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.