Arquivos Binários e Serialização

CIÊNCIA DA COMPUTAÇÃO

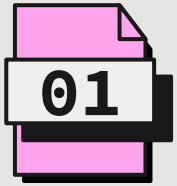
ATITUS EDUCAÇÃO

- Aula 21 -
Pensamento Computacional

Prof. Me. Lucas R. C. Pessutto

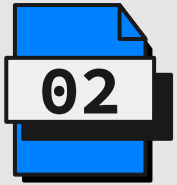
ATITUS EDUCAÇÃO

Na aula de hoje...



Arquivos Binários

Conceito
Salvando valores inteiros



Serialização de Objetos

Biblioteca `pickle`

Tipos de Arquivos

* Arquivos em Python podem ser utilizados em dois modos de processamento dos bytes que compõem o stream:
  - **Modo binário** (estruturado)
  - Modo texto (caracteres)

Arquivo Binário

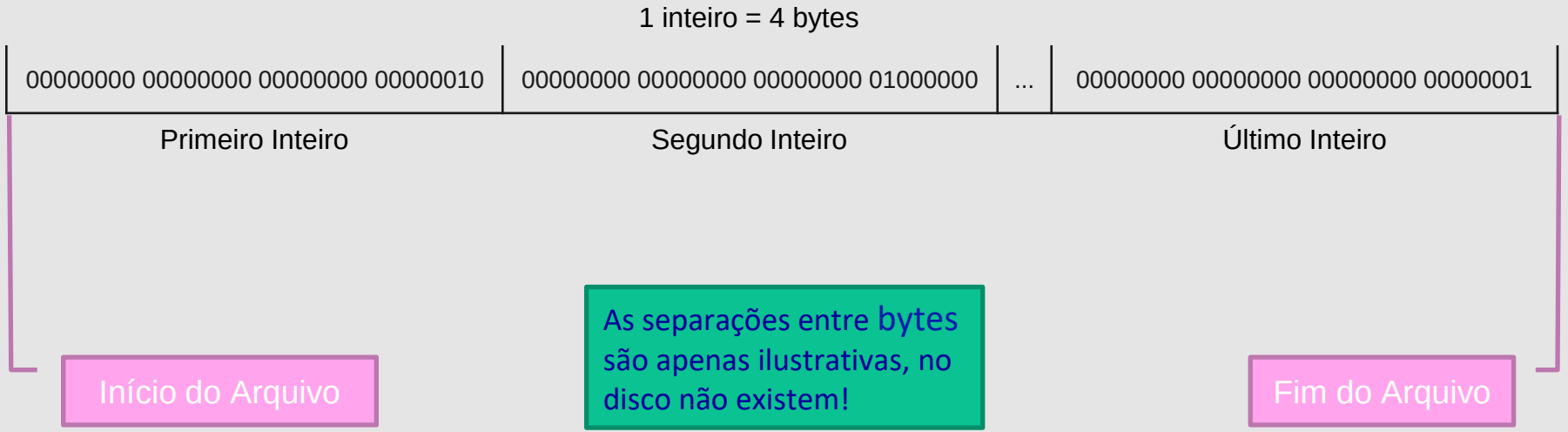
* Agregado sequencial de bytes
* Logicamente, o stream pode ser visto como um conjunto de dados simples ou estruturados
* Não tem limitação teórica de tamanho
* Comprimento do arquivo: número de elementos que apresenta no momento
* Elementos não tem nome
* Somente um elemento é acessível a cada momento: Elemento corrente
* Acesso aos dados: sequencial ou randômico.

O que está armazenado aqui?

FF	D8	FF	E1	1D	FE	45	78	69	66	00	00	49	49	2A	00
08	00	00	00	09	00	0F	01	02	00	06	00	00	00	7A	00
00	00	10	01	02	00	14	00	00	00	80	00	00	00	12	01
03	00	01	00	00	00	01	00	00	00	1A	01	05	00	01	00
00	00	A0	00	00	00	1B	01	05	00	01	00	00	00	A8	00
00	00	28	01	03	00	01	00	00	00	02	00	00	00	32	01
02	00	14	00	00	00	B0	00	00	00	13	02	03	00	01	00
00	00	01	00	00	00	69	87	04	00	01	00	00	00	C4	00
00	00	3A	06	00	00	43	61	6E	6F	6E	00	43	61	6E	6F
6E	20	50	6F	77	65	72	53	68	6F	74	20	41	36	30	00
00	00	00	00	00	00	00	00	00	00	00	00	B4	00	00	00
01	00	00	00	B4	00	00	00	01	00	00	00	32	30	30	34
3A	30	36	3A	32	35	20	31	32	3A	33	30	3A	32	35	00
1F	00	9A	82	05	00	01	00	00	00	86	03	00	00	9D	82
05	00	01	00	00	00	8E	03	00	00	00	90	07	00	04	00

Arquivo Binário: Armazenamento

Arquivo que armazena números inteiros:



Motivação

* 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 utiliza na RAM (todos os números inteiros são armazenados com 4 bytes)

Abertura do Arquivos

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

```
arquivo = open("notas.bin", "rb")
```

* O parâmetro opcional mode (no modo binário) pode ser:

  → "rb" para indicar abertura de arquivo para leitura (valor padrão)
    - Se o arquivo não existir, ocorrerá um erro de leitura

  → "r+b" para indicar abertura de arquivo para leitura e escrita
    - Se o arquivo não existir, ocorrerá um erro de leitura

  → "wb" para indicar abertura de arquivo para escrita
    - Se o arquivo já existir, este será sobrescrito

Arquivos Binários

* Podemos usar o métodos de leitura (read) e escrita (write) para manipular arquivos binários

```
numero = 1234
with open("teste.bin", "wb") as arq:
    arq.write(numero)
```

* No entanto, precisamos converter os dados para bytes!

```
Traceback (most recent call last):
  File "01-arquivo-escrita-errado.py", line 3, in <module>
    arq.write(numero)
TypeError: a bytes-like object is required, not 'int'
```

Arquivos Binários

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

```
numero = 1234
bin = numero.to_bytes(4, byteorder="big", signed=True)
print(bin)
```

```
b'\x00\x00\x04\xd2'
```

* O ‘b’ antes dos dados indica que os valores são bytes.

Escrevendo dados no arquivo

```python
with open("numeros.bin", "wb") as arquivo:
    for i in range(5):
        num = int(input("Informe um número: "))
        num_bin = num.to_bytes(4, byteorder="big", signed=True)
        arquivo.write(num_bin)
```

```
Informe um número: 8
Informe um número: 256
Informe um número: 9
Informe um número: -12
Informe um número: 66
```

Lendo dados do arquivo

```
with open("numeros.bin", "rb") as arquivo:
    for i in range(5):
        num_bin = arquivo.read(4)
        numero = int.from_bytes(num_bin,  byteorder="big", signed=True)
        print(f"Li o valor {numero}")
```

```
Li o valor 8
Li o valor 256
Li o valor 9
Li o valor -12
Li o valor 66
```

Problema 1: Arquivo de Números

Escreva um programa que leia três valores inteiros: `min`, `max` e `qtde`.

Seu programa deve escrever em um **arquivo binário** `qtde` números aleatórios no intervalo [`min`, `max`]

Em seguida, escreva outro programa que leia os valores que foram salvos no arquivo e apresente o maior valor que está salvo no arquivo.

Serialização de objetos

* A biblioteca `pickle` do Python permite guardar objetos diretamente em arquivos no formato binário

```
import pickle
```

* Utilizando essa biblioteca não é necessário nos preocupar em como representar os bytes nos objetos que serão salvos no arquivo.

Serialização de Objetos

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

```
import pickle

print(pickle.dumps(1234))
```

```
b'\x80\x04\x95\x04\x00\x00\x00\x00\x00\x00\x00M\xd2\x04.'
```

* Este formato **não é óbvio** para humanos, mas o objetivo aqui é ser fácil para que os métodos da biblioteca pickle interpretem os dados.

```
x = pickle.dumps(1234)
y = pickle.loads(x)
print(y)
```

```
1234
```

Serialização de Objetos

Podemos serializar qualquer tipo de dado, como uma lista usando o método `pickle.dumps()`:

```
import pickle

lista = [1, 2, 3]
x = pickle.dumps(lista)

print(x)

outra_lista = pickle.loads(x)
print(outra_lista)
```

```
b'\x80\x04\x95\x0b\x00\x00\x00\x00\x00\x00\x00]\x94(K\x01K\x02K\x03e.'
[1, 2, 3]
```

Salvando dados no arquivo

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

```
pickle.dump(<objeto>, <arquivo>)
```

* Exemplo:

```
import pickle

lista = [1, 2, 3]

try:
    with open("teste.bin", "wb") as arq:
        pickle.dump(lista, arq)
except:
    print("Problemas com a abertura do arquivo!")
```

Lendo dados do arquivo

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

```
<objeto> = pickle.load(<arquivo>)
```

* Exemplo:

```
import pickle

try:
    with open("teste.bin", "rb") as arq:
        lista = pickle.load(arq)
        print("Lido do arquivo:", lista)
except:
    print("Problemas com a abertura do arquivo!")
```

O método load reconhece automaticamente o tipo de objeto salvo em arq, **carrega** este para a memória e atribui para a variável lista.

Salvando Sequência de Dados

```
import pickle
ARQUIVO = "teste-muitos-dados.bin"
try:
    with open(ARQUIVO, "wb") as arq:
        pickle.dump([1, 2, 3], arq)
        pickle.dump("String de Teste", arq)
        pickle.dump(3.14159265358979323846, arq)
except:
    print("Problemas com a abertura do arquivo!")
else:
    print("Todos os dados foram salvos no arquivo...")
```

Salvando Sequência de Dados

```
try:
    with open(ARQUIVO, "rb") as arq:
        lista = pickle.load(arq)
        print(lista)
        string = pickle.load(arq)
        print(string)
        pi = pickle.load(arq)
        print(pi)
except:
    print("Problemas com a abertura do arquivo!")
```

Problema 1: Hashmon®

Enunciado de um problema:

Você resolveu criar um jogo de cartas chamado Hashmon®. No seu jogo, cada carta representa um personagem com as seguintes características: nome (`str`), tipo (grama, água ou fogo) (`str`), HP (quantidade de vida) (`int`) e fraqueza (os mesmos valores do tipo) (`str`). Para auxiliar na criação do baralho você resolve fazer um programa em Python onde você vai digitando iterativamente as características das cartas e o programa gera um aviso em duas situações:

1. caso você tente inserir uma carta com nome repetido
2. caso você tente inserir uma carta com as outras características repetidas

Seu programa deve permitir salvar e recuperar de um arquivo binário seu baralho de Hashmons

Problema 3: Hashmon®

Descrição dos Dados e Algoritmos

Chaves: nome (`str`) → **Valores:** características (`dict`)


```
baralho = {
    "Hashzard": {
        "hp": 20,
        "tipo": "fogo",
        "fraqueza": "grama"
    },
    "Squirrel":
    {...}
}
```