Flask

Routes, requests and responses in a few lines of Python.

01 Install & run

```
pip install flask
flask --app app run --debug
```

```
from flask import Flask
app = Flask(__name__)
@app.route('/')
def home():
    return 'Hello World!'
```

--debug reloads on changes and shows a debugger, as does app.run(debug=True). Never in production.

02 Configuration

```
import os
from flask_cors import CORS
app.config['SECRET_KEY'] = os.environ.get(
    'SECRET_KEY', 'dev')
app.json.sort_keys = False  # keep key order
CORS(app, origins=['http://localhost:3000'])
```

SECRET_KEY signs the session cookie. JSON_SORT_KEYS and FLASK_ENV were removed in 2.3: use app.json, and --debug or FLASK_DEBUG.

03 Routes & variable rules

```
@app.route('/users/<int:user_id>')
def get_user(user_id):
    return {'id': user_id}
@app.route('/docs/<path:name>')
def get_doc(name):
    return send_from_directory('docs', name)
```

int converts the value and 404s on anything else; path also matches slashes. Never pass a user path to send_file().

04 HTTP methods

```
@app.route('/posts',
           methods=['GET', 'POST'])
def posts():
    if request.method == 'POST':
        return create_post()
    return get_posts()
```

Without methods a route answers GET only; other verbs get a 405. @app.get() and @app.post() are one-method shortcuts.

05 Request data

request.get_json()	Parsed JSON body
request.args	Query string values
request.form	Form fields
request.files	Uploaded files
request.headers	Request headers

Prefer .get('key'): ['key'] answers 400 when args, form or files lack it. get_json() answers 415 unless the body is JSON.

06 Query strings & uploads

```
q = request.args.get('q', '')
page = request.args.get('page', 1, type=int)
@app.route('/upload', methods=['POST'])
def upload():
    f = request.files['file']
    name = secure_filename(f.filename)
    f.save(f'uploads/{name}')
    return {'saved': name}, 201
```

type=int falls back to the default on bad input. secure_filename (werkzeug.utils) strips ../ from names.

07 Responses & redirects

```
@app.route('/users', methods=['POST'])
def create_user():
    user = save_user(request.get_json())
    return jsonify(user), 201
@app.route('/old-url')
def old_route():
    return redirect(url_for('new_route'))
```

A dict or list is sent as JSON. A tuple adds the status and headers. url_for() builds a URL from a view name.

08 Headers & cookies

```
@app.route('/login')
def login():
    resp = make_response({'status': 'ok'})
    resp.headers['X-Custom'] = 'value'
    resp.set_cookie('session_id', 'abc123',
                    httponly=True)
    return resp
```

Read them back with request.cookies.get().

09 Error handlers

```
@app.errorhandler(404)
def not_found(error):
    return {'error': 'Not found'}, 404
@app.errorhandler(HTTPException)
def http_error(e):
    return {'error': e.name,
            'detail': e.description}, e.code
```

abort(404) stops a request with that error. A handler for Exception also catches HTTP errors (a 405 becomes a 500), so give HTTPException its own.

Flask

Blueprints, databases, auth, tests and deployment.

10 Blueprints

```
users_bp = Blueprint('users', __name__)
@users_bp.route('/<int:user_id>')
def get_user(user_id):
    user = db.get_or_404(User, user_id)
    return user.to_dict()
app.register_blueprint(
    users_bp, url_prefix='/api/users')
```

Endpoints get the blueprint prefix: url_for('users.get_user', user_id=1). A blueprint 404 handler only sees abort(404) from its own views.

11 Setup & models

```
app.config['SQLALCHEMY_DATABASE_URI'] = \
    os.environ['DATABASE_URL']
db = SQLAlchemy(app)
class User(db.Model):
    id = db.Column(db.Integer,
                   primary_key=True)
    username = db.Column(db.String(80))
```

Give models a to_dict(): jsonify() can't serialize them. db.relationship('Post', backref='author') links models.

12 Queries & CRUD

```
user = User(username=data['username'])
db.session.add(user)
db.session.commit()
user = db.session.get(User, user_id)
user = db.get_or_404(User, user_id)
users = db.session.scalars(db.select(User))
```

User.query is the legacy API. Outside a request, run db calls in a with app.app_context() block.

13 Tables & migrations

`db.create_all()`	Create missing tables
`flask db init`	Start a migrations/ folder
`flask db migrate`	Write a migration
`flask db upgrade`	Apply it to the database

create_all() never alters an existing table. The flask db commands come from Flask-Migrate: Migrate(app, db).

14 Request hooks

```
@app.before_request
def before():
    g.start_time = time.time()
@app.after_request
def after(response):
    took = time.time() - g.start_time
    response.headers['X-Time'] = str(took)
    return response
```

g holds data for one request. A before_request that returns a response ends the request early.

15 Flask-Login

```
login_manager = LoginManager(app)
@login_manager.user_loader
def load_user(uid):
    return db.session.get(User, int(uid))
@app.route('/api/protected')
@login_required
def protected():
    return {'user': current_user.username}
```

User needs UserMixin. Without login_view, @login_required answers 401. logout_user() logs out.

16 Passwords & log in

```
class User(UserMixin, db.Model):
    def set_password(self, pw):
        self.password_hash = \
            generate_password_hash(pw)
    def check_password(self, pw):
        return check_password_hash(
            self.password_hash, pw)
```

To log in: if user.check_password(pw): login_user(user).

17 Testing with pytest

```
os.environ['DATABASE_URL'] = 'sqlite://'
from app import app, db  # import after
@pytest.fixture
def client():
    with app.app_context():
        db.create_all()
    return app.test_client()
def test_home(client):
    res = client.get('/')
    assert res.status_code == 200
```

SQLAlchemy(app) reads the config once: a URI set after the import is ignored, so tests would hit the real database.

18 Deploy with Gunicorn

```
FROM python:3.14-slim
WORKDIR /app
COPY . .
RUN pip install -r requirements.txt
CMD ["gunicorn", "-b", "0.0.0.0", "app:app"]
```

-b 0.0.0.0 listens on port 8000; -w 4 runs 4 workers. Put Nginx in front, with ProxyFix to trust X-Forwarded headers.