Arquivos

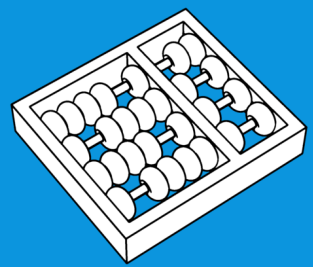
MC102 - Algoritmos e Programação de Computadores

06/24

24

Santiago Valdés Ravelo
https://ic.unicamp.br/~santiago/
ravelo@unicamp.br

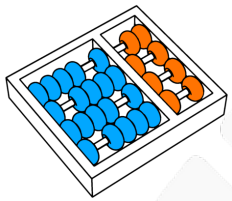
UNICAMP



“Você pode ter dados sem informação, mas não informação sem dados.”

Daniel Keys Moran

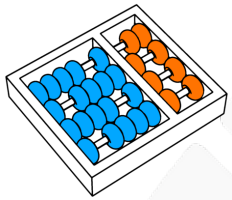
Dúvidas da aula anterior



Dúvidas selecionadas

- O maior problema que percebi nesta aula foi o raciocínio por trás pro código do que o código em si. Tem alguma maneira de aprimorar isso?
- Há uma maneira mais eficiente de resolver o problema de Hanoi?
- Embora o sistema de backtracking possivelmente resolva algum problema ele não obrigatoriamente o resolve da melhor forma possível certo? Por exemplo, no caso do cavalo embora ele chegue em uma conclusão correta, não seria possível, embora muito mais complicado, resolver esse problema "de uma vez"?
- Ainda não entendi muito bem o backtracking, Todo algoritmo recursivo que envolve uma lista que eventualmente tem algum de seus elementos apagados é um exemplo de backtracking?
- O que eu preciso analisar para saber se uma recursão funcionará corretamente?
- Como, usando backtracking escolher o melhor caminho? Apenas percorrendo todos e comparando?
- Se for possível resolver um problema com ou sem recursão, com eficiência parecida, eu devo priorizar recursão?
- Como é possível medir a complexidade de um algoritmo de backtracking?
- Qual é uma boa forma de pensar antes de resolver problemas difíceis de recursão?
- Há uma maneira de otimizar ainda mais o backtraking?
- Quais outros problemas interessantes resolvemos com backtracking?
- Para um número muito grande de discos na torre de hanoi, é possível ocorrer stack over flow? Nesse caso, como otimizar o código para que isso não aconteça?
- Imagino que quando mais complexo o algoritmo de recursão, mais possível de ter um problema de stack overflow. Tem alguma forma preventiva para não ter?

Introdução



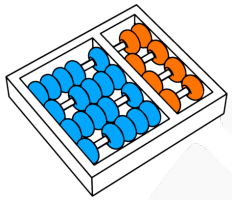
Arquivos

Arquivos são uma forma de armazenar dados:

- São blocos de dados armazenados permanentemente.
- Podem estar no HD, SD, pendrive, etc.
- E são acessados logicamente por um sistema de arquivos.

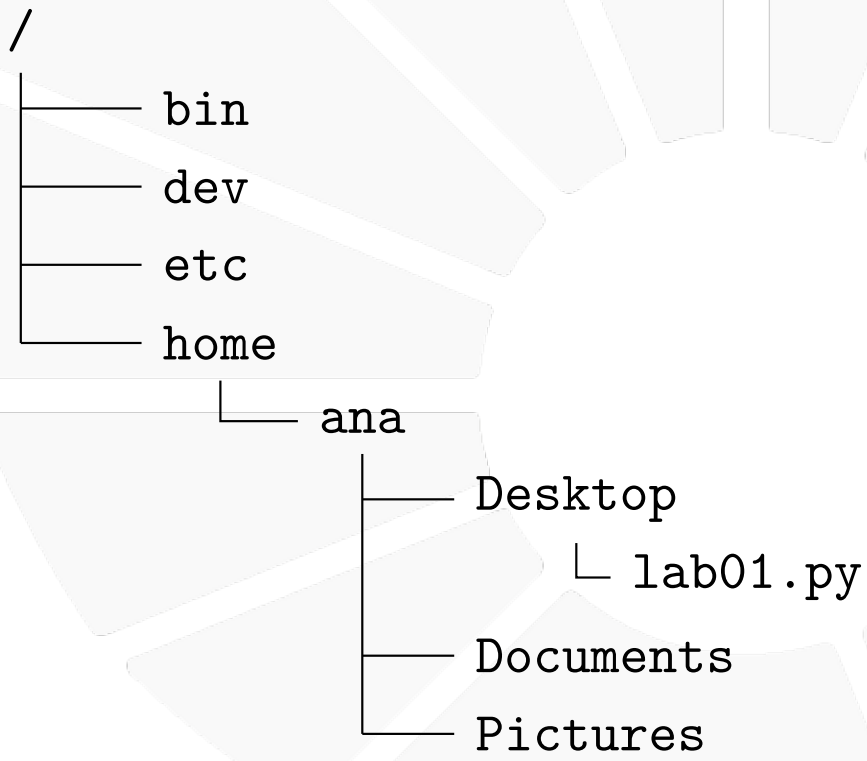
O sistema de arquivos é uma estrutura de acesso:

- Arquivos têm não apenas nomes, mas também um endereço.
- Os arquivos são armazenados em pastas (ou diretórios).
- Diretórios podem conter vários arquivos e vários diretórios.
- O sistema tem um ponto de origem, chamado de raiz:
  - Como se fosse um diretório que contém tudo.
  - No Windows, cada dispositivo tem a sua própria raiz.

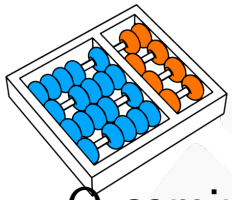


Exemplo de um sistema de arquivos

Exemplo de um sistema unix (linux, macOS, etc.)



Caminho: **/home/ana/Desktop/lab01.py**



Caminhos absolutos e relativos

O caminho **/home/ana/Desktop/lab01.py** é absoluto:

- Ele nos dá o caminho do arquivo desde a raiz **/**.
- No Windows, poderia ser algo do tipo: **C:\Usuarios\ana\Desktop\lab01.py**

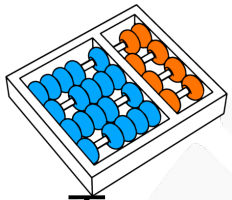
Podemos também ter caminhos relativos ao diretório atual:

- Se estamos em **/home/ana/**, podemos usar o caminho **Desktop/lab01.py**.
- Se estamos em **/home/ana/Desktop/**, podemos usar o caminho **lab01.py**.

Há também alguns atalhos:

- **.** indica o diretório atual.
- **..** indica o diretório pai.

O diretório atual do programa é o diretório onde o programa foi executado (não necessariamente onde o **.py** está).



Módulo **os**

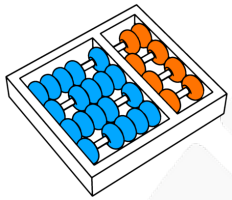
Temos diferenças entre os caminhos de acordo com o sistema operacional:

- Linux/MacOS: **/home/ana/Desktop/lab01.py**.
- Windows: **C:\Usuarios\ana\Desktop\lab01.py**.

O módulo **os** nos ajuda com isso:

- **os.sep** diz qual é o separador: **'/'** ou **'\'**.
- **os.path.join** é útil para construir um caminho:
  - **os.path.join('ana', 'Desktop', 'lab01.py')**.
  - Linux/MacOS: **ana/Desktop/lab01.py**.
  - Windows: **ana\Desktop\lab01.py**.
- Além de ter vários outros métodos úteis.

É uma boa prática de programação usar o **os** porque você programa independentemente do sistema operacional!

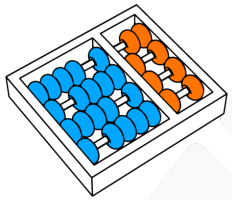


Outras coisas úteis de **os**

- **os.chdir**: Mudar o diretório atual.
- **os.mkdir**: Criar um diretório.
- **os.remove**: Remover um arquivo (cuidado...).
- **os.rename**: Renomear arquivo/diretório.
- **os.scandir**: Pegar os arquivos e diretórios de um diretório.
- **os.path.exists**: Verificar se um caminho existe.
- **os.path.isdir**: Verificar se um caminho representa um diretório.
- Entre muitos outros.

Só tome cuidado com o que você faz...

Leitura de arquivos



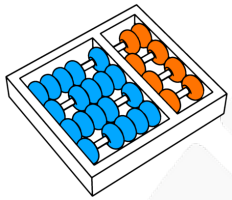
Tipos de leitura

O arquivo é lido do começo para o final (EOF — End of File):

- **arquivo.read()**: Lê todo restante do arquivo.
- **arquivo.read(k)**: Lê os próximos **k** caracteres do arquivo:
  - Se tiver menos do que **k**, lê menos.
- **arquivo.readline()**: Lê até a próxima quebra de linha (**\n**).

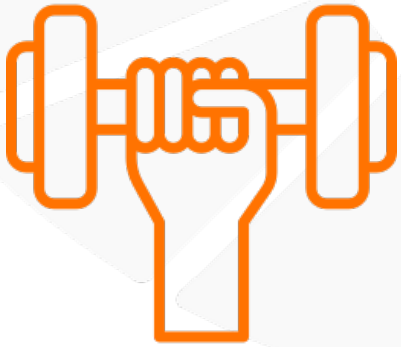
Também é possível ler o arquivo linha a linha usando **for**.

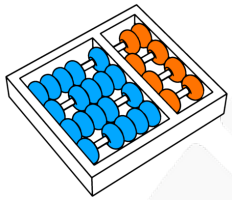
```
1 arquivo = open("arquivo.txt")
2 for linha in arquivo:
3     print(linha)
4 arquivo.close()
```



Sobre leitura de arquivos

Vamos fazer alguns exercícios?

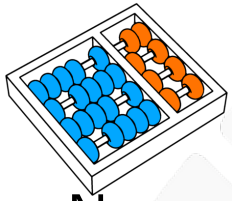




Exercícios

1. Faça um programa que lê uma lista de números de um arquivo, imprime a lista e sua média aritmética.
2. Faça um programa que lê um arquivo pbm (imagem em preto-e-branco) e armazena a imagem em uma matriz.

Escrita



Abrindo um arquivo para escrita

Novamente precisamos abrir o arquivo.

- Porém, precisamos informar para abrir para a escrita!

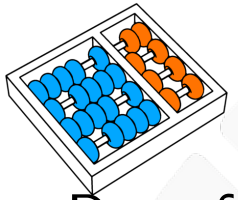
Alguns modos de abrir um arquivo:

'r'	read	leitura (padrão)
'w'	write	escrita (apaga o conteúdo atual)
'a'	append	acréscimo
'x'	new file	escrita apenas em arquivo novo

Ex: **f = open('arq.txt', 'w')**

Existem também formas de abrir um arquivo para leitura e escrita simultânea: **r+**, **w+**, **a+**:

- Nesse caso, você precisará andar pelo arquivo usando o método **seek**.



Escrevendo em um arquivo

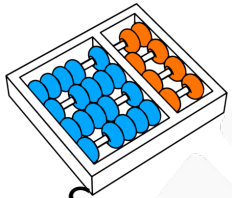
Duas formas de escrever:

- **arquivo.write(texto)**: recebe uma string **texto** e escreve no arquivo:
  - Não quebra linha automaticamente como o **print**.
- **arquivo.writelines(lista)**: escreve as strings de **lista** no arquivo:
  - Apesar do nome, não quebra linhas automaticamente...

```
1 f = open("arquivo.txt", "w")
2 f.write("Olá, Mundo!\n")
3 f.close()
```

Dados escritos podem ficar na memória até fechar o arquivo:

- São salvos no arquivo quando ele é fechado.
- Ou se você fizer **arquivo.flush()**.



Uma boa prática no Python

Se o seu programa tiver uma exception, o arquivo pode não ser fechado e:

- O conteúdo pode não ser salvo.
- Você pode atingir o limite de arquivos abertos.
- Seu programa pode ficar mais lento por consumo de memória.
- O sistema operacional pode impedir o acesso de outros programas ao arquivo.

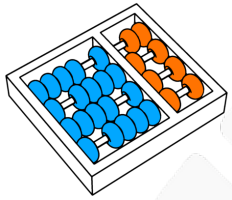
Você poderia usar **try...except** para isso...

- Mas tem um jeito mais fácil.

```
1 with open("arquivo.txt", "w") as f:
2     f.write("Olá, Mundo!\n")
```

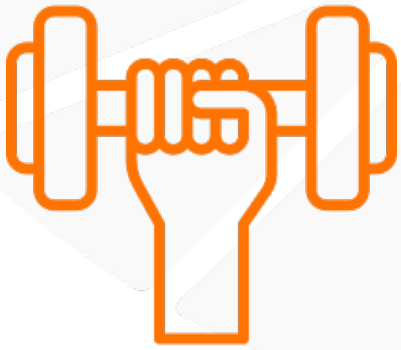
Quando o bloco acaba, o arquivo é fechado.

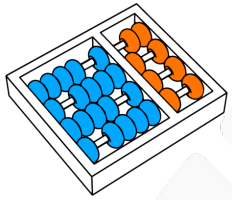
- Mesmo se houver exception.



Sobre escrita de arquivos

Vamos fazer alguns exercícios?

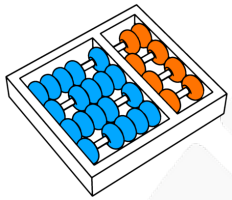




Exercícios

1. Faça uma função que salva os elementos de uma lista, um por linha, em um arquivo.
2. Faça uma função que, dada uma matriz, salva a matriz no formato pbm.

Encoding



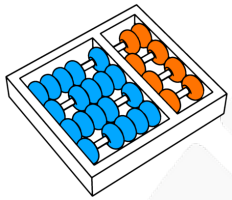
Encoding

Nós lemos e escrevemos strings em arquivos:

- Mas, como tudo no computador, o que temos são bytes. . .
- Quando escrevemos **'a'**, quais bytes são armazenados no arquivo?

Isso depende do **encoding** utilizado:

- É uma grande tabela, dizendo qual sequência de bits corresponde a um caractere.



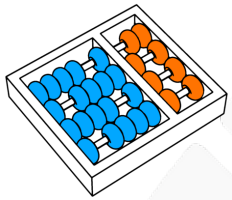
ASCII
American Standard Code for Information Interchange

O ASCII representa:

- Alfabeto latino (ou romano).
- Dígitos.
- Algumas pontuação e alguns símbolos.
- O necessário para escrever um texto em inglês.

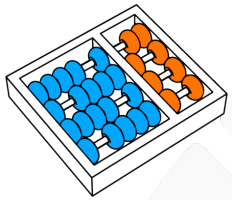
Utiliza um número de 0 a 127 para tanto.

- Um byte, sempre iniciado com zero.



A Tabela ASCII

0	nul	1	soh	2	stx	3	etx	4	eot	5	enq	6	ack	7	bel
8	bs	9	ht	10	nl	11	vt	12	np	13	cr	14	so	15	si
16	dle	17	dc1	18	dc2	19	dc3	20	dc4	21	nak	22	syn	23	etb
24	can	25	em	26	sub	27	esc	28	fs	29	gs	30	rs	31	us
32	sp	33	!	34	"	35	#	36	$	37	%	38	&	39	'
40	(	41	)	42	*	43	+	44	,	45	-	46	.	47	/
48	0	49	1	50	2	51	3	52	4	53	5	54	6	55	7
56	8	57	9	58	:	59	;	60	<	61	=	62	>	63	?
64	@	65	A	66	B	67	C	68	D	69	E	70	F	71	G
72	H	73	I	74	J	75	K	76	L	77	M	78	N	79	O
80	P	81	Q	82	R	83	S	84	T	85	U	86	V	87	W
88	X	89	Y	90	Z	91	[	92	\	93	]	94	^	95	_
96	`	97	a	98	b	99	c	100	d	101	e	102	f	103	g
104	h	105	i	106	j	107	k	108	l	109	m	110	n	111	o
112	p	113	q	114	r	115	s	116	t	117	u	118	v	119	w
120	x	121	y	122	z	123	{	124	|	125	}	126	~	127	del



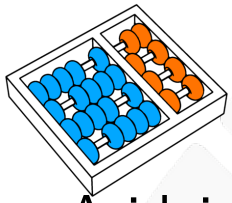
Outros encodings

Alguns encodings estendem ASCII usando valores até 255:

- São chamados **Extended ASCII**.
- Ex: ISO 8859-1 (latin-1):
  - Cobre línguas como português e espanhol.
  - Define vários caracteres acentuados: **'ç'**, **'é'**, **'ñ'**, etc.
  - Era o padrão usado no Brasil.
- Existem vários outros para várias línguas.

O problema é que podemos ter textos que misturem várias línguas:

- Ou não sabermos a língua original do texto e, com isso, não sabermos qual tabela usar...



UTF - Unicode Transformation Format

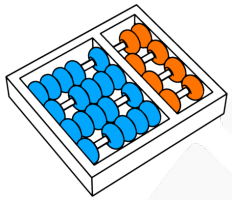
A ideia é resolver esses problemas usando um código único.

UTF-8:

- Utiliza de 1 a 4 bytes para representar os caracteres.
- É retrocompatível com ASCII.
- É o formato mais comum na internet hoje.
- É usado nas strings do Python.
- É econômico para textos em inglês.

UTF-16:

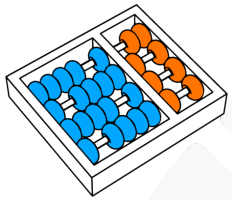
- Utiliza um ou dois blocos de 16 bits.
- Mais econômico para textos em línguas asiáticas.
- Menos usado.



Lidando com encodings no Python

Se você precisar usar outro encoding:

- Você precisará abrir o arquivo com o encoding correto.
- **f = open("arq.txt", "r", encoding="latin_1")**
- Na hora de abrir o arquivo, o Python utiliza o encoding padrão do sistema se nada for especificado.



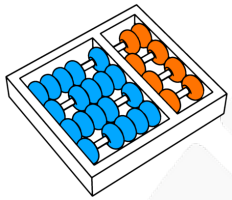
Duas funções úteis

ord: Devolve o código UTF-8 (em **int**) do caractere.

- Ex: **ord('a') == 97**.

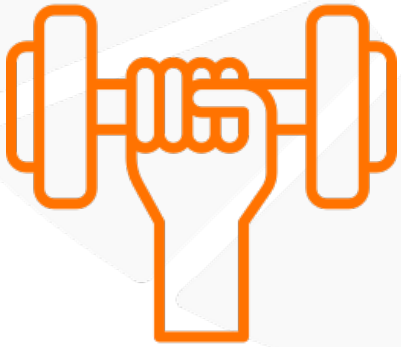
chr: Devolve o caracter correspondente ao código UTF-8.

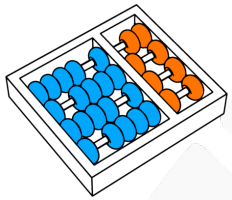
- Ex: **chr(97) == 'a'**



Sobre encoding

Vamos fazer alguns exercícios?

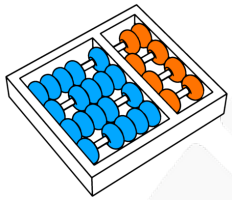




Exercícios

1. Dada uma string, devolva a lista de caracteres da string que representam dígitos.
2. Faça uma função que devolve o conjunto de todas as letras minúsculas latinas não acentuada.

Arquivos binários



Arquivos binários

Os arquivos que lemos e escrevemos até agora são arquivos texto:

- Sequência de bytes interpretada de acordo com um encoding.

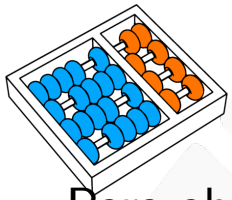
Porém, existem arquivos onde os valores dos bytes são simplesmente escritos:

- Arquivos que não representam texto.

Esses arquivos são chamados de arquivos binários.

Se tentarmos ler esses arquivos normalmente, podemos ter um erro:

- Ex: **UnicodeDecodeError** ao tentar ler um **.png**.



Abrindo, lendo e escrevendo em arquivos binários

Para abrir, precisamos indicar o modo adicional **b**:

- **rb**, **wb**, **ab**, **rb+**, etc.

Podemos ler, mas precisamos ler bytes:

- Usamos o tipo **bytes**.
- Podemos converter de bytes para string **bytes.decode** (em algum encoding).
- Podemos converter de **bytes** para **int** usando **int.from_bytes**.
- etc. . .

Podemos escrever, mas precisamos escrever bytes:

- Usamos o tipo **bytes**.
- Podemos converter string em bytes usando **str.encode** (em algum encoding).
- Podemos converter **int** em **bytes** usando **int.to_bytes**.
- etc. . .

Arquivos

MC102 - Algoritmos e Programação de Computadores

06/24

24

Santiago Valdés Ravelo
https://ic.unicamp.br/~santiago/
ravelo@unicamp.br

UNICAMP