tutorial.djangogirls.org

Django installation · Django Girls Tutorial

DjangoGirls

7-9 minuti

> **Note** If you're using a Chromebook, skip this chapter and make sure you follow the Chromebook Setup instructions.
>
> **Note** If you already worked through the Installation steps then you've already done this – you can go straight to the next chapter!
>
> Part of this section is based on tutorials by Geek Girls Carrots (https://github.com/ggcarrots/django-carrots).
>
> Part of this section is based on the django-marcador tutorial licensed under the Creative Commons Attribution-ShareAlike 4.0 International License. The django-marcador tutorial is copyrighted by Markus Zapke-Gründemann et al.

Virtual environment

Before we install Django we will get you to install an extremely useful tool to help keep your coding environment tidy on your computer. It's possible to skip this step, but it's highly recommended. Starting with the best possible setup will save you a lot of trouble in the future!

So, let's create a **virtual environment** (also called a *virtualenv*). Virtualenv will isolate your Python/Django setup on a per-project basis. This means that any changes you make to one website won't affect any others you're also developing. Neat, right?

All you need to do is find a directory in which you want to create the `virtualenv`; your home directory, for example. On Windows it might look like `C:\Users\Name\` (where `Name` is the name of your login).

> **NOTE:** On Windows, make sure that this directory does not contain accented or special characters; if your username contains accented characters, use a different directory, for example `C:\djangogirls`.

For this tutorial we will be using a new directory `djangogirls` from your home directory:

command-line

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

We will make a virtualenv called `myvenv`. The general command will be in the format:

command-line

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

Virtual environment: Windows

To create a new `virtualenv`, you need to open the command prompt and run `python -m venv myvenv`. It will look like this:

command-line

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

Where `myvenv` is the name of your `virtualenv`. You can use any other name, but stick to lowercase and use no spaces, accents or special characters. It is also good idea to keep the name short – you'll be referencing it a lot!

Virtual environment: Linux and OS X

Creating a `virtualenv` on both Linux and OS X is as simple as running `python3 -m venv myvenv`. It will look like this:

command-line

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

`myvenv` is the name of your `virtualenv`. You can use any other name, but stick to lowercase and use no spaces. It is also good idea to keep the name short as you'll be referencing it a lot!

> **NOTE:** On some versions of Debian/Ubuntu you may receive the following error:
>
> command-line
>
> ```
> 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-get install python3-venv
> You may need to use sudo with that command.
> After installing the python3-venv package,
> recreate your virtual environment.
> ```
>
> In this case, follow the instructions above and install the `python3-venv` package:
>
> command-line
>
> ```
> $ sudo apt-get install python3-venv
> ```
>
> **NOTE:** On some versions of Debian/Ubuntu initiating the virtual environment like this currently gives the following error:
>
> command-line

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

To get around this, use the `virtualenv` command instead.

command-line

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

NOTE: If you get an error like

command-line

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

then instead run:

command-line

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

Working with virtualenv

The command above will create a directory called `myvenv` (or whatever name you chose) that contains our virtual environment (basically a bunch of directory and files).

Working with virtualenv: Windows

Start your virtual environment by running:

command-line

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

NOTE: on Windows 10 you might get an error in the Windows PowerShell that says `execution of scripts is disabled on this system`. In this case, open another Windows

> PowerShell with the "Run as Administrator" option. Then try typing the following command before starting your virtual environment:
>
> command-line
>
> ```
> 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
> ```

Working with virtualenv: Linux and OS X

Start your virtual environment by running:

command-line

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

Remember to replace `myvenv` with your chosen `virtualenv` name!

> **NOTE:** sometimes `source` might not be available. In those cases try doing this instead:
>
> command-line
>
> ```
> $ . myvenv/bin/activate
> ```

You will know that you have `virtualenv` started when you see that the prompt in your console is prefixed with `(myvenv)`.

When working within a virtual environment, `python` will automatically refer to the correct version so you can use `python` instead of `python3`.

OK, we have all important dependencies in place. We can finally install Django!

Installing Django

Now that you have your `virtualenv` started, you can install Django.

Before we do that, we should make sure we have the latest version of `pip`, the software that we use to install Django:

command-line

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

Then run `pip install django~=1.11.0` (note that we use a tilde followed by an equal sign: `~=`) to install Django.

command-line

```
(myvenv) ~$ pip install django~=1.11.0
Collecting django~=1.11.0
  Downloading Django-1.11.3-py2.py3-none-any.whl
(6.8MB)
Installing collected packages: django
Successfully installed django-1.11.3
```

Installing Django: Windows

> If you get an error when calling pip on Windows platform, please check if your project pathname contains spaces, accents or special characters (for example, `C:\Users\User Name\djangogirls`). If it does, please consider using another place without spaces, accents or special characters (suggestion:

> `C:\djangogirls`). Create a new virtualenv in the new directory, then delete the old one and try the above command again. (Moving the virtualenv directory won't work since virtualenv uses absolute paths.)

Installing Django: Windows 8 and Windows 10

> Your command line might freeze after when you try to install Django. If this happens, instead of the above command use:
>
> command-line
>
> ```
> C:\Users\Name\djangogirls> python -m pip install
> django~=1.11.0
> ```

> If you get an error when calling pip on Ubuntu 12.04 please run `python -m pip install -U --force-reinstall pip` to fix the pip installation in the virtualenv.

That's it! You're now (finally) ready to create a Django application!

Django models

What we want to create now is something that will store all the posts in our blog. But to be able to do that we need to talk a little bit about things called `objects`.

Objects

There is a concept in programming called `object-oriented programming`. The idea is that instead of writing everything as a boring sequence of programming instructions, we can model things and define how they interact with each other.

So what is an object? It is a collection of properties and actions. It sounds weird, but we will give you an example.

If we want to model a cat, we will create an object `Cat` that has some properties such as `color`, `age`, `mood` (like good, bad, or sleepy ;)), and `owner` (which could be assigned a `Person` object – or maybe, in case of a stray cat, this property could be empty).

Then the `Cat` has some actions: `purr`, `scratch`, or `feed` (in which case, we will give the cat some `CatFood`, which could be a separate object with properties, like `taste`).

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


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

So basically the idea is to describe real things in code with properties (called `object properties`) and actions (called `methods`).

tutorial.djangogirls.org

Django ORM (Querysets) · Django Girls Tutorial

DjangoGirls

6-7 minuti

In this chapter you'll learn how Django connects to the database and stores data in it. Let's dive in!

What is a QuerySet?

A QuerySet is, in essence, a list of objects of a given Model. QuerySets allow you to read the data from the database, filter it and order it.

It's easiest to learn by example. Let's try this, shall we?

Django shell

Open up your local console (not on PythonAnywhere) and type this command:

command-line

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

The effect should be like this:

command-line

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

You're now in Django's interactive console. It's just like the Python prompt, but with some additional Django magic. :) You can use all the Python commands here too, of course.

All objects

Let's try to display all of our posts first. You can do that with the following command:

command-line

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

Oops! An error showed up. It tells us that there is no Post. It's correct – we forgot to import it first!

command-line

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

This is simple: we import the model `Post` from `blog.models`. Let's try displaying all posts again:

command-line

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

This is a list of the posts we created earlier! We created these posts using the Django admin interface. But now we want to create new posts using Python, so how do we do that?

Create object

This is how you create a new Post object in database:

command-line

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

But we have one missing ingredient here: `me`. We need to pass an instance of `User` model as an author. How do we do that?

Let's import User model first:

command-line

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

What users do we have in our database? Try this:

command-line

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

This is the superuser we created earlier! Let's get an instance of the user now:

command-line

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

As you can see, we now `get` a `User` with a `username` that equals 'ola'. Neat! Of course, you have to adjust this line to use your own username.

Now we can finally create our post:

command-line

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

Hurray! Wanna check if it worked?

command-line

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

There it is, one more post in the list!

Add more posts

You can now have a little fun and add more posts to see how it works. Add two or three more and then go ahead to the next part.

Filter objects

A big part of QuerySets is the ability to filter them. Let's say we want to find all posts that user ola authored. We will use `filter` instead of `all` in `Post.objects.all()`. In parentheses we state what condition(s) a blog post needs to meet to end up in our queryset. In our case, the condition is that `author` should be equal to `me`. The way to write it in Django is `author=me`. Now our piece of code looks like this:

command-line

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

Or maybe we want to see all the posts that contain the word 'title' in the `title` field?

command-line

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

> There are two underscore characters (_) between `title` and `contains`. Django's ORM uses this rule to separate field names ("title") and operations or filters ("contains"). If you use only one underscore, you'll get an error like "FieldError: Cannot resolve keyword title_contains".

You can also get a list of all published posts. We do this by filtering all the posts that have `published_date` set in the past:

command-line

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

Unfortunately, the post we added from the Python console is not published yet. But we can change that! First get an instance of a post we want to publish:

command-line

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

And then publish it with our `publish` method:

command-line

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

Now try to get list of published posts again (press the up arrow key three times and hit `enter`):

command-line

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

Ordering objects

QuerySets also allow you to order the list of objects. Let's try to order them by `created_date` field:

command-line

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

We can also reverse the ordering by adding `-` at the beginning:

command-line

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

Chaining QuerySets

You can also combine QuerySets by **chaining** them together:

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

This is really powerful and lets you write quite complex queries.

Cool! You're now ready for the next part! To close the shell, type this:

command-line

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

tutorial.djangogirls.org

Django templates · Django Girls Tutorial

DjangoGirls

4-5 minuti

Time to display some data! Django gives us some helpful built-in **template tags** for that.

You see, in HTML, you can't really write Python code, because browsers don't understand it. They know only HTML. We know that HTML is rather static, while Python is much more dynamic.

Django template tags allow us to transfer Python-like things into HTML, so you can build dynamic websites faster and easier. Cool!

Display post list template

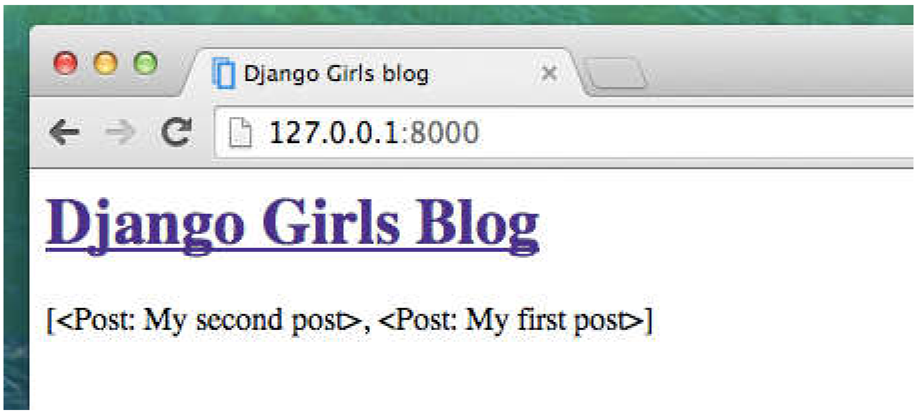
In the previous chapter we gave our template a list of posts in the `posts` variable. Now we will display it in HTML.

To print a variable in Django templates, we use double curly brackets with the variable's name inside, like this:

blog/templates/blog/post_list.html

```
{{ posts }}
```

Try this in your `blog/templates/blog/post_list.html` template. Replace everything from the second `<div>` to the third `</div>` with `{{ posts }}`. Save the file, and refresh the page to see the results:



Django Girls Blog

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

As you can see, all we've got is this:

blog/templates/blog/post_list.html

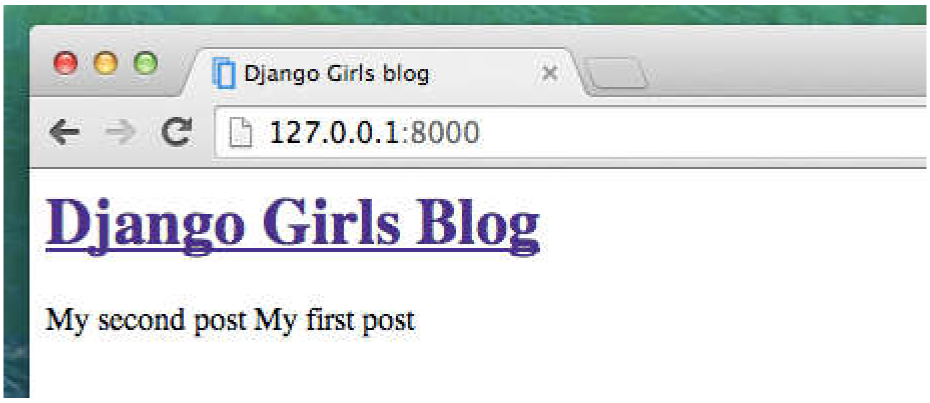
```
<QuerySet [<Post: My second post>, <Post: My
first post>]>
```

This means that Django understands it as a list of objects. Remember from **Introduction to Python** how we can display lists? Yes, with for loops! In a Django template you do them like this:

blog/templates/blog/post_list.html

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

Try this in your template.



Django Girls Blog

My second post My first post

It works! But we want the posts to be displayed like the static posts we created earlier in the **Introduction to HTML** chapter.

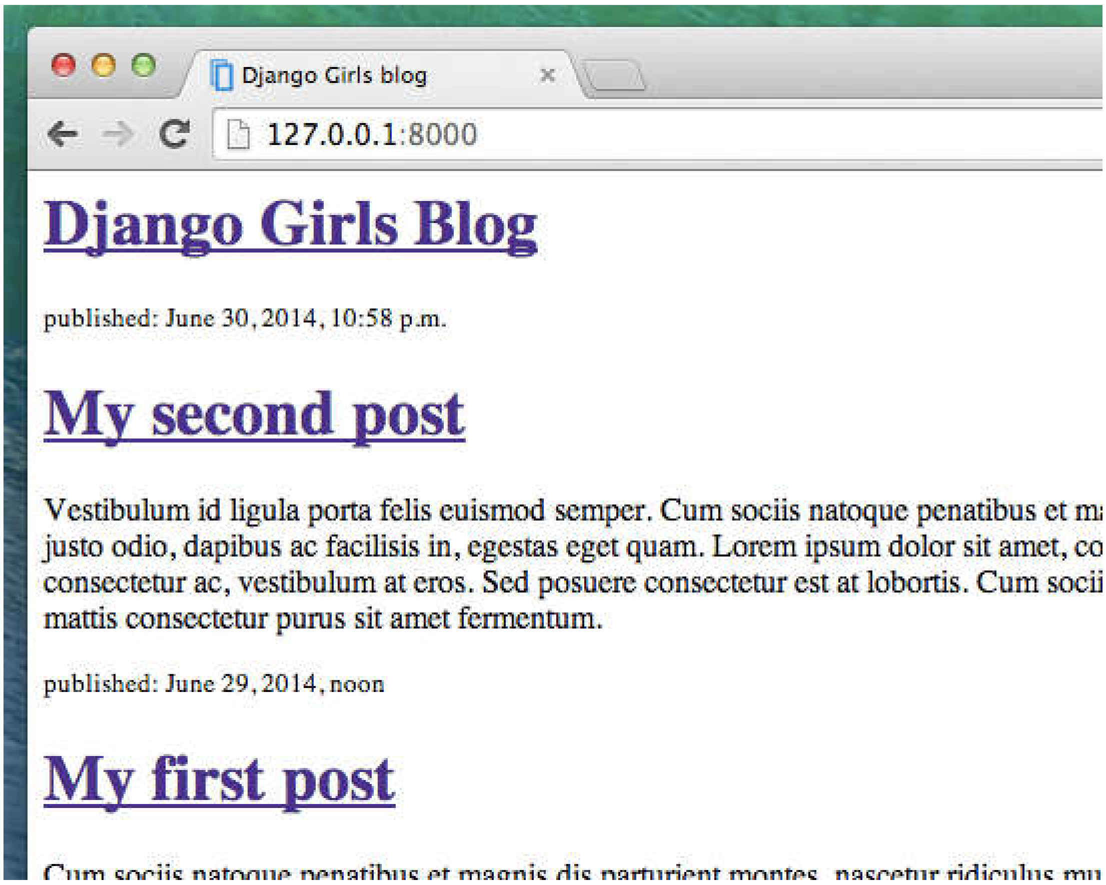
You can mix HTML and template tags. Our `body` will look like this:

blog/templates/blog/post_list.html

```
<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 %}
```

Everything you put between `{% for %}` and `{% endfor %}` will be repeated for each object in the list. Refresh your page:



Django Girls Blog

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

My second post

Vestibulum id ligula porta felis euismod semper. Cum sociis natoque penatibus et m
justo odio, dapibus ac facilisis in, egestas eget quam. Lorem ipsum dolor sit amet, co
consectetur ac, vestibulum at eros. Sed posuere consectetur est at lobortis. Cum socii
mattis consectetur purus sit amet fermentum.

published: June 29, 2014, noon

My first post

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

Have you noticed that we used a slightly different notation this time (`{{ post.title }}` or `{{ post.text }}`)? We are accessing data in each of the fields defined in our `Post` model. Also, the `|linebreaksbr` is piping the posts' text through a filter to convert line-breaks into paragraphs.

One more thing

It'd be good to see if your website will still be working on the public Internet, right? Let's try deploying to PythonAnywhere again. Here's a recap of the steps…

- First, push your code to Github

  command-line

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

- Then, log back in to PythonAnywhere and go to your **Bash console** (or start a new one), and run:

  PythonAnywhere command-line

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

- Finally, hop on over to the Web tab and hit **Reload** on your web app. Your update should be live! If the blog posts on your PythonAnywhere site don't match the posts appearing on the blog hosted on your local server, that's OK. The databases on your local computer and Python Anywhere don't sync with the rest of your files.

Congrats! Now go ahead and try adding a new post in your Django admin (remember to add published_date!) Make sure you are in the Django admin for your pythonanywhere site, https://yourname.pythonanywhere.com/admin. Then refresh your page to see if the post appears there.

Works like a charm? We're proud! Step away from your computer for a bit – you have earned a break. :)