Flask Python micro web framework

START HERE

`flask --app <module> run`
Start the dev server

`render_template("<file>.html", x=1)`
Render a Jinja template

`flask --app <module> run --debug`
Reloader and debugger on

`url_for("<endpoint>")`
Build a URL

`flask run -p <port>`
Use another port

`flask routes`
List all URL rules

`flask shell`
Python shell with app context

`@app.route("/")`
Map a URL to a function

Request from the browser → `URL rule` → **View function** @app.route → `return value` → **Response** str, dict, tuple → `served by` → **Client** HTML or JSON

CONCEPTS

Name your file app.py or wsgi.py and you can drop --app. Never name it flask.py.

request, session and g are context locals: they only work while a request or app context is active.

The dev server is not for production: put the app behind a WSGI server such as gunicorn.

RUN

`flask --app <module> run`
Start the dev server
Serves on http://127.0.0.1:5000 by default.

`flask --app <module> run --debug`
Run with reloader and debugger
Never expose debug mode to untrusted networks.

`flask run -p <port>`
Listen on another port

`flask run --no-reload`
Debugger without the reloader

CONFIG & DISCOVERY

`flask --app <module>:<name> routes`
Pick the app variable explicitly

`flask --app "<module>:create_app('<arg>')" run`
Call an app factory with arguments
Arguments are parsed as Python literals.

`FLASK_APP=<module>`
Environment form of --app

`pip install python-dotenv`
Enable .env and .flaskenv loading
.flaskenv is for public values; keep .env out of git.

SHELL & ROUTES

`flask shell`
Python shell with the app context pushed

`flask routes`
List routes, endpoints and methods

CUSTOM COMMANDS

`@app.cli.command("<name>")`
Register flask <name>
Runs with an app context.

`@click.option("--<flag>")`
Add an option to the command

ROUTING

`@app.route("/")`
Bind a URL to a view

`@app.route("/user/<name>")`
Dynamic URL part

`@app.route("/post/<int:id>")`
Typed converter
Converters: string, int, float, path, uuid.

`@app.route("/login", methods=["GET", "POST"])`
Accept more HTTP methods
Routes answer GET only by default; HEAD and OPTIONS are automatic.

`@app.post("/login")`
Method shortcut decorator
Also get, put, delete, patch.

`url_for("<endpoint>", id=1)`
Build the URL for a view
Unknown keywords become query parameters.

ROUTING (cont.)

`url_for("static", filename="<file>")`
URL of a static file

`Blueprint("<name>", __name__)`
Group routes into a module

`app.register_blueprint(<bp>, url_prefix="/<p>")`
Attach a blueprint

REQUESTS

`request.method`
HTTP method of the request

`request.args.get("<key>")`
Query string value
Use get; [] raises a 400 if missing.

`request.form["<key>"]`
Form field from POST

`request.get_json()`
Parsed JSON body

`request.files["<name>"]`
Uploaded file
Form needs enctype="multipart/form-data".

`file.save(secure_filename(file.filename))`
Store an upload safely
Never trust the client filename.

RESPONSES

`return "<p>Hi</p>"`
String becomes a 200 HTML response

`return {"a": 1}`
Dict or list becomes JSON

`return <body>, 404`
Body plus status code
Also (body, status, headers).

`redirect(url_for("<endpoint>"))`
Redirect to another view

`abort(404)`
Stop with an HTTP error

`resp.set_cookie("<name>", "<value>")`
Set a cookie on a response

`@app.errorhandler(404)`
Custom error page
Return the template and the status, e.g. return render_template(...), 404.

TEMPLATES & STATE

`render_template("<file>.html", x=1)`
Render templates/<file>.html

`{{ x }}`
Print an escaped value in Jinja

TEMPLATES & STATE (cont.)

`{% extends "base.html" %}`
Inherit a layout

`app.secret_key = <bytes>`
Key needed for sessions and flash
Generate one with python -c 'import secrets; print(secrets.token_hex())'.

`session["<key>"] = <value>`
Per-user data in a signed cookie

`flash("<msg>")`
Message for the next request

RECIPES

Minimal app

```
app = Flask(__name__)
@app.route("/")
def hello():
    return "<p>Hi</p>"
```

Dotenv defaults

```
# .flaskenv
FLASK_APP=app
FLASK_RUN_PORT=8000
# .env (not committed)
SECRET_KEY=<random>
```

JSON endpoint

```
@app.post("/api/items")
def add():
    d = request.get_json()
    return {"name": d["name"]},
201
```