

# Linguagem de Programação Orientada a Objetos I

Flask

Prof. Tales Bitelo Viegas

<https://fb.com/ProfessorTalesViegas>

# O que é o Flask

- ▶ Em 2004 foi criado o grupo Pocoo, formado por entusiastas Python que trabalham em cima de diversos projetos Python
  - Flask
  - Jinja2
  - Pygments
  - Sphinx
  - Werkzeug WSGI

# O que é o Flask

- ▶ Um micro-framework Web para Python
  - Núcleo simples, mas extensível
  - Não há camada de abstração de BD
  - Não valida formulários
  - Feito através de extensões
- ▶ Baseado na biblioteca WSGI Werkzeug e Jinja2
- ▶ Utilizado por Pinterest e LinkedIn

# O que é o Flask



- ▶ Foi criado pelo austríaco Armin Ronacher, do Pocoo, em 2010, quando ele tinha 21 anos, para fazer uma piada de Primeiro de Abril com micro-frameworks
- ▶ <http://lucumr.pocoo.org/2010/4/3/april-1st-post-mortem/>

# Passo a Passo

- ▶ Instale o Flask através do gerenciador de pacotes PIP (Python Package Index)
  - `pip install flask`

# Aplicação rodando

- ▶ Crie uma instância da aplicação

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

# Defina as rotas

---

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def index:
    return '<h1>Hello World</h1>'

@app.route('/user/<name>')
def user(name):
    return '<h1>Hello, {0}!</h1>'.format(name)
```

# Inicialize o Servidor

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def index:
    return '<h1>Hello World</h1>'

@app.route('/user/<name>')
def user(name):
    return '<h1>Hello, {0}!</h1>'.format(name)

if __name__ == '__main__':
    app.run(debug=True)
```

```
/Users/talesviegas/PycharmProjects/AulaFlask/venv/bin/python -m flask run
 * Serving Flask app "app"
 * Forcing debug mode off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

# Decorators

- ▶ Decorators são usados pelo Framework e extensões para registrar funções da aplicação e callbacks
- ▶ Principais Decorators:
  - **route** – Registra funções para tratar rotas
  - **before\_request** – Registra uma função para executar antes do tratador de requisições
  - **before\_first\_request** – Igual ao anterior, mas irá executar apenas após a primeira requisição ao servidor
  - **after\_request** – Registra uma função para executar após o tratador de requisições
  - **teardown\_request** – Registra uma função para executar após o tratador de requisições, mesmo se uma exceção ocorrer
  - **errorhandler** – Define um tratador de erros

# Contextos Globais

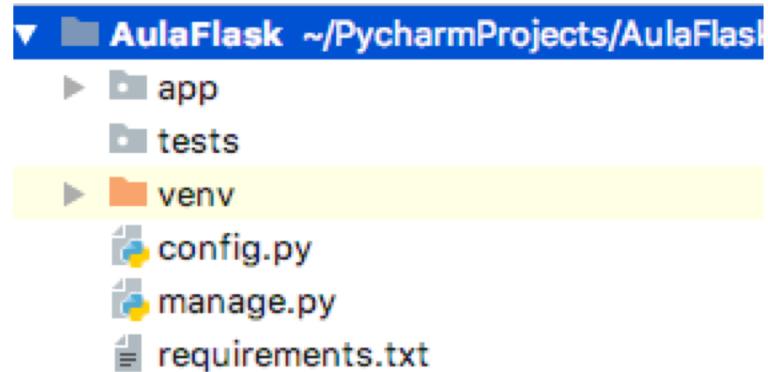
- ▶ Evitam a necessidade de passar parâmetros aos tratadores de requisições
- ▶ Application Context
  - **current\_app** – Instância da aplicação
  - **g** – Dicionário global para armazenamento de dados
- ▶ Request Context
  - **request** – A requisição que está sendo processada
  - **session** – A sessão do usuário corrente

# Funções Auxiliares

- ▶ **url\_for()** – Gera link para rotas ou arquivos estáticos
- ▶ **render\_template()** – Renderiza templates Jinja2
- ▶ **redirect()** – Gera uma resposta de redirect
- ▶ **jsonify()** – Gera um JSON como resposta
- ▶ **abort()** – Gera uma resposta de erro (lança exceção)
- ▶ **flash()** – Registra uma mensagem para mostrar ao usuário

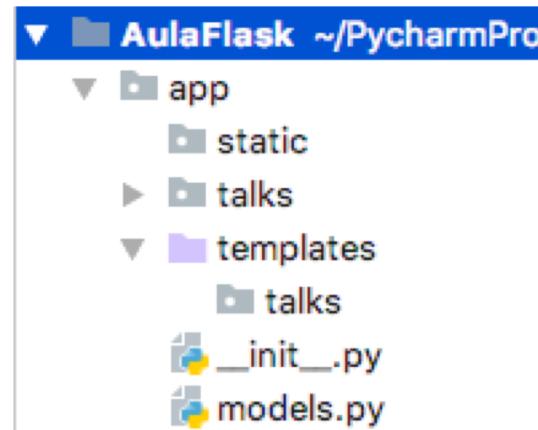
# Na prática - Versão 0.1

- ▶ Estrutura do Projeto (sugestão)
  - app – Pacote da aplicação
  - tests – Testes unitários
  - venv – Virtual Environment
  - requirements.txt – Dependências
  - config.py – Configurações
  - manage.py – Script de inicialização



# O pacote de Aplicação

- ▶ Templates e arquivos estáticos ganham uma pasta dedicada
- ▶ Blueprints são implementados em sub-pacotes, e também uma sub-pasta em templates
- ▶ Funcionalidades comuns, como modelos, são implementadas como módulos



# Configuração

- ▶ A classe básica de configuração possui configurações básicas, que podem ser sobrescritas em subclasses se necessário

```
import os

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

class DevelopmentConfig(Config):
    DEBUG = True
    SECRET_KEY = os.environ.get('SECRET_KEY') or 'SeGr&d0'

class TestingConfig(Config):
    TESTING = True

class ProductionConfig(Config):
    pass

config = {
    'development': DevelopmentConfig,
    'testing': TestingConfig,
    'production': ProductionConfig,
    'default': DevelopmentConfig
}
```

# Application Factory Pattern

- ▶ A instância da aplicação é criada e configurada no momento da execução
- ▶ Rotas são importadas de um Blueprint

```
from flask import Flask
from config import config

def create_app(config_name):
    app = Flask(__name__)
    app.config.from_object(config[config_name])

    from .talks import talks as talks_blueprint
    app.register_blueprint(talks_blueprint)

    return app
```

# Blueprints

- ▶ Blueprints são containers para rotas, arquivos estáticos e/ou templates
- ▶ Um Blueprint se torna parte de uma aplicação quando ele é registrado

```
_init_.py x
from flask import Blueprint
talks = Blueprint('talks', __name__)
from . import routes
```

```
routes.py x
1  from flask import render_template
2  from . import talks
3
4  @talks.route('/')
5  def index():
6      return render_template('talks/index.html')
7
8  @talks.route('/user/<username>')
9  def user(username):
10     return render_template('talks/user.html', username=username)
11
```

# Templates

- ▶ Templates ajudam a separar a lógica e a apresentação
- ▶ O renderizador de templates padrão do Flask é o Jinja2 (do mesmo desenvolvedor)
- ▶ Partes dinâmicas são especificadas em placeholders
- ▶ Possui várias diretivas de templates, que devem ser verificadas na documentação do Jinja

# Script de inicialização

- ▶ Flask-Script é uma extensão do Flask que adiciona opções de linha de comando
  - (pip install flask-script)
  - python manage.py runserver

---

```
#!/usr/bin/env python
import os
from app import create_app
from flask.ext.script import Manager

app = create_app(os.getenv('FLASK_CONFIG') or 'default')
manager = Manager(app)

if __name__ == '__main__':
    manager.run()
```

# Na Prática – v.0.0.2

- ▶ Extensão para integração com Twitter Bootstrap
  - pip install flask-bootstrap

---

```
from flask import Flask
from config import config
from flask_bootstrap import Bootstrap
bootstrap = Bootstrap

def create_app(config_name):
    app = Flask(__name__)
    app.config.from_object(config[config_name])

    from .talks import talks as talks_blueprint
    app.register_blueprint(talks_blueprint)

    # Inicializando app Bootstrap
    bootstrap(app)

    return app
```

---

```
{% extends "bootstrap/base.html" %}

{% block title %}Talks{% endblock %}

{% block navbar %}
{% endblock %}

{% block content %}


<h1>Hello World!</h1>
</div>
{% endblock %}


```

# Na Prática – v.0.0.3

- ▶ Herança de Templates podem ser usadas para eliminar duplicidade de código HTML

```
{% extends "bootstrap/base.html" %}

{% block title %}Talks{% endblock %}

{% block navbar %}
{% endblock %}

{% block content %}
<div class="container">
    {% block page_content %}{% endblock %}
</div>
{% endblock %}
```

# Herança de Templates

- ▶ Templates de aplicação herdam do template base

```
{% extends "base.html" %}

{% block page_content %}
<h1>Hello World!</h1>
{% endblock %}
```

```
{% extends "base.html" %}

{% block page_content %}
    <div class="page-header">
        <h1>Hello, {{ username }}!</h1>
    </div>
{% endblock %}
```

# Na Prática – v.0.0.4 – Links

- ▶ A função `url_for()` é usada para gerar links de aplicação
- ▶ Para rotas criadas com `app.route`, é utilizado o nome da função
  - `url_for('index')`
- ▶ Para rotas com o blueprint, usa-se `blueprint.nome da função`
  - `url_for('talks.index')`
- ▶ Arquivos estáticos podem ser referenciados por `static` e o nome do arquivo
  - `url_for('static', filename='styles.css')`

# Links

```
{% extends "bootstrap/base.html" %}

{% block title %}Talks{% endblock %}

{% block navbar %}
    <a class="navbar-brand" href="{{ url_for('talks.index') }}>Talks</a>
    <ul class="nav navbar-nav">
        <li><a href="{{ url_for('talks.index') }}>Home</a></li>
        <li><a href="{{ url_for('talks.user', username='Tales') }}>Profile</a></li>
    </ul>
{% endblock %}

{% block content %}
<div class="container">
    {% block page_content %}{% endblock %}
</div>
{% endblock %}
```

# Na Prática – v.0.0.5 – Database

## ▶ Flask SQL-Alchemy

- Integração entre o Flask e o SQLAlchemy, um framework de ORM para Python
  - <http://flask-sqlalchemy.pocoo.org/2.3/>
  - <http://docs.sqlalchemy.org/en/latest/>
- pip install flask-sqlalchemy

# Database

## ▶ config.py

```
basedir = os.path.abspath(os.path.dirname(__file__))

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

class DevelopmentConfig(Config):
    DEBUG = True
    SECRET_KEY = os.environ.get('SECRET_KEY') or 'SeGr&d0'
    SQLALCHEMY_DATABASE_URI = os.environ.get('DEV_DATABASE_URL') or \
        'sqlite:///data-dev.sqlite'

class TestingConfig(Config):
    TESTING = True
    SQLALCHEMY_DATABASE_URI = os.environ.get('TEST_DATABASE_URL') or \
        'sqlite:///data-test.sqlite'

class ProductionConfig(Config):
    SQLALCHEMY_DATABASE_URI = os.environ.get('DATABASE_URL') or \
        'sqlite:///data.sqlite'
    pass
```

# Database

## ▶ app/\_\_init\_\_.py

```
from flask import Flask
from config import config
from flask_bootstrap import Bootstrap
from flask_sqlalchemy import SQLAlchemy

bootstrap = Bootstrap()
db = SQLAlchemy()

def create_app(config_name):
    app = Flask(__name__)
    app.config.from_object(config[config_name])

    from .talks import talks as talks_blueprint
    app.register_blueprint(talks_blueprint)

    # Inicializando app Bootstrap
    bootstrap(app)

    # Inicializando Database
    db.init_app(app)

    return app
```

# Definição de Modelos

- ▶ Modelos são definidos como classes Python

---

```
from datetime import datetime
from . import db

class User(db.Model):
    __tablename__ = 'users'
    id = db.Column(db.Integer, primary_key=True)
    email = db.Column(db.String(64), nullable=False, unique=True, index=True)
    username = db.Column(db.String(64), nullable=False, unique=True, index=True)
    is_admin = db.Column(db.Boolean)
    password_hash = db.Column(db.String(256))
    name = db.Column(db.String(64))
    location = db.Column(db.String(64))
    bio = db.Column(db.Text)
    member_since = db.Column(db.DateTime(), default=datetime.utcnow)
    avatar_hash = db.Column(db.String(256))
```

# Criação da Base de Dados

- ▶ Pode ser feito a partir do Python shell

```
(venv) talesviegas:AulaFlask $ python manage.py shell
```

```
>>> from app import db  
>>> db.create_all()  
>>> db.drop_all()
```

- ▶ Para aplicações maiores, recomenda-se utilizar um framework de migrações como o Flask-Migrate (Alembic)

# Na prática 0.0.6 - Hash Password

- ▶ Como visto na disciplina de Segurança de Sistemas, não armazenamos senhas abertas em base de dados
- ▶ Em Python utilizamos o Werkzeug security, que possibilita funções de hash e validações

# Hash Password

## ▶ Na Model

```
from datetime import datetime
from . import db
from werkzeug.security import generate_password_hash, check_password_hash

@property
def password(self):
    raise AttributeError('password is not a readable attribute')

@password.setter
def password(self, password):
    self.password_hash = generate_password_hash(password)

def verify_password(self, password):
    return check_password_hash(self.password_hash, password)
```

# Na prática 0.0.7 – Autenticação

- ▶ Autenticação pode ter múltiplos Blueprints

```
from flask import Blueprint
auth = Blueprint('auth', __name__)
from . import routes
```

---

```
from flask import render_template
from . import auth

@auth.route('/login')
def login():
    return render_template('auth/login.html')
```

# Autenticação

- ▶ Registrando blueprint com url\_prefix

```
# Inicializando Autenticacao
from .auth import auth as auth_blueprint
app.register_blueprint(auth_blueprint, url_prefix='/auth')
```

# Na prática 0.0.8 – Registro

- ▶ Usuários são registrados via linha de comando, via um comando Flask customizado

```
from app import db
from app.models import User

app = create_app(os.getenv('FLASK_CONFIG') or 'default')
manager = Manager(app)

@manager.command
def adduser(email, username, admin=False):
    """ Registra um novo usuário """
    from getpass import getpass
    password = getpass()
    password2 = getpass(prompt="Confirme: ")
    if password != password2:
        import sys
        sys.exit('Erro: senhas não conferem')
    db.create_all()
    user = User(email=email, username=username, password=password, is_admin=admin)
    db.session.add(user)
    db.session.commit()
    print('Usuário {0} foi registrado com sucesso.'.format(username))
```

# Registro

- ▶ Na linha de comando
  - `python manage.py adduser -help`
  - `python manage.py adduser john@doe.com john`

# Na prática 0.0.9 – Login

- ▶ Web–Forms
- ▶ A extensão Flask WTF provê uma abstração para trabalhar com Web–Forms
  - A classe Form representa o formulário
  - Subclasses de Field representam os campos
  - Validadores podem ser aplicados

```
from flask_wtf import Form
from wtforms import StringField, PasswordField, BooleanField, SubmitField
from wtforms.validators import DataRequired, Length, Email

class LoginForm(Form):
    email = StringField('Email', validators=[DataRequired(), Length(1, 64), Email()])
    password = PasswordField('Password', validators=[DataRequired()])
    remember_me = BooleanField('Keep me logged In')
    submit = SubmitField('Log In')
```

# Login

---

```
from flask import render_template
from .forms import LoginForm
from . import auth

@auth.route('/login', methods=['GET', 'POST'])
def login():
    form = LoginForm()
    if form.validate_on_submit():
        pass
    return render_template('auth/login.html', form=form)
```

# Login

---

```
{% import "bootstrap/wtf.html" as wtf %}

{% block page_content %}

{{ wtf.quick_form(form) }}

{% endblock %}
```

# Na prática 0.0.10 – Logando

- ▶ Flask-login mantém o registro do usuário logado em sessão
  - pip install flask-login

```
login_manager = LoginManager()
login_manager.login_view = 'auth.login'

def create_app(config_name):
    app = Flask(__name__)
    app.config.from_object(config[config_name])

    from .talks import talks as talks_blueprint
    app.register_blueprint(talks_blueprint)

    # Inicializando app Bootstrap
    bootstrap(app)

    # Inicializando Database
    db.init_app(app)

    # Inicializando Autenticacao
    from .auth import auth as auth_blueprint
    app.register_blueprint(auth_blueprint, url_prefix='/auth')
    login_manager.init_app(app)
```

# Logando

- ▶ A classe de usuário necessita herdar de UserMixin, or implementar os seguintes métodos:
  - is\_authenticated()
  - is\_active()
  - is\_anonymous()
  - get\_id()
- ▶ A aplicação precisa registrar um loader callback
- ▶ Request-Handlers podem ser protegidos com o decorator login\_required

# Logando

```
from datetime import datetime
from . import db, login_manager
from werkzeug.security import generate_password_hash, check_password_hash
from flask_login import UserMixin

class User(UserMixin, db.Model):

    @login_manager.user_loader
    def load_user(user_id):
        return User.query.get(int(user_id))
```

# Logando

```
from flask import render_template, redirect, request, url_for, flash
from flask_login import login_user
from .forms import LoginForm
from . import auth
from ..models import User

@auth.route('/login', methods=['GET', 'POST'])
def login():
    form = LoginForm()
    if form.validate_on_submit():
        user = User.query.filter_by(email=form.email.data).first()
        if user is None or not user.verify_password(form.password.data):
            flash('Invalid email or password')
            return redirect(url_for('.login'))
        login_user(user, form.remember_me.data)
        return redirect(request.args.get('next') or url_for('talks.index'))
    return render_template('auth/login.html', form=form)
```

# Mensagens Flash

```
{% for message in get_flashed_messages() %}  
  <div class="alert alert-warning">  
    <button type="button" class="close" data-dismiss="alert">x</button>  
    {{ message }}  
  </div>  
{% endfor %}
```

# Acessando o usuário logado

- ▶ O usuário corrente pode ser acessado através do contexto global `current_user`
- ▶ Podem ser criados itens de navegação específicos para o usuário

# Na prática 0.0.11 – Logout

- ▶ O decorator `login_required` previne acesso a um usuário não-logado a uma rota

```
@auth.route('/logout')
@login_required
def logout():
    logout_user()
    flash('You have been logged out.')
    return redirect(url_for('talks.index'))
```

- ▶ Mostrar link de login ou logout

```
{% if current_user.is_authenticated %}
<li><a href="{{ url_for('talks.user', username=current_user.username) }}>Profile</a></li>
<li><a href="{{ url_for('auth.logout') }}>Logout</a></li>
{% else %}
<li><a href="{{ url_for('auth.login') }}>Login</a></li>
{% endif %}
```