

Writing Great Documentation

Jacob Kaplan-Moss
jacob@heroku.com

Co-BDFL

django

Director of Security

 **heroku**

My premise:
**great documentation
drives adoption.**

Python web: 2004

Albatross

Nevow

Spark

Aquarium

PSE

Spyce

Cheetah

PSP

Twisted

CherryPy

Pyrooxide

Wasp

Cymbeline

Quixote

Webware

Enamel

SkunkWeb

Zope

Python web: 2014



PyramidTM




Flask

web development,
one drop at a time

django

Pyramid Documentation — x

docs.pylonsproject.org/en/latest/docs/pyramid.html

 **Pyramid**

The Pyramid Project Documentation »

previous | next | index

[Edit me on GitHub](#)

Pyramid Documentation

Table Of Contents

- Pyramid Documentation
 - Getting Started
 - Main Documentation
 - Stable branch : version 1.5 (latest)
 - Development branch : upcoming 1.6 (master)
 - Tutorials and Cookbook
 - Recipes
 - Previous versions
 - Pyramid Add-ons
 - Supported Add-ons
 - Unsupported Add-Ons
 - Sample Pyramid Applications
 - Sample Pyramid Development Environments

Previous topic

The Pylons Project

Next topic

Getting Started

If you are new to Pyramid, you should start with the Getting Started guide.

- [Installation](#)
- [To see the documentation](#)
- [Pyramid Add-ons](#)
- [To get started with Pyramid](#)
- [Like to contribute?](#)
- [Need help?](#)

Main Documentation

Stable branch

- [Pyramid 1.5 \(latest\)](#)
- [Pyramid 1.4](#)

About Flask

Flask is a micro webdevelopment framework for Python.

Useful Links

[The Flask Website](#)
[Flask @ PyPI](#)
[Flask @ github](#)
[Issue Tracker](#)

Versions

[Development](#) (unstable)
[Flask 0.10.x](#) (stable)
[Flask 0.9.x](#)

Quick search

Welcome to Flask

Welcome to Flask's documentation. This documentation is divided into different parts. I recommend that you start with the [Installation](#) guide.

Installation

over to the [Installation](#) guide for the quickstart or the [FAQ](#) for more details with Flask documentation.

Flask depends on the WSGI to be able to document the application.

- [Jinja2](#)
- [Werkzeug](#)

User's Guide

This part of the documentation provides information with Flask.

- [Foreword](#)

Welcome to Flask — Flask x

flask.pocoo.org/docs/0.10/

Welcome to Flask

Welcome to Flask's documentation. This documentation is divided into different parts. I recommend that you start with the [Installation](#) guide.

Installation

over to the [Installation](#) guide for the quickstart or the [FAQ](#) for more details with Flask documentation.

Flask depends on the WSGI to be able to document the application.

- [Jinja2](#)
- [Werkzeug](#)

User's Guide

This part of the documentation provides information with Flask.

- [Foreword](#)

Django documentation | Django x

https://docs.djangoproject.com/en/1.7/



[Home](#) [Download](#) [Documentation](#) [Weblog](#) [Community](#) [Code](#)

Django documentation

Django documentation

Everything you need to know about Django.

First steps

Are you new to Django or to programming? This is the place to start!

- **From scratch:** [Overview](#) | [Installation](#)
- **Tutorial:** [Part 1](#) | [Part 2](#) | [Part 3](#) | [Part 4](#) | [Part 5](#) | [Part 6](#)
- **Advanced Tutorials:** [How to write reusable apps](#) | [Writing your first patch for Django](#)

The model layer

Django provides an abstraction layer (the "models") for structuring and manipulating the data of your Web application. Learn more about it below:

- **Models:** [Model syntax](#) | [Field types](#) | [Meta options](#)
- **QuerySets:** [Executing queries](#) | [QuerySet method reference](#) | [Query-related classes](#) | [Lookup expressions](#)
- **Model instances:** [Instance methods](#) | [Accessing related objects](#)
- **Migrations:** [Introduction to Migrations](#) | [Operations reference](#) | [SchemaEditor](#)
- **Advanced:** [Managers](#) | [Raw SQL](#) | [Transactions](#) | [Aggregation](#) | [Custom fields](#) | [Multiple databases](#) | [Custom lookups](#)
- **Other:** [Supported databases](#) | [Legacy databases](#) | [Providing initial data](#) | [Optimize database access](#)

Getting help

Having trouble? We'd like to help!

- Try the [FAQ](#) – it's got answers to many common questions.
- Looking for specific information? Try the [Index](#), [Module Index](#) or the [detailed table of contents](#).
- Search for information in the archives of the [django-users](#) mailing list, or [post a question](#).
- Ask a question in the [#django IRC channel](#), or search the [IRC logs](#) to see if it's been asked before.
- Report bugs with Django in our [ticket tracker](#).

Search

Version: [Django 1.7](#)

Browse

- Prev: [Django documentation contents](#)
- Next: [Getting started](#)
- [Table of contents](#)
- [General Index](#)
- [Python Module Index](#)

You are here:

- [Django 1.7 documentation](#)
 - Django documentation

Download:

Offline (Django 1.7): [HTML](#) | [PDF](#) | [ePub](#)
Provided by [Read the Docs](#).

django

81

121,000 lines of English

In Search of Lost Time

1,500,000

Django

566,000

Infinite Jest

484,000

New Testament

180,000

Your first manuscript

60,000

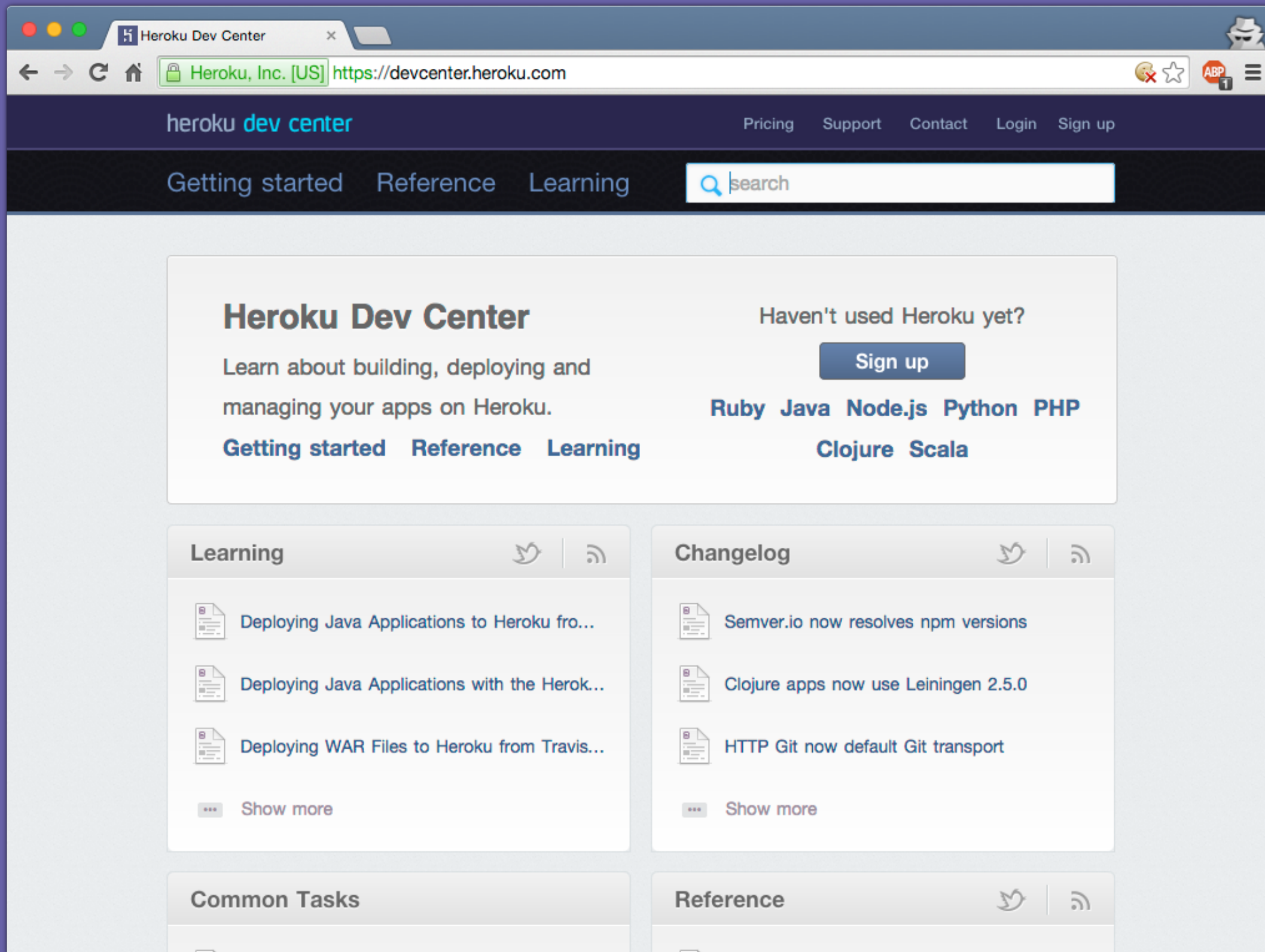
“The **documentation and community are second to none.**”

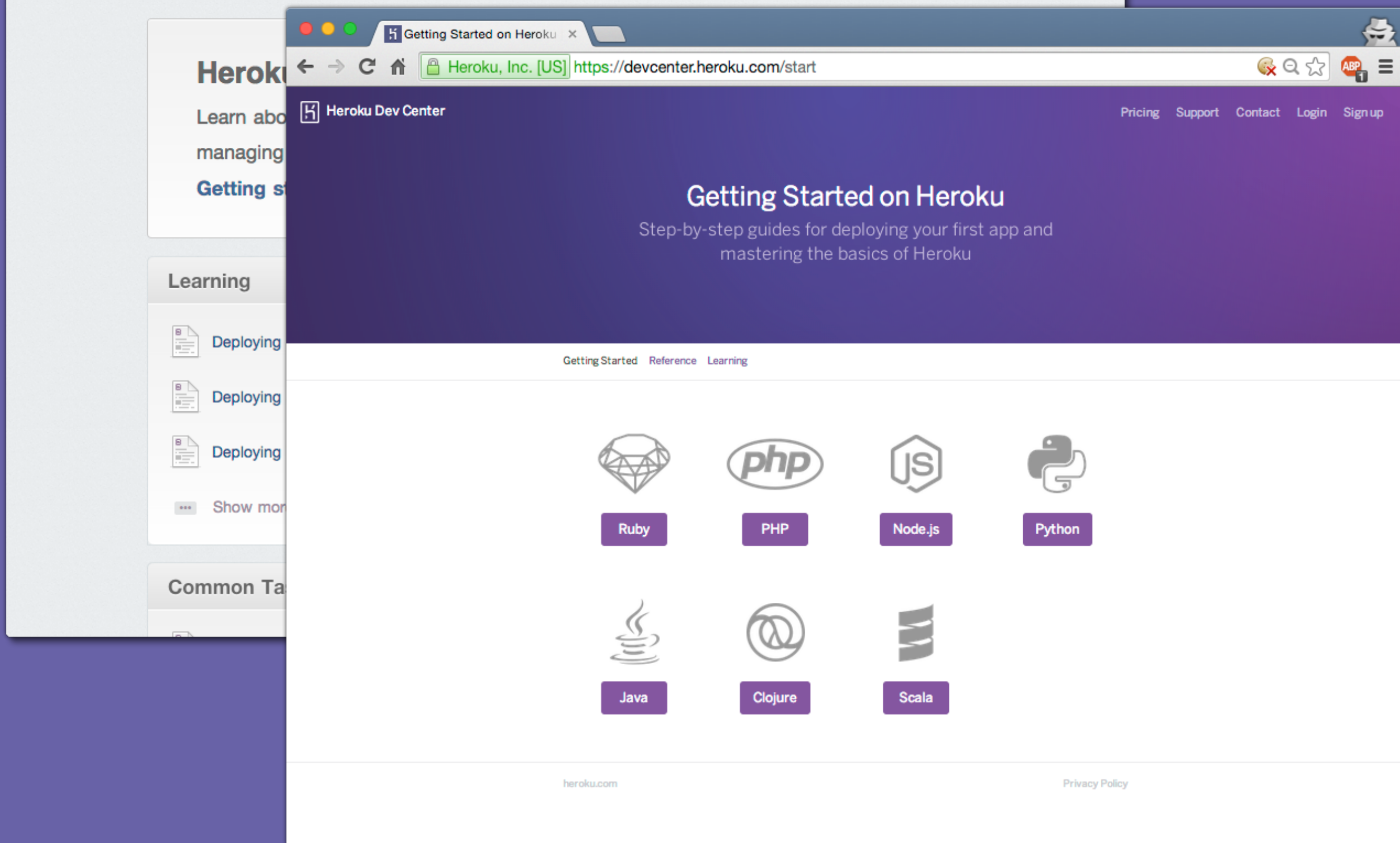
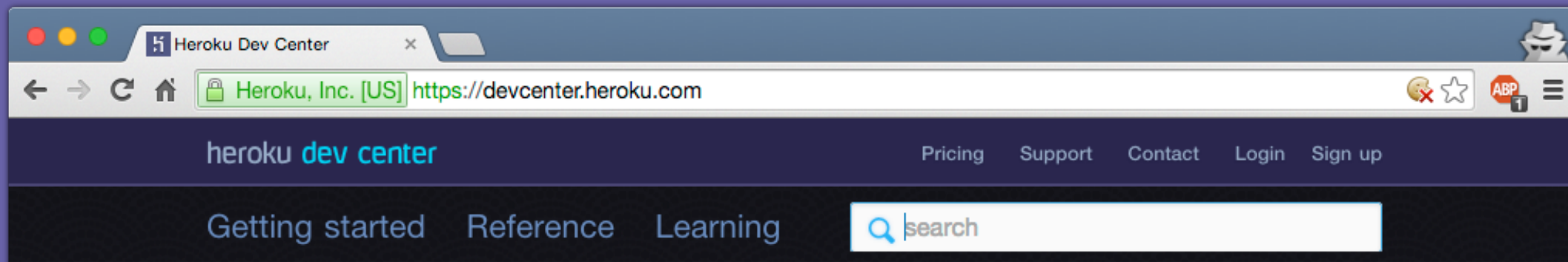
“[W]e’ve found that people ...can get up-to-speed relatively quickly thanks to the **excellent documentation...**”

“Django ... provides an **excellent developer experience**, with **great documentation** and tutorials...”

“Our initial choice ... was based on the **strength of the Django community and documentation...**”

“Productive development, **good documentation**, flexibility, and it just works.”







DOCUMENTATION

Getting Started

- Embedded Form
- Custom Forms
- Mobile Apps
- Charging Cards
- Sending Transfers
- Your Account

REFERENCES

- Stripe.js
- Checkout
- Webhooks
- Testing
- Examples
- API Upgrades
- API Libraries
- Full API Reference

SUBSCRIPTIONS

- Overview
- Getting started
- Integration guide

CONNECT

- Overview
- Getting Started
- Integrating OAuth

Getting Started

Below you'll find tutorials that will teach you how to use Stripe, and reference documentation for all the moving parts.



On your website

Start accepting payments on your website with our JavaScript libraries. [Learn more](#)



In your mobile app

We have toolkits for native iPhone and Android applications. [Learn more](#)

Stripe plugins for 3rd party software

As well as the official API libraries listed above, there are a number of third-party plugins and libraries built by our community, such as for Wordpress and Drupal. [Learn more](#)



Use Stripe with other services

Stripe has lots of third-party integrations which require no programming, hosting or complicated setup on your behalf, such as Shopify and Wufoo. [Learn more](#)





DOCUMENTATION

Getting Started

Embedded Form

Custom Forms

Mobile Apps

Charging Cards

Sending Transfers

Your Account

REFERENCES

Stripe.js

Checkout

Webhooks

Testing

Examples

API Upgrades

API Libraries

Full API Reference

SUBSCRIPTIONS

Overview

Getting started

Integration guide

CONNECT

Overview

Getting Started

Integrating OAuth

Integrating Checkout

This tutorial helps you accept your first payment with Stripe. If you need help, check out our answers to [common questions](#) or chat live with other developers in #stripe on freenode.

An easy way to integrate Stripe is via the Checkout, which will take care of building forms, validating input, and securing your customers' card data.

The Checkout is fairly high level, so remember that you can still use Stripe with your **own forms** via [Stripe.js](#) if it doesn't fit your use-case.

Try clicking on the example below, filling in the form with one of Stripe's [test card numbers](#), `4242 4242 4242 4242`, any three digit CVC code, and a valid expiry date.



Here's an overview of what you'll accomplish in this tutorial:

1. Collect credit card information with the Checkout
2. Convert those details to what we call a single-use token
3. Send that token, with the rest of your form, to your server to create the actual charge

Step 1: Embed the Checkout

To get started, add the following code to your page.

Note

We have Checkout integration guides for several platforms and languages to get you up and running as quickly as possible.

- [Sinatra](#)
- [Rails](#)
- [Flask](#)
- [PHP](#)



DOCUMENTATION

Getting Started

Embedded Form

Custom Forms

Mobile Apps

Charging Cards

Sending Transfers

Your Account

REFERENCES

Stripe.js

Checkout

Webhooks

Testing

Examples

API Upgrades

API Libraries

Full API Reference

SUBSCRIPTIONS

Overview

Getting started

Integration guide

CONNECT

Overview

Getting Started

Integrating OAuth

Integrating Checkout

This tutorial helps you accept your first payment with Stripe. If you need help, check out our answers to [common questions](#) or chat live with other developers in #stripe on freenode.

An easy way to integrate Stripe is via the Checkout, which will take care of building forms, validating input, and securing your customers' card data.

The Checkout is fairly high level, so remember that you can still use Stripe with your **own forms** via [Stripe.js](#) if it doesn't fit your use-case.

Try clicking on the example below, filling in the form with one of Stripe's [test card numbers](#), `4242 4242 4242 4242`, any three digit CVC code, and a valid expiry date.



Here's an overview of what you'll accomplish in this tutorial:

1. Collect credit card information with the Checkout
2. Convert those details to what we call a single-use token
3. Send that token, with the rest of your form, to your server to create the actual charge

Step 1: Embed the Checkout

To get started, add the following code to your page.

Note

We have Checkout integration guides for several platforms and languages to get you up and running as quickly as possible.

- [Sinatra](#)
- [Rails](#)
- [Flask](#)
- [PHP](#)

Integrating Checkout

Stripe, Inc. [US]https://stripe.com/docs/tutorials/checkout

Full API Reference

SUBSCRIPTIONS

Overview

Getting started

Integration guide

CONNECT

Overview

Getting Started

Integrating OAuth

Collecting Fees

Shared Customers

Reference

FAQ

Getting Paid

Disputes

SSL

Security

MORE

Contact

Global Users

Gallery

Integrations

Here's an overview of what you'll accomplish in this tutorial:

1. Collect credit card information with the Checkout

2. Convert those details to what we call a single-use token

3. Send that token, with the rest of your form, to your server to create the actual charge

Step 1: Embed the Checkout

To get started, add the following code to your page.

```
<form action="" method="POST">
  <script
    src="https://checkout.stripe.com/checkout.js" class="stripe-button"
    data-key="pk_test_6pRNASCoBOKtIshFeQd4XMUh"
    data-amount="2000"
    data-name="Demo Site"
    data-description="2 widgets ($20.00)"
    data-image="/128x128.png">
  </script>
</form>
```

The key thing to notice is the `data-key` attribute we added on the script tag which identifies your website when communicating with Stripe. Note that we've pre-filled the example with your **test publishable** API key. Remember to replace the test key with your live key in production. You can get all your keys from [your account page](#), or find out about [livemode and testing](#).

An alternative to the blue button demonstrated above is to use your own [custom button with our JavaScript API](#). This allows you to use any HTML element or JavaScript event to open the payment overlay, as well as being able to specify dynamic arguments, such as custom amounts.

Read more about [Checkout's reference](#).

Step 2: Sending tokens to your server

Integrating Checkout

Stripe, Inc. [US]https://stripe.com/docs/tutorials/checkout

Full API Reference

SUBSCRIPTIONS

Overview

Getting started

Integration guide

CONNECT

Overview

Getting Started

Integrating OAuth

Collecting Fees

Shared Customers

Reference

FAQ

Getting Paid

Disputes

SSL

Security

MORE

Contact

Global Users

Gallery

Integrations

Here's an overview of what you'll accomplish in this tutorial:

1. Collect credit card information with the Checkout

2. Convert those details to what we call a single-use token

3. Send that token, with the rest of your form, to your server to create the actual charge

Step 1: Embed the Checkout

To get started, add the following code to your page.

```
<form action="" method="POST">
  <script
    src="https://checkout.stripe.com/checkout.js" class="stripe-button"
    data-key="pk_test_6pRNASCoBOKtIshFeQd4XMUh"
    data-amount="2000"
    data-name="Demo Site"
    data-description="2 widgets ($20.00)"
    data-image="/128x128.png">
  </script>
</form>
```

The key thing to notice is the `data-key` attribute we added on the script tag which identifies your website when communicating with Stripe. Note that we've pre-filled the example with your **test publishable** API key. Remember to replace the test key with your live key in production. You can get all your keys from [your account page](#), or find out about [livemode and testing](#).

An alternative to the blue button demonstrated above is to use your own [custom button with our JavaScript API](#). This allows you to use any HTML element or JavaScript event to open the payment overlay, as well as being able to specify dynamic arguments, such as custom amounts.

Read more about [Checkout's reference](#).

Step 2: Sending tokens to your server

“Wow, if the documentation is this good, **the product must be awesome!**”

Why do people read documentation?

Why do people read documentation?

Who should write documentation?

Why do people read documentation?

Who should write documentation?

What should we document?

Why do people read documentation?

Who should write documentation?

What should we document?

First contact - new users.

First contact - new users.

Education - new & existing users.

First contact - new users.

Education - new & existing users.

Support - experienced users.

First contact - new users.

Education - new & existing users.

Support - experienced users.

Troubleshooting - annoyed users.

First contact - new users.

Education - new & existing users.

Support - experienced users.

Troubleshooting - annoyed users.

Internals - your fellow developers.

First contact - new users.

Education - new & existing users.

Support - experienced users.

Troubleshooting - annoyed users.

Internals - your fellow developers.

Reference - everyone.

Documentation is
Communication.

Great documentation
has to serve **multiple,
conflicting masters.**

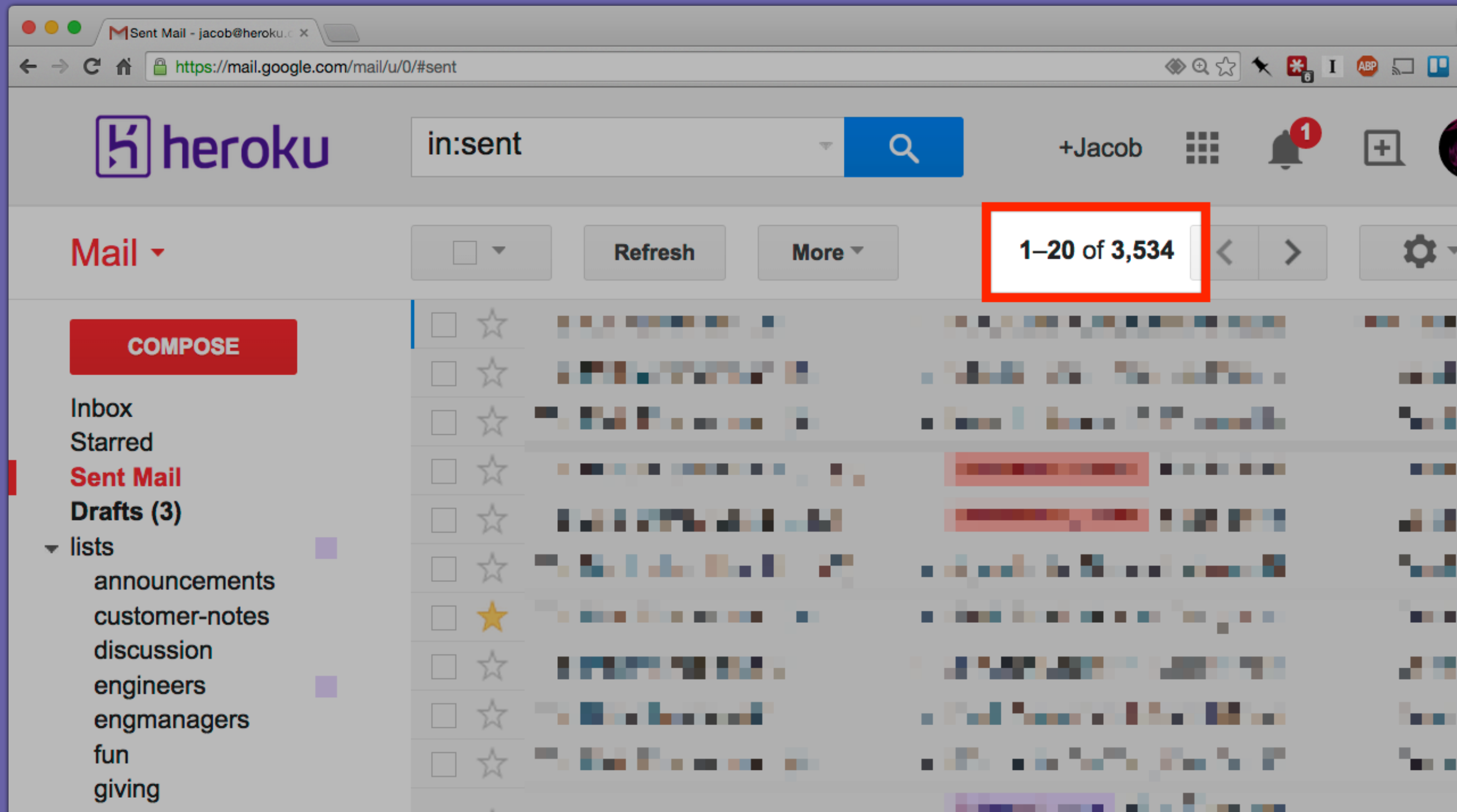
Why do people read documentation?

Who should write documentation?

What should we document?

“I’m bad at writing”

“I’m bad at writing”



Anyone can write!

Anyone can write!

But...

1. Set up a wiki

2.?

3. Profit!

A wiki tells me that you don't really care about your documentation.

So why should I care about your software?

Great documentation
is written by
great developers.

“The code required to fix a problem... is an essential part of a patch, **but it is not the only part**. A good patch should also include a **regression test** to validate the behavior that has been fixed.”

— <http://django.me/patch-style>

“If the... patch adds a new feature, or modifies behavior of an existing feature, **the patch should also contain documentation.**”

— <http://django.me/patch-style>



“If it’s not in Dev Center, **it doesn’t exist**”

— Mountjoy’s Law

Why do people read documentation?

Who should write documentation?

What should we document?

Introduction

Introduction

Explanation

Introduction

Explanation

Reference

Introduction

Explanation

Reference

Troubleshooting

Introduction

Quick - a new user should experience success within 30 minutes.

Easy - help users feel epic win.

Not too easy - don't sugar-coat the truth.

Show off how the project feels.

Explanation

Conceptual - foster understanding,
not parroting.

Comprehensive - explain in detail.

Tell me the **why** of the topic.

Reference

Complete. Docs or it doesn't exist.

Designed for **experienced users**.

Give me the **how** of the topic.

Troubleshooting

Answers to **questions asked in anger**.

FAQs are great as long as the **Qs are really FA'd**.



Great documentation
is **fractal**

Introduction

Explanation

Reference

Troubleshooting

Project

Introduction

Explanation

Reference

Troubleshooting

Project

Introduction

Tutorials, getting started

Explanation

Reference

Troubleshooting

Project

Introduction

Tutorials, getting started

Explanation

Topic guides, How-to guides

Reference

Troubleshooting

Project

Introduction

Tutorials, getting started

Explanation

Topic guides, How-to guides

Reference

Reference material, APIs, indexes, search

Troubleshooting

Project

Introduction

Tutorials, getting started

Explanation

Topic guides, How-to guides

Reference

Reference material, APIs, indexes, search

Troubleshooting

Troubleshooting guides, FAQs, KBs

Django documentation

Everything you need to know about Django.

First steps

Are you new to Django or to programming? This is the place to start!

- **From scratch:** [Overview](#) | [Installation](#)
- **Tutorial:** [Part 1](#) | [Part 2](#) | [Part 3](#) | [Part 4](#) | [Part 5](#) | [Part 6](#)
- **Advanced Tutorials:** [How to write reusable apps](#) | [Writing your first patch for Django](#)

The model layer

Django provides an abstraction layer (the "models") for structuring and manipulating the data of your Web application. Learn more about it below:

- **Models:** [Model syntax](#) | [Field types](#) | [Meta options](#)
- **QuerySets:** [Executing queries](#) | [QuerySet method reference](#) | [Query-related classes](#) | [Lookup expressions](#)
- **Model instances:** [Instance methods](#) | [Accessing related objects](#)
- **Migrations:** [Introduction to Migrations](#) | [Operations reference](#) | [SchemaEditor](#)
- **Advanced:** [Managers](#) | [Raw SQL](#) | [Transactions](#) | [Aggregation](#) | [Custom fields](#) | [Multiple databases](#) | [Custom lookups](#)
- **Other:** [Supported databases](#) | [Legacy databases](#) | [Providing initial data](#) | [Optimize database access](#)

The view layer

Django has the concept of "views" to encapsulate the logic responsible for processing a user's request and for returning the response. Find all you need to know about views via the links below:

- **The basics:** [URLconfs](#) | [View functions](#) | [Shortcuts](#) | [Decorators](#)
- **Reference:** [Built-in Views](#) | [Request/response objects](#) | [TemplateResponse objects](#)
- **File uploads:** [Overview](#) | [File objects](#) | [Storage API](#) | [Managing files](#) | [Custom storage](#)
- **Class-based views:** [Overview](#) | [Built-in display views](#) | [Built-in editing views](#) | [Using mixins](#) | [API reference](#) | [Flattened index](#)
- **Advanced:** [Generating CSV](#) | [Generating PDF](#)
- **Middleware:** [Overview](#) | [Built-in middleware classes](#)

The template layer

The template layer provides a designer-friendly syntax for rendering the information to be presented to the user. Learn how this syntax can be used by designers and how it can be extended by programmers:

- **For designers:** [Syntax overview](#) | [Built-in tags and filters](#) | [Web design helpers](#) | [Humanization](#)
- **For programmers:** [Template API](#) | [Custom tags and filters](#)

Forms

Django provides a rich framework to facilitate the creation of forms and the manipulation of form data.

Getting help

Having trouble? We'd like to help!

- Try the [FAQ](#) – it's got answers to many common questions.
- Looking for specific information? Try the [Index](#), [Module Index](#) or the [detailed table of contents](#).
- Search for information in the archives of the [django-users](#) mailing list, or [post a question](#).
- Ask a question in the [#django IRC channel](#), or search the [IRC logs](#) to see if it's been asked before.
- Report bugs with Django in our [ticket tracker](#).

Search

Version: [Django 1.7](#)

Browse

- Prev: [Django documentation contents](#)
- Next: [Getting started](#)
- [Table of contents](#)
- [General Index](#)
- [Python Module Index](#)

You are here:

- [Django 1.7 documentation](#)
- [Django documentation](#)

Download:

Offline (Django 1.7): [HTML](#) | [PDF](#) | [ePub](#)
Provided by [Read the Docs](#).

Django documentation | Django

←

→

↺

🏠

🔒

https://docs.djangoproject.com/en/1.7/

🔍

☆

ABP

☰

django

HomeDownloadDocumentationWeblogCommunityCode

Django documentation

Django documentation

Introduction

What Django.

First steps

Are you new to Django or to programming? This is the place to start!

▪ **From scratch:** Overview | Installation

▪ **Tutorial:** Part 1 | Part 2 | Part 3 | Part 4 | Part 5 | Part 6

▪ **Advanced Tutorials:** How to write reusable apps | Writing your first patch for Django

The model layer

Django provides an abstraction layer (the "models") for structuring and manipulating the data of your Web application. Learn more about it below:

▪ **Models:** Model syntax | Field types | Meta options

▪ **QuerySets:** Executing queries | QuerySet method reference | Query-related classes | Lookup expressions

▪ **Model instances:** Instance methods | Accessing related objects

▪ **Migrations:** Introduction to Migrations | Operations reference | SchemaEditor

▪ **Advanced:** Managers | Raw SQL | Transactions | Aggregation | Custom fields | Multiple databases | Custom lookups

▪ **Other:** Supported databases | Legacy databases | Providing initial data | Optimize database access

The view layer

Django has the concept of "views" to encapsulate the logic responsible for processing a user's request and for returning the response. Find all you need to know about views via the links below:

▪ **The basics:** URLconfs | View functions | Shortcuts | Decorators

▪ **Reference:** Built-in Views | Request/response objects | TemplateResponse objects

▪ **File uploads:** Overview | File objects | Storage API | Managing files | Custom storage

▪ **Class-based views:** Overview | Built-in display views | Built-in editing views | Using mixins | API reference | Flattened index

▪ **Advanced:** Generating CSV | Generating PDF

▪ **Middleware:** Overview | Built-in middleware classes

The template layer

The template layer provides a designer-friendly syntax for rendering the information to be presented to the user. Learn how this syntax can be used by designers and how it can be extended by programmers:

▪ **For designers:** Syntax overview | Built-in tags and filters | Web design helpers | Humanization

▪ **For programmers:** Template API | Custom tags and filters

Forms

Django provides a rich framework to facilitate the creation of forms and the manipulation of form data.

Getting help

Having trouble? We'd like to help!

▪ Try the **FAQ** – it's got answers to many common questions.

▪ Looking for specific information? Try the **Index**, **Module Index** or the **detailed table of contents**.

▪ Search for information in the archives of the **django-users** mailing list, or **post a question**.

▪ Ask a question in the **#django IRC channel**, or search the **IRC logs** to see if it's been asked before.

▪ Report bugs with Django in our **ticket tracker**.

Search

Version: Django 1.7

Search

Browse

▪ Prev: Django documentation contents

▪ Next: Getting started

▪ Table of contents

▪ General Index

▪ Python Module Index

You are here:

▪ Django 1.7 documentation

▪ Django documentation

Download:

Offline (Django 1.7): HTML | PDF | ePub

Provided by Read the Docs.

Documentation version: 1.7

Django documentation

Everything you need to know about Django.

First steps

Are you new to Django or to programming? This is the place to start!

- **From scratch:** [Overview](#) | [Installation](#)
- **Tutorial:** [Part 1](#) | [Part 2](#) | [Part 3](#) | [Part 4](#) | [Part 5](#) | [Part 6](#)
- [How to write reusable apps](#) | [Writing your first Django app](#)

Explanation

The model layer

Django provides an abstraction layer (the "models") for structuring and manipulating the data of your Web application. Learn more about it below:

- **Models:** [Model syntax](#) | [Field types](#) | [Meta options](#)
- **QuerySets:** [Executing queries](#) | [QuerySet method reference](#) | [Query-related classes](#) | [Lookup expressions](#)
- **Model instances:** [Instance methods](#) | [Accessing related objects](#)
- **Migrations:** [Introduction to Migrations](#) | [Operations reference](#) | [SchemaEditor](#)
- **Advanced:** [Managers](#) | [Raw SQL](#) | [Transactions](#) | [Aggregation](#) | [Custom fields](#) | [Multiple databases](#) | [Custom lookups](#)
- **Other:** [Supported databases](#) | [Legacy databases](#) | [Providing initial data](#) | [Optimize database access](#)

The view layer

Django has the concept of "views" to encapsulate the logic responsible for processing a user's request and for returning the response. Find all you need to know about views via the links below:

- **The basics:** [URLconfs](#) | [View functions](#) | [Shortcuts](#) | [Decorators](#)
- **Reference:** [Built-in Views](#) | [Request/response objects](#) | [TemplateResponse objects](#)
- **File uploads:** [Overview](#) | [File objects](#) | [Storage API](#) | [Managing files](#) | [Custom storage](#)
- **Class-based views:** [Overview](#) | [Built-in display views](#) | [Built-in editing views](#) | [Using mixins](#) | [API reference](#) | [Flattened index](#)
- **Advanced:** [Generating CSV](#) | [Generating PDF](#)
- **Middleware:** [Overview](#) | [Built-in middleware classes](#)

The template layer

The template layer provides a designer-friendly syntax for rendering the information to be presented to the user. Learn how this syntax can be used by designers and how it can be extended by programmers:

- **For designers:** [Syntax overview](#) | [Built-in tags and filters](#) | [Web design helpers](#) | [Humanization](#)
- **For programmers:** [Template API](#) | [Custom tags and filters](#)

Forms

Django provides a rich framework to facilitate the creation of forms and the manipulation of form data.

Getting help

Having trouble? We'd like to help!

- Try the [FAQ](#) – it's got answers to many common questions.
- Looking for specific information? Try the [Index](#), [Module Index](#) or the [detailed table of contents](#).
- Search for information in the archives of the [django-users](#) mailing list, or [post a question](#).
- Ask a question in the [#django IRC channel](#), or search the [IRC logs](#) to see if it's been asked before.
- Report bugs with Django in our [ticket tracker](#).

Search

Version: [Django 1.7](#)

Browse

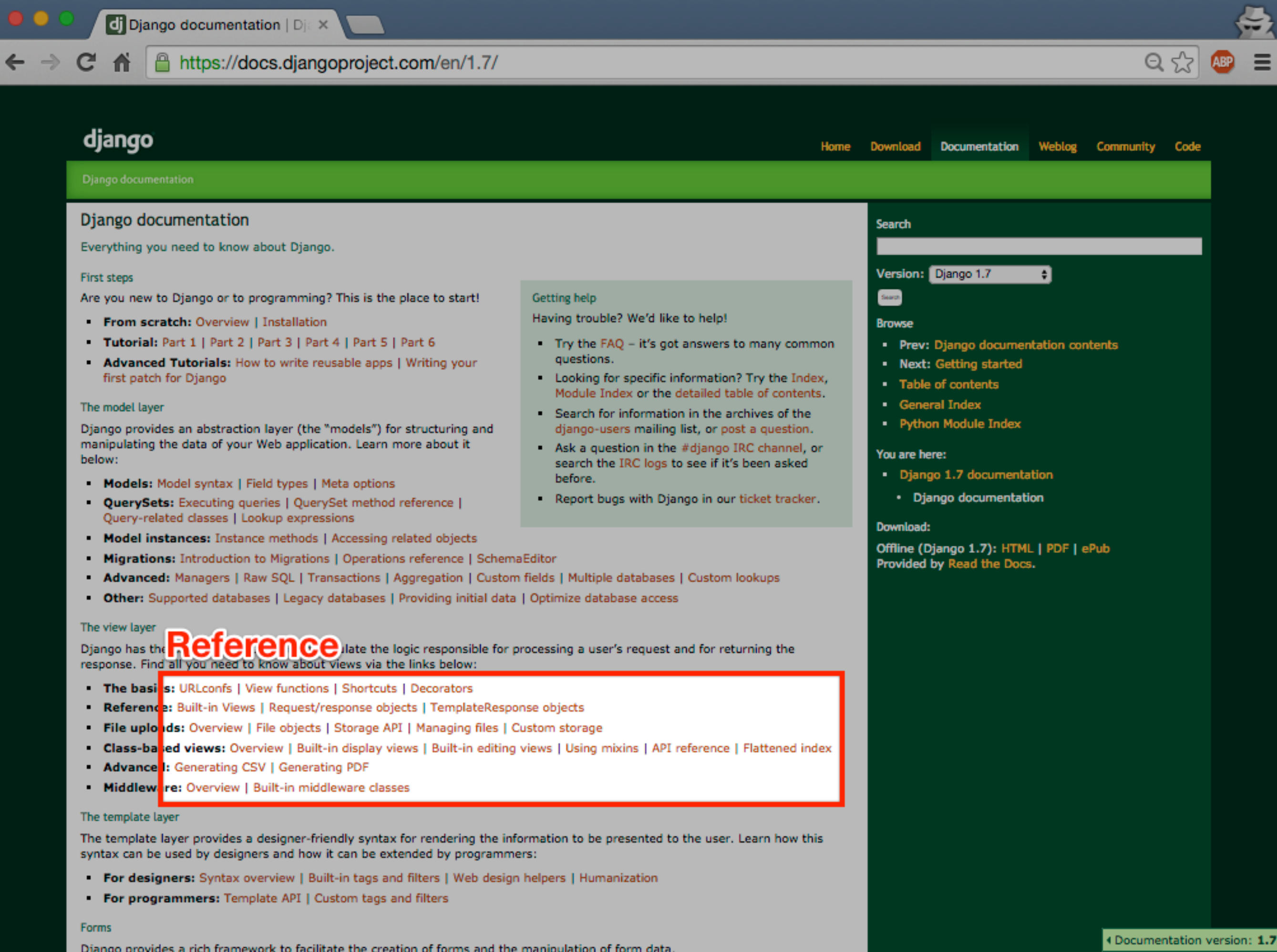
- **Prev:** [Django documentation contents](#)
- **Next:** [Getting started](#)
- [Table of contents](#)
- [General Index](#)
- [Python Module Index](#)

You are here:

- [Django 1.7 documentation](#)
- [Django documentation](#)

Download:

Offline (Django 1.7): [HTML](#) | [PDF](#) | [ePub](#)
Provided by [Read the Docs](#).



Reference

Django documentation | Django

←

→

↺

🏠

🔒

https://docs.djangoproject.com/en/1.7/

🔍

☆

ABP

☰

django

HomeDownloadDocumentationWeblogCommunityCode

Django documentation

Django documentation

Everything you need to know about Django.

First steps

Are you new to Django or to programming? This is the place to start!

▪ **From scratch:** [Overview](#) | [Installation](#)

▪ **Tutorial:** [Part 1](#) | [Part 2](#) | [Part 3](#) | [Part 4](#) | [Part 5](#) | [Part 6](#)

▪ **Advanced Tutorials:** [How to write reusable apps](#) | [Writing your first patch for Django](#)

The model layer

Django provides an abstraction layer (the "models") for structuring and manipulating the data of your Web application. Learn more about it below:

▪ **Models:** [Model syntax](#) | [Field types](#) | [Meta options](#)

▪ **QuerySets:** [Executing queries](#) | [QuerySet method reference](#) | [Query-related classes](#) | [Lookup expressions](#)

▪ **Model instances:** [Instance methods](#) | [Accessing related objects](#)

▪ **Migrations:** [Introduction to Migrations](#) | [Operations reference](#) | [SchemaEditor](#)

▪ **Advanced:** [Managers](#) | [Raw SQL](#) | [Transactions](#) | [Aggregation](#) | [Custom fields](#) | [Multiple databases](#) | [Custom lookups](#)

▪ **Other:** [Supported databases](#) | [Legacy databases](#) | [Providing initial data](#) | [Optimize database access](#)

The view layer

Django has the concept of "views" to encapsulate the logic responsible for processing a user's request and for returning the response. Find all you need to know about views via the links below:

▪ **The basics:** [URLconfs](#) | [View functions](#) | [Shortcuts](#) | [Decorators](#)

▪ **Reference:** [Built-in Views](#) | [Request/response objects](#) | [TemplateResponse objects](#)

▪ **File uploads:** [Overview](#) | [File objects](#) | [Storage API](#) | [Managing files](#) | [Custom storage](#)

▪ **Class-based views:** [Overview](#) | [Built-in display views](#) | [Built-in editing views](#) | [Using mixins](#) | [API reference](#) | [Flattened index](#)

▪ **Advanced:** [Generating CSV](#) | [Generating PDF](#)

▪ **Middleware:** [Overview](#) | [Built-in middleware classes](#)

The template layer

The template layer provides a designer-friendly syntax for rendering the information to be presented to the user. Learn how this syntax can be used by designers and how it can be extended by programmers:

▪ **For designers:** [Syntax overview](#) | [Built-in tags and filters](#) | [Web design helpers](#) | [Humanization](#)

▪ **For programmers:** [Template API](#) | [Custom tags and filters](#)

Forms

Django provides a rich framework to facilitate the creation of forms and the manipulation of form data.

Troubleshooting

Getting help

Having trouble? We'd like to help!

▪ Try the [FAQ](#) – it's got answers to many common questions.

▪ Looking for specific information? Try the [Index](#), [Module Index](#) or the [detailed table of contents](#).

▪ Search for information in the archives of the [django-users](#) mailing list, or [post a question](#).

▪ Ask a question in the [#django IRC channel](#), or search the [IRC logs](#) to see if it's been asked before.

▪ Report bugs with Django in our [ticket tracker](#).

Search

Version: Django 1.7

Search

Browse

▪ Prev: [Django documentation contents](#)

▪ Next: [Getting started](#)

▪ [Table of contents](#)

▪ [General Index](#)

▪ [Python Module Index](#)

You are here:

▪ [Django 1.7 documentation](#)

▪ [Django documentation](#)

Download:

Offline (Django 1.7): [HTML](#) | [PDF](#) | [ePub](#)

Provided by [Read the Docs](#).

Documentation version: 1.7

Document

Introduction

Explanation

Reference

Troubleshooting

Document

Introduction

Overview guide

Explanation

Reference

Troubleshooting

Document

Introduction

Overview guide

Explanation

Details, usage, explanation

Reference

Troubleshooting

Document

Introduction

Overview guide

Explanation

Details, usage, explanation

Reference

APIs, cross-references, next steps

Troubleshooting

Document

Introduction

Overview guide

Explanation

Details, usage, explanation

Reference

APIs, cross-references, next steps

Troubleshooting

Notes, warnings

Middleware

Middleware is a framework of hooks into Django's request/response processing. It's a light, low-level "plugin" system for globally altering Django's input or output.

Each middleware component is responsible for doing some specific function. For example, Django includes a middleware component, `TransactionMiddleware`, that wraps the processing of each HTTP request in a database transaction.

This document explains how middleware works, how you activate middleware, and how to write your own middleware. Django ships with some built-in middleware you can use right out of the box. They're documented in the [built-in middleware reference](#).

Activating middleware

To activate a middleware component, add it to the `MIDDLEWARE_CLASSES` tuple in your Django settings.

In `MIDDLEWARE_CLASSES`, each middleware component is represented by a string: the full Python path to the middleware's class name. For example, here's the default value created by `django-admin.py startproject`:

```
MIDDLEWARE_CLASSES = (  
    'django.contrib.sessions.middleware.SessionMiddleware',  
    'django.middleware.common.CommonMiddleware',  
    'django.middleware.csrf.CsrfViewMiddleware',  
    'django.contrib.auth.middleware.AuthenticationMiddleware',  
    'django.contrib.messages.middleware.MessageMiddleware',  
    'django.middleware.clickjacking.XFrameOptionsMiddleware',  
)
```

Search

Version: Django 1.7

Search

Contents

- **Middleware**
 - **Activating middleware**
 - **Hooks and application order**
 - **Writing your own middleware**
 - `process_request`
 - `process_view`
 - `process_template_response`
 - `process_response`
 - **Dealing with streaming responses**
 - `process_exception`
 - `__init__`
 - **Marking middleware as unused**
 - **Guidelines**

Browser Documentation version: 1.7

Middleware

Introduction

Middleware is a framework of hooks into Django's request/response processing. It's a light, low-level "plugin" system for globally altering Django's input or output.

Each middleware component is responsible for doing some specific function. For example, Django includes a middleware component, `TransactionMiddleware`, that wraps the processing of each HTTP request in a database transaction.

This document explains how middleware works, how you activate middleware, and how to write your own middleware. Django ships with some built-in middleware you can use right out of the box. They're documented in the [built-in middleware reference](#).

Activating middleware

To activate a middleware component, add it to the `MIDDLEWARE_CLASSES` tuple in your Django settings.

In `MIDDLEWARE_CLASSES`, each middleware component is represented by a string: the full Python path to the middleware's class name. For example, here's the default value created by `django-admin.py startproject`:

```
MIDDLEWARE_CLASSES = (  
    'django.contrib.sessions.middleware.SessionMiddleware',  
    'django.middleware.common.CommonMiddleware',  
    'django.middleware.csrf.CsrfViewMiddleware',  
    'django.contrib.auth.middleware.AuthenticationMiddleware',  
    'django.contrib.messages.middleware.MessageMiddleware',  
    'django.middleware.clickjacking.XFrameOptionsMiddleware',  
)
```

Search

Version: Django 1.7

Search

Contents

- **Middleware**
 - **Activating middleware**
 - **Hooks and application order**
 - **Writing your own middleware**
 - `process_request`
 - `process_view`
 - `process_template_response`
 - `process_response`
 - **Dealing with streaming responses**
 - `process_exception`
 - `__init__`
 - **Marking middleware as unused**
 - **Guidelines**

Browser Documentation version: 1.7

Middleware

Middleware is a framework of hooks into Django's request/response processing. It's a light, low-level "plugin" system for globally altering Django's input or output.

Each middleware component is responsible for doing some specific function. For example, Django includes a middleware component, `TransactionMiddleware`, that wraps the processing of each HTTP request in a database transaction.

This document explains how middleware works, how you activate middleware, and how to write your own middleware. Django ships with some built-in middleware you can use right out of the box. They're documented in the [built-in middleware reference](#).

Activating middleware

To activate a middleware component, add it to the `MIDDLEWARE_CLASSES` tuple in your Django settings.

In `MIDDLEWARE_CLASSES`, each middleware component is represented by a string: the full Python path to the middleware's class name. For example, here's the default value created by `django-admin.py startproject`:

```
MIDDLEWARE_CLASSES = (  
    'django.contrib.sessions.middleware.SessionMiddleware',  
    'django.middleware.common.CommonMiddleware',  
    'django.middleware.csrf.CsrfViewMiddleware',  
    'django.contrib.auth.middleware.AuthenticationMiddleware',  
    'django.contrib.messages.middleware.MessageMiddleware',  
    'django.middleware.clickjacking.XFrameOptionsMiddleware',  
)
```

Search

Version: Django 1.7

Search

Contents

Middleware

- [Activating middleware](#)
- [Hooks and application order](#)
- [Writing your own middleware](#)

- [process_request](#)
- [process_view](#)
- [process_template_response](#)
- [process_response](#)
- [Dealing with streaming responses](#)
- [process_exception](#)
- [__init__](#)
- [Marking middleware as unused](#)
- [Guidelines](#)

Explanation

Browser Documentation version: 1.7

Middleware

Middleware is a framework of hooks into Django's request/response processing. It's a light, low-level "plugin" system for globally altering Django's input or output.

Each middleware component is responsible for doing some specific function. For example, Django includes a middleware component, `TransactionMiddleware`, that wraps the processing of each HTTP request in a database transaction.

This document explains how middleware works, how you activate middleware, and how to write your own middleware. Django ships with some built-in middleware you can use right out of the box. They're documented in the [built-in middleware reference](#).

Activating middleware

To activate a middleware component, add it to the `MIDDLEWARE_CLASSES` tuple in your Django settings.

In `MIDDLEWARE_CLASSES`, each middleware component is represented by a string: the full Python path to the middleware's class name. For example, here's the default value created by `django-admin.py startproject`:

```
MIDDLEWARE_CLASSES = (  
    'django.contrib.sessions.middleware.SessionMiddleware',  
    'django.middleware.common.CommonMiddleware',  
    'django.middleware.csrf.CsrfViewMiddleware',  
    'django.contrib.auth.middleware.AuthenticationMiddleware',  
    'django.contrib.messages.middleware.MessageMiddleware',  
    'django.middleware.clickjacking.XFrameOptionsMiddleware',  
)
```

Search

Version: Django 1.7

Search

Contents

- **Middleware**
 - **Activating middleware**
 - **Hooks and application order**
 - **Reference**
 - `process_request`
 - `process_view`
 - `process_template_response`
 - `process_response`
 - **Dealing with streaming responses**
 - `process_exception`
 - `__init__`
 - **Marking middleware as unused**
 - **Guidelines**

Browser Documentation version: 1.7

Middleware

Middleware is a framework of hooks into Django's request/response processing. It's a light, low-level "plugin" system for globally altering Django's input or output.

Each middleware component is responsible for doing some specific function. For example, Django includes a middleware component, `TransactionMiddleware`, that wraps the processing of each HTTP request in a database transaction.

This document explains how middleware works, how you activate middleware, and how to write your own middleware. Django ships with some built-in middleware you can use right out of the box. They're documented in the [built-in middleware reference](#).

Activating middleware

To activate a middleware component, add it to the `MIDDLEWARE_CLASSES` tuple in your Django settings.

In `MIDDLEWARE_CLASSES`, each middleware component is represented by a string: the full Python path to the middleware's class name. For example, here's the default value created by `django-admin.py startproject`:

```
MIDDLEWARE_CLASSES = (  
    'django.contrib.sessions.middleware.SessionMiddleware',  
    'django.middleware.common.CommonMiddleware',  
    'django.middleware.csrf.CsrfViewMiddleware',  
    'django.contrib.auth.middleware.AuthenticationMiddleware',  
    'django.contrib.messages.middleware.MessageMiddleware',  
    'django.middleware.clickjacking.XFrameOptionsMiddleware',  
)
```

Search

Version: Django 1.7

Search

Contents

- **Middleware**
 - **Activating middleware**
 - **Hooks and application order**
 - **Writing your own middleware**
 - `process_request`
 - `process_view`
 - `process_template_response`
 - `process_response`
 - **Dealing with streaming responses**
 - `process_exception`
 - `__init__`
 - **Marking middleware as**
 - **Guidelines**

Troubleshooting

Browser Documentation version: 1.7

Section

Introduction

Explanation

Reference

Troubleshooting

Section

Introduction

Overview

Explanation

Reference

Troubleshooting

Section

Introduction

Overview

Explanation

Tasks and examples

Reference

Troubleshooting

Section

Introduction

Overview

Explanation

Tasks and examples

Reference

Detailed APIs, cross-references, next steps

Troubleshooting

Section

Introduction

Overview

Explanation

Tasks and examples

Reference

Detailed APIs, cross-references, next steps

Troubleshooting

Common pitfalls, warnings

Activating middleware

To activate a middleware component, add it to the `MIDDLEWARE_CLASSES` tuple in your Django settings.

In `MIDDLEWARE_CLASSES`, each middleware component is represented by a string: the full Python path to the middleware's class name. For example, here's the default value created by `django-admin.py startproject`:

```
MIDDLEWARE_CLASSES = (
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
)
```

A Django installation doesn't require any middleware — `MIDDLEWARE_CLASSES` can be empty, if you'd like — but it's strongly suggested that you at least use `CommonMiddleware`.

The order in `MIDDLEWARE_CLASSES` matters because a middleware can depend on other middleware. For instance, `AuthenticationMiddleware` stores the authenticated user in the session; therefore, it must run after `SessionMiddleware`. See [Middleware ordering](#) for some common hints about ordering of Django middleware classes.

Hooks and application order

During the request phase, before calling the view, Django applies middleware in the order it's defined in `MIDDLEWARE_CLASSES`, top-down. Two hooks are available:

- `process_request`
- `process_view`
- `process_template_response`
- `process_response`
 - [Dealing with streaming responses](#)
- `process_exception`
- `__init__`
 - [Marking middleware as unused](#)
- [Guidelines](#)

Browse

- Prev: [Generic views](#)
- Next: [How to use sessions](#)
- [Table of contents](#)
- [General Index](#)
- [Python Module Index](#)

You are here:

- [Django 1.7 documentation](#)
 - [Using Django](#)
 - [Handling HTTP requests](#)

Activating middleware

Introduction

To activate a middleware component, add it to the `MIDDLEWARE_CLASSES` tuple in your Django settings.

In `MIDDLEWARE_CLASSES`, each middleware component is represented by a string: the full Python path to the middleware's class name. For example, here's the default value created by `django-admin.py startproject`:

```
MIDDLEWARE_CLASSES = (  
    'django.contrib.sessions.middleware.SessionMiddleware',  
    'django.middleware.common.CommonMiddleware',  
    'django.middleware.csrf.CsrfViewMiddleware',  
    'django.contrib.auth.middleware.AuthenticationMiddleware',  
    'django.contrib.messages.middleware.MessageMiddleware',  
    'django.middleware.clickjacking.XFrameOptionsMiddleware',  
)
```

A Django installation doesn't require any middleware — `MIDDLEWARE_CLASSES` can be empty, if you'd like — but it's strongly suggested that you at least use `CommonMiddleware`.

The order in `MIDDLEWARE_CLASSES` matters because a middleware can depend on other middleware. For instance, `AuthenticationMiddleware` stores the authenticated user in the session; therefore, it must run after `SessionMiddleware`. See [Middleware ordering](#) for some common hints about ordering of Django middleware classes.

Hooks and application order

During the request phase, before calling the view, Django applies middleware in the order it's defined in `MIDDLEWARE_CLASSES`, top-down. Two hooks are available:

- `process_request`
- `process_view`
- `process_template_response`
- `process_response`
 - [Dealing with streaming responses](#)
- `process_exception`
- `__init__`
 - [Marking middleware as unused](#)
- [Guidelines](#)

Browse

- [Prev: Generic views](#)
- [Next: How to use sessions](#)
- [Table of contents](#)
- [General Index](#)
- [Python Module Index](#)

You are here:

- [Django 1.7 documentation](#)
 - [Using Django](#)
 - [Handling HTTP requests](#)

Activating middleware

To activate a middleware component, add it to the `MIDDLEWARE_CLASSES` tuple in your Django settings.

Explanation

In `MIDDLEWARE_CLASSES`, each middleware component is represented by a string: the full Python path to the middleware's class name. For example, here's the default value created by `django-admin.py startproject`:

```
MIDDLEWARE_CLASSES = (
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
)
```

A Django installation doesn't require any middleware — `MIDDLEWARE_CLASSES` can be empty, if you'd like — but it's strongly suggested that you at least use `CommonMiddleware`.

The order in `MIDDLEWARE_CLASSES` matters because a middleware can depend on other middleware. For instance, `AuthenticationMiddleware` stores the authenticated user in the session; therefore, it must run after `SessionMiddleware`. See [Middleware ordering](#) for some common hints about ordering of Django middleware classes.

Hooks and application order

During the request phase, before calling the view, Django applies middleware in the order it's defined in `MIDDLEWARE_CLASSES`, top-down. Two hooks are available:

- `process_request`
- `process_view`
- `process_template_response`
- `process_response`
 - [Dealing with streaming responses](#)
- `process_exception`
- `__init__`
 - [Marking middleware as unused](#)
- [Guidelines](#)

Browse

- [Prev: Generic views](#)
- [Next: How to use sessions](#)
- [Table of contents](#)
- [General Index](#)
- [Python Module Index](#)

You are here:

- [Django 1.7 documentation](#)
 - [Using Django](#)
 - [Handling HTTP requests](#)

Activating middleware

To activate a middleware component, add it to the `MIDDLEWARE_CLASSES` tuple in your Django settings.

In `MIDDLEWARE_CLASSES`, each middleware component is represented by a string: the full Python path to the middleware's class name. For example, here's the default value created by `django-admin.py startproject`:

```
MIDDLEWARE_CLASSES = (  
    'django.contrib.sessions.middleware.SessionMiddleware',  
    'django.middleware.common.CommonMiddleware',  
    'django.middleware.csrf.CsrfViewMiddleware',  
    'django.contrib.auth.middleware.AuthenticationMiddleware',  
    'django.contrib.messages.middleware.MessageMiddleware',  
    'django.middleware.clickjacking.XFrameOptionsMiddleware',  
)
```

Reference

A Django installation doesn't require any middleware — `MIDDLEWARE_CLASSES` can be empty, if you'd like — but it's strongly suggested that you at least use `CommonMiddleware`.

The order in `MIDDLEWARE_CLASSES` matters because a middleware can depend on other middleware. For instance, `AuthenticationMiddleware` stores the authenticated user in the session; therefore, it must run after `SessionMiddleware`. See [Middleware ordering](#) for some common hints about ordering of Django middleware classes.

Hooks and application order

During the request phase, before calling the view, Django applies middleware in the order it's defined in `MIDDLEWARE_CLASSES`, top-down. Two hooks are available:

- `process_request`
- `process_view`
- `process_template_response`
- `process_response`
 - [Dealing with streaming responses](#)
- `process_exception`
- `__init__`
 - [Marking middleware as unused](#)
- [Guidelines](#)

Browse

- [Prev: Generic views](#)
- [Next: How to use sessions](#)
- [Table of contents](#)
- [General Index](#)
- [Python Module Index](#)

You are here:

- [Django 1.7 documentation](#)
 - [Using Django](#)
 - [Handling HTTP requests](#)

Activating middleware

To activate a middleware component, add it to the `MIDDLEWARE_CLASSES` tuple in your Django settings.

In `MIDDLEWARE_CLASSES`, each middleware component is represented by a string: the full Python path to the middleware's class name. For example, here's the default value created by `django-admin.py startproject`:

```
MIDDLEWARE_CLASSES = (
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
)
```

A Django installation doesn't require any middleware — `MIDDLEWARE_CLASSES` can be empty, if you'd like — but it's strongly suggested that you at least use

Troubleshooting

The order in `MIDDLEWARE_CLASSES` matters because a middleware can depend on other middleware. For instance, `AuthenticationMiddleware` stores the authenticated user in the session; therefore, it must run after `SessionMiddleware`. See [Middleware ordering](#) for some common hints about ordering of Django middleware classes.

Hooks and application order

During the request phase, before calling the view, Django applies middleware in the order it's defined in `MIDDLEWARE_CLASSES`, top-down. Two hooks are available:

- `process_request`
- `process_view`
- `process_template_response`
- `process_response`
 - [Dealing with streaming responses](#)
- `process_exception`
- `__init__`
 - [Marking middleware as unused](#)
- [Guidelines](#)

Browse

- [Prev: Generic views](#)
- [Next: How to use sessions](#)
- [Table of contents](#)
- [General Index](#)
- [Python Module Index](#)

You are here:

- [Django 1.7 documentation](#)
 - [Using Django](#)
 - [Handling HTTP requests](#)

Element

Introduction

Explanation

Reference

Troubleshooting

Element

Introduction

Basic usage, example, API detail

Explanation

Reference

Troubleshooting

Element

Introduction

Basic usage, example, API detail

Explanation

Detailed instructions

Reference

Troubleshooting

Element

Introduction

Basic usage, example, API detail

Explanation

Detailed instructions

Reference

Arguments/return types, defaults, cross-references

Troubleshooting

Element

Introduction

Basic usage, example, API detail

Explanation

Detailed instructions

Reference

Arguments/return types, defaults, cross-references

Troubleshooting

“If it didn’t work...”

`process_view`

`process_view(request, view_func, view_args, view_kwargs)`

`request` is an `HttpRequest` object. `view_func` is the Python function that Django is about to use. (It's the actual function object, not the name of the function as a string.) `view_args` is a list of positional arguments that will be passed to the view, and `view_kwargs` is a dictionary of keyword arguments that will be passed to the view. Neither `view_args` nor `view_kwargs` include the first view argument (`request`).

`process_view()` is called just before Django calls the view.

It should return either `None` or an `HttpResponse` object. If it returns `None`, Django will continue processing this request, executing any other `process_view()` middleware and, then, the appropriate view. If it returns an `HttpResponse` object, Django won't bother calling any other view or exception middleware, or the appropriate view; it'll apply response middleware to that `HttpResponse`, and return the result.



Note

Accessing `request.POST` or `request.REQUEST` inside middleware from `process_request` or `process_view` will prevent any view running after the middleware from being able to **modify the upload handlers for the request**, and should normally be avoided.

The `CsrfViewMiddleware` class can be considered an exception, as it provides the `csrf_exempt()` and `csrf_protect()` decorators which allow views to explicitly control at what point the CSRF validation should occur.

`process_template_response`

`process_template_response(request, response)`

process_view

Introduction

`process_view(request, view_func, view_args, view_kwargs)`

`request` is an `HttpRequest` object. `view_func` is the Python function that Django is about to use. (It's the actual function object, not the name of the function as a string.) `view_args` is a list of positional arguments that will be passed to the view, and `view_kwargs` is a dictionary of keyword arguments that will be passed to the view. Neither `view_args` nor `view_kwargs` include the first view argument (`request`).

`process_view()` is called just before Django calls the view.

It should return either `None` or an `HttpResponse` object. If it returns `None`, Django will continue processing this request, executing any other `process_view()` middleware and, then, the appropriate view. If it returns an `HttpResponse` object, Django won't bother calling any other view or exception middleware, or the appropriate view; it'll apply response middleware to that `HttpResponse`, and return the result.



Note

Accessing `request.POST` or `request.REQUEST` inside middleware from `process_request` or `process_view` will prevent any view running after the middleware from being able to **modify the upload handlers for the request**, and should normally be avoided.

The `CsrfViewMiddleware` class can be considered an exception, as it provides the `csrf_exempt()` and `csrf_protect()` decorators which allow views to explicitly control at what point the CSRF validation should occur.

process_template_response

`process_template_response(request, response)`

process_view

process_view(request, view_func, view_args, view_kw) **Explanation**

`request` is an `HttpRequest` object. `view_func` is the Python function that Django is about to use. (It's the actual function object, not the name of the function as a string.) `view_args` is a list of positional arguments that will be passed to the view, and `view_kwargs` is a dictionary of keyword arguments that will be passed to the view. Neither `view_args` nor `view_kwargs` include the first view argument (`request`).

`process_view()` is called just before Django calls the view.

It should return either `None` or an `HttpResponse` object. If it returns `None`, Django will continue processing this request, executing any other `process_view()` middleware and, then, the appropriate view. If it returns an `HttpResponse` object, Django won't bother calling any other view or exception middleware, or the appropriate view; it'll apply response middleware to that `HttpResponse`, and return the result.



Note

Accessing `request.POST` or `request.REQUEST` inside middleware from `process_request` or `process_view` will prevent any view running after the middleware from being able to **modify the upload handlers for the request**, and should normally be avoided.

The `CsrfViewMiddleware` class can be considered an exception, as it provides the `csrf_exempt()` and `csrf_protect()` decorators which allow views to explicitly control at what point the CSRF validation should occur.

process_template_response

process_template_response(request, response)

`process_view`

`process_view(request, view_func, view_args, view_kwargs)`

`request` is an `HttpRequest` object. `view_func` is the Python function that Django is about to use. (It's the actual function object, not the name of the function as a string.) `view_args` is a list of positional arguments that will be passed to the view, and `view_kwargs` is a dictionary of keyword arguments that will be passed to the view. Neither `view_args` nor `view_kwargs` include the first view argument (`request`).

`process_view()` is called just before Django calls the **Reference**

It should return either `None` or an `HttpResponse` object. If it returns `None`, Django will continue processing this request, executing any other `process_view()` middleware and, then, the appropriate view. If it returns an `HttpResponse` object, Django won't bother calling any other view or exception middleware, or the appropriate view; it'll apply response middleware to that `HttpResponse`, and return the result.



Note

Accessing `request.POST` or `request.REQUEST` inside middleware from `process_request` or `process_view` will prevent any view running after the middleware from being able to **modify the upload handlers for the request**, and should normally be avoided.

The `CsrfViewMiddleware` class can be considered an exception, as it provides the `csrf_exempt()` and `csrf_protect()` decorators which allow views to explicitly control at what point the CSRF validation should occur.

`process_template_response`

`process_template_response(request, response)`

process_view

process_view(request, view_func, view_args, view_kwargs)

`request` is an `HttpRequest` object. `view_func` is the Python function that Django is about to use. (It's the actual function object, not the name of the function as a string.) `view_args` is a list of positional arguments that will be passed to the view, and `view_kwargs` is a dictionary of keyword arguments that will be passed to the view. Neither `view_args` nor `view_kwargs` include the first view argument (`request`).

`process_view()` is called just before Django calls the view.

It should return either `None` or an `HttpResponse` object. If it returns `None`, Django will continue processing this request, executing any other `process_view()` middleware and, then, the appropriate view. If it returns an `HttpResponse` object, Django won't bother calling any other view or exception middleware, or the appropriate view; it'll return that `HttpResponse`, and return the result.

Troubleshooting



Note

Accessing `request.POST` or `request.REQUEST` inside middleware from `process_request` or `process_view` will prevent any view running after the middleware from being able to **modify the upload handlers for the request**, and should normally be avoided.

The `CsrfViewMiddleware` class can be considered an exception, as it provides the `csrf_exempt()` and `csrf_protect()` decorators which allow views to explicitly control at what point the CSRF validation should occur.

process_template_response

process_template_response(request, response)

Documentation is fractal

	Introduction	Explanation	Reference	Trouble-shooting
Project	Tutorials, Getting started	Guides, How-tos	APIs, indexes, search	FAQs, KB
Document	Introductory material	How-to guides	See also, next steps	Notes, warnings
Section	Overview	Tasks, examples	Cross-ref - other topics	Common pitfalls
Element	Examples	Detailed instructions	Cross-ref - API docs	“If it didn’t work...”

Why do people read documentation?

Who should write documentation?

What should we document?

Everyone reads documentation.

Who should write documentation?

What should we document?

Everyone reads documentation.

Developers write the best docs.

What should we document?

Everyone reads documentation.

Developers write the best docs.

Great documentation is **fractal**.



Write Great Documentation!

Jacob Kaplan-Moss
jacob@heroku.com