Riferimento Rapido Flask

Route, template, richieste, blueprint, database, estensioni

Configurazione

App Minimale

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

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

Eseguire l'App

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

Struttura del Progetto

app.py	Punto di ingresso dell'applicazione
templates/	Template HTML Jinja2
static/	CSS, JS, immagini
models.py	Modelli database
requirements.txt	Dipendenze Python

Route

Route Base

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

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

Variabili URL

<variable>	Stringa (default)
<int:id>	Intero
<float:price>	Float
<path:subpath>	Stringa con slash
<uuid:item_id>	UUID

Metodi HTTP

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

Costruzione URL

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

Template

Renderizza Template

```
from flask import render_template

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

Sintassi Jinja2

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

Ereditarietà Template

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

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

Filtri Comuni

|safe	Renderizza HTML grezzo
|escape	Escaping HTML della stringa
|length	Conta elementi
|default('N/A')	Fallback per valori vuoti
|tojson	Serializza in JSON

Richiesta e Risposta

Oggetto Request

```
from flask import request

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

Proprietà Request

request.args	Parametri query string
request.form	Dati form POST
request.json	Body JSON parsato
request.files	File caricati
request.headers	Header HTTP
request.cookies	Valori cookie

Helper Risposta

```
from flask import jsonify, redirect, make_response

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

Sessione

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

Form

Integrazione WTForms

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

Definire un Form

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

Utilizzo nella 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 nel Template

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

Database

Configurazione SQLAlchemy

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

Definire un Modello

```
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')
```

Operazioni 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()
```

Query Comuni

Model.query.all()	Tutti i record
Model.query.get(id)	Per chiave primaria
.filter_by(name='X')	Filtro semplice per uguaglianza
.filter(Model.age > 18)	Filtro con espressione
.order_by(Model.name)	Ordina i risultati
.limit(10).offset(20)	Pagina i risultati

Blueprint

Crea Blueprint

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

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

Registra Blueprint

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

Costruzione URL Blueprint

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

Riferimento Rapido Flask

Struttura Blueprint

`url_prefix`	Prefisso tutte le route nel blueprint
`template_folder`	Directory template personalizzata
`static_folder`	File statici specifici del blueprint
`@bp.before_request`	Esegui prima di ogni richiesta del blueprint

Gestione Errori

Pagine Errore Personalizzate

```
@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
```

Interrompere Richieste

```
from flask import abort

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

Eccezioni Personalizzate

```
from werkzeug.exceptions import HTTPException

class InsufficientFunds(HTTPException):
    code = 402
    description = 'Fondi insufficienti'
```

Logging

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

Configurazione

Metodi di Config

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

Pattern Classe Config

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

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

Impostazioni Comuni

`SECRET_KEY`	Chiave di firma sessione (obbligatorio)
`DEBUG`	Abilita modalità debug
`TESTING`	Abilita modalità test
`SQLALCHEMY_DATABASE_URI`	Stringa di connessione database
`MAX_CONTENT_LENGTH`	Dimensione massima upload in byte
`JSON_SORT_KEYS`	Ordina le chiavi nell'output JSON

Estensioni

Estensioni Popolari

`Flask-SQLAlchemy`	Integrazione ORM
`Flask-Migrate`	Migrazioni database con Alembic
`Flask-WTF`	Gestione form con CSRF
`Flask-Login`	Gestione sessioni utente
`Flask-Mail`	Invio email
`Flask-CORS`	Condivisione risorse cross-origin
`Flask-RESTful`	Costruzione API REST
`Flask-Caching`	Cache risposte e funzioni

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 volta)
# flask db migrate -m "add users"
# flask db upgrade
```