Référence rapide Flask

Routes, templates, requêtes, blueprints, base de données, extensions

Configuration

Application minimale

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

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

Lancer l'application

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

Structure du projet

app.py	Point d'entrée de l'application
templates/	Templates HTML Jinja2
static/	CSS, JS, images
models.py	Modèles de base de données
requirements.txt	Dépendances Python

Routes

Routes de base

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

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

Variables d'URL

<variable>	Chaîne (défaut)
<int:id>	Entier
<float:price>	Nombre flottant
<path:subpath>	Chaîne avec barres obliques
<uuid:item_id>	UUID

Méthodes HTTP

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

Construction d'URLs

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

Templates

Rendu de template

```
from flask import render_template

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

Syntaxe Jinja2

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

Héritage de templates

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

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

Filtres courants

|safe	Rendre le HTML brut
|escape	Échapper la chaîne HTML
|length	Compter les éléments
|default('N/A')	Valeur de repli pour les champs vides
|tojson	Sérialiser en JSON

Requête et réponse

Objet requête

```
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
```

Propriétés de la requête

request.args	Paramètres de chaîne de requête
request.form	Données POST du formulaire
request.json	Corps JSON analysé
request.files	Fichiers téléversés
request.headers	En-têtes HTTP
request.cookies	Valeurs des cookies

Helpers de réponse

```
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')
```

Formulaires

Intégration WTForms

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

Définir un formulaire

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

Utilisation dans la vue

```
@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)
```

Formulaire dans le template

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

Base de données

Configuration SQLAlchemy

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

Définir un modèle

```
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')
```

Opérations 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()
```

Requêtes courantes

Model.query.all()	Tous les enregistrements
Model.query.get(id)	Par clé primaire
.filter_by(name='X')	Filtre d'égalité simple
.filter(Model.age > 18)	Filtre par expression
.order_by(Model.name)	Trier les résultats
.limit(10).offset(20)	Paginer les résultats

Blueprints

Créer un blueprint

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

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

Enregistrer un blueprint

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

Construction d'URL de blueprint

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

Référence rapide Flask

Structure du blueprint

`url_prefix`	Préfixer toutes les routes du blueprint
`template_folder`	Répertoire de templates personnalisé
`static_folder`	Fichiers statiques spécifiques au blueprint
`@bp.before_request`	Exécuter avant chaque requête du blueprint

Gestion des erreurs

Pages d'erreur personnalisées

```
@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
```

Abandonner des requêtes

```
from flask import abort

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

Exceptions personnalisées

```
from werkzeug.exceptions import HTTPException

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

Journalisation

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

Configuration

Méthodes de configuration

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

Pattern de classe de configuration

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

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

Paramètres courants

`SECRET_KEY`	Clé de signature de session (requise)
`DEBUG`	Activer le mode debug
`TESTING`	Activer le mode test
`SQLALCHEMY_DATABASE_URI`	Chaîne de connexion à la base de données
`MAX_CONTENT_LENGTH`	Taille maximale d'upload en octets
`JSON_SORT_KEYS`	Trier les clés de sortie JSON

Extensions

Extensions populaires

`Flask-SQLAlchemy`	Intégration ORM
`Flask-Migrate`	Migrations de base de données avec Alembic
`Flask-WTF`	Gestion des formulaires avec CSRF
`Flask-Login`	Gestion des sessions utilisateur
`Flask-Mail`	Envoi d'emails
`Flask-CORS`	Partage de ressources cross-origin
`Flask-RESTful`	Construction d'API REST
`Flask-Caching`	Mise en cache des réponses et des fonctions

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
```