



PYTHON
ACADEMY

INTRODUÇÃO AO PYDANTIC: VALIDAÇÃO E MODELAGEM DE DADOS SIMPLIFICADA COM PYTHON

Nesse ebook veremos como utilizar o Pydantic para validação e modelagem de dados em Python de forma simplificada.

PYTHONACADEMY.COM.BR

Este ebook foi gerado por



Crie Ebooks técnicos incríveis em minutos com IA

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



Chega de formatar código no Google Docs

Deixe que nossa IA faça o trabalho pesado

 Syntax Highlight

 Adicione Banners Promocionais

 Edite em Markdown em Tempo Real

 Infográficos feitos por IA

TESTE AGORA 

 PRIMEIRO CAPÍTULO 100% GRÁTIS

✓ **Atualizado para Pydantic V2** (Janeiro 2025)

Pydantic 2.x com performance 17x mais rápida, BaseModel, validação automática, casos práticos reais.

Salve salve Pythonista 🙌

A validação e modelagem de dados são aspectos importantes no desenvolvimento de qualquer aplicação em Python.

O **Pydantic** surge como uma ferramenta poderosa para lidar com isso de maneira eficiente.

Neste artigo, exploraremos o que é o Pydantic, como ele pode ajudar na validação automática de tipos, na manipulação de dados válidos e inválidos, e como ele facilita a serialização e desserialização de dados em JSON.

Com seu foco na simplicidade e robustez, o Pydantic pode ser um diferencial no seu projeto.

O que é o Pydantic e por que utilizá-lo?

Pydantic é uma biblioteca de modelagem e validação de dados para Python.

Ao trabalhar com dados, é comum a necessidade de garantir que eles estejam no formato e tipo corretos.

Pydantic simplifica essa tarefa com uma sintaxe clara e intuitiva!

Seus principais benefícios incluem a **validação automática de tipos**, a capacidade de converter tipos complexos com facilidade e a possibilidade de gerar dados confiáveis de fontes não confiáveis, como entradas de usuário.

Quando usar o Pydantic?

Use o Pydantic quando precisar garantir que dados estejam consistentes e seguros, especialmente em APIs, configurações, ou qualquer situação em que entradas externas sejam processadas.

Instalação e configuração inicial

Para começar a usar o Pydantic, você precisa instalá-lo em seu ambiente Python. Isso pode ser feito facilmente com o gerenciador de pacotes pip:

```
pip install pydantic
```

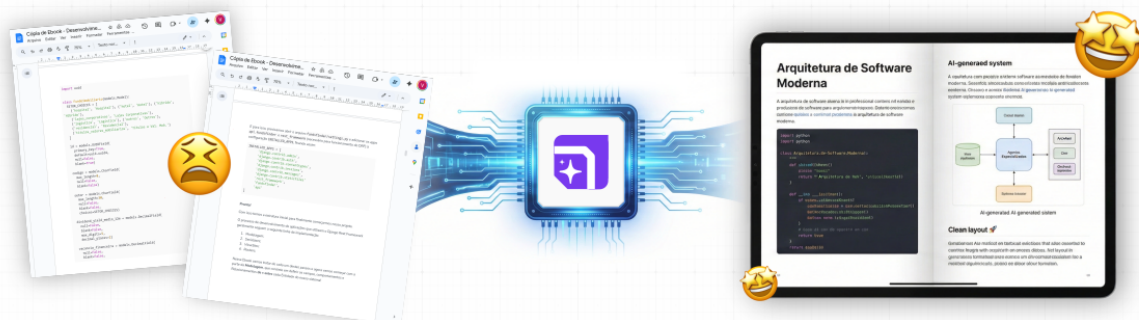
Após a instalação, você pode importar suas funcionalidades em seu projeto e começar a aproveitar seus benefícios.



*Estou desenvolvendo o **DevBook**, uma plataforma que usa IA para gerar ebooks técnicos profissionais. Te convido a conhecer!*

Crie Ebooks técnicos incríveis em minutos com IA

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



Chega de formatar código no Google Docs

Deixe que nossa IA faça o trabalho pesado

Syntax Highlight

Adicione Banners Promocionais

Edite em Markdown em Tempo Real

Infográficos feitos por IA

TESTE AGORA! PRIMEIRO CAPÍTULO 100% GRÁTIS

Introdução aos modelos básicos (BaseModel)

O coração do Pydantic é a classe **BaseModel**, que serve como ponto de partida para definir a estrutura dos seus dados. Vamos começar criando um exemplo básico de modelo:

```
from pydantic import BaseModel

class Usuario(BaseModel):
    nome: str
    idade: int
    email: str
```

Explicação do código:

1. **Importação:** Importamos a classe `BaseModel` do Pydantic.
2. **Definição de Classe:** Criamos a classe `Usuario` que herda de `BaseModel`.
3. **Atributos do Modelo:** Definimos os atributos `nome`, `idade` e `email`, especificando seus tipos.

Ao definir os tipos de dados, garantimos que o Pydantic valide automaticamente as entradas, reduzindo erros e melhorando a confiabilidade da aplicação.

Validação automática de tipos com Pydantic

A magia do Pydantic reside na sua capacidade de validar automaticamente o tipo de dados.

Vamos testar isso criando uma instância de `Usuario` com dados corretos e incorretos:

```
usuario_valido = Usuario(nome="João", idade=30,
                        email="joao@example.com")
print(usuario_valido)

try:
    usuario_invalido = Usuario(nome="Maria", idade="trinta", email="ma-
                              ria@example.com")
except ValueError as e:
    print(f"Erro de validação: {e}")
```

Explicação do código:

1. **Instância Válida:** Criamos um `usuario_valido` com os tipos corretos de dados para cada atributo.
2. **Instância Inválida:** Tentamos criar um `usuario_invalido` com um tipo de dado incorreto para `idade`.
3. **Captura de Erro:** O Pydantic lança um `ValueError` ao encontrar um tipo incorreto, que capturamos e imprimimos.

A capacidade do Pydantic em lidar com erros automaticamente é uma de suas características mais poderosas, ajudando a manter a integridade dos dados em sua aplicação.

Trabalhando com dados válidos e inválidos

Quando trabalhamos com dados, é comum encontrar entradas inválidas. O Pydantic fornece ferramentas para facilitar a manipulação desses casos. Vamos ver como ele trata entradas válidas e inválidas.

Dados válidos

Com o Pydantic, você pode contar com a conversão automática de tipos quando possível. Veja como:

```
usuario = Usuario(nome="Ana", idade="25", email="ana@example.com")
print(usuario)
```

E a saída será:

```
nome='Ana' idade=25 email='ana@example.com'
```

Explicação do código:

1. **Conversão de Tipos:** Embora `idade` seja uma string, o Pydantic converte para `int` quando possível.

Dados inválidos

Quando a conversão automática falha, o Pydantic lança um erro. Vamos ilustrar isso:

```
try:
    usuario = Usuario(nome=123, idade=25.5, email=None)
except ValueError as e:
    print(f"Erro de validação: {e}")
```

E a saída será:

```
Erro de validação: 3 validation errors for Usuario
nome
  Input should be a valid string [type=string_type, input_value=123,
input_type=int]
  For further information visit https://errors.pydantic.dev/2.10/v/
string_type
idade
  Input should be a valid integer, got a number with a fractional part
[type=int_from_float, input_value=25.5, input_type=float]
  For further information visit https://errors.pydantic.dev/2.10/v/
int_from_float
email
  Input should be a valid string [type=string_type, input_value=None,
input_type=NoneType]
  For further information visit https://errors.pydantic.dev/2.10/v/
string_type
```


Explicação do código:

1. **Erro de Tipos:** Passamos tipos errados que o Pydantic não pode converter automaticamente.
2. **Exceção Capturada:** Um `ValueError` é lançado informando o erro de tipo.

Serialização e desserialização de dados em JSON

Uma das funcionalidades mais práticas do Pydantic é a fácil conversão entre modelos e JSON, essencial para aplicações web.

Serialização

A partir de um modelo Pydantic, podemos facilmente serializá-lo em JSON:

```
usuario = Usuario(nome="Carlos", idade=40, email="carlos@example.com")
usuario_json = usuario.model_dump_json()
print(usuario_json)
```

E a saída será:

```
{"nome": "Carlos", "idade": 40, "email": "carlos@example.com"}
```

Explicação do código:

1. **Método JSON:** Usamos o método `model_dump_json()` da instância `Usuario` para serializá-la.
2. **Formato JSON:** A saída é uma string JSON representando o modelo.

Desserialização

Da mesma forma, podemos criar um modelo a partir de uma string JSON:

```
dados_json = '{"nome": "Luiza", "idade": 22, "email": "luiza@example.com"}'
usuario = Usuario.model_validate_json(dados_json)
print(usuario)
```

E a saída será:

```
nome='Luiza' idade=22 email='luiza@example.com'
```

Explicação do código:

1. **Método parse_raw:** `model_validate_json()` desserializa a string JSON para uma instância de `Usuario`.
2. **Objeto Python:** `usuario` agora é um objeto `Usuario` com os dados do JSON.

Caso Prático: Validação de API Request

```
from pydantic import BaseModel, EmailStr, Field, validator
from typing import Optional
import re

class CriarUsuarioRequest(BaseModel):
    nome: str = Field(..., min_length=3, max_length=50)
    email: EmailStr
    senha: str = Field(..., min_length=8)
    idade: int = Field(..., gt=0, lt=150)
    telefone: Optional[str] = None

    @validator('senha')
    def validar_senha_forte(cls, v):
        if not re.search(r'[A-Z]', v):
            raise ValueError('Senha deve ter maiúscula')
        if not re.search(r'[0-9]', v):
            raise ValueError('Senha deve ter número')
        return v

    @validator('telefone')
    def validar_telefone(cls, v):
        if v and not re.match(r'^\(\d{2}\) \d{5}-\d{4}$', v):
            raise ValueError('Formato: (11) 99999-9999')
        return v

# Uso em API FastAPI
from fastapi import FastAPI, HTTPException

app = FastAPI()

@app.post("/usuarios")
def criar_usuario(usuario: CriarUsuarioRequest):
    # Pydantic valida automaticamente!
    return {"mensagem": f"Usuário {usuario.nome} criado"}
```

Conclusão

Neste guia sobre **Pydantic**, você aprendeu:

- ✓ **Instalação** - Setup do Pydantic V2
- ✓ **BaseModel** - Criar modelos de dados tipados
- ✓ **Validação automática** - Tipos checados automaticamente
- ✓ **Validators** - Regras de validação customizadas
- ✓ **JSON** - Serialização e desserialização
- ✓ **Caso prático** - Validação de API request

Principais lições: - Pydantic **valida automaticamente** tipos de dados - **17x mais rápido** na versão 2.x - Ideal para **APIs** (FastAPI usa Pydantic) - **Validators** para regras complexas - Serializa/desserializa **JSON facilmente**

Próximos passos: - Aprenda sobre [Modelos complexos](#) - Explore [Tipos avançados](#)
- Use com FastAPI para APIs robustas

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