JSON: estrutura, aplicações, como usar em Python

Neste documento, vamos discutir sobre arquivos JSON. Trata-se de um padrão de arquivo muito usado no mundo da tecnologia para representar, armazenar e distribuir dados. Como é um formato padronizado, as mais distintas aplicações conseguem usar corretamente os arquivos JSON.

O nome JSON significa **J**ava**S**cript **O**bject **N**otation. Apesar do nome, não é algo apenas específico da linguagem JavaScript, mas sim um padrão aberto adotado amplamente.

O que é?

Um arquivo JSON é um arquivo texto (você consegue abri-lo facilmente no bloco de notas, VSCode, navegador, etc) com uma sintaxe que permite representar **um objeto** ou **uma sequência de objetos**.

Um objeto na representação JSON é muito parecido com um dicionário Python. A diferença é que o JSON é um pouco mais estrito nos tipos de dados que podem ser usados como chaves. Além disso, em JSON não podemos usar aspas simples para representar strings -- apenas aspas duplas.

Exemplo de JSON representando um único objeto:

```
{
    "nome" : "Alessandra",
    "sobrenome" : "Lima",
    "idade" : 39,
    "CPF" : "123456789-0"
}
```

Note, portanto, que a sintaxe é

```
{
    chave1 : valor1,
    chave2 : valor2,
    ...
}
```

Já um JSON representando uma sequência de objetos tem uma sintaxe similar à **lista** do Python: a lista começa com `[`, termina com `]` e os diferentes elementos são separados por vírgula `,`.

Por exemplo, um arquivo JSON representando várias pessoas poderia ser assim:

```
[
    {
        "nome" : "Alessandra",
        "sobrenome" : "Lima",
        "idade" : 39,
```

```
        "CPF" : "123456789-0"
    },
    {
        "nome" : "Joaquim",
        "sobrenome" : "Perez",
        "idade" : 18,
        "CPF" : "123123123-0"
    }
]
```

(Note que também é permitido que uma lista definida assim tenha um único item, ou até mesmo seja vazia.)

Portanto, o esquema geral seria:

```
[
    representacao_do_objeto_1,
    representacao_do_objeto_2,
    ...
    representacao_do_objeto_N
]
```

Onde cada objeto é representado com a sintaxe mostrada acima para um único objeto.

Exemplos mais complexos

Note que o valor em um campo de dados do JSON pode ser **outro objeto JSON** ou até uma **lista de outros objetos**.

Por exemplo, um JSON representando clientes de uma loja pode salvar um histórico de pedidos de cada pessoa.

```
[
    {
        "nome" : "José",
        "CPF" : "321321321-0",
        "pedidos" : [
            {
                "id_pedido" : 1234,
                "data" : "2023-09-22",
                "valor" : 1214.50
            },
            {
                "id_pedido" : 1245,
                "data" : "2024-01-10",
                "valor" : 799.00
            },
        ]
    },
```

```
    {
        "nome" : "Bruna",
        "CPF" : "456123789-0",
        "pedidos" : [
            {
                "id_pedido" : 1236,
                "data" : "2023-10-02",
                "valor" : 1113.80
            }
        ]
    }
]
```

(Observe que, no exemplo acima, a lista de pedidos da cliente Bruna tem apenas um pedido. Um cliente novo poderia ter uma lista de pedidos vazia. Ainda assim, salvamos como lista pois é o padrão escolhido nesse cenário de exemplo.)

Para que serve?

O padrão JSON é uma forma de representar dados de maneira que facilite seu uso e compartilhamento posterior.

- Permite interoperabilidade, pois diversos sistemas, programas, linguagens etc. suportam JSON;
- É fácil de usar porque a maioria das linguagens permite importar e exportar JSON (usando-se uma biblioteca adequada para isso);
- Em cenários simples, pode ser uma solução mais fácil do que usar banco de dados, porém mais robusta e estruturada do que um arquivo TXT simples ou CSV;
- É muito usado como formato de arquivo para distribuição de dados através de APIs.

Usando JSON no Python

Agora, veremos como:

1. Ler um arquivo JSON dentro de um programa Python, para acessar seu conteúdo;
2. Escrever um arquivo JSON para salvar informações de dentro do programa Python;
3. Acessar um JSON de uma API pública.

1. Como ler JSON no Python?

Precisaremos importar o módulo `json` dentro do Python, através de `import json`.

Para **leitura** de um arquivo JSON, precisamos:

1. Abrir o arquivo com o `open()` do Python;
2. Usar o método `load()` presente no módulo `json`.

Vamos mostrar isso através de exemplos concretos.

Exemplo 1

Salve o texto abaixo como um arquivo chamado `cliente.json`:

```
{
    "nome" : "Alessandra",
    "sobrenome" : "Lima",
    "idade" : 39,
    "CPF" : "123456789-0"
}
```

Na mesma pasta em que está o arquivo JSON, crie um programa python com o seguinte conteúdo:

```
import json

with open('cliente.json') as arquivo:
    cliente = json.load(arquivo)

print(cliente) # imprime dicionário todo

# podemos acessar campos específicos:
print(f'Nome completo: {cliente['nome']} {cliente['sobrenome']}.')
```

O método `json.load()` retorna um objeto Python representando o dado presente no JSON. Esse objeto pode ser:

- Um dicionário, caso o JSON represente objeto único;
- Uma lista, caso o JSON represente uma lista de objetos.

(O `load` irá retornar uma lista mesmo que seja uma lista com um único elemento! Caso isso seja confuso, reveja os exemplos de JSON mais acima.)

No caso do nosso exemplo, o arquivo `cliente.json` representa um objeto só. Portanto, no programa Python acima, a variável `cliente` vai receber um dicionário representando os dados presentes no JSON!

Caso o programa acima não esteja encontrando o arquivo JSON, certifique-se de abrir no VSCode a pasta em que o programa e o arquivo JSON estão armazenados.

Exemplo 1

Salve o texto abaixo como um arquivo chamado `clientes.json` (clientes, no plural!):

```
[
    {
        "nome" : "Alessandra",
        "sobrenome" : "Lima",
        "idade" : 39,
```

```
        "CPF" : "123456789-0"
    },
    {
        "nome" : "Jorge",
        "sobrenome" : "Lemos",
        "idade" : 61,
        "CPF" : "001002003-0"
    }
]
```

Na mesma pasta em que está o arquivo JSON, crie um programa python com o seguinte conteúdo:

```
import json

with open('clientes.json') as arquivo:
    pessoas = json.load(arquivo)

print(pessoas) # imprime lista toda

print('Sobrenomes dos clientes:')
for pessoa in pessoas:
    print(pessoa['sobrenome'])
```

No exemplo acima, como o arquivo JSON representa uma lista de objetos, a variável `pessoas` no código irá receber essa lista.

Note que **cada item da lista é um dicionário**, com os campos correspondentes ao que vemos no JSON.

2. Como escrever JSON no Python?

Dentro do módulo `json`, também temos o método `dump()` que nos permite **escrever** um arquivo JSON para salvar dados que colhemos durante a execução do programa.

Exemplo

Execute o seguinte programa Python:

```
import json

# cria um dicionário com dados obtidos por input
paciente = {
    'nome' : input('Qual o nome do paciente? '),
    'idade': int(input('Qual a idade do paciente? '))
}

with open('paciente.json', 'w') as arquivo:
    json.dump(paciente, arquivo, indent=4)
```

O programa acima pergunta o nome e a idade de um paciente. Após a leitura desses dados, ele cria um arquivo `paciente.json` onde armazena as informações obtidas na forma de JSON.

O método `json.dump()` tem dois parâmetros obrigatórios:

1. O dado sendo gravado no arquivo;
2. O arquivo (já aberto pela função `open()`).

Note que no exemplo acima ainda passamos um argumento opcional, `indent=4`, apenas para que a formatação fique mais "bonita" (legível), pois realiza indentação no JSON.

3. Como acessar uma API Pública?

Existem diversas APIs públicas que oferecem dados no formato JSON. (Muitas também dão suporte a outros formatos como CSV.)

- Algumas APIs oferecem dados simplesmente através de um acesso direto por link, como o `ExchangeRate-API.com`, que oferece dados de câmbio. Por exemplo, você pode acessar https://api.exchangerate-api.com/v4/latest/EUR no navegador e verá que carrega o JSON com dados de câmbio a partir do Euro (você pode trocar o EUR por outra coisa no fim do link para acessar câmbio a partir de outras moedas)
- Algumas exigem um cadastro gratuito. Cada pessoa que se cadastra no serviço recebe uma chave necessária para os acessos. Falaremos disso mais adiante.
- Outras ainda exigem pagamento. Elas ainda são consideradas "públicas" no sentido de que qualquer pessoa pode se cadastrar e pagar para acessar o serviço. É comum que APIs pagas também ofereçam um serviço gratuito mais básico.

Voltando ao exemplo das APIs que não exigem chave: você pode acessar https://api.exchangerate-api.com/v4/latest/EUR e salvar o arquivo como `cambio.json`.

Então, poderá usar o método já explicado acima para dar `json.load()` nesse conteúdo. Ótimo, não? Porém ainda podemos automatizar mais o processo. Em vez de baixar o arquivo manualmente, podemos pegar o arquivo por um **request HTTP** gerado pelo próprio programa Python.

Request HTTP

Para isso, precisaremos da biblioteca `requests`. Essa biblioteca traz utilidades para comunicação por protocolo HTTP. Como consumir JSON é uma tarefa **muito** comum nesse contexto, a própria biblioteca `requests` traz métodos para ler JSON.

Exemplo de programa:

```
import requests

url = 'https://api.exchangerate-api.com/v4/latest/EUR'


# faz uma requisição à URL e salva a resposta do servidor
resposta = requests.get(url)
```

```
# a resposta é 200 se tiver dado certo
if resposta.status_code == 200:
    cambio = resposta.json()
else:
    print('Falha em acessar o servidor com os dados. Terminando...')
    exit() # termina o programa

print('Cambio do euro (EUR) da seguinte data:', cambio['date'])
print('Qual a moeda alvo? Digite o codigo com tres letras')
print('Exemplo: real = BRL, libra = GBP, peso argentino = ARS')

moeda_desejada = input()
print('1 EUR = ', cambio['rates'][moeda_desejada], moeda_desejada)
```

A chamada `resposta.json()` equivale ao `json.load()` de quando estamos lendo arquivo obtido localmente.

Se o programa falhar por ausência da biblioteca `requests`, instale usando `pip`: `pip install requests`. Veja a apostila `pip.pdf` se necessário.

Request com chave de acesso

Se a API que você quer acessar exige chave de acesso, você precisará:

1. Cadastrar-se no serviço da API em questão;
2. Procurar instruções de como gerar uma chave (provavelmente você consegue ver no seu perfil ou receberá por e-mail).

Exemplo genérico de acesso usando chave:

```
import requests

url = 'https://jsonplaceholder.typicode.com/todos/1'
chave = 'MINHA_CHAVE' # coloque aqui a chave que você recebeu, como string

# no cabeçalho da requisição HTTP, vamos nos identificar
cabeçalho = {
    # a chave é nossa identidade
    'Authorization': f'Bearer {chave}'
}

resposta = requests.get(url, headers=cabeçalho)

if resposta == 200:
    print('Deu certo!')
else:
    print('Algo deu errado')
```

Dependendo da API específica que você está querendo acessar, o processo pode ter especificidades adicionais ou diferir do esquema acima. Procure a documentação da API e também peça ajuda ao professor se precisar 😃