

# Programação Script

Arquivos

## Aula 15

Prof. Felipe A. Louza



- 1 Introdução
- 2 **Arquivos texto**
- 3 O método `write()`
- 4 O método `read()`
- 5 Acessos aleatórios
- 6 **Arquivos binários**
- 7 O módulo `pickle`
- 8 Exemplos
- 9 Referências

- 1 Introdução
- 2 Arquivos texto
- 3 O método `write()`
- 4 O método `read()`
- 5 Acessos aleatórios
- 6 Arquivos binários
- 7 O módulo `pickle`
- 8 Exemplos
- 9 Referências

# Tipos de Memória

Até agora, todos os programas que vimos obtiveram os dados de entrada **via teclado** e apresentaram resultados **na tela**.

```
1 n = input("Digite um número: ")  
2 print(resultado)
```

- Esses **programas** utilizam a **memória principal (RAM)** para armazenar dados.
- Quando o **programa termina** ou **acaba energia**, as informações são perdidas.



---

Essa tecnologia que requer **alimentação constante** de energia para que **informações** sejam preservadas.

# Tipos de Memória

Para gravar **informações** de forma *persistente*, devemos escrever em **arquivos** em uma **memória secundária**.

- Hard Disks, SSD, pendrive, ...



- **Vantagem:** capacidade (muito) maior de armazenamento.
- **Desvantagem:** é muito mais lenta do que a memória RAM.

---

Vamos apresentar de forma sucinta os conceitos de arquivos em **Python**.

# Tipos de Memória

Para gravar **informações** de forma *persistente*, devemos escrever em **arquivos** em uma **memória secundária**.

- Hard Disks, SSD, pendrive, ...



- **Vantagem:** capacidade (muito) maior de armazenamento.
- **Desvantagem:** é muito mais lenta do que a memória RAM.

---

Vamos apresentar de forma sucinta os conceitos de arquivos em **Python**.

# Tipos de Memória

Para gravar **informações** de forma *persistente*, devemos escrever em **arquivos** em uma **memória secundária**.

- Hard Disks, SSD, pendrive, ...



- **Vantagem:** capacidade (muito) maior de armazenamento.
- **Desvantagem:** é muito mais lenta do que a memória RAM.

---

Vamos apresentar de **forma sucinta** os conceitos de arquivos em **Python**.

# Nomes e extensões

Arquivos são identificados por **um nome**:

- Esse nome pode conter um **extensão** que indica o **conteúdo do arquivo**.

arq.txt	arquivo texto simples
arq.c	código fonte em C
arq.pdf	<i>portable document format</i>
arq*	arquivo <b>executável</b> (UNIX)
arq.exe	arquivo <b>executável</b> (WINDOWS)

Figura: Algumas extensões

Nessa aula, vamos assumir que os arquivos estão no **mesmo diretório** do script **Python**.



# Tipos de arquivos

Do **ponto de vista do programador** existem apenas dois tipos de arquivo:

- ❶ **Arquivo texto:** Armazena **caracteres** seguindo uma codificação (UTF-8, por exemplo)

```
1 "The quick brown fox jumps over the lazy dog" is an English-language
2 pangram a sentence that contains all of the letters of the alphabet.
3 ...
```

- ❷ **Arquivo binário:** Sequência de **bits** sujeita às convenções dos programas que o gerou, não legíveis diretamente.
  - Exemplos: **arquivos executáveis**, arquivos zip, documento PDF, ...

---

Na verdade, qualquer arquivo é uma **sequência de bits** com o seu conteúdo.

# Tipos de arquivos

Do **ponto de vista do programador** existem apenas dois tipos de arquivo:

- 1 **Arquivo texto:** Armazena **caracteres** seguindo uma codificação (UTF-8, por exemplo)

```
1 "The quick brown fox jumps over the lazy dog" is an English-language
2 pangram a sentence that contains all of the letters of the alphabet.
3 ...
```

- 2 **Arquivo binário:** Sequência de **bits** sujeita às convenções dos **programas que o gerou**, não legíveis diretamente.
  - Exemplos: **arquivos executáveis**, arquivos zip, documento PDF, ...

---

Na verdade, **qualquer arquivo** é uma **sequencia de bits** com o seu conteúdo.

- 1 Introdução
- 2 **Arquivos texto**
- 3 O método `write()`
- 4 O método `read()`
- 5 Acessos aleatórios
- 6 **Arquivos binários**
- 7 O módulo `pickle`
- 8 Exemplos
- 9 Referências

# Arquivos texto

O **primeiro passo** para trabalhar com arquivos é **abri-lo e associá-lo** com uma variável.

```
1 arq = open(filename, mode='r')
```

- A variável será **um objeto do tipo file** que **contém métodos** para **ler e escrever** no arquivo.

```
1 >>> arq = open("teste.txt", 'w')
2 >>> type(arq)
3 <class '_io.TextIOWrapper'>
4 >>> dir(arq)
```

# O método `open()`

Sobre os parâmetros:

```
1 arq = open(filename, mode='r') # 'r' é default
```

❶ **filename:**

- nome absoluto: `"/home/usr/teste.txt"`
- nome relativo: `"teste.txt"`

❷ **mode:**

Modo	Operações	Observações
"r"	leitura	(arquivo <b>deve</b> existir)
"r+"	leitura e escrita	
"w"	escrita	(sobrescreve o arquivo, se existir)
"a"	escrita	(acrescenta dados no fim do arquivo)
"x"	cria o arquivo	(arquivo <b>não</b> pode existir)

Figura: Modos de **abertura** de um arquivo

---

Podemos adicionar `"b"` para indicar "modo" binário (`"t"` de texto é *default*).

# O método `open()`

Sobre os parâmetros:

```
1 arq = open(filename, mode='r') # 'r' é default
```

❶ **filename:**

- nome absoluto: `"/home/usr/teste.txt"`
- nome relativo: `"teste.txt"`

❷ **mode:**

Modo	Operações	Observações
"r"	leitura	(arquivo <b>deve</b> existir)
"r+"	leitura e escrita	
"w"	escrita	(sobrescreve o arquivo, se existir)
"a"	escrita	(acrescenta dados no fim do arquivo)
"x"	cria o arquivo	(arquivo <b>não pode</b> existir)

Figura: Modos de **abertura** de um arquivo

---

Podemos adicionar `"b"` para indicar "modo" binário (`"t"` de texto é *default*).

# Arquivos texto

Depois de manipular um arquivo, sempre precisamos **fechar o arquivo** com o **método `close()`**:

```
1 >>> arq = open("teste.txt", 'x') # apenas cria um arquivo
2 >>> arq.name
3 'teste.txt'
4 >>> dir(arq)
5 ...
6 ...
7 >>> arq.close()
```

- 1 Este comando **garante** que os dados serão **efetivamente escritos** no arquivo.
- 2 Ele também **libera recursos** que são alocados para manter a associação da variável com o arquivo.

- 1 Introdução
- 2 Arquivos texto
- 3 O método `write()`
- 4 O método `read()`
- 5 Acessos aleatórios
- 6 Arquivos binários
- 7 O módulo `pickle`
- 8 Exemplos
- 9 Referências



# Escrevendo em um arquivo texto

Para escrever em um arquivo, ele deve **ser aberto** de forma apropriada usando o modo **"w"**, **"a"** ou **"r+"**.

```
1 >>> arq = open("teste.txt", 'w')
```

- O método **write()**:

```
1 >>> arq.write("Primeiro teste\n1 2 3\n")  
2 22
```

- Escreve o conteúdo da **string** no arquivo, retorna o **número de caracteres** escritos.

Não se esqueça de **fechar o arquivo** depois.

---

No modo **"r+"** o arquivo deve existir anteriormente.

# Escrevendo em um arquivo texto

Para escrever em um arquivo, ele deve **ser aberto** de forma apropriada usando o modo **"w"**, **"a"** ou **"r+"**.

```
1 >>> arq = open("teste.txt", 'w')
```

- O método **write()**:

```
1 >>> arq.write("Primeiro teste\n1 2 3\n")
2 22
```

- Escreve o conteúdo da **string** no arquivo, retorna o **número de caracteres** escritos.

Não se esqueça de **fechar o arquivo** depois.

---

No modo **"r+"** o arquivo deve existir anteriormente.

# Modo "a"

Adiciona conteúdo no **final do arquivo**:

```
1 >>> arq = open("teste.txt", "a")
2 >>> arq.write("Adicionei no fim do arquivo\n")
3 28
4 >>> arq.close()
```

- Arquivo "teste.txt":

```
1 $ more teste.txt
2 Primeiro teste
3 1 2 3
4 Adicionei no fim do arquivo
```

---

Se o arquivo **não existisse**, um **novo** seria criado.

## Modo "w"

Apaga todo **conteúdo anterior** e escreve um novo texto.

```
1 >>> arq = open("teste.txt", "w")
2 >>> arq.write("Arquivo novo do zero\n")
3 21
4 >>> arq.close()
```

- Arquivo "teste.txt":

```
1 $ more teste.txt
2 Arquivo novo do zero
```

---

Se o arquivo **não existisse**, um **novo** seria criado.

## Modo "r+"

Sobrescreve o arquivo do início (também permite leitura).

```
1 >>> arq = open("teste.txt", "r+")
2 >>> arq.write("==Alterado==")
3 12
4 >>> arq.close()
```

- Arquivo "teste.txt":

```
1 $ more teste.txt
2 ==Alterado== do zero
```

---

No modo "r+" o arquivo deve existir anteriormente.

# Escrevendo em um arquivo texto

Depois que um arquivo é fechado, as tentativas de usar o arquivo **falharão automaticamente**.

```
1 ...  
2 >>> arq.close()  
3 >>> arq.write("Novo texto")  
4 Traceback (most recent call last):  
5   File "<stdin>", line 1, in <module>  
6   ValueError: I/O operation on closed file.
```

- 1 Introdução
- 2 Arquivos texto
- 3 O método `write()`
- 4 O método `read()`
- 5 Acessos aleatórios
- 6 Arquivos binários
- 7 O módulo `pickle`
- 8 Exemplos
- 9 Referências

# Lendo de um arquivo texto

Para ler de um arquivo, ele deve ser aberto de forma apropriada usando o modo "r" ou "r+".

```
1 >>> arq = open("teste.txt", 'r')
```

- Utilizaremos os métodos `read()` e `readline()`.

Antes, vamos falar sobre possíveis erros.

---

Nesses dois modos o arquivo deve existir anteriormente.



# Lendo de um arquivo texto

Para ler de um arquivo, ele deve ser aberto de forma apropriada usando o modo "r" ou "r+".

```
1 >>> arq = open("teste.txt", 'r')
```

- Utilizaremos os métodos `read()` e `readline()`.

Antes, vamos falar sobre possíveis erros.

---

Nesses dois modos o arquivo **deve existir** anteriormente.

# Abrindo um arquivo

Se tentarmos abrir um arquivo **para leitura** ("**r**" ou "**r+**") e ele **não existir**, **ocorrerá um erro**:

```
1 >>> arq = open("notas-finais.txt", 'r')
2 Traceback (most recent call last):
3   File "<pyshell#6>", line 1, in <module>
4     arq = open("notas-finais.txt", 'r')
5 FileNotFoundError: [Errno 2] No such file or directory: 'notas-finais.txt'
```

# Abrindo um arquivo

Se tentarmos abrir um arquivo **para leitura** ("**r**" ou "**r+**") e ele **não existir**, **ocorrerá um erro**:

```
1 >>> arq = open("notas-finais.txt", 'r')
2 Traceback (most recent call last):
3   File "<pyshell#6>", line 1, in <module>
4     arq = open("notas-finais.txt", 'r')
5 FileNotFoundError: [Errno 2] No such file or directory: 'notas-finais.txt'
```

- Podemos tratar esse **tipo de erro** no **Python** usando os comandos **try-except**.

# Tratamento de exceções

Todo erro em **Python** gera o que chamamos de **exceção**:

- Por exemplo:

```
1 >>> 10 * (1/0)
2 Traceback (most recent call last):
3   File "<stdin>", line 1, in <module>
4 ZeroDivisionError: division by zero
5 >>>
6 >>> '2' + 2
7 Traceback (most recent call last):
8   File "<stdin>", line 1, in <module>
9 TypeError: Can't convert 'int' object to str implicitly
```

- Na verdade, é (sempre) bom tratar as exceções.

# Tratamento de exceções

Se sabemos que um trecho de código pode gerar uma exceção devemos usar a construção `try-except`.

```
1 try:
2     comandos1
3     comandos2
4     ...
5 except:
6     comandos executados se houver alguma exceção
```

- Dentro do bloco `try`, se ocorrer uma exceção (erro), **automaticamente** os comandos do bloco `except` são executados.

# Tratamento de exceções

Um exemplo simples:

```
1 >>> x = int(input("Digite um número: "))
2 dez
3 Traceback (most recent call last):
4   File "<pyshell#73>", line 1, in <module>
5     x = int(input())
6 ValueError: invalid literal for int() with base 10: 'dez'
```

```
1 try:
2     x = int(input("Digite um número: "))
3     print(x**2)
4 except:
5     print("Formato de número inválido")
```

# Abrindo um Arquivo

Ao trabalhar com **arquivos** é bom colocar a **abertura do arquivo** no **bloco try**, e o tratamento da exceção no **bloco except**.

```
1 try:
2     arq = open("notas-finais.txt", "r")
3     print("Abri arquivo com sucesso.")
4 except:
5     print("Não foi possível abrir o arquivo.")
```

# Lendo de um arquivo texto

Primeiro **abrimos** o arquivo:

```
1 >>> try:
2     arq = open("teste.txt", "r")
3     except:
4         print("Não foi possível abrir o arquivo.")
```

- O método **read()**:

```
1 >>> arq.read()
2 '==Alterado== do zero\n'
```

- Sem parâmetro é retornado uma **string** com **todo o arquivo!**



# Lendo de um arquivo texto

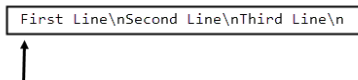
- O método `read(n)`:

```
1 >>> try:
2     arq = open("teste.txt", "r")
3 except:
4     print("Não foi possível abrir o arquivo.")
5
6 >>> arq.read(1)
7 '='
8 >>> arq.read(1)
9 '='
10 >>> arq.read(2)
11 'Al'
12 >>> arq.read(3)
13 'ter'
```

- `read(n)`: retorna uma `string` com os *próximos* `n` bytes do arquivo.

# Lendo de um arquivo texto

Quando um **arquivo** é aberto, um **indicador de posição** é criado, e **posicionado** no **início do arquivo**.

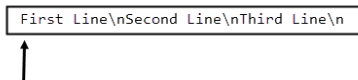


- Para cada **dado lido** do arquivo, este indicador de posição é **automaticamente** incrementado.
- Quando o indicador chega ao **fim do arquivo** o método `read()` devolve um **string** vazia.

```
1 >>> arq.read()
2 'ado== do zero\n'
3 >>> arq.read()
4 ''
```

# Lendo de um arquivo texto

Quando um **arquivo** é aberto, um **indicador de posição** é criado, e **posicionado** no **início do arquivo**.

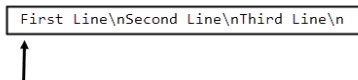


- Para cada **dado lido** do arquivo, este indicador de posição é **automaticamente** incrementado.
- Quando o indicador chega ao **fim do arquivo** o método `read()` devolve um **string** vazia.

```
1 >>> arq.read()
2 'ado== do zero\n'
3 >>> arq.read()
4 ''
```

# Lendo de um arquivo texto

Quando um **arquivo** é aberto, um **indicador de posição** é criado, e **posicionado** no **início do arquivo**.



A rectangular box containing the text "First Line\nSecond Line\nThird Line\n". Below the box, an upward-pointing arrow indicates the current file position at the beginning of the first line.

- Para cada **dado lido** do arquivo, este indicador de posição é **automaticamente** incrementado.
- Quando o indicador chega ao **fim do arquivo** o método `read()` devolve um **string** vazia.

```
1 >>> arq.read()
2 'ado== do zero\n'
3 >>> arq.read()
4 ''
```

# Programa `more.py`

O exemplo a seguir mostra o conteúdo do arquivo `"teste.txt"` na tela.

- O comando `read(1)`, lê 1 byte por vez.

```
1 try:
2     arq = open("teste.txt", "r")
3 except:
4     print("Não foi possível abrir o arquivo.")
5
6 s = None
7 while(s != ""):
8     s = arq.read(1)
9     print(s, end="")
10
11 arq.close()
```

```
1 $ python3 more.py
2 ==Alterado== do zero
```

## Programa `more.py` (versão 2)

O programa anterior pode ser alterado com comando `read()`, para ler **todo conteúdo** de **uma vez**.

```
1 try:
2     arq = open("teste.txt", "r")
3 except:
4     print("Não foi possível abrir o arquivo.")
5
6 s = arq.read()
7 print(s, end="")
8
9 arq.close()
```

- Mas cuidado, se o arquivo for **muito grande** pode **sobrecargar a memória** do seu computador: pode **ficar lento** ou mesmo **travar**.

## Programa `more.py` (versão 3)

Uma forma **melhor do que ler** um **byte por vez** é ler **uma linha** por vez com o comando `readline()`:

```
1 try:
2     arq = open("teste.txt", "r")
3 except:
4     print("Não foi possível abrir o arquivo.")
5
6 s = None
7 while(s != ""):
8     s = arq.readline()
9     print(s, end="")
10
11 arq.close()
```

- O `readline()` é menos arriscado do que se ler todo o arquivo de uma única vez.

## Programa `more.py` (versão 4)

Outra possibilidade para ler **linhas do arquivo** é iterar diretamente pelo **objeto arquivo**:

```
1 try:
2     arq = open("teste.txt", "r")
3 except:
4     print("Não foi possível abrir o arquivo.")
5
6 for linha in arq:
7     print(linha, end='')
8
9 arq.close()
```

- Esse código é equivalente à versão 3.



# Lendo de um arquivo texto

Podemos ainda ler **todas as linhas** de um arquivo **em uma lista**:

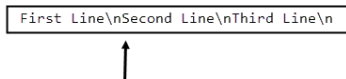
`list(arq)` ou `arq.readlines()`:

```
1 >>> try:
2     arq = open("teste.txt", "r")
3 except:
4     print("Não foi possível abrir o arquivo.")
5
6 >>> list(arq) # ou arq.readlines()
7 ...
8 >>> arq.read(1)
9 ''
```

- Lembrando que o **indicador de posição** é incrementado automaticamente.

## Lendo de um arquivo texto

No caso do `readline()` o **indicador de posição** é movido para a próxima linha.

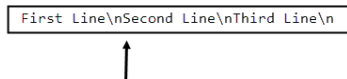


- Ao chegar no **fim do arquivo** o método também devolve um `string` vazia.

Para voltar ao início do arquivo você pode **fechá-lo e abri-lo** novamente, ou usar o método `seek(0)`.

## Lendo de um arquivo texto

No caso do `readline()` o **indicador de posição** é movido para a próxima linha.



- Ao chegar no **fim do arquivo** o método também devolve um `string` vazia.

Para voltar ao início do arquivo você pode **fechá-lo e abri-lo** novamente, ou usar o método `seek(0)`.

- 1 Introdução
- 2 **Arquivos texto**
- 3 O método `write()`
- 4 O método `read()`
- 5 **Acessos aleatórios**
- 6 **Arquivos binários**
- 7 O módulo `pickle`
- 8 Exemplos
- 9 Referências

# Acessos aleatórios

O método `seek()` permite re-posicionar o **indicador de posição**

First Line\nSecond Line\nThird Line\n



```
1 >>> arq.seek(offset, referencia='0')
```

- O **primeiro parâmetro** indica **quantos bytes** mover a partir de uma **referência**:
  - 0** : **início** do arquivo;
  - 1** : **posição** atual no arquivo;
  - 2** : **final** do arquivo.

# Acessos aleatórios

Com o método `seek()` podemos fazer **acessos aleatórios** ao arquivo:

```
1 try:
2     arq = open("teste.txt", "r+")
3 except:
4     print("Não foi possível abrir o arquivo.")
5
6 arq.seek(13) # move 13 bytes a partir do início do arquivo
7 s = arq.read()
8 print(s)
9
10 arq.seek(0) # move 0 bytes a partir do início
11 arq.write("##")
12
13 arq.seek(0) # move para o início do arquivo
14 s = arq.read()
15 print(s)
16
17 arq.close()
```

- Alteramos os 2 primeiros **caracteres** do arquivo.

# Acessos aleatórios

Em **arquivos texto** é permitido **seek()** relativo apenas ao **início do arquivo**.

```
1 >>> arq.seek(0)      # move para o início do arquivo
2 ...
3 >>> arq.seek(0, 2)
```

- Exceto **seek(0, 2)**: move para o final do arquivo.

First Line\nSecond Line\nThird Line\n



# Acessos aleatórios

O método `tell()` informa a posição do **indicador de posição** (em bytes):

```
1 >>> arq.tell()
```

- Podemos descobrir o tamanho de um arquivo:

```
1 try:
2     arq = open("teste.txt", "r+")
3 except:
4     print("Não foi possível abrir o arquivo.")
5
6 arq.seek(0, 2) # move para o fim-do-arquivo
7 b = arq.tell()
8 print(b, "bytes")
9
10 arq.close()
```



- 1 Introdução
- 2 **Arquivos texto**
- 3 O método `write()`
- 4 O método `read()`
- 5 Acessos aleatórios
- 6 **Arquivos binários**
- 7 O módulo `pickle`
- 8 Exemplos
- 9 Referências



# Motivação

Vamos pensar na seguinte situação:

- Para representar um **número inteiro** como **texto**, gastamos um número **variável de bytes**:

número	tamanho
10	2 bytes
1000	4 bytes
100000	6 bytes
...	...
2147483648	10 bytes

- Com **arquivos binários** podemos armazenar dados em arquivos de **forma análoga a utilizada na RAM** (4 bytes para um **inteiro**) .

# Arquivos binários em Python

Assim como em **arquivos texto**, também devemos abrir o **arquivo binário** com a função **open()**:

```
1 arq = open(filename, mode='rb')
```

Modos de acesso:

Modo	Operações	Observações
"rb"	leitura	(arquivo <b>deve</b> existir)
"r+b"	escrita e leitura	
"wb"	escrita	(sobrescreve o arquivo, se existir)

Figura: Modos de abertura de arquivo binário

---

"b" indica "modo" binário

# Arquivos binários em Python

Podemos usar os métodos vistos anteriormente para manipular **arquivos binários**.

- No entanto, precisamos *converter os dados* para **bytes**.

```
1 >>> try:
2     arq = open("teste.bin", "wb")
3 except:
4     print("Não foi possível abrir o arquivo.")
5
6 >>> numero = 100000
7 >>>
8 >>> arq.write(numero)
9 Traceback (most recent call last):
10   File "<stdin>", line 1, in <module>
11   TypeError: a bytes-like object is required, not 'int'
```

---

Métodos: **read()**, **write()**, **seek()**, **tell()**.

# Arquivos binários em Python

Para converter um **número inteiro** em **bytes**, podemos usar o método `to_bytes()`.

```
1 >>> numero = 100000
2 >>> bin = numero.to_bytes(4, byteorder='big', signed=True)
3 >>> print(bin)
4 b'\x00\x01\x86\xa0'
```

- O `'b'` antes dos dados indica que estamos lendo **bytes**.

# Arquivos binários em Python

Agora sim podemos **escrever** no arquivo binário:

```
1 >>> try:
2     arq = open("teste.bin", "wb")
3 except:
4     print("Não foi possível abrir o arquivo.")
5
6 >>> numero = 100000
7 >>> arq.write(numero.to_bytes(4, byteorder='big', signed=True))
8 4
9 >>> arq.close()
```

```
1 $ vim teste.bin
2 ^@^A<86>
```

---

O **'b'** antes dos dados indica que estamos lendo **bytes**.

# Arquivos binários em Python

Precisamos **conhecer as especificações** do **arquivo binário** para **ler os dados** de volta:

```
1 >>> arq.write(numero.to_bytes(4, byteorder='big', signed=True))
```

```
1 >>> try:
2     arq = open("teste.bin", "rb")
3 except:
4     print("Não foi possível abrir o arquivo.")
5
6 >>> x = arq.read(4)
7 >>> int.from_bytes(x, byteorder='big', signed=True)
8 100000
9 >>> arq.close()
```

- Precisamos saber também como ele **está organizado**.

---

Por isso, falamos que o formato do arquivo binário é **dependente de aplicação**.



- 1 Introdução
- 2 **Arquivos texto**
- 3 O método `write()`
- 4 O método `read()`
- 5 Acessos aleatórios
- 6 **Arquivos binários**
- 7 O módulo `pickle`
- 8 Exemplos
- 9 Referências

# Arquivos binários em Python: `pickle`

O Python possui uma biblioteca chamada **pickle** para escrever objetos em formato binário:

```
1 import pickle
```

- Não precisaremos nos preocupar em como os `bytes` representam os **objetos** nos arquivos.

---

Existem [outras bibliotecas](#), como as que escrevem objetos no **formato JSON**, que é um formato comum para trocas de informações.

# Arquivos binários em Python: pickle

O método `pickle.dumps()` recebe um objeto como parâmetro e retorna uma representação binária:

```
1 >>> import pickle
2 >>> pickle.dumps(100000)
3 b'\x80\x04\x95\x06\x00\x00\x00\x00\x00\x00J\xa0\x86\x01\x00.'
```

- O formato **não é óbvio** para **leitores humanos**; o objetivo é que **seja fácil** para o **pickle** interpretar.

```
1 >>> x = pickle.dumps(100000)
2 >>> y = pickle.loads(x)
3 >>> y
4 100000
```

# Arquivos binários em Python: pickle

- Podemos serializar listas com o `pickle.dumps()`:

```
1 >>> import pickle
2 >>> l1 = [1, 2, 3]
3 >>> x = pickle.dumps(l1)
4 b'\x80\x04\x95\x0b\x00\x00\x00\x00\x00\x00\x00]\x94(K\x01K\x02K\x03e.'
5 >>>
6 >>> l2 = pickle.loads(x)
7 >>> l2
8 [1, 2, 3]
```

# Arquivos binários em Python: pickle

Podemos converter e escrever (diretamente) um objeto em um **arquivo binário** com o método `pickle.dump()`

```
1 pickle.dump(objeto, arq)
```

- Exemplo:

```
1 import pickle
2 try:
3     arq = open("teste.bin", "wb")
4 except:
5     print("Problemas com o arquivo teste.bin")
6
7 lista = [1, 2, 3]
8 pickle.dump(lista, arq)
9
10 arq.close()
```

# Arquivos binários em Python: pickle

Para **ler um objeto** de um **arquivo binário** usamos o método `pickle.load()`.

```
1 varivel = pickle.load(arq)
```

- Exemplo:

```
1 import pickle
2 try:
3     arq = open("teste.bin", "rb")
4 except:
5     print("Problemas com o arquivo teste.bin")
6
7 l = pickle.load(arq)
8 print(l)
9
10 arq.close()
```

- O método automaticamente reconhece o tipo de objeto salvo em `arq`, **carrega** este para a memória e atribui para a `variavel`.

- 1 Introdução
- 2 **Arquivos texto**
- 3 O método `write()`
- 4 O método `read()`
- 5 Acessos aleatórios
- 6 **Arquivos binários**
- 7 O módulo `pickle`
- 8 **Exemplos**
- 9 Referências

## Exemplo 1: Lista aleatória (arquivo texto)

Escreva um programa em **Python** que recebe *via linha de comando* um inteiro  $N$ , e gera uma **lista aleatória** com  $N$  inteiros no intervalo  $[0, N)$ . Depois, salve essa lista em um **arquivo texto**:

```
1 $ python3 ex1.py 1000
2 ????
```

- Imprima na tela o tamanho (em bytes) do arquivo gerado



# Exemplo 1: Lista aleatória (arquivo texto)

Relembrando o módulo `sys`:

```
1 import sys
2
3 def main():
4     if(len(sys.argv) != 2):
5         print("Digite o valor de N")
6     else:
7         N = int(sys.argv[1])
8         try:
9             arq = open("random.txt", mode='w')
10        except:
11            print("Erro ao abrir o arquivo!")
12        ....
13        ....
14        arq.close()
15
16 if __name__ == "__main__":
17     main()
```

# Exemplo 1: Lista aleatória (arquivo texto)

Relembrando o módulo `sys`:

```
1 import sys
2
3 def main():
4     if(len(sys.argv) != 2):
5         print("Digite o valor de N")
6     else:
7         N = int(sys.argv[1])
8         try:
9             arq = open("random.txt", mode='w')
10        except:
11            print("Erro ao abrir o arquivo!")
12        ....
13        ....
14        arq.close()
15
16 if __name__ == "__main__":
17     main()
```

## Exemplo 1: Lista aleatória (arquivo texto)

Função que gera a lista aleatória:

```
1 def lista_aleatoria(N):  
2     import random  
3     l = []  
4     for i in range(N):  
5         l.append(random.randint(0, N))  
6     return l
```

## Exemplo 1: Lista aleatória (arquivo texto)

Escreve no arquivo texto e informa o tamanho:

```
1 def main():
2     if(len(sys.argv) != 2):
3         print("Digite o valor de N")
4     else:
5         N = int(sys.argv[1])
6         try:
7             arq = open("random.txt", mode='w')
8         except:
9             print("Erro ao abrir o arquivo!")
10
11     lista = lista_aleatoria(N)
12     for item in lista:
13         arq.write(str(item)+"\n")
14
15     print(arq.tell(), "bytes")
16     arq.close()
```

## Exemplo 2: Lista aleatória (arquivo binário)

Escreva um programa em **Python** que recebe *via linha de comando* o nome de um **arquivo texto** que contém números inteiros (**um por linha**). Salve os números lidos em um **arquivo binário**:

```
1 $ python3 ex2.py random.txt
2 ????
```

- Imprima na tela o tamanho (em bytes) do arquivo gerado

## Exemplo 2: Lista aleatória (arquivo binário)

```
1 def main():
2     if(len(sys.argv) != 2):
3         print("Digite o nome do arquivo")
4     else:
5         try:
6             arq1 = open(sys.argv[1], mode='r')
7         except:
8             print("Erro ao abrir o arquivo (texto) !")
9
10        try:
11            arq2 = open("random.bin", mode='wb')
12        except:
13            print("Erro ao abrir o arquivo (binario)!")
14
15        lista = [int(x) for x in list(arq1)]
16        arq1.close()
17
18        pickle.dump(lista, arq2)
19
20        print(arq2.tell(), "bytes")
21        arq2.close()
```

## Exemplo 2: Lista aleatória (arquivo binário)

Comparando o tamanho dos arquivos:

```
1 $ python3 ex1.py 100000
2 589029 bytes
3 $ python3 ex2.py random.txt
4 369142 bytes
```

```
1 $ ls random.* -lah
2 -rw-rw-r--. 1 ... .. 361K Jun 11 08:55 random.bin
3 -rw-rw-r--. 1 ... .. 576K Jun 11 08:55 random.txt
```

Fim

Dúvidas?



- 1 Introdução
- 2 **Arquivos texto**
- 3 O método `write()`
- 4 O método `read()`
- 5 Acessos aleatórios
- 6 **Arquivos binários**
- 7 O módulo `pickle`
- 8 Exemplos
- 9 **Referências**

- ① Materiais adaptados dos slides do Prof. Eduardo C. Xavier, da Universidade Estadual de Campinas.
- ② [panda.ime.usp.br/pensepy/static/pensepy/index.html](http://panda.ime.usp.br/pensepy/static/pensepy/index.html)