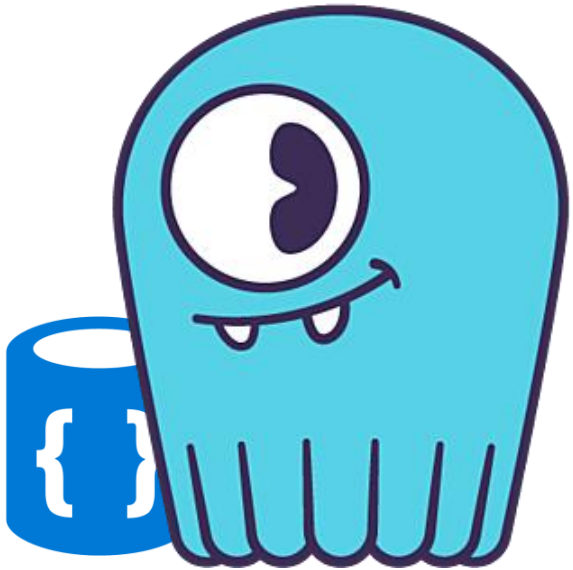
Primeiros Passos com o *Framework Flask*

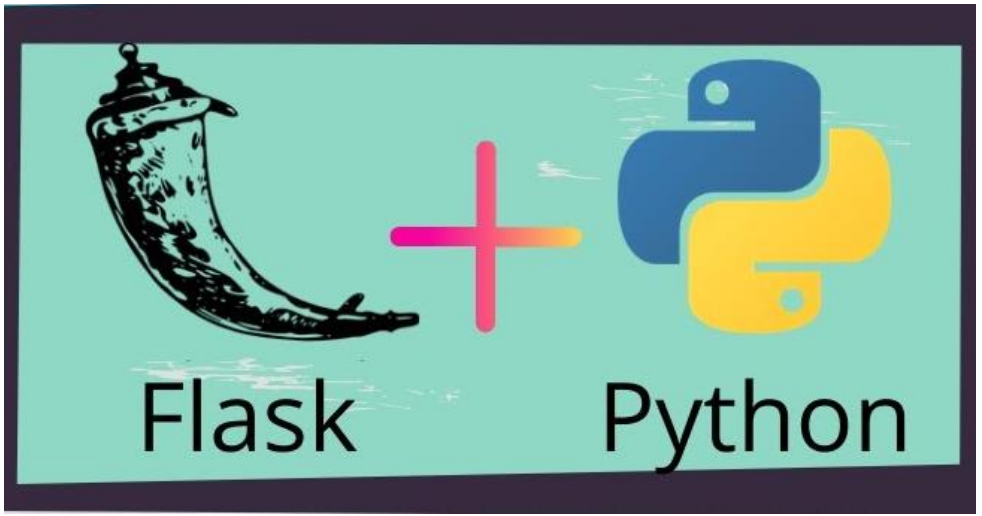
Prof. Dr. Diego Bruno

Education Tech Lead na DIO
Doutor em Robótica e *Machine Learning* pelo ICMC-USP



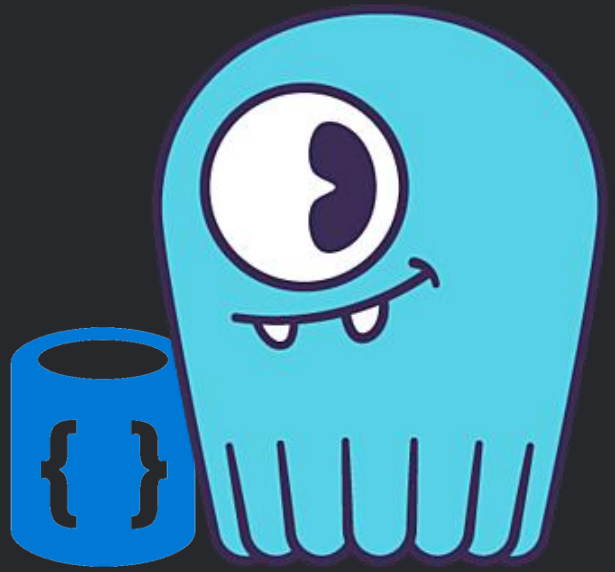
Roteiro para nossa aula de hoje

- → Introdução
- → Objetivos
- → Base teórica
- → Motivação
- → Conclusões



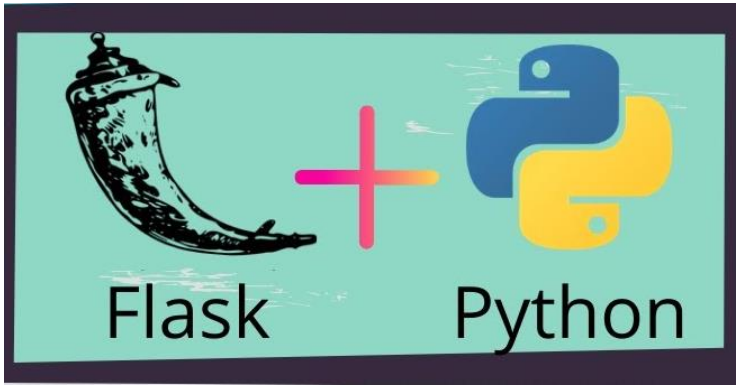
Flask

Prof. Dr. Diego Bruno



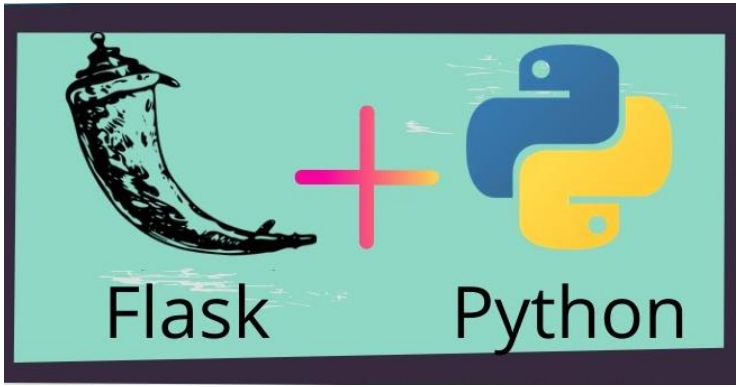
Mas o que é Flask?

Flask é um pequeno framework web escrito em Python. É classificado como um microframework porque não requer ferramentas ou bibliotecas particulares, mantendo um núcleo simples, porém, extensível.



Mas o que é Flask?

O ***Flask Python*** é basicamente um framework do tipo “faça você mesmo”. Isso significa que não tem nenhuma interação interna com banco de dados, mas o pacote flask-sqlalchemy irá conectar o banco de dados SQL a um aplicativo Flask. O pacote flask-sqlalchemy precisa somente de uma coisa para se conectar o banco de dados SQL: o banco de dados URL.



Mas o que é Flask?

O Flask precisa que o banco de dados URL seja parte da sua configuração central através do atributo **SQLALCHEMY_DATABASE_URI**. Uma solução rápida e suja pra isso é fazer um hardcode do banco de dados dentro do aplicativo.

```
1  # top of app.py
2  from flask import Flask
3  from flask_sqlalchemy import SQLAlchemy
4
5  app = Flask(__name__)
6  app.config['SQLALCHEMY_DATABASE_URI'] = 'postgres://localhost:5432/flask_todo'
7  db = SQLAlchemy(app)
```

Simplicidade

Você pode tornar as coisas mais simples utilizando variáveis de ambiente. Elas irão garantir que independente da máquina que você rode o código, ele irá sempre apontar na coisa certa, se essa coisa estiver configurada no ambiente que está sendo rodado. Ele também garante que, mesmo que você precise da informação para rodar o aplicativo, nunca irá aparecer como um valor ***hardcoded*** no ***source control***.

Simplicidade

No mesmo lugar que você declarou o FLASK_APP, declare um DATABASE_URL apontando para o lugar do seu banco de dados Postgres. O desenvolvimento tende a funcionar localmente, então aponte para o seu banco de dados local.

```
# Também no seu script active

export DATABASE_URL='postgres://localhost:5432/flask_todo
```

Agora em **app.py** inclua o banco de dados URL no seu aplicativo web.

Python

```
app.config['SQLALCHEMY_DATABASE_URI'] =
os.environ.get('DATABASE_URL', '')
db = SQLAlchemy(app)
```

Simplicidade

No mesmo lugar que você declarou o FLASK_APP, declare um DATABASE_URL apontando para o lugar do seu banco de dados Postgres. O desenvolvimento tende a funcionar localmente, então aponte para o seu banco de dados local.

```
# Também no seu script active

export DATABASE_URL='postgres://localhost:5432/flask_todo
```

Agora em **app.py** inclua o banco de dados URL no seu aplicativo web.

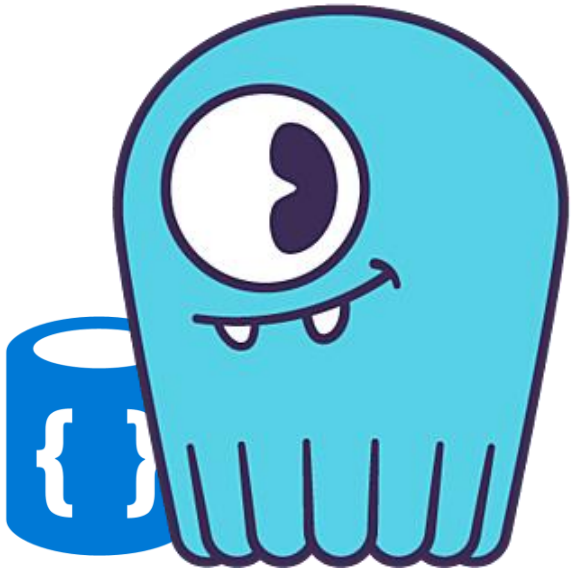
Python

```
app.config['SQLALCHEMY_DATABASE_URI'] =
os.environ.get('DATABASE_URL', '')
db = SQLAlchemy(app)
```

Simplicidade

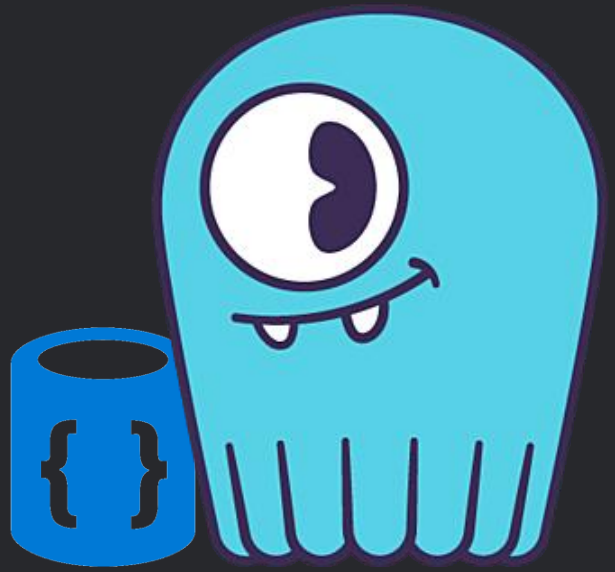
E simplesmente dessa forma, o seu aplicativo agora tem uma conexão com um banco de dados.

```
app.config['SQLALCHEMY_DATABASE_URI'] =
os.environ.get('DATABASE_URL', '')
db = SQLAlchemy(app)
```



Obrigado!

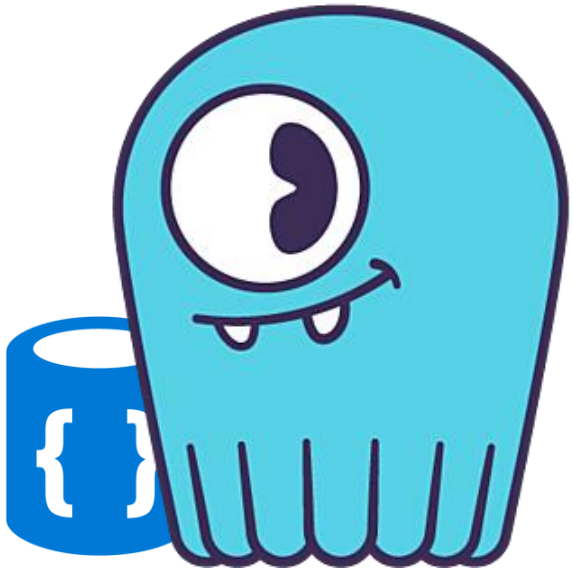
Prof. Dr. Diego Bruno



Aplicações
Framework Flask

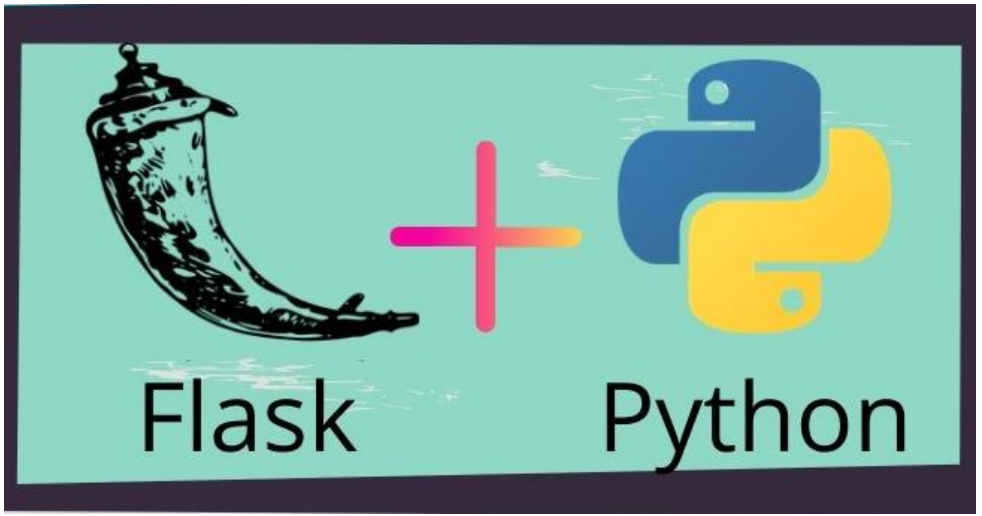
Prof. Dr. Diego Bruno

Education Tech Lead na DIO
Doutor em Robótica e *Machine Learning* pelo ICMC-USP



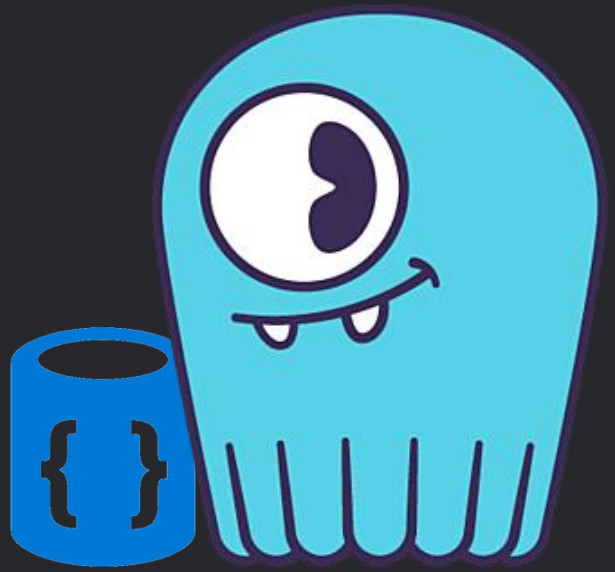
Roteiro para nossa aula de hoje

- → Introdução
- → Objetivos
- → Base teórica
- → Motivação
- → Conclusões



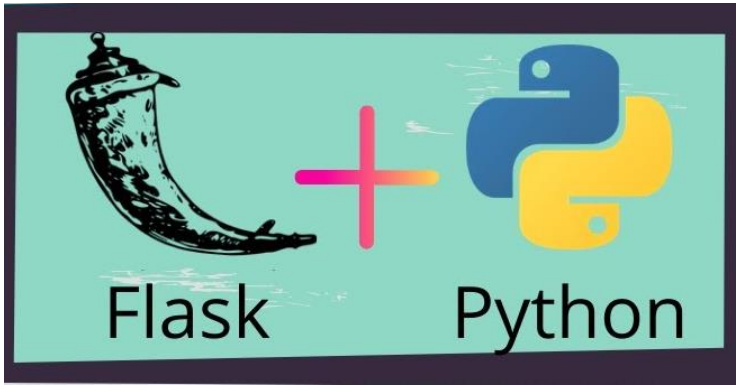
Flask

Prof. Dr. Diego Bruno



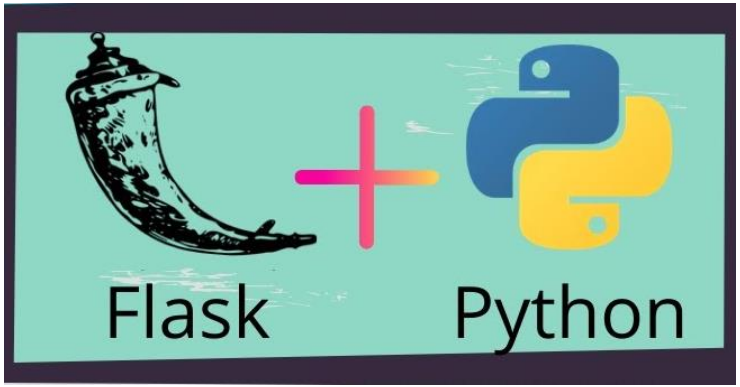
Mas o que é Flask?

Flask é um pequeno framework web escrito em Python. É classificado como um microframework porque não requer ferramentas ou bibliotecas particulares, mantendo um núcleo simples, porém, extensível.



Mas o que é Flask?

O ***Flask Python*** é basicamente um framework do tipo “faça você mesmo”. Isso significa que não tem nenhuma interação interna com banco de dados, mas o pacote flask-sqlalchemy irá conectar o banco de dados SQL a um aplicativo Flask. O pacote flask-sqlalchemy precisa somente de uma coisa para se conectar o banco de dados SQL: o banco de dados URL.



Mas o que é Flask?

O Flask precisa que o banco de dados URL seja parte da sua configuração central através do atributo **SQLALCHEMY_DATABASE_URI**. Uma solução rápida e suja pra isso é fazer um hardcode do banco de dados dentro do aplicativo.

```
1  # top of app.py
2  from flask import Flask
3  from flask_sqlalchemy import SQLAlchemy
4
5  app = Flask(__name__)
6  app.config['SQLALCHEMY_DATABASE_URI'] = 'postgres://localhost:5432/flask_todo'
7  db = SQLAlchemy(app)
```

Simplicidade

No mesmo lugar que você declarou o FLASK_APP, declare um DATABASE_URL apontando para o lugar do seu banco de dados Postgres. O desenvolvimento tende a funcionar localmente, então aponte para o seu banco de dados local.

```
# Também no seu script active

export DATABASE_URL='postgres://localhost:5432/flask_todo
```

Agora em **app.py** inclua o banco de dados URL no seu aplicativo web.

Python

```
app.config['SQLALCHEMY_DATABASE_URI'] =
os.environ.get('DATABASE_URL', '')
db = SQLAlchemy(app)
```

Simplicidade

No mesmo lugar que você declarou o FLASK_APP, declare um DATABASE_URL apontando para o lugar do seu banco de dados Postgres. O desenvolvimento tende a funcionar localmente, então aponte para o seu banco de dados local.

```
# Também no seu script active

export DATABASE_URL='postgres://localhost:5432/flask_todo
```

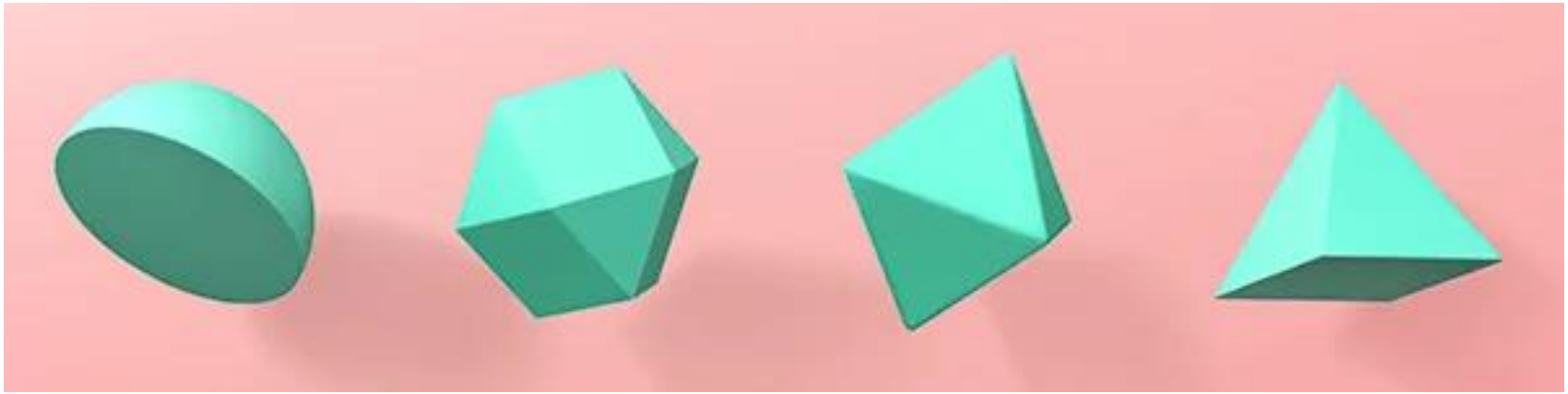
Agora em **app.py** inclua o banco de dados URL no seu aplicativo web.

Python

```
app.config['SQLALCHEMY_DATABASE_URI'] =
os.environ.get('DATABASE_URL', '')
db = SQLAlchemy(app)
```

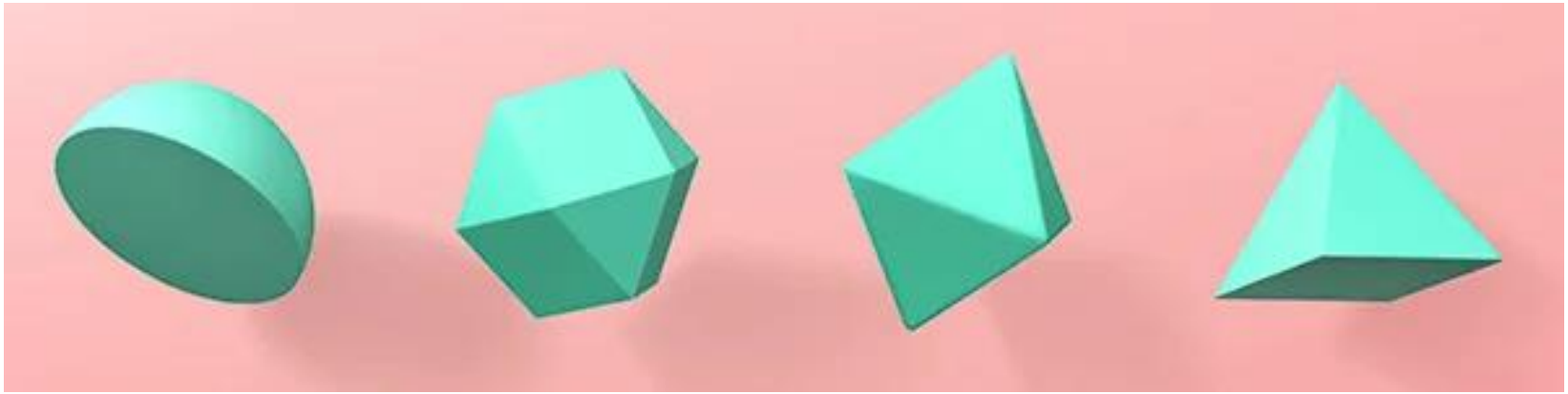
Definindo Objetos em Flask

Ter um banco de dados para estar conectado é um ótimo primeiro passo. Agora é hora de definir alguns objetos para preencher o banco de dados..



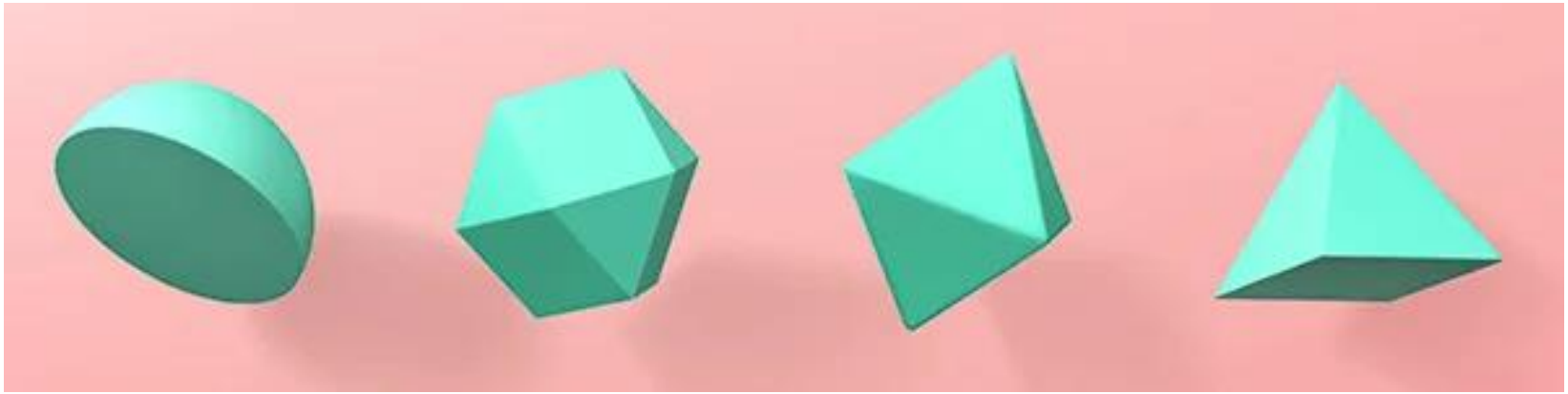
Definindo Objetos em Flask

No desenvolvimento de aplicativos, um “model” refere-se a representação real ou conceitual de algum objeto. Por exemplo, caso esteja fazendo um app para uma concessionária de carros, você talvez defina um model chamado **car** que encapsule todos os atributos e comportamentos de um carro.



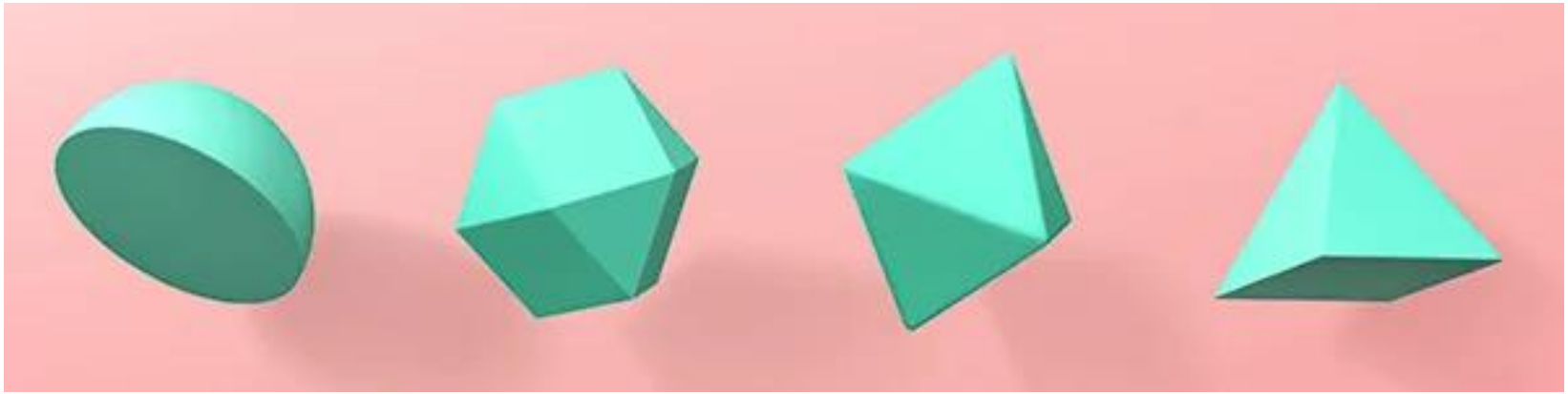
Definindo Objetos em Flask

Nesse caso, você estará fazendo um To-do List com Tasks, e cada Task pertence a um usuário. Antes de você pensar mais a fundo sobre como eles estão relacionados, comece definindo os objetos para Tasks e Users.



Definindo Objetos em Flask

O pacote **flask-sqlalchemy** aproveita o SQLalchemy para configurar e informar a estrutura do banco de dados. Você irá definir um modelo que irá viver no banco de dados herdando do objeto **db.Model** e irá definir o atributo desses models como instâncias **db.Column**. Para cada coluna, você deve especificar um tipo de dado, e então você irá passar esse data type para o comando **db.Column** como primeiro argumento..



Fonte: https://harve.com.br/blog/programacao-python-blog/introducao-e-tutorial-ao-flask-python/

Definindo Objetos em Flask

Pelo fato da definição do model ocupar um espaço conceitual diferente do que da configuração do aplicativo, faça com que o **models.py** mantenha definições de model de forma separada do **app.py**. O modelo do Task deve ser construído para que tenha os seguintes atributos:

- **id:** Um valor que é um identificador único para puxar do banco de dados.
- **name:** O nome ou o título da task que o usuário irá ver quando task for listada.
- **note:** Quaisquer comentários adicionais que uma pessoa queira deixar com sua task.
- **creation_date:** A data e o horário que a task foi criada.
- **due_date:** A data e o horário que a task deve ser concluída (se houver).
- **completed:** Uma forma de indicar se a task foi concluída ou não.

Fonte: https://harve.com.br/blog/programacao-python-blog/introducao-e-tutorial-ao-flask-python/

Definindo Objetos em Flask

Dado essa lista de atributos para objetos Task, o objeto **Task** do aplicativo pode ser definida dessa forma:

```python
from .app import db
from datetime import datetime

class Task(db.Model):
    """Tasks for the To Do list."""
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.Unicode, nullable=False)
    note = db.Column(db.Unicode)
    creation_date = db.Column(db.DateTime, nullable=False)
    due_date = db.Column(db.DateTime)
    completed = db.Column(db.Boolean, default=False)

    def __init__(self, *args, **kwargs):
        """On construction, set date of creation."""
        super().__init__(*args, **kwargs)
        self.creation_date = datetime.now()
```

Fonte: https://harve.com.br/blog/programacao-python-blog/introducao-e-tutorial-ao-flask-python/

Relacionamento dos models

Em certos aplicativos web, você talvez queira expressar relacionamentos entre objetos. No exemplo do To-do list, usuários são donos de várias tasks, e cada tarefa pertence a somente a um usuário.

Esse é um exemplo de um relacionamento “many-to-one”, também conhecido como foreign key relationship, onde as tasks são os “many” (muitos) e o usuário dono delas é o “one” (um).

Relacionamento dos models

No Flask Python, um relacionamento many-to-one pode ser especificado utilizando a função **db.relationship**. Primeiro, construa o objeto User.

```
class User(db.Model):
    """The User object that owns tasks."""
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.Unicode, nullable=False)
    email = db.Column(db.Unicode, nullable=False)
    password = db.Column(db.Unicode, nullable=False)
    date_joined = db.Column(db.DateTime, nullable=False)
    token = db.Column(db.Unicode, nullable=False)

    def __init__(self, *args, **kwargs):
        """On construction, set date of creation."""
        super().__init__(*args, **kwargs)
        self.date_joined = datetime.now()
        self.token = secrets.token_urlsafe(64)
```

Inicializando o banco de dados

Agora que os modelos e os relacionamentos de modelos estão configurados, comece configurando o seu banco de dados. O Flask Python não vem com a sua própria utilidade de gerenciamento de banco de dados, então você terá que escrever o seu próprio (até um certo ponto).

Inicializando o banco de dados

Crie um script chamado initializedb.py próximo do setup.py para gerenciar o banco de dados. (Claro, não precisa ser chamado assim, mas porque não dar nomes que são apropriados a função do arquivo?) Dentro de initializedb.py, importe o objeto db de app.py e use-o para criar e eliminar tabelas. initializedb.py deve parecer dessa forma:

```
1  from todo.app import db
2  import os
3
4  if bool(os.environ.get('DEBUG', '')):
5      db.drop_all()
6  db.create_all()
```

Views e Configuração da URL

As últimas partes necessárias para conectar o aplicativo inteiro são os views e routes. Em desenvolvimento web, um “view” (conceito) é uma funcionalidade que roda quando um ponto de acesso específico (“route”) é atingido.

Em Flask, views aparecem como funções; por exemplo, consegue ver a view “hello world” ali em cima. Pra ser prático, vamos mostrar aqui novamente:

Quando o route do **http://domainname/** é acessado, o cliente recebe a resposta, “Hello, world!”.

```
1  @app.route('/')
2  def hello_world():
3      """Print 'Hello, world!' as the response body."""
4      return 'Hello, world!'
```

Views e Configuração da URL

Comece por um view que lida somente com solicitações **get**, e responda com o JSON representando todas os routes que serão acessíveis e os métodos que podem ser utilizados para acessar elas:

```python
from flask import jsonify

@app.route('/api/v1', methods=["GET"])
def info_view():
    """List of routes for this API."""
    output = {
        'info': 'GET /api/v1',
        'register': 'POST /api/v1/accounts',
        'single profile detail': 'GET /api/v1/accounts/<username>',
        'edit profile': 'PUT /api/v1/accounts/<username>',
        'delete profile': 'DELETE /api/v1/accounts/<username>',
        'login': 'POST /api/v1/accounts/login',
        'logout': 'GET /api/v1/accounts/logout',
        "user's tasks": 'GET /api/v1/accounts/<username>/tasks',
        "create task": 'POST /api/v1/accounts/<username>/tasks',
        "task detail": 'GET /api/v1/accounts/<username>/tasks/<id>',
        "task update": 'PUT /api/v1/accounts/<username>/tasks/<id>',
        "delete task": 'DELETE /api/v1/accounts/<username>/tasks/<id>'
    }
    return jsonify(output)
```

Views e Configuração da URL

A função view encima pega o que é efetivamente uma lista de todo o route que esse API pretende lidar e envia ao cliente todas vez que o route http://domainname/api/v1 for acessada. Tenha em mente que, por si mesmo, o Flask oferece suporte ao roteamento para URLs exatamente iguais, então acessar essa rota com um trailing / iria criar um erro 404. Se você quisesse lidar com a mesma função view, você precisaria de um stack de decoradores dessa forma:

```
1  @app.route('/api/v1', methods=["GET"])
2  @app.route('/api/v1/', methods=["GET"])
3  def info_view():
4      # blah blah blah more code
```

Obrigado!

Prof. Dr. Diego Bruno