SIMPLEISBETTERTHANCOMPLEX

By **Vitor Freitas**

I'm a passionate software developer and researcher from Brazil, currently living in Finland. I write about Python, Django and Web Development on a weekly basis. Read more.

Table of Contents

Introduction
Why Django?
- Who's Using Django?

Installation
- Installing Python 3.6.2
- Installing Virtualenv
- Installing Django 1.11.4

Starting a New Project
Django Apps
Hello, World!
Conclusions

scroll to top
go to comments
go to series index

SERIES

A Complete Beginner's Guide to Django - Part 1

Sep 4, 2017 20 minutes read 109 comments 204,893 views Part 1 of 7

Mac Windows Linux

Series 1/7

python 3.6.2 django 1.11.4

Introduction

Today I'm starting a new tutorial series about Django fundamentals. It's a complete beginner's guide to start learning Django. The material is divided into seven parts. We're going to explore all the basic concepts in great detail, from installation, preparation of the development environment, models, views, templates, URLs to more advanced topics such as migrations, testing, and deployment.

I wanted to do something different. A tutorial that would be easy to follow, informative and fun to read. That was when I came up with the idea to create some comics along the text to illustrate some concepts and scenarios. I hope you enjoy the reading!

But before we start…

Back when I worked as a substitute professor in a university, I used to teach an introduction to web development discipline for the newcomer students in the Computer Science course. And I would always start new classes with this Confucius quote:

So, hands on! Don't just read the tutorials. Let's do it together! You will learn much more by doing and practicing.

Why Django?

Django is a Web framework written in Python. A Web framework is a software that supports the development of dynamic Web sites, applications, and services. It provides a set of tools and functionalities that solves many common problems associated with Web development, such as security features, database access, sessions, template processing, URL routing, internationalization, localization, and much more.

Using a Web framework, such as Django, enables us to develop secure and reliable Web applications very quickly in a standardized way, without having to reinvent the wheel.

So, what's so special about Django? For starters, it's a Python Web framework, which means you can benefit from wide a range of open source libraries out there. The Python Package Index repository hosts over **116K** packages (as per 6 of Sep. 2017). If you need to solve a specific problem, the chances are someone has already implemented a library for it.

Django is one of the most popular Web frameworks written in Python. It's definitely the most complete, offering a wide range of features out-of-the-box, such as a standalone Web server for development and testing, caching, middleware system, ORM, template engine, form processing, interface with Python's unit testing tools. Django also comes with *battery included*, offering built-in applications such as an authentication system, an administrative interface with automatically

generated pages for CRUD operations, generation of syndication feeds (RSS/Atom), sitemaps. There's even a Geographic Information System (GIS) framework built within Django.

The development of Django is supported by the Django Software Foundation, and it's sponsored by companies like JetBrains and Instagram. Django has also been around for quite some time now. It's under active development for more than 12 years now, proving to be a mature, reliable and secure Web framework.

Who's Using Django?

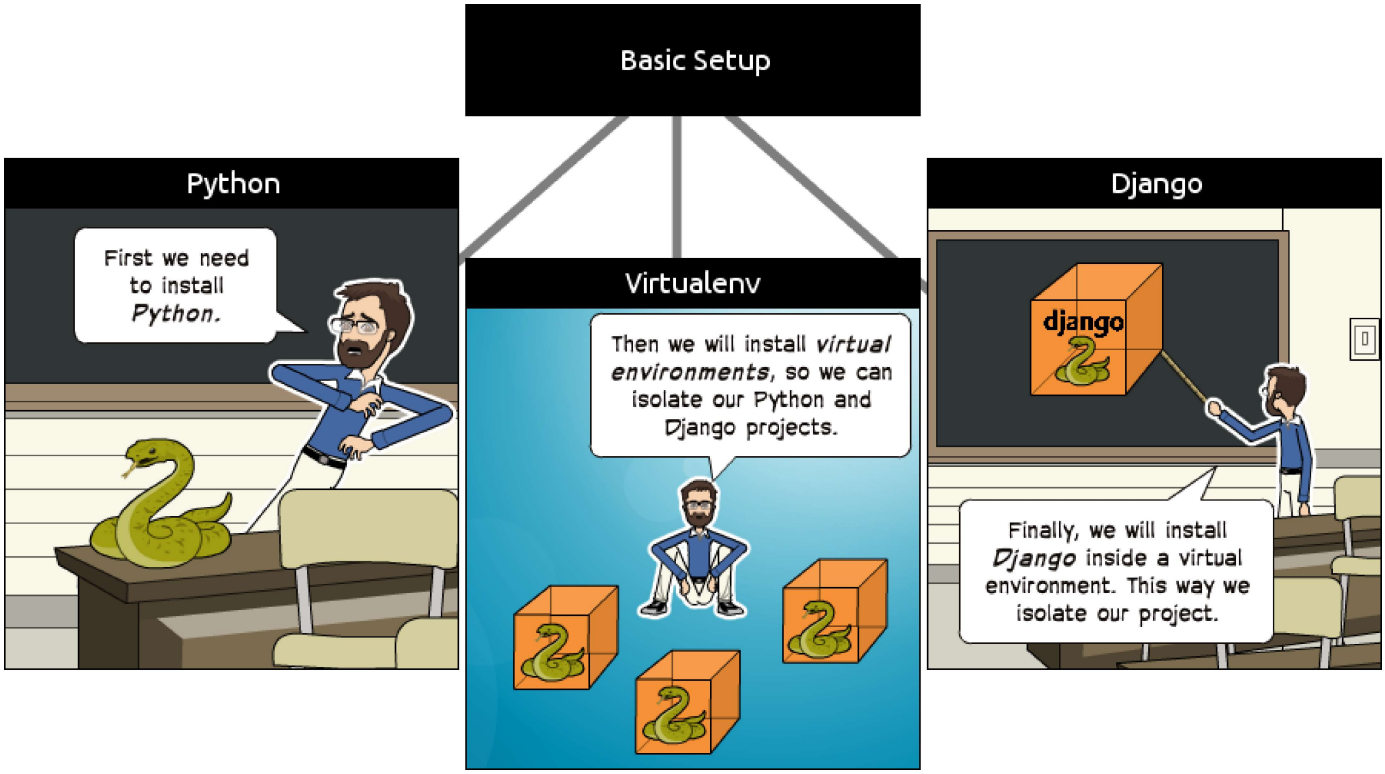
It's good to know who is using Django out there, so to have an idea what you can do with it. Among the biggest Web sites using Django we have: Instagram, Disqus, Mozilla, Bitbucket, Last.fm, National Geographic.

For more examples you can see the Django Sites database, they offer a list of over **5K** Django-powered Web sites.

By the way, last year, in the Django Under The Hood 2016 conference, Carl Meyer, a Django core developer, and Instagram employee, gave a talk on how Instagram use Django at scale and how it supported their growth. It's a one hour talk, but if you are interested in learning more, it was an entertaining talk.

Installation

The first thing we need to do is install some programs on our machine so to be able to start playing with Django. The basic setup consists of installing **Python**, **Virtualenv**, and **Django**.



Using virtual environments is not mandatory, but it's highly recommended. If you are just getting started, it's better to start with the right foot.

When developing Web sites or Web projects with Django, it's very common to have to install external libraries to support the development. Using virtual environments, each project you develop will have its isolated environment. So the dependencies won't clash. It also allows you to maintain in your local machine projects that run on different Django versions.

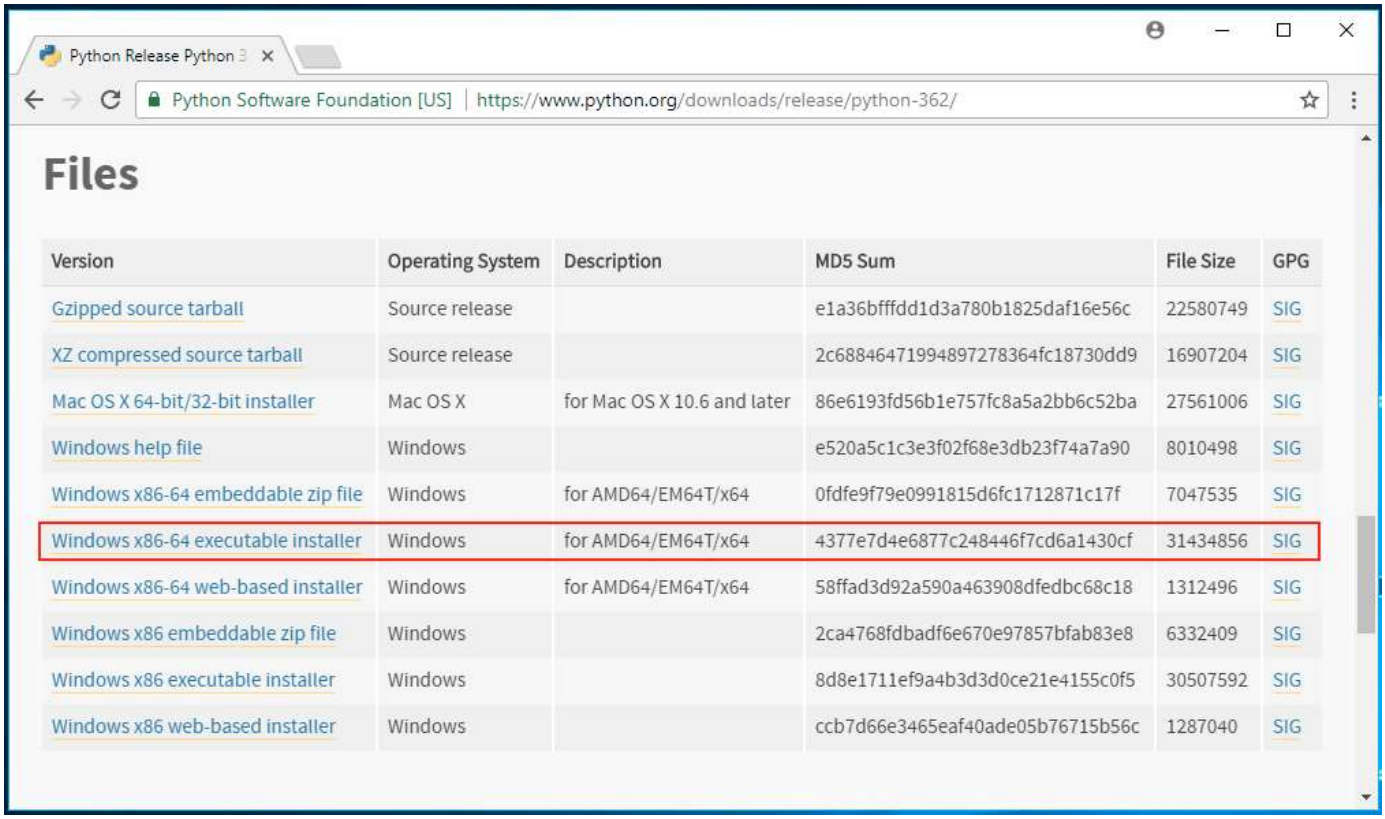
It's very straightforward to use it, you will see!

Installing Python 3.6.2

The first thing we want to do is install the latest Python distribution, which is **Python 3.6.2**. At least it was, by the time I was writing this tutorial. If there's a newer version out there, go with it. The next steps should remain more or less the same.

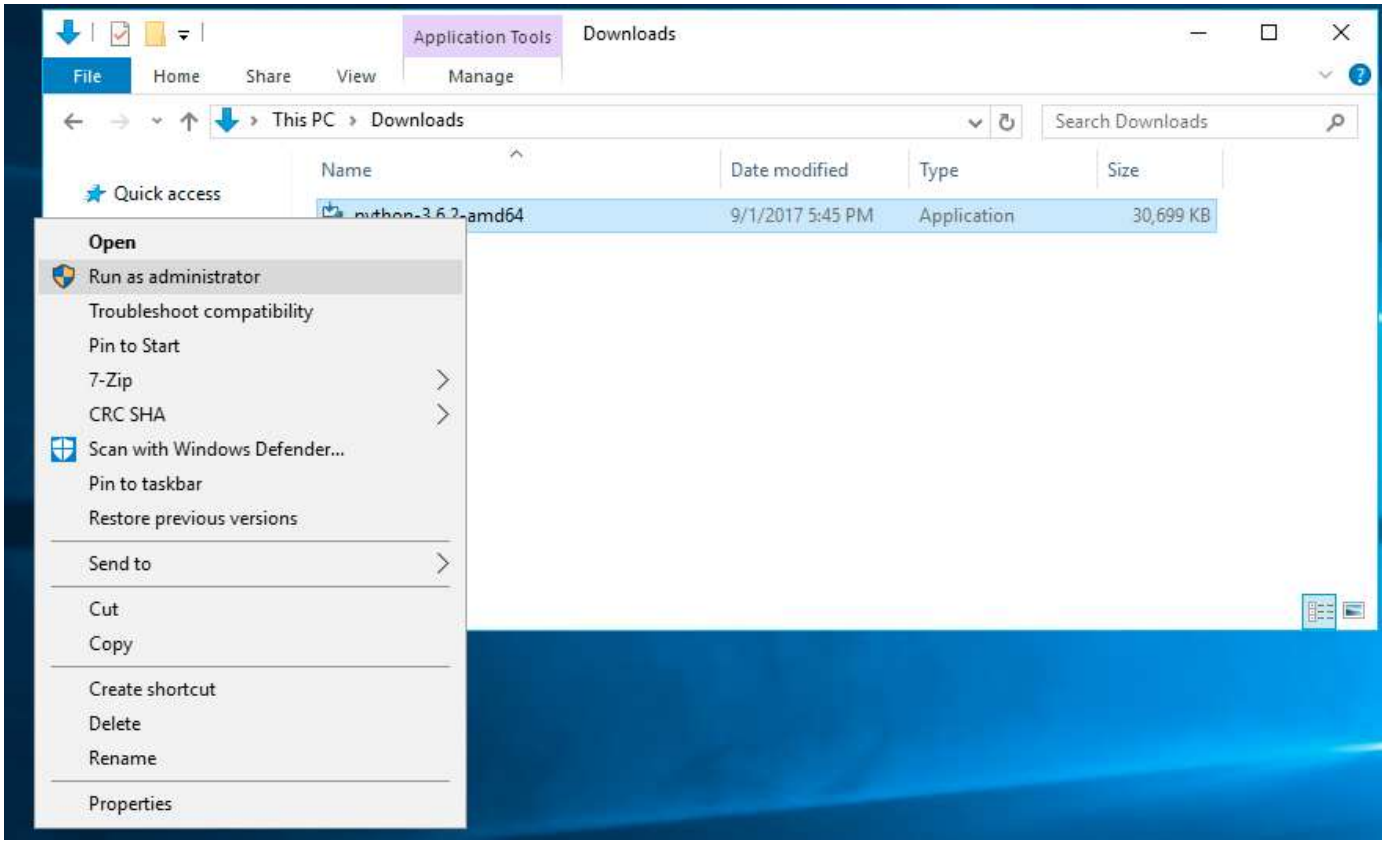
We are going to use Python 3 because the most important Python libraries have already been ported to Python 3 and also the next major Django version (2.x) won't support Python 2 anymore. So Python 3 is the way to go.

Go to www.python.org click on the Python 3.6.2 download page, scroll down until you see the download files listed below:

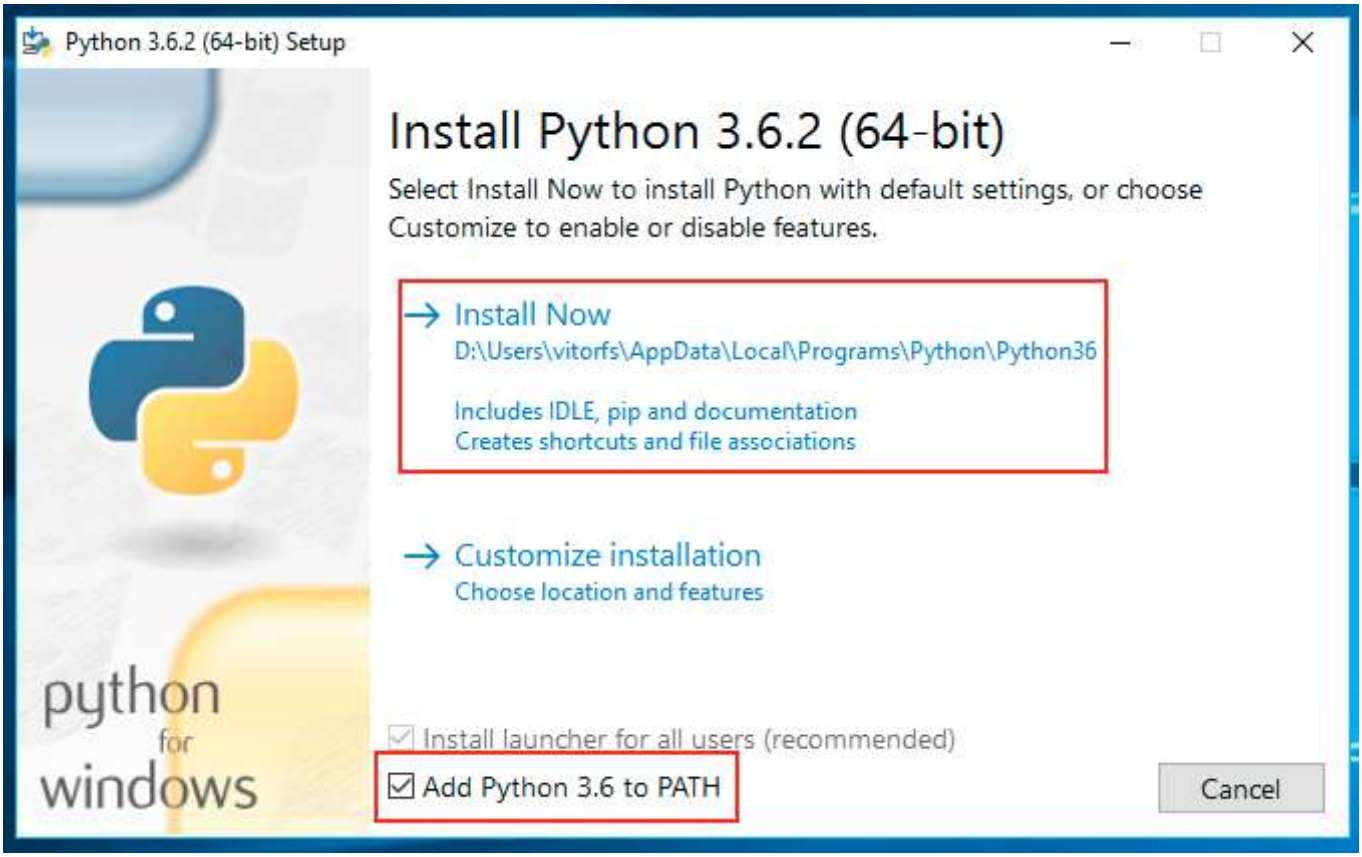


Pick the right version accordingly to your Windows distribution. If you are not sure which one is the right for you, the chances are you want to download the **Windows x86-64 executable installer** version.

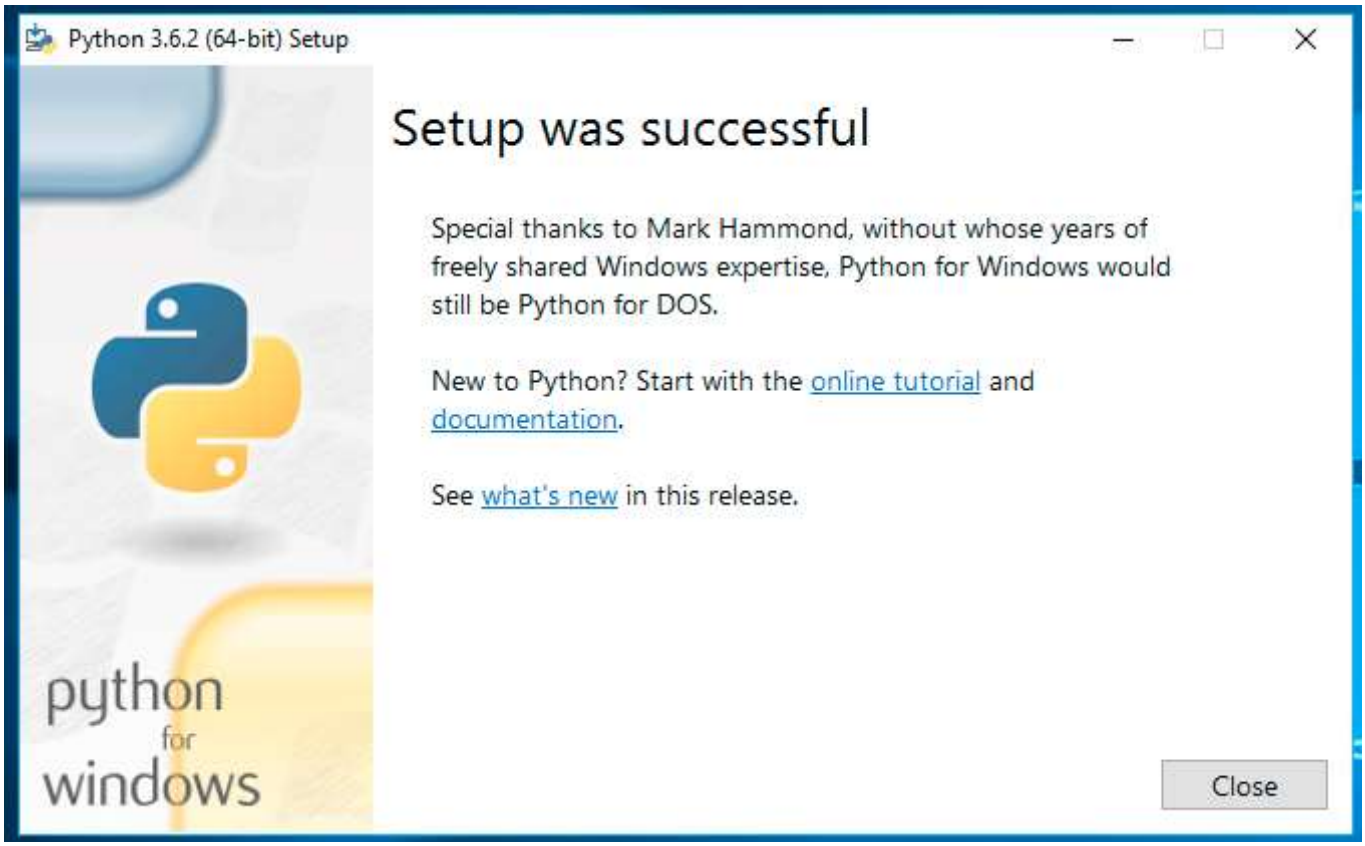
Go to your Downloads directory, right click on the installer and click on **Run as administrator**.



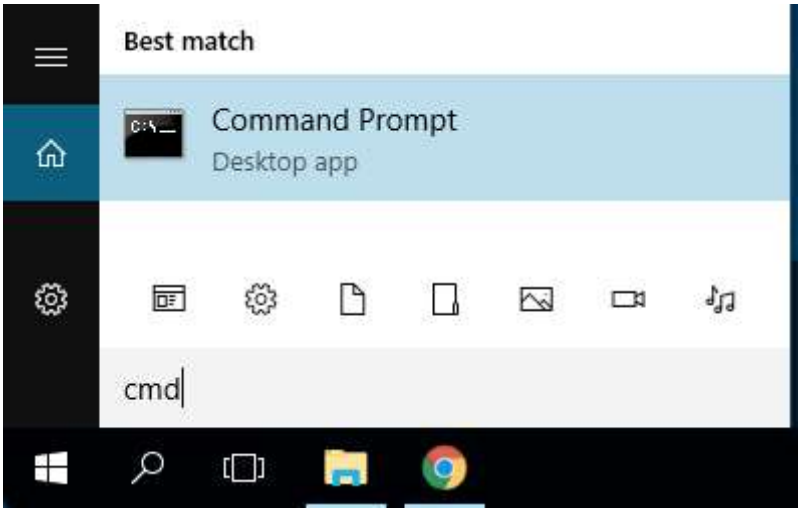
Make sure you check the option **Add Python 3.6 to PATH** and click on the **Install Now** option.



After the installation completes, you should see the following screen:



Now search for the **Command Prompt** program and open it:

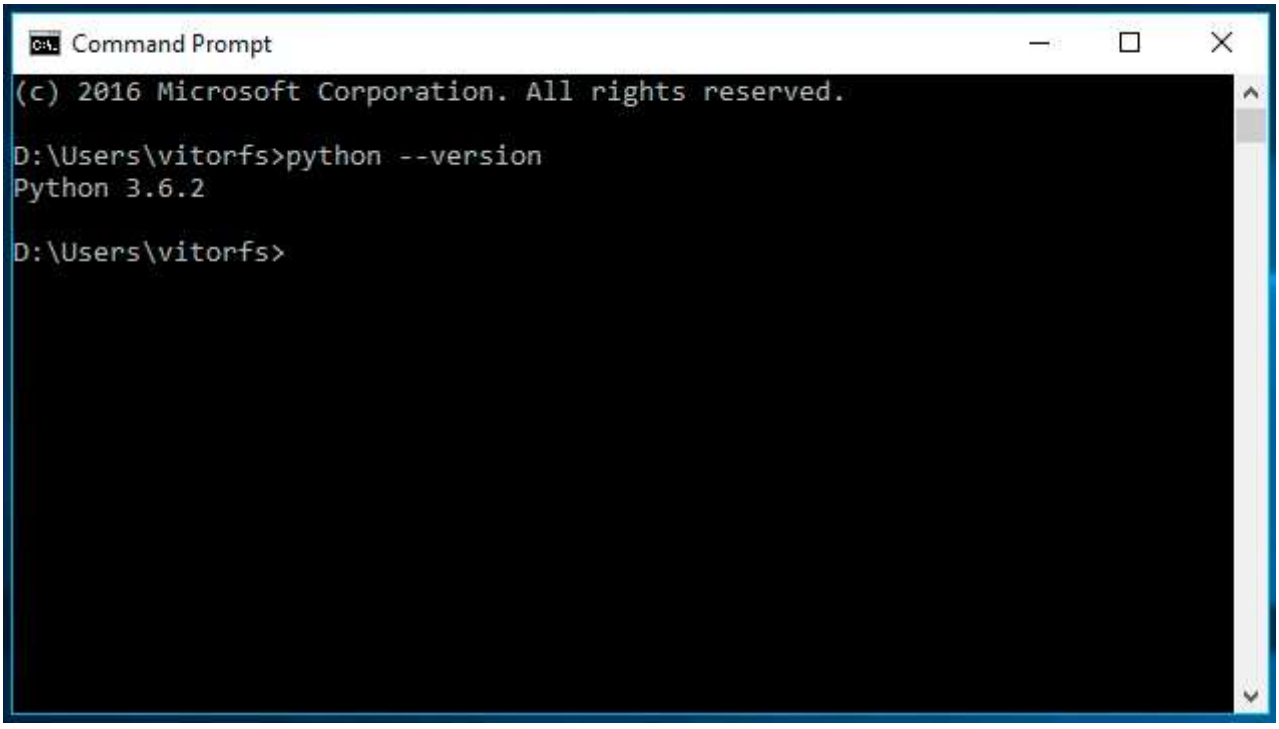


To test if everything is working fine so far, type following command:

```
python --version
```

As an output you should see:

```
Python 3.6.2
```



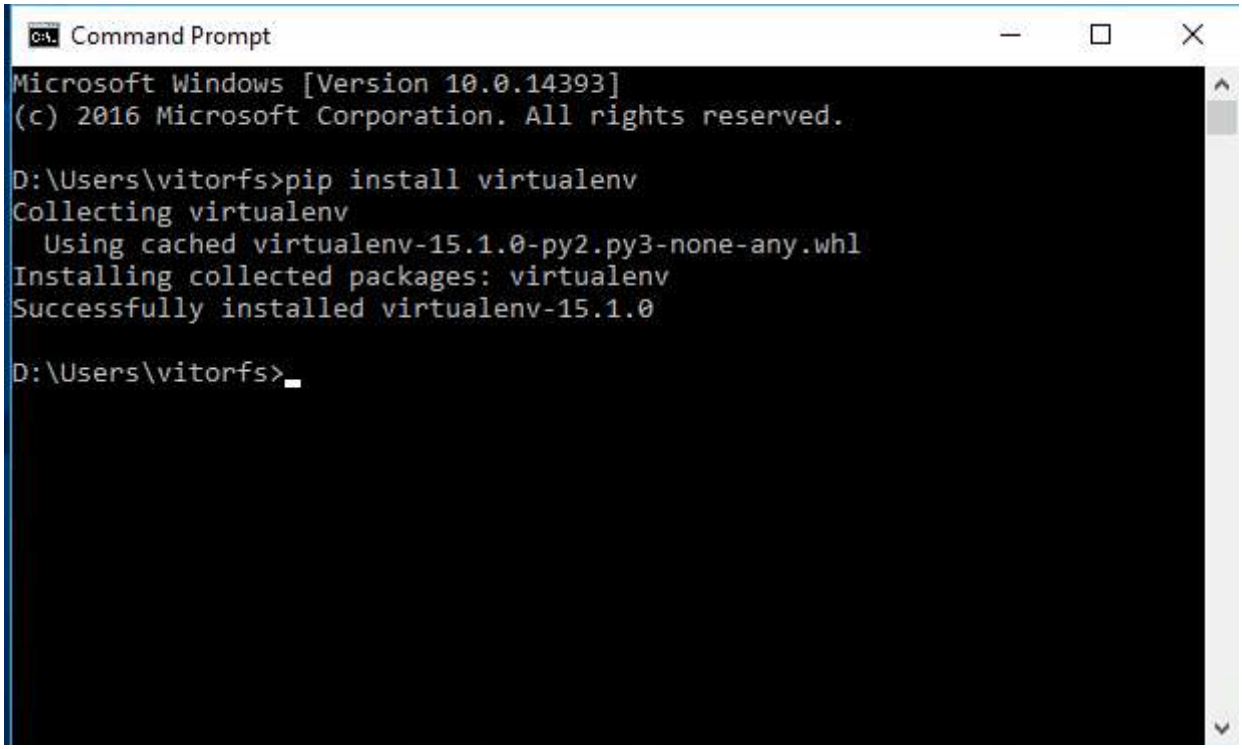
Great, Python is up and running. Next step: Virtual Environments!

Installing Virtualenv

For the next step, we are going to use **pip**, a tool to manage and install Python packages, to install **virtualenv**.

In the Command Prompt, execute the command below:

```
pip install virtualenv
```



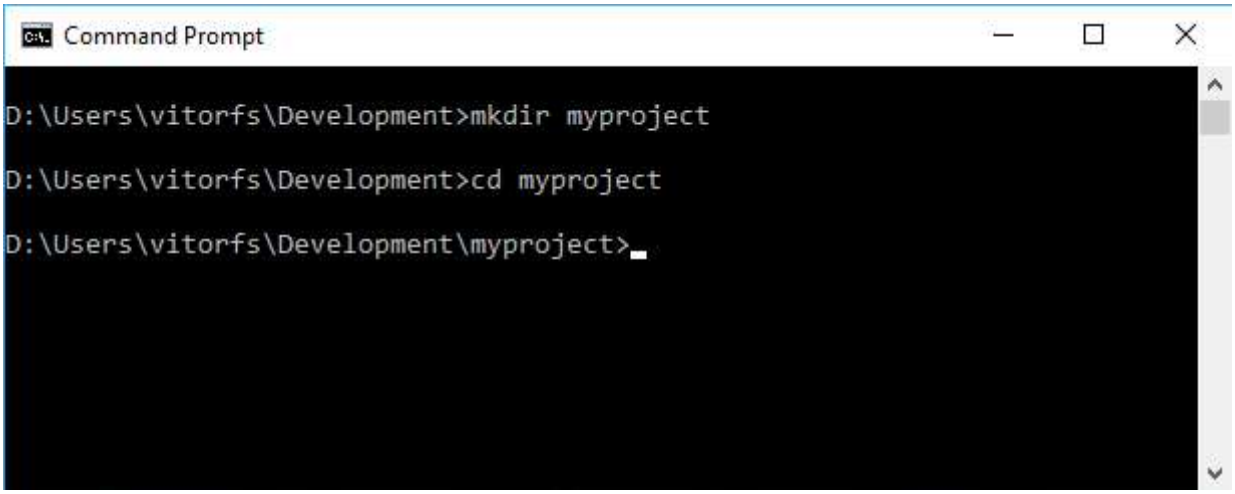
So far the installations that we performed was system-wide. From now on, everything we install, including Django itself, will be installed inside a Virtual Environment.

Think of it like this: for each Django project you start, you will first create a Virtual Environment for it. It's like having a sandbox for each Django project. So you can play around, install packages, uninstall packages without breaking anything.

I like to create a folder named **Development** on my personal computer. Then, I use it to organize all my projects and websites. But you can follow the next steps creating the directories wherever it feels right for you.

Usually, I start by creating a new folder with the project name inside my **Development** folder. Since this is going to be our very first project, we don't need to pick a fancy name or anything. For now, we can call it **myproject**.

```
mkdir myproject
cd myproject
```

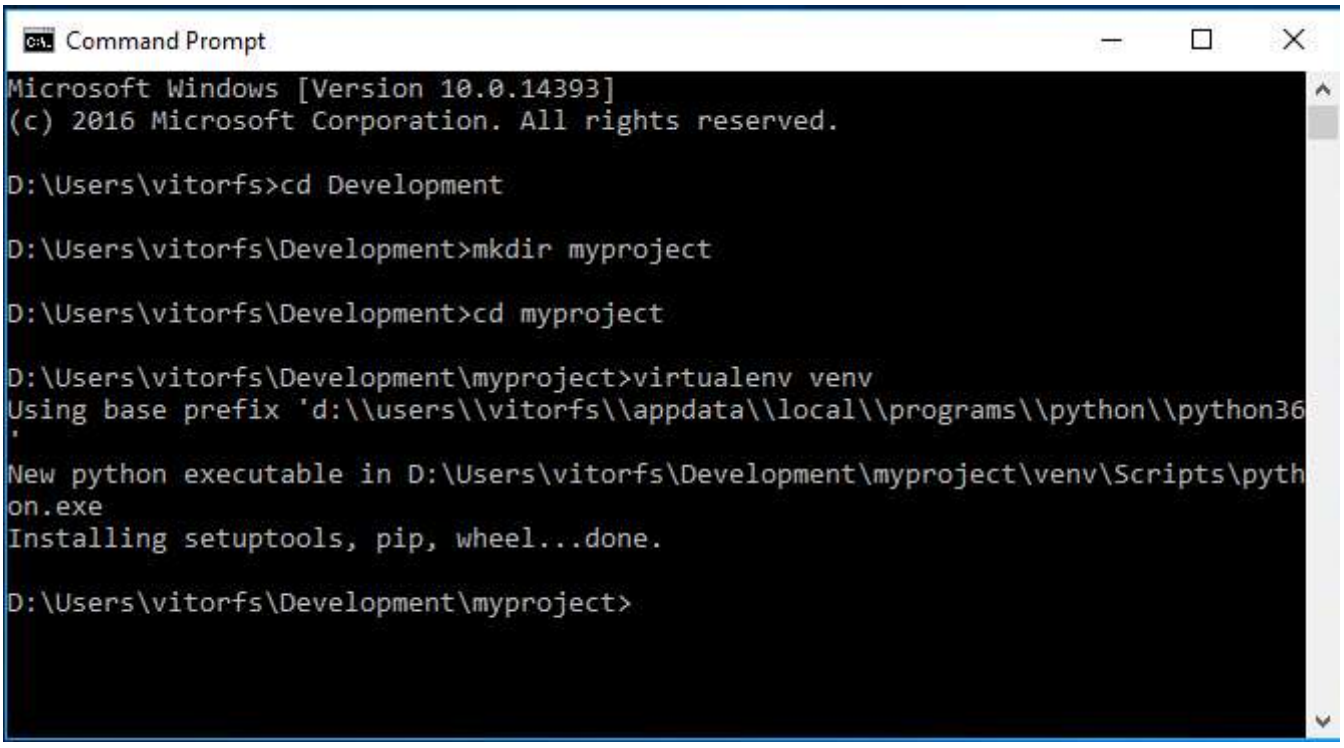


This folder is the higher level directory that will store all the files and things related to our Django project, including its virtual environment.

So let's start by creating our very first virtual environment and installing Django.

Inside the **myproject** folder:

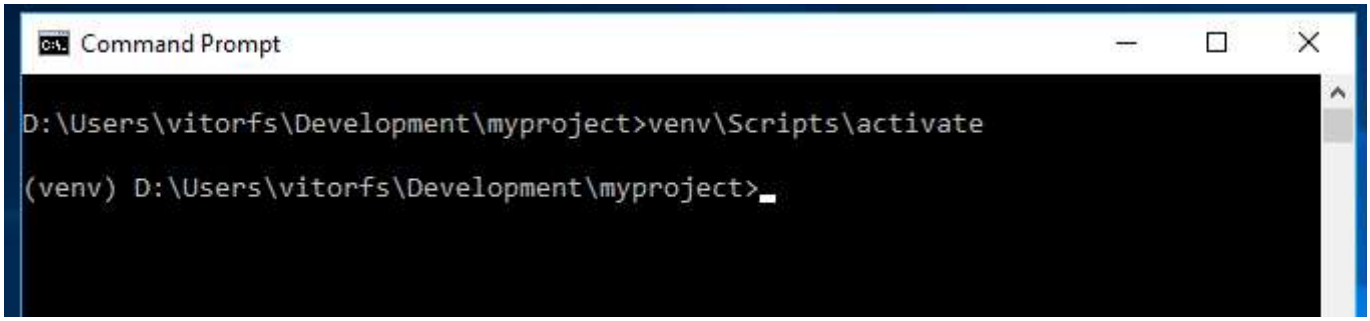
```
virtualenv venv
```



Our virtual environment is created. Now before we start using it, we need to activate:

```
venv\Scripts\activate
```

You will know it worked if you see **(venv)** in front of the command line, like this:



Let's try to understand what happened here. We created a special folder named **venv**. It contains a copy of Python inside this folder. After we activated the **venv** environment, when we run the `python` command, it will use our local copy, stored inside **venv**, instead of the other one we installed earlier.

Another important thing is that the **pip** program is already installed as well, and when we use it to install a Python package, like Django, it will be installed *inside* the **venv** environment.

By the way, to deactivate the **venv** run the command below:

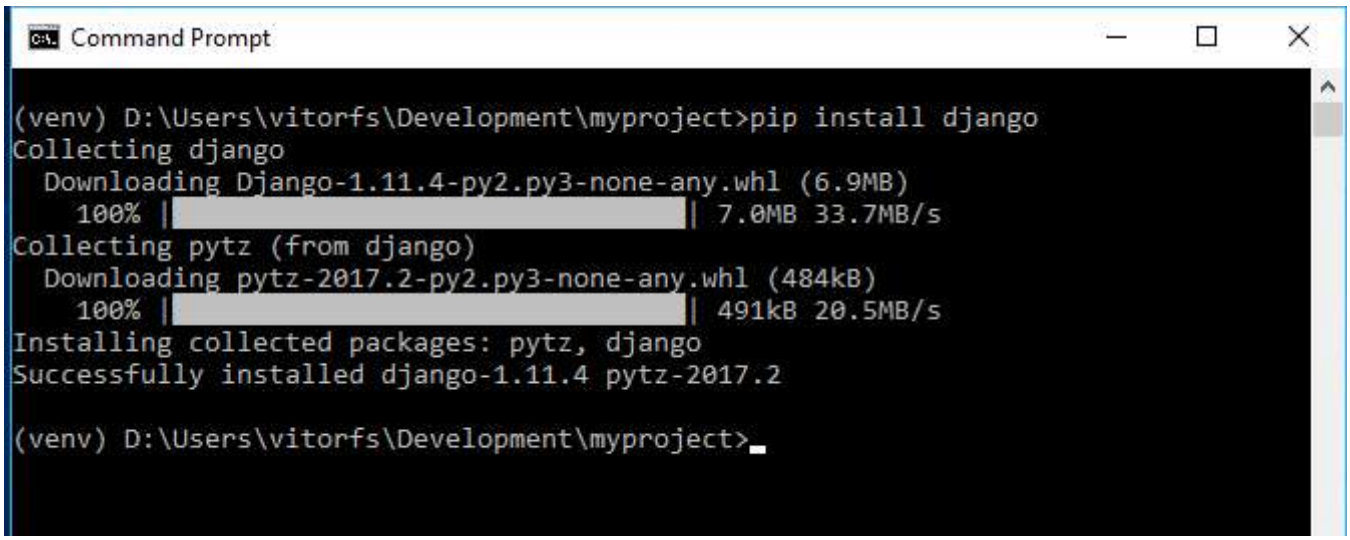
```
venv\Scripts\deactivate.bat
```

But let's keep it activated for the next steps.

Installing Django 1.11.4

It's very straightforward. Now that we have the **venv** activated, run the following command to install Django:

```
pip install django
```



We are all set up now!

Starting a New Project

To start a new Django project, run the command below:

```
django-admin startproject myproject
```

The command-line utility **django-admin** is automatically installed with Django.

After we run the command above, it will generate the base folder structure for a Django project.

Right now, our **myproject** directory looks like this:

```
myproject/                  <-- higher level folder
 |-- myproject/             <-- django project folder
 |    |-- myproject/
 |    |    |-- __init__.py
 |    |    |-- settings.py
 |    |    |-- urls.py
 |    |    |-- wsgi.py
 |    +-- manage.py
 +-- venv/                  <-- virtual environment folder
```

Our initial project structure is composed of five files:

- **manage.py**: a shortcut to use the **django-admin** command-line utility. It's used to run management commands related to our project. We will use it to run the development server, run tests, create migrations and much more.
- **__init__.py**: this empty file tells Python that this folder is a Python package.
- **settings.py**: this file contains all the project's configuration. We will refer to this file all the time!
- **urls.py**: this file is responsible for mapping the routes and paths in our project. For example, if you want to show something in the URL `/about/`, you have to map it here first.

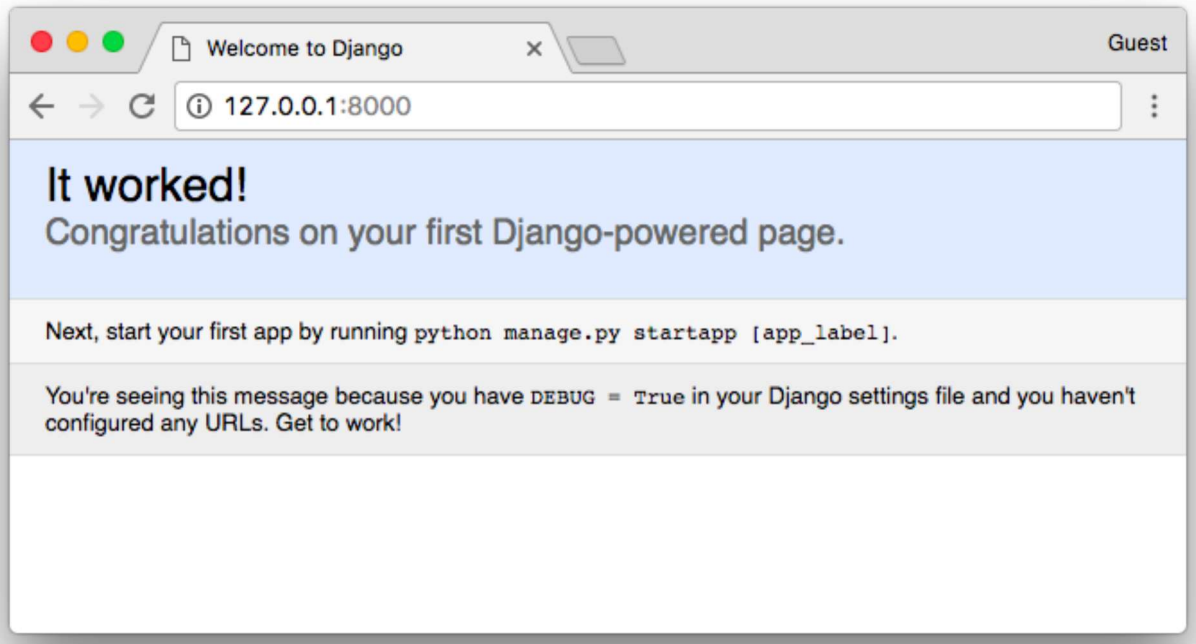
- **wsgi.py**: this file is a simple gateway interface used for deployment. You don't have to bother about it. Just let it be for now.

Django comes with a simple web server installed. It's very convenient during the development, so we don't have to install anything else to run the project locally. We can test it by executing the command:

```
python manage.py runserver
```

For now, you can ignore the migration errors; we will get to that later.

Now open the following URL in a Web browser: **http://127.0.0.1:8000** and you should see the following page:



Hit CTRL + BREAK to stop the development server.

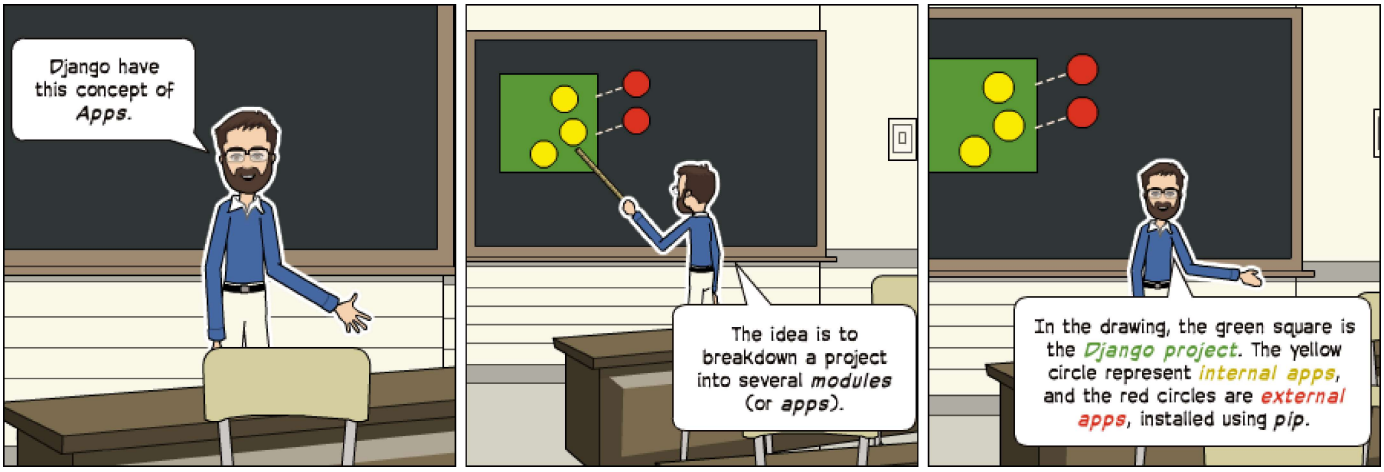
Django Apps

In the Django philosophy we have two important concepts:

- **app**: is a Web application that does something. An app usually is composed of a set of models (database tables), views, templates, tests.

- **project**: is a collection of configurations and apps. One project can be composed of multiple apps, or a single app.

It's important to note that you can't run a Django **app** without a **project**. Simple websites like a blog can be written entirely inside a single app, which could be named **blog** or **weblog** for example.



It's a way to organize the source code. In the beginning, it's not very trivial to determine what is an app or what is not. How to organize the code and so on. But don't worry much about that right now! Let's first get comfortable with Django's API and the fundamentals.

Alright! So, to illustrate let's create a simple Web Forum or Discussion Board. To create our first app, go to the directory where the **manage.py** file is and executes the following command:

```
django-admin startapp boards
```

Notice that we used the command **startapp** this time.

This will give us the following directory structure:

```
myproject/
 |-- myproject/
 |    |-- boards/                <-- our new django app!
 |    |    |-- migrations/
 |    |    |    +-- __init__.py
 |    |    |-- __init__.py
 |    |    |-- admin.py
 |    |    |-- apps.py
 |    |    |-- models.py
 |    |    |-- tests.py
 |    |    +-- views.py
 |    |-- myproject/
 |    |    |-- __init__.py
 |    |    |-- settings.py
 |    |    |-- urls.py
```

```
|   |   |-- wsgi.py
|   +-- manage.py
+-- venv/
```

So, let's first explore what each file does:

- **migrations/**: here Django store some files to keep track of the changes you create in the **models.py** file, so to keep the database and the **models.py** synchronized.
- **admin.py**: this is a configuration file for a built-in Django app called **Django Admin**.
- **apps.py**: this is a configuration file of the app itself.
- **models.py**: here is where we define the entities of our Web application. The models are translated automatically by Django into database tables.
- **tests.py**: this file is used to write unit tests for the app.
- **views.py**: this is the file where we handle the request/response cycle of our Web application.

Now that we created our first app, let's configure our project to *use* it.

To do that, open the **settings.py** and try to find the `INSTALLED_APPS` variable:

settings.py

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

As you can see, Django already come with 6 built-in apps installed. They offer common functionalities that most Web applications need, like authentication, sessions, static files management (images, javascripts, css, etc.) and so on.

We will explore those apps as we progress in this tutorial series. But for now, let them be and just add our **boards** app to the list of `INSTALLED_APPS`:

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

```
    'django.contrib.staticfiles',

    'boards',
]
```

Using the analogy of the square and circles from the previous comic, the yellow circle would be our **boards** app, and the **django.contrib.admin**, **django.contrib.auth**, etc, would be the red circles.

Hello, World!

Let's write our first **view**. We will explore it in great detail in the next tutorial. But for now, let's just experiment how it looks like to create a new page with Django.

Open the **views.py** file inside the **boards** app, and add the following code:

views.py

```
from django.http import HttpResponse

def home(request):
    return HttpResponse('Hello, World!')
```

Views are Python functions that receive an `HttpRequest` object and returns an `HttpResponse` object. Receive a *request* as a parameter and returns a *response* as a result. That's the flow you have to keep in mind!

So, here we defined a simple view called **home** which simply returns a message saying **Hello, World!**.

Now we have to tell Django *when* to serve this view. It's done inside the **urls.py** file:

urls.py

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

from boards import views

urlpatterns = [
    url(r'^$', views.home, name='home'),
    url(r'^admin/', admin.site.urls),
]
```

If you compare the snippet above with your **urls.py** file, you will notice I added the following new line: `url(r'^$', views.home, name='home')` and imported the **views** module from our app **boards** using `from boards import views`.

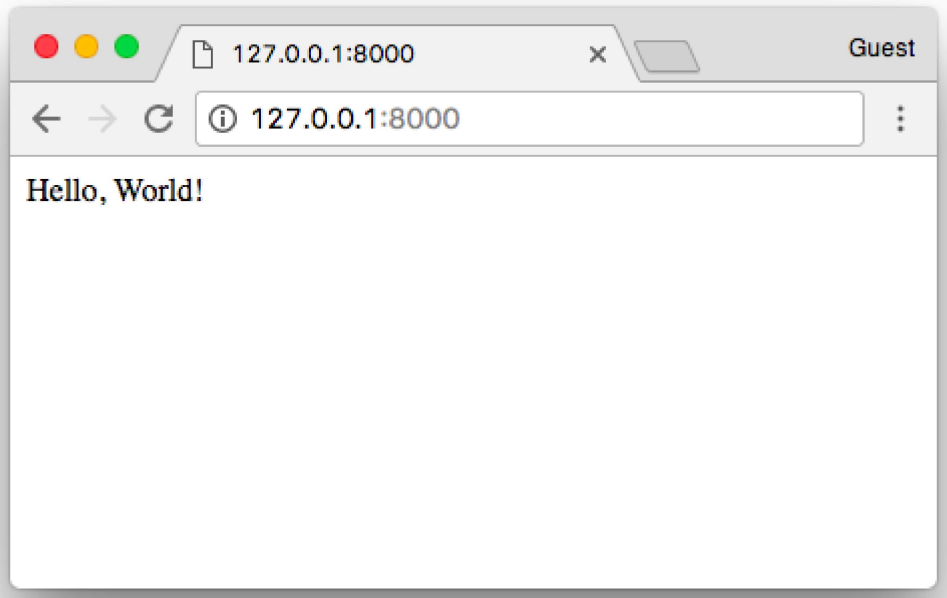
As I mentioned before, we will explore those concepts in great detail later on.

But for now, Django works with **regex** to match the requested URL. For our **home** view, I'm using the `^$` regex, which will match an empty path, which is the homepage (this url: **http://127.0.0.1:8000**). If I wanted to match the URL **http://127.0.0.1:8000/homepage/**, my url would be: `url(r'^homepage/$', views.home, name='home')`.

Let's see what happen:

```
python manage.py runserver
```

In a Web browser, open the http://127.0.0.1:8000 URL:



That's it! You just created your very first view.

Conclusions

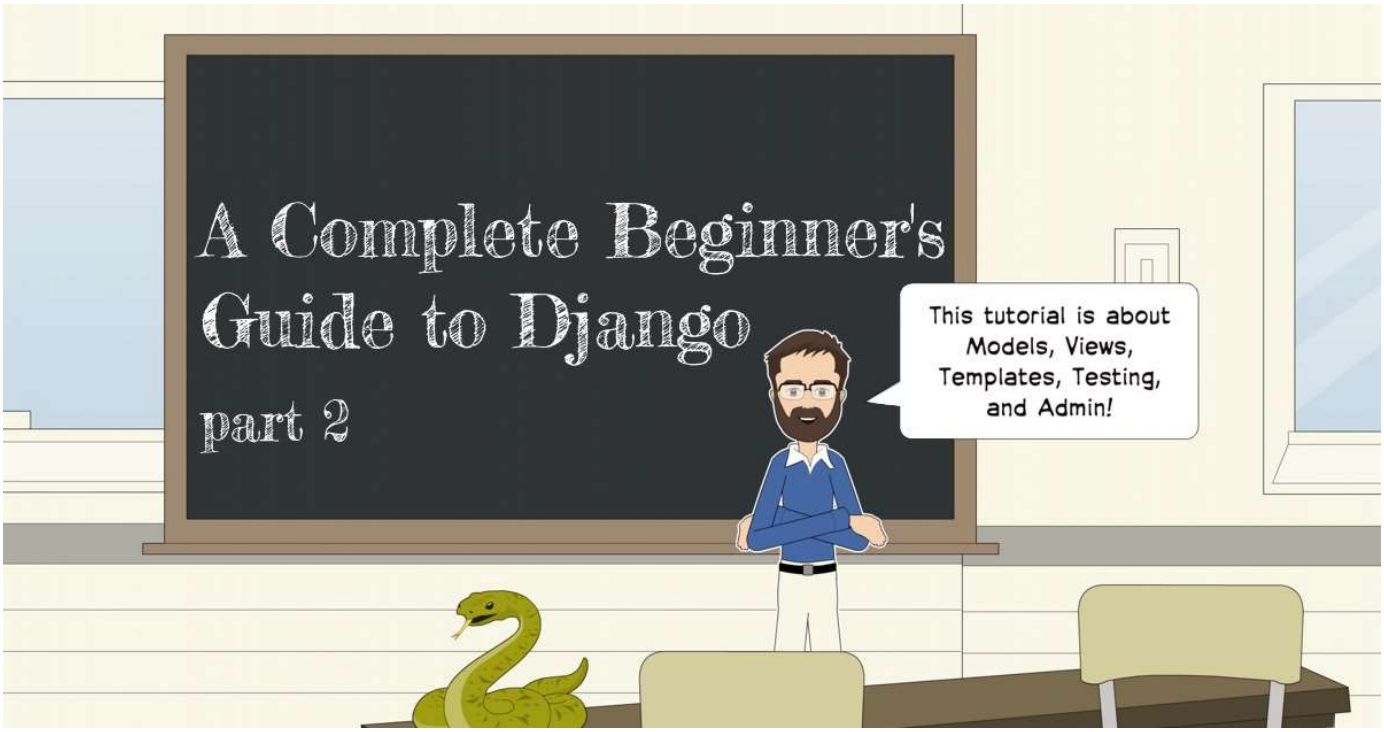
That was the first part of this tutorial series. In this tutorial, we learned how to install the latest Python version and how to setup the development environment. We also had an introduction to virtual environments and started our very first Django project and already created our initial app.

I hope you enjoyed the first part! The second part is coming out next week, on Sep 11, 2017. It's going to be about models, views, templates, and URLs. We will explore together all the Django fundamentals! If you would like to get notified when the second part is out, you can subscribe to our mailing list.

Just so we can stay on the same page, I made the source code available on GitHub. The current state of the project can be found under the release tag **v0.1-lw**. The link below will take you to the right place:

https://github.com/sibtc/django-beginners-guide/tree/v0.1-lw

← Tutorial Series Index



Part 2 - Fundamentals →

python django guide beginners app project

Share this post

109 Comments Simple is Better Than Complex 1 Login

Recommend 24 Tweet Share Sort by Best

Join the discussion…

LOG IN WITH OR SIGN UP WITH DISQUS ?

Name

gamesbook • 2 years ago

Hi Vitor - this is much needed. But *please*, do incorporate actual, real, comprehensive tests (along with a method to write and update them) as soon as possible. This is what is lacking (or just an add-on) in most other tutorials out there. Remember the words of one of the the founders of Django - "Code without tests is broken as designed" ~ Jacob Kaplan-Moss

9 • Reply • Share ›

Vitor Freitas Mod gamesbook • 2 years ago

Thanks a lot! This was a great advice.
I will definitively keep that in mind! In fact I'm reviewing the next tutorial so to introduce it earlier and make it an integral part of the tutorial series, rather than just having a section about testing.
I really appreciate your feedback! Thanks!
7 • Reply • Share ›

glenfant • 2 years ago
Congratulation for this valuable tutorial for newcomers, and perhaps re-comers.
Just mentioning that Python 3.6 comes with the "venv" package in the stdlib. So you don't need to install virtualenv unless I missed something. Just replace "virtualenv venv -p python3" with "python3 -m venv venv"
5 • Reply • Share ›

Vitor Freitas Mod → glenfant • 2 years ago
Thanks a lot for the heads up!
I mentioned to another commenter here, I wasn't aware of the `venv` package on Python 3.6, I've been using `virtualenv` for a long time now, and never really explored the other options hehe :-)
I will just make sure everything works fine in all platforms etc and I'll update the article later on :-)
It would be much better to simply use the `venv`, one less thing to install. Much better!
Thanks a lot for sharing!
• Reply • Share ›

leahlang4d • a year ago
If you are having trouble with the url.py part for django 2.0.2, the new version of django does not use regular expressions so therefor the code in url.py should be

```
urlpatterns = [
path('', views.home, name='home'),
path('admin/', admin.site.urls),
]
```

At least this is what worked for me.
4 • Reply • Share ›

sonnesen → leahlang4d • a year ago
Hello **@leahlang4d**,

Just one correction. Django 2 works with regular expression in urls.py. You have to use the re_path module.
• Reply • Share ›

Wilver González • a year ago
My best tutorial about Django ! Thanks.
3 • Reply • Share ›

Axoska • 2 years ago
This is great I always get tutorials that are at least a year old. This time I won't be a future

This is great I always get tutorials that are at least a year old. This time I won't be a future reader.

Thanks.

3 ^ v • Reply • Share ›

Anita Kiju • 2 years ago

I guess a small point is left out. Well, a simple one but might be a matter of headache for 'the beginners'. It's just that before running the server, we should be in the django project folder 'myproject' , within the higher level directory 'myproject' in the command prompt.

2 ^ v • Reply • Share ›

Mitra ➚ Anita Kiju • a year ago

that's what I ran into then noticed I can't use manage.py if the command is not running under myproject/myproject

^ v • Reply • Share ›

Robert Hafner • 2 years ago

Since you're using python3 wouldn't it make more sense to use `venv` rather than `virtualenv`?

2 ^ v • Reply • Share ›

Vitor Freitas Mod ➚ Robert Hafner • 2 years ago

Hey Robert!
Thanks for the heads up! I have to be honest with you -- I've been using `virtualenv` for a long time now, and never really explored other options.
And as you mentioned, since I'm writing it focused on Python 3, using `venv` would indeed be more natural. One less thing to install!
I will try it out on all platforms (mac/windows/linux) and see if everything works smoothly I will update the article.
Thanks for the great comment! :-)

^ v • Reply • Share ›

Douglas Barbosa dos Santos • a year ago

Congrats for this amazing tutorial. But I would like to know, if you will upgrade this tutorial to a 2.0 version of Django?

2 ^ v • Reply • Share ›

Gerald Brown • 2 years ago

django-admin startproject myproject

I just discovered today that if a space+period is placed at the end of the above command it makes the project tree
much neater as the project with the manage.py file is created in the current directory.

All other tutorials that I have seen also omit the space+period in their commands.

Previously I was wondering why there were so many directories with the same name and the space+period
eliminates this problem.

BTW Docker shouts at you if the space+period is omitted in a build command.

2 ^ v • Reply • Share ›

Pelle Gøeg • a year ago

Hi Vitor,

I your latest tutorials, you started using django 2.0, are you going to update this tutorial, maybe with pointing out the differences between versions? - Pelle

1 ^ v • Reply • Share ›

dbinott ➚ Pelle Gøeg • a year ago

be really great if you could.

^ v • Reply • Share ›

Dick Liu • 2 years ago

nice and valuable

1 ^ v • Reply • Share ›

AldyAhsn • 2 years ago

Great tutorial! Congratz **@Vitor Freitas**

1 ^ v • Reply • Share ›

P. H • 2 years ago

A small spelling mistake... important should be imported in the url.py section, in the text "important the views module from our app boards". Other than that great tutorial and can't wait to see the rest of it.

1 ^ v • Reply • Share ›

Vitor Freitas Mod ➚ P. H • 2 years ago

Thanks Philip! Fixed it :-)

^ v • Reply • Share ›

mircea prodan • 2 years ago

Great! Congratulation Vitor! Maybe, after you finish your tutorials it is good to write an ebook with all of this. I buy for example, a book like this. Maybe in yours future tutorials you show us how to:

- add comments system in a Django website
- add images
- add paginations
- add markdown (or anything to add HTML tag and link and so on in a blog post/page).

In a word, everything that means a complete website

1 ^ v • Reply • Share ›

Vitor Freitas Mod ➚ mircea prodan • 2 years ago

Thanks a lot, Mircea!

That's a great idea! I will definitively look into it. Maybe I can even expand the content in a ebook :-)

I will cover those topics, comments, image upload, pagination and so on!

Thanks again for your comments!

Reply • Share ›

Gerald Brown ➔ Vitor Freitas • 2 years ago

I would also be interested in an ebook of these tutorials.

Reply • Share ›

Gautam Mandewalker • 2 years ago • edited

Very nice presentation Victor, Can't wait to see other tutorials. Thanks.

1 Reply • Share ›

Vitor Freitas Mod ➔ Gautam Mandewalker • 2 years ago

Thanks Gautam! Glad to hear you are liking it! :-)

Reply • Share ›

Oleg Kleschynov • 2 years ago

Good job! Great tutorial!

1 Reply • Share ›

Vitor Freitas Mod ➔ Oleg Kleschynov • 2 years ago

Thanks, Oleg! :-)

Reply • Share ›

Oleg Kleschynov ➔ Vitor Freitas • 2 years ago

Vitor, are you planning to consider using jinja2 in templates or something about caching in this series or, maybe, in further articles? Thanks!

1 Reply • Share ›

Show more replies

Natasha Shankar • 6 days ago • edited

I'm having an issue after the url.py. It's giving me an error in command prompt that there is no such attribute called 'home' and hence the web page isn't loading either.

Update: I realized that there are two folders called boards and two different files for each called views. If you put this code into myproject -> myproject -> boards -> views then it works. If you put it in myproject -> boards -> views it won't work and you're left with an error.

Reply • Share ›

Monsoon • 2 months ago

Although not updated, this is very easy to follow! Hopefully it is like that till the end. Couldn't help but chime in that I hate *foo/bar* examples too!

Reply • Share ›

Anuj • 3 months ago

Hi Vitor Freitas, thanks for writing such a helpful post for me and for many students like me who is just trying to leaarn from scratch as a beginner. I am really thankful to you.

Reply • Share ›

Xian Liu • 5 months ago

The best Django tutorial I ever read. Thanks a lot!

Reply • Share ›

sivaprasad • 5 months ago

Really you gave what am searching as a beginner thanks a lot from bottom of my heart .....

Reply • Share ›

Nguyen Kim Son • 5 months ago

Hi Vitor, great tutorial, I really enjoy reading it! Django 2 has changed the way to handle "urlpatterns" which is more accessible than the regex based one, would be great if the tutorial can be updated to match that :)!

Reply • Share ›

Niels Kampert • 6 months ago

Hi Victor, great tutorial. The best i have found. Thanks a lot
Niels, from the netherlands.

Reply • Share ›

Laura Malt Schneiderman • 6 months ago

I have gotten to the point where I type python manage.py runserver. It says "Starting development server at http://127.0.0.1:8000". I go to Chrome, look at http://127.0.0.1:8000 and I see This site can't be reached

127.0.0.1 refused to connect.
Try:
Checking the connection
Checking the proxy and the firewall
ERR_CONNECTION_REFUSED

I don't have a proxy. This is my pc hard drive. What am I doing wrong?

Reply • Share ›

Laura Malt Schneiderman ➚ Laura Malt Schneiderman • 6 months ago

I had to type in cmd: python manage.py runserver 127.0.0.1:8000

Reply • Share ›

Abhinav Mane • 6 months ago • edited

When i do python manage.py runserver it gives me the following error :-
During handling of the above exception, another exception occurred:

Traceback (most recent call last):
File "manage.py", line 15, in <module>
execute_from_command_line(sys.argv)
File "C:\Users\user\Anaconda3\lib\site-packages\django\core\management__init__.py", line 381, in execute_from_command_line
utility.execute()
File "C:\Users\user\Anaconda3\lib\site-packages\django\core\management__init__.py", line

375, in execute
self.fetch_command(subcommand).run_from_argv(self.argv)
File "C:\Users\user\Anaconda3\lib\site-packages\django\core\management\base.py", line 329, in run_from_argv
connections.close_all()
File "C:\Users\user\Anaconda3\lib\site-packages\django\db\utils.py", line 220, in close_all
for alias in self:
File "C:\Users\user\Anaconda3\lib\site-packages\django\db\utils.py", line 214, in __iter__

see more

• Reply • Share ›

Natasha Shankar ➚ Abhinav Mane • 6 days ago

Almost the same error occurred for me. How did you fix this (if you did)?

• Reply • Share ›

Aleksey • 9 months ago

Is this tutorial`s code still relevant for this time while Django 2.1 released already? Or it will be a lot of headache by updating code to Python 3.7 and Django 2.1 ?

• Reply • Share ›

hep93 • 9 months ago • edited

thanks for the great tutorial

• Reply • Share ›

Sam Watt • 10 months ago

Thanks for sharing an Informative blog with us, personally I like your blog. and also check https://www.botreetechnolog...

• Reply • Share ›

Shahid • a year ago

Please, upgrade Django 2 version.

• Reply • Share ›

lakshmi prasanna • a year ago

I am getting this error

• Reply • Share ›

lakshmi prasanna ➚ lakshmi prasanna • a year ago

• Reply • Share ›

Steev James • a year ago

The best Tutorial for Django !

• Reply • Share ›

이승우 • a year ago

this is my best tutorial

• Reply • Share ›

rick39guitarist • a year ago

Thanks Vitor, I am brand new to django and appreciate this information. Thanks!!!
Rick

• Reply • Share ›

kokserek • a year ago

Hi, to be honest, I followed this wonderful tutorial to the part 7 without problem via python 2.7. But in pt7 I see that I need python3 (to install decouple). I downloaded python3 via Homebrew and installed it successfully, but when I am trying to reach it via

$ python3 , I see: python3: posix_spawn: /Library/Frameworks/Python.framework/Versions/3.6/bin/python3.62.7: No such file or directory

and if I ask: $ python --version

I see: Python 2.7.10 (my previous version).

What should I do to be able to continue my work with python3 as prescribed here?

Thanks,

• Reply • Share ›

Casey Key • a year ago

Please create a tutorial for creating a website like this! I want to make tutorials too.

• Reply • Share ›

Load more comments

11 comments • 9 months ago

Vitor Freitas — Next video coming up I will be adding bootstrap 4 to the forms :D

Crispy Forms

22 comments • 8 months ago

Eric Kiser — First I love your tutorials on Django, now my question is how do you use "as_crispy_field" with a filefield or …

Popular Posts

How to Extend Django User Model

How to Setup a SSL Certificate on Nginx for a Django Application

How to Deploy a Django Application to Digital Ocean