

# Apostila Completa: Python Backend + AI/DS

## Índice

1. [Fundamentos de Python e Git](#)
2. [Bancos de Dados e Django Básico](#)
3. [Python Avançado e Estruturas de Dados](#)
4. [Django Avançado, FastAPI e DevOps](#)
5. [Ciência de Dados e Machine Learning](#)
6. [Projetos Práticos e Preparação para Entrevistas](#)
7. [Temas Avançados para Desenvolvedor Júnior](#)
8. [Recursos e Referências](#)

---

## 1. Fundamentos de Python e Git {#fundamentos-python-git}

### 1.1 Sintaxe Básica de Python

#### Tipos de Dados Básicos

```
python

# Números
inteiro = 42
decimal = 3.14
complexo = 2 + 3j

# Strings
texto = "Hello, World!"
texto_multilinhas = """
Texto com
múltiplas linhas
"""

# Booleanos
verdadeiro = True
falso = False
```

#### Estruturas de Dados

##### Listas

```
python
```

```
# Criação e manipulação de listas
lista = [1, 2, 3, 4, 5]
lista_mista = [1, "texto", 3.14, True]

# Métodos essenciais
lista.append(6)      # Adiciona elemento
lista.remove(2)       # Remove elemento
lista.pop()           # Remove último elemento
lista.insert(0, 0)     # Insere em posição específica
```

## Dicionários

```
python

# Criação e manipulação
pessoa = {
    "nome": "João",
    "idade": 30,
    "cidade": "São Paulo"
}

# Acesso e modificação
print(pessoa["nome"])
pessoa["profissao"] = "Desenvolvedor"
pessoa.update({"telefone": "123456789"})
```

## Condicionais e Loops

### Estruturas Condicionais

```
python

idade = 18

if idade >= 18:
    print("Maior de idade")
elif idade >= 16:
    print("Pode votar")
else:
    print("Menor de idade")
```

## Loops

```
python
```

```
# For loop
for i in range(5):
    print(f"Número: {i}")

# While loop
contador = 0
while contador < 5:
    print(contador)
    contador += 1

# List comprehension
quadrados = [x**2 for x in range(10)]
```

## Funções

```
python

def calcular_area_retangulo(largura, altura):
    """Calcula a área de um retângulo"""
    return largura * altura

def saudacao(nome, sobrenome=""):
    """Função com parâmetro opcional"""
    if sobrenome:
        return f"Olá, {nome} {sobrenome}!"
    return f"Olá, {nome}!"

# Função lambda
quadrado = lambda x: x**2
```

## 1.2 Controle de Versão com Git

### Comandos Básicos

```
bash
```

```

# Configuração inicial
git config --global user.name "Seu Nome"
git config --global user.email "seu.email@exemplo.com"

# Inicializando repositório
git init
git clone https://github.com/usuario/repositorio.git

# Comandos de trabalho diário
git add .          # Adiciona arquivos ao staging
git commit -m "Mensagem"  # Confirma alterações
git push origin main    # Envia para repositório remoto
git pull origin main    # Baixa alterações do remoto

# Gerenciamento de branches
git branch nova-feature  # Cria nova branch
git checkout nova-feature  # Muda para branch
git merge nova-feature    # Faz merge da branch

```

## Boas Práticas Git

- Commits pequenos e frequentes
- Mensagens descritivas
- Uso de .gitignore para arquivos desnecessários
- Branching strategy (Git Flow)

## 2. Bancos de Dados e Django Básico {#bancos-dados-django}

### 2.1 Fundamentos de Banco de Dados

#### SQL Básico

```
sql
```

```
-- Criação de tabela
CREATE TABLE usuarios (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    nome VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE,
    idade INTEGER
);

-- Inserção de dados
INSERT INTO usuarios (nome, email, idade)
VALUES ('João Silva', 'joao@email.com', 30);

-- Consultas
SELECT * FROM usuarios;
SELECT nome, email FROM usuarios WHERE idade > 25;

-- Atualização
UPDATE usuarios SET idade = 31 WHERE id = 1;

-- Exclusão
DELETE FROM usuarios WHERE id = 1;
```

## Relacionamentos

```
sql

-- Tabela de posts
CREATE TABLE posts (
    id INTEGER PRIMARY KEY,
    titulo VARCHAR(200),
    conteudo TEXT,
    usuario_id INTEGER,
    FOREIGN KEY (usuario_id) REFERENCES usuarios(id)
);

-- Join entre tabelas
SELECT u.nome, p.titulo
FROM usuarios u
JOIN posts p ON u.id = p.usuario_id;
```

## NoSQL com MongoDB

```
javascript
```

```

// Inserção de documento
db.usuarios.insertOne({
    nome: "Maria Silva",
    email: "maria@email.com",
    idade: 28,
    hobbies: ["leitura", "natação"]
});

// Consulta
db.usuarios.find({ idade: { $gt: 25 } });

// Atualização
db.usuarios.updateOne(
    { email: "maria@email.com" },
    { $set: { idade: 29 } }
);

```

## 2.2 Django Básico

### Configuração Inicial

```

bash

# Instalação
pip install django

# Criar projeto
django-admin startproject meu_projeto
cd meu_projeto

# Criar app
python manage.py startapp blog

# Migrations
python manage.py makemigrations
python manage.py migrate

# Servidor de desenvolvimento
python manage.py runserver

```

### Models (ORM)

```
python
```

```
# models.py
from django.db import models
from django.contrib.auth.models import User

class Post(models.Model):
    titulo = models.CharField(max_length=200)
    conteudo = models.TextField()
    autor = models.ForeignKey(User, on_delete=models.CASCADE)
    data_criacao = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.titulo

    class Meta:
        ordering = ['-data_criacao']
```

## Views

```
python

# views.py
from django.shortcuts import render, get_object_or_404
from django.http import HttpResponse
from .models import Post

def lista_posts(request):
    posts = Post.objects.all()
    return render(request, 'blog/lista.html', {'posts': posts})

def detalhe_post(request, post_id):
    post = get_object_or_404(Post, id=post_id)
    return render(request, 'blog/detalhe.html', {'post': post})

# Class-based views
from django.views.generic import ListView
class PostListView(ListView):
    model = Post
    template_name = 'blog/lista.html'
    context_object_name = 'posts'
```

## URLs

```
python
```

```
# urls.py do app
from django.urls import path
from . import views

urlpatterns = [
    path("", views.lista_posts, name='lista_posts'),
    path('post/<int:post_id>', views.detalhe_post, name='detalhe_post'),
]
```

```
# urls.py do projeto
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path("admin/", admin.site.urls),
    path("blog/", include("blog.urls")),
]
```

## Templates

html

```
<!-- templates/blog/base.html -->
<!DOCTYPE html>
<html>
<head>
    <title>Meu Blog</title>
</head>
<body>
    <nav>
        <a href="{% url 'lista_posts' %}">Home</a>
    </nav>

    <main>
        {% block content %}
        {% endblock %}
    </main>
</body>
</html>
```

```
<!-- templates/blog/lista.html -->
{% extends 'blog/base.html' %}

{% block content %}
    <h1>Posts do Blog</h1>
    {% for post in posts %}
        <article>
            <h2>
                <a href="{% url 'detalhe_post' post.id %}">{{ post.titulo }}</a>
            </h2>
            <p>Por {{ post.autor }} em {{ post.data_criacao }}</p>
        </article>
    {% endfor %}
    {% endblock %}
```

## 3. Python Avançado e Estruturas de Dados {#python-avancado-estruturas}

### 3.1 Programação Orientada a Objetos

#### Classes e Objetos

python

```

class Pessoa:
    def __init__(self, nome, idade):
        self.nome = nome
        self.idade = idade
        self._cpf = None # Atributo protegido

    def apresentar(self):
        return f"Olá, eu sou {self.nome} e tenho {self.idade} anos"

    def fazer_aniversario(self):
        self.idade += 1

    @property
    def cpf(self):
        return self._cpf

    @cpf.setter
    def cpf(self, valor):
        if len(valor) == 11:
            self._cpf = valor
        else:
            raise ValueError("CPF deve ter 11 dígitos")

    # Herança
    class Desenvolvedor(Pessoa):
        def __init__(self, nome, idade, linguagem):
            super().__init__(nome, idade)
            self.linguagem = linguagem

        def programar(self):
            return f"{self.nome} está programando em {self.linguagem}"

```

## Métodos Especiais

python

```

class ContaBancaria:
    def __init__(self, saldo=0):
        self.saldo = saldo

    def __str__(self):
        return f'Conta com saldo: R$ {self.saldo:.2f}'

    def __add__(self, valor):
        return ContaBancaria(self.saldo + valor)

    def __len__(self):
        return len(str(int(self.saldo)))

# Uso
conta = ContaBancaria(100)
print(conta) # Conta com saldo: R$ 100.00
nova_conta = conta + 50
print(len(conta)) # 3

```

## 3.2 Estruturas de Dados

### Pilha (Stack)

```

python

class Pilha:
    def __init__(self):
        self.items = []

    def empilhar(self, item):
        self.items.append(item)

    def desempilhar(self):
        if not self.vazia():
            return self.items.pop()
        raise IndexError("Pilha vazia")

    def topo(self):
        if not self.vazia():
            return self.items[-1]
        return None

    def vazia(self):
        return len(self.items) == 0

```

### Fila (Queue)

python

```
from collections import deque
```

```
class Fila:
```

```
    def __init__(self):  
        self.items = deque()
```

```
    def enfileirar(self, item):  
        self.items.append(item)
```

```
    def desenfileirar(self):
```

```
        if not self.vazia():  
            return self.items.popleft()  
        raise IndexError("Fila vazia")
```

```
    def vazia(self):
```

```
        return len(self.items) == 0
```

## Árvore Binária

python

```

class No:
    def __init__(self, valor):
        self.valor = valor
        self.esquerda = None
        self.direita = None

class ArvoreBinaria:
    def __init__(self):
        self.raiz = None

    def inserir(self, valor):
        if self.raiz is None:
            self.raiz = No(valor)
        else:
            self._inserir_recursivo(self.raiz, valor)

    def _inserir_recursivo(self, no, valor):
        if valor < no.valor:
            if no.esquerda is None:
                no.esquerda = No(valor)
            else:
                self._inserir_recursivo(no.esquerda, valor)
        else:
            if no.direita is None:
                no.direita = No(valor)
            else:
                self._inserir_recursivo(no.direita, valor)

```

### 3.3 Algoritmos de Ordenação

#### Bubble Sort

```

python

def bubble_sort(lista):
    n = len(lista)
    for i in range(n):
        for j in range(0, n - i - 1):
            if lista[j] > lista[j + 1]:
                lista[j], lista[j + 1] = lista[j + 1], lista[j]
    return lista

```

#### Quick Sort

```

python

```

```
def quick_sort(lista):
    if len(lista) <= 1:
        return lista

    pivot = lista[len(lista) // 2]
    esquerda = [x for x in lista if x < pivot]
    meio = [x for x in lista if x == pivot]
    direita = [x for x in lista if x > pivot]

    return quick_sort(esquerda) + meio + quick_sort(direita)
```

### 3.4 Complexidade Algorítmica (Big-O)

python

```
# O(1) - Constante
def acessar_elemento(lista, indice):
    return lista[indice]
```

```
# O(n) - Linear
def busca_linear(lista, item):
    for elemento in lista:
        if elemento == item:
            return True
    return False
```

```
# O(log n) - Logarítmica
def busca_binaria(lista, item):
    inicio = 0
    fim = len(lista) - 1

    while inicio <= fim:
        meio = (inicio + fim) // 2
        if lista[meio] == item:
            return meio
        elif lista[meio] < item:
            inicio = meio + 1
        else:
            fim = meio - 1

    return -1
```

```
# O(n2) - Quadrática
def bubble_sort(lista):
    n = len(lista)
    for i in range(n):
        for j in range(0, n - i - 1):
            if lista[j] > lista[j + 1]:
                lista[j], lista[j + 1] = lista[j + 1], lista[j]
```

## 3.5 Padrões de Projeto

### Singleton

```
python
```

```
class Singleton:  
    _instance = None  
  
    def __new__(cls):  
        if cls._instance is None:  
            cls._instance = super().__new__(cls)  
        return cls._instance
```

## Factory

python

```
class AnimalFactory:  
    @staticmethod  
    def criar_animal(tipo):  
        if tipo == "cachorro":  
            return Cachorro()  
        elif tipo == "gato":  
            return Gato()  
        else:  
            raise ValueError("Tipo de animal não suportado")
```

## Observer

python

```
class Observable:  
    def __init__(self):  
        self._observers = []  
  
    def adicionar_observer(self, observer):  
        self._observers.append(observer)  
  
    def notificar_observers(self, *args, **kwargs):  
        for observer in self._observers:  
            observer.update(*args, **kwargs)
```

## 4. Django Avançado, FastAPI e DevOps {#django-avancado-fastapi}

### 4.1 Django REST Framework

#### Serializers

python

```
# serializers.py
from rest_framework import serializers
from .models import Post

class PostSerializer(serializers.ModelSerializer):
    class Meta:
        model = Post
        fields = ['id', 'titulo', 'conteudo', 'autor', 'data_criacao']
        read_only_fields = ['autor', 'data_criacao']
```

## ViewSets

```
python

# views.py
from rest_framework import viewsets, permissions
from rest_framework.decorators import action
from rest_framework.response import Response
from .models import Post
from .serializers import PostSerializer

class PostViewSet(viewsets.ModelViewSet):
    queryset = Post.objects.all()
    serializer_class = PostSerializer
    permission_classes = [permissions.IsAuthenticated]

    def perform_create(self, serializer):
        serializer.save(autor=self.request.user)

    @action(detail=True, methods=['post'])
    def curtir(self, request, pk=None):
        post = self.get_object()
        # Lógica para curtir o post
        return Response({"status": "curtido"})
```

## Autenticação JWT

```
python
```

```
# settings.py
INSTALLED_APPS = [
    # ...
    'rest_framework',
    'rest_framework_simplejwt',
]

REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': (
        'rest_framework_simplejwt.authentication.JWTAuthentication',
    )
}

# urls.py
from rest_framework_simplejwt.views import (
    TokenObtainPairView,
    TokenRefreshView,
)

urlpatterns = [
    path('api/token/', TokenObtainPairView.as_view()),
    path('api/token/refresh/', TokenRefreshView.as_view()),
]
```

## 4.2 FastAPI

### Aplicação Básica

python

```
# main.py
from fastapi import FastAPI, HTTPException, Depends
from pydantic import BaseModel
from typing import List, Optional
import uvicorn

app = FastAPI(title="Minha API", version="1.0.0")

# Modelos Pydantic
class Usuario(BaseModel):
    id: Optional[int] = None
    nome: str
    email: str
    idade: int

class UsuarioCreate(BaseModel):
    nome: str
    email: str
    idade: int

# Simulação de banco de dados
usuarios_db = []

@app.get("/")
async def root():
    return {"message": "Hello World"}

@app.get("/usuarios/", response_model=List[Usuario])
async def listar_usuarios():
    return usuarios_db

@app.get("/usuarios/{user_id}", response_model=Usuario)
async def obter_usuario(user_id: int):
    for usuario in usuarios_db:
        if usuario.id == user_id:
            return usuario
    raise HTTPException(status_code=404, detail="Usuário não encontrado")

@app.post("/usuarios/", response_model=Usuario)
async def criar_usuario(usuario: UsuarioCreate):
    novo_usuario = Usuario(
        id=len(usuarios_db) + 1,
        **usuario.dict()
    )
```

```
usuarios_db.append(novo_usuario)
return novo_usuario
```

## Dependências e Autenticação

```
python

from fastapi import Depends, HTTPException, status
from fastapi.security import HTTPBearer, HTTPAuthorizationCredentials
import jwt

security = HTTPBearer()

def verificar_token(credentials: HTTPAuthorizationCredentials = Depends(security)):
    try:
        payload = jwt.decode(credentials.credentials, "SECRET_KEY", algorithms=["HS256"])
        return payload
    except jwt.InvalidTokenError:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Token inválido"
        )

@app.get("/perfil/")
async def obter_perfil(token_data = Depends(verificar_token)):
    return {"user_id": token_data["user_id"]}
```

## 4.3 Docker

### Dockerfile para Django

```
dockerfile

FROM python:3.11-slim

WORKDIR /app

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY .

EXPOSE 8000

CMD ["python", "manage.py", "runserver", "0.0.0.0:8000"]
```

## Docker Compose

```
yaml

# docker-compose.yml
version: '3.8'

services:
  web:
    build: .
    ports:
      - "8000:8000"
    volumes:
      - ./app
    depends_on:
      - db
    environment:
      - DATABASE_URL=postgresql://user:password@db:5432/myapp

  db:
    image: postgres:13
    volumes:
      - postgres_data:/var/lib/postgresql/data/
    environment:
      - POSTGRES_DB=myapp
      - POSTGRES_USER=user
      - POSTGRES_PASSWORD=password

  volumes:
    postgres_data:
```

## 4.4 Testes Automatizados

### Pytest

```
python
```

```

# test_models.py
import pytest
from django.test import TestCase
from django.contrib.auth.models import User
from blog.models import Post

@pytest.mark.django_db
class TestPost:
    def test_criar_post(self):
        user = User.objects.create_user(
            username='testuser',
            password='testpass'
        )
        post = Post.objects.create(
            titulo='Teste',
            conteudo='Conteúdo de teste',
            autor=user
        )
        assert post.titulo == 'Teste'
        assert post.autor == user

# test_api.py
from fastapi.testclient import TestClient
from main import app

client = TestClient(app)

def test_criar_usuario():
    response = client.post(
        "/usuarios/",
        json={"nome": "João", "email": "joao@test.com", "idade": 30}
    )
    assert response.status_code == 200
    assert response.json()["nome"] == "João"

```

## 5. Ciência de Dados e Machine Learning {#ciencia-dados-ml}

### 5.1 NumPy

#### Arrays Multidimensionais

python

```

import numpy as np

# Criação de arrays
arr1d = np.array([1, 2, 3, 4, 5])
arr2d = np.array([[1, 2, 3], [4, 5, 6]])
zeros = np.zeros((3, 3))
ones = np.ones((2, 4))
identidade = np.eye(3)

# Operações matemáticas
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])

soma = a + b
produto = a * b
produto_escalar = np.dot(a, b)

# Funções estatísticas
dados = np.array([1, 2, 3, 4, 5])
media = np.mean(dados)
desvio = np.std(dados)
maximo = np.max(dados)

```

## Indexação e Slicing

```

python

arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

# Acessando elementos
elemento = arr[1, 2] # 6
linha = arr[1, :] # [4, 5, 6]
coluna = arr[:, 1] # [2, 5, 8]

# Condições booleanas
maiores_que_5 = arr[arr > 5] # [6, 7, 8, 9]

```

## 5.2 Pandas

### DataFrames

```

python

```

```

import pandas as pd

# Criação de DataFrame
dados = {
    'nome': ['Ana', 'Bruno', 'Carlos'],
    'idade': [25, 30, 35],
    'cidade': ['SP', 'RJ', 'BH']
}
df = pd.DataFrame(dados)

# Leitura de arquivos
df_csv = pd.read_csv('dados.csv')
df_excel = pd.read_excel('dados.xlsx')

# Informações básicas
print(df.head())      # Primeiras 5 linhas
print(df.info())       # Informações gerais
print(df.describe())   # Estatísticas descritivas

```

## Manipulação de Dados

```

python

# Seleção
nomes = df['nome']
subset = df[['nome', 'idade']]
filtro = df[df['idade'] > 25]

# Adição de colunas
df['salario'] = [5000, 6000, 7000]
df['categoria'] = df['idade'].apply(lambda x: 'jovem' if x < 30 else 'adulta')

# Groupby
agrupado = df.groupby('categoria')['salario'].mean()

# Merge
df2 = pd.DataFrame({
    'nome': ['Ana', 'Bruno'],
    'departamento': ['TI', 'RH']
})
df_merged = pd.merge(df, df2, on='nome', how='left')

```

## Limpeza de Dados

```
python
```

```
# Valores ausentes
df.isnull().sum()      # Contar valores nulos
df.dropna()            # Remover linhas com valores nulos
df.fillna(0)           # Preencher valores nulos

# Duplicatas
df.duplicated().sum() # Contar duplicatas
df.drop_duplicates()   # Remover duplicatas

# Conversão de tipos
df['idade'] = df['idade'].astype(int)
df['data'] = pd.to_datetime(df['data'])
```

## 5.3 Matplotlib

### Gráficos Básicos

python

```
import matplotlib.pyplot as plt
```

# Gráfico de linha

```
x = [1, 2, 3, 4, 5]
```

```
y = [2, 4, 6, 8, 10]
```

```
plt.figure(figsize=(8, 6))
```

```
plt.plot(x, y, marker='o')
```

```
plt.title("Gráfico de Linha")
```

```
plt.xlabel("Eixo X")
```

```
plt.ylabel("Eixo Y")
```

```
plt.grid(True)
```

```
plt.show()
```

# Gráfico de barras

```
categorias = ['A', 'B', 'C', 'D']
```

```
valores = [23, 45, 56, 78]
```

```
plt.bar(categorias, valores)
```

```
plt.title("Gráfico de Barras")
```

```
plt.show()
```

# Histograma

```
dados = np.random.normal(0, 1, 1000)
```

```
plt.hist(dados, bins=30, alpha=0.7)
```

```
plt.title("Histograma")
```

```
plt.show()
```

## 5.4 Scikit-Learn

### Regressão Linear

python

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# Dados de exemplo
X = np.array([[1], [2], [3], [4], [5]])
y = np.array([2, 4, 6, 8, 10])

# Divisão treino/teste
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Treinamento
modelo = LinearRegression()
modelo.fit(X_train, y_train)

# Predição
y_pred = modelo.predict(X_test)

# Avaliação
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"MSE: {mse}")
print(f"R²: {r2}")
```

## Classificação

python

```

from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix

# Carregando dados
iris = load_iris()
X, y = iris.data, iris.target

# Divisão treino/teste
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Treinamento
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)

# Predição
y_pred = clf.predict(X_test)

# Avaliação
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))

```

## Validação Cruzada

```

python

from sklearn.model_selection import cross_val_score

# Validação cruzada
scores = cross_val_score(clf, X, y, cv=5)
print(f"Acurácia média: {scores.mean():.2f} (+/- {scores.std() * 2:.2f})")

```

# 6. Projetos Práticos e Preparação para Entrevistas {#projetos-entrevistas}

## 6.1 Projeto: API de Blog com Django REST

```
python
```

```
# models.py
from django.db import models
from django.contrib.auth.models import User

class Categoria(models.Model):
    nome = models.CharField(max_length=100)
    slug = models.SlugField(unique=True)

    def __str__(self):
        return self.nome

class Post(models.Model):
    titulo = models.CharField(max_length=200)
    slug = models.SlugField(unique=True)
    conteudo = models.TextField()
    resumo = models.TextField(max_length=300)
    autor = models.ForeignKey(User, on_delete=models.CASCADE)
    categoria = models.ForeignKey(Categoria, on_delete=models.CASCADE)
    publicado = models.BooleanField(default=False)
    data_criacao = models.DateTimeField(auto_now_add=True)
    data_atualizacao = models.DateTimeField(auto_now=True)

    class Meta:
        ordering = ['-data_criacao']

# serializers.py
from rest_framework import serializers
from .models import Post, Categoria

class CategoriaSerializer(serializers.ModelSerializer):
    class Meta:
        model = Categoria
        fields = '__all__'

class PostSerializer(serializers.ModelSerializer):
    autor_nome = serializers.CharField(source='autor.username', read_only=True)
    categoria_nome = serializers.CharField(source='categoria.nome', read_only=True)

    class Meta:
        model = Post
        fields = ['id', 'titulo', 'slug', 'conteudo', 'resumo', 'autor',
                  'autor_nome', 'categoria', 'categoria_nome', 'publicado',
                  'data_criacao', 'data_atualizacao']
        read_only_fields = ['autor', 'data_criacao', 'data_atualizacao']

# views.py
```

```

from rest_framework import viewsets, filters, permissions
from rest_framework.decorators import action
from rest_framework.response import Response
from django_filters.rest_framework import DjangoFilterBackend
from .models import Post, Categoria
from .serializers import PostSerializer, CategoriaSerializer

class PostViewSet(viewsets.ModelViewSet):
    queryset = Post.objects.filter(publicado=True)
    serializer_class = PostSerializer
    filter_backends = [DjangoFilterBackend, filters.SearchFilter, filters.OrderingFilter]
    filterset_fields = ['categoria', 'autor']
    search_fields = ['titulo', 'conteudo']
    ordering_fields = ['data_criacao', 'titulo']

    def get_permissions(self):
        if self.action in ['list', 'retrieve']:
            permission_classes = [permissions.AllowAny]
        else:
            permission_classes = [permissions.IsAuthenticated]
        return [permission() for permission in permission_classes]

    def perform_create(self, serializer):
        serializer.save(autor=self.request.user)

    @action(detail=False, methods=['get'])
    def meus_posts(self, request):
        posts = Post.objects.filter(autor=request.user)
        serializer = self.get_serializer(posts, many=True)
        return Response(serializer.data)

```

## 6.2 Projeto: Dashboard de Análise de Dados

python

```
# dashboard.py
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from flask import Flask, render_template, jsonify
import json
import plotly
import plotly.express as px

app = Flask(__name__)

class DashboardAnalytics:
    def __init__(self, dados_vendas):
        self.df = pd.read_csv(dados_vendas)
        self.preparar_dados()

    def preparar_dados(self):
        # Limpeza e preparação
        self.df['data'] = pd.to_datetime(self.df['data'])
        self.df['mes'] = self.df['data'].dt.month
        self.df['ano'] = self.df['data'].dt.year
        self.df = self.df.dropna()

    def vendas_por_mes(self):
        vendas_mensais = self.df.groupby('mes')['valor'].sum().reset_index()

        fig = px.line(vendas_mensais, x='mes', y='valor',
                      title='Vendas por Mês')
        graphJSON = json.dumps(fig, cls=plotly.utils.PlotlyJSONEncoder)
        return graphJSON

    def produtos_mais_vendidos(self, top_n=10):
        produtos = self.df.groupby('produto')['quantidade'].sum().nlargest(top_n)

        fig = px.bar(x=produtos.values, y=produtos.index,
                      orientation='h', title=f'Top {top_n} Produtos Mais Vendidos')
        graphJSON = json.dumps(fig, cls=plotly.utils.PlotlyJSONEncoder)
        return graphJSON

    def metricas_resumo(self):
        return {
            'total_vendas': self.df['valor'].sum(),
            'total_pedidos': len(self.df),
            'ticket_medio': self.df['valor'].mean(),
            'produto_mais_vendido': self.df.groupby('produto')['quantidade'].sum().idxmax()
        }
```

```
@app.route('/')
def dashboard():
    analytics = DashboardAnalytics('dados_vendas.csv')
    metricas = analytics.metricas_resumo()
    return render_template('dashboard.html', metricas=metricas)

@app.route('/api/vendas-mes')
def api_vendas_mes():
    analytics = DashboardAnalytics('dados_vendas.csv')
    return analytics.vendas_por_mes()

@app.route('/api/produtos-top')
def api_produtos_top():
    analytics = DashboardAnalytics('dados_vendas.csv')
    return analytics.produtos_mais_vendidos()
```

## 6.3 Projeto: Sistema de Recomendação Simples

python

```

# recomendador.py
import pandas as pd
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import TfidfVectorizer

class SistemaRecomendacao:
    def __init__(self):
        self.df_filmes = None
        self.df_ratings = None
        self.tfidf_matrix = None
        self.cosine_sim = None

    def carregar_dados(self, arquivo_filmes, arquivo_ratings):
        self.df_filmes = pd.read_csv(arquivo_filmes)
        self.df_ratings = pd.read_csv(arquivo_ratings)
        self.preparar_modelo()

    def preparar_modelo(self):
        # Vetorização TF-IDF dos gêneros
        tfidf = TfidfVectorizer(stop_words='english')
        self.tfidf_matrix = tfidf.fit_transform(self.df_filmes['genres'])

        # Matriz de similaridade
        self.cosine_sim = cosine_similarity(self.tfidf_matrix)

    def recomendar_por_conteudo(self, titulo_filme, num_recomendacoes=5):
        # Encontrar índice do filme
        idx = self.df_filmes[self.df_filmes['title'] == titulo_filme].index[0]

        # Calcular similaridades
        sim_scores = list(enumerate(self.cosine_sim[idx]))
        sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

        # Obter índices dos filmes mais similares
        movie_indices = [i[0] for i in sim_scores[1:num_recomendacoes+1]]

        return self.df_filmes['title'].iloc[movie_indices].tolist()

    def recomendar_por_usuario(self, user_id, num_recomendacoes=5):
        # Filtrar ratings do usuário
        user_ratings = self.df_ratings[self.df_ratings['userId'] == user_id]

        if user_ratings.empty:
            return self.df_filmes.nlargest(num_recomendacoes, 'rating')['title'].tolist()

```

```

# Filmes já avaliados pelo usuário
filmes_avaliados = user_ratings['movied'].tolist()

# Encontrar usuários similares
user_item_matrix = self.df_ratings.pivot_table(
    index='userId', columns='movied', values='rating'
).fillna(0)

user_similarity = cosine_similarity(user_item_matrix)
user_idx = list(user_item_matrix.index).index(user_id)

similar_users = list(enumerate(user_similarity[user_idx]))
similar_users = sorted(similar_users, key=lambda x: x[1], reverse=True)[1:6]

# Recomendar filmes baseado em usuários similares
recomendacoes = set()
for similar_user_idx, _ in similar_users:
    similar_user_id = user_item_matrix.index[similar_user_idx]
    similar_user_movies = self.df_ratings[
        (self.df_ratings['userId'] == similar_user_id) &
        (self.df_ratings['rating'] >= 4.0) &
        (~self.df_ratings['movied'].isin(filmes_avaliados))
    ]['movied'].tolist()

    recomendacoes.update(similar_user_movies[:num_recomendacoes])

filme_titles = []
for movie_id in list(recomendacoes)[:num_recomendacoes]:
    title = self.df_filmes[self.df_filmes['movied'] == movie_id]['title']
    if not title.empty:
        filme_titles.append(title.iloc[0])

return filme_titles

# Uso do sistema
recomendador = SistemaRecomendacao()
recomendador.carregar_dados('movies.csv', 'ratings.csv')

# Recomendações por conteúdo
filmes_similares = recomendador.recomendar_por_conteudo('Toy Story (1995)')

# Recomendações por usuário
recomendacoes_usuario = recomendador.recomendar_por_usuario(1)

```

## 6.4 Preparação para Entrevistas

## Algoritmos Essenciais

### Busca em Profundidade (DFS)

```
python

def dfs_recursivo(grafo, vertice, visitados=None):
    if visitados is None:
        visitados = set()

    visitados.add(vertice)
    print(vertice)

    for vizinho in grafo[vertice]:
        if vizinho not in visitados:
            dfs_recursivo(grafo, vizinho, visitados)

    return visitados

def dfs_iterativo(grafo, inicio):
    visitados = set()
    pilha = [inicio]

    while pilha:
        vertice = pilha.pop()
        if vertice not in visitados:
            visitados.add(vertice)
            print(vertice)
            pilha.extend(reversed(grafo[vertice]))

    return visitados
```

### Busca em Largura (BFS)

```
python
```

```
from collections import deque
```

```
def bfs(grafo, inicio):
    visitados = set()
    fila = deque([inicio])
    visitados.add(inicio)

    while fila:
        vertice = fila.popleft()
        print(vertice)

        for vizinho in grafo[vertice]:
            if vizinho not in visitados:
                visitados.add(vizinho)
                fila.append(vizinho)

    return visitados
```

## Problemas Clássicos

python

```

# Fibonacci
def fibonacci(n):
    if n <= 1:
        return n
    return fibonacci(n-1) + fibonacci(n-2)

# Fibonacci com memoização
def fibonacci_memo(n, memo={}):
    if n in memo:
        return memo[n]
    if n <= 1:
        return n

    memo[n] = fibonacci_memo(n-1, memo) + fibonacci_memo(n-2, memo)
    return memo[n]

# Palíndromo
def eh_palindromo(s):
    s = s.lower().replace(' ', '')
    return s == s[::-1]

# Dois soma (Two Sum)
def dois_soma(nums, target):
    num_map = {}
    for i, num in enumerate(nums):
        complement = target - num
        if complement in num_map:
            return [num_map[complement], i]
        num_map[num] = i
    return []

```

## Perguntas Técnicas Comuns

### 1. Diferença entre lista e tupla em Python

python

```

# Lista - mutável
lista = [1, 2, 3]
lista.append(4) # OK

# Tupla - imutável
tupla = (1, 2, 3)
# tupla.append(4) # Erro!

# Quando usar cada uma:
# Lista: quando precisar modificar
# Tupla: dados que não mudam, chaves de dicionário

```

## 2. Geradores vs Listas

```

python

# Lista - carrega tudo na memória
lista_quadrados = [x**2 for x in range(1000000)]

# Gerador - lazy evaluation
def gerador_quadrados(n):
    for i in range(n):
        yield i**2

# Uso de memória muito menor
gen = gerador_quadrados(1000000)

```

## 3. Decorators

```

python

def timing_decorator(func):
    import time
    def wrapper(*args, **kwargs):
        start = time.time()
        result = func(*args, **kwargs)
        end = time.time()
        print(f"{func.__name__} levou {end - start:.2f} segundos")
        return result
    return wrapper

@timing_decorator
def operacao_lenta():
    time.sleep(1)
    return "Concluída"

```

## 7. Temas Avançados para Desenvolvedor Júnior {#temas-avancados}

### 7.1 Containerização e DevOps

#### Docker Avançado

```
dockerfile
```

```
# Multi-stage build
```

```
FROM python:3.11-slim as builder
```

```
WORKDIR /app
```

```
COPY requirements.txt .
```

```
RUN pip install --user -r requirements.txt
```

```
FROM python:3.11-slim
```

```
WORKDIR /app
```

```
COPY --from=builder /root/.local /root/.local
```

```
COPY ..
```

```
# Criar usuário não-root
```

```
RUN useradd --create-home --shell /bin/bash app
```

```
USER app
```

```
ENV PATH=/root/.local/bin:$PATH
```

```
CMD ["gunicorn", "--bind", "0.0.0.0:8000", "myapp.wsgi:application"]
```

#### Kubernetes Básico

```
yaml
```

```
# deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: webapp-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: webapp
  template:
    metadata:
      labels:
        app: webapp
    spec:
      containers:
        - name: webapp
          image: myapp:latest
          ports:
            - containerPort: 8000
          env:
            - name: DATABASE_URL
              valueFrom:
                secretKeyRef:
                  name: db-secret
                  key: url
```

```
---
apiVersion: v1
kind: Service
metadata:
  name: webapp-service
spec:
  selector:
    app: webapp
  ports:
    - port: 80
      targetPort: 8000
  type: LoadBalancer
```

## CI/CD com GitHub Actions

```
yaml
```

```
# .github/workflows/deploy.yml
name: Deploy to Production

on:
push:
  branches: [ main ]

jobs:
test:
  runs-on: ubuntu-latest

  steps:
    - uses: actions/checkout@v2

    - name: Set up Python
      uses: actions/setup-python@v2
      with:
        python-version: 3.11

    - name: Install dependencies
      run: |
        pip install -r requirements.txt
        pip install pytest

    - name: Run tests
      run: pytest

    - name: Run linting
      run: |
        pip install flake8
        flake8 . --count --select=E9,F63,F7,F82 --show-source --statistics

deploy:
  needs: test
  runs-on: ubuntu-latest

  steps:
    - uses: actions/checkout@v2

    - name: Build Docker image
      run: docker build -t myapp:${{ github.sha }} .

    - name: Deploy to production
      run: |
```

```
# Comandos de deploy (ex: docker push, kubectl apply)
echo "Deploying to production..."
```

## 7.2 Mensageria e Filas

### Celery com Redis

```
python
```

```

# celery_app.py
from celery import Celery

app = Celery('myapp',
             broker='redis://localhost:6379/0',
             backend='redis://localhost:6379/0')

@app.task
def enviar_email(destinatario, assunto, corpo):
    # Simulação de envio de email
    import time
    time.sleep(2) # Simula operação demorada
    print(f"Email enviado para {destinatario}")
    return f"Email enviado com sucesso para {destinatario}"

@app.task
def processar_imagem(caminho_imagem):
    from PIL import Image

    # Redimensionar imagem
    with Image.open(caminho_imagem) as img:
        img.thumbnail((300, 300))
        novo_caminho = f"thumb_{caminho_imagem}"
        img.save(novo_caminho)

    return novo_caminho

# views.py (Django)
from django.http import JsonResponse
from .celery_app import enviar_email, processar_imagem

def enviar_email_async(request):
    if request.method == 'POST':
        email = request.POST.get('email')
        assunto = request.POST.get("assunto")
        corpo = request.POST.get('corpo')

        # Executa tarefa assíncrona
        task = enviar_email.delay(email, assunto, corpo)

    return JsonResponse({
        'task_id': task.id,
        'status': 'processando'
    })

```

python

```
import pika
import json

class MessageQueue:
    def __init__(self, host='localhost'):
        self.connection = pika.BlockingConnection(
            pika.ConnectionParameters(host)
        )
        self.channel = self.connection.channel()

    def criar_fila(self, nome_fila):
        self.channel.queue_declare(queue=nome_fila, durable=True)

    def publicar_mensagem(self, fila, mensagem):
        self.channel.basic_publish(
            exchange='',
            routing_key=fila,
            body=json.dumps(mensagem),
            properties=pika.BasicProperties(
                delivery_mode=2, # Torna a mensagem persistente
            )
        )

    def consumir_mensagens(self, fila, callback):
        def wrapper(ch, method, properties, body):
            mensagem = json.loads(body)
            try:
                callback(mensagem)
                ch.basic_ack(delivery_tag=method.delivery_tag)
            except Exception as e:
                print(f"Erro ao processar mensagem: {e}")
                ch.basic_nack(delivery_tag=method.delivery_tag, requeue=False)

        self.channel.basic_consume(queue=fila, on_message_callback=wrapper)
        self.channel.start_consuming()

    # Uso
    mq = MessageQueue()
    mq.criar_fila('processar_pedidos')

    # Publisher
    mq.publicar_mensagem('processar_pedidos', {
        'pedido_id': 123,
        'usuario_id': 456,
        'valor': 99.90
    })
```

```

# Consumer

def processar_pedido(mensagem):
    print(f"Processando pedido {mensagem['pedido_id']}")
    # Lógica de processamento...

mq.consumir_mensagens('processar_pedidos', processar_pedido)

```

## 7.3 Design de APIs

### Versionamento de API

```

python

# FastAPI com versionamento
from fastapi import FastAPI, Header
from typing import Optional

app = FastAPI()

# Versionamento por header
@app.get("/usuarios/")
async def listar_usuarios(api_version: Optional[str] = Header(None)):
    if api_version == "v2":
        return {"usuarios": [], "version": "2.0", "metadata": {}}
    else: # Default v1
        return {"usuarios": []}

# Versionamento por URL
@app.get("/v1/usuarios/")
async def listar_usuarios_v1():
    return {"usuarios": []}

@app.get("/v2/usuarios/")
async def listar_usuarios_v2():
    return {
        "usuarios": [],
        "metadata": {"total": 0, "page": 1},
        "version": "2.0"
    }

```

### GraphQL com Strawberry

```

python

```

```

import strawberry
from typing import List, Optional

@strawberry.type
class Usuario:
    id: int
    nome: str
    email: str
    posts: Optional[List['Post']] = None

@strawberry.type
class Post:
    id: int
    titulo: str
    conteudo: str
    autor: Usuario

@strawberry.type
class Query:
    @strawberry.field
    def usuarios(self) -> List[Usuario]:
        # Buscar usuários do banco
        return []

    @strawberry.field
    def usuario(self, id: int) -> Optional[Usuario]:
        # Buscar usuário específico
        return None

@strawberry.type
class Mutation:
    @strawberry.mutation
    def criar_usuario(self, nome: str, email: str) -> Usuario:
        # Criar usuário
        return Usuario(id=1, nome=nome, email=email)

schema = strawberry.Schema(query=Query, mutation=Mutation)

# FastAPI + GraphQL
from strawberry.fastapi import GraphQLRouter

graphql_app = GraphQLRouter(schema)
app.include_router(graphql_app, prefix="/graphql")

```

## 7.4 Padrões de Arquitetura

## Arquitetura Hexagonal (Ports and Adapters)

python

```
# domain/entities.py
from dataclasses import dataclass
from typing import List

@dataclass
class Usuario:
    id: int
    nome: str
    email: str

    def __post_init__(self):
        if not self.email or '@' not in self.email:
            raise ValueError("Email inválido")

# domain/repositories.py
from abc import ABC, abstractmethod

class UsuarioRepository(ABC):
    @abstractmethod
    def salvar(self, usuario: Usuario) -> Usuario:
        pass

    @abstractmethod
    def buscar_por_id(self, id: int) -> Usuario:
        pass

    @abstractmethod
    def listar.todos(self) -> List[Usuario]:
        pass

# application/services.py
class UsuarioService:
    def __init__(self, repository: UsuarioRepository):
        self.repository = repository

    def criar_usuario(self, nome: str, email: str) -> Usuario:
        # Regras de negócio
        if len(nome) < 2:
            raise ValueError("Nome muito curto")

        usuario = Usuario(id=0, nome=nome, email=email)
        return self.repository.salvar(usuario)

# infrastructure/repositories.py
class SqlUsuarioRepository(UsuarioRepository):
    def __init__(self, db_connection):
```

```

self.db = db_connection

def salvar(self, usuario: Usuario) -> Usuario:
    # Implementação SQL
    cursor = self.db.cursor()
    cursor.execute(
        "INSERT INTO usuarios (nome, email) VALUES (?, ?)",
        (usuario.nome, usuario.email)
    )
    usuario.id = cursor.lastrowid
    return usuario

def buscar_por_id(self, id: int) -> Usuario:
    # Implementação SQL
    pass

# api/controllers.py
from fastapi import FastAPI, Depends

def get_usuario_service() -> UsuarioService:
    db = get_database_connection()
    repository = SqlUsuarioRepository(db)
    return UsuarioService(repository)

@app.post("/usuarios/")
def criar_usuario(
    nome: str,
    email: str,
    service: UsuarioService = Depends(get_usuario_service)
):
    return service.criar_usuario(nome, email)

```

## CQRS (Command Query Responsibility Segregation)

python

```
# commands.py
from dataclasses import dataclass

@dataclass
class CriarUsuarioCommand:
    nome: str
    email: str

@dataclass
class AtualizarUsuarioCommand:
    id: int
    nome: str
    email: str

# queries.py
@dataclass
class BuscarUsuarioQuery:
    id: int

@dataclass
class ListarUsuariosQuery:
    page: int = 1
    size: int = 10

# handlers.py
class UsuarioCommandHandler:
    def __init__(self, repository: UsuarioRepository):
        self.repository = repository

    def handle_criar_usuario(self, command: CriarUsuarioCommand):
        usuario = Usuario(0, command.nome, command.email)
        return self.repository.salvar(usuario)

    def handle_atualizar_usuario(self, command: AtualizarUsuarioCommand):
        usuario = self.repository.buscar_por_id(command.id)
        usuario.nome = command.nome
        usuario.email = command.email
        return self.repository.salvar(usuario)

class UsuarioQueryHandler:
    def __init__(self, read_repository):
        self.repository = read_repository

    def handle_buscar_usuario(self, query: BuscarUsuarioQuery):
        return self.repository.buscar_por_id(query.id)
```

```
def handle_listar_usuarios(self, query: ListarUsuariosQuery):
    return self.repository.listar_paginado(query.page, query.size)
```

## 7.5 Testes Avançados

### Test-Driven Development (TDD)

python

```
# test_usuario_service.py
import pytest
from unittest.mock import Mock
from domain.entities import Usuario
from application.services import UsuarioService

class TestUsuarioService:
    def setup_method(self):
        self.mock_repository = Mock()
        self.service = UsuarioService(self.mock_repository)

    def test_criar_usuario_com_dados_validos(self):
        # Arrange
        nome = "João Silva"
        email = "joao@email.com"
        usuario_esperado = Usuario(1, nome, email)
        self.mock_repository.salvar.return_value = usuario_esperado

        # Act
        resultado = self.service.criar_usuario(nome, email)

        # Assert
        assert resultado.nome == nome
        assert resultado.email == email
        self.mock_repository.salvar.assert_called_once()

    def test_criar_usuario_com_nome_curto_deve_falhar(self):
        # Arrange
        nome = "A"
        email = "a@email.com"

        # Act & Assert
        with pytest.raises(ValueError, match="Nome muito curto"):
            self.service.criar_usuario(nome, email)

        self.mock_repository.salvar.assert_not_called()

# Testes de integração
@pytest.mark.integration
class TestUsuarioIntegration:
    def test_criar_e_buscar_usuario(self, db_session):
        # Setup
        repository = SqlUsuarioRepository(db_session)
        service = UsuarioService(repository)

        # Criar usuário
```

```

usuario = service.criar_usuario("Test User", "test@email.com")

# Buscar usuário
usuario_encontrado = repository.buscar_por_id(usuario.id)

assert usuario_encontrado.nome == "Test User"
assert usuario_encontrado.email == "test@email.com"

```

## Testes de Performance

```

python

import time
import pytest
from concurrent.futures import ThreadPoolExecutor

def test_performance_criar_usuarios():
    service = UsuarioService(Mock())

    start_time = time.time()

    for i in range(1000):
        service.criar_usuario(f"User {i}", f"user{i}@email.com")

    end_time = time.time()
    execution_time = end_time - start_time

    # Deve criar 1000 usuários em menos de 1 segundo
    assert execution_time < 1.0

def test_concurrent_user_creation():
    service = UsuarioService(ThreadSafeRepository())

    def criar_usuario(index):
        return service.criar_usuario(f"User {index}", f"user{index}@email.com")

    with ThreadPoolExecutor(max_workers=10) as executor:
        futures = [executor.submit(criar_usuario, i) for i in range(100)]
        results = [future.result() for future in futures]

    # Todos os usuários devem ser criados com sucesso
    assert len(results) == 100
    assert all(user.id is not None for user in results)

```

## 8. Recursos e Referências {#recursos-referencias}

## 8.1 Documentação Oficial

### Python

- [Documentação Python](#)
- [PEP 8 - Style Guide](#)
- [Python Enhancement Proposals](#)

### Frameworks

- [Django Documentation](#)
- [FastAPI Documentation](#)
- [Flask Documentation](#)

### Banco de Dados

- [PostgreSQL Documentation](#)
- [MongoDB Manual](#)
- [Redis Documentation](#)

### Data Science

- [NumPy Documentation](#)
- [Pandas Documentation](#)
- [Scikit-learn Documentation](#)
- [Matplotlib Documentation](#)

## 8.2 Cursos Gratuitos

### Python Básico

- [Python para Zumbis \(PUC-Rio\)](#)
- [Curso em Vídeo - Python](#)
- [freeCodeCamp Python Course](#)

### Web Development

- [Django Girls Tutorial](#)
- [FastAPI do Zero](#)
- [Full Stack Python](#)

### Data Science

- [Data Science Academy](#)

- [Kaggle Learn](#)
- [Coursera - Machine Learning](#)

## 8.3 Livros Recomendados

### Gratuitos Online

- [Automate the Boring Stuff with Python](#)
- [\[Python Data Science Handbook\]](#)