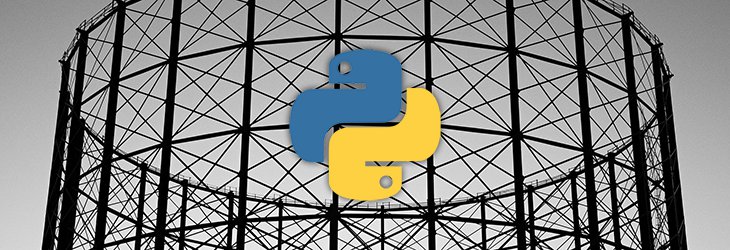
**A Beginner's Introduction to Python Frameworks**

[**Python**](https://stxnext.com/blog/category/python/)



So, you’ve started to learn [Python](https://stxnext.com/services/python-js-development).

It doesn’t seem too bad; you can code after all, so it’s just a matter of learning the differences in syntax. Perhaps you’re just at the stage of practicing tutorials and reading books.

But it is high time to start collecting hard experience in Python. It’s time to create your first Python project.

What should you start with? With an idea, obviously, but I’m sure that won’t be a problem. You already have several great concepts waiting for some of your spare time.

What’s next? The choice of a framework. And that’s where the real conundrum starts, because the ecosystem of Python frameworks is quite extensive and varied. Almost like a jungle, if you will.

In this article I’m going describe the most common Python frameworks so you can choose the one you want to start with. Be warned that this is a purely subjective comparison, because it mainly results from my own more-or-less successful attempts to use those frameworks in commercial projects.

**Django**

The **most popular Python framework is definitely Django**. Its characteristic feature is that within a single package there is everything you need to build a web application, from low- to high-end.

Django applications are based on a design pattern similar to MVC, the so-called **MVT (Model-View-Template) pattern**. Models are defined using Django ORM. SQL databases are mainly used as storage. Django has a built-in admin panel, allowing for easy management of the database content.

With minimal configuration this panel is automatically generated based on the defined models. Views can include both functions and classes. The assignment of URLs to views is done in one place (the urls.py file), so that after reviewing that single file you can learn what URLs are supported. Templates are created using a fairly simple **Django Templates system**.

Django is praised for strong community support, and for detailed documentation describing the functionality of the framework. This documentation coupled with the fact that after the installation you get a complete environment, the entry threshold is rather low. After going through [the official tutorial](https://docs.djangoproject.com/en/1.11/intro/tutorial01/), you’ll be able to do most of the things needed to build an application.

Unfortunately, Django’s monolithism also has its drawbacks. It’s difficult, though not impossible, to replace one of the built-in elements with another implementation. For example, using some other ORM (e.g. SQLAlchemy) requires abandoning or completely rebuilding such items as the admin panel, authorization, session handling or generating forms.

Because Django is complete but inflexible, it is suitable for standard applications (i.e. the vast majority of software projects). However, if you need to implement some unconventional design, it leads to a struggle with the framework rather than pleasant programming.

**Sample model in Django:**

**Django**

**class** **Company**(models.Model):

name = models.CharField(max\_length=255)

email = models.EmailField(max\_length=75, null=**True**, blank=**True**)

website\_url = models.URLField(blank=**True**, null=**True**)

city = models.CharField(max\_length=100, null=**True**, blank=**True**)

street = models.CharField(max\_length=100, null=**True**, blank=**True**)

size = models.IntegerField(null=**True**, blank=**True**)

date\_founded = models.CharField(

help\_text='MM/YYYY', null=**True**, blank=**True**, max\_length=7,

)

@property

**def** **urls**(self):

**return** {

'view': reverse('view-company', args=(self.pk,)),

'edit': reverse('edit-company', args=(self.pk,)),

}

**def** **\_\_unicode\_\_**(self):

**return** self.name

**Flask**

Flask is considered a **micro-framework**. It comes with basic functionality, but also allows you to easily expand it. Therefore, Flask works more as the glue that allows you to join libraries with each other. For example, “pure Flask” does not provide support for any storage, but there are a number of different implementations that you can install and use interchangeably for that purpose (e.g. Flask-SQLAlchemy, Flask-MongoAlchemy, and Flask-Redis). Similarly, the basic template system is Jinja2, but you can use a replacement (e.g. Mako).

The motto of this framework is “one drop at a time”, and this is reflected in its comprehensive documentation. Knowledge of how to build an application is acquired in portions here – after reading a few paragraphs, you will be able to perform basic tasks. You do not have to know the more advanced stuff – you’ll get to learn it only when you need it. Thanks to this, the Flask student can gather knowledge smoothly and avoid boredom, making Flask suitable for learning.

A large number of **Flask extensions**, unfortunately, are not as well supported as the framework itself. It happens quite often that the plug-ins are no longer being developed or their documentation is not outdated. In such situations, you need to spend some time googling a replacement that offers similar functionality, but is still actively supported. Building your application with packages from different authors, you might have to put quite a bit of sweat into integrating them with each other. You will rarely find ready-made instructions how to do this in the plug-ins’ documentation, but in such situations the flask community and websites such as Stack Overflow may be helpful.

**Sample view in Flask:**

@image\_view.route(

'/api/<string:version>/products/<int:prod\_id>/images',

methods=['GET'],

)

@auth\_required()

@documented(prod\_id="ID of a product")

@output(ProductImagesSeq)

@errors(MissingProduct)

@jsonify

**def** **images\_get**(version, prod\_id):

"""Retrieves a list of product images."""

**return** [i.serialize() **for** i **in** find\_product(prod\_id).images]

**Others**

The world of Python frameworks includes many more interesting examples. Each of these frameworks focuses on a different issue, was built for distinct tasks, or has a particular history. The first one that comes to mind is **Pyramid**, for which a crucial issue is one-hundred-percent test coverage and simplicity of design without performance loss. Pyramid drew its inspiration and experience from Pylons and repoze.bfg – two pretty decent frameworks that aren’t being developed anymore.

**Zope2**is one of the oldest frameworks, but it is still used mainly as part of a CMS called Plone. Zope3 (later renamed BlueBream) was created as Zope2’s successor. The framework was supposed to allow for easier creation of large applications, but has not won too much popularity, mainly because of the need to master fairly complex concepts (e.g. Zope Component Architecture) very early in the learning process.

Also noteworthy is**Google App Engine, which allows you to run applications written in Python**, among others. This platform lets you create applications in any framework compatible with WSGI. The SDK for App Engine includes a simple framework called webapp2 and exactly this approach is often used in web applications adapted to this environment.

Another interesting example is **Tornado**developed by FriendFeed and made available by Facebook. This framework includes libraries supporting asynchronicity, so you can build applications supporting multiple simultaneous connections (e.g. long pooling, WebSocket). Other similar libraries include **Pulsar**(async), **Twisted**(callbacks), and **Gevent**(greenlet). These libraries allow you to build any network applications (e.g. multiplayer games, chat rooms), but but they also perform well at handling HTTP requests.

Developing applications using these frameworks and libraries is more difficult and requires you to explore some more difficult concepts. I’d recommend using them later on in your venture into the Python world.

**Conclusion**

I hope this short summary of Python frameworks will help you decide which framework you should delve into first.

Now then, it’s time for your move. Read a tutorial, write some practice code and then get started with your first project in Python.