PYTHON ACADEMY

COMO MANIPULAR JSON COM PYTHON

Guia completo json: loads/dumps, load/dump (arquivos), indent, ensure_ascii, casos práticos (APIs, configuração, nested), json vs dict, validação.

PYTHONACADEMY.COM.BR

Crie ebooks profissionais incríveis em minutos com IA

Chega de formatar texto no Google Docs, Word ou ferramentas que só te fazem perder tempo...

E deixe que nossa IA faça o trabalho pesado!

- Capas gerados por IA
- Infográficos feitos por IA
- Edite em Markdown em Tempo Real
- Adicione Banners Promocionais

TESTE AGORA

PRIMEIRO CAPÍTULO 100% GRÁTIS

> ✅ ***Atualizado para Python 3.13*** *(Dezembro 2025) Biblioteca json nativa - trabalhe com APIs, configs e dados estruturados!*

Salve salve Pythonista!

JSON (JavaScript Object Notation) é o formato mais usado em APIs e configurações! Biblioteca `json` é nativa.

Neste guia:

- ✅ **loads/dumps** - String ↔ Python
- ✅ **load/dump** - Arquivo ↔ Python
- ✅ **Casos práticos** - APIs, configs
- ✅ **indent/ensure_ascii** - Formatação

O JSON (JavaScript Object Notation) é um formato de dados muito popular na web, que permite a troca de informações estruturadas entre diferentes sistemas.

Ele é amplamente utilizado em aplicações RESTful, em bancos de dados NoSQL, em APIs e em muitos outros contextos.

No Python, a biblioteca padrão `json` oferece suporte para a manipulação de JSON de forma fácil e eficiente!

Neste artigo, vamos aprender como manipular JSON com Python. Você vai aprender a escrever dados em um arquivo JSON, ler dados de um arquivo JSON, converter uma string JSON em um objeto Python e também a converter um objeto Python em uma string JSON.

Então vamos nessa, que temos bastante conteúdo!

Escrevendo dados em um arquivo JSON

A primeira competência que vamos aprender é como escrever dados em um arquivo JSON usando Python.

Para isso, precisaremos da biblioteca `json` e do método `dump()`. Veja o exemplo abaixo:

```
import json

dados = {
    "nome": "João",
    "idade": 30,
    "cidade": "São Paulo"
}

arquivo = open("dados.json", "w")
json.dump(dados, arquivo)
arquivo.close()
```

No exemplo acima, criamos um dicionário chamado `dados`, que contém informações pessoais.

O arquivo `dados.json` será criado e as informações contidas no dicionário serão escritas nesse arquivo.

Note que utilizamos o modo `"w"` para abrir o arquivo em modo de escrita.

Lendo dados de um arquivo JSON

Agora que já sabemos como escrever dados em um arquivo JSON, vamos aprender como lê-los de um arquivo para dentro do Python.

Para isso, utilizamos o método `load()` da biblioteca `json`. Veja o exemplo a seguir:

```
import json

arquivo = open("dados.json", "r")
dados = json.load(arquivo)
arquivo.close()

print(dados)
```

No exemplo acima, abrimos o arquivo `dados.json` em modo de leitura (`"r"`) e utilizamos o método `json.load()` para carregar os dados desse arquivo em uma variável chamada `dados`.

Por fim, exibimos os dados na tela com a função `print()`.

> 💡 *Estou desenvolvendo o **Ebookr.ai**, uma plataforma que transforma suas ideias em ebooks profissionais usando IA — com geração de capa, infográficos e exportação em PDF. Te convido a conhecer!*

Convertendo uma string JSON em um objeto Python

Além de escrever e ler dados de um arquivo JSON, podemos também converter uma string JSON em um objeto Python.

Para isso, utilizamos o método `loads()` da biblioteca `json`. Veja o exemplo abaixo:

```python
import json

string_json = '{"nome": "Ana", "idade": 25, "cidade": "Rio de
        Janeiro"}'
dados = json.loads(string_json)

print(dados)
```

No exemplo acima, temos uma string chamada `string_json`, que contém os mesmos dados do exemplo anterior.

Utilizando o método `json.loads()`, convertemos essa string em um objeto Python. Em seguida, exibimos os dados na tela.

Convertendo um objeto Python em uma string JSON

Da mesma forma que podemos converter uma string JSON em um objeto Python, também podemos converter um objeto Python em uma string JSON.

Para isso, utilizamos o método `dumps()` da biblioteca `json`. Veja o exemplo a seguir:

```
import json

dados = {
    "nome": "Pedro",
    "idade": 35,
    "cidade": "Belo Horizonte"
}

string_json = json.dumps(dados)

print(string_json)
```

No exemplo acima, temos um dicionário chamado `dados`, que contém as mesmas informações dos exemplos anteriores.

Utilizando o método `json.dumps()`, convertemos esse dicionário em uma string JSON.

Por fim, exibimos a string na tela com a função `print()`.

Conclusão

Casos Práticos

1. Trabalhar com APIs

```python
import json
import requests

# Fazer requisição
response = requests.get('https://api.example.com/users')

# Parse JSON
users = response.json()  # Ou: json.loads(response.text)

for user in users:
    print(f"{user['name']} - {user['email']}")
```

2. Arquivo de Configuração

```python
import json

# Ler config
with open('config.json', 'r') as f:
    config = json.load(f)

print(config['database']['host'])
print(config['api_key'])

# Atualizar e salvar
config['debug'] = False
with open('config.json', 'w') as f:
    json.dump(config, f, indent=2)
```

3. JSON Nested (Aninhado)

```python
import json

data = {
    'usuario': {
        'nome': 'Alice',
        'endereco': {
            'rua': 'Rua A',
            'cidade': 'São Paulo'
        }
    }
}

print(data['usuario']['endereco']['cidade'])  # São Paulo

# Serializar com indentação
json_str = json.dumps(data, indent=2, ensure_ascii=False)
print(json_str)
```

4. Validação de JSON

```python
import json

json_string = '{"nome": "Bob", "idade": 25}'

try:
    dados = json.loads(json_string)
    print("JSON válido!")
except json.JSONDecodeError as e:
    print(f"JSON inválido: {e}")
```

json vs dict

JSON é apenas uma **string formatada**. Dict é um **objeto Python**.

```
import json

# Dict Python
dict_py = {'nome': 'Alice', 'idade': 25}

# JSON (string)
json_str = '{"nome": "Alice", "idade": 25}'

# Conversão
json_from_dict = json.dumps(dict_py)  # dict → JSON
dict_from_json = json.loads(json_str)  # JSON → dict
```

Conclusão

Neste guia da biblioteca **json**, você aprendeu:

✅ **loads()** - String JSON → Python ✅ **dumps()** - Python → String JSON ✅ **load()/dump()** - Arquivos ✅ **indent** - Formatar saída ✅ **Casos práticos** - APIs, configs, validação

Principais lições:

- JSON é **string**, dict é **objeto**
- `loads/dumps` para **strings**
- `load/dump` para **arquivos**
- `indent=2` deixa JSON **legível**
- `ensure_ascii=False` preserva **acentos**

Próximos passos:

- Explore [CSV][csv-post] para dados tabulares
- Aprenda `requests` para APIs
- Pratique Pydantic para validação
- Estude jsonschema para schemas

Neste artigo, aprendemos como manipular JSON com Python!

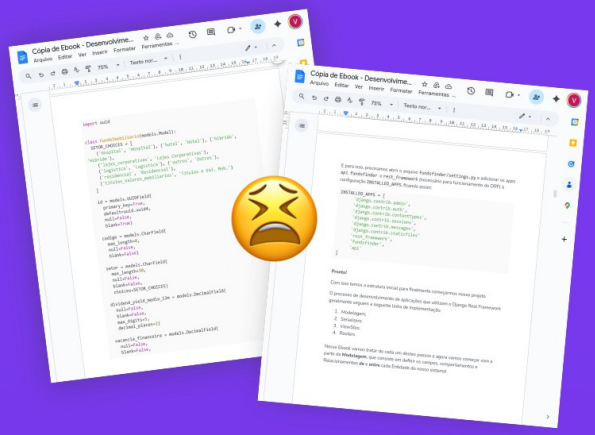
Com a biblioteca `json`, podemos realizar essas tarefas de forma simples e eficiente.

Com o tempo e a prática, você se tornará um especialista em manipulação de JSON com Python!

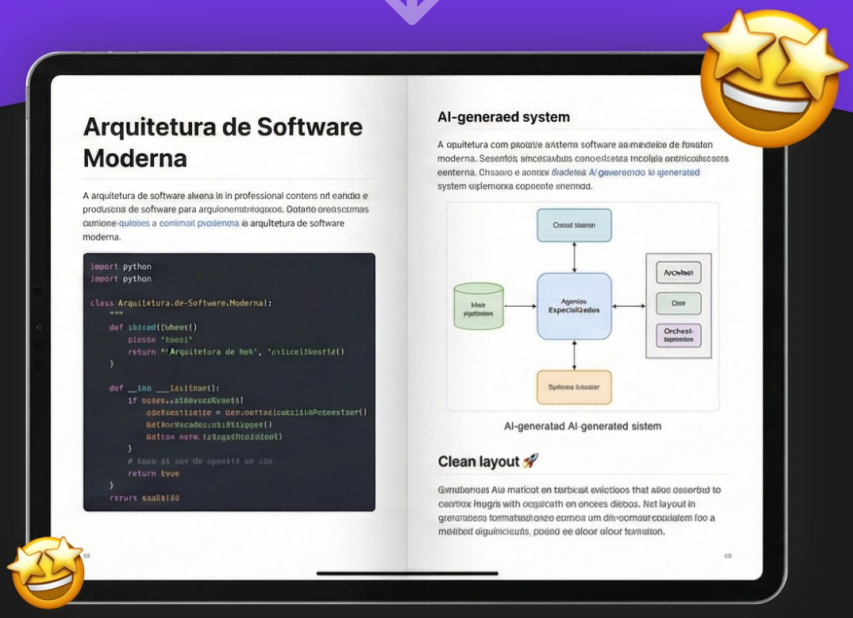
Não se esqueça de conferir!

Ebookr

Crie Ebooks profissionais em minutos com IA



Chega de formatar código no Google Docs ou Word



Capas gerados por IA

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