Aula 03

Módulo json

O módulo json do Python é uma biblioteca integrada que permite codificar e decodificar dados no formato JSON (JavaScript Object Notation). O JSON é um formato de troca de dados leve e independente de plataforma que é amplamente utilizado para transmitir dados estruturados pela web.

Método dumps

O método dumps do módulo json do Python é usado para serializar objetos Python em uma string JSON. Ele recebe um objeto Python como argumento e retorna uma string JSON correspondente.

A sintaxe básica do método dumps é a seguinte:

```
>>> json.dumps(objeto, *, skipkeys=False, ensure_ascii=True,
 check_circular=True, allow_nan=True, cls=None, indent=None,
 separators=None, default=None, sort_keys=False, **kw)
```

Aqui está uma descrição detalhada dos parâmetros da função:

objeto: o objeto Python a ser serializado em formato JSON.

- skipkeys: se for True, as chaves que não são strings (como chaves numéricas em dicionários) serão ignoradas durante a serialização. O padrão é False.
- ensure_ascii: se for True, todos os caracteres não ASCII na string JSON resultante serão escapados como sequências de escape ASCII. O padrão é True.
- check_circular: se for True, a função irá verificar se há referências circulares no objeto Python. O padrão é True.
- allow_nan: se for True, a função irá permitir valores NaN, Infinity e -Infinity como valores numéricos na string JSON resultante. O padrão é True.
- cls: uma classe personalizada que pode ser usada para serializar objetos personalizados. O padrão é None.
- indent: um valor inteiro que define o número de espaços a serem usados para a indentação na string JSON resultante. Se indent for um valor inteiro maior que 1, a string JSON resultante será formatada com a indentação especificada. O padrão é None.
- separators: uma tupla de dois valores que especifica os separadores a serem usados na string JSON resultante. O primeiro valor é o separador para os pares chave-valor na estrutura JSON (o padrão é (", ", ": ")), e o segundo valor é o separador para os itens em listas JSON (o padrão é (", ", ": ")).
- default: uma função que pode ser usada para serializar objetos personalizados que não são serializáveis por padrão. O padrão é None.
- sort_keys: se for True, as chaves dos objetos JSON serão ordenadas em ordem alfabética na string JSON resultante. O padrão é False.
- kw: argumentos nomeados adicionais que podem ser passados para a função. Esses argumentos serão usados como valores padrão para a serialização de objetos personalizados.

Aqui temos um exemplo simples de uso do método dumps:

```python
"""módulo main.py"""
import json


if __name__ == '__main__':
    # objeto Python a ser serializado
    calculo = {'num_1': 10, 'num_2': 5, 'operacao': '+', 'resultado': 15}

    # serializar o objeto Python em uma string JSON
    json_string = json.dumps(calculo)

    # exibir a string JSON resultante
    print(json_string)

```

Neste exemplo, um objeto Python simples é criado e, em seguida, a função dumps é usada para serializar o objeto em uma string JSON correspondente. A string JSON resultante é então exibida na tela.

A função dumps é extremamente útil para enviar dados estruturados entre diferentes aplicativos ou plataformas, pois a string JSON pode ser facilmente convertida de e para objetos Python usando as funções loads e load do módulo json.

Aqui está um exemplo mais complexo de uso do método dumps com alguns argumentos opcionais:

```python
"""módulo main.py"""
import json


if __name__ == '__main__':
    # objeto Python a ser serializado
    calculo = {'num_1': 10, 'num_2': 5, 'operacao': '+',
               'resultado': 15, 'lista': [1, 2, 3], 'comentario': 'teste'}

    # serializar o objeto Python em uma string JSON
    json_string = json.dumps(calculo, indent=4, sort_keys=True)

    # exibir a string JSON resultante
    print(json_string)

```

```
PS C:\Users\gutoh\OneDrive\0.Python> python .\main.py
{
    "comentario": "teste",
    "lista": [
        1,
        2,
        3
    ],
    "num_1": 10,
    "num_2": 5,
    "operacao": "+",
    "resultado": 15
}
PS C:\Users\gutoh\OneDrive\0.Python>
```

Neste exemplo, um objeto Python mais complexo é criado, contendo um dicionário, uma lista e um valor numérico. A função dumps é usada para serializar o objeto em uma string JSON correspondente, usando a indentação de 4 espaços e ordenando as chaves do dicionário em ordem alfabética. A string JSON resultante é então exibida na tela.

Método loads

O método loads do módulo json do Python é usado para desserializar uma string JSON em um objeto Python. Essa função é usada quando se recebe uma string JSON de uma fonte externa e deseja convertê-la de volta em um objeto Python para manipulação.

Aqui está um exemplo simples de uso do método loads:

```python
"""módulo main.py"""
import json


if __name__ == '__main__':
    #objeto Python a ser serializado
    calculo_str = "{'num_1': 10, 'num_2': 5, 'operacao': '+', 'resultado': 15}"

    #serializar o objeto Python em uma string JSON
    calculo = json.loads(calculo_str)

    #exibir a string JSON resultante
    print(calculo)

```

Neste exemplo, uma string JSON simples é criada e armazenada na variável calculo_str. O método loads é usado para desserializar a string JSON em um objeto Python correspondente, que é então armazenado na variável objeto. O objeto Python resultante é então exibido na tela usando a função print.

O método loads pode lidar com vários tipos de dados JSON, incluindo objetos, arrays, números, strings e valores booleanos e nulos. Ele pode ser usado em conjunto com o método dumps para serializar e desserializar objetos Python em JSON.

É importante lembrar que a função loads só deve ser usada em dados JSON confiáveis e seguros, pois a desserialização de dados maliciosos pode representar um risco de segurança. É uma boa prática sempre validar e verificar os dados JSON antes de desserializá-los em objetos Python.

Comando with

O comando "with" é utilizado em Python para criar um contexto que garante que determinadas ações sejam realizadas de forma segura e automática, mesmo se ocorrerem exceções ou erros durante a execução do código.

Por exemplo, se quisermos abrir um arquivo e ler seu conteúdo, podemos usar o comando "with" da seguinte forma:

```python
"""módulo main.py"""


def abre_arquivo():
    """abre um arquivo e retorna seu conteúdo"""
    with open('operacoes.txt', 'r', encoding='utf-8') as arq:
        conteudo = arq.read()
    return conteudo


if __name__ == '__main__':
    print(abre_arquivo())

```

Nesse caso, o arquivo é aberto em modo de leitura ("r") e a variável "arq" é usada para referenciar o objeto que representa o arquivo aberto. Dentro do bloco de código, o método "read()" é usado para ler o conteúdo do arquivo e armazená-lo na variável "conteudo".

O comando "with" garante que o arquivo será fechado automaticamente ao final do bloco de código, mesmo se ocorrerem erros ou exceções durante a leitura do arquivo. Isso é feito pelo método "exit()" do objeto "arq", que é chamado automaticamente ao final do bloco.

O comando "with" em Python é frequentemente usado para gerenciar recursos que precisam ser abertos e fechados de forma adequada, mas também pode ser utilizado em outros contextos. Alguns exemplos incluem:

- Gerenciamento de transações de banco de dados: ao usar uma biblioteca de acesso a banco de dados que suporta o contexto "with", é possível iniciar uma transação de banco de dados no início do bloco de código e, automaticamente, fazer um rollback ou commit da transação no final do bloco;
- Gerenciamento de conexões de rede: ao usar sockets em Python, é possível criar uma conexão de rede usando o comando "with" e, automaticamente, fechar a conexão ao final do bloco de código.
- Gerenciamento de arquivos temporários: ao usar a biblioteca "tempfile" do Python, é possível criar arquivos temporários dentro de um contexto "with" e, automaticamente, excluí-los ao final do bloco.
- Gerenciamento de threads: ao usar a biblioteca "threading" do Python, é possível criar uma nova thread dentro de um contexto "with" e, automaticamente, encerrar a thread ao final do bloco.

Esses são apenas alguns exemplos de como o comando "with" pode ser utilizado em contextos diferentes do gerenciamento de recursos. O uso do comando "with" nesses contextos permite que o código seja escrito de forma mais legível e organizada, e garante que os recursos sejam utilizados de forma adequada e segura.

json e with

Além das funções dumps() e loads(), o módulo json do Python também fornece outras funções para personalizar a serialização e desserialização de objetos Python. Por exemplo, a função json.dump() pode ser usada para serializar um objeto Python diretamente em um arquivo, enquanto a função json.load() pode ser usada para desserializar um arquivo JSON em um objeto Python.

Método dump

O método dump do módulo json do Python é usado para serializar um objeto Python em um arquivo JSON. Em outras palavras, o método dump é usado para gravar dados estruturados em um arquivo JSON em disco.

Aqui está um exemplo simples de uso do método dump:

```python
"""módulo main.py"""
import json


if __name__ == '__main__':
    # objeto Python a ser serializado
    calculo = {'num_1': 10, 'num_2': 5, 'operacao': '+',
               'resultado': 15, 'lista': [1, 2, 3], 'comentario': 'teste'}

    with open('dados.json', 'w') as arq:
        json.dump(calculo, arq)
```

Como vai ficar o arquivo json:

```
{"num_1": 10, "num_2": 5, "operacao": "+", "resultado": 15, "lista": [1, 2, 3], "comentario": "teste"}
```

Neste exemplo, um objeto Python é criado e armazenado na variável objeto. O método dump é usado para serializar o objeto Python em um arquivo JSON, que é então gravado em disco usando a função open do Python. A opção 'w' no argumento open indica que o arquivo deve ser aberto no modo de escrita.

O método dump também suporta vários argumentos opcionais para personalizar a serialização, como a indentação, ordenação das chaves, codificação, entre outros. Aqui está um exemplo de uso do método dump com alguns argumentos opcionais:

```python
"""módulo main.py"""
import json


if __name__ == '__main__':
    # objeto Python a ser serializado
    calculo = {'num_1': 10, 'num_2': 5, 'operacao': '+',
               'resultado': 15, 'lista': [1, 2, 3], 'comentario': 'teste'}

    with open('dados.json', 'w') as arq:
        json.dump(calculo, arq, indent=4, sort_keys=True)
```

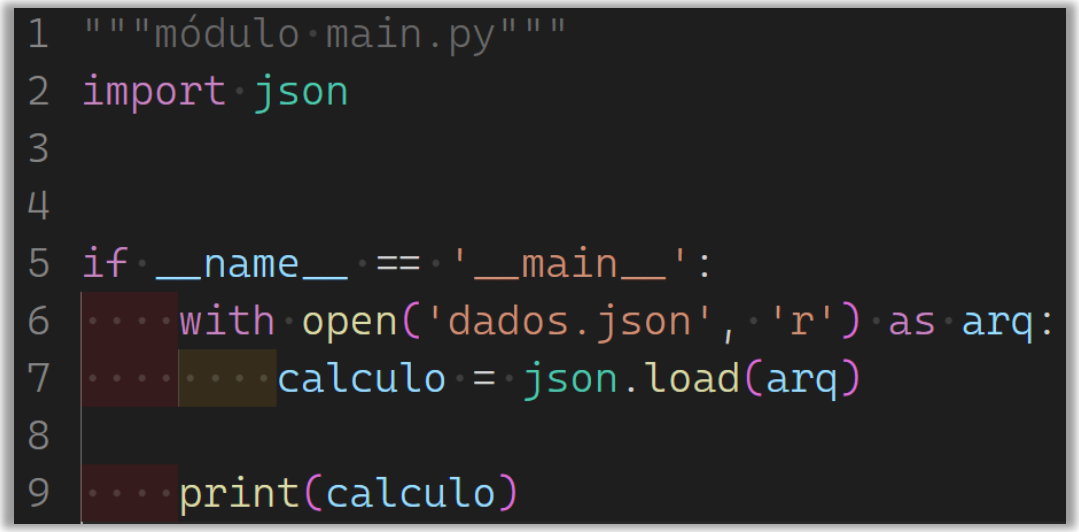
Como ficará o arquivo json:

Neste exemplo, o objeto Python é serializado em um arquivo JSON com indentação de 4 espaços e as chaves do dicionário são ordenadas em ordem alfabética.

Método load

O método load do módulo json do Python é usado para desserializar um arquivo JSON em um objeto Python. Em outras palavras, o método load é usado para carregar dados JSON de um arquivo em disco e convertê-los de volta em um objeto Python.

Aqui está um exemplo simples de uso do método load:



Neste exemplo, um arquivo JSON é carregado a partir do disco usando a função open do Python. A opção 'r' no argumento open indica que o arquivo deve ser aberto no modo de leitura. O método load é então usado para desserializar o conteúdo do arquivo JSON em um objeto Python correspondente, que é armazenado na variável calculo. O objeto Python resultante é então exibido na tela usando a função print.

Assim como o método loads, o método load pode lidar com vários tipos de dados JSON, incluindo objetos, arrays, números, strings e valores booleanos e nulos. Ele pode ser usado em conjunto com o método dump para serializar e desserializar objetos Python em JSON.

É importante lembrar que o método load só deve ser usado em dados JSON confiáveis e seguros, pois a desserialização de dados maliciosos pode representar um risco de segurança. É uma boa prática sempre validar e verificar os dados JSON antes de desserializá-los em objetos Python. Além disso, é importante garantir que o arquivo JSON esteja no formato correto antes de tentar carregá-lo usando o método load. Caso contrário, o método load pode gerar uma exceção JSONDecodeError.

Arquivos – parte deux

A função open() em Python é usada para abrir um arquivo e retorna um objeto file que pode ser usado para ler, escrever ou manipular o arquivo. O file objeto é uma instância da classe io.TextIOWrapper e fornece vários métodos para ler e escrever dados no arquivo. Entre esses métodos, estão os readline() e readlines(), que são usados para ler linhas de um arquivo.

readline()

O método readline() lê uma única linha do arquivo. Quando você chama readline(), ele lê a próxima linha do arquivo e retorna como uma string. Cada chamada subsequente de readline() lerá a próxima linha até que o final do arquivo seja alcançado. A sintaxe básica do readline() é a seguinte:

```python
"""módulo main.py"""
if __name__ == '__main__':
    with open('arquivo.txt', 'r', encoding='utf-8') as arq:
        linha = arq.readline()
```

O argumento 'r' no modo de abertura indica que estamos abrindo o arquivo em modo de leitura. Dentro do bloco with, arq.readline() lê a primeira linha do arquivo 'arquivo.txt' e a atribui à variável linha. Cada vez que arq.readline() é chamado, a próxima linha do arquivo é lida.

readlines()

O método readlines() lê todas as linhas do arquivo e as retorna como uma lista de strings. Cada elemento da lista é uma linha do arquivo. A sintaxe básica do readlines() é a seguinte

```python
"""módulo main.py"""
if __name__ == '__main__':
    with open('arquivo.txt', 'r', encoding='utf-8') as arq:
        linhas = arq.readlines()
```

arq.readlines() lê todas as linhas do arquivo 'arquivo.txt' e as retorna como uma lista. A lista de linhas é atribuída à variável linhas. Podemos iterar por cada elemento da lista de linhas para processá-las uma a uma.

É importante observar que, em ambos os métodos, os caracteres de nova linha (\n) são incluídos nas strings retornadas. Se você deseja remover esses caracteres, pode usar o método strip() da string para removê-los.

Por fim, é importante ressaltar que o uso do with garante que o arquivo será fechado automaticamente após o término do bloco de código, mesmo se ocorrer uma exceção. Isso garante que os recursos do sistema sejam liberados adequadamente.

Encoding

O parâmetro encoding da função open() em Python é usado para especificar o conjunto de caracteres que deve ser usado para ler ou escrever um arquivo. O valor padrão para o parâmetro encoding é None, o que significa que o Python usará o conjunto de caracteres padrão do sistema operacional para o arquivo.

No entanto, é uma boa prática especificar o conjunto de caracteres explicitamente, especialmente quando você está trabalhando com arquivos que contêm caracteres não-ASCII, como caracteres acentuados ou caracteres em outros alfabetos, como o chinês ou o japonês.

O parâmetro encoding aceita uma string que representa o nome do conjunto de caracteres. Alguns exemplos comuns de codificações são utf-8, latin-1, ascii, cp1252, iso-8859-1 e cp850.

```python
"""módulo main.py"""
import json


if __name__ == '__main__':
    with open('dados.json', 'r', encoding='utf-8') as arq:
        calculo = json.load(arq)

    print(calculo)
```

Caso não seja especificado um conjunto de caracteres que suporte os caracteres presentes no arquivo, será levantada uma exceção UnicodeDecodeError. Para evitar esse erro, é importante selecionar a codificação correta do arquivo.

Por fim, é importante lembrar que o parâmetro encoding também deve ser especificado ao escrever em um arquivo, caso contrário, o Python usará a codificação padrão do sistema operacional, o que pode não ser apropriado para os caracteres que você está escrevendo no arquivo.

.

Exercícios para Praticar

Exercícios sobre dumps e loads

1. Crie um dicionário Python com o nome e a idade de uma pessoa e converta-o em uma string JSON usando o método json.dumps.
2. Carregue uma string JSON com informações de uma música usando o método json.loads.
3. Crie uma lista Python com nomes de cores e converta-a em uma string JSON usando o método json.dumps.
4. Carregue uma string JSON com informações de um filme usando o método json.loads.
5. Crie um dicionário Python com informações de um livro e converta-o em uma string JSON usando o método json.dumps.
6. Carregue uma string JSON com informações de um carro usando o método json.loads.
7. Crie uma lista Python com informações de frutas e converta-a em uma string JSON usando o método json.dumps.
8. Carregue uma string JSON com informações de um jogo usando o método json.loads.
9. Crie um dicionário Python com informações de uma casa e converta-o em uma string JSON usando o método json.dumps.
10. Carregue uma string JSON com informações de uma cidade usando o método json.loads.
11. Crie uma lista Python com informações de animais e converta-a em uma string JSON usando o método json.dumps.
12. Carregue uma string JSON com informações de uma receita usando o método json.loads.
13. Crie um dicionário Python com informações de uma universidade e converta-o em uma string JSON usando o método json.dumps.
14. Carregue uma string JSON com informações de um programa de TV usando o método json.loads.
15. Crie uma lista Python com informações de países e converta-a em uma string JSON usando o método json.dumps.
16. Carregue uma string JSON com informações de uma loja usando o método json.loads.
17. Crie um dicionário Python com informações de um restaurante e converta-o em uma string JSON usando o método json.dumps.
18. Carregue uma string JSON com informações de um evento usando o método json.loads.
19. Crie uma lista Python com informações de livros e converta-a em uma string JSON usando o método json.dumps.
20. Carregue uma string JSON com informações de um museu usando o método json.loads.
21. Crie um dicionário Python com informações de um jogo de tabuleiro e converta-o em uma string JSON usando o método json.dumps.
22. Carregue uma string JSON com informações de um curso online usando o método json.loads.
23. Crie uma lista Python com informações de alimentos e converta-a em uma string JSON usando o método json.dumps.
24. Carregue uma string JSON com informações de um aplicativo usando o método json.loads.
25. Crie um dicionário Python com informações de um programa de rádio e converta-o em uma string JSON usando o método json.dumps.
26. Carregue uma string JSON com informações de um parque de diversões usando o método json.loads.
27. Crie uma lista Python com informações de cidades e converta-a em uma string JSON usando o método json.dumps.
28. Carregue uma string JSON com informações de um esporte usando o método json.loads.
29. Crie um dicionário Python com informações de uma pessoa, como nome, idade e cidade, e converta-o em uma string JSON usando o método json.dumps.
30. Carregue uma string JSON com informações de um livro usando o método json.loads e imprima apenas o título do livro.
31. Crie uma lista Python com números e converta-a em uma string JSON usando o método json.dumps.
32. Carregue uma string JSON com informações de um filme usando o método json.loads e imprima apenas o nome do diretor.
33. Crie um dicionário Python com informações de uma empresa, como nome e endereço, e converta-o em uma string JSON usando o método json.dumps.

34. Carregue uma string JSON com informações de um produto usando o método json.loads e imprima apenas o preço.
35. Crie uma lista Python com informações de carros e converta-a em uma string JSON usando o método json.dumps.
36. Carregue uma string JSON com informações de um programa de TV usando o método json.loads e imprima apenas o nome da emissora.
37. Crie um dicionário Python com informações de um filme, como título, ano de lançamento e gênero, e converta-o em uma string JSON usando o método json.dumps.
38. Carregue uma string JSON com informações de uma música usando o método json.loads e imprima apenas o nome do artista.
39. Crie uma lista Python com informações de jogos de videogame e converta-a em uma string JSON usando o método json.dumps.
40. Carregue uma string JSON com informações de um restaurante usando o método json.loads e imprima apenas o nome do chef.

Exercícios sobre dump, load e with

41. Escreva um dicionário Python em formato JSON em um arquivo chamado "dados.json" usando o método json.dump e a estrutura with para lidar com o arquivo.
42. Abra o arquivo "dados.json" em modo de leitura e use o método json.load para ler o conteúdo JSON e convertê-lo em um dicionário Python, usando a estrutura with.
43. Escreva uma lista Python em formato JSON em um arquivo chamado "lista.json" usando o método json.dump e a estrutura with para lidar com o arquivo.
44. Abra o arquivo "lista.json" em modo de leitura e use o método json.load para ler o conteúdo JSON e convertê-lo em uma lista Python, usando a estrutura with.
45. Escreva um conjunto Python em formato JSON em um arquivo chamado "conjunto.json" usando o método json.dump e a estrutura with para lidar com o arquivo.
46. Abra o arquivo "conjunto.json" em modo de leitura e use o método json.load para ler o conteúdo JSON e convertê-lo em um conjunto Python, usando a estrutura with.
47. Escreva um valor booleano Python em formato JSON em um arquivo chamado "booleano.json" usando o método json.dump e a estrutura with para lidar com o arquivo.
48. Abra o arquivo "booleano.json" em modo de leitura e use o método json.load para ler o conteúdo JSON e convertê-lo em um valor booleano Python, usando a estrutura with.
49. Escreva uma string Python em formato JSON em um arquivo chamado "string.json" usando o método json.dump e a estrutura with para lidar com o arquivo.
50. Abra o arquivo "string.json" em modo de leitura e use o método json.load para ler o conteúdo JSON e convertê-lo em uma string Python, usando a estrutura with.
51. Escreva um número inteiro Python em formato JSON em um arquivo chamado "inteiro.json" usando o método json.dump e a estrutura with para lidar com o arquivo.
52. Abra o arquivo "inteiro.json" em modo de leitura e use o método json.load para ler o conteúdo JSON e convertê-lo em um número inteiro Python, usando a estrutura with.
53. Escreva um número de ponto flutuante Python em formato JSON em um arquivo chamado "flutuante.json" usando o método json.dump e a estrutura with para lidar com o arquivo.
54. Abra o arquivo "flutuante.json" em modo de leitura e use o método json.load para ler o conteúdo JSON e convertê-lo em um número de ponto flutuante Python, usando a estrutura with.
55. Escreva um objeto Python com propriedades aninhadas em formato JSON em um arquivo chamado "objeto.json" usando o método json.dump e a estrutura with para lidar com o arquivo.
56. Crie um programa que escreva um dicionário em um arquivo JSON usando o método dump e with.
57. Crie um programa que leia um arquivo JSON usando o método load e with e imprima na tela o conteúdo lido.
58. Modifique o programa anterior para que ele leia um arquivo JSON e salve o conteúdo em um novo arquivo usando o método dump e with.
59. Crie um programa que leia um arquivo JSON e converta o conteúdo em um objeto Python usando o método load e with.

60. Crie um programa que leia um arquivo JSON contendo uma lista de números e imprima a soma dos números usando o método load e with.
61. Crie um programa que leia um arquivo JSON contendo uma lista de nomes e ordene a lista em ordem alfabética usando o método load e with.
62. Modifique o programa anterior para que ele salve a lista ordenada em um novo arquivo usando o método dump e with.
63. Crie um programa que leia um arquivo JSON contendo informações de estudantes (nome, idade, nota) e calcule a média de notas usando o método load e with.
64. Crie um programa que leia um arquivo JSON contendo informações de funcionários (nome, salário, cargo) e calcule a média salarial dos funcionários usando o método load e with.
65. Modifique o programa anterior para que ele salve a média salarial em um novo arquivo usando o método dump e with.
66. Crie um programa que leia um arquivo JSON contendo informações de produtos (nome, preço, quantidade em estoque) e calcule o valor total do estoque usando o método load e with.
67. Crie um programa que leia um arquivo JSON contendo informações de clientes (nome, endereço, telefone) e permita que o usuário busque um cliente pelo nome usando o método load e with.
68. Modifique o programa anterior para que ele salve o resultado da busca em um novo arquivo usando o método dump e with.
69. Crie um programa que leia um arquivo JSON contendo informações de filmes (título, diretor, ano de lançamento) e permita que o usuário busque um filme pelo título usando o método load e with.
70. Modifique o programa anterior para que ele salve o resultado da busca em um novo arquivo usando o método dump e with.
71. Crie um programa que leia um arquivo JSON contendo informações de livros (título, autor, ano de lançamento) e crie uma lista de autores usando o método load e with.
72. Crie um programa que leia um arquivo JSON contendo informações de músicas (título, artista, ano de lançamento) e crie uma lista de artistas usando o método load e with.
73. Crie um programa que leia um arquivo JSON contendo informações de países (nome, capital, população) e permita que o usuário busque um país pelo nome usando o método load e with.
74. Modifique o programa anterior para que ele salve o resultado da busca em um novo arquivo usando o método dump e with.
75. Crie um programa que leia um arquivo JSON contendo informações de usuários (nome, email, senha) e permita que o usuário faça login usando o método load e with.

Exercícios de Arquivo

76. Crie um arquivo de texto simples com algumas linhas de texto.
77. Abra o arquivo usando a função open() em modo de leitura e leia a primeira linha usando o método readline().
78. Abra o arquivo novamente e leia todas as linhas usando o método readlines().
79. Crie uma função que leia um arquivo de texto e retorne uma lista com todas as linhas.
80. Crie uma função que leia um arquivo de texto e retorne uma lista com todas as palavras contidas nele.
81. Crie uma função que leia um arquivo de texto e retorne uma lista com as palavras únicas contidas nele.
82. Crie uma função que leia um arquivo de texto e conte o número de ocorrências de cada palavra.
83. Crie uma função que leia um arquivo de texto e conte o número de caracteres em cada linha.
84. Crie uma função que leia um arquivo de texto e conte o número de palavras em cada linha.
85. Crie uma função que leia um arquivo de texto e conte o número total de palavras nele.
86. Crie uma função que leia um arquivo de texto e retorne a linha com o maior número de palavras.
87. Crie uma função que leia um arquivo de texto e retorne a linha com o menor número de palavras.
88. Crie uma função que leia um arquivo de texto e retorne a primeira palavra da primeira linha.
89. Crie uma função que leia um arquivo de texto e retorne a última palavra da última linha.
90. Crie uma função que leia um arquivo de texto e retorne uma lista de todas as palavras que começam com a letra 'a'.
91. Crie uma função que leia um arquivo de texto e retorne uma lista de todas as palavras que terminam com a letra 'o'.

92. Crie uma função que leia um arquivo de texto e retorne uma lista de todas as palavras que têm mais de 5 letras.
93. Crie uma função que leia um arquivo de texto e retorne uma lista de todas as palavras que têm menos de 5 letras.
94. Crie uma função que leia um arquivo de texto e retorne uma lista de todas as palavras que contêm a letra 'e'.
95. Crie uma função que leia um arquivo de texto e retorne uma lista de todas as palavras que contêm a letra 'q'.
96. Crie uma função que leia um arquivo de texto e retorne uma lista de todas as linhas que contêm a palavra 'python'.
97. Crie uma função que leia um arquivo de texto e retorne uma lista de todas as linhas que não contêm a palavra 'python'.
98. Crie uma função que leia um arquivo de texto e retorne uma lista de todas as linhas que começam com a letra 'A'.
99. Crie uma função que leia um arquivo de texto e retorne uma lista de todas as linhas que terminam com o caractere '.'.
100. Crie uma função que leia um arquivo de texto e retorne uma lista de todas as linhas que contêm um número de telefone no formato (xxx) xxx-xxxx.