1. In Python, a **subdirectory that includes a \_\_init\_\_.py** file is considered a **package**, and can be imported. When you import a package, the \_\_init\_\_.py executes and defines what symbols the package exposes to the outside world.
2. In your flask application, have a python script file at the top of the folder hierarchy in the application folder. This script should define/create the flask application instance. This will be the entry point for your application.
3. The flask run command will look for a Flask application instance in the module referenced by the FLASK\_APP environment variable, which in our project’s case is 'microblog.py'. The command sets up a web server that is configured to forward requests to this application. When you run flask run without setting the FLASK\_APP environment variable, Flask looks for an application instance in a file named app.py or wsgi.py within the current directory. If it finds the application instance (app), Flask initializes the application.   
   The application context, which includes configuration, routes, and other components, is created only when needed—such as during a request or when explicitly pushed (app.app\_context().push()).
4. The **with** statement can only be used with objects that implement the context manager protocol, meaning they have both \_\_enter\_\_() and \_\_exit\_\_() methods. These objects are called context managers.
5. The **app.shell\_context\_processor** decorator registers the function as a shell context function. When the flask shell command runs, it will invoke this function and register the items returned by it in the shell session. The reason the function returns a dictionary and not a list is that for each item you have to also provide a name under which it will be referenced in the shell, which is given by the dictionary keys.
6. A **stock server machine** just means a server with a **basic, default operating system setup** — like a fresh Ubuntu or CentOS install — with **no customizations or containers**.
7. A **production-scale web server** is a robust, reliable web server designed to handle real-world traffic in production environments. In contrast, something like Flask’s built-in server is **not** production-scale — it’s only for development.
8. Deploying a web application to Heroku is done through the git version control tool, so you must have your application in a git repository. Heroku looks for a file called *Procfile* in the application's root directory for instructions on how to start the application. For Python projects, Heroku also expects a *requirements.txt* file that lists all the module dependencies that need to be installed. After the application is uploaded to Heroku's servers through a git push operation, you are essentially done and just need to wait a few seconds until the application is online.
9. '**python-dotenv**' package is used to load environment variables from a .env (or in our case .flaskenv) file into the environment variables of your project. It reads key-value pairs from a .env file and sets them as environment variables in your application. Flask allows you to register environment variables that you want to be automatically used when you run the flask run command using this 'python-dotenv' package and having all the environment variables in .flaskenv file. While .flaskenv is specifically recognized and loaded by Flask, it's a common practice to store environment variables in a file named .env. Flask does not load .env automatically unless you specifically use a package like python-dotenv to load it. However, if you have a .flaskenv file, Flask will handle this for you.
10. Classes sqlalchemy.schema.Column and sqlalchemy.types.Integer: When we use db.Column or db.Integer, we are actually referring to the above sqlalchemy classes.
11. Applications deployed on production web servers typically listen on port 443, or sometimes 80 if they do not implement encryption, but access to these ports requires administration rights. Since this application is running in a development environment, Flask uses port 5000 by default. Alternatively, you can use the following command if you want to set the port yourself: flask run --port 5001.
12. It is a standard practice to always respond to a POST request generated by a web form submission with a redirect. This helps mitigate an annoyance with how the refresh command is implemented in web browsers. When you hit the refresh key, the web browser just re-issues the last request. If the last request is a POST request with a form submission, then a refresh will re-submit the form. Because this is unexpected, the browser is going to ask the user to confirm the duplicate submission, but most users will not understand what the browser is asking them. If a POST request is answered with a redirect, the browser is instructed to send a GET request to grab the page indicated in the redirect, so now the last request is not a POST request anymore, and the refresh command works in a more predictable way.
13. Elasticsearch is a search engine that lets you store and search huge amounts of text or structured data super-fast. To organize this data, Elasticsearch groups related information into **indices** (similar to tables in a database). Inside each index, you store **documents**, which are individual records represented as JSON objects (like { "name": "Alice", "age": 30 }).   
      
    When you store (or "index") a document, Elasticsearch automatically breaks it down into individual fields and builds optimized search structures behind the scenes. The main structure it creates is an **inverted index**, which maps words ➔ document IDs where the words appear, allowing full-text searches to be lightning-fast. For numeric, date, and geo fields, it also builds structures like **BKD trees** and **doc values** for efficient sorting and aggregations. Thanks to this design, Elasticsearch can find relevant data in huge datasets very quickly.