PYTHON ACADEMY

TRABALHANDO COM ESTRUTURAS COMPLEXAS NO PYDANTIC

Neste ebook, veremos como o Pydantic lida com modelos aninhados, permitindo representar relações entre dados em Python. Vamos explorar como o Pydantic lida com listas, dicionários e estruturas JSON complexas.

PYTHONACADEMY.COM.BR

Salve salve Pythonista 👋

No desenvolvimento de aplicações Python, a manipulação de estruturas de dados complexas pode ser desafiadora.

Com o **Pydantic v2**, a tarefa de validação e modelagem torna-se mais intuitiva.

Este artigo foca nos modelos aninhados, que permitem representar relações entre dados em Python.

Vamos explorar como o Pydantic lida com listas, dicionários e estruturas JSON complexas.

Além disso, veremos como converter modelos aninhados em dicionários e JSON, essencial para transferências de dados.

Representando relações entre dados

O **Pydantic** permite a construção de modelos aninhados, ou seja, modelos que contêm outros modelos como campos.

Esta função é útil para representar estruturas hierárquicas.

Veja como criar um modelo aninhado simples:

```python
from pydantic import BaseModel

class Endereco(BaseModel):
    rua: str
    cidade: str

class Usuario(BaseModel):
    nome: str
    email: str
    endereco: Endereco
```

Neste exemplo, `Usuario` possui um campo `endereco` do tipo `Endereco`.

Isso estabelece uma relação entre `Usuario` e seu endereço.

Validação de listas e dicionários

O **Pydantic** também lida bem com coleções de dados.

Podemos validar listas e dicionários de modelos aninhados facilmente.

Veja um exemplo com listas:

```python
class Pedido(BaseModel):
    id_pedido: int
    descricao: str

class Cliente(BaseModel):
    nome: str
    pedidos: list[Pedido]
```

Aqui, `Cliente` tem uma lista de `pedidos`, cada um representado pelo modelo `Pedido`.

Vamos ver como instanciar e validar:

```
pedido1 = Pedido(id_pedido=1, descricao="Pedido 1")
pedido2 = Pedido(id_pedido=2, descricao="Pedido 2")
cliente = Cliente(nome="Alice", pedidos=[pedido1, pedido2])
print(cliente)
```

E a saída será:

```
nome='Alice' pedidos=[Pedido(id_pedido=1, descricao='Pedido 1'), Pedi-
do(id_pedido=2, descricao='Pedido 2')]
```

Essa abordagem promove **organização e clareza** no código, especialmente com **estruturas JSON complexas** em APIs.

> ***Modelos complexos na prática:*** *Estou usando Pydantic para estruturar dados no* ***DevBook****, uma plataforma que gera ebooks técnicos com IA. Syntax highlighting perfeito, infográficos automáticos e exportação em PDF profissional. Vale conferir!*

Validando estruturas JSON complexas

Estruturas JSON complexas são comuns ao lidar com APIs.

O Pydantic facilita a **validação** e manipulação dessas estruturas.

Veja como usar modelos para validar um JSON complexo:

```python
import json

dados_json = '''
{
    "nome": "Carlos",
    "email": "carlos@example.com",
    "endereco": {
        "rua": "Rua das Flores",
        "cidade": "São Paulo"
    }
}
'''

dados = json.loads(dados_json)
usuario = Usuario(**dados)
print(usuario)
```

E a saída será:

```
nome='Carlos' email='carlos@example.com' endereco=Endereco(rua='Rua das Flores', cidade='São Paulo')
```

Neste exemplo, o JSON é convertido em um modelo `Usuario`, que é validado automaticamente.

Conversão de modelos em dicionários e JSON

O **Pydantic** permite converter modelos em dicionários e JSON facilmente.

Isso é útil para exportar dados de forma estruturada.

Conversão para dicionários

Usando o método `model_dump`, é possível converter para dicionário:

```
usuario_dict = usuario.model_dump()
print(usuario_dict)
```

A saída será:

```
{'nome': 'Carlos', 'email': 'carlos@example.com', 'endereco': {'rua':
'Rua das Flores', 'cidade': 'São Paulo'}}
```

Conversão para JSON

Com `model_dump_json`, transformamos para JSON:

```
usuario_json = usuario.model_dump_json()
print(usuario_json)
```

A saída será:

```
{"nome": "Carlos", "email": "carlos@example.com", "endereco": {"rua":
"Rua das Flores", "cidade": "São Paulo"}}
```

Esses métodos são fundamentais para transferir dados em aplicações web.

Conclusão

Exploramos como o **Pydantic v2** facilita o trabalho com estruturas complexas por meio de modelos aninhados.

Vimos como representá-los e validá-los, garantindo consistência e precisão.

Aprendemos a integrar dados usando listas e dicionários, e a lidar com estruturas JSON complexas.

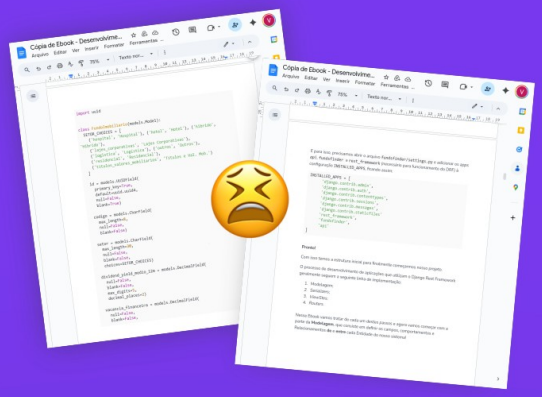
Além disso, mostramos a conversão eficiente para dicionários e JSON, essencial para muitas aplicações.

Não se esqueça de conferir!

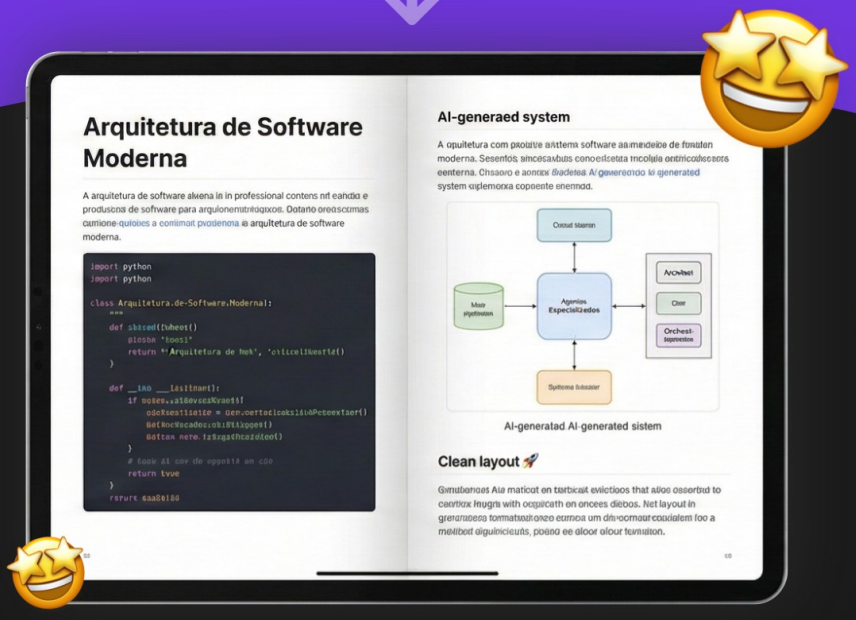
DevBook

Crie Ebooks técnicos em minutos com IA

Conheça a 1ª IA Especializada na criação de Ebooks **com código**!



Chega de formatar código no Google Docs



Syntax Highlight

Infográficos feitos por IA

Adicione Banners Promocionais

Deixe que nossa IA faça o trabalho pesado

Edite em Markdown em Tempo Real

TESTE AGORA

PRIMEIRO CAPÍTULO 100% GRÁTIS