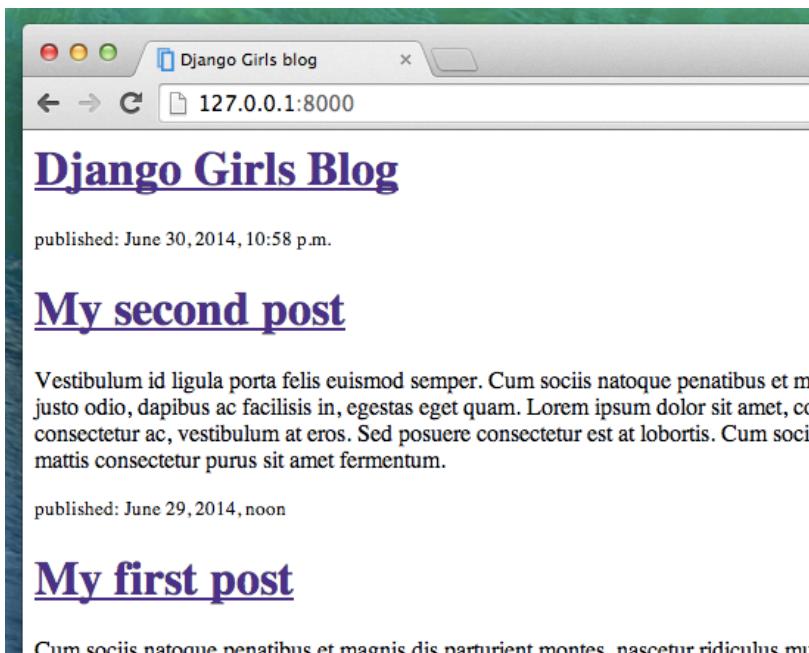


図 24.3: 図 13.3

post 変数がさっさと違って、{{ 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。|linebreaksbr 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。

24.3 もう一つ...

今の時点でのウェブサイトを公開して見てみませんか？もう一度 PythonAnywhere でデプロイしてみましょう。デプロイのステップをおさらいします。

- まず、Github にあなたのコードを Push しましょう

```
{% filename %}command-line{% endfilename %}
```

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

- そしたら、PythonAnywhere に戻って、Bash コンソール（か、新しいコンソール）に入って、次のようにコマンドを打ちましょう：

```
{% filename %}PythonAnywhere command-line{% endfilename %}
```

```
$ cd $USER.pythonanywhere.com  
$ git pull  
[...]
```

- 最後に「Web」ページを開いてアプリをリロードします。（コンソールから他の PythonAnywhere ページにアクセスするには、右上のメニュー ボタンを使います。）更新された内容が <https://yourname.pythonanywhere.com> に公開されているはずです。ブラウザで確認しましょう！ PythonAnywhere サイトで表示されるブログの記事が、あなたのパソコンの中のローカルサーバーのものと違っていても大丈夫です。ローカルコンピュータにあるデータベースと、PythonAnywhere 上のデータベースは同期しています。

おめでとうございます！ よくできました！ 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！）。PythonAnywhere サイトの Django 管理者であることを確認してください (<https://yourname.pythonanywhere.com/admin>)。それから、投稿したもの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：）

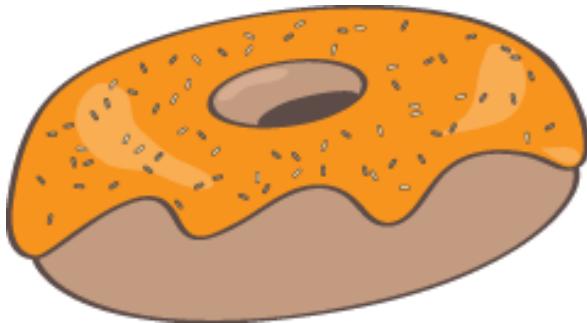


図 24.4: 図 13.4

第 25 章

CSS でカワイくしよう！

ブログは作ったものの、まだなんかダサイですよね。かわいくしましょう！
そのためには CSS を使います。

25.1 CSS とは？

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 ページをメ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。

25.2 Bootstrap を使いましょう！

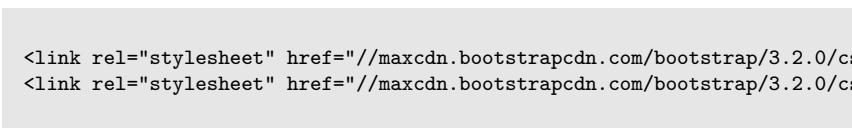
Bootstrap は美しい Web サイトを開発するための HTML と CSS のフレームワークとしてとても有名です: <http://getbootstrap.com/>

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。

25.3 Bootstrap のインストール

Bootstrap をインストールするため、.html ファイル (blog/templates/blog/post_list.html) をコードエディタで開き <head> の中にこれを追加しましょう：

```
{% filename %}blog/templates/blog/post_list.html{% endfilename %}
```



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。



図 25.1: 図 14.1

これだけでずいぶん見た目が良くなりましたね！

25.4 Django の静的ファイル

最後に、静的ファイルと呼ばれるものを詳しく見てていきましょう。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。

静的ファイルはプロジェクトのどこに置けばいいの？

Django は、ビルトインの "admin" アプリにより、静的ファイルをどこで探せばいいのかわかっています。私たちがやることは、blog アプリのための静的ファイルを追加することだけです。

そのために、blog アプリの中に static というフォルダを作ります。

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

Django は、全てのアプリのフォルダ内の "static" と名づけられた全てのフォルダを自動的に探して、その中身を静的ファイルとして使えるようにします。

25.5 最初の CSS ファイル !

CSS ファイルを作って、Web サイトにあなたのスタイルを設定していきましょう。 static ディレクトリの中に css というディレクトリを作成しましょう。そして、その css ディレクトリの中に blog.css という新規ファイルを作ります。準備 OK ?

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

ついに CSS を書くときが来ました！ コードエディタで blog/static/css/blog.css ファイルを開きましょう。

ここでは CSS のカスタマイズや学習方法については詳しく説明しません。さらに知りたい場合は、このページの最後に無料の CSS の学習コースを紹介して

いますので、そちらを参考にしてください。

ただ、せめて少しここでやってみましょう。 ヘッダーの色を変えてみるのもいいかもしませんね。 色を理解するために、コンピュータは特殊なコードを使います。 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。 たとえば、青色のコードは #0000FF です。 カラーコードのサンプルはこのサイト <http://www.colorpicker.com/> で確認できます。 red や green といった定義済みの色を利用することもできます。

blog/static/css/blog.css ファイルに、次のコードを追加しましょう。

```
{% filename %}blog/static/css/blog.css{% endfilename %}
```

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

h1 a は CSS セレクタと呼ばれるものです。 h1 要素の中にある a 要素にスタイルを適用します、という意味になります。 よって <h1>link</h1> となっているとき、 h1 a スタイルが適用されます。 この場合、その要素を #FCA205 に、つまりオレンジ色にしようとしています。 またはあなたの好きな色に変えられます！

CSS ファイルには、HTML ファイルの各要素のスタイルを指定していきます。 まずは要素名でもって、その要素を識別します。 HTML のタグ名は覚えがあるでしょう。 例えば a, h1, body などが要素名の例です。 また、class 属性や、id 属性によって要素を識別することができます。 class や id は、あなたが自分で要素につけることができる名前です。 class は要素のグループを定義して、id は特定の要素を指定します。 例えば、次のタグは、タグ名 a、class 名 external_link、id 名 link_to_wiki_page、どれを使っても CSS によって識別されます。

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

CSS セレクタについては [CSS Selectors in w3schools](#) を見てください。

さて、CSS を追加したことを HTML テンプレートに教えないといけません。 blog/templates/blog/post_list.html を開いて、先頭にこの行を追加しま

しよう :

```
{% filename %}blog/templates/blog/post_list.html{% endfilename %}
```

```
{% load static %}
```

これで、このテンプレートに静的ファイルを読み込むことができました^^。そして、`<head>` と `</head>` の中にある Bootstrap CSS ファイルのリンクの下に、この行を追加しましょう：

```
{% filename %}blog/templates/blog/post_list.html{% endfilename %}
```

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

ブラウザは上から書いた順番でファイルを読み込むので、記述する箇所はよく確かめる必要があります。順番が逆になると、私たちが書いたファイルが Bootstrap のファイルに上書きされてしまうかもしれません。これで、テンプレートに CSS ファイルがある場所を教えました。

ファイルは次のようになっているはずです：

```
{% filename %}blog/templates/blog/post_list.html{% endfilename %}
```

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

保存して、サイトを更新してください。

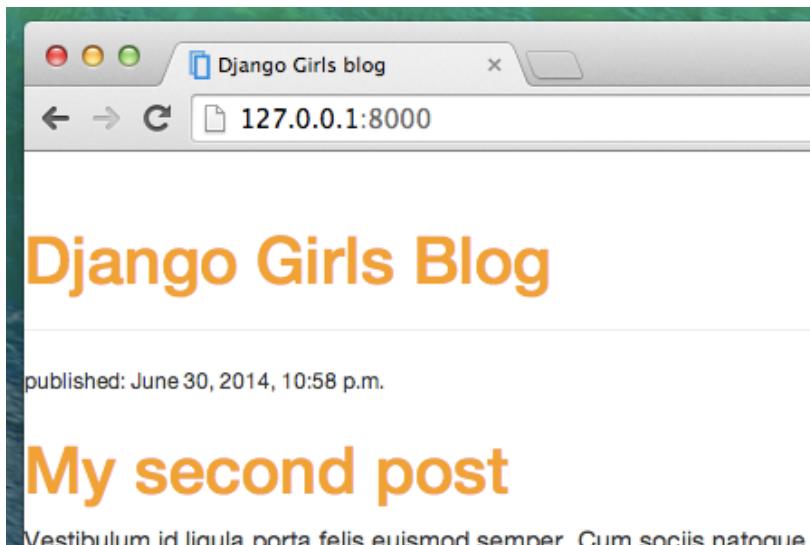


図 25.2: 図 14.2

素晴らしいですね！ あとは、左サイドの余白幅を少し広げて、余裕を持たせてあげたらもっと良くなると思いませんか？ やってみましょう！

```
{% filename %}blog/static/css/blog.css{% endfilename %}
```

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

これを CSS に追加して、保存してください。どのように変化したか、さあ見てみましょう！



図 25.3: 図 14.3

ヘッダーのフォントを変えてみませんか？ファイル `blog/templates/blog/post_list.html` の `<head>` タグの中に次の一行を貼り付けましょう。

```
{% filename %}blog/templates/blog/post_list.html{% endfilename %}
```

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

まずは、このリンクが `blog/static/css/blog.css` より前にあることを確かめましょう。この1行は **Lobster** というフォントを Google Fonts (<https://www.google.com/fonts>) から読み込むということを意味しています。

`blog/static/css/blog.css` ファイルの中の、`h1 a` という宣言ブロックを見つけてください（コードは { } で囲まれています）。そして、そのカッコの中に `font-family: 'Lobster';` と1行追加して、web サイトを更新して

みましょう：

```
{% filename %}blog/static/css/blog.css{% endfilename %}
```

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



図 25.4: 図 14.3

素晴らしいです！

前述のように、CSS はクラスの概念を持っています。それにより、HTML コードの一部に名前を指定し、他の部分に影響を与えるずにこの部分にだけスタイルを適用することができます。なんて便利なんでしょう！例えば、2つの div 要素（ヘッダーと記事など）があったとして、これらのそれぞれに違うスタイルを適用することができます。クラスを利用すると、違う見た目にできるのです。

先に進んで、HTML コードの一部に名前をつけましょう。ヘッダーを含む

div要素に、`page-header`というクラス名をつけましょう：

```
{% filename %}blog/templates/blog/post_list.html{% endfilename %}
```

```
<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。

```
{% filename %}blog/templates/blog/post_list.html{% endfilename %}
```

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

そして、さまざまな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に関する多くのチュートリアルがあり、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、`blog/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：

```
{% filename %}blog/static/css/blog.css{% endfilename %}
```

```
.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 {
    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;
}
```

では、投稿を表示しているHTMLコードをクラス宣言で囲みましょう。
blog/templates/blog/post_list.html中のこの部分を
{% filename %}blog/templates/blog/post_list.html{% endfilename %}

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

```

<p>{{ post.text|linebreaksbr }}</p>
</div>
{% endfor %}

```

これで置き換えて下さい：

```
{% filename %}blog/templates/blog/post_list.html{% endfilename %}
```

```

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

```

ファイルを保存して Web サイトを更新してみましょう。



図 25.5: 図 14.4

やったー！ ほら凄いでしょ？ 貼り付けたコードを見て、HTML にクラスを追加した場所やそのクラスが CSS の中で使われている場所を探してみましょう。日付の色をターコイズブルーにするには、どこを変更すればいいでしょうか？

CSS をいじって表示が壊れることを恐れないで、どんどん変更を加えてみてください。CSS で遊んでみることで、何がどうなっているかを理解できてきます。たとえ何かを壊してしまっても、心配しないで！ すぐに元に戻すことができます。

この章の最後に、[Codecademy HTML & CSS course](#) という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。web サイトを CSS で美しくするための多くのことが学べます。

さて、次の章にいく準備はできましたか？ ^皿^

第 26 章

テンプレートを拡張しよう

Django のまた別の素敵なところはテンプレート拡張です。これは何を意味するのでしょうか？ それは HTML の共通部分をウェブサイトの異なるページで使えるということです。

テンプレートは同じ情報やレイアウトを複数の場所で利用したいときに役立ちます。各ファイル内で繰り返す必要はありません。さらにもし何か変更したい場合、各テンプレートを変更する必要はなく、1回変更すればいいだけです！

26.1 基本テンプレートを作成する

基本テンプレートはあなたのウェブサイトの各ページを拡張するための最も基礎的なテンプレートです。

blog/templates/blog/以下に base.html ファイルを作ってみましょう。

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

それからコードエディタで開いて、以下のように post_list.html から base.html ファイルにすべてコピーしましょう。

```
{% filename %}blog/templates/blog/base.html{% endfilename %}
```

```
{% load static %}  
<html>  
    <head>  
        <title>Django Girls blog</title>  
        <link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap.min.css" type="text/css"/>  
        <link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap-theme.min.css" type="text/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' %}" type="text/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.get_absolute_url }}"/>{{ post.title }}</a></h1>  
                            <p>{{ post.text|linebreaksbr }}</p>  
                        </div>  
                    {% endfor %}  
                </div>  
            </div>  
        </div>  
    </body>  
</html>
```

それから `base.html` 内の`<body>`全体 (`<body>`と`</body>`の間のすべて) を次で置き換えます。

```
{% filename %}blog/templates/blog/base.html{% endfilename %}
```

```
<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>
</body>
```

{% raw %}{% for post in posts %} から {% endfor %} が以下のように置き換えられたことに気づいたでしょうか。 {% endraw %}

{% filename %}blog/templates/blog/base.html{% endfilename %}

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

でも何のために？ あなたはただ block を作っただけです！ {% block %} テンプレートタグを、これから HTML を挿入しようとする場所に使いました。その HTML はこのテンプレート (base.html) を拡張した別のテンプレートからやってきます。どうやって行うかはすぐに示します。

base.html を保存し、もう一度 blog/templates/blog/post_list.html をコードエディタで開きます。 {% raw %} {% for post in posts %} の上と {% endfor %} の下すべてを削除しましょう。それが終わったら以下のようになっていると思います。 {% endraw %}

{% filename %}blog/templates/blog/post_list.html{% endfilename %}

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

これを content ブロックに対応するテンプレートのパートとして使いたいです。このファイルに block タグを追加する時です！

{% raw %}追加する block タグは base.html ファイル中のタグにマッチしてほしいですよね。また、block タグには content ブロックに属するすべてのコードを含めたいですよね。そうするには、すべてを {% block content %} と {% endblock %} の間に入れてあげればよいです。このように:{% endraw %}

```
{% filename %}blog/templates/blog/post_list.html{% endfilename %}
```

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

あとやることは一つだけです。これら二つのテンプレートをくっつけてあげる必要があります。これがテンプレートを拡張するということのすべてです！ そういうするにはこのファイルの先頭に extends タグを追加します。次のように：

```
{% filename %}blog/templates/blog/post_list.html{% endfilename %}
```

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

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

以上です！ ファイルを保存して、ウェブサイトがまだちゃんと動いているか確認しましょう。:)

もし `TemplateDoesNotExist` というエラーが出ていたら、`blog/base.html` ファイルがないという意味で、コンソールで `runserver` が実行されたままになっていると思います。これを止め (Ctrl+C - Control と C のキーを同時押し)、それから `python manage.py runserver` コマンドを入力して再度サーバーを動かしてみてください。

```
{% set warning_icon = '<span class="glyphicon glyphicon-exclamation-sign" style="color: red;" aria-hidden="true" data-toggle="tooltip" title="An error is expected when you run this code!" ></span>' %}
```


第 27 章

アプリケーションを拡張しよう

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、またウェブサイトを素敵にするやり方もわかります。

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

27.1 投稿の詳細へのテンプレートリンクを作成する

blog/templates/blog/post_list.html ファイルにリンクを追加していきましょう。コードエディタで開いたら、次のようになっていますよね： {filename %}blog/templates/blog/post_list.html{endfilename %}

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

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

```
{% endblock %}
```

{% raw %}投稿リストの投稿のタイトルから投稿の詳細ページへのリンクを作りたいです。 投稿の詳細ページにリンクするように<h1>{{ post.title }}</h1>を変更しましょう。{% endraw %}

```
{% filename %}{% warningicon %} blog/templates/blog/postlist.html{% endfilename %}
```

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

{% raw %}不思議な{% url 'post_detail' pk = post.pk %}を説明します。気づいたかもしれません、{% %}という表記は Django のテンプレートタグを使用していることを意味しています。今私たちはこれを URL を作るために使います！ {% endraw %}

post_detail の部分は、Django が blog/urls.py に書かれた name=post_detail の URL を待ち受けることを表しています。

そして pk=post.pk についてはどうでしょうか？ pk はプライマリキーの略で、データベースの各レコードのユニークな名前です。Post モデルでプライマリキーを指定しなかったので、Django は私たちのために 1 つのキーを作成し（デフォルトでは、各レコードごとに 1 ずつ増える数字で、たとえば 1、2、3 です）、各投稿に pk というフィールド名で追加します。Post オブジェクトの他のフィールド（title、author など）にアクセスするのと同じ方法で、post.pk と書くことによってプライマリキーにアクセスします！

さて、私たちが <http://127.0.0.1:8000/> に行くとエラーが出ます（知っての通り、URL も post_detail のビューもまだ作っていないので）。このようになります：

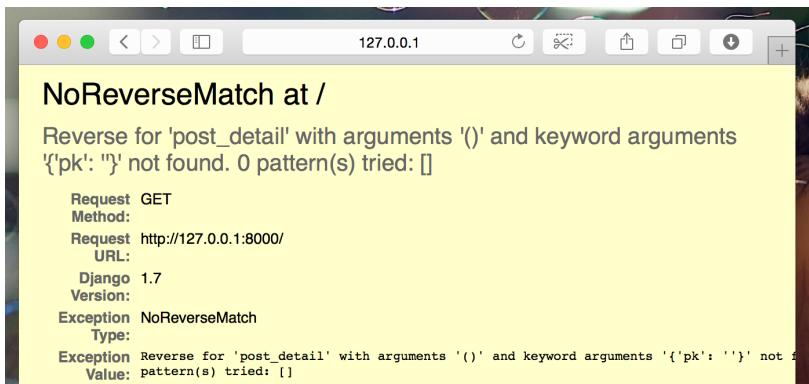


図 27.1: NoReverseMatch error

27.2 投稿の詳細への URL を作成する

post_detail ビュー用に urls.py に URL を作成しましょう！

最初の投稿の詳細がこの URL で表示されるようにします：
http://127.0.0.1:8000/post/1/

投稿の内容を表示する post_detail というビューを Django に示すように、blog/urls.py ファイルで URL を作りましょう。blog/urls.py をコードエディタで開いて、path('post/<int:pk>/', views.post_detail, name='post_detail')，という行を追加しましょう。ファイルは次のようになるでしょう。

```
{% filename %}{{ warning_icon }} blog/urls.py{% endfilename %}
```

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.post_list, name='post_list'),
    path('post/<int:pk>/', views.post_detail, name='post_detail'),
]
```

post/<int:pk>/ の部分は URL パターンを指定しています。それについて説

明しましょう：

- `post/` は URL が `post` に続けて `/` で始まることを意味します。ここまでは順調ですね。
- `<int:pk>` - この部分はトリッキーです。これは Django は整数の値を期待し、その値が `pk` という名前の変数でビューに渡されることを意味しています。
- `/` - それから URL の最後に再び `/` が必要です。

つまり、ブラウザに `http://127.0.0.1:8000/post/5/` を入力すると、Django は `post_detail` というビューを探していると理解します。そして `pk` が 5 という情報をそのビューに転送します。

よし、私たちは `blog/urls.py` に新しい URL パターンを追加しました！ ページを更新しましょう：`http://127.0.0.1:8000/` ドーン！ サーバーが再び実行を停止しました。コンソールを見てください - 予想通り、もう一つのエラーがあります！

```
return _bootstrap._gcd_import(name[level:], package, level)
File "<frozen importlib._bootstrap>", line 2231, in _gcd_import
File "<frozen importlib._bootstrap>", line 2214, in _find_and_load
File "<frozen importlib._bootstrap>", line 2203, in _find_and_load_unlocked
File "<frozen importlib._bootstrap>", line 1200, in _load_unlocked
File "<frozen importlib._bootstrap>", line 1129, in _exec
File "<frozen importlib._bootstrap>", line 1448, in exec_module
File "<frozen importlib._bootstrap>", line 321, in call_with_frames_removed
File "/home/hel/code/djangogirls/workthrough/blog/urls.py", line 6, in <module>
    url(r'^post/(?P<pk>[0-9]+)/$', views.post_detail, name='post_detail'),
AttributeError: 'module' object has no attribute 'post_detail'
```

図 27.2: AttributeError

あなたは次のステップが何であるか覚えていますか？ ビューを追加する！ ですね。

27.3 投稿の詳細ビューを追加する

今回はビューに追加のパラメータ `pk` が与えられます。私たちのビューはそれを受け取る必要がありますね？ そこで関数を `def post_detail(request, pk):` として定義します。urls で指定した名前 (`pk`) とまったく同じ名前を使用する必要があることに注意してください。この変数を省略するのは正しくない

のでエラーになってしまいます！

今、私たちは1つだけブログ投稿を取得したいと考えています。これを行うには、次のようなクエリセットが使用できます。

```
{% filename %}{% warning_icon %} blog/views.py{% endfilename %}
```

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

しかし、このコードには問題があります。与えられたプライマリキー（pk）を持つPostが存在しない場合、とてもダサいエラーが発生します。



図 27.3: DoesNotExist error

私たちはそれを望んでいません！しかし幸運にもDjangoにはそれを処理するものがあります：`get_object_or_404`です。与えられたpkのPostがない場合、前よりもっとよいPage Not Found 404ページが表示されます。



図 27.4: Page not found

いい知らせとして実際には自分の `Page not found` ページを作って自分の好きなようにきれいにすることができます。しかしそれは今すごく重要ではないので、私たちはそれをスキップします。

よし、今こそビューを `views.py` ファイルに追加するときです！

`blog/urls.py` では `views.post_detail` というビューを参照する `post_detail` という名前の URL ルールを作成しました。これは、Django が `blog/views.py` 内の `post_detail` というビュー関数を待っていることを意味します。

`blog/views.py` をコードエディタで開き、他の `from` 行の近くに次のコードを追加する必要があります。

```
{% filename %}blog/views.py{% endfilename %}
```

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

ファイルの最後にビューを追加します：

```
{% filename %}blog/views.py{% endfilename %}
```

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

ページを更新してみましょう : <http://127.0.0.1:8000/>

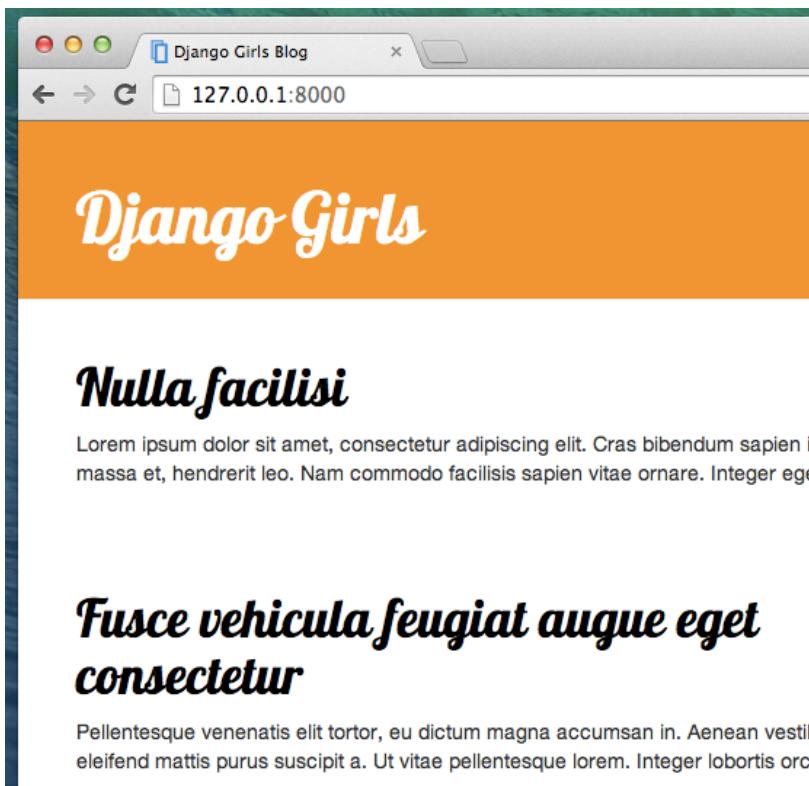


図 27.5: Post list view

出来ましたね！しかし、ブログ投稿のタイトルのリンクをクリックするとどうなりますか？

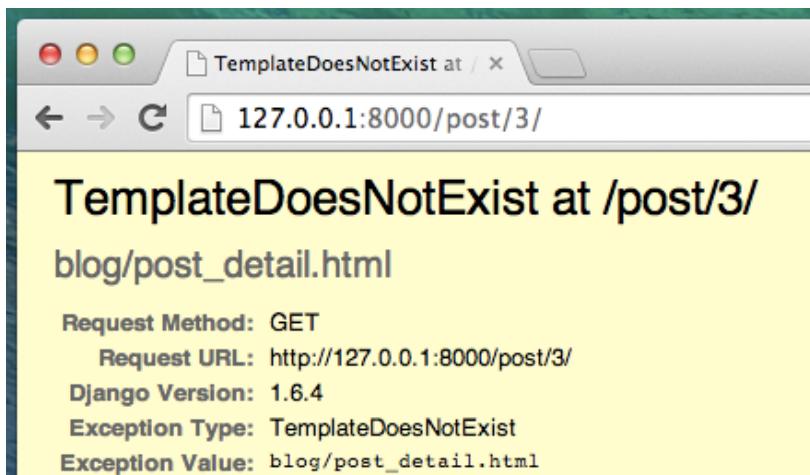


図 27.6: TemplateDoesNotExist error

あらいやだ！ 別のエラー！ しかし、私たちはすでにそれに対処する方法をすでに知っていますね。 そう！ テンプレートを追加する必要があります！

27.4 投稿の詳細へのテンプレートリンクを作成する

blog/templates/blog に post_detail.html というファイルを作成し、コードエディタで開きます。

こんな感じですね。

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

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

```
<p>{{ post.text|linebreaksbr }}</p>
</div>
{% endblock %}
```

もう一度 `base.html` を拡張します。`content` ブロックでは、投稿の公開日(存在する場合)、タイトル、およびテキストを表示します。ここで重要なポイントについて見てみます。

`{% raw %}{% if ... %} ... {% endif %}` は、何かをチェックしたいときに使用できるテンプレートタグです。(`if ... else...` を Python 入門のチャプターでやったのを覚えていますか？) この場合、私たちは投稿の公開日(`published_date`)が空でないかを確認したいです。`{% endraw %}`

これで、ページを更新して `TemplateDoesNotExist` がもうなくなったかどうか確認できます。



図 27.7: Post detail page

イエーイ！ うまくできていますね！

第 28 章

デプロイの時間です！

あなたのウェブサイトがまだ PythonAnywhere 上で動くとしたらいいでしょう？ またデプロイしてみましょう。

```
{% filename %}command-line{% endfilename %}
```

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

それから、PythonAnywhere Bash コンソールで：

```
{% filename %}command-line{% endfilename %}
```

```
$ cd ~/<your-pythonanywhere-username>.pythonanywhere.com  
$ git pull  
[...]
```

(<your-pythonanywhere-username>の部分を、自分の実際の PythonAnywhere のユーザー名に山カッコをはずして置き換えることを忘れずに)

28.1 サーバー上の静的ファイルの更新

PythonAnywhereのようなサーバは、(CSS ファイルのような)「静的ファイル」を Python ファイルとは違って扱うのが好きです。なぜなら、それらが高速に読み込まれるように最適化できるからです。その結果、CSS ファイルを変更するたびに、サーバ上で追加のコマンドを実行して、更新するように指示する必要があります。コマンドは `collectstatic` です。

もし仮想環境 (virtualenv) が有効になっていなければ有効化するところから始めましょう (PythonAnywhere ではこれを行うために `workon` というコマンドを使用します。これはあなたが自身のコンピュータで使用している `source myenv/bin/activate` コマンドと同じようなものです)。

```
{% filename %}command-line{% endfilename %}
```

```
$ workon <your-pythonanywhere-username>.pythonanywhere.com
(ola.pythonanywhere.com)$ python manage.py collectstatic
[...]
```

`manage.py collectstatic` コマンドは、`manage.py migrate` のようなものです。私たちはコードをいくつか変更してから、Django にサーバの静的ファイルのコレクションまたはデータベースに変更を適用するよう指示します。

いずれにしても、「Web」ページ（コンソールの右上のメニュー ボタンから）を開き、Reload をクリックする準備ができました。そしてそれから <https://yourname.pythonanywhere.com> ページを見て結果を確認しましょう。

うまくいってるはずです！ おめでとう :)

第 29 章

Django フォーム

私たちの Web サイトで最終的にやりたいことは、記事を追加したり編集したりするためのよい方法を作ることです。 Django admin はかなりいいですが、カスタマイズしたりかわいくいい感じにするのはちょっと大変です。 フォームによってインターフェイスを完璧にコントロールできるようになります。想像するほどんど全てのことができます！

Django フォームのよいところは、フォームをゼロから定義できたり、フォームの結果をモデルに保存できる `ModelForm` を作れたりするところです。

これはまさに私たちがやりたいことです：`Post` モデルのためのフォームを作ります。

Django の他の重要なパートと同様に、フォームは自身のファイルがあります: `forms.py`

これは `blog` ディレクトリの下に `forms.py` の名前で作る必要があります。

```
blog
└── forms.py
```

このファイルをエディタで開き、次のコードを入力してください。

```
{% filename %}blog/forms.py{% endfilename %}
```

```
from django import forms
from .models import Post
```

```
class PostForm(forms.ModelForm):  
  
    class Meta:  
        model = Post  
        fields = ('title', 'text',)
```

最初に Django の forms をインポート (`from django import forms`) し、Post モデルもインポート (`from .models import Post`) する必要があります。

`PostForm` とは何かと思うかもしれません、これはフォームの名前です。このフォームが `ModelForm` の一種だと Django に伝える必要があります (Django が私たちのためにいくつか魔法をかけられるように)。`forms.ModelForm` がその役割を果たします。

次に `class Meta` ですが、ここで Django にフォームを作るときにどのモデルを使えばいいか (`model = Post`) を伝えます。

最後にフォームのフィールドに何を置くか書きます。ここでは、私たちは `title` (タイトル) と `text` (本文) のみをフォームで使用します。`author` は現在ログインしている人 (あなた) です。`created_date` は (コードによって) 自動的に記事を書いた日時が設定されます。

ひとまずこれでおしまいです！あとはフォームをビューで使い、それをテンプレート内に表示しさえすればいいです。

もう一度、ページへのリンク、URL、ビューとテンプレートを作ります。

29.1 フォームにおけるページへのリンク

`blog/templates/blog/base.html` をエディタで開きましょう。`page-header` と名付けた `div` 中に次のリンクを追加します：

```
{% filename %}blog/templates/blog/base.html{% endfilename %}
```

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

新しいビュー `post_new` を呼び出すことに注意してください。`"glyphicon glyphicon-plus"` クラスは、使用している Bootstrap テーマによって提供され、プラス記号を表示します。

行を追加すると、このような html ファイルになります。

```
{% filename %}blog/templates/blog/base.html{% endfilename %}
```

```
{% load static %}
<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">
        <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-home"></span> 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>
```

ファイルを保存して、ページ `http://127.0.0.1:8000` をリロードすると見覚えのある `NoReverseMatch` エラーが表示されると思います。実際にそうなってますか？ いいですね！

29.2 URL

`blog/urls.py` をエディタで開き、次の内容を追加します。

```
{% filename %}blog/urls.py{% endfilename %}
```

```
path('post/new', views.post_new, name='post_new'),
```

すると最終的なコードは次のようにになります:

```
{% filename %}blog/urls.py{% endfilename %}
```

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.post_list, name='post_list'),
    path('post/<int:pk>/', views.post_detail, name='post_detail'),
    path('post/new/', views.post_new, name='post_new'),
]
```

サイトをリロードした後、`AttributeError` が出来ます。`post_new` ビューの実装がないからです。ファイルに追加してみましょう。

29.3 post_new ビュー

`blog/views.py` をエディタで開き、`from` の行の後に次の内容を追加してみましょう。

```
{% filename %}blog/views.py{% endfilename %}
```

```
from .forms import PostForm
```

その後にビューを追加します。

```
{% filename %}blog/views.py{% endfilename %}
```

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

`Post` フォームを新しく作るには、`PostForm()` を呼び出し、それをテンプレートに渡す必要があります。あとでこの ビューに戻ってきますが、今はフォームのためのテンプレートをすぐに作ってしまいましょう。

29.4 テンプレート

`blog/templates/blog` ディレクトリに `post_edit.html` ファイルを作り、コードエディタで開きましょう。フォームを動かす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。 私たちは（例えば）`{% raw %}{{ form.as_p }}{% endraw %}` でこれを行なうことができます。
- 上記の行は HTML の `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。これを HTML の `button` タグで行います：`<button type="submit">Save</button>`
- 最後に`<form ...>` タグの開始直後に、`{% raw %}{% csrf_token %}{% endraw %}` を追加する必要があります。 フォームをセキュアにするためこれは非常に重要です！ これを忘れると、Django はフォームを保存しようとすると文句を言うでしょう：

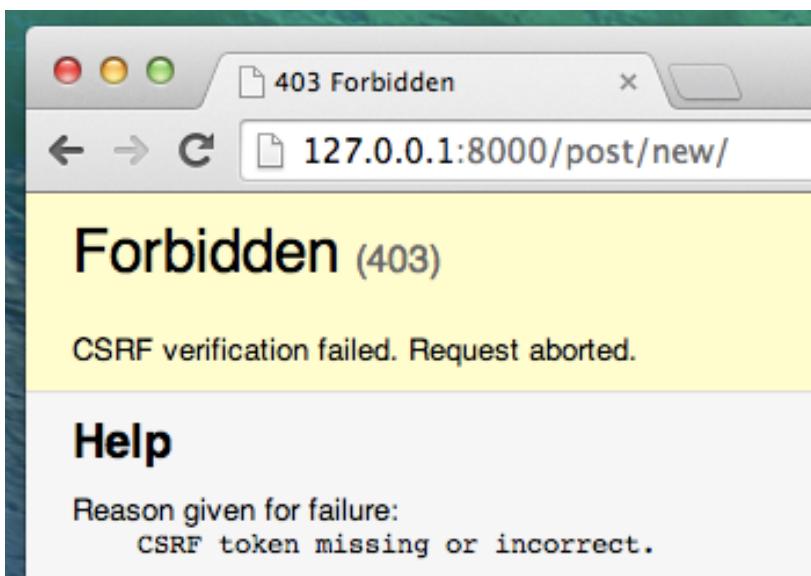


図 29.1: CSRF 禁止のページ

では、`post_edit.html` の HTML がどのようになるか見てみましょう：

```
{% filename %}blog/templates/blog/post_edit.html{% endfilename %}
```

```
{% 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 %}
```

更新をしてみましょう。やった！ フォームが表示されます。

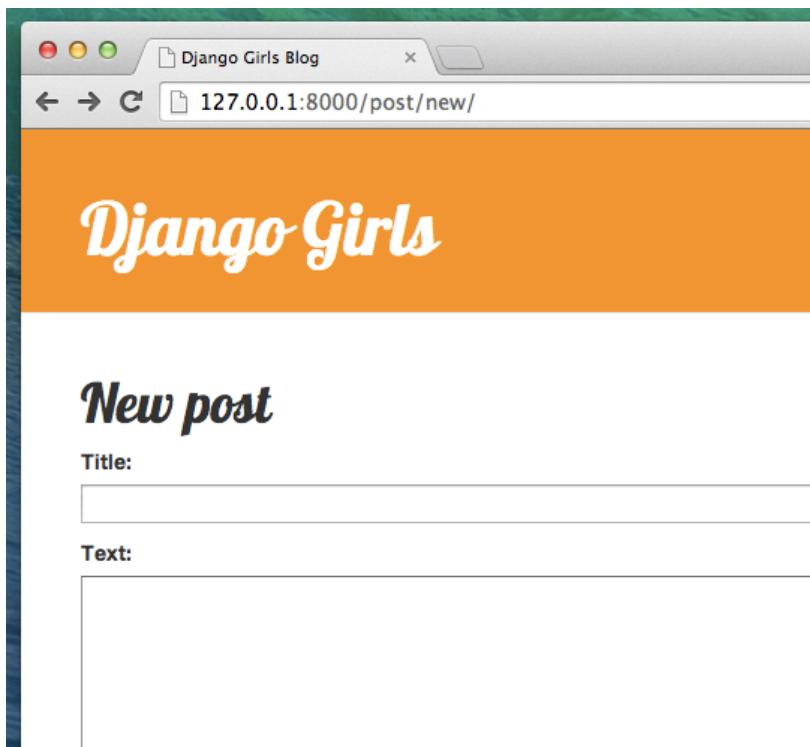


図 29.2: New form

ちょっと待ってみて下さい。title と text フィールドに何か入力して保存するとどうなりますか？

何も起きません！ もう一度同じページに戻りテキストはどこかに行ってしましました… そして新しい投稿は追加されていません。何がいけなかったのでしょうか？

答えは：何も間違ってない、です。ビュー でもう少し作業を行う必要があります。

29.5 フォームを保存する

`blog/views.py` をもう一度エディタで開きます。現在の `post_new` ビューはこうなっています。

```
{% filename %}blog/views.py{% endfilename %}
```

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

フォームを送信したとき、同じビューに戻されていましたが、このとき `request`、もっと詳しくいうと `request.POST` にデータが追加されています（この POST という名前はブログ投稿 "post" とは関係ありません。このデータは送られてきたもの、というコトと関係しています）。HTML ファイルの `<form>` タグで、`method="POST"` という変数があったのを覚えていませんか？これによってフォームのすべてのフィールドは今 `request.POST` にあります。POST という名前を何か別のものに変えることはできません（他に唯一の有効な `method` の値は `GET` ですが、その違いを説明する時間はありません）。

私たちの ビュー では、扱わなくてはならない 2 つの別々のシチュエーションがあります：1 つ目は、最初にページにアクセスしてきた時で空白のフォームが必要な場合。2 つ目はすべてのフォームデータが入力された状態でビューに戻ってくる場合です。したがって条件分岐を追加する必要があります（そのためには `if` を使います）：

```
{% filename %}blog/views.py{% endfilename %}
```

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

ドット [...] の部分を埋めていきましょう。`method` が `POST` の場合、フォームのデータを使って `PostForm` を構築します。私たちはそれを次のようにします：

```
{% filename %}blog/views.py{% endfilename %}
```

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

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、不正な値が送信されないこと）。`form.is_valid()`で行います。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。

```
{% filename %}blog/views.py{% endfilename %}
```

```
if form.is_valid():
    post = form.save(commit=False)
    post.author = request.user
    post.published_date = timezone.now()
    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(`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`モデルをまだ保存しないという意味です。保存前に`author`を追加したいので。ほとんどの場合、`commit = False`なしで`form.save()`を使用しますが、この場合はそれを指定する必要があります。`post.save()`は変更を保存し(作成者を追加しつつ)、新しいブログ投稿が作成されます！

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？そのため次のインポートを追加します：

```
{% filename %}blog/views.py{% endfilename %}
```

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

ファイルの先頭に追加します。これで新しく作成されたポストの`post_detail`ページに移動する処理を書けます。

```
{% filename %}blog/views.py{% endfilename %}
```

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

`post_detail` は移動したいビューの名前です。このビューでは `pk` 変数が必須であることを覚えていますか？ビューにそれを渡すため、`pk=post.pk` を使います。この `post` は新しく作られたブログポストです！

ふー、たくさんのこと話を話してきましたが、そろそろ ビューの全体がどんな感じか見てみたい頃じゃないでしょうか？

```
{% filename %}blog/views.py{% endfilename %}
```

```
def post_new(request):
    if request.method == "POST":
        form = PostForm(request.POST)
        if form.is_valid():
            post = form.save(commit=False)
            post.author = request.user
            post.published_date = timezone.now()
            post.save()
            return redirect('post_detail', pk=post.pk)
    else:
        form = PostForm()
    return render(request, 'blog/post_edit.html', {'form': form})
```

では動作確認してみましょう。<http://127.0.0.1:8000/post/new/> に行き、`title` と `text` を追加し、保存すると……じゃじゃーん！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！

ブログ記事を保存する前に公開日をセットしていることに気づいたかもしれません。後ほど、Django Girls Tutorial: Extensions にて 公開ボタンを導入します。

素晴らしい！

最近まで Django admin を使ってきていたので、システム上で今まだログイン状態かと思います。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..）。投稿を作成するときに、ログインユーザーがわからないというエラーが発生した場合は、管理ページ <http://127.0.0.1:8000/admin> にアクセスして再度ログインしてください。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。

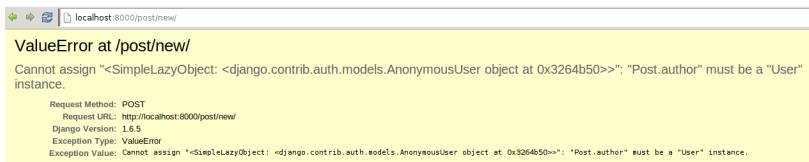


図 29.3: ログインエラー

29.6 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 (`published_date` とは対照的に)、Django 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ることを期待します。

`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?

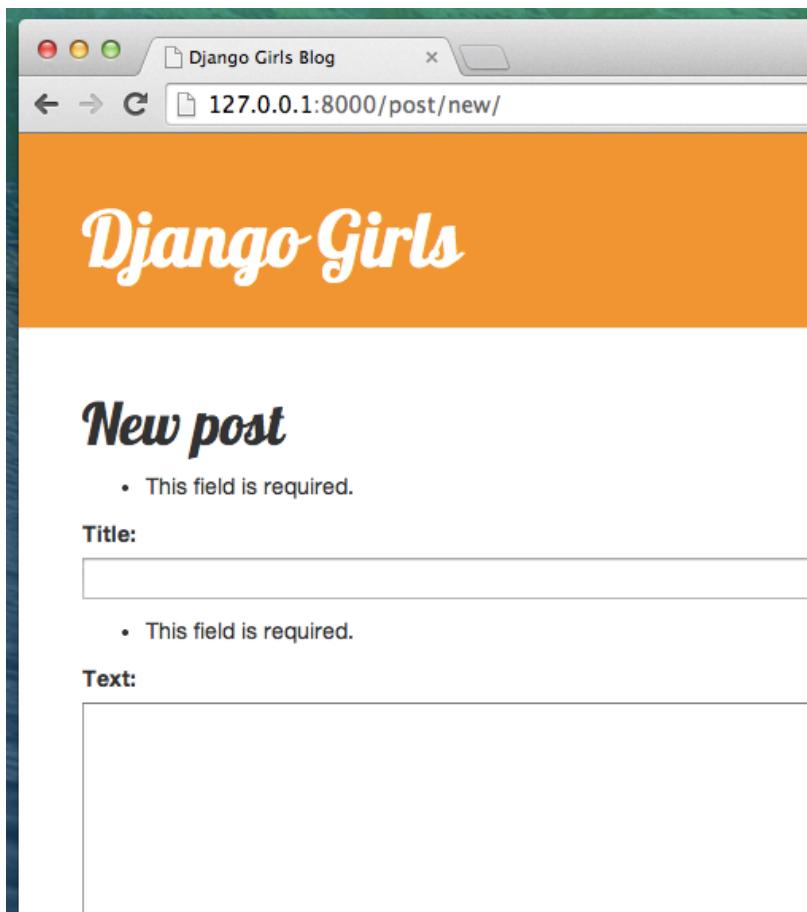


図 29.4: フォームのバリデーション (検証)

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？

29.7 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 をエディタで開いて次の行を追加します

```
{% filename %}blog/templates/blog/post_detail.html{% endfilename %}
```

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

テンプレートは次のようにになります:

```
{% filename %}blog/templates/blog/post_detail.html{% endfilename %}
```

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

{% block content %}
    <div class="post">
        {% if post.published_date %}
            <div class="date">
                {{ post.published_date }}
            </div>
        {% endif %}
        <a class="btn btn-default" href="{% url 'post_edit' pk=post.pk %}"><span class="gl
```

blog/urls.py をエディタで開き、次の内容を追加します。

```
{% filename %}blog/urls.py{% endfilename %}
```

```
path('post/<int:pk>/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。そして残るはビューです。

blog/views.py をエディタで開いて次の内容をファイルの最後に追加します:
{% filename %}blog/views.py{% endfilename %}

```
def post_edit(request, pk):
    post = get_object_or_404(Post, pk=pk)
    if request.method == "POST":
        form = PostForm(request.POST, instance=post)
        if form.is_valid():
            post = form.save(commit=False)
            post.author = request.user
            post.published_date = timezone.now()
            post.save()
            return redirect('post_detail', pk=post.pk)
    else:
        form = PostForm(instance=post)
    return render(request, 'blog/post_edit.html', {'form': form})
```

post_new 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 から追加の pk パラメータを渡します。次に編集したい Post モデルを get_object_or_404(Post, pk=pk) で取得し、フォームを作るときは以下の 2 つのケースのようにそのポストを instance (インスタンス) として渡します。フォームを保存するときは…

{% filename %}blog/views.py{% endfilename %}

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

…このポストを編集するためにただフォームを開く場合は:

{% filename %}blog/views.py{% endfilename %}

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

よし、ちゃんと動くか試してみましょう！post_detail ページにいきましょう。その右上に [編集] ボタンがあるはずです：



図 29.5: 編集ボタン

クリックするとブログ記事のフォームが表示されると思います：

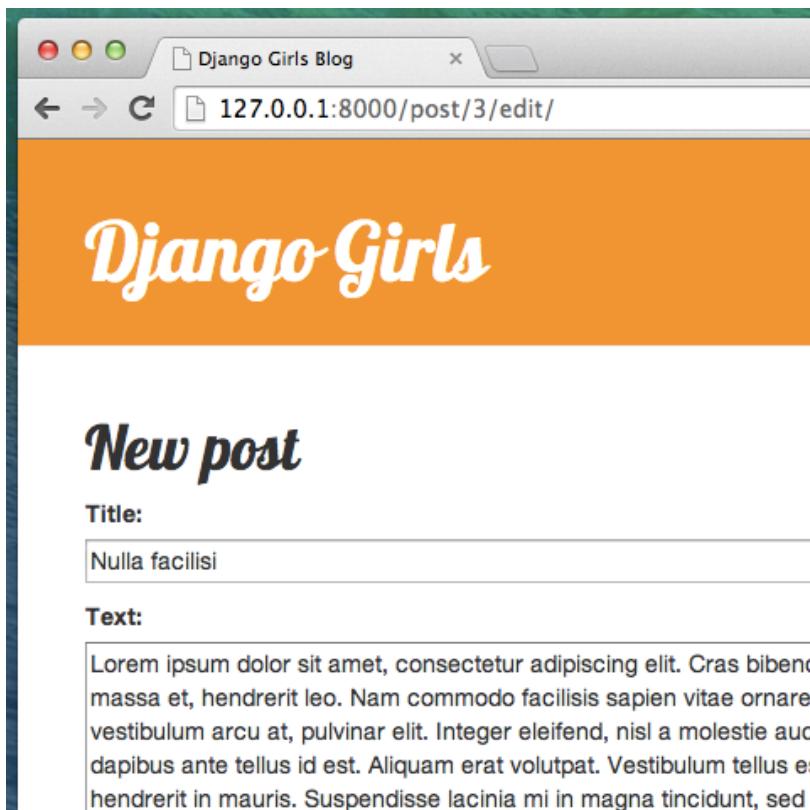


図 29.6: 編集フォーム

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もっと知りたい場合、Django Project のドキュメントを読んでください: <https://docs.djangoproject.com/ja/2.0/topics/forms/>

29.8 セキュリティ

リンクをクリックするだけで新しい投稿を作成できることは素晴らしいことです！ しかし、今、あなたのサイトにアクセスした人は誰でも新しいブログ投稿を作成することができます。それはおそらくあなたが望むものではありません。

ボタンはあなたのためには表示されますが、他の人には表示されないようにしましょう。

blog/templates/blog/base.html をエディタで開き、page-header と名付けた div とそこに以前に入力したアンカータグを見つけます。これは次のようになります。

```
{% filename %}blog/templates/blog/base.html{% endfilename %}
```

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

これに{% if %}タグを追加し、管理者でログインしているユーザーのみにリンクを表示します。今は、あなただけです！<a> タグを以下のように変更します：

```
{% filename %}blog/templates/blog/base.html{% endfilename %}
```

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

この{% if %}は、ページをリクエストしているユーザーがログインしている場合にのみ、リンクがブラウザに送信されるようにします。これは新しい投稿の作成を完全に保護するものではありませんが、それは良い第一歩です。私たちちは拡張レッスンでより多くのセキュリティをカバーします。

詳細ページに追加した編集アイコンを覚えてますか？他の人が既存の投稿を編集できないように、同じ変更を追加したいと思います。

blog/templates/blog/post_detail.html をエディタで開いて次の行を見つけてください：

```
{% filename %}blog/templates/blog/post_detail.html{% endfilename %}
```

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

以下のように変更してください：

```
{% filename %}blog/templates/blog/post_detail.html{% endfilename %}
```

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

あなたはログインしている可能性が高いので、ページを更新しても、何も変わらないかもしれません。ただし、別のブラウザやシークレットウィンドウ（Windows Edge では「InPrivate」と呼ばれます）でページを読み込むと、リンクが表示されず、アイコンも表示されないでしょう！

29.9 もう一つ：デプロイの時間です！

では PythonAnywhere 上で動作するかを確認しましょう。再度デプロイします。

- まず、Github にあなたの新しく書いたコードを Commit して、Push してみましょう。

```
{% filename %}command-line{% endfilename %}
```

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

- それから、PythonAnywhere の bash コンソールで：

```
{% filename %}command-line{% endfilename %}
```

```
$ cd ~/<your-pythonanywhere-username>.pythonanywhere.com
$ git pull
[...]
```

(<your-pythonanywhere-username>の部分を、自分の実際の PythonAnywhere のユーザー名に山カッコをはずして置き換えることを忘れずに)

- 最後に、「Web」ページに飛んで（コンソールの右上のメニュー ボタンを使って）それから **Reload** を押しましょう。変更を見るためにあなたのブログ <https://yourname.pythonanywhere.com> を再読み込みしましょう。

うまくいってるはずです！ おめでとう :)

第 30 章

次のステップは？

おめでとうございます。素晴らしいです。よくやりました！ <3

今何をする？

休憩してリラックスしてください。あなたは本当に大きなことを成し遂げました。

休んだ後は、Facebook や Twitter で Django Girls をぜひフォローしてくださいね。最新の情報を入手できます。

さらにオススメの資料はありますか？

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 の公式チュートリアル
- 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 1.11: Best Practices for Django Web Framework book
- Hello Web App: Learn How to Build a Web App - 著者の Tracy Osborn

第30章 次のステップは？

(??) に連絡して無料の eBook ライセンスをリクエストすることもできます

Django Girls Tutorial

.djangogirls.org

2018年10月8日 ver 1.0
著者 Django Girls Tokyo

(C) 2018 Django Girls Tokyo