

FLASK クイックリファレンス

ルート、テンプレート、リクエスト、ブループリント、データベース、拡張

セットアップ

最小アプリ

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

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

アプリの起動

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

プロジェクト構造

app.py	アプリケーションのエントリーポイント
templates/	Jinja2 HTML テンプレート
static/	CSS、JS、画像
models.py	データベースモデル
requirements.txt	Python の依存関係

ルート

基本ルート

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

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

URL 変数

<variable>	文字列 (デフォルト)
<int:id>	整数
<float:price>	浮動小数点
<path:subpath>	スラッシュを含む文字列
<uuid:item_id>	UUID

HTTP メソッド

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

URL の構築

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

テンプレート

テンプレートのレンダリング

```
from flask import render_template

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

Jinja2 構文

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

テンプレート継承

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

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

よく使うフィルター

 safe	生の HTML をレンダリング
 escape	文字列を HTML エスケープ
 length	アイテムを数える
 default('N/A')	空の値のフォールバック
 tojson	JSON にシリアライズ

リクエストとレスポンス

リクエストオブジェクト

```
from flask import request

request.method # 'GET', 'POST'
request.args.get('q') # クエリ文字列 ?q=value
request.form['name'] # フォーム POST データ
request.json # パース済み JSON ボディ
```

リクエストプロパティ

request.args	クエリ文字列パラメーター
request.form	フォーム POST データ
request.json	パース済み JSON ボディ
request.files	アップロードされたファイル
request.headers	HTTP ヘッダー
request.cookies	Cookie 値

レスポンスのヘルパー

```
from flask import jsonify, redirect, make_response

return jsonify({'status': 'ok'}) # JSON レスポンス
return redirect(url_for('index')) # リダイレクト
resp = make_response('body', 200) # リダイレクト
resp.headers['X-Custom'] = 'value'
```

セッション

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

フォーム

WTForms の統合

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

フォームの定義

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

ビューでの使用

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    form = LoginForm()
    if form.validate_on_submit():
        user = form.username.data
        password = form.password.data
        return redirect(url_for('dashboard'))
    return render_template("login.html", form=form)
```

テンプレートでのフォーム

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

データベース

SQLAlchemy のセットアップ

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

モデルの定義

```
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')
```

CRUD 操作

```
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()
```

よく使うクエリ

Model.query.all()	すべてのレコード
Model.query.get(id)	プライマリキーで取得
filter_by(name='X')	シンプルな等値フィルター
filter(Model.age > 18)	式フィルター
order_by(Model.name)	結果をソート
limit(10).offset(20)	結果をページネート

ブループリント

ブループリントの作成

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

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

ブループリントの登録

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

ブループリントの URL 構築

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

ブループリントの構造

url_prefix	ブループリント内のすべてのルートにレディックスを付加
template_folder	カスタムテンプレートディレクトリ
static_folder	ブループリント固有の静的ファイル
@bp.before_request	各ブループリントリクエストの前に実行

エラーハンドリング

カスタムエラーページ

```
@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
```

リクエストの中断

```
from flask import abort
```

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

カスタム例外

```
from werkzeug.exceptions import HTTPException

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

ロギング

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

設定

設定方法

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

設定クラスパターン

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

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

よく使う設定

SECRET_KEY	セッション署名キー (必須)
DEBUG	デバッグモードを有効化
TESTING	テストモードを有効化
SQLALCHEMY_DATABASE_URI	データベース接続文字列
MAX_CONTENT_LENGTH	最大アップロードサイズ (バイト)
JSON_SORT_KEYS	JSON 出力のキーをソート

拡張

人気の拡張

Flask-SQLAlchemy	ORM 統合
Flask-Migrate	Alembic データベースマイグレーション
Flask-WTF	CSRF 付きフォーム処理
Flask-Login	ユーザーセッション管理
Flask-Mail	メール送信
Flask-CORS	クロスオリジンリソース共有
Flask-RESTful	REST API の構築
Flask-Caching	レスポンスと関数のキャッシング

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 (最初の一度)
# flask db migrate -m "add users"
# flask db upgrade
```