

### Instituto de Computação

UNIVERSIDADE ESTADUAL DE CAMPINAS



## MC102 - Aula 17 Algoritmos de Busca

Algoritmos e Programação de Computadores



Prof. Lise R. R. Navarrete

lrommel@ic.unicamp.br

Quinta-feira, 19 de maio de 2022 19:00h - 21:00h (CB06)



### Instituto de Computação

AL DE CAMPINAS



**MC102** – Algoritmos e Programação de Computadores

Turmas OVXZ

https://ic.unicamp.br/~mc102/

Site da Coordenação de MC102

### Aulas teóricas:

Terça-feira, 21:00h - 23:00h (CB06) Quinta-feira, 19:00h - 21:00h (CB06)

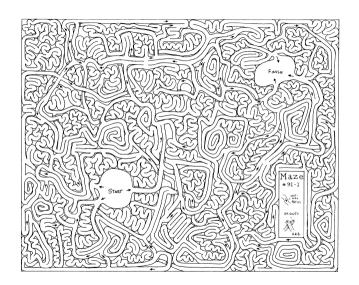
### Conteúdo

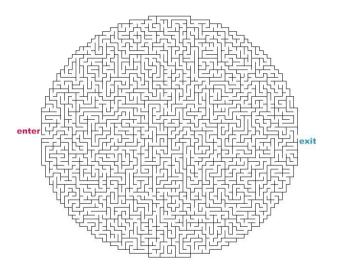
- O problema da Busca
- Busca Sequencial
- Eficiência da Busca Sequencial
- Busca Binaria
- Eficiência da Busca Binaria
- Exemplo comparativo
- Exercícios

# O problema da Busca



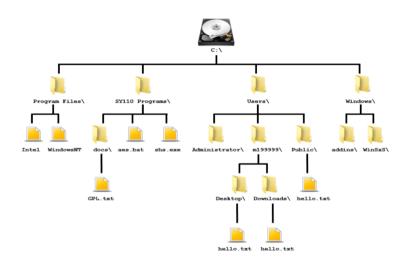








Recentes Pavorko	alphabet,	onds	data	prosite-AC- PA-dat	PSOCCOL. provide	PS00004.	PS00005. preside	PS00006. preside	PS00007. prosite	PSOCOOS. prosite	PS00009. preside	PS00000. prosite	PSCCCEL. provide	PS00012. prosite	PS00054. proste	PS00006.	PS00017. prosite	PSCOCES.	PSOCCES.	PSOCCO.	PSO0021. prosite	PS00022. preside	PS00023. prosite	PS00034. prosite	PS000 presi
tasto pessooil magens Josepa	PSCCC26.	PS-00027. prosite	PS00028. presite	PSO0023. proobs	PSOCESI. proofs	PS00032. prosite	PS00033. preste	PS00034. presite	PS00035. proofe	PSOCO36. prosite	PS00035. proobs	PS00041. prosite	PS0004Z. prosite	PSOCO45.	PSCCCAG.	PS00047. prosite	PS00048. prosite	PSCCC49.	PSCCOSO. proots	PS00051. prosite	P500052. presite	PS00053. preshe	PSC005-E. proofits	PS 00055. proofe	PSO pro
ocumento.	PSCGOST. prosite	PSOCOSB. prosite	PS00058. presite	PS00060. presite	PSOCOEL. prosite	PS00062. prosite	PS00063. preste	PS00064. presite	PS00065. prosite	PSOCOSS. prosite	PS00067. presite	PS00068. prosite	PS00069. prosite	PSOCC70. proste	PS00073. prosite	PS00072. prosite	PS00073. prosite	PS00074. prosite	PSCCC75. prosite	PSOCETE. prosite	PS00077. presite	PS00078. presite	PSCCC79. prosite	PSOCOD. prosite	PSC pr
inica dece	PSCCOR2.	PSOCOR3.	PS00084. preside	PSOCORS. prosite	PSCOORS. prosts	PS00067. prosite	PS00088.	PS00089. preside	PS00090. prosite	PSCOOSS. prosite	PS00092. preside	PS00083. prosite	PS00094. provibe	PS00095.	PSC0096.	PS00097. prosite	PSGOOSE. provine	PSC0099. proste	PS00000. provite	PSOSSEL proste	PSOCEE2.	PSOCEO3. prosite	PSOCSOL. prosite	PS00105. prosite	PS
Countrials	PS00007. prosts	PS00008. prosta	PSOCEPS. preside	PS00000. proshe	PSORIII. proste	P500012. prosite	PSOOLIS. preside	PSOCISA. presite	PS00115. provide	PSOCOS. proste	P900037. provite	PSOCES. provins	PSOCEIS. provine	PSOCE20. prosite	PS00125. Prosta	PSO0122. prosite	PSO0123. provine	PSOCE24. prosite	PS00125. prosite	PS00126. proste	PSO0127. preside	PSOCE28. proside	PSOCI29. prosite	PSOULSO. prouke	PSC
	P500132. proste	P500133. prioste	PSO0134. presite	PSOCE35. proste	PSOCE36. proofit	PSOBLET. prosite	PSO0138. preste	PSOCE39. presite	PSOCS40. prosite	PS0004L proste	P500142. preste	P500543. prosite	PS00144. provine	PSOULES. proste	PS00046. pNote	PS00549. prosite	PSO0150. prosite	PSOUSE. proste	PS00152. prosite	PSO0253. proste	PSOOLS4. preside	PSOCESS. prosite	PSOCISE. prosite	PSOUS7. proste	PS
	PSOCSSA. prosts	PSOOSES. prosite	PSOCSES. presite	PSOCS62. preshe	PSOCSES. prosite	PS00364. prosite	PSOCIES. preste	PSOCEEE. preshe	PSOCSEZ. prosite	PSCOSSIA. prosite	PSOSSES. presite	PSOX10. presite	PSOCE7S. provide	PSOCE74. prosite	PS00S75. proste	PSOOSHS. prosite	PSOCE77. presibe	PSOCEZIE. prosite	PSOCSBO. prosite	PSOOSEL proste	PSOCER2. preside	PSOCSA3. preshe	PSOCSB4. prosite	PSOCSAS. prosite	PS
	PSOCSEZ. prosite	PSOCOSE. prosite	PSOCERS. preside	PS00256. preshe	PSOCSS4. prosite	PS00195. prosite	PSOCING. presite	PSOCEST. preside	PSOCESE. proofe	PSOCOCE. prosite	PSOCCEZ. presite	P500003. presite	PS00204. provibe	PS00205. proste	Psocos. prosta	PS00307. prosite	PS00208. prosite	PS00209. proste	P500233. prosite	PSOSSIS. prosite	P500212. presite	PSO0213. preshe	PSO0254. prosite	PS00256. prosite	15
	PS00238. prosta	PS-00229. prosite	PS00230. preshe	PS00225. presite	PS00222. prosite	PS08223. prosite	PS00224. preste	PS00226. presite	PSO0227. prosite	PS00228. proste	PS00229. presite	P500210. presite	PS00231. prosibe	PS00232. proste	PS00233. prosta	P50023-6. presite	PS00235. presibe	PS00234. proste	PS00237. prosite	PS00236. prosite	PS00229. presite	PS00248. presite	PS00342. prosite	PS00243. prosite	75
	PS00245. prosts	PS00046. prouks	PS00247. preste	PS00248. proste	PSC0048.	P500250. proste	PSOCOSS. proute	PS00253. preshe	PS00253.	PS00254. proste	PS00255. proute	PS00256. proste	PS00257. provine	PS00258.	Psocosa.	P500060. prosits	PS00291. proube	PSOCOSO. proste	PSCEDES.	PS00064. prouke	PS00265. proube	PS00066. proste	PSOCOST. provite	PS00368. proute	PSI
	P500270. prosts	P500271. prosite	P500272. presite	PSO0273. proste	P500274. prosts	P500275. proute	P500276. proste	PS00277. preside	PSOCZ78.	PS00279. proste	P500280. prosts	PS00281. proste	PS00282.	PSOCORS.	PS00084.	P500085. prosite	P500386. provibe	PSOCOBJ.	PSCE288.	PSOCORS. prouke	P500290. proubs	PSO0295. proubs	PSO0292.	P500293. proste	PSI
	PSCE295. proofe	PS00096. proske	PSO0297. preside	PS00298. proste	PSOC299. proote	P500300. proste	PS00901. preste	PS00303. preshe	P500333. proste	PS00304. proste	PS00305. presite	P500386. proste	PS00307. prosite	P500308. proste	P500310. prosts	PS00311. produc	PS00352. prosite	PSOESES.	PS00314. prosite	P500315. proste	P500316. preste	PS00318. preshe	PSO03ES. proofie	PS00320. proste	PS
	P500322. proots	P500323. prooks	PS00334. presite	PS00326. proche	PS00327. proote	P500328. proste	P502329. preste	P500330. presite	PS00331. proote	P500332. proste	P500333. preste	P5:00335. proshe	P500336. prosite	P500337. proste	PS00338. prosts	PS00341. prooks	P500342. prosibe	PS00364. proote	PSC0345. prosite	PS003-66. prosite	PS00347. presite	PS00348. preshe	PS00349. proofix	PS00350. proote	PS
	PS000S2. prosta	PS00353. prosite	PS00354. presite	PS00355. preshe	PS0035G. prosite	PS00357. prosite	PS003S8. presite	PS00359. presite	PS00060. prosite	PSCODES.	PS00362. presite	PS00343. presita	P500364.	PS00365.	PS00066.	PS00367, presida	PS00368. preside	PS00069.	PS00370. prosite	PS00371. prosite	PS00372. presite	PS00373. preshe	PS00374. prosite	PS00375. prosite	PSI
	PS00377. presta	PS00378. prosite	PS00379. presite	PS00380. presão	PS00381. prosts	PS00382. prosite	P500363. preste	PS00384. preside	PS00385. proste	PS00387. proste	PS00388. preside	PS00389. prosite	PS00393. prosite	PS00390. proste	PS00392. prosta	P500393. presite	PS00384. prosibe	PS00395. proste	PSC0396. prosta	PS00397. prosite	PS00398. preste	PS00399. preshe	PSC0430. prostx	PS00005. prosite	PSI
	P506406. proste	P500407. proste	P500408. prosite	PS00409. proste	P500410. proste	P5004IL prosite	PS00412. prosite	P500413. prosite	PS00414. prosite	PS00425. proste	P500436. proste	PS004EX. prosite	PS00408. provibe	P500429. prosite	P500420. p1058#	P500421. prosite	PS00422. prosibe	P500423. proste	PS00424. proste	P500426. proste	P500427. prosite	PS00428. prosite	PS00429. prosite	P500430. proste	PS
	PSCOAN	P500411	PSONAN	PSONUS	PSONON	PSROATZ		PSOME	PSODAN	PSOCAN	PSOMA	PSONAS	PSODAS	PSODARZ	Promate	PSOOMS	PSOMS	PSONESI	PSCOAS2	PSODASS	PSONASA	P500457	PSONESA	P500450	



SUIRATTIGAS (E) CINEREBAMOCHA UMSCEPHEUSDLAARONIMSINACS NO I ROOIRGOE ANMESUPULCIRAC INX S V S A S R O I R C E T U S G S I R L I E U HOYAKUENCRATERTNAUCROSNLL PCPUSPECOIMSRALIHAAUSNAUP LETKSETOOBSUNROCIRPACASMT E R E S I L A R T S U A A N O R O C E L O T E U O DORXNYLOAAL MUVRIASGIPXCTR RSAAMRONULDUESECBEAAIESUO A E E I R A A A R B I L A N T P E R S E U S I C I CLOEBRSBUEAUTELRAPUSMGPSA OUUPEONOSUNGYCARIESIPPUPM LCUOUJERIDANUS SRONIMOELEA URLICAMELOPARDALISUIPROCS MEESHMWANTLIAAZNDENSEOBER BHUSASGLACERTARDYHAIEGNNU AYSAMINIMEGTAPHOENIXURCTR NDNCANESVENATICIDHATTIGAS A R A G E A O U A R I U S H U R S R A B I V Z U A CUTALCORVUSURGOMULUCITERM USCTELESCOPIUMRSDROTCIPUI THOROLOGIUMUEFORNAXHAILSN TRIANGULUMODARODISUCUIHPO A G I R U A O U I L A L U C E P L U V O L A N S R

Ara - Aries - Auriga - Boötes - Caelum - Camelopardalis - Cancer - Canes Venatici - Canis Major - Canis Minor - Capricornus - Carina - Casiopeia - Contaunts - Cephes - Cette - Schameleon - Circinus - Caloria - Carina -

#### O problema da Busca





#### O problema da Busca





· Vamos estudar alguns algoritmos para o seguinte problema:

### Definição do Problema

Dada uma chave de busca e uma coleção de elementos, onde cada elemento possui um identificador único, desejamos encontrar o elemento da coleção que possui o identificador igual ao da chave de busca ou verificar que não existe nenhum elemento na coleção com a chave fornecida.

- Nos nossos exemplos, a coleção de elementos será representada por uma lista de inteiros.
  - O identificador do elemento será o próprio valor de cada elemento.
- Apesar de usarmos inteiros, os algoritmos que estudaremos servem para buscar elementos em qualquer coleção de elementos que possuam chaves que possam ser comparadas.



- O problema da busca é um dos mais básicos na área de Computação e possui diversas aplicações.
  - · Buscar um aluno dado o seu RA.
  - · Buscar um cliente dado o seu CPF.
  - · Buscar uma pessoa dado o seu RG.
- Estudaremos algoritmos simples para realizar a busca assumindo que os dados estão em uma lista.
- Existem estruturas de dados e algoritmos mais complexos utilizados para armazenar e buscar elementos. Estas abordagens não serão estudadas nesta disciplina.



- · Vamos criar uma função busca(lista, chave):
  - A função deve receber uma lista de números inteiros e uma chave para busca.
  - A função deve retornar o índice da lista que contém a chave ou o valor -1, caso a chave não esteja na lista.



• No primeiro exemplo, a função deve retornar 5, enquanto no segundo exemplo, a função deve retornar —1.



## **Busca Sequencial**



- · A busca sequencial é o algoritmo mais simples de busca:
  - Percorra a lista comparando a chave com os valores dos elementos em cada uma das posições.
  - Se a chave for igual a algum dos elementos, retorne a posição correspondente na lista.
  - Se a lista toda foi percorrida e a chave n\u00e3o for encontrada, retorne o valor -1.



```
def buscaSequencial(lista, chave):
   indice = 0
   for número in lista:
   if número == chave:
      return indice
   indice = indice + 1
   return -1
```

```
def buscaSequencial(lista, chave):
    n = len(lista)
    for indice in range(n):
        if lista[indice] == chave:
            return indice

return -1
```

 Podemos usar também a função enumerate(lista), que retorna uma lista com tuplas da forma (índice, elemento).

```
def buscaSequencial(lista, chave):
   for (indice, número) in enumerate(lista):
      if número == chave:
        return indice
   return -1
```

```
def buscaSequencial(lista, chave):
    . . .
_{4} chave = 45
5 lista = [20, 5, 15, 24, 67, 45, 1, 76, 21, 11]
6
pos = buscaSequencial(lista, chave)
9 if pos != -1:
    print("Posição da chave", chave, "na lista:", pos)
11 else:
    print("A chave", chave, "não se encontra na lista")
# Posição da chave 45 na lista: 5
```

```
def buscaSequencial(lista, chave):
    . . .
\alpha chave = 100
s lista = [20, 5, 15, 24, 67, 45, 1, 76, 21, 11]
6
pos = buscaSequencial(lista, chave)
9 if pos != -1:
    print("Posição da chave", chave, "na lista:", pos)
11 else:
    print("A chave", chave, "não se encontra na lista")
# A chave 100 não se encontra na lista
```

# Eficiência da Busca Sequencial

### Complexidade da Busca Sequencial

- A busca sequencial é a estratégia de busca mais simples que existe.
- Assume-se que os dados não estão ordenados.
- No pior caso, percorre-se a estrutura que armazena os dados na sua totalidade, desde sua primeira posição até sua última posição.
- Como o custo deste processo é linear respeito ao número de elementos do "container" é chamada também de "busca linear".

- Na melhor das hipóteses, a chave de busca estará na posição 0.
   Portanto, teremos um único acesso em lista[0].
- Na pior das hipóteses, a chave é o último elemento ou não pertence à lista e, portanto, acessamos todos os n elementos da lista.
- É possível mostrar que, se as chaves possuirem a mesma probabilidade de serem requisitadas, o número médio de acessos nas buscas cujas chaves encontram-se na lista será igual a:

$$\frac{n+1}{2}$$

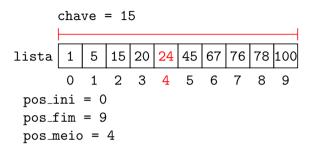


### **Busca Binaria**



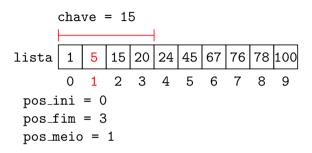
- A busca binária é um algoritmo mais eficiente, entretanto, requer que a lista esteja ordenada pelos valores da chave de busca.
- A ideia do algoritmo é a seguinte (assuma que a lista está ordenada pelos valores da chave de busca):
  - Verifique se a chave de busca é igual ao valor da posição do meio da lista.
  - · Caso seja igual, devolva esta posição.
  - Caso o valor desta posição seja maior que a chave, então repita o processo, mas considere uma lista reduzida, com os elementos do começo da lista até a posição anterior a do meio.
  - Caso o valor desta posição seja menor que chave, então repita o processo, mas considere uma lista reduzida, com os elementos da posição seguinte a do meio até o final da lista.





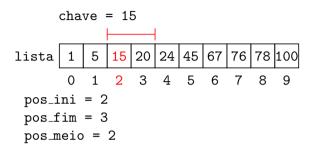
 Como lista[pos\_meio] > chave, devemos continuar a busca na primeira metade da região e, para isso, atualizamos a variável pos fim.





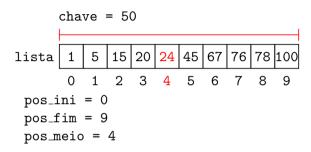
 Como lista[pos\_meio] < chave, devemos continuar a busca na segunda metade da região e, para isso, atualizamos a variável pos ini.





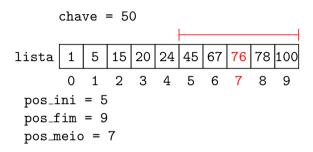
Finalmente, encontramos a chave (lista[pos\_meio] = chave) e, sendo assim, devolvemos a sua posição na lista (pos\_meio).





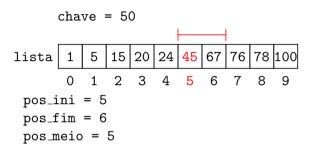
 Como lista[pos\_meio] < chave, devemos continuar a busca na segunda metade da região e, para isso, atualizamos a variável pos\_ini.





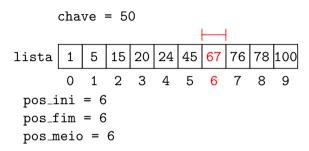
 Como lista[pos\_meio] > chave, devemos continuar a busca na primeira metade da região e, para isso, atualizamos a variável pos\_fim.





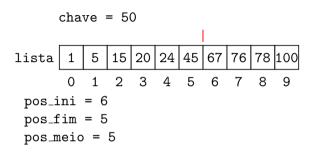
 Como lista[pos\_meio] < chave, devemos continuar a busca na segunda metade da região e, para isso, atualizamos a variável pos\_ini.





 Como lista[pos\_meio] > chave, devemos continuar a busca na primeira metade da região e, para isso, atualizamos a variável pos\_fim.





 Como pos\_ini > pos\_fim, determinamos que a chave n\u00e3o est\u00e1 na lista e retornamos o valor -1.



```
def buscaBinária(lista, chave):
    pos ini = 0
    pos fim = len(lista) - 1
    while pos ini <= pos fim:
      pos meio = (pos ini + pos fim) // 2
      if lista[pos meio] == chave:
8
        return pos meio
      if lista[pos_meio] > chave:
10
        pos fim = pos meio - 1
      if lista[pos meio] < chave:</pre>
        pos_ini = pos_meio + 1
14
    return -1
```

```
def buscaBinária(lista, chave):
    pos ini = 0
    pos fim = len(lista) - 1
    while pos ini <= pos fim:
      pos meio = (pos ini + pos fim) // 2
      if lista[pos meio] == chave:
8
        return pos meio
      if lista[pos_meio] > chave:
10
        pos fim = pos meio - 1
      else:
        pos ini = pos meio + 1
14
    return -1
```

```
def buscaBinária(lista. chave):
    . . .
_{4} chave = 15
5 # Para usar a busca binária a lista deve estar ordenada
6 lista = [1, 5, 15, 20, 24, 45, 67, 76, 78, 100]
pos = buscaBinária(lista, chave)
10 if pos != -1:
    print("Posição da chave", chave, "na lista:", pos)
12 else:
    print("A chave", chave, "não se encontra na lista")
14
# Posição da chave 15 na lista: 2
```

```
def buscaBinária(lista. chave):
_{4} chave = 50
5 # Para usar a busca binária a lista deve estar ordenada
6 lista = [1, 5, 15, 20, 24, 45, 67, 76, 78, 100]
pos = buscaBinária(lista, chave)
10 if pos != -1:
    print("Posição da chave", chave, "na lista:", pos)
12 else:
   print("A chave", chave, "não se encontra na lista")
14
# A chave 50 não se encontra na lista
```

### Eficiência da Busca Binaria



- Na melhor das hipóteses, a chave de busca estará na posição do meio da lista. Portanto, teremos um único acesso.
- Na pior das hipóteses, dividimos a lista até a que ela fique com um único elemento (último acesso realizado à lista).
- Note que, a cada acesso, o tamanho da lista é diminuído, pelo menos, pela metade.
- Quantas vezes um número pode ser dividido por dois antes dele se tornar igual a um?
- · Esta é exatamente a definição de logaritmo na base 2.
- Ou seja, no pior caso o número de acesso é igual a  $\log_2 n$ .
- É possível mostrar que, se as chaves possuirem a mesma probabilidade de serem requisitadas, o número médio de acessos nas buscas cujas chaves encontram-se na lista será igual a:

 $(\log_2 n) - 1$ 



45 / 54

## **Exemplo comparativo**



- Para se ter uma ideia da diferença de eficiência dos dois algoritmos, considere uma lista com um milhão de itens (10<sup>6</sup> itens).
- Com a busca sequencial, para buscar um elemento qualquer da lista necessitamos, em média, de:

$$(10^6 + 1)/2 \approx 500000$$
 acessos.

 Com a busca binária, para buscar um elemento qualquer da lista necessitamos, em média, de:

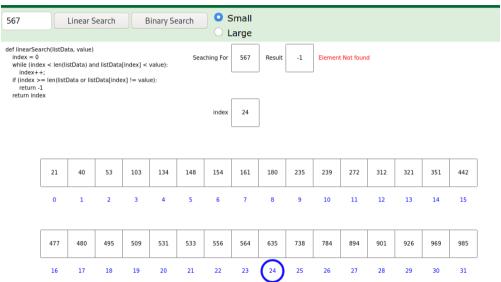
$$(\log_2 10^6) - 1 \approx 19$$
 acessos.



- Uma ressalva importante deve ser feita: para utilizar a busca binária, a lista precisa estar ordenada.
- Se você tiver um cadastro onde vários itens são removidos e inseridos com frequência e a busca deve ser feita de forma intercalada com essas operações, então a busca binária pode não ser a melhor opção, já que você precisará manter a lista ordenada.
- Caso o número de buscas seja muito maior que as demais operações de atualização do cadastro, então a busca binária pode ser uma boa opção.



#### https://www.cs.usfca.edu/~galles/visualization/Search.html



## **Exercícios**



 Refaça as funções de busca sequencial e busca binária assumindo que a lista possui chaves que podem ocorrer múltiplas vezes na lista. Neste caso, você deve retornar uma lista com todas as posições onde a chave foi encontrada. Se a chave não for encontrada na lista, retornar uma lista vazia.



- 2. Mostre como implementar uma variação da busca binária que retorne um inteiro k entre 0 e n, tal que, ou lista[k] = chave, ou a chave não se encontra na lista, mas poderia ser inserida entre as posições (k-1) e k de forma a manter a lista ordenada. Note que, se k = 0, então a chave deveria ser inserida antes da primeira posição da lista, assim como, se k = n, a chave deveria ser inserida após a última posição da lista.
- 3. Use a função desenvolvida acima para, dada uma lista ordenada de n números inteiros e distintos e dois outros inteiros X e Y, retornar o número de chaves da lista que são maiores ou iguais a X e menores ou iguais a Y.



# Perguntas ....



### Referências

 Zanoni Dias, MC102, Algoritmos e Programação de Computadores, IC/UNICAMP, 2021. https://ic.unicamp.br/~mc102/ Aula Introdutória [ slides ] [ vídeo Primeira Aula de Laboratório [ slides ] [ vídeo Python Básico: Tipos, Variáveis, Operadores, Entrada e Saída [ slides ] [ vídeo ] Comandos Condicionais [ slides ] [ vídeo ] Comandos de Repetição [ slides ] [ vídeo ] Listas e Tuplas [ slides ] | vídeo Strings [ slides ] [ vídeo Dicionários [ slides ] [ vídeo ] Funções [ slides ] [ vídeo ] Objetos Multidimensionais [ slides ] [ vídeo ] Algoritmos de Ordenação [ slides ] [ vídeo ] Algoritmos de Busca [ slides ] [ vídeo ] Recursão [ slides ] [ vídeo Algoritmos de Ordenação Recursivos [ slides ] [ vídeo ] Arquivos [ slides ] [ vídeo Expressões Regulares [ slides ] [ vídeo ] Execução de Testes no Google Cloud Shell [ slides ] [ vídeo ] Numpy [ slides ] [ vídeo Pandas | slides | | vídeo Panda - Cursos de Computação em Python (IME -USP) https://panda.ime.usp.br/ Como Pensar Como um Cientista da Computação https://panda.ime.usp.br/pensepy/static/pensepy/ Aulas de Introdução à Computação em Python https://panda.ime.usp.br/aulasPython/static/aulasPython/ Fabio Kon, Introdução à Ciência da Computação com Python http://bit.ly/FabioKon/ Socratica, Python Programming Tutorials http://bit.ly/SocraticaPython/ Google - online editor for cloud-native applications (Python programming) https://shell.cloud.google.com/ w3schools - Python Tutorial https://www.w3schools.com/python/ Outros citados nos Slides

54 / 54