

FLASK QUICK REFERENCE

Routes, templates, requests, blueprints, database, extensions

Setup

Minimal App

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

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

Run the App

```
pip install flask
flask -app app run --debug
# or: python -m flask run --debug
```

Project Structure

app.py	Application entry point
templates/	Jinja2 HTML templates
static/	CSS, JS, images
models.py	Database models
requirements.txt	Python dependencies

Routes

Basic Routes

```
@app.route('/about')
def about():
    return render_template('about.html')

@app.route('/user/<username>')
def profile(username):
    return f'User: {username}'
```

URL Variables

<variable>	String (default)
<int:id>	Integer
<float:price>	Float
<path:subpath>	String with slashes
<uuid:item_id>	UUID

HTTP Methods

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        return do_login()
    return render_template('login.html')
```

URL Building

```
from flask import url_for
url_for('profile', username='alice')
# => /user/alice
```

Templates

Render Template

```
from flask import render_template

@app.route('/posts')
def posts():
    items = get_posts()
    return render_template('posts.html', posts=items)
```

Jinja2 Syntax

```
{% variable %}
{% if user %}Welcome, {{ user.name }}!{% endif %}
{% for item in items %}
    <li>{{ item }}</li>
{% endfor %}
```

Template Inheritance

```
{# base.html #}
<html><body>{% block content %}{% endblock %}</body></html>

{# child.html #}
{% extends "base.html" %}
{% block content %}<h1>Page</h1>{% endblock %}
```

Common Filters

 safe	Render raw HTML
 escape	HTML-escape string
 length	Count items
 default('N/A')	Fallback for empty
 tojson	Serialize to JSON

Request & Response

Request Object

```
from flask import request

request.method # 'GET', 'POST'
request.args.get('q') # query string ?q=value
request.form['name'] # form POST data
request.json # parsed JSON body
```

Request Properties

request.args	Query string parameters
request.form	Form POST data
request.json	Parsed JSON body
request.files	Uploaded files
request.headers	HTTP headers
request.cookies	Cookie values

Response Helpers

```
from flask import jsonify, redirect, make_response

return jsonify({'status': 'ok'}) # JSON response
return redirect(url_for('index')) # redirect
resp = make_response('body', 200)
resp.headers['X-Custom'] = 'value'
```

Session

```
from flask import session
app.secret_key = 'your-secret-key'
session['user_id'] = 42
uid = session.get('user_id')
```

Forms

WTForms Integration

```
pip install flask-wtf
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField
from wtforms.validators import DataRequired
```

Define Form

```
class LoginForm(FlaskForm):
    username = StringField('User', validators=[DataRequired()])
    password = PasswordField('Pass', validators=[DataRequired()])
```

Use in View

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    form = LoginForm()
    if form.validate_on_submit():
        user = form.username.data
        return redirect(url_for('dashboard'))
    return render_template('login.html', form=form)
```

Form in Template

```
<form method="post">
  {{ form.hidden_tag() }}
  {{ form.username.label }} {{ form.username() }}
  {{ form.password.label }} {{ form.password() }}
  <button type="submit">Login</button>
</form>
```

Database

SQLAlchemy Setup

```
pip install flask-sqlalchemy
from flask_sqlalchemy import SQLAlchemy
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///app.db'
db = SQLAlchemy(app)
```

Define Model

```
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(80), nullable=False)
    email = db.Column(db.String(120), unique=True)
    posts = db.relationship('Post', backref='author')
```

CRUD Operations

```
user = User(name='Alice', email='alice@example.com')
db.session.add(user)
db.session.commit()
User.query.filter_by(name='Alice').first()
db.session.delete(user)
db.session.commit()
```

Common Queries

Model.query.all()	All records
Model.query.get(id)	By primary key
filter_by(name='X')	Simple equality filter
filter(Model.age > 18)	Expression filter
order_by(Model.name)	Sort results
limit(10).offset(20)	Paginate results

Blueprints

Create Blueprint

```
from flask import Blueprint
blog = Blueprint('blog', __name__, url_prefix='/blog')

@app.route('/')
def index():
    return render_template('blog/index.html')
```

Register Blueprint

```
# app.py
from blog import blog
app.register_blueprint(blog)
```

Blueprint URL Building

```
url_for('blog.index') # => '/blog/'
url_for('blog.post', id=5) # => '/blog/post/5'
```

Blueprint Structure

url_prefix	Prefix all routes in blueprint
template_folder	Custom template directory
static_folder	Blueprint-specific static files
@bp.before_request	Run before each blueprint request

Error Handling

Custom Error Pages

```
@app.errorhandler(404)
def not_found(e):
    return render_template('404.html'), 404

@app.errorhandler(500)
def server_error(e):
    return render_template('500.html'), 500
```

Abort Requests

```
from flask import abort

@app.route('/admin')
def admin():
    if not current_user.is_admin:
        abort(403)
    return render_template('admin.html')
```

Custom Exceptions

```
from werkzeug.exceptions import HTTPException

class InsufficientFunds(HTTPException):
    code = 402
    description = 'Insufficient funds'
```

Logging

```
app.logger.info('User %s logged in', username)
app.logger.warning('Disk space low')
app.logger.error('Payment failed: %s', err)
```

Configuration

Config Methods

```
app.config['DEBUG'] = True
app.config.from_object('config.ProductionConfig')
app.config.from_envvar('APP_SETTINGS')
```

Config Class Pattern

```
class Config:
    SECRET_KEY = os.environ.get('SECRET_KEY')
    SQLALCHEMY_TRACK_MODIFICATIONS = False

class DevConfig(Config):
    DEBUG = True
    SQLALCHEMY_DATABASE_URI = 'sqlite:///dev.db'
```

Common Settings

SECRET_KEY	Session signing key (required)
DEBUG	Enable debug mode
TESTING	Enable test mode
SQLALCHEMY_DATABASE_URI	Database connection string
MAX_CONTENT_LENGTH	Max upload size in bytes
JSON_SORT_KEYS	Sort JSON output keys

Extensions

Popular Extensions

Flask-SQLAlchemy	ORM integration
Flask-Migrate	Alembic database migrations
Flask-WTF	Form handling with CSRF
Flask-Login	User session management
Flask-Mail	Email sending
Flask-CORS	Cross-origin resource sharing
Flask-RESTful	REST API building
Flask-Caching	Response and function caching

Flask-Login

```
from flask_login import LoginManager, login_required
login_manager = LoginManager(app)
login_manager.login_view = 'login'

@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))
```

Flask-Migrate

```
from flask_migrate import Migrate
migrate = Migrate(app, db)
# flask db init (once)
# flask db migrate -m "add users"
# flask db upgrade
```