



# Raciocínio Computacional

## UNIDADE 07


### Exceções e arquivos

*Esta unidade tem por objetivo dar uma visão de aplicação dos conceitos necessários para persistir dados em arquivos, de forma a armazená-los. É de grande importância para qualquer desenvolvedor saber manipular arquivos, a fim de criar backups, consumir uma lista de alguma planilha ou qualquer outro motivo. Por isso, a maioria das linguagens de programação possui meios para essa manipulação.*

## Arquivos: parâmetros para ler e escrever

---

Há diversos modos de uso, como podemos ver na imagem a seguir:

#ParaTodosVerem 

Character	Meaning
'r'	Leitura
'w'	Escrita. Substitui o conteúdo do arquivo existente
'x'	Escrita. Retorna um erro caso o arquivo já exista
'a'	Escrita. Insere os novos dados no final do arquivo
'b'	Modo binário
't'	Modo de texto (padrão)
'+'	Atualizar. Tanto leitura quanto escrita

Autor

Para utilizar, devemos pensar no método ***open***, com o qual abrimos o arquivo.

1. *def abrindo\_arquivo():*
2. *arquivo = open("dados.txt",a)*
3. *# agora podemos ler ou escrever no arquivo*

## Arquivos: escrevendo em um arquivo

---

O segundo ponto importante é escrever em um arquivo; para isso, utilizaremos o comando básico ***write***.

1. *arquivo = open("contatos.txt", "a")*
2. *arquivo.write("Olá, mundo!")*

## Arquivos: lendo dados de arquivos

---

Além de escrever dados, devemos também saber ler os dados de forma mais performática. Para isso, temos vários métodos, sendo um deles o ***readline()***.

1. *def ler\_caracteres\_primeira\_linha (numero):*
2. *arquivo = open("texto.txt", "r")*
3. *return (arquivo.readline(numero))*
4. *print (ler\_caracteres\_primeira\_linha(10))*

O método de exemplo lê um número de caracteres da primeira linha.

## Arquivos: lendo todo o arquivo

---

Agora, veremos o método ***readlines()***, que lê todo o arquivo.

1. *def ler\_arquivo (numero):*
2. *arquivo = open("texto.txt", "r")*
3. *return (arquivo.readlines())*
4. *print (ler\_arquivo())*

Então, para trabalhar com arquivos, não existe muito segredo. Temos de aprender alguns parâmetros e estruturas para utilizar, como em qualquer necessidade de entrada e saída de dados.

## Arquivos em binário

---

Para nosso estudo, arquivo binário é um arquivo cujo conteúdo está em um formato binário, que consiste em uma série de bytes sequenciais, normalmente cada um possuindo oito bits de comprimento. O conteúdo deve ser interpretado por um programa ou um processador de *hardware* que entenda antecipadamente a forma exata como esse conteúdo é formatado e como ler os dados.

Normalmente, o binário é utilizado quando não existe necessidade de interpretação pelo ser humano ou quando se necessita de desempenho. Os arquivos binários incluem uma ampla gama de tipos de arquivo, incluindo executáveis, bibliotecas, gráficos, bancos de dados, arquivos e outros, ou seja, são usados em aplicativos e outros tipos de *software*.

E como trabalhamos em Python com arquivos?

Simples, basta utilizar ‘r’ com os parâmetros de arquivo.

Veja a seguir um código para gerar arquivo. Vamos analisar com muito cuidado alguns aspectos importantes. Primeiramente, tem de ser entendido que, para tratar de arquivos, muitas vezes temos de importar uma biblioteca externa:

## *import pickle*

Neste caso, estamos importando a biblioteca Pickle.

Para escrever um objeto em um arquivo binário, usamos o método:

***pickle.dump (objeto, arquivo)***

**objeto:** este é o objeto a ser salvo em arquivo.

**arquivo:** esta é a variável associada a um arquivo previamente aberto em modo binário.

```
1. import pickle
2. try:
3. arquivo = open("teste.bin", "wb")
4. lista = [1, 2, 3]
5. pickle.dump(lista, arquivo)
6. arquivo.close()
7. except:
8. print("Problemas com o arquivo.")
```

**Importante:** todo acesso externo, que sai dos “domínios” de seu código, deve ser contornado por *try* e *except*, para tratar eventuais problemas de leitura e escrita.

Agora, vamos tratar de fazer o caminho contrário para ler o arquivo binário. Note que aqui utilizamos a função **load**.

```
1. import pickle
2. try:
3. arquivo = open("teste.bin", "rb")
4. l = pickle.load(arquivo)
5. print(l)
6. arquivo.close()
7. except:
8. print("Problemas com o arquivo.")
```

## | Arquivos JSON

O formato JSON é amplamente conhecido pelas pessoas desenvolvedoras mais

experientes. Embora seja mais conhecido por ser uma sintaxe de declaração de objetos originalmente inspirada e implementada na linguagem JavaScript, é também disponibilizado e utilizado na implementação de sistemas escritos em Python.

O JSON é um formato de arquivo, um modelo de estrutura de dados, muito utilizado no desenvolvimento de aplicações em boa parte das linguagens de programação atuais. Ele funciona no processamento e na manipulação dos mais diversos tipos de dado.

Em Python, especificamente, usar o JSON possibilita uma facilitação quando precisamos acessar e alterar valores e chaves em dicionários, por exemplo. Faz isso de maneira mais simplificada do que as disponibilizadas convencionalmente pela linguagem.

Para utilizar a função que implementa e possibilita o uso do modelo JSON Python, precisamos importar o pacote **json**. Confira, a seguir, como realizar esse procedimento e mais alguns detalhes!

1. *import json*
2. *with open('nomedoarquivo.json') as arquivo:*
3. *nomedoarquivo = json.load(arquivo)*

Para explicar como o Python opera os diversos tipos de dado possíveis em JSON, precisamos compreender um conceito básico da linguagem em relação a essa estrutura: a serialização, que define como é a transformação de dados em uma série de bytes – daí vem o “serial”, ou seja, o processo em que ocorre a conversão dos dados característicos da linguagem para que sejam armazenados.

Arquivos JSON não pertencem ao Python, mas são universais e, inclusive, muito utilizados para transferência de dados entre servidores e clientes, no caso de *browsers* na internet.

## Use o *parse* JSON em Python

---

*Parse* é uma palavra de origem inglesa, que pode ser definida como o processamento de apenas um pedaço de uma aplicação escrita em Python e a conversão desses códigos na linguagem de máquina.

De maneira geral, é possível dizer que *parse* é um comando usado para “dividir o código” do programa que escrevemos em um pequeno pedaço. Fazemos isso para que seja possível analisar se determinada sintaxe em JSON está correta para ser

manipulada pelo Python ou verificar se algum dado está presente no conjunto de elementos que temos disponíveis.

Confira, a seguir, um exemplo de *parse* realizado em Python, buscando um elemento dentro de um JSON, representado por um dicionário:

```
import json

# nosso dicionário

a = '{ "nome": "Elton", "idade": 32, "cidade": "Campinas" }'

# parse elemento "a":

y = json.loads(a)

# imprimindo o resultado, pedindo a idade do elemento do dicionário:

print(y["idade"])
```

## Videoaula

---

Para encerrar a unidade, assista abaixo a videoaula sobre Arquivos e JSON.

### Arquivos e JSON



## | Referências Bibliográficas

BANIN, S. L. **Python 3: conceitos e aplicações uma abordagem didática**. São Paulo: Érica, 2018.

PYTHON SOFTWARE FOUNDATION. **Documentação Python 3.9.0**. Disponível em: <https://docs.python.org/pt-br/3/>. Acesso em: 27 out. 2020.



© PUCPR - Todos os direitos reservados.