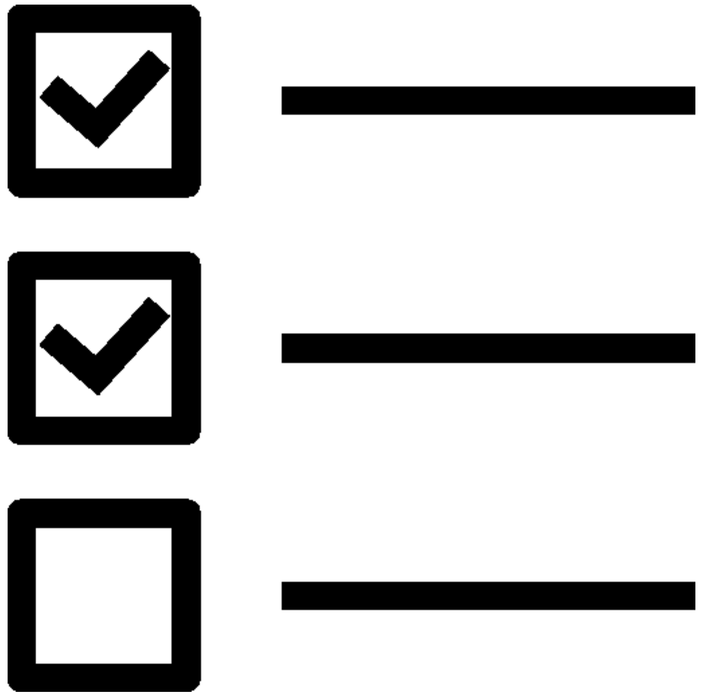
# Web Application Development using Python

## Flask Basics - Part 1

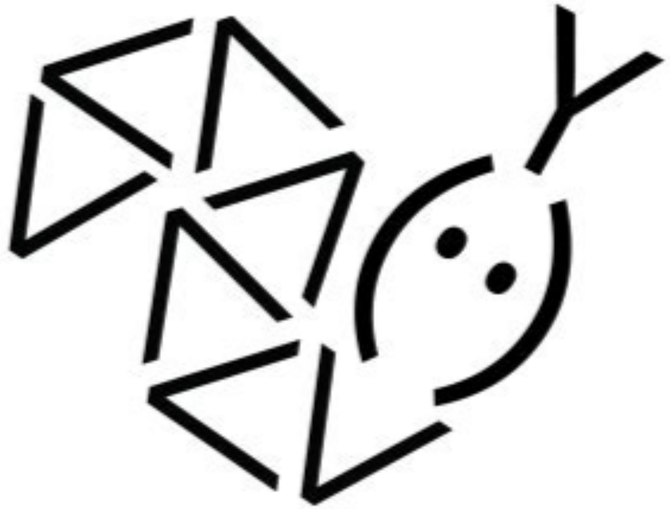
Prepared by George Khoury

# Outline

- **Virtual Environments**
  - What are Virtual Environments?
  - Using Virtual Environments
- **Flask**
  - What is Flask?
  - A Minimal Application
  - Project Structure
  - Flask Basics



# Virtual Environments



# What are Virtual Environments?

- Use a virtual environment to manage the dependencies for your project, both in development and in production.

- **What problem does a virtual environment solve?**
  - Different projects may require different versions of Python libraries, or even Python itself.
  - Newer versions of libraries for one project can break compatibility in another project.
- Virtual environments are independent groups of Python libraries, one for each project. **Packages installed for one project will not affect other projects or the operating system's packages.**
- Python 3 comes bundled with the `venv` module to create virtual environments.

# Using Virtual Environments

## Create an Environment

- To create a virtual environment we can use the `venv` module bundled with Python 3.
  - `python -m venv myenv`
- A common directory location for a virtual environment is `.venv`
  - This name keeps the directory typically hidden in your shell.
  - The name that explains why the directory exists.

# Using Virtual Environments

## Activating / Deactivating an Environment

- Before you can start installing or using packages in your virtual environment you'll need to activate it.
  - We can activate our newly created environment by running:
    - `source .venv/bin/activate`
- If you want to switch projects or otherwise leave your virtual environment, simply run:
  - `deactivate`

# Using Virtual Environments

## Installing Packages

- Now that our virtual environment is active, we can install packages.
- Let's install the Flask library from the Python Package Index (PyPI):
  - `pip install flask`
- As long as your virtual environment is activated pip will install packages into that specific environment and you'll be able to import and use packages in your Python application.

# Flask

# What is Flask?

- Flask is a micro web framework written in Python.
- It is classified as a micro-framework 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.
- Flask supports Python 3.5 and newer.

# What is Flask?

## Dependencies

- **Werkzeug** implements WSGI, the standard Python interface between applications and servers.
- **Jinja** is a template language that renders the pages your application serves.
- **MarkupSafe** comes with Jinja. It escapes untrusted input when rendering templates to avoid injection attacks.
- **ItsDangerous** securely signs data to ensure its integrity. This is used to protect Flask's session cookie.
- **Click** is a framework for writing command line applications. It provides the flask command and allows adding custom management commands.

# A Minimal Application

# A Minimal Application

## hello_flask/

- First we imported the Flask class. An instance of this class will be our WSGI application.
- Next we create an instance of this class. The first argument is the name of the application's module or package. If you are using a single module (as in this example), you should use __name__ because depending on if it's started as application or imported as module the name will be different ('__main__' versus the actual import name). This is needed so that Flask knows where to look for templates, static files, and so on. For more information have a look at the Flask documentation.
- We then use the route() decorator to tell Flask what URL should trigger our function.
- The function is given a name which is also used to generate URLs for that particular function, and returns the message we want to display in the user's browser.

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, World!'
```

# A Minimal Application

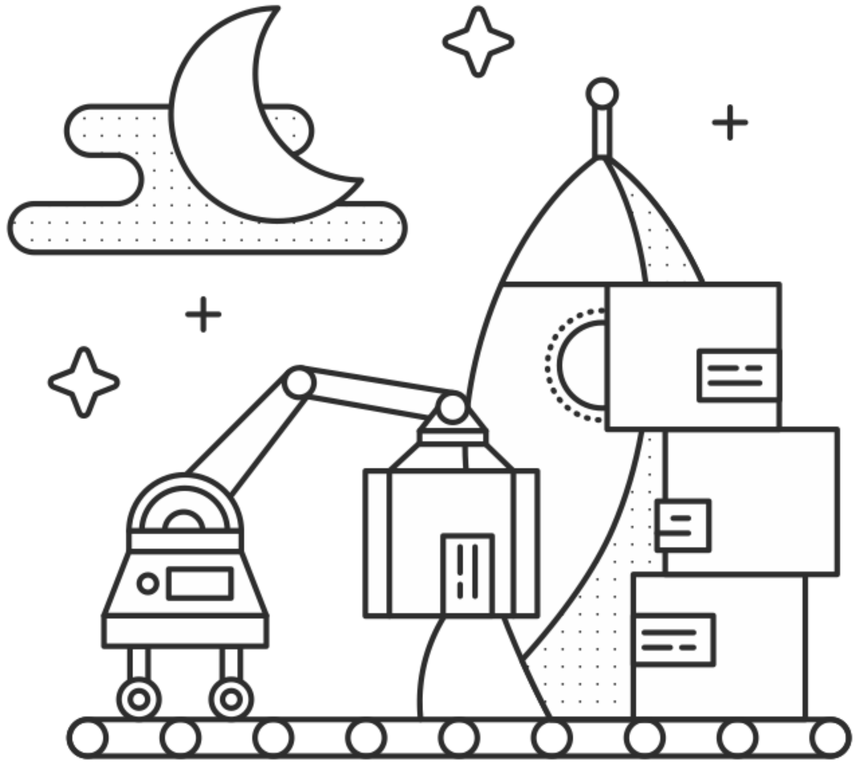
## Running our application

- To run the application you can either use the flask command or python's -m switch with Flask.
- Before you can do that you need to tell your terminal the application to work with by exporting the FLASK_APP environment variable:
  - `export FLASK_APP=hello.py`
  - `flask run`

```python
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, World!'
```

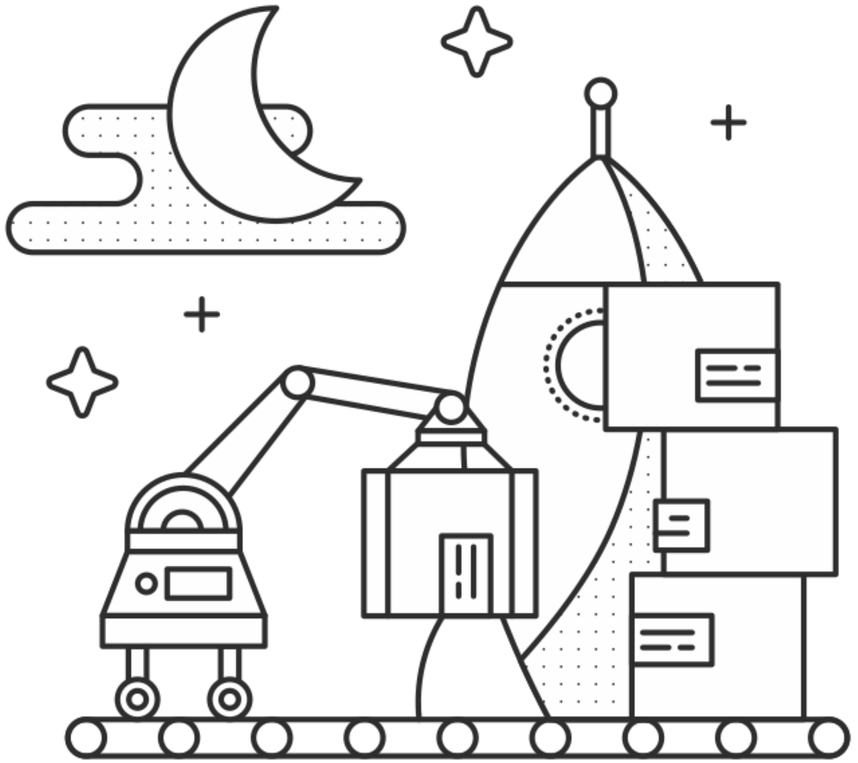
# Project Structure



# Project Structure

## Organization Patterns

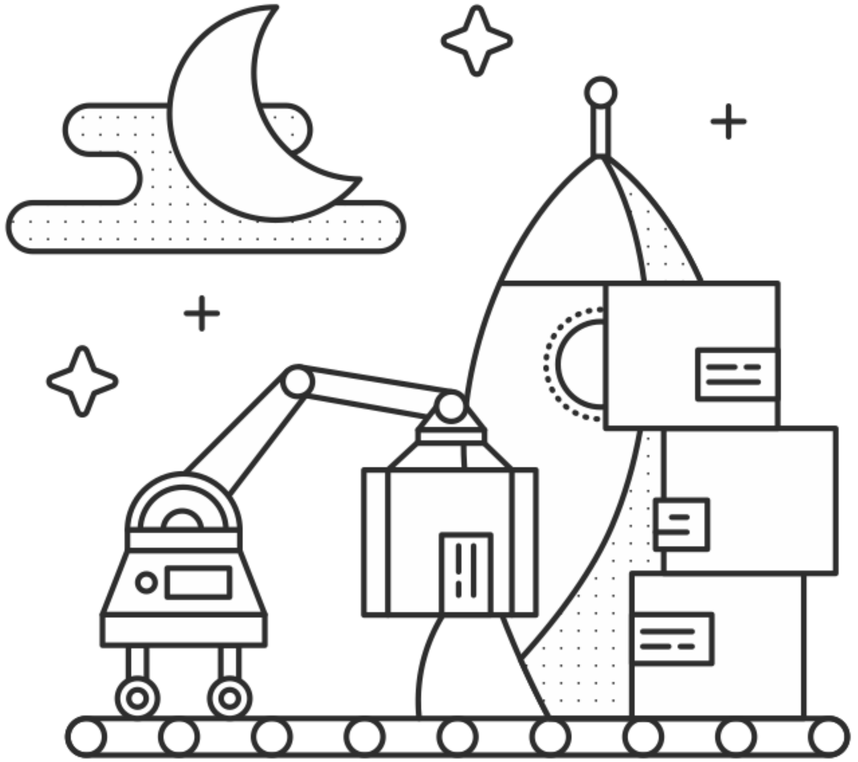
- **Python Module**
  - When you have a few routes.
  - Few hundred lines of code.
- **Python Package**
  - For more complex projects, we can factor out the different components of our app into a group of inter-connected modules — a package.



# Project Structure

## hello_flask/

- Our first application `hello_flask/` is a simple web application with some basic routes.
- We can organize this application as a single module application.
  - `hello_flask/` - our project
    - `app.py` - contains our flask app application
    - `.venv` - our virtual environment



# A Quick Note

- Remember that you can (*and should*) always use Git to keep track of changes to your project.
- Keep your working directory and working tree clean.
- Write concise and informative commit messages.
- Commit often. Remember to push to remote.

# Learning Resources

- https://docs.python.org/3.8/library/venv.html
- https://packaging.python.org/guides/installing-using-pip-and-virtual-environments/#creating-a-virtual-environment
- https://flask.palletsprojects.com/en/1.1.x/installation/#virtual-environments
- https://flask.palletsprojects.com/en/1.1.x/foreword/
- https://flask.palletsprojects.com/en/1.1.x/tutorial/#tutorial