

# 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

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

<b>&lt;variable&gt;</b>	String (padrão)
<b>&lt;int:id&gt;</b>	Inteiro
<b>&lt;float:price&gt;</b>	Float
<b>&lt;path:subpath&gt;</b>	String com barras
<b>&lt;uuid:item_id&gt;</b>	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

<b> safe</b>	Renderizar HTML bruto
<b> escape</b>	Escapar string HTML
<b> length</b>	Contar itens
<b> default('N/A')</b>	Fallback para vazio
<b> tojson</b>	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

<b>request.args</b>	Parâmetros de query string
<b>request.form</b>	Dados POST de formulário
<b>request.json</b>	Corpo JSON parseado
<b>request.files</b>	Arquivos enviados
<b>request.headers</b>	Headers HTTP
<b>request.cookies</b>	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

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

<b>url_prefix</b>	Prefixar todas as rotas do blueprint
<b>template_folder</b>	Diretório de templates personalizado
<b>static_folder</b>	Arquivos estáticos específicos do blueprint
<b>@bp.before_request</b>	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

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

## Extensões

### Extensões Populares

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