Referencia Rápida de Flask

Rutas, plantillas, solicitudes, blueprints, base de datos, extensiones

Configuración

App mínima

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

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

Ejecutar la app

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

Estructura del proyecto

app.py	Punto de entrada de la aplicación
templates/	Plantillas HTML Jinja2
static/	CSS, JS, imágenes
models.py	Modelos de base de datos
requirements.txt	Dependencias Python

Rutas

Rutas básicas

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

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

Variables de URL

<variable>	Cadena (por defecto)
<int:id>	Entero
<float:price>	Float
<path:subpath>	Cadena con barras
<uuid:item_id>	UUID

Métodos HTTP

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

Construcción de URL

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

Plantillas

Renderizar plantilla

```
from flask import render_template

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

Sintaxis Jinja2

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

Herencia de plantillas

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

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

Filtros comunes

|safe	Renderizar HTML sin escapar
|escape	Escapar cadena HTML
|length	Contar elementos
|default('N/A')	Valor por defecto para vacíos
|tojson	Serializar a JSON

Solicitud y respuesta

Objeto Request

```
from flask import request

request.method          # 'GET', 'POST'
request.args.get('q')   # query string ?q=value
request.form['name']    # datos de formulario POST
request.json            # cuerpo JSON parseado
```

Propiedades de Request

request.args	Parámetros de query string
request.form	Datos de formulario POST
request.json	Cuerpo JSON parseado
request.files	Archivos subidos
request.headers	Cabeceras HTTP
request.cookies	Valores de cookies

Helpers de respuesta

```
from flask import jsonify, redirect, make_response

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

Sesión

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

Formularios

Integración con WTForms

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

Definir formulario

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

Usar en vista

```
@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)
```

Formulario en plantilla

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

Base de datos

Configuración de SQLAlchemy

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

Definir modelo

```
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')
```

Operaciones CRUD

```
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()
```

Consultas comunes

Model.query.all()	Todos los registros
Model.query.get(id)	Por clave primaria
.filter_by(name='X')	Filtro de igualdad simple
.filter(Model.age > 18)	Filtro de expresión
.order_by(Model.name)	Ordenar resultados
.limit(10).offset(20)	Paginar resultados

Blueprints

Crear Blueprint

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

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

Registrar Blueprint

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

Construcción de URL con Blueprint

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

Referencia Rápida de Flask

Estructura de Blueprint

`url_prefix`	Prefijo para todas las rutas del blueprint
`template_folder`	Directorio de plantillas personalizado
`static_folder`	Archivos estáticos específicos del blueprint
`@bp.before_request`	Ejecutar antes de cada solicitud del blueprint

Manejo de errores

Páginas de error personalizadas

```
@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
```

Abortar solicitudes

```
from flask import abort

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

Excepciones personalizadas

```
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)
```

Configuración

Métodos de configuración

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

Patrón de clase de configuración

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

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

Ajustes comunes

`SECRET_KEY`	Clave de firma de sesión (requerida)
`DEBUG`	Habilitar modo debug
`TESTING`	Habilitar modo de prueba
`SQLALCHEMY_DATABASE_URI`	Cadena de conexión a la base de datos
`MAX_CONTENT_LENGTH`	Tamaño máximo de carga en bytes
`JSON_SORT_KEYS`	Ordenar claves de salida JSON

Extensiones

Extensiones populares

`Flask-SQLAlchemy`	Integración ORM
`Flask-Migrate`	Migraciones de base de datos con Alembic
`Flask-WTF`	Manejo de formularios con CSRF
`Flask-Login`	Gestión de sesiones de usuario
`Flask-Mail`	Envío de emails
`Flask-CORS`	Compartir recursos entre orígenes
`Flask-RESTful`	Construcción de APIs REST
`Flask-Caching`	Caché de respuestas y funciones

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    (una vez)
# flask db migrate -m "add users"
# flask db upgrade
```