

Python Health: Inovação em Saúde com Django e Flask

Python

Introdução

Python é uma linguagem de programação poderosa e versátil, amplamente utilizada em diversas áreas, incluindo o desenvolvimento web. Entre os frameworks mais populares para criar aplicações web com Python estão Django e Flask. Este ebook oferece uma visão geral desses frameworks, com exemplos de código simples para ajudar você a começar a construir soluções digitais inovadoras para a área de saúde.



Capítulo 1

Python

Python é uma linguagem de programação de alto nível, interpretada e de propósito geral, conhecida por sua simplicidade e legibilidade.

Python

Linguagem de alto nível

Sintaxe Simples e Legível

Python é famoso por sua sintaxe clara, que facilita a leitura e a escrita do código. O uso de indentação para delimitar blocos de código torna-o intuitivo.

```
python

def greet(name):
    print(f"Hello, {name}!")
```

Tipagem Dinâmica

Python é dinamicamente tipada, o que significa que você não precisa declarar explicitamente o tipo das variáveis.

```
python

x = 10
x = "Hello"
```

Suporte a Paradigmas de Programação Diversos

Python suporta múltiplos paradigmas de programação, incluindo programação orientada a objetos, funcional e procedimental.

```
python

# Orientação a objetos
class Patient:
    def __init__(self, name, birth_date):
        self.name = name
        self.birth_date = birth_date

# Programação funcional
def square(x):
    return x * x

squares = list(map(square, [1, 2, 3, 4]))
```

Bibliotecas Padrão Abrangentes

A biblioteca padrão do Python é extensa, incluindo módulos para manipulação de strings, arquivos, redes, e muito mais.

```
python

import os

files = os.listdir('.')
print(files)
```


Gestão de Pacotes e Módulos

Python facilita a organização do código em módulos e pacotes, permitindo a reutilização e manutenção eficiente.

```
python

# mymodule.py
def add(a, b):
    return a + b

# main.py
from mymodule import add

result = add(3, 4)
print(result)
```

Interpretação e Portabilidade

Python é interpretado, o que significa que o código é executado linha a linha, facilitando a depuração e a portabilidade entre diferentes sistemas operacionais.

```
bash

python script.py
```

Gerenciamento de Memória Automático

Python possui coleta de lixo automática, que gerencia a alocação e liberação de memória.

```
python

# Exemplo simples de uso de memória
a = [1, 2, 3]
b = a
del a # Memória será liberada automaticamente quando não houver referências
```

Integração com Outras Linguagens e Tecnologias

Python pode interagir com código escrito em outras linguagens como C, C++, e Java, através de APIs e bibliotecas específicas.

```
python

import ctypes

# Chamada de função em uma biblioteca C
libc = ctypes.CDLL("libc.so.6")
libc.printf(b"Hello, World!\n")
```

Amplo Suporte de Comunidade e Ecossistema de Bibliotecas

Python tem uma vasta comunidade e um ecossistema rico de bibliotecas e frameworks para diversas finalidades, como Django para desenvolvimento web, NumPy para computação científica, e Pandas para análise de dados.

```
python

import numpy as np

array = np.array([1, 2, 3, 4])
print(array * 2)
```

Python é uma linguagem de programação versátil e poderosa, adequada para uma ampla gama de aplicações, desde scripts simples até sistemas complexos. Suas características técnicas, como sintaxe clara, tipagem dinâmica, suporte a múltiplos paradigmas de programação, biblioteca padrão abrangente, e uma vasta comunidade de suporte, fazem do Python uma escolha excelente para desenvolvedores de todos os níveis.

Capítulo 2

Django

Django é um framework web de alto nível que simplifica o desenvolvimento de aplicações web robustas

Django

O Framework Completo

Administração Automática

Django vem com uma interface administrativa pronta para uso, permitindo gerenciar modelos e dados sem necessidade de desenvolvimento adicional.

```
python
```

```
# Registering a model in the admin interface
from django.contrib import admin
from .models import Patient

admin.site.register(Patient)
```

Sistema de Templates

O sistema de templates do Django permite a criação de páginas dinâmicas e reutilizáveis.

html

```
<!-- Example template (templates/patient_list.html) -->
<!DOCTYPE html>
<html>
<head>
    <title>Patient List</title>
</head>
<body>
    <h1>Patient List</h1>
    <ul>
        {% for patient in patients %}
        <li>{{ patient.name }} - {{ patient.birth_date }}</li>
        {% endfor %}
    </ul>
</body>
</html>
```

ORM (Object-Relational Mapping)

O ORM do Django mapeia modelos de banco de dados a classes Python, facilitando o trabalho com bancos de dados sem escrever SQL.

```
python

# Defining a model
from django.db import models

class Patient(models.Model):
    name = models.CharField(max_length=100)
    birth_date = models.DateField()
    medical_record = models.TextField()
```

Sistema de URLs

Django oferece um sistema de roteamento de URLs que mapeia URLs para views específicas.

```
python

# Defining URL patterns
from django.urls import path
from . import views

urlpatterns = [
    path('patients/', views.patient_list),
]
```

Views e Lógica de Negócio

As views em Django processam requisições, executam a lógica de negócio e retornam respostas.

python

```
# Example view
from django.shortcuts import render
from .models import Patient

def patient_list(request):
    patients = Patient.objects.all()
    return render(request, 'patients/patient_list.html', {'patients': patients})
```

Formulários

Django facilita a criação e manipulação de formulários HTML com validação automática.

python

```
# Example form
from django import forms

class PatientForm(forms.ModelForm):
    class Meta:
        model = Patient
        fields = ['name', 'birth_date', 'medical_record']
```

Sistema de Autenticação

Inclui ferramentas integradas para autenticação de usuários, gerenciamento de permissões e grupos de usuários.

```
python
```

```
# Using built-in authentication views
from django.contrib.auth import views as auth_views

urlpatterns = [
    path('login/', auth_views.LoginView.as_view(), name='login'),
    path('logout/', auth_views.LogoutView.as_view(), name='logout'),
]
```

Django fornece uma estrutura poderosa e abrangente para desenvolvimento web, incluindo administração automática, sistema de templates, ORM, roteamento de URLs, views, formulários e autenticação. Essas funcionalidades permitem criar aplicações complexas de forma rápida e eficiente.

Flask

Flask é um micro framework que oferece a flexibilidade de escolher apenas os componentes necessários para sua aplicação, tornando-o ideal para projetos menores ou mais específicos.

Flask

O Micro Framework Flexível

Flask é um micro framework web para Python, conhecido por sua simplicidade e flexibilidade. Aqui estão algumas de suas principais funções técnicas.

Rotas e Roteamento de URLs

Flask usa decoradores para mapear URLs a funções Python, permitindo a criação de rotas simples e claras.

```
python
```

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def home():
    return 'Welcome to the Healthcare App'

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

Templates e Jinja2

Flask utiliza o mecanismo de templates Jinja2, que permite a criação de páginas HTML dinâmicas.

html

```
<!-- Example template (templates/patient_list.html) -->
<!DOCTYPE html>
<html>
<head>
    <title>Patient List</title>
</head>
<body>
    <h1>Patient List</h1>
    <ul>
        {% for patient in patients %}
        <li>{{ patient.name }} - {{ patient.birth_date }}</li>
        {% endfor %}
    </ul>
</body>
</html>
```

python

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/patients')
def patient_list():
    patients = [
        {'name': 'John Doe', 'birth_date': '1990-01-01'},
        {'name': 'Jane Smith', 'birth_date': '1985-05-12'}
    ]
    return render_template('patient_list.html', patients=patients)

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

Manipulação de Requisições e Respostas

Flask facilita o acesso a dados de requisições e a criação de respostas HTTP personalizadas.

python

```
from flask import request, jsonify

@app.route('/api/patient', methods=['POST'])
def add_patient():
    data = request.json
    # Process the data (e.g., save to database)
    return jsonify({'message': 'Patient added successfully'}), 201
```

Blueprints para Modularidade

Blueprints permitem organizar a aplicação em componentes menores e reutilizáveis.

python

```
from flask import Blueprint

patients_bp = Blueprint('patients', __name__)

@patients_bp.route('/patients')
def patient_list():
    return 'Patient List'

app.register_blueprint(patients_bp)
```

Extensões

Flask pode ser estendido com várias extensões para adicionar funcionalidades, como autenticação, gerenciamento de formulários e interações com bancos de dados.

```
python

from flask_sqlalchemy import SQLAlchemy

app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///patients.db'
db = SQLAlchemy(app)

class Patient(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100))
    birth_date = db.Column(db.Date)

# Create the database and tables
db.create_all()
```

Sistema de Formulários

Flask-WTF integra-se com WTForms para facilitar a criação e validação de formulários HTML.

```
python

from flask_wtf import FlaskForm
from wtforms import StringField, DateField, SubmitField

class PatientForm(FlaskForm):
    name = StringField('Name')
    birth_date = DateField('Birth Date')
    submit = SubmitField('Submit')
```

Autenticação e Autorização

Flask-Login é uma extensão que gerencia sessões de usuários e autenticação.

```
python

from flask_login import LoginManager, UserMixin, login_user

login_manager = LoginManager(app)

class User(UserMixin):
    # User model implementation

@login_manager.user_loader
def load_user(user_id):
    return User.get(user_id) # Retrieve user by ID

# In your login route
login_user(user)
```

Flask é um framework leve e flexível que oferece funcionalidades essenciais para o desenvolvimento web, incluindo roteamento de URLs, templates Jinja2, manipulação de requisições e respostas, modularidade com Blueprints, extensões para funcionalidades adicionais, suporte a formulários e autenticação. Essas características tornam o Flask ideal para criar aplicações web personalizadas e escaláveis.

Capítulo 4

Desenvolvendo o conteúdo

Agora iremos aplicar um pouco do que aprendemos, com a mesma didática utilizada nos slides anteriores.

Criando um Projeto Django

“healthcare_project”:

```
bash

// django-admin startproject healthcare_project
cd healthcare_project
python manage.py startapp patients
```

Modelando Dados com Django

Vamos definir um modelo simples para armazenar informações de pacientes:

```
python

# patients/models.py
from django.db import models

class Patient(models.Model):
    name = models.CharField(max_length=100)
    birth_date = models.DateField()
    medical_record = models.TextField()

    def __str__(self):
        return self.name
```

Exibindo Dados no Django Admin

O Django Admin é uma ferramenta poderosa para gerenciar dados. Para registrar o modelo Patient:

```
python

# patients/admin.py
from django.contrib import admin
from .models import Patient

admin.site.register(Patient)
```

Exibindo Pacientes em uma Página Web

Vamos criar uma view e um template para listar os pacientes:

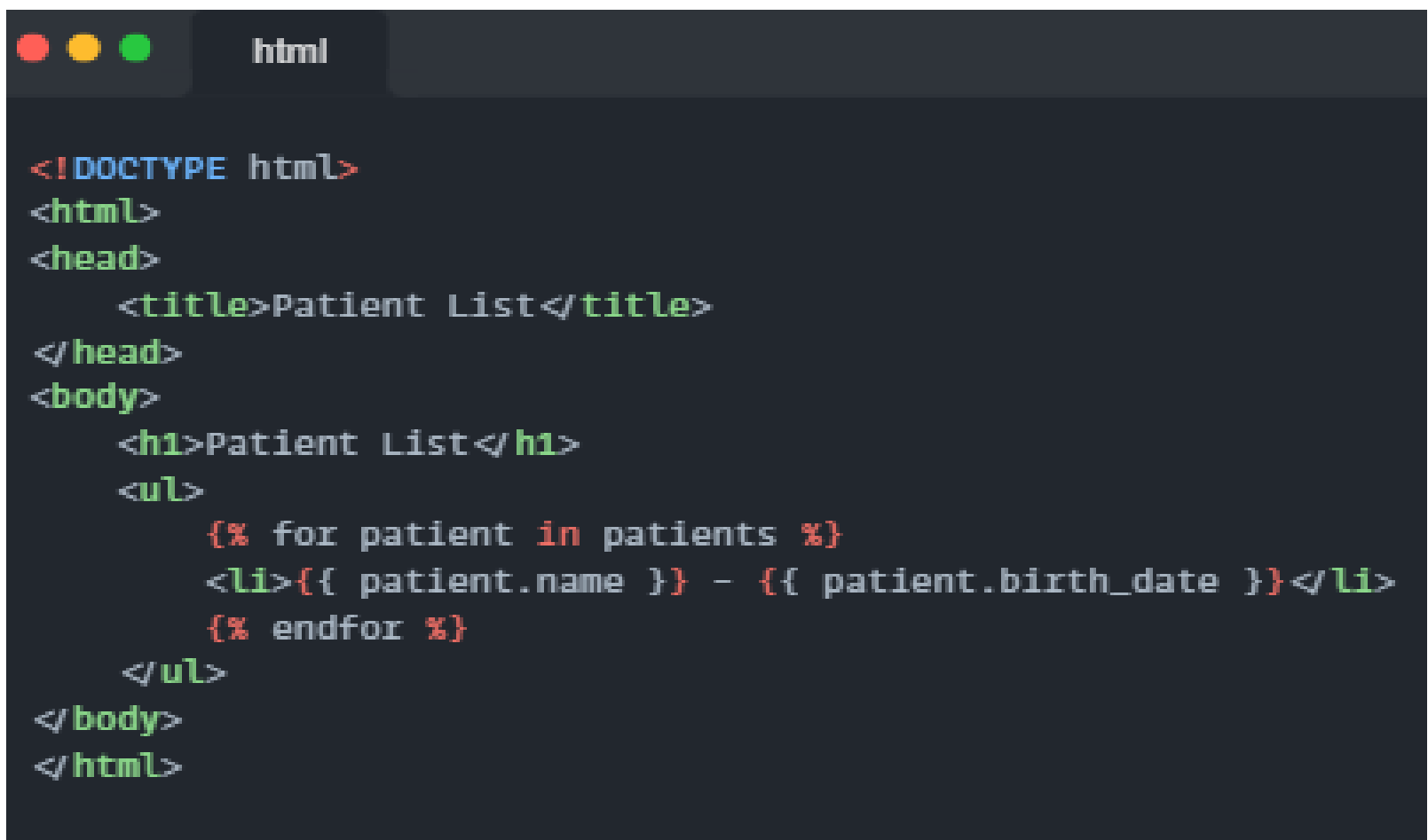
```
python

# patients/views.py
from django.shortcuts import render
from .models import Patient

def patient_list(request):
    patients = Patient.objects.all()
    return render(request, 'patients/patient_list.html', {'patients': patients})

# healthcare_project/urls.py
from django.contrib import admin
from django.urls import path
from patients import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('patients/', views.patient_list),
]
```

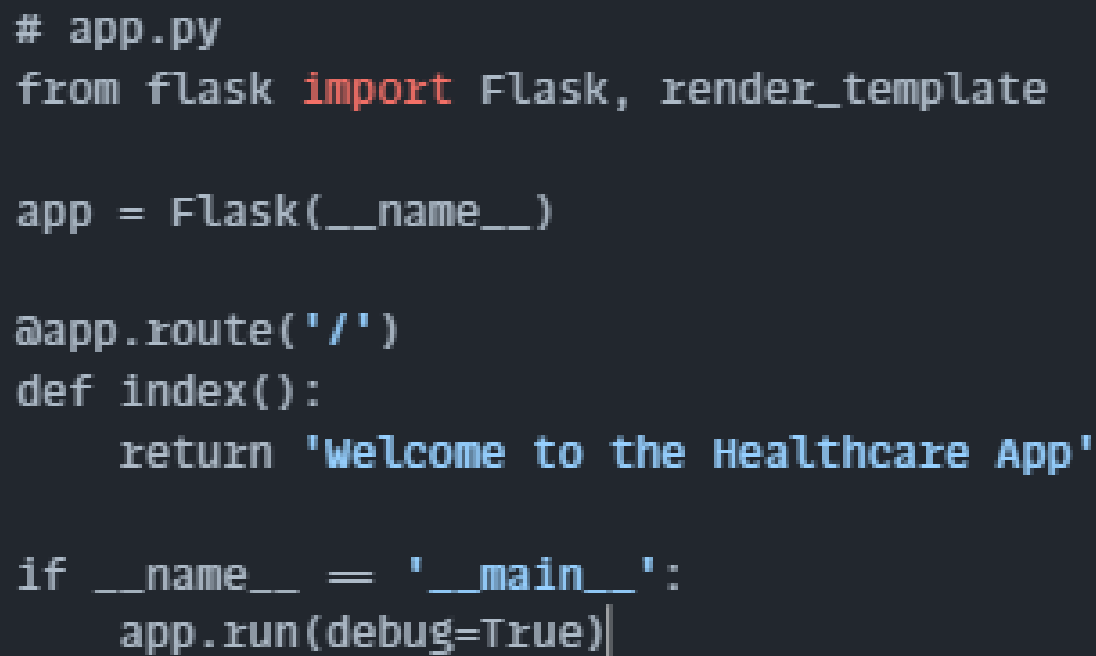


```
<!DOCTYPE html>
<html>
<head>
  <title>Patient List</title>
</head>
<body>
  <h1>Patient List</h1>
  <ul>
    {% for patient in patients %}
      <li>{{ patient.name }} - {{ patient.birth_date }}</li>
    {% endfor %}
  </ul>
</body>
</html>
```

Django é um poderoso frameworks para o desenvolvimento de aplicações web com Python. Django oferece um conjunto completo de ferramentas para construir aplicações robustas e seguras rapidamente.

Criando uma Aplicação Flask

Vamos criar uma aplicação Flask básica:



```
# app.py
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/')
def index():
    return 'Welcome to the Healthcare App'

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

Criando Rotas e Templates no Flask

Vamos criar uma rota e um template para listar os pacientes:

```
# app.py
from flask import Flask, render_template

app = Flask(__name__)

patients = [
    {'name': 'John Doe', 'birth_date': '1990-01-01'},
    {'name': 'Jane Smith', 'birth_date': '1985-05-12'}
]

@app.route('/patients')
def patient_list():
    return render_template('patient_list.html', patients=patients)

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

# templates/patient_list.html
<!DOCTYPE html>
<html>
<head>
    <title>Patient List</title>
</head>
<body>
    <h1>Patient List</h1>
    <ul>
        {% for patient in patients %}
        <li>{{ patient.name }} - {{ patient.birth_date }}</li>
        {% endfor %}
    </ul>
</body>
</html>
```


AGRADECIMENTOS

OBRIGADA POR LER ATÉ AQUI!

Este ebook foi criado por inteligência artificial e diagramado por um humano.

O conteúdo editável você encontra lá no meu perfil do GitHub, onde eu mostro o meu desenvolvimento e o processo migratório de carreira.



<https://github.com/BrunaSementino/Ebook---DIO>