BCM0505-22 – Processamento da Informação

Vetores - Parte 2

Carla Negri Lintzmayer
carla.negri@ufabc.edu.br
http://professor.ufabc.edu.br/~carla.negri/

Outline

Operações em vetor em Python

Vetores e funções

Explicação longa

Voltando...

Mais exemplos de aplicações de vetores

Exercícios

Operações em vetor em Python

Percorrendo elementos de um vetor sequencialmente

```
1  frutas = ["banana", "uva", "pera"]
2  i = 0
3  while i < len(frutas):
4      print(frutas[i])
5      i += 1
```

Percorrendo elementos de um vetor sequencialmente

```
frutas = ["banana", "uva", "pera"]
i = 0
while i < len(frutas):
    print(frutas[i])
    i += 1
```

```
frutas = [3, 10, 5, 9]
i = 0
while i < len(frutas):
    print(frutas[i])
    i += 1
```

Percorrendo elementos de um vetor sequencialmente

```
frutas = ["banana", "uva", "pera"]
i = 0
while i < len(frutas):
    print(frutas[i])
    i += 1
```

```
frutas = [3, 10, 5, 9]
i = 0
while i < len(frutas):
    print(frutas[i])
    i += 1
```

```
frutas = ["banana", "uva", "pera"]
for i in range(len(frutas)):
    print(frutas[i])
```

Percorrendo elementos de um vetor sequencialmente

```
frutas = ["banana", "uva", "pera"]
i = 0
while i < len(frutas):
    print(frutas[i])
    i += 1
```

```
frutas = [3, 10, 5, 9]
i = 0
while i < len(frutas):
    print(frutas[i])
    i += 1
```

```
frutas = ["banana", "uva", "pera"]
for i in range(len(frutas)):
    print(frutas[i])
```

```
frutas = [3, 10, 5, 9]
for i in range(len(frutas)):
    print(frutas[i])
```

Percorrendo elementos de um vetor sequencialmente

Lembre-se que a construção `for` do Python faz com que a variável assuma o valor de cada elemento da sequência

- Agora sabemos que o nome correto da sequência é `list` (tipo vetor do Python)

Percorrendo elementos de um vetor sequencialmente

Lembre-se que a construção `for` do Python faz com que a variável assuma o valor de cada elemento da sequência

- Agora sabemos que o nome correto da sequência é `list` (tipo vetor do Python)

```
1  frutas = ["banana", "uva", "pera"]
2  for fruta in frutas:
3      print(fruta)
```

```
1  frutas = [3, 10, 5, 9]
2  for fruta in frutas:
3      print(fruta)
```

Vetores e funções

Apelido vs Cópia

```
frutas = ["mirtilo", "morango", "limão", "framboesa"]
vermelhas = frutas
vermelhas[2] = "cereja"
print("vermelhas:", vermelhas)
print("frutas:", frutas)
```

Apelido vs Cópia

```
frutas = ["mirtilo", "morango", "limão", "framboesa"]
vermelhas = frutas
vermelhas[2] = "cereja"
print("vermelhas:", vermelhas)
print("frutas:", frutas)
```

```
vermelhas: ['mirtilo', 'morango', 'cereja', 'framboesa']
frutas: ['mirtilo', 'morango', 'cereja', 'framboesa']
```

Algo estranho aconteceu aqui!

- Vamos tentar com um tipo numérico...

Apelido vs Cópia

```
1  x = 10
2  y = x
3  y = 5
4  print("y:", y)
5  print("x:", x)
```

Apelido vs Cópia

```
1  x = 10
2  y = x
3  y = 5
4  print("y:", y)
5  print("x:", x)
```

```
y: 5
x: 10
```

E agora???

Apelido vs Cópia

A semântica da atribuição de um elemento do tipo `list` funciona de forma distinta da atribuição de um elemento do tipo `int`, `float` e `str`.

- Precisamos estar atentos a essa distinção

Apelido vs Cópia

A semântica da atribuição de um elemento do tipo `list` funciona de forma distinta da atribuição de um elemento do tipo `int`, `float` e `str`.

- Precisamos estar atentos a essa distinção

Como funciona a atribuição de uma variável/valor do tipo `list`?

Apelido vs Cópia

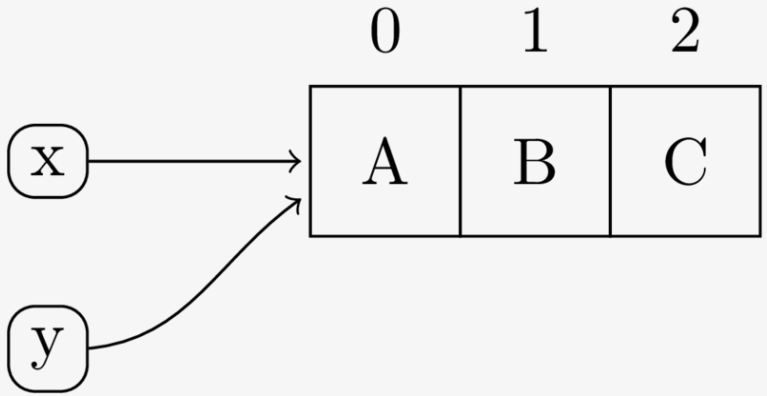
A semântica da atribuição de um elemento do tipo `list` funciona de forma distinta da atribuição de um elemento do tipo `int`, `float` e `str`.

- Precisamos estar atentos a essa distinção

Como funciona a atribuição de uma variável/valor do tipo `list`?

- **Resposta Curta:** Quando fazemos a atribuição de uma variável do tipo vetor a outra variável, estamos criando um apelido para o vetor: não é feita uma cópia, ambos os identificadores apontarão para a mesma sequência de dados.

```
x = ["A", "B", "C"]
y = x
```



Apelido vs Cópia

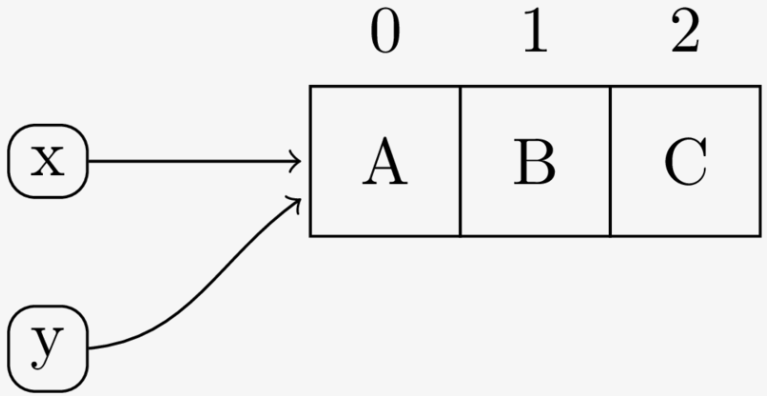
A semântica da atribuição de um elemento do tipo `list` funciona de forma distinta da atribuição de um elemento do tipo `int`, `float` e `str`.

- Precisamos estar atentos a essa distinção

Como funciona a atribuição de uma variável/valor do tipo `list`?

- **Resposta Curta:** Quando fazemos a atribuição de uma variável do tipo vetor a outra variável, estamos criando um apelido para o vetor: não é feita uma cópia, ambos os identificadores apontarão para a mesma sequência de dados.

```
x = ["A", "B", "C"]
y = x
```



- **Resposta Longa:** Senta que lá vem história...

Explicação longa

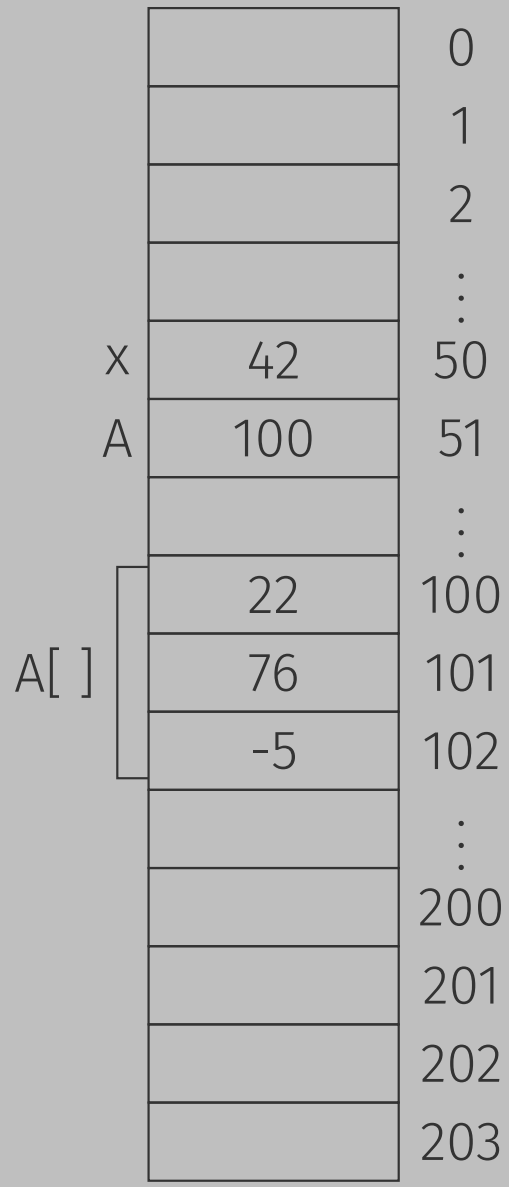
`list` por baixo do capô

Essa diferença no comportamento se deve pela forma como a linguagem lida com vetores por baixo do capô.

```
x = 42
A = [22, 76, -5]
```

- A memória de um computador pode ser vista como um grande vetor, onde cada entrada é capaz de armazenar um Byte e cada Byte pode ser acessado por seu endereço de memória (i.e. seu índice).
- Instruções de máquina como as listadas abaixo são capazes de mover um valor entre uma posição de memória e um registrador

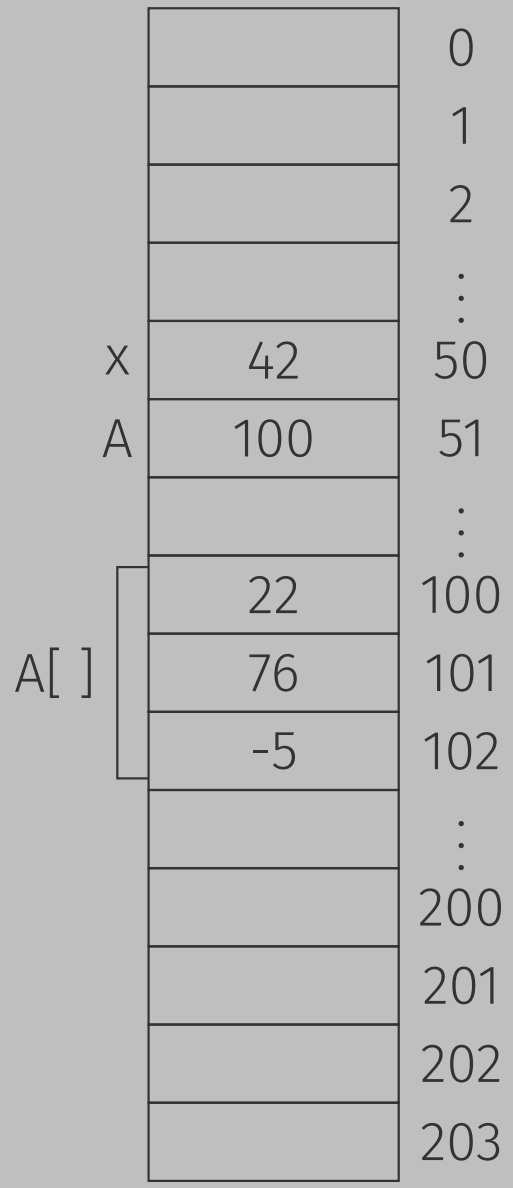
```
lw $s0, 100($t0) # s0 = t0[100]
sw $s0, 100($t0) # t0[100] = s0
```



`list` por baixo do capô

```
x = 42
A = [22, 76, -5]
y = x
B = A
```

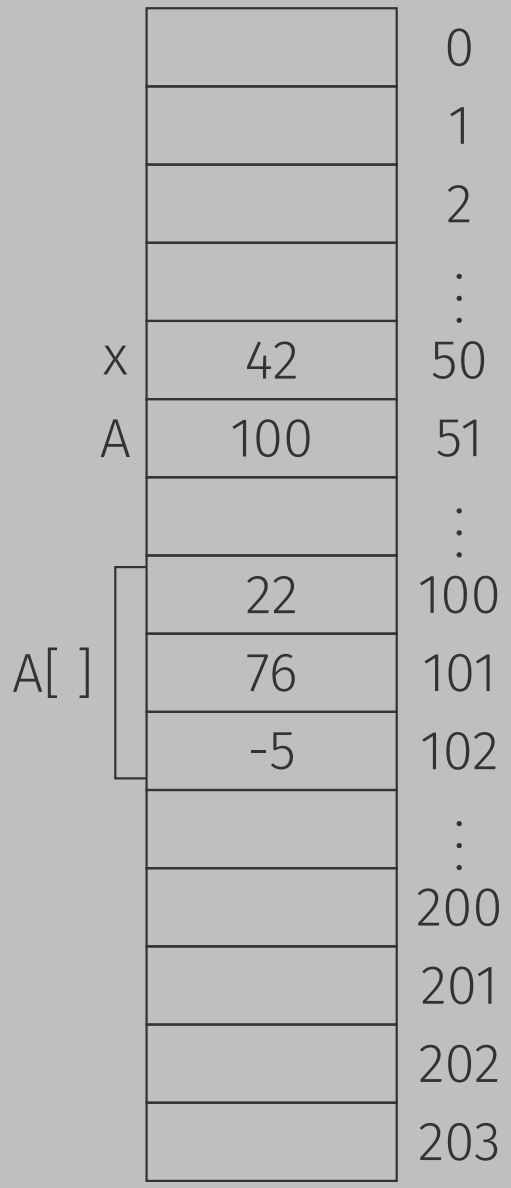
- `x`, tipo `int`, armazena o seu valor



`list` por baixo do capô

```
x = 42
A = [22, 76, -5]
y = x
B = A
```

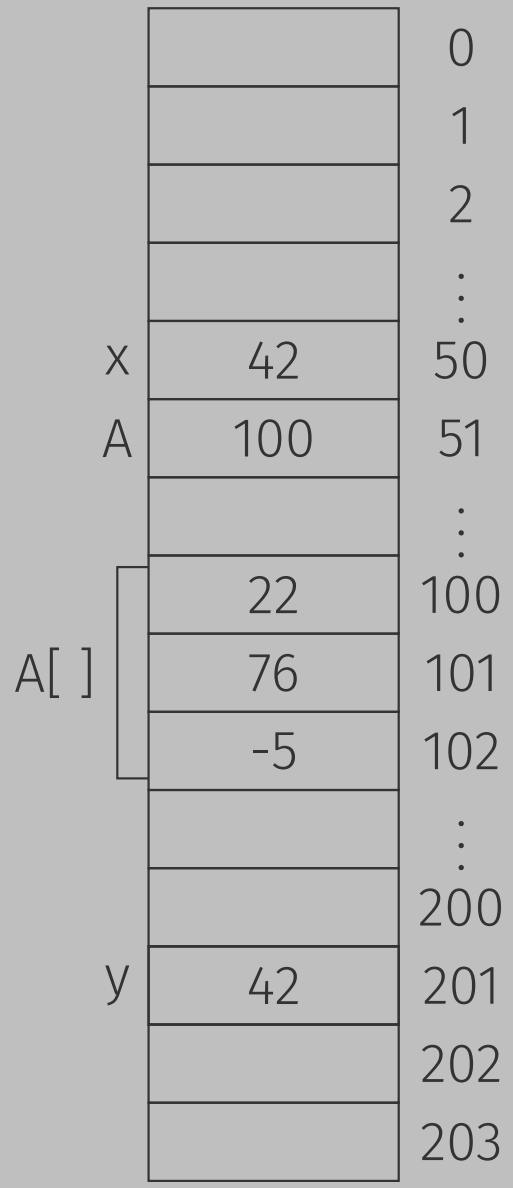
- `x`, tipo `int`, armazena o seu valor
- `A`, tipo `list`, armazena o endereço de memória (*endereço base*) do início da posição de memória no qual os valores do vetor foram armazenados de forma contígua



`list` por baixo do capô

```
x = 42
A = [22, 76, -5]
y = x
B = A
```

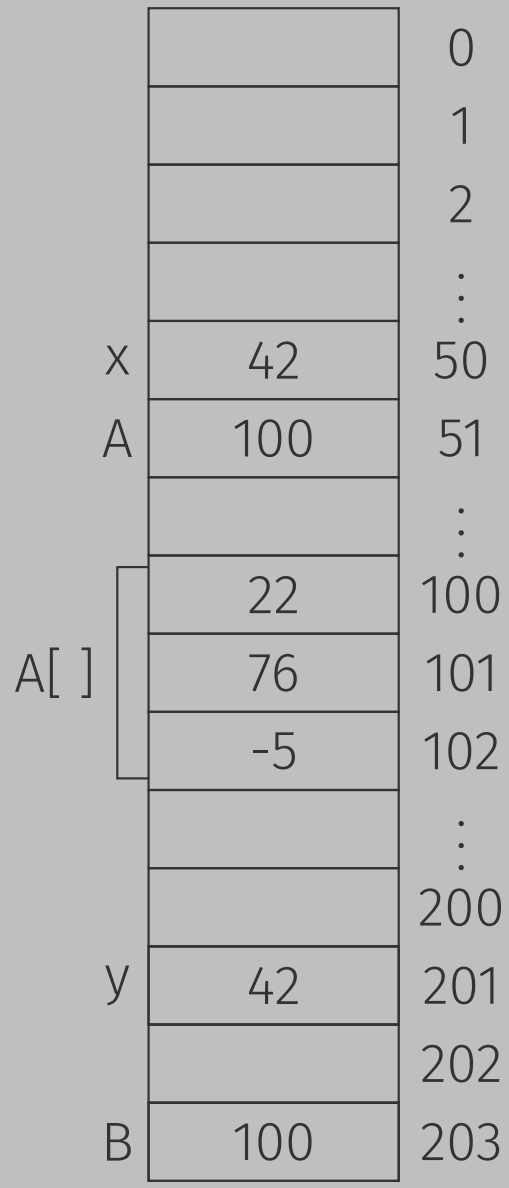
- `x`, tipo `int`, armazena o seu valor
- `A`, tipo `list`, armazena o endereço de memória (*endereço base*) do início da posição de memória no qual os valores do vetor foram armazenados de forma contígua
- Quando fazemos `y = x`, estamos copiando o conteúdo de `x`, o número 42, para `y`



`list` por baixo do capô

```
x = 42
A = [22, 76, -5]
y = x
B = A
```

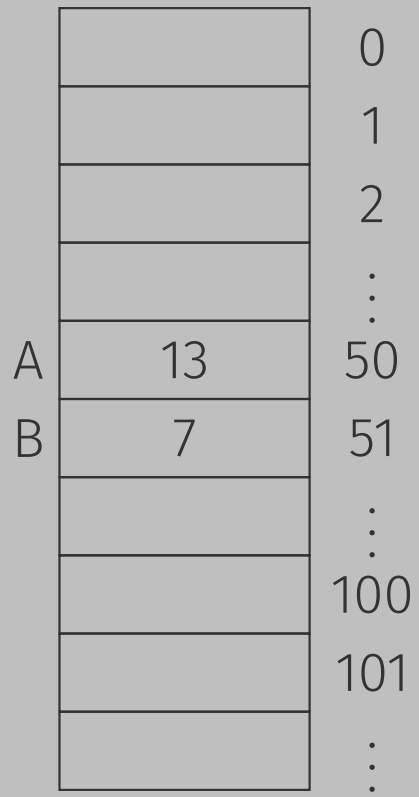
- `x`, tipo `int`, armazena o seu valor
- `A`, tipo `list`, armazena o endereço de memória (*endereço base*) do início da posição de memória no qual os valores do vetor foram armazenados de forma contígua
- Quando fazemos `y = x`, estamos copiando o conteúdo de `x`, o número 42, para `y`
- Quando fazemos `B = A`, estamos copiando o conteúdo de `A`, o endereço base, para `B`



`list` como argumento de função

Existe uma vantagem nessa abordagem

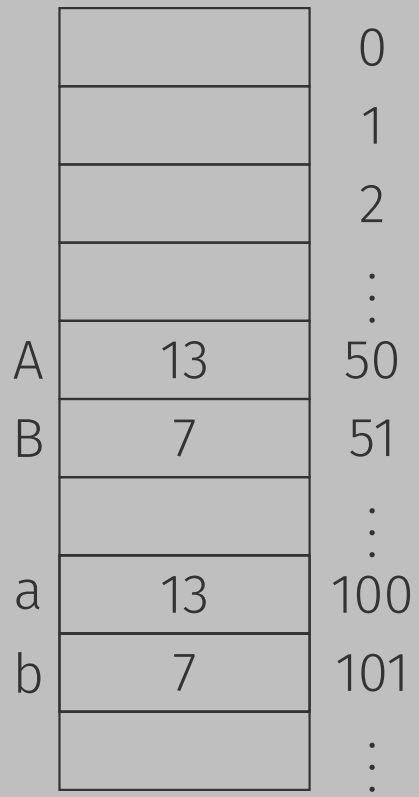
```
1  def maior(a, b):
2      if a >= b:
3          return a
4      else:
5          return b
6
7  A = 13
8  B = 7
9  print("Maior elemento:", maior(A, B))
```



`list` como argumento de função

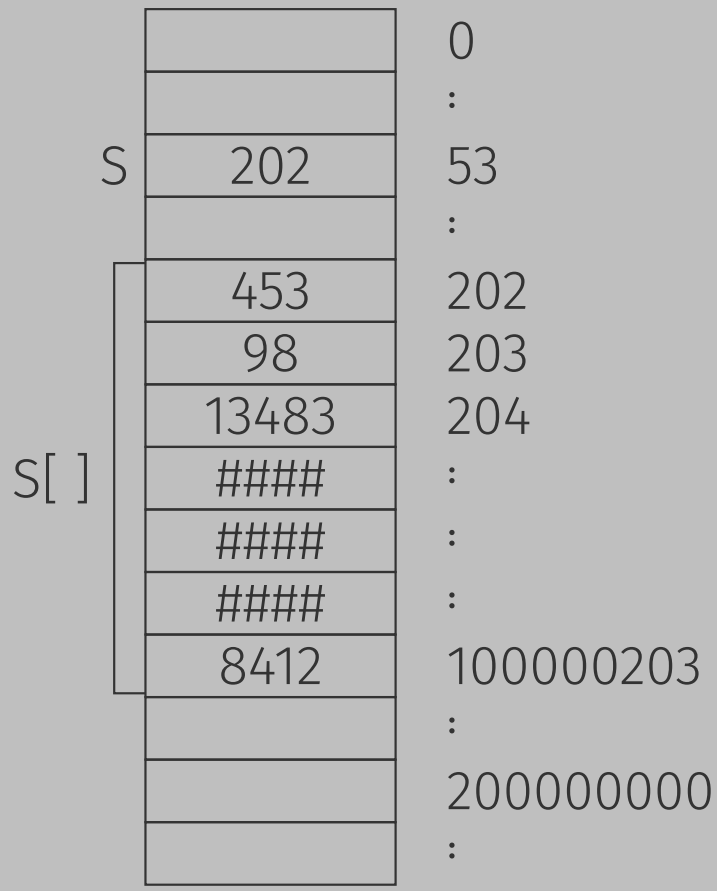
Existe uma vantagem nessa abordagem

```
1  def maior(a, b):
2      if a >= b:
3          return a
4      else:
5          return b
6
7  A = 13
8  B = 7
9  print("Maior elemento:", maior(A, B))
```



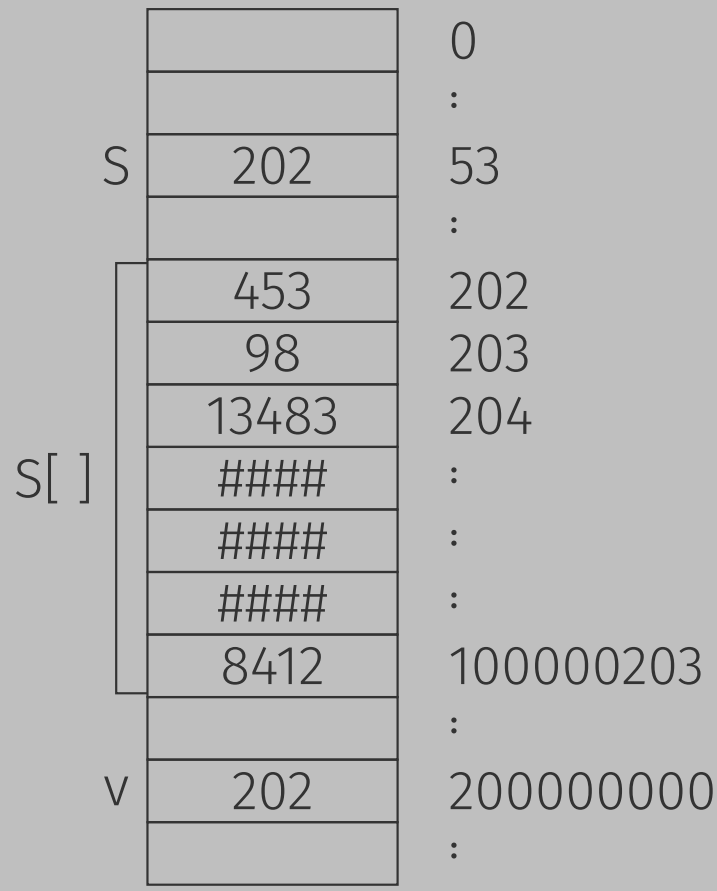
`list` como argumento de função

```
1  from random import randint
2
3  def maior(v):
4      max = v[0]
5      for i in range(1, len(v)):
6          if v[i] > max:
7              max = v[i]
8      return max
9
10 S = []
11 for i in range(100000000):
12     S.append(randint(1, 50000))
13 print("Maior elemento de S:", maior(S))
```



`list` como argumento de função

```
1  from random import randint
2
3  def maior(v):
4      max = v[0]
5      for i in range(1, len(v)):
6          if v[i] > max:
7              max = v[i]
8      return max
9
10 S = []
11 for i in range(100000000):
12     S.append(randint(1, 50000))
13 print("Maior elemento de S:", maior(S))
```



Voltando...

Diferença de vetor para outros tipos em chamada de função

```
1  def inc(x):
2    x += 1
3
4  x = 5
5  inc(x)
6  print("x:", x)
```

Diferença de vetor para outros tipos em chamada de função

```
1  def inc(x):
2    x += 1
3
4  x = 5
5  inc(x)
6  print("x:", x)
```

```
x: 5
```

Diferença de vetor para outros tipos em chamada de função

```
1  def inc(v):
2    for i in range(len(v)):
3      v[i] = v[i] + 1
4
5  v = [1, 2, 3]
6  inc(v)
7  print("v:", v)
```

Diferença de vetor para outros tipos em chamada de função

```
1  def inc(v):
2    for i in range(len(v)):
3      v[i] = v[i] + 1
4
5  v = [1, 2, 3]
6  inc(v)
7  print("v:", v)
```

```
v: [2, 3, 4]
```

Criando uma cópia de um vetor

Para fazer uma cópia do conteúdo de um vetor, usamos um recurso do Python chamado *slicing*

Slicing

Seja `v` uma variável que armazena um valor do tipo `list` (vetor) e sejam `i` e `j` variáveis que armazenam valores do tipo `int`. Então, `v[i:j]` devolve um novo vetor contendo os elementos `v[i]`, `v[i + 1]`, `v[i + 2]`, `v[i + 3]`, ..., `v[j - 1]`, nesta ordem.

Criando uma cópia de um vetor

Para fazer uma cópia do conteúdo de um vetor, usamos um recurso do Python chamado *slicing*

Slicing

Seja `v` uma variável que armazena um valor do tipo `list` (vetor) e sejam `i` e `j` variáveis que armazenam valores do tipo `int`. Então, `v[i:j]` devolve um novo vetor contendo os elementos `v[i]`, `v[i + 1]`, `v[i + 2]`, `v[i + 3]`, ..., `v[j - 1]`, nesta ordem.

Se `i` $\geq$ `j`, então é devolvido o vetor vazio `[]`

Criando uma cópia de um vetor

Para fazer uma cópia do conteúdo de um vetor, usamos um recurso do Python chamado *slicing*

Slicing

Seja `v` uma variável que armazena um valor do tipo `list` (vetor) e sejam `i` e `j` variáveis que armazenam valores do tipo `int`. Então, `v[i:j]` devolve um novo vetor contendo os elementos `v[i]`, `v[i + 1]`, `v[i + 2]`, `v[i + 3]`, ..., `v[j - 1]`, nesta ordem.

Se `i` $\geq$ `j`, então é devolvido o vetor vazio `[]`

Açúcar Sintático

`v[:j]` $\equiv$ `v[0:j]`

`v[i:]` $\equiv$ `v[i:len(v)]`

`v[:]` $\equiv$ `v[0:len(v)]`

Mais exemplos de aplicações de vetores

Busca em vetor

Escreva uma função chamada `busca(vet, x)` que recebe um vetor `vet` e um valor `x` e devolve o menor índice `i` para o qual `vet[i] = x`, ou devolve -1, caso `x` não esteja presente em `vet`.

Busca em vetor

Escreva uma função chamada `busca(vet, x)` que recebe um vetor `vet` e um valor `x` e devolve o menor índice `i` para o qual `vet[i] = x`, ou devolve -1, caso `x` não esteja presente em `vet`.

```python
def le_vetor():
    n = int(input())
    vet = []
    for i in range(n):
        vet.append(int(input()))
    return vet

def busca(vet, x):
    for i in range(len(vet)):
        if vet[i] == x:
            return i
    return -1

v = le_vetor()
x = int(input())
i = busca(v, x)
if i != -1:
    print(f"Elemento {x} encontrado na posição {i}")
```

Repetição

Escreva uma função chamada `eh_conjunto(vet)` que recebe um vetor `vet` e verifica se o mesmo é um conjunto (não contém elementos repetidos), devolvendo `True` se for e `False` se não for.

Repetição

Escreva uma função chamada `eh_conjunto(vet)` que recebe um vetor `vet` e verifica se o mesmo é um conjunto (não contém elementos repetidos), devolvendo `True` se for e `False` se não for.

```
1  def eh_conjunto(vet):
2      for i in range(len(vet)-1):
3          for j in range(i+1, len(vet)):
4              if vet[i] == vet[j]:
5                  return False
6      return True
7
8  v = le_vetor()
9  if eh_conjunto(v):
10     print("Não há repetição entre os elementos dados")
```

Inserção em conjuntos

Escreva uma função chamada `insere_elemento(conj, e)` que recebe um vetor `conj` que armazena os elementos de um conjunto e um elemento `e` e insere tal elemento no conjunto, se for possível.

Inserção em conjuntos

Escreva uma função chamada `insere_elemento(conj, e)` que recebe um vetor `conj` que armazena os elementos de um conjunto e um elemento `e` e insere tal elemento no conjunto, se for possível.

```
1  def insere_elemento(conj, e):
2      if busca(conj, e) == -1:
3          conj.append(e)
4
5  v = le_vetor()
6  e = int(input())
7  insere_elemento(v, e)
```

Exercícios

Exercícios

1. A inversão é uma operação de rearranjo de genoma que consiste em selecionar um segmento de uma sequência e invertê-lo, preservando as demais partes na mesma ordem. Esse tipo de modificação pode alterar a organização dos genes sem modificar seu conteúdo, impactando processos evolutivos e a regulação genética.
   Faça uma função que recebe um vetor `vet` e dois índices `i` e `j`, sendo $i \leq j$. Sua função deve inverter o trecho entre os índices naquele vetor, sem criar um novo vetor.
   Por exemplo, se `vet = [3,6,4,8,1,0,3,4,6,7]`, `i = 3` e `j = 7`, ao final deveremos ter `vet = [3,6,4,4,3,0,1,8,6,7]`
2. Faça uma função que recebe um vetor e devolve-o sem elementos repetidos.
3. Faça uma função que recebe um vetor `vet` e um elemento `k` e devolve um vetor que tem todos os elementos de `vet` até a primeira aparição do elemento `k`.