Referência Rápida Flask

Rotas, templates, requisições, blueprints, banco de dados, extensões

Configuração

App Mínima

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

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

Executar a App

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

Estrutura do Projeto

app.py	Ponto de entrada da aplicação
templates/	Templates HTML Jinja2
static/	CSS, JS, imagens
models.py	Modelos de banco de dados
requirements.txt	Dependências Python

Rotas

Rotas Básicas

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

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

Variáveis de URL

<variable>	String (padrão)
<int:id>	Inteiro
<float:price>	Float
<path:subpath>	String com 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')
```

Construção de URL

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

Templates

Renderizar Template

```
from flask import render_template

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

Sintaxe Jinja2

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

Herança de Template

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

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

Filtros Comuns

|safe	Renderizar HTML bruto
|escape	Escapar string HTML
|length	Contar itens
|default('N/A')	Fallback para vazio
|tojson	Serializar para JSON

Requisição e Resposta

Objeto Request

```
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
```

Propriedades do Request

request.args	Parâmetros de query string
request.form	Dados POST de formulário
request.json	Corpo JSON parseado
request.files	Arquivos enviados
request.headers	Headers HTTP
request.cookies	Valores de cookie

Helpers de Resposta

```
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'
```

Sessão

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

Formulários

Integração WTForms

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

Definir Formulário

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

Usar na 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)
```

Formulário no Template

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

Banco de Dados

Configuração SQLAlchemy

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

Definir 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')
```

Operações 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()
```

Queries Comuns

Model.query.all()	Todos os registros
Model.query.get(id)	Por chave primária
.filter_by(name='X')	Filtro de igualdade simples
.filter(Model.age > 18)	Filtro de expressão
.order_by(Model.name)	Ordenar resultados
.limit(10).offset(20)	Paginar resultados

Blueprints

Criar 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)
```

Construção de URL no Blueprint

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

Referência Rápida Flask

Estrutura do Blueprint

`url_prefix`	Prefixar todas as rotas do blueprint
`template_folder`	Diretório de templates personalizado
`static_folder`	Arquivos estáticos específicos do blueprint
`@bp.before_request`	Executar antes de cada requisição do blueprint

Tratamento de Erros

Páginas de Erro 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 Requisições

```
from flask import abort

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

Exceções 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)
```

Configuração

Métodos de Config

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

Padrão de 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'
```

Configurações Comuns

`SECRET_KEY`	Chave de assinatura de sessão (obrigatória)
`DEBUG`	Habilitar modo debug
`TESTING`	Habilitar modo de teste
`SQLALCHEMY_DATABASE_URI`	String de conexão com banco de dados
`MAX_CONTENT_LENGTH`	Tamanho máximo de upload em bytes
`JSON_SORT_KEYS`	Ordenar chaves de saída JSON

Extensões

Extensões Populares

`Flask-SQLAlchemy`	Integração ORM
`Flask-Migrate`	Migrações de banco com Alembic
`Flask-WTF`	Tratamento de formulários com CSRF
`Flask-Login`	Gerenciamento de sessão de usuário
`Flask-Mail`	Envio de email
`Flask-CORS`	Compartilhamento de recursos de origem cruzada
`Flask-RESTful`	Construção de API REST
`Flask-Caching`	Cache de resposta e função

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
```