Pydantic

Live de python # 165

Roteiro

1. Pydantic
O que é e para que serve

2. Dataclasses
Dataclasses + runtime

3. Modelos
Criando nossa classes

4. Validadores
Fazendo validações dos campos

5. Campos
Pydantic além do typing

6. Modelos de configuração
Montando arquivos de configuração

picpay.me/dunossauro

apoia.se/livedepython

PIX

Ajude o projeto

Ademar Peixoto, Alex Lima, Alexandre Harano, Alexandre Santos, Alexandre Tsuno, Alexandre Villares, Alynne Ferreira, Alysson Oliveira, Amaziles Carvalho, Andre Rodrigues, André Rocha, Arnaldo Turque, Bruno Batista, Bruno Oliveira, Caio Nascimento, César Almeida, César Moreira, Davi Ramos, David Kwast, Diego Guimarães, Dilenon Delfino, Douglas Bastos, Elias Soares, Eugenio Mazzini, Everton Alves, Fabiano Gomes, Fabio Barros, Fabio Castro, Fabrício Coelho, Flavkaze Flavkaze, Franklin Silva, Fábio Serrão, Gabriel Simonetto, Gabriel Soares, Gabriela Santiago, Geandreson Costa, Guilherme Felitti, Guilherme Marson, Guilherme Ostrock, Gustavo Chacon, Henrique Machado, Hélio Neto, Israel Fabiano, Italo Silva, Johnny Tardin, Jonatas Leon, Jorge Plautz, José Prado, João Lugão, João Schiavon, Juan Gutierrez, Jônatas Silva, Júlia Kastrup, Kaneson Alves, Leonardo Cruz, Leonardo Galani, Leonardo Mello, Lidiane Monteiro, Lorena Ribeiro, Lucas Barros, Lucas Mello, Lucas Mendes, Lucas Teixeira, Lucas Valino, Luciano Ratamero, Marcelo Rodrigues, Maiquel Leonel, Maiquel Leonel, Marcela Campos, Maria Clara, Matheus Vian, Melissa Mendonça, Natan Cervinski, Nicolas Teodosio, Patric Lacouth, Patricia Minamizawa, Patrick Gomes, Paulo Tadei, Pedro Pereira, Peterson Santos, Rafael Lino, Reinaldo Silva, Revton Silva, Rodrigo Ferreira, Rodrigo Mende, Rodrigo Vaccari, Sandro Mio, Silvio Xm, Thiago Araujo, Thiago Borges, Thiago Bueno, Tyrone Damasceno, Victor Geraldo, Vinícius Bastos, Vinícius Ferreira, Vítor Gomes, Wesley Mendes, Willian Lopes, Willian Lopes, Willian Rosa, Wilson Duarte, Ronaldo Silva, Wendel Rios, Érico Andrei

Obrigado você

O que é?
Como se alimenta?

Pydantic

Pydantic

Pydantic é uma biblioteca python para validação de dados (serialização) e gerenciamento de configurações.

- Licença: MIT
- Primeira versão: Março 2017 (~4 anos de vida)
- Suporte: 3.6+

Por que migrar ou usar?

Existem diversas opções para fazer totalmente ou parcialmente o que o pydantic faz:

- Attrs
- Valideer
- Marshmallow
- Django Rest Framework
- Cerberus
- ...

Package	Version	Relative Performance	Mean validation time
pydantic	`1.7.3`		93.7µs
attrs + cattrs	`20.3.0`	1.5x slower	143.6µs
valideer	`0.4.2`	1.9x slower	175.9µs
marshmallow	`3.10.0`	2.4x slower	227.6µs
voluptuous	`0.12.1`	2.7x slower	257.5µs
trafaret	`2.1.0`	3.2x slower	296.7µs
schematics	`2.1.0`	10.2x slower	955.5µs
django-rest-framework	`3.12.2`	12.3x slower	1148.4µs
cerberus	`1.3.2`	25.9x slower	2427.6µs

https://pydantic-docs.helpmanual.io/benchmarks/

Pydantic

O grande trunfo do Pydantic, e aqui é a minha opinião, é fazer o uso da tipagem em tempo de execução (tipagem ganso [goose typing]).

- Pode-se validar dados em tempo de execução

```python
from pydantic import validate_arguments


@validate_arguments
def soma(x: int, y: int) -> int:
    return x + y
```

```python
from pydantic import validate_arguments


@validate_arguments
def soma(x: int, y: int) -> int:
    return x + y
```

```
>>> soma(1, 2)
3
```

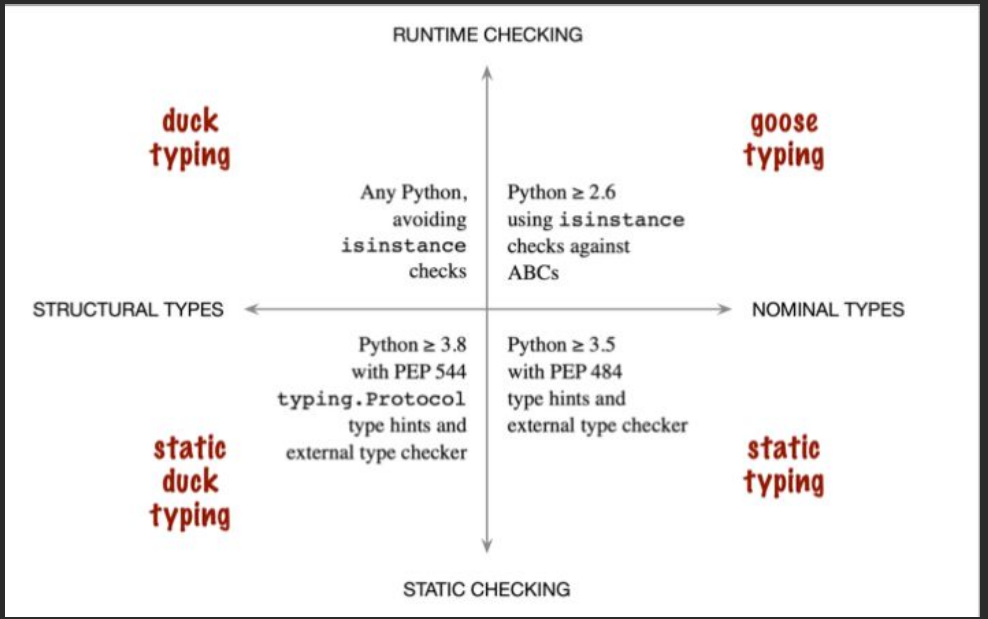
```python
from pydantic import validate_arguments


@validate_arguments
def soma(x: int, y: int) -> int:
    return x + y
```

```
>>> soma('', '')
ValidationError: 2 validation errors for Soma
x
  value is not a valid integer (type=type_error.integer)
y
  value is not a valid integer (type=type_error.integer)

```

Tipagem ganso?



Fluent Python, 2en Edition - capítulo 13

Luciano Ramalho

https://youtu.be/AJK2LqrlnTE

pip install pydantic

Bora instalar o pydantic

Data classes

Agora com validação

Uma breve conversa

Já conversamos sobre Dataclasses e obsessão por primitivos na **Live de Python #150**. Basicamente o que precisamos ter em mente, é o seguinte fluxo:

- **Dicionário**: Quando precisamos dar nomes as coisas
  - d = {'a': 1}
  - **d['a']**
- **Namedtuple**: Quando precisamos acessar pelos nomes
  - d(a=1)
  - **d.a**

Uma breve conversa

Já conversamos sobre Dataclasses e obsessão por primitivos na **Live de Python #150**. Basicamente o que precisamos ter em mente, é o seguinte fluxo:

- **Dataclass**: Validar tipos estaticamente
  - d: int = 7
- **Pydantic**: Validar tipos em tempo de execução (ganso)
  - d: int = 7

Estático x Em runtime (Code!)

```python
from dataclasses import dataclass

@dataclass
class Pessoa:
    idade: int
    nome: str
```

```python
from pydantic.dataclasses import (
  dataclass
)

@dataclass
class Pessoa:
    idade: int
    nome: str
```

```python
p = Pessoa(28, 'Eduardo')
```

Dando poderes estáticos ao Pydantic (Shell)

Você pode exportar os super poderes do pydantic ao **mypy.ini**:

```
1   [mypy]
2   plugins = pydantic.mypy
```

Criando modelos para validação

Models

Models

Os modelos do Pydantic são responsáveis pelas validações de dados.

Exemplos:

- Em um servidor web: Validar payloads de entradas e saídas.
  - **Entrada**: {'user': 'fausto@livedepython.com', 'senha': 12345}
  - **Saída**: {'message': 'Login efetuado com sucesso'}
- Validações com pandas:
  - Verificar se consistência dos dados
- Validação de configurações
- ...

Models

Os modelos do pydantic são criados via Herança da classe **BaseModel**.

```python
from pydantic import BaseModel

class Cadastro(BaseModel):
    nome: str
    idade: int
    sobrenome: str
    email: str
    senha: str
    ativo: bool = True
```

Models

Os modelos do pydantic são criados via Herança da classe **BaseModel**.

Default

```python
from pydantic import BaseModel

class Cadastro(BaseModel):
    nome: str
    idade: int
    sobrenome: str
    email: str
    senha: str
    ativo: bool = True
```

Models

Os modelos do pydantic são criados via Herança da classe **BaseModel**.

```python
from pydantic import BaseModel

class Cadastro(BaseModel):
    nome: str
    idade: int
    sobrenome: str
    email: str
    senha: str
    ativo: bool = True
```

```python
json = {
  'nome': 'Fausto',
  'idade': 29,
  'sobrenome': 'Mago',
  'email': 'fausto@livedepython.com',
  'senha': 1234
}
```

```python
from pydantic import BaseModel

class Cadastro(BaseModel):
    nome: str
    idade: int
    sobrenome: str
    email: str
    senha: str
    ativo: bool = True
```

```python
json = {
  'nome': 'Fausto',
  'idade': 29,
  'sobrenome': 'Mago',
  'email': 'fausto@livedepython.com',
  'senha': 1234
}
```

```
>>> dado = Cadastro(**json)

>>> dado.nome
'Fausto'

>>> dado.email
'fausto@livedepython.com'

>>> dado.ativo
True
```

```python
from pydantic import BaseModel

class Cadastro(BaseModel):
    nome: str
    idade: int
    sobrenome: str
    email: str
    senha: str
    ativo: bool = True
```

Em breve falaremos sobre isso

```python
json = {
  'nome': 'Fausto',
  'idade': 29,
  'sobrenome': 'Mago',
  'email': 'fausto@livedepython.com',
  'senha': 1234
}
```

```
>>> dado = Cadastro(**json)

>>> dado.nome
'Fausto'

>>> dado.email
'fausto@livedepython.com'

>>> dado.ativo
True
```

Validadores

Validando campos

Validadores

Os validadores são adicionais a tipagem. Existem 3 tipos de validadores:

- **default**: Validação durante o carregamento
- **pré**: Validação antes do carregamento
- **Por item**: Validação de containers (lista, tupla, dict, ...)

Exemplo de Cadastro

```python
from pydantic import BaseModel, validator

class Cadastro(BaseModel):
  email: str
  senha_1: str
  senha_2: str
```

Exemplo de Cadastro

```python
from pydantic import BaseModel, validator

class Cadastro(BaseModel):
  email: str
  senha_1: str
  senha_2: str
```

```python
  @validator('email')
  def email_tem_arroba(cls, v):
    if '@' not in v:  # v = email
      raise ValueError('Email não tem @')
    return v
```

Exemplo de Cadastro

```python
from pydantic import BaseModel, validator

class Cadastro(BaseModel):
    email: str
    senha_1: str
    senha_2: str
```

```python
    @validator('email')
    def email_tem_arroba(cls, v):
        if '@' not in v:  # v = email
            raise ValueError('Email não tem @')
        return v
```

Exemplo de Cadastro

```python
from pydantic import BaseModel, validator

class Cadastro(BaseModel):
    email: str
    senha_1: str
    senha_2: str
```

```python
    @validator('email')
    def email_tem_arroba(cls, v):
        if '@' not in v:  # v = email
            raise ValueError('Email não tem @')
        return v
```

Exemplo de Cadastro

```python
from pydantic import BaseModel, validator

class Cadastro(BaseModel):
  email: str
  senha_1: str
  senha_2: str
```

```python
  @validator('senha_1', 'senha_2')
  def senha_tem_10_caracteres(cls, v):
    if len(v) >= 10:
      return v
    raise ValueError('Senha menor que 10')
```

Exemplo de Cadastro

```python
from pydantic import BaseModel, validator

class Cadastro(BaseModel):
    email: str
    senha_1: str
    senha_2: str
```

```python
    @validator('*')
    def minusculo(cls, v):
        if v.lower() == v:
            return v
        raise ValueError('Campo comtém letras maiúsulas')
```

Exemplo de Cadastro

```python
from pydantic import BaseModel, validator

class Cadastro(BaseModel):
  email: str
  senha_1: str
  senha_2: str
```

```python
  @validador('senha_2')
  def senhas_iguais(cls, v, values, **kwargs):
    if v == values['senha_1']:
      return v
    raise ValueError('Senhas diferentes')
```

Exemplo de Cadastro

```python
from pydantic impor

class Cadastro(Base
    email: str
    senha_1: str
    senha_2: str
```

Em breve falaremos sobre isso

```python
    @validador('senha_2')
    def senhas_iguais(cls, v, values, **kwargs):
        if v == values['senha_1']:
            return v
        raise ValueError('Senhas diferentes')
```

Pré validação

O pré validador pode ser usado em chamadas e de transição ou coisas que são recebidas e uma forma, mas devem ser processadas de outra.

```python
from pydantic import BaseModel, validator

class Pedidos(BaseModel):
  ids: list[int]

  @validator('ids', pre=True)
  def parse_ids(cls, v):
    return v.split(',')
```

Validação por item

Quando lidamos com containers (itens que contém outros itens) podemos validar individualmente cada item

```python
from pydantic import BaseModel, validator

class Pedidos(BaseModel):
  ids: list[int]

  @validator('ids', each_item=True)
  def parse_ids(cls, v):
    if v > 0:
      return v
    raise ValueError('Id menor que zero')
```

Pydantic além do typing

Cam pos

Fields

Além dos campos definidos na linguagem, o Pydantic conta com uma diversidade de campos:

- Constrained
  - PositiveInt, NegativeFloat, ...
- Strinct
  - StrictInt, StrictFloat, ...
  - valores sem conversões
- Tipos do Pydantic

Pydantic Field

Além das validações tradicionais, o pydantic conta com Fields próprios:

- EmailStr - *email_validator*
- NameEmail - *email_validator*
- FilePath
- DirectoryPath
- Cartão de crédito
- Json
- Color
- Urls
- Secrets

https://pydantic-docs.helpmanual.io/usage/types/#pydantic-types

Field

O pydantic conta com um campo genérico, que você pode configurar como achar melhor e não necessariamente criar um validador

Exemplo de um número maior que 0, menor igual a 10

```python
from pydantic import BaseModel, Field

class MyClass(BaseModel):
    numero: int = Field(1, le=10, gt=0)
    lista: list[int] = Field([], max_items=3)
```

https://pydantic-docs.helpmanual.io/usage/schema/#field-customisation

Gerenciando configurações do pydantic

Config

Arquivos de configuração

O pydantic também provém uma camada para gerenciar arquivos de configuração. A classe **BaseSettings**.

Usando BaseSettings podemos carregar variáveis de ambiente e fazer o load de arquivo .env [python-dotenv]

Um arquivo básico de configuração

```
1   from pydantic import BaseSettings, Field
2
3   class Settings(BaseSettings):
4     postgres_url: str = Field(..., env='postgres_url')
```

Um arquivo básico de configuração

```python
from pydantic import BaseSettings, Field

class Settings(BaseSettings):
    postgres_url: str = Field(..., env='postgres_url')
```

Um arquivo básico de configuração

```python
from pydantic import BaseSettings, Field

class Settings(BaseSettings):
    postgres_url: str = Field(..., env='postgres_url')
```

Nome da variável de ambiente POSTGRES_URL

Usando prefixos

```python
from pydantic import BaseSettings, Field

class TestingSettings(BaseSettings):
  postgres_url: PostgresDsn

  class Config:
    env_prefix = 'testing_'
```

Usando Literal para escolher

```python
from typing import Literal, Union
from pydantic import BaseModel, BaseSettings

class ProductionSettings(BaseSettings):
  env: Literal['prod']
  server_url: str = 'https://'

  class Config:
    env_prefix = 'prod_'
```

```python
class TestingSettings(BaseSettings):
  env: Literal['test']
  server_url: str = 'http://'

  class Config:
    env_prefix = 'test_'
```

```python
class Config(BaseModel):
  config: Union[TestingSettings, ProdSettings]
```

picpay.me/dunossauro

apoia.se/livedepython

PIX

Ajude o projeto