Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries.It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions.

it's a microframework that doesn't include an ORM (Object Relational Manager).

{Object-relational mapping (ORM) is a mechanism that makes it possible to address, access and manipulate objects without having to consider how those objects relate to their data sources. ORM lets programmers maintain a consistent view of objects over time, even as the sources that deliver them, the sinks that receive them and the applications that access them change.}

An object-relational mapper (ORM) is a code library that automates the transfer of data stored in relational database tables into objects that are more commonly used in application code.

It does have many cool features like url routing, template engine. It is a WSGI(The Web Server Gateway Interface) web app framework.

Why is Flask a good web framework choice?
Flask is considered more Pythonic than the Django web framework because in common situations the equivalent Flask web application is more explicit. Flask is also easy to get started with as a beginner because there is little boilerplate code for getting a simple app up and running. On top of that it's very explicit, which increases readability. To create the "Hello World" app, you only need a few lines of code.

What is a Web Framework?
A Web Application Framework or a simply a Web Framework represents a collection of libraries and modules that enable web application developers to write applications without worrying about low-level details such as protocol, thread management, and so on.

Flask is based on the Werkzeg WSGI toolkit and the Jinja2 template engine.Both are Pocco projects.

WSGI
The Web Server Gateway Interface (Web Server Gateway Interface, WSGI) has been used as a standard for Python web application development. WSGI is the specification of a common interface between web servers and web applications.

Werkzeug

Werkzeug is a WSGI toolkit that implements requests, response objects, and utility functions. This enables a web frame to be built on it. The Flask framework uses Werkzeg as one of its bases.

jinja2

jinja2 is a popular template engine for Python.A web template system combines a template with a specific data source to render a dynamic web page.

Parameter	Django	Flask
Type of framework	Django is a full-stack web framework that enables ready to use solutions with its batteries-included approach.	Flask is a lightweight framework that gives abundant features without external libraries and minimalist features.
Working of Framework/Data Model	Django follows an object-oriented approach that enables object-relational mapping (linking databases and tables with classes)	Flask works on a modular approach that enables working through outsourced libraries and extensions.
Project Layout	Django is suitable for multiple page applications.	Flask is suitable for only single-page applications.
Bootstrapping Tool	-Django-admin is the in-built bootstrapping tool of Django that allows the creation of web applications without any external input.	Flask does not come with an in-built bootstrapping tool.
Database Support	Django supports the most popular relational database management systems like MySQL, Oracle etc.	Flask does not support the basic database management system and uses SQLAlchemy for database requirements.
Flexibility	Django is less flexible because of its in-built features and tools. Developers cannot make changes to the modules.	Flask is a micro-based framework with extensible libraries making itself a flexible framework for developers.

Template Engine	Django is inspired by the Ninja2 template but has its built-in model view template that makes the development process easier.	Flask used Ninja2 template design
Control	Developers do not have full control over the modules and functions of Django because of built-in libraries.	Flask allows developers full control over the creation of applications with no dependencies from external libraries.
Working Style	The working style of Django is Monolithic	The working style of Flask is diversified style.
Debugger	Django does not support any virtual debugging.	Flash has an in-built debugger that offers virtual debugging
Routing and Views	Django framework supports the mapping of URL to views through a request.	Flask web framework allows mapping of URL to class-based view with Werkzeug.
Structure	Django framework structure is more conventional.	Flask web framework structure is random.
HTML	Django supports dynamic HTML pages	Flask framework does not support dynamic HTML pages
Best Features	• Open-Source Great Community • Fast Development • Easy to learn • Secure	• Extensive Documentation • Lightweight • Minimal Features • Full Control over the development process • Open-Source
Usage	Django is suitable for high-end technology companies like Instagram, Udemy, Coursera etc.	Flask is suitable for companies and projects that want experimentation with the module, architecture of the framework like Netflix, Reddit, Airbnb, etc.

Flask: Pros and Cons

Pros/Advantages

- Adaptable to the latest technologies
- Independent framework enables experimentation with architecture, libraries.
- Suitable for small case projects
- Requires small codebase size for simple functions
- Ensures scalability for simplistic applications
- Easy to build a quick prototype
- Routing URL functions through Werkzeug makes the process easier.
- Hassle-free application development and maintenance.
- Database integration is easy
- Extensible and easy core system.
- The power of the framework lies in its minimalistic features.
- Flexible and allow full control access.

Cons/Drawbacks

- MVP(Minimum Viable Product) development process is slow.
- Not suitable for big applications or projects.
- Complex maintenance for intricate implementations or system updates.

- There is no in-built admin site for maintaining models, insert, update or delete records.
- Does not support a proper database system and lacks Object- Relation Mapping.
- Absence of a strong community for support and growth.
- Security is uncertain, with no function for user authentication or login.

How To Create Your First Web Application Using Flask and Python 3

You'll install Flask, write and run a Flask application, and run the application in development mode. You'll use routing to display various web pages that serve different purposes in your web application. You'll also use view functions to allow users to interact with the application through dynamic routes. Finally, you'll use the debugger to troubleshoot errors.

Prerequisites:
A local Python 3 programming environment.

An understanding of basic Python 3 concepts, such as data types, lists, functions.

An understanding of basic HTML concepts.

Step 1 — Installing Flask
In this step, you'll activate your Python environment and install Flask using the pip package install command.

You've created the project folder, a virtual environment, and installed Flask. You can now move on to setting up a simple application.

Step 2 — Creating a Simple Application
Now that you have your programming environment set up, you'll start using Flask. In this step, you'll make a small Flask web application inside a Python file, in which you'll write HTML code to display on the browser.

In your flask_app directory, open a file named app.py for editing, use nano or your favorite text editor: nano app.py

```
from flask import Flask

app = Flask(__name__)

@app.route('/')

def hello():

    return '<h1>Hello, World!</h1>'
```

Step 3 — Running the Application
After creating the file that contains the Flask application, you'll run it using the Flask command line interface to start the development server and render on the browser the HTML code you wrote as a return value for the hello() view function in the previous step.

- The application is running locally on the URL http://127.0.0.1:5000/. 127.0.0.1 is the IP that represents your machine's localhost and :5000 is the port number.

You now have a small Flask web application. You've run your application and displayed information on the web browser. Next, you'll learn about routes and how to use them to serve multiple web pages. Step 4 — Routes and View Functions
In this step, you'll add a few routes to your application to display different pages depending on the requested URL. You'll also learn about view functions and how to use them.

A *route* is a URL you can use to determine what the user receives when they visit your web application on their browser. For example, http://127.0.0.1:5000/ is the main route that might be used to display an index page. The URL http://127.0.0.1:5000/about may be another route used for an about page that gives the visitor some information about your web application. Similarly, you can create a route that allows users to sign in to your application at http://127.0.0.1:5000/login.

Next, you'll use dynamic routes to allow users to control the application's response.

Step 5 — Dynamic Routes

In this step, you'll use dynamic routes to allow users to interact with the application. You'll make a route that capitalizes words passed through the URL, and a route that adds two numbers together and displays the result.

Next, we"ll troubleshoot and debug your Flask application in case of an error.

Step 6 — Debugging A Flask Application

When developing a web application, you will frequently run into situations where the application displays an error instead of the behavior you expect. You may misspell a variable or forget to define or import a function. To make fixing these problems easier, Flask provides a debugger when running the application in development mode. In this step, you will learn how to fix errors in your application using the Flask debugger.