MÓDULO 1

PYTHON

AULA 12

Prof. Danilo Pereira

ÍNDICE

OBJETIVOS

- Consolidar todo o conhecimento em um projeto prático.
- Fazer a conexão final e mostrar como os conceitos se aplicam diretamente em um projeto Django.

DESAFIO

- Vamos construir uma aplicação de Agenda de Contatos via terminal.

REQUISITOS

1. Deve permitir Adicionar, Listar, Buscar e Remover contatos.
2. Cada contato terá nome, telefone e e-mail.
3. Os dados devem ser salvos em um arquivo para não serem perdidos.
4. O programa deve ter um menu interativo e tratar entradas inválidas.

PEÇAS DO QUEBRA-CABEÇA (CONCEITOS QUE USAREMOS)

- class Contato: Para modelar nossos dados (POO).
- lista_contatos = []: Para guardar os objetos Contato em memória (Listas e Objetos).
- Menu com while True: Para manter o programa rodando (Laços).
- Funções para cada ação (adicionar_contato(), listar_contatos(), etc.) (Funções).
- with open(): Para salvar e carregar os dados de um arquivo (Manipulação de Arquivos).
- try...except: Para lidar com erros de usuário ou de arquivo (Tratamento de Erros).

PEÇAS DO QUEBRA-CABEÇA (CONCEITOS QUE USAREMOS)

- json.load(arquivo) e json.dump(dados_para_salvar, arquivo, indent=4, ensure_ascii=False) para manipular arquivos json;
- except (FileNotFoundError, json.JSONDecodeError) para tratar erros do arquivo;
- if __name__ == "__main__": menu_principal() para chamar a função principal do programa;

• PASSO 1: A CLASSE CONTATO

```
class Contato:
    """Classe que representa um contato na agenda."""
    def __init__(self, nome, telefone, email):
        self.nome = nome
        self.telefone = telefone
        self.email = email
```

PASSO 2: FUNÇÕES DE PERSISTÊNCIA (SALVAR/CARREGAR)

- **salvar_contatos(contatos): Itera na lista, transforma cada objeto em uma string formatada (ex: Nome;Telefone;Email) e salva no arquivo.**

- **carregar_contatos(): Lê o arquivo, quebra cada linha no ; e recria os objetos Contato.**

PASSO 3: FUNÇÕES DE NEGÓCIO

- adicionar_contato(), listar_contatos(), etc.

PASSO 4: O LOOP PRINCIPAL (WHILE TRUE)

- Chama as funções com base na escolha do usuário.

PONTE PARA O DJANGO: ONDE TUDO SE CONECTA

- Vamos ver como nosso projeto se parece em Django.

PONTE PARA O DJANGO: ONDE TUDO SE CONECTA

- **agenda/models.py (A nossa classe Contato)**

```
from django.db import models
class Contato(models.Model):
    nome = models.CharField(max_length=120)
    telefone = models.CharField(max_length=20)
    email = models.EmailField()
```

PONTE PARA O DJANGO: ONDE TUDO SE CONECTA

- **agenda/views.py (Nossas funções de negócio)**

```
from django.shortcuts import render
from .models import Contato
def listar_contatos(request):
    contatos = Contato.objects.all() # Pega todos os contatos do banco
    return render(request, 'lista.html', {'contatos': contatos})
```

PONTE PARA O DJANGO: ONDE TUDO SE CONECTA

- **agenda/templates/lista.html (A parte de visualização)**

```
<h1>Meus Contatos</h1>
<ul>
  {% for contato in contatos %}
    <li>{{ contato.nome }} - {{ contato.telefone }}</li>
  {% endfor %}
</ul>
```

PONTE PARA O DJANGO: ONDE TUDO SE CONECTA

- Viram? Classe vira Model. Lista vira QuerySet. Laço for vira laço no template. Funções viram Views. É a mesma lógica, com as ferramentas do Django!

REVISÃO E ENCERRAMENTO

- Você agora tem uma base sólida de Python e, mais importante, entende como essa base se aplica na construção de aplicações web complexas.

- O caminho do desenvolvimento está aberto!

DANILO PEREIRA

danilocesarpereira@gmail.com