PROGRAMAÇÃO
COMERCIAL

PROFESSOR!

THIAGO MAGALHÃES

A criatividade está na simplicidade. Quanto mais simples, mais impactante.

INTERNET - UM GRANDE SISTEMA DISTRIBUÍDO

CONCEITUAÇÃO

Um sistema distribuído é um conjunto de computadores independentes entre si que se apresenta a seus usuários como um sistema único e coerente – Van Steen Tanenbaum.

Coleção de computadores autônomos interconectados por uma rede, com software projetado para produzir uma aplicação integrada.

Você sabe que existe um sistema distribuído quando a falha de um computador que você nunca ouviu falar impede que você faça qualquer trabalho.

DESENVOLVIMENTO DE APLICAÇÕES WEB

Em computação, aplicação web designa, de forma geral, sistemas de informática projetados para utilização em um navegador (browser), desenvolvidos utilizando-se tecnologias web (HTML, JavaScript e CSS). Pode ser executado a partir de um servidor HTTP (Web Host) ou localmente, no dispositivo do usuário.

Ter uma visão do todo é muito importante para as partes.

FUNÇÃO DO SERVIDOR WEB

Receber uma solicitação (requisição) e devolver conteúdo (resposta) para o cliente. Geralmente os servidores enviam instruções para o browser escritas em HTML. O HTML diz ao browser como apresentar conteúdo ao usuário web.

ENVIANDO UMA REQUEST HTTP

telnet www.uft.edu.br 80

GET / HTTP/1.1

Host: www.uft.edu.br

WORLD WIDE WEB

Let’s start...

WEB OU WORLD WIDE WEB

Sistema de servidores de internet que suportam documentos especialmente formatados.

Os documentos são formatados em uma linguagem de marcação chamada HTML (HyperText Markup Language) que suporta links para outros documentos, bem como gráficos, áudio e arquivos de vídeo.

HTML (HyperText Markup Language)

- Linguagem de marcação padrão utilizada para criar páginas web.
- Não é uma linguagem de programação.
- Utilizada para fins estruturais.
- HTML apenas encapsula dados e descreve o que fazer com eles, não como fazer.

Elementos HTML formam os blocos de construção de páginas.

INICIALIZANDO OS MOTORES

Estruturando nosso ambiente de desenvolvimento

SISTEMA DE CONTROLE DE VERSÃO **VCS**

Definição

Software com a finalidade de gerenciar diferentes versões de um documento qualquer. Esses sistemas são comumente utilizados no desenvolvimento de software para controlar as diferentes versões (histórico e desenvolvimento) dos códigos-fontes e também da documentação.

INICIALIZANDO O PRIMEIRO **PROJETO**

- Criar uma conta no https://github.com/
- Criar um repositório: programacao_comercial
- Clonar o repositório:
  - *$ git clone URL*
- Criar nossos primeiros arquivos HTML, CSS, JS
  - aula-1.html
  - Js/funcoes.js
  - css/styles.css

ENVIANDO ALTERAÇÕES PARA O **VERSIONADOR**

Enviar todo o conteúdo para o repositório de código

I. *$ git status*
II. *$ git add ./*
III. *$ git commit -m 'Primeira aula - Introdução ao desenvolvimento web'*
IV. *$ git pull origin master*
V. *$ git push origin master.*

FRAMEWORKS PARA DESENVOLVIMENTO WEB

CONTEITO

Um framework é um conjunto de conceitos usado para resolver um problema de um domínio específico. Framework conceitual não se trata de um software executável, mas sim de um modelo de dados para um domínio.

Framework de software compreende de um conjunto de classes implementadas em uma linguagem de programação específica, usadas para auxiliar o desenvolvimento de software. O framework atua onde há funcionalidades em comum a várias aplicações, porém para isso as aplicações devem ter algo razoavelmente grande em comum para que o mesmo possa ser utilizado em várias aplicações.

FRAMEWORKS PARA DESENVOLVIMENTO WEB

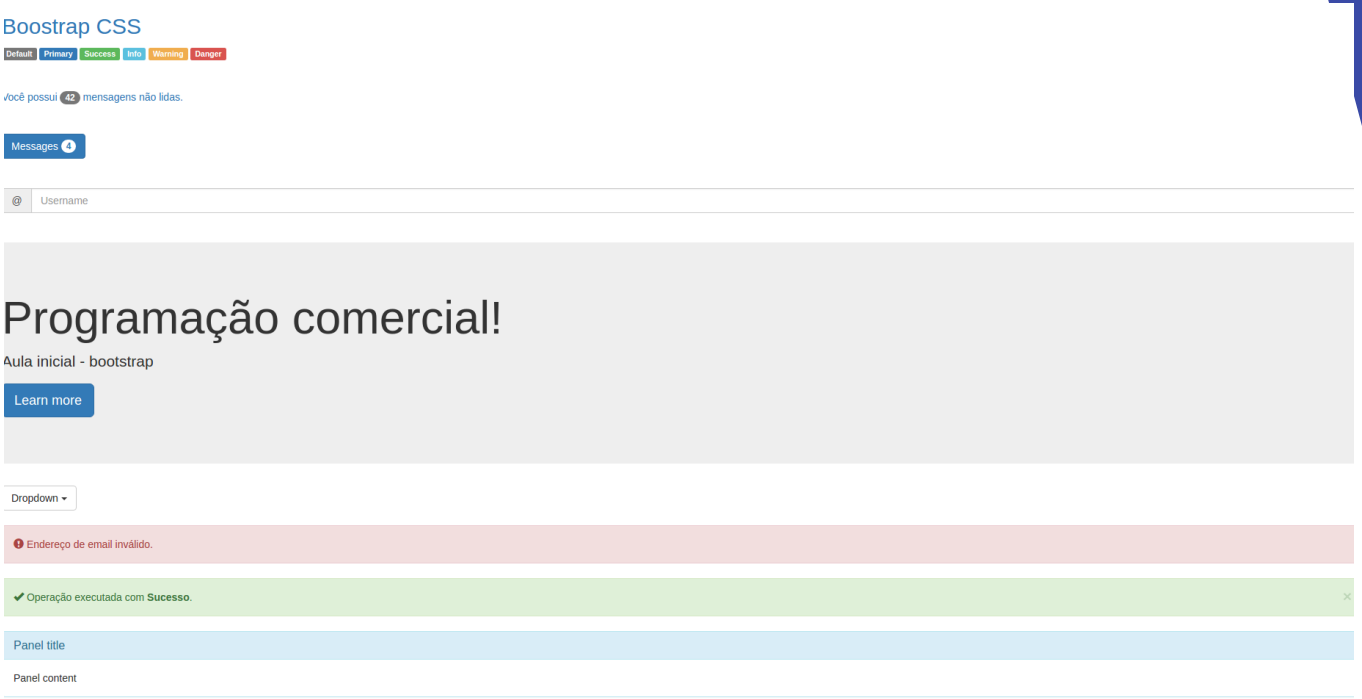
Bootstrap
O mais popular framework HTML, CSS, e JS para desenvolvimento de projetos responsivos.

JQuery
Biblioteca JavaScript *cross-browser* desenvolvida para simplificar os scripts *client-side* que interagem com o HTML. Usada por cerca de 77% dos 10 mil sites mais visitados do mundo, jQuery é a mais popular das bibliotecas JavaScript.

EVOLUINDO O PRIMEIRO **PROJETO**

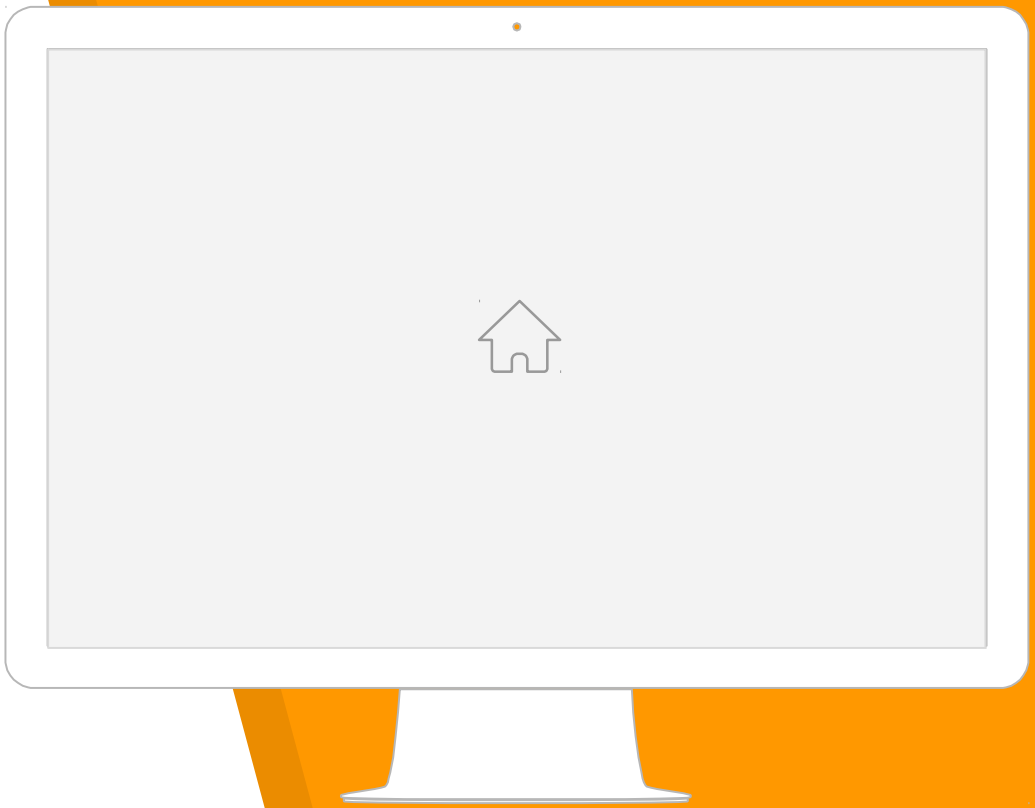
- Criar nosso segundo arquivo HTML
  - aula-2.html
- Fazer download do framework Bootstrap
- Fazer download da biblioteca Javascript Jquery
- Iterar com a developer tools do browser

RESULTADOS DO PRIMEIRO PROJETO



DESENVOLVIMENTO WEB

PADRÕES ARQUITETURAIS



EVOLUÇÃO DO DESENVOLVIMENTO WEB

A curto prazo a internet e a web revolucionaram o acesso à informação superando todos os outros desenvolvimentos tecnológicos da história, e cresceram rapidamente com relação à extensão de uso, afetando significativamente todos os aspectos da vida do ser humano.Indústrias, tais como de fabricação, viagens e hotelaria, educação e governo estão fazendo uso desses canais de informações afim de melhorar e aperfeiçoar suas tarefas diárias.

O comércio eletrônico tem se expandido rapidamente, atravessando fronteiras internacionais. Até mesmo os sistemas de informação legados e banco de dados estão migrando para a web.

PADRÃO ARQUITETURAL **MVC**

Model-View-Controller é um padrão de arquitetura de software que separa a representação da informação da interação do usuário com ele.

Com o aumento da complexidade das aplicações desenvolvidas, sempre visando a programação orientada a objeto, torna-se relevante a separação entre os dados e a apresentação das aplicações. Desta forma, alterações feitas no layout não afetam a manipulação de dados, e estes poderão ser reorganizados sem alterar o layout.

Esse padrão resolve este problema através da separação das tarefas de acesso aos dados e lógica de negócio, lógica de apresentação e de interação com o utilizador, introduzindo um componente entre os dois: o controlador.

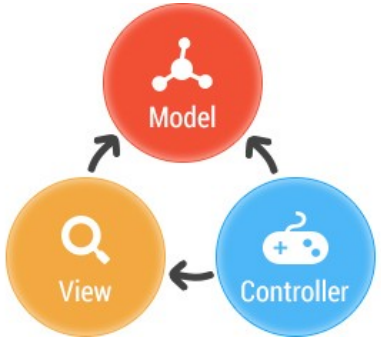
PADRÃO ARQUITETURAL **MVC**

MODELO (MODEL)

Consiste nos dados da aplicação, regras de negócios, lógica e funções.

VISÃO (VIEW)

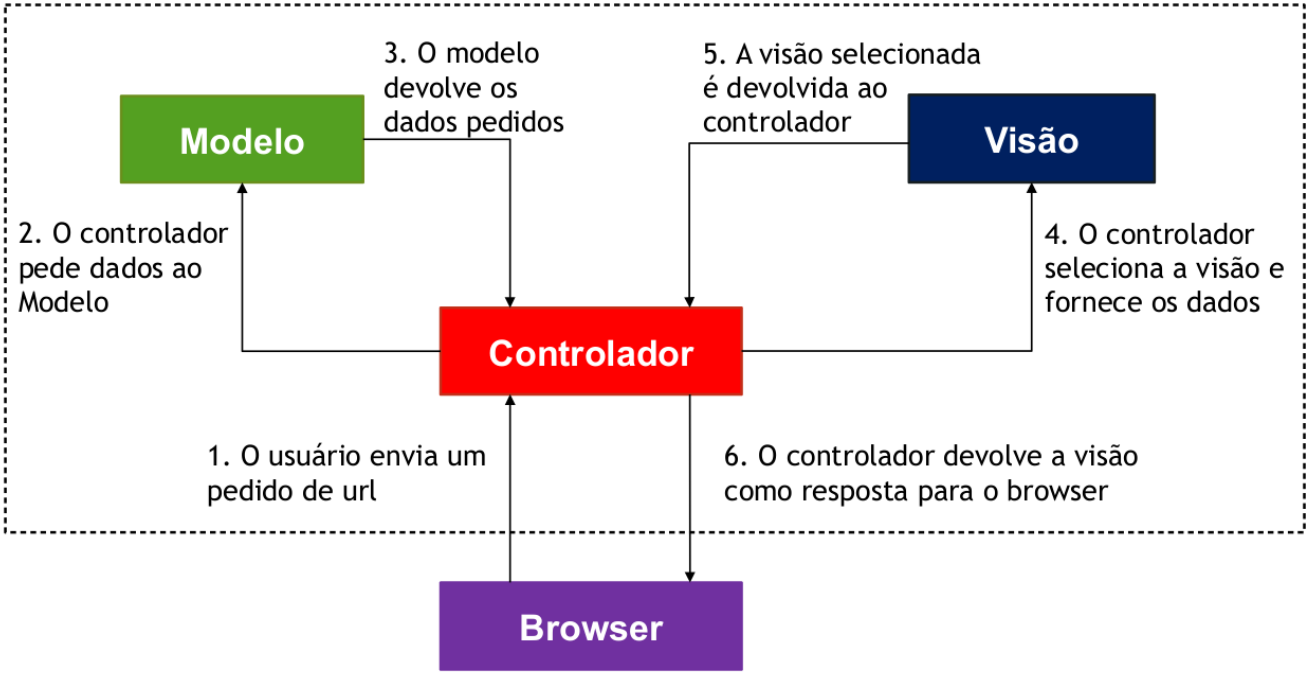
Pode ser qualquer saída de representação dos dados, como uma tabela ou um diagrama. É possível ter várias visões do mesmo dado, como um gráfico de barras para gerenciamento e uma visão tabular para contadores.



CONTROLADOR (CONTROLLER)

Faz a mediação da entrada, convertendo-a em comandos para o modelo ou visão. As ideias centrais por trás do MVC são a reusabilidade de código e separação de conceitos.

PADRÃO ARQUITETURAL **MVC**



ORM - MAPEAMENTO OBJETO RELACIONAL

CONCEITUAÇÃO ...

Técnica de desenvolvimento utilizada para reduzir a demanda de trabalho em programação orientada a objetos ao se utilizar bancos de dados relacionais.

As tabelas do banco de dados são representadas através de classes e os registros de cada tabela são representados como instâncias das classes correspondentes.

Com esta técnica, o programador não precisa se preocupar com os comandos em linguagem SQL; ele irá usar uma interface de programação simples que faz todo o trabalho de persistência.

Não é necessária uma correspondência direta entre as tabelas de dados e as classes do programa.

ORM - MAPEAMENTO OBJETO RELACIONAL

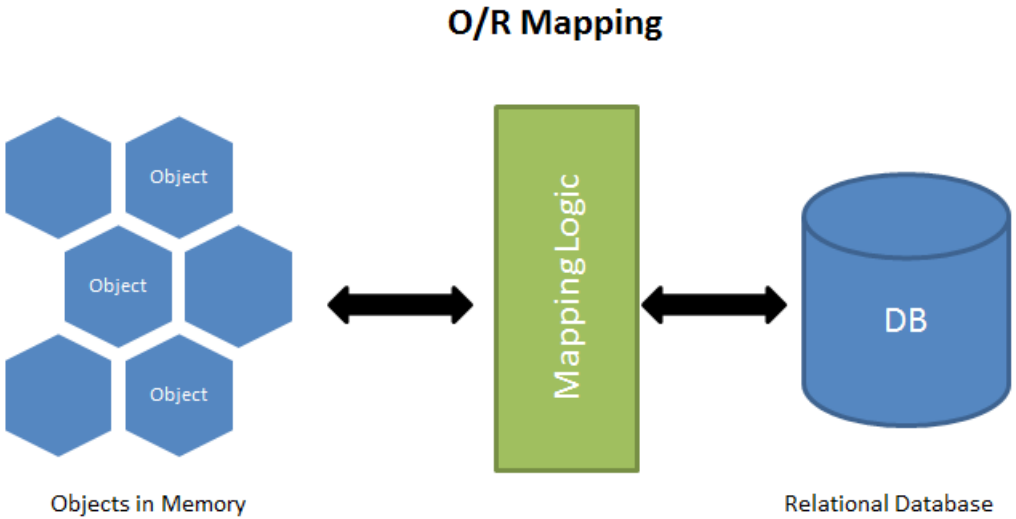
... CONCEITUAÇÃO

A relação entre as tabelas onde originam os dados e o objecto que os disponibiliza é configurada pelo programador, isolando o código do programa das alterações à organização dos dados nas tabelas do banco de dados.

A forma como este mapeamento é configurado depende da ferramenta que estamos a usar. Como exemplo, o programador que use Hibernate na linguagem Java pode usar arquivos XML ou o sistema de anotações que a linguagem providencia. Em outros casos o mapeamento é feito diretamente no código, através de herança de classes especiais como é o caso do ORM do Django e do SqlAlchemy na linguagem Python.

Algumas ferramentas gráficas podem ser usadas para gerar o código que representa o modelo do banco, como o ORM Pony também da linguagem Python.

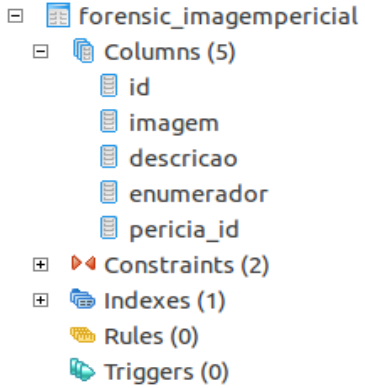
ORM - MAPEAMENTO OBJETO RELACIONAL



ORM - MAPEAMENTO OBJETO RELACIONAL

```python
class ImagemPericial(models.Model):
    """
    Classe para modelagem de imagens periciais
    """
    pericia = models.ForeignKey(Pericia, related_name='imagens_periciais')
    imagem = models.ImageField(upload_to=forensic_image_path)
    descricao = models.CharField(max_length=TAMANHO_TEXTO_GRANDE, default='', verbose_name='Descrição')
    enumerador = models.IntegerField(verbose_name='Enumerador da imagem')

    class Meta:
        ordering = ['enumerador']
```



Model

View

Template

PYTHON DJANGO FRAMEWORK

LINGUAGEM PYTHON

Python fits your mind (Python se adapta à sua mente)

CARACTERÍSTICAS

- Alto nível
- Interpretada
- Dinamicamente Tipada
- Multi-paradigma

JÁ VEM COM BATERIAS

Python Standard Library

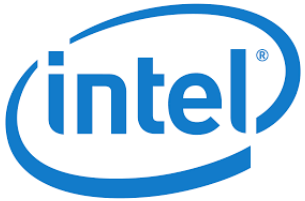
http://docs.python.org/library/

FILOSOFIA

>>> import this

The Zen of Python, by Tim Peters
Bonito é melhor que feio.
Explícito é melhor que implícito.
Simples é melhor que complexo.
Complexo é melhor que complicado.
Plano é melhor que aninhado.
Esparso é melhor que denso.
Legibilidade conta.
Casos especiais não são especiais o bastante para se quebrar as regras.
Embora a simplicidade supere o purismo.
Erros nunca deveriam passar silenciosamente.
A menos que explicitamente silenciados.
Ao encarar a ambiguidade, recuse a tentação de adivinhar.
Deveria haver uma – e preferencialmente apenas uma – maneira óbvia de se fazer isto.
Embora aquela maneira possa não ser óbvia à primeira vista se você não for holandês.
Agora é melhor que nunca.
Embora nunca, seja muitas vezes melhor que pra já.
Se a implementação é difícil de explicar, é uma má idéia.
Sea implementação é fácil de explicar, pode ser uma boa idéia.
Namespaces são uma grande ideia – vamos fazer mais deles!

QUEM USA PYTHON



https://us.pycon.org/2016/sponsors/

CONFIGURANDO O **AMBIENTE DE DESENVOLVIMENTO**

VIRTUALENV - FERRAMENTA PARA CRIAÇÃO DE AMBIENTES PYTHON ISOLADOS

- Verificando a instalação do virtualenv
  - *$ virtualenv --version*
- Criar ambiente virtual com o virtualenv
  - *$ virtualenv virtual*
- Excluindo arquivos de configuração do virtualenv do versionador de código
  - Criar o arquivo .gitignore e dicionar o seguinte conteúdo:

```
# Exclui todo o conteudo do diretorio virtual, exceto o sub-diretorio sistema
virtual/*
!virtual/sistema
# Exclui todos os arquivos com as seguintes extensoes
*.pyc
```

CONFIGURANDO O **AMBIENTE DE DESENVOLVIMENTO**

VIRTUALENV - FERRAMENTA PARA CRIAÇÃO DE AMBIENTES PYTHON ISOLADOS

- Ativando o ambiente virtual
  - *$ source bin/activate*

- Desativando o ambiente virtual
  - *$ deactivate*

IPYTHON – SHELL PYTHON PODEROSO

- Instalando o ipython (pip – instalador de pacotes python – disponível com o virtualen)
  - *$ pip install ipython*

CONHECENDO PYTHON

TIPAGEM DINÂMICA

```
variavel = 1
type(variavel)
id(variavel)
variavel = 'Thiago'
type(variavel)
id(variavel)
```

LAÇOS DE REPETIÇÃO

```
for item in range(1, 11):
    print item
```

ARRAYS

```
vetor = [1, 2, 3, 4, 5]
for item in vetor:
    print item

vetor = [1, 2, 3, 4, 'Thiago']
for item in vetor:
    print item, type(item)

for indice, valor in enumerate(vetor):
    print indice, valor

vetor.reverse()
print vetor
len(vetor)
vetor.append(10)
len(vetor)
vetor.pop()
len(vetor)
```

CONHECENDO PYTHON

DICIONÁRIOS (CHAVE-VALOR)

```
dicionario = {
    'nome': 'Thiago',
    'idade': 30
}

print dicionario
dicionario.keys()
dicionario.values()
dicionario.get('nome')
```

MÉTODOS E FUNÇÕES

```
def soma(a, b):
    return a + b
c = soma(5, 10)
print c
```

CLASSES

```
class MinhaClasse:
    """
    Documentacao da classe
    """
    atributo_publico = None
    __atributo_privado = 5

    def __init__(self):
        self.atributo_publico = 10

    def metodo(self):
        return self.atributo_publico * 2

    def get_atributo_privado(self):
        return self.__atributo_privado
```

DJANGO FRAMEWORK

Django é um web framework Python de alto nível, livre e open-source que encoraja o desenvolvimento rápido e limpo e o design programático. Construído por desenvolvedores experientes, toma conta de muitos dos aborrecimentos do desenvolvimento web, assim, o desenvolvedor pode manter o foco na implementação do aplicativo.

O Django utiliza o princípio DRY (Don’t Repeat Yourself), que permite ao desenvolvedor aproveitar ao máximo o código já feito, evitando a repetição.

Utilizado para o desenvolvimento do *backend* do aplicativo *mobile* do Instagram.

DJANGO FRAMEWORK

ARQUITETURA MVT

Django utiliza uma variante do padrão arquitetural MVC, chamado MVT

- (M)odel: Camada de acesso a dados. Contém toda e qualquer definição acerca dos dados: como acessá-los, como validá-los, seus comportamentos e quais relacionamentos existem entre os mesmos.
- (V)iew: A camada de lógica de negócios. Camada que associa modelos a template(s).
- (T)emplate: Camada de apresentação. Essa camada contem as decisões relacionadas à apresentação: como algo deve ser apresentado na página web ou outro tipo de documento.

DJANGO FRAMEWORK

PRINCIPAIS CARACTERÍSTICAS ...

- Mapeamento Objeto-Relacional (ORM)
  - O ORM do Django permite a modelagem de dados através de classes Python. Isso permite a criação e manipulação de tabelas no banco de dados sem a necessidade de utilizar SQL.
- Interface Administrativa
  - Com o Django, é possível a geração automática de uma interface para administração dos modelos criados pelo ORM.
- Formulários:
  - E possível gerar formulários automaticamente através dos modelos de dados.

DJANGO FRAMEWORK

... PRINCIPAIS CARACTERÍSTICAS

- URLs amigáveis:
  - Permite a criação de URLs de maneira simples e que tenham aparência amigável.
- Sistemas de Cache:
  - O framework possui um sistema de cache que se integra ao memcached ou em outros frameworks de cache.
- Internacionalização:
  - Possui total suporte para aplicações multi-idioma. Permite ao desenvolvedor especificar strings de tradução e fornece ganchos para funcionalidades específicas do idioma.

INSTALANDO **DJANGO** EM NOSSO AMBIENTE VIRTUAL

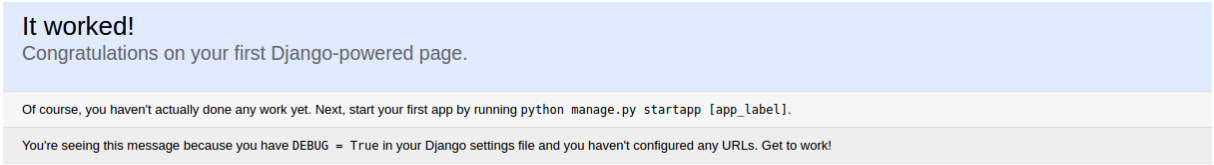
- Entrar no diretorio virtual e instalar framework Django
  - *$ cd virtual*
  - *$ pip install Django*

- Criar a estrutura base do sistema com o framework Django
  - *$ python -m django –version*
  - *$ django-admin startproject sistema*
  - *$ python manage.py runserver*
  - *$ chmod +x manage.py*

INSTALANDO DJANGO EM NOSSO AMBIENTE VIRTUAL

Tutorial disponível em :

- https://docs.djangoproject.com/en/1.10/intro/tutorial01/



ESTRUTURA DE ARQUIVOS DO NOSSO PROJETO DJANGO

```
sitema/
    manage.py
    db.sqlite3
    sistema/
        __init__.py
        settings.py
        urls.py
        wsgi.py
```

...CONFIGURANDO O AMBIENTE DO PROJETO

- Definindo os formatos de data, hora e de linguagem
  - No arquivo de configurações do projeto (sistema/sistema/settings.py)
    - Substitua as definições de # Internationalization:

```
LANGUAGE_CODE = 'en-us'
TIME_ZONE = 'UTC'
USE_I18N = True
USE_L10N = True
USE_TZ = True
```

    - Por:

```
LANGUAGE_CODE = 'pt-br'
TIME_ZONE = 'America/Araguaina'
USE_I18N = True
USE_L10N = True
USE_TZ = True
DATE_FORMAT = '%d/%m/%Y'
TIME_FORMAT = '%H:%M:%S'
DATE_INPUT_FORMATS = ('%d/%m/%Y',)
```

CONFIGURANDO O AMBIENTE DO PROJETO...

- Instalando pacote para gestão de diretórios
  - *$ pip install Unipath*
- Configurando diretório base do projeto
  - No arquivo de configurações do projeto (sistema/sistema/settings.py)
    - Substitua as seguintes linhas:

```
import os

# Build paths inside the project like this: os.path.join(BASE_DIR, ...)

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

    - Por:

```
from unipath import Path

BASE_DIR = Path(__file__).parent
```

...CONFIGURANDO O AMBIENTE DO PROJETO

- Definindo a localização dos arquivos de templates
  - No arquivo de configurações do projeto (sistema/sistema/settings.py)
    - Substitua a seguinte linha na definição da variável TEMPLATES:

      ```
      'DIRS': [],
      ```

    - Por:

      ```
      'DIRS': [BASE_DIR.parent.child('templates')],
      ```

...CONFIGURANDO O AMBIENTE DO PROJETO

- Definindo a localização dos arquivos estáticos (js, css, imagens)
  - No arquivo de configurações do projeto (sistema/sistema/settings.py)
    - Substitua a definição da variável STATIC_URL:

      ```
      STATIC_URL = '/static/'
      ```

    - Por:

      ```
      STATIC_URL = '/static/'
      STATICFILES_DIRS = (BASE_DIR.child('static'),)
      STATIC_ROOT = BASE_DIR.parent.child('static')
      ```

- Verificando se todas as alterações estão corretas
  - *$ python manage.py check*

...CONFIGURANDO O AMBIENTE DO PROJETO

- Definindo a base de dados utilizada pelo projeto
  - Instalando o sistema gerenciador de banco de dados PostgreSQL
    - ....
  - Instalando o driver de conexao com o SGBD
    - *$ pip install psycopg2*
  - Com o auxílio da ferramenta pg_admin crie um novo banco de dados
    - programacao_comercial

...CONFIGURANDO O AMBIENTE DO PROJETO

- Configurando a base de dados utilizada pelo projeto
  - No arquivo de configurações do projeto (sistema/sistema/settings.py)
    - Substitua a definição da variável DATABASES:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}
```

    - Por:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'programacao_comercial',
        'USER': 'postgres',
        'PASSWORD': 'postgres',
        'HOST': '127.0.0.1',
        'PORT': '5432',
    }
}
```

CONSTRUINDO UM SISTEMA PARA

CADASTRO E LOCAÇÃO DE VEÍCULOS

CRIANDO O APP **VEICULOS** E ADICIONANDO AO PROJETO...

- Filosofia
  - Aplicações Django são "plugáveis": Você pode usar um aplicativo em múltiplos projetos.
  - Um sistema Django consiste em um conjunto de APP's fracamente acopladas.
- Criando o APP veiculos:
  - *$ python manage.py startapp veiculos*
- Estrutura básica gerada para o APP veiculos (sistema/veiculos/)
  - veiculos/
    migrations/
    admin.py
    apps.py
    __init__.py
    models.py
    tests.py
    views.py

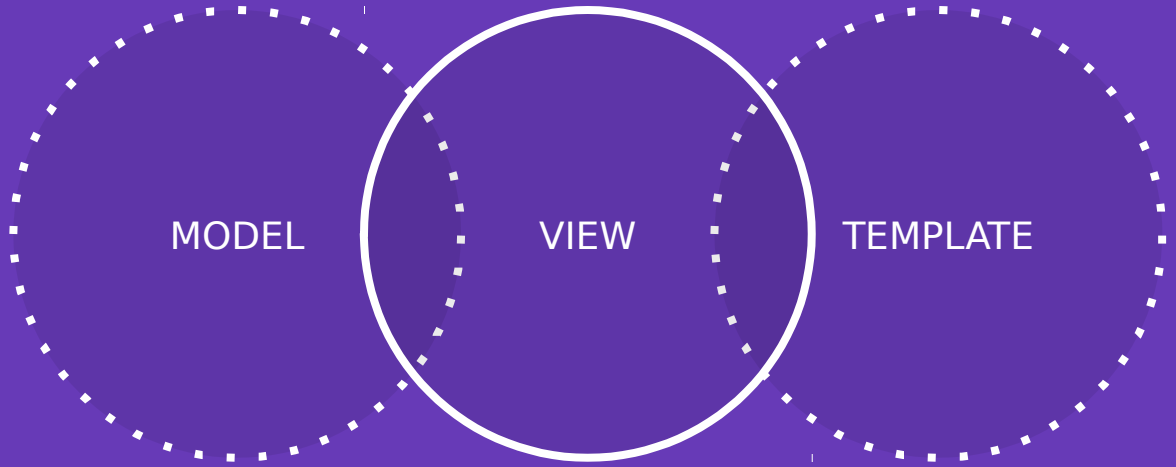
...CRIANDO O APP **VEICULOS** E ADICIONANDO AO PROJETO

- Adicionando o APP veiculos à lista de APP's do sistema
  - No arquivo de configurações do projeto (sistema/sistema/settings.py)
    - Altere a definição da variável INSTALLED_APPS:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

    - Por:

```
INSTALLED_APPS = [
    'veiculos.apps.VeiculosConfig',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```



DEFININDO OS COMPONENTES **MVT** DO APP

DEFININDO NOSSO **MODELO...**

- No arquivo veiculos/models.py adicione o seguinte código:

```
class Veiculo(models.Model):
    marca = models.CharField(max_length=100)
    modelo = models.CharField(max_length=100)
    ano_fabricacao = models.IntegerField()
    modelo_fabricacao = models.IntegerField()
    combustivel = models.SmallIntegerField(choices=[(1, 'ETANOL'), (2, 'FLEX'), (3, 'GASOLINA')])

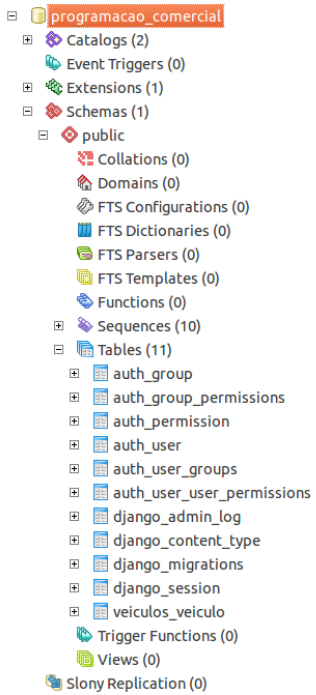

  def __str__(self):
    return '{0} - {1} ({2}/{3})'.format(self.marca, self.modelo, self.ano_fabricacao, self.modelo_fabricacao)
```

...DEFININDO NOSSO **MODELO**

- Sincronizando modelos e base de dados
  - *$ python manage.py makemigrations veiculos*
  - *$ python manage.py migrate*

```
Migrations for 'veiculos':
  0001_initial.py:
    - Create model Veiculo
```

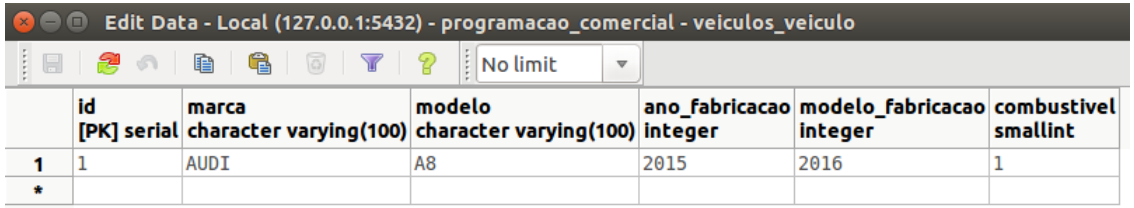
```
Operations to perform:
  Apply all migrations: admin, contenttypes, veiculos, auth, sessions
Running migrations:
  Rendering model states... DONE
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying sessions.0001_initial... OK
  Applying veiculos.0001_initial... OK
```



DEFININDO NOSSO **MODELO...**

- Manipulando objetos por meio do shell do projeto
  - *$ python manage.py shell*

```
from veiculos.models import *
primeiro = Veiculo()
primeiro.marca = 'AUDI'
primeiro.modelo = 'A8'
primeiro.ano_fabricacao = 2015
primeiro.modelo_fabricacao = 2016
primeiro.combustivel = 1
primeiro.save()
```



	id [PK] serial	marca character varying(100)	modelo character varying(100)	ano_fabricacao integer	modelo_fabricacao integer	combustivel smallint
1	1	AUDI	A8	2015	2016	1
*						

DEFININDO NOSSO **MODELO...**

- Manipulando objetos por meio do shell do projeto (continuação)
  - segundo = Veiculo(marca='KIA', modelo='OPTIMA', ano_fabricacao=2016, modelo_fabricacao=2017, combustivel=2)
    segundo.save()

Edit Data - Local (127.0.0.1:5432) - programacao_comercial - veiculos_veiculo

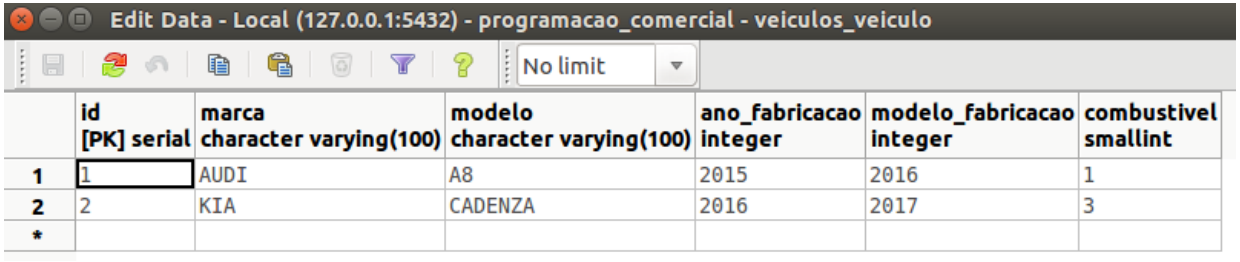
No limit

	id [PK] serial	marca character varying(100)	modelo character varying(100)	ano_fabricacao integer	modelo_fabricacao integer	combustivel smallint
1	1	AUDI	A8	2015	2016	1
2	2	KIA	OPTIMA	2016	2017	2
*						

DEFININDO NOSSO **MODELO...**

- Manipulando objetos por meio do shell do projeto (continuação)

  - ```
    segundo.modelo = 'CADENZA'
    segundo.combustivel = 3
    segundo.save()
    ```



	id [PK] serial	marca character varying(100)	modelo character varying(100)	ano_fabricacao integer	modelo_fabricacao integer	combustivel smallint
1	1	AUDI	A8	2015	2016	1
2	2	KIA	CADENZA	2016	2017	3
*						
    ```

DEFININDO NOSSO **MODELO...**

- Manipulando objetos por meio do shell do projeto (continuação)

```
In [19]: Veiculo.objects.filter(marca='AUDI').count()
Out[19]: 1

In [20]: for veiculo in Veiculo.objects.all():
    ...:     print veiculo.marca
    ...:
AUDI
KIA

In [21]: veiculo = Veiculo.objects.get(id=1)

In [22]: veiculo.modelo
Out[22]: u'A8'

In [23]: for veiculo in Veiculo.objects.exclude(marca='AUDI'):
    ...:     print veiculo.marca
    ...:
KIA

In [24]:

In [24]: Veiculo.objects.get(id=1).delete()
Out[24]: (1, {u'veiculos.Veiculo': 1})

In [25]:

In [25]: for veiculo in Veiculo.objects.all():
    ...:     print veiculo.marca
    ...:
    ...:
KIA

In [26]:
```

DEFININDO NOSSA **VIEW...**

- Views Django são acionadas a partir do recebimento de requisições a URL's
- O arquivo urls.py define os mapeamentos de URLs. Utiliza o padrãoo regex (expressão regular) para analisar a sequência de caracteres da URL e mapeá-la para suas devidas funções.
- A URL é totalmente desassociada da estrutura do projeto, assim, por exemplo, pode-se renomear o aplicativo ou classes sem afetar as URLs.
- Django utiliza HttpRequest e HttpResponse para a comunicação. HttpRequest contém metadados sobre o que foi solicitado e cada view é responsável por devolver um objeto HttpResponse.
- A URL pode ser considerada o DNS do aplicativo Django. É simplesmente um endereço da web e pode ser vista na barra de endereços do navegador. Todas solicitações passam pelo urls.py.

...DEFININDO NOSSA **VIEW**

- Nossa primeira URL para o APP veiculos
- Criar o arquivo veiculos/urls.py
  - No arquivo veiculos/urls.py inclua o seguinte código:

```
from django.conf.urls import url

from veiculos import views

urlpatterns = [

    url(r'^$', views.index, name='index'),

]
```

...DEFININDO NOSSA **VIEW**

O próximo passo é associar as URL's do APP veiculos ao módulo principal de rotas do sistema.

- No arquivo sistema/sitema/urls.py, adicione:

```
from django.conf.urls import include, url

from django.contrib import admin

urlpatterns = [

    url(r'^veiculos/', include('veiculos.urls')),

    url(r'^admin/', admin.site.urls),

]
```

...DEFININDO NOSSA **VIEW**

Por fim, vamos escrever nossa primeira view.

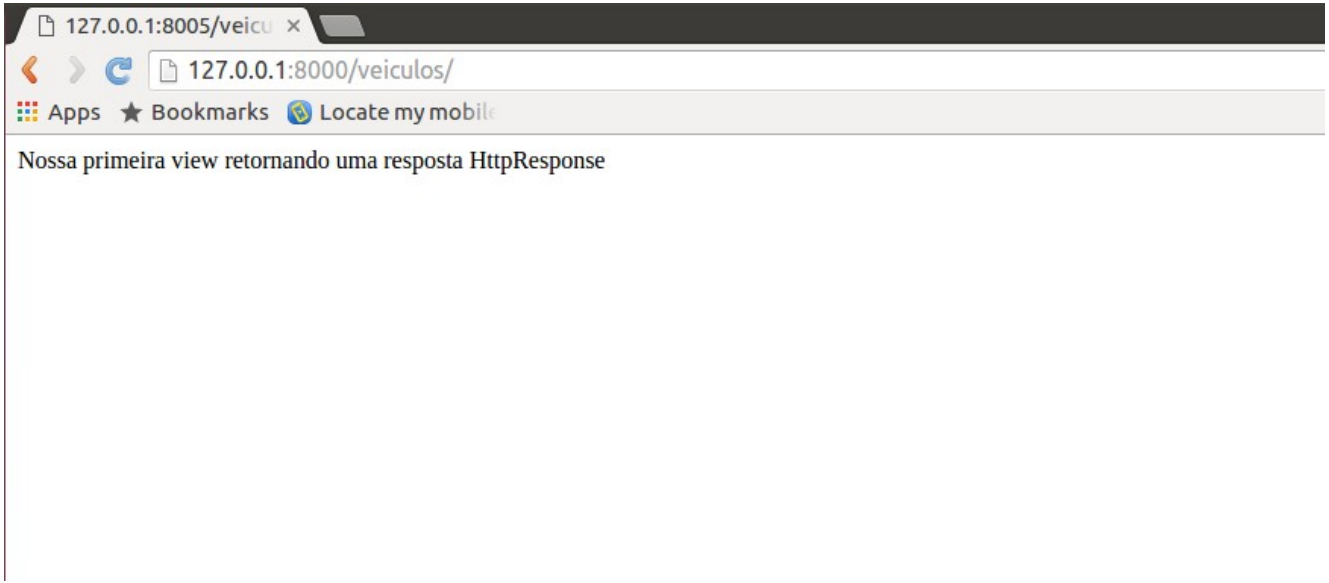
- No arquivo veiculos/views.py, adicione:

```
from django.http import HttpResponse

def index(request):

    return HttpResponse("Nossa primeira view retornando uma resposta HttpResponse")
```

...DEFININDO NOSSA **VIEW**



...DEFININDO NOSSA **VIEW**

Conhecendo melhor o protocolo HTTP.

- Hypertext Transfer Protocol (HTTP) - RFC 2616.

- O protocolo HTTP é baseado em requisições e respostas entre clientes e servidores. O cliente — navegador ou dispositivo que fará a requisição; também é conhecido como *user agent* — solicita um determinado recurso (*resource*), enviando um pacote de informações contendo alguns cabeçalhos (*headers*) a um URI ou, mais especificamente, URL. O servidor recebe estas informações e envia uma resposta, que pode ser um recurso ou um simplesmente um outro cabeçalho.

...DEFININDO NOSSA **VIEW**

Métodos HTTP.

- Ao executar uma requisição, é preciso especificar qual o método será utilizado. Os métodos HTTP (também conhecidos como verbos) identificam qual a ação que deve ser executada em um determinado recurso. Existem 8 métodos HTTP, mas apenas 4 são mais utilizados.

  - GET - Solicita a representação de um determinado recurso. Não deve ser usado para disparar uma ação (remover um usuário, por exemplo).
  - POST - As informações enviadas no corpo (body) da requisição são utilizadas para criar um novo recurso.
  - DELETE - Remove um recurso. Deve retornar o status 204 caso não exista nenhum recurso para a URI especificada.
  - PUT - Atualiza um recurso na URI especificada. Caso o recurso não exista, ele pode criar um.

...DEFININDO NOSSA **VIEW**

CLASS BASED VIEWS

São views (django) com base em classes (python), não em funções. Isto significa que para dominá-las é preciso entender tanto as views em Django quanto as classes em python. Como visto anteriormente, uma view em Django consiste em um pedaço de código que processa uma solicitação HTTP e devolve uma resposta HTTP, e uma classe em Python é a implementação do conceito de orientação à objetos.

As CBVs incentivam o reuso de código e a criação de views modulares, o seu sucesso vem em detrimento da simplicidade e do uso de heranças. Este é um padrão que permite reutilizar funcionalidades comuns e ajuda a manter o princípio do DRY do Django.

...DEFININDO NOSSA **VIEW**

CLASS BASED VIEWS

O framework Django fornece algumas diretrizes a serem seguidas quando houver a criação de CBVs:

- Não repetir código nas views.
- Manter views com código o menor possível (coesão).
- Views devem lidar com a lógica de apresentação e não com a lógica de negócios, pois esta deve-se tentar manter nos Forms ou Modelos de Classes.
- As views e os *mixins* (serão estudados mais tarde) devem ser simples.

...DEFININDO NOSSA **VIEW**

- Refatorando nossa view para torná-la uma class based
  - Alterar o código do arquivo veiculos/views.py para:

```
# -*- coding: utf-8 -*-
from django.views.generic import View
from django.shortcuts import render

class VeiculosList(View):
    """
    View para listar veiculos cadastrados.
    """

    def get(self, request):
        context = {
            'object_list': Veiculo.objects.all().order_by('marca')
        }
        return render(request, 'veiculos/listar.html', context)
```

...DEFININDO NOSSA **VIEW**

- Refatorando nossa URL para torná-la apta a manipular *class based views*
- Alterar o arquivo de URL's
  - Altere o código do arquivo veiculos/urls.py para o seguinte:

```
from django.conf.urls import url
from veiculos import views
urlpatterns = [
    url(r'^$', views.VeiculosList.as_view(), name='listar-veiculos'),
]
```

DEFININDO NOSSOS TEMPLATES...

- Criando a estrutura de templates
  - No diretório raiz do projeto, criar os seguintes diretórios:
    - templates/
    - templates/veiculos
  - Criar o seguinte arquivo de template em templates/veiculos:
    - listar.html

...DEFININDO NOSSOS TEMPLATES

- Alterando o arquivo de template para listar todos os veiculos cadastrados.
  - Adicione o seguinte código ao arquivo de template: templates/veiculos/listar.html

```
<html>
   <head>
      <meta charset="UTF-8">
      <title>Lista de veículos cadastrados</title>
   </head>
   <body>
      <ul>
         {% for veiculo in object_list %}
            <li>
               {{ veiculo }}
            </li>
         {% endfor %}
      </ul>
   </body>
</html>
```

  - O arquivo de template listar.html itera sobre uma lista de veiculos (object_list) associada ao contexto (context) retornado pela view.

...DEFININDO NOSSOS TEMPLATES

- Resultado

UTLIZANDO **MIXIN'S** EM VIEWS...

Um mixin é uma classe que fornece funcionalidade para ser herdada, mas não está destinada a instanciação de si própria. Em linguagens de programação com herança múltipla, mixins são um meio de coleta de funcionalidade.

Facilitam o trabalho do desenvolvedor ao prover um conjunto de funcionalidades prontas.

...UTLIZANDO **MIXIN'S** EM VIEWS

- Refatorando a view VeiculosList
- Adicione herança do Mixin ListView à view VeiculosList
  - Substitua o código do arquivo veiculos/views.py pelo seguinte:

```
# -*- coding: utf-8 -*-
from django.views.generic import ListView
from veiculos.models import *

class VeiculosList(ListView):
    """
    View para listar veiculos cadastrados.
    """
    model = Veiculo
    template_name = 'veiculos/listar.html'
```

DJANGO MODELFORMS – FORMULÁRIOS A PARTIR DE MODELOS

- Django fornece recursos para a criação de formulários a partir de modelos.
- Criando um formulário (veiculos/forms.py) para o modelo Veiculo:
  - Crie um novo arquivo chamado forms.py dentro do APP veiculos:
  - Adicione o seguinte código ao arquivo criado:

```
# -*- coding: utf-8 -*-
from django import forms
from veiculos.models import *

class FormularioVeiculo(forms.ModelForm):
    """
    Formulario para o model Veiculo
    """

    class Meta:
        model = Veiculo
        exclude = []
```

ADICIONANDO VIEW PARA **CADASTRO** DE NOVO VEÍCULOS...

- Criando uma nova view para gerenciar o cadastro de novos veículos:
  - Adicione as importações do Mixin CreateView, do formulário FormularioVeiculo e do método reverse_lazy (DNS reverso) no arquivo veiculos/views.py:

```
from django.views.generic import ListView, CreateView
from django.core.urlresolvers import reverse_lazy
from veiculos.forms import *
```

  - Adicione a definição da view para cadastro ao final do arquivo veiculos/views.py:

```
class VeiculosNew(CreateView):
    """
    View para a criação de novos veiculos.
    """
    model = Veiculo
    form_class = FormularioVeiculo
    template_name = 'veiculos/novo.html'
    success_url = reverse_lazy('listar-veiculos')
```

...ADICIONANDO VIEW PARA **CADASTRO** DE NOVO VEÍCULOS

- Criando uma nova view para gerenciar o cadastro de novos veículos:
  - Adicionar nova url ao arquivo veiculos/urls.py

```
url(r'^novo/$', views.VeiculosNew.as_view(), name='novo-veiculo'),
```

  - Criar um novo arquivo de template em templates/veiculos/novo.html:

```
<html>
  <head>
    <meta charset="UTF-8">
    <title>Cadastrar novo veículo</title>
  </head>
  <body>
    <form action="/veiculos/novo/" method="post">
      {% csrf_token %}
      {{ form }}
      <input type="submit" value="Salvar" />
    </form>
  </body>
</html>
```

ADICIONANDO VIEW PARA **EDIÇÃO** DE VEÍCULOS CADASTRADOS...

- Criando uma nova view para gerenciar a edição de veículos já cadastrados:
  - Adicione a importação do Mixin UpdateView no arquivo veiculos/views.py:

```
from django.views.generic import ListView, CreateView, UpdateView
```

  - Adicione a definição da view para edição ao final do arquivo veiculos/views.py:

```
class VeiculosEdit(UpdateView):
    """
    View para a edição de veiculos já cadastrados.
    """
    model = Veiculo
    form_class = FormularioVeiculo
    template_name = 'veiculos/editar.html'
    success_url = reverse_lazy('listar-veiculos')
```

...ADICIONANDO VIEW PARA **EDIÇÃO** DE VEÍCULOS CADASTRADOS

- Criando uma nova view para gerenciar a edição de veículos já cadastrados:
  - Adicionar nova url ao arquivo veiculos/urls.py

```
url(r'^(?P<pk>[0-9]+)/$', views.VeiculosEdit.as_view(), name='editar-veiculo'),
```

  - Criar um novo arquivo de template em templates/veiculos/editar.html:

```
<html>
   <head>
      <meta charset="UTF-8">
      <title>Alterar veículo</title>
   </head>
   <body>
      <form action="/veiculos/{{ object.pk }}/" method="post">
         {% csrf_token %}
         {{ form }}
         <input type="submit" value="Salvar" />
      </form>
   </body>
</html>
```

ADICIONANDO VIEW PARA EXCLUSÃO DE VEÍCULOS CADASTRADOS...

- Criando uma nova view para gerenciar a exclusão de veículos já cadastrados:
  - Adicione a importação do Mixin DeleteView no arquivo veiculos/views.py:

```
from django.views.generic import ListView, CreateView, UpdateView, DeleteView
```

  - Adicione a definição da view para exclusão ao final do arquivo veiculos/views.py:

```
class VeiculosDelete(DeleteView):
    """
    View para a exclusão de veiculos.
    """
    model = Veiculo
    template_name = 'veiculos/deletar.html'
    success_url = reverse_lazy('listar-veiculos')
```

...ADICIONANDO VIEW PARA EXCLUSÃO DE VEÍCULOS CADASTRADOS

- Criando uma nova view para gerenciar a exclusão de veículos já cadastrados:
  - Adicionar nova url ao arquivo veiculos/urls.py

```
url(r'^excluir/(?P<pk>[0-9]+)/$', views.VeiculosDelete.as_view(), name='deletar-veiculo'),
```

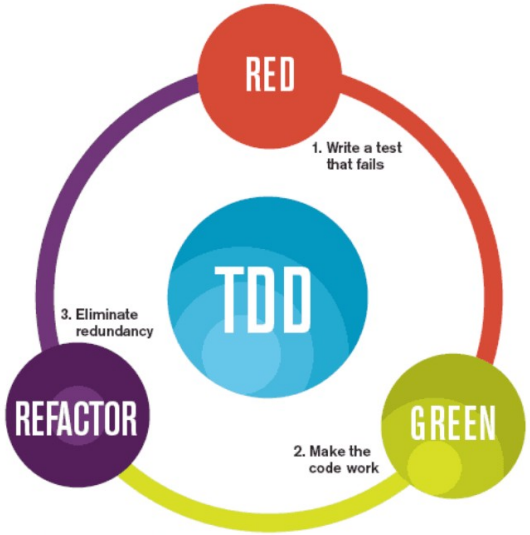
  - Criar um novo arquivo de template em templates/veiculos/deletar.html:

```
<html>
   <head>
      <meta charset="UTF-8">
      <title>Excluir veículo</title>
   </head>
   <body>
      <form action="/veiculos/excluir/{{ object.pk }}/" method="post">
         {% csrf_token %}
         <p>
            Tem certeza que deseja excluir o veículo "{{ object }}" ?
         </p>
         <input type="submit" value="Confirmar" />
      </form>
   </body>
</html>
```

TEST DRIVEN DEVELOPMENT – TDD

- Test Driven Development (TDD) ou em português desenvolvimento orientado a testes é uma técnica de desenvolvimento de software que se baseia em um ciclo curto de repetições (baby steps):
  - Primeiramente o desenvolvedor escreve um caso de teste automatizado que define uma melhoria desejada ou uma nova funcionalidade.
  - Então, é produzido código que possa ser validado pelo teste.
  - Posteriormente o código é refatorado para um código sob padrões aceitáveis.
- Kent Beck (criador do eXtreme Programming e um dos signatários do Manifesto Ágil), considerado o criador ou o 'descobridor' da técnica, declarou em 2003 que TDD encoraja designs de código simples e inspira confiança.
- Através de TDD, programadores podem aplicar o conceito de melhorar e depurar código legado desenvolvido a partir de técnicas antigas.
- Mais do que simplesmente testar seu código, TDD é uma filosofia, uma cultura.

TEST DRIVEN DEVELOPMENT – TDD



The mantra of Test-Driven Development (TDD) is "red, green, refactor."

APLICANDO TEST DRIVEN DEVELOPMENT AO PROJETO...

- Código sem testes é código já quebrado quando foi planejado
  - Jacok Kaplan Moss (um dos criadores do Django)
- O framework Django fornece uma API de alto nível para TDD
  - Testes são rotinas simples que checam o funcionamento do seu código.
  - Testes não só identificam problemas, mas também os previnem.
  - Testes tornam seu código mais atraente.
  - Testes ajudam as equipes a trabalharem juntos.
- Estratégias básicas de testes (pessoal)
  - Escrever testes para os Modelos
  - Escrever testes para as Views
  - Escrever testes para as API's (caso existam)

...APLICANDO TEST DRIVEN DEVELOPMENT AO PROJETO

- Escrevendo os testes para a View VeiculosList
  - No arquivo de testes veiculos/tests.py adicione o seguinte conteúdo:

```
from django.test import TestCase
from django.core.urlresolvers import reverse
from veiculos.models import *
from veiculos.forms import *

class TestViewVeiculosList(TestCase):
    '''
    Classe de testes para a view VeiculosList
    '''
    def setUp(self):
        self.url = reverse('listar-veiculos')
        Veiculo(marca='aaa', modelo='aaa', ano_fabricacao=1, modelo_fabricacao=2, combustivel=3).save()

    def test_get(self):
        response = self.client.get(self.url)
        self.assertEqual(response.status_code, 200)
        self.assertEqual(len(response.context.get('object_list')), 1)
```

...APLICANDO TEST DRIVEN DEVELOPMENT AO PROJETO

- Escrevendo os testes para a View VeiculosNew
  - No final do arquivo de testes veiculos/tests.py adicione o seguinte conteúdo:

```
class TestViewVeiculosNew(TestCase):
    '''
    Classe de testes para a view VeiculosNew
    '''
    def setUp(self):
        self.url = reverse('novo-veiculo')

    def test_get(self):
        response = self.client.get(self.url)
        self.assertEqual(response.status_code, 200)
        self.assertIsInstance(response.context.get('form'), FormularioVeiculo)

    def test_post(self):
        data = {'marca': 'aaa', 'modelo': 'aaa', 'ano_fabricacao': 1, 'modelo_fabricacao': 2, 'combustivel': 3}
        response = self.client.post(self.url, data)

        # Verifica se apos a insercao houve um redirecionamento para a view VeiculosList
        self.assertEqual(response.status_code, 302)
        self.assertRedirects(response, reverse('listar-veiculos'))

        self.assertEqual(Veiculo.objects.count(), 1)
        self.assertEqual(Veiculo.objects.first().marca, 'aaa')
```

...APLICANDO TEST DRIVEN DEVELOPMENT AO PROJETO

- Escrevendo os testes para a View VeiculosEdit
  - No final do arquivo de testes veiculos/tests.py adicione o seguinte conteúdo:

```
class TestViewVeiculosEdit(TestCase):
    '''
    Classe de testes para a view VeiculosEdit
    '''
    def setUp(self):
        self.instance = Veiculo.objects.create(marca='aaa', modelo='aaa', ano_fabricacao=1, modelo_fabricacao=2, combustivel=3)
        self.url = reverse('editar-veiculo', kwargs={'pk': self.instance.pk})

    def test_get(self):
        response = self.client.get(self.url)
        self.assertEqual(response.status_code, 200)
        self.assertIsInstance(response.context.get('object'), Veiculo)
        self.assertIsInstance(response.context.get('form'), FormularioVeiculo)
        self.assertEqual(response.context.get('object').pk, self.instance.pk)

    def test_post(self):
        data = {'marca': 'zzz', 'modelo': 'zzz', 'ano_fabricacao': 3, 'modelo_fabricacao': 2, 'combustivel': 1}
        response = self.client.post(self.url, data)
        self.assertEqual(response.status_code, 302)
        self.assertRedirects(response, reverse('listar-veiculos'))
        self.assertEqual(Veiculo.objects.count(), 1)
        self.assertEqual(Veiculo.objects.first().marca, 'zzz')
        self.assertEqual(Veiculo.objects.first().pk, self.instance.pk)
```

...APLICANDO TEST DRIVEN DEVELOPMENT AO PROJETO

- Escrevendo os testes para a View VeiculosDelete
  - No final do arquivo de testes veiculos/tests.py adicione o seguinte conteúdo:

```
class TestViewVeiculosDelete(TestCase):
    '''
    Classe de testes para a view VeiculosDelete
    '''
    def setUp(self):
        self.instance = Veiculo.objects.create(marca='aaa', modelo='aaa', ano_fabricacao=1, modelo_fabricacao=2, combustivel=3)
        self.url = reverse('deletar-veiculo', kwargs={'pk': self.instance.pk})

    def test_get(self):
        response = self.client.get(self.url)
        self.assertEqual(response.status_code, 200)
        self.assertIsInstance(response.context.get('object'), Veiculo)
        self.assertEqual(response.context.get('object').pk, self.instance.pk)

    def test_post(self):
        response = self.client.post(self.url)

        # Verifica se apos a exclusao houve um redirecionamento para a view VeiculosList
        self.assertEqual(response.status_code, 302)
        self.assertRedirects(response, reverse('listar-veiculos'))
        self.assertEqual(Veiculo.objects.count(), 0)
```

...APLICANDO TEST DRIVEN DEVELOPMENT AO PROJETO

- Escrevendo os testes para o Model Veiculo
  - Testes de modelos devem focar nas regras de negócio definidas no model
  - Adicione ao arquivo de testes veiculos/tests.py o seguinte conteúdo:

```
from datetime import datetime

class TestModelVeiculo(TestCase):
    '''
    Classe de testes para o model Veiculo
    '''
    def setUp(self):
        self.instance = Veiculo(marca='aaa', modelo='aaa', ano_fabricacao=datetime.now().year, modelo_fabricacao=2, combustivel=3)

    def test_is_new(self):
        self.assertTrue(self.instance.veiculo_novo)
        self.instance.ano_fabricacao = datetime.now().year – 5
        self.assertFalse(self.instance.veiculo_novo)

    def test_years_use(self):
        self.assertEqual(self.instance.anos_de_uso(), 0)
        self.instance.ano_fabricacao = datetime.now().year – 5
        self.assertEqual(self.instance.anos_de_uso(), 5)
```

...APLICANDO TEST DRIVEN DEVELOPMENT AO PROJETO

- Escrevendo os testes para o Model Veiculo
  - Property:
    - Também conhecido como 'atributos gerenciáveis'.
    - Esta é uma maneira elegante de criar atributos que são acessíveis como atributos, mas que podem ser implementados através de chamadas de métodos (*getters e setters*).
  - Adicionando uma property ao model Veiculo
    - No arquivo de modelos veiculos/models.py adicione o seguinte código:

```
from datetime import datetime


@property
def veiculo_novo(self):
    return self.ano_fabricacao == datetime.now().year
```

  - Adicionando um método para verificar anos de uso ao model Veiculo
    - No arquivo de modelos veiculos/models.py adicione o seguinte código:

```
def anos_de_uso(self):
    return datetime.now().year - self.ano_fabricacao
```

DJANGO **ADMIN MANAGER**

Adicionando gerenciamento do sistema por meio dos recursos oferecidos pelo framework.

DJANGO ADMIN MANAGER

- O framework Django fornece também uma API para gestão administrativa de modelos.
  - Basta registrar os modelos junto ao admin do Django
- O acesso aos recursos de gestão administrativa é controlado com base nas permissões dos usuários.
  - Um sistema pode conter inúmeros usuários
  - Usuários estão associados a permissões
- Criando o super usuário (admin) do nosso projeto
  - *$ python manage.py createsuperuser*

ADICIONANDO GESTOR ADMINISTRATIVO AO PROJETO

- Adicione o seguinte código ao arquivo veiculos/admin.py:

```
from django.contrib import admin
from veiculos.models import *

class VeiculoAdmin(admin.ModelAdmin):
    """
    Admin para o model Veiculo
    """
    list_display = ['marca', 'modelo', 'ano_fabricacao', 'modelo_fabricacao', 'combustivel']
    search_fields = ['marca', 'modelo']
    list_filter = ['combustivel']

admin.site.register(Veiculo, VeiculoAdmin)
```

- Por fim, acesse o painel administrativo do sistema:
  - http://127.0.0.1:8000/admin/

RESULTADO


Administração do Si
127.0.0.1:8000/admin/
Apps
Bookmarks
Locate my mobile
Form handling w
Administração do Django
Administração do Site
AUTENTICAÇÃO E AUTORIZAÇÃO
Grupos
Adicionar
Modificar
Usuários
Adicionar
Modificar
VEICULOS
Veiculos
Adicionar
Modificar
Ações Recentes
Minhas Ações
Nenhum disponível

FIM!

Dúvidas?

thiagombrodrigues@mail.uft.edu.br