MAC 110 — Introdução à Ciência da Computação

Aula 16

Nelson Lago

BMAC – 2024

Brincando com listas

Estas criam e devolvem uma nova lista:

Brincando com listas

Estas criam e devolvem uma nova lista:

```
print("Oi, " + "tudo bem?")
```

Brincando com listas

Estas criam e devolvem uma nova lista:

```
print("Oi, " + "tudo bem?")
```

Oi, tudo bem?

Brincando com listas

Estas criam e devolvem uma nova lista:

```
print("Oi, " + "tudo bem?")
pequenas = ["elétron", "nêutron", "próton"]
```

Oi, tudo bem?

Brincando com listas

Estas criam e devolvem uma nova lista:

```
print("Oi, " + "tudo bem?")
pequenas = ["elétron", "nêutron", "próton"]
grandes = ["átomo", "molécula", "polímero"]
```

Oi, tudo bem?

Brincando com listas

Estas criam e devolvem uma nova lista:

```
print("Oi, " + "tudo bem?")
pequenas = ["elétron", "nêutron", "próton"]
grandes = ["átomo", "molécula", "polímero"]
todas = pequenas + grandes
```

Oi, tudo bem?

Brincando com listas

Estas criam e devolvem uma nova lista:

```
print("Oi, " + "tudo bem?")
pequenas = ["elétron", "nêutron", "próton"]
grandes = ["átomo", "molécula", "polímero"]
todas = pequenas + grandes
print(todas)
```

```
Oi, tudo bem?
```

Brincando com listas

Estas criam e devolvem uma nova lista:

```
print("Oi, " + "tudo bem?")
pequenas = ["elétron", "nêutron", "próton"]
grandes = ["átomo", "molécula", "polímero"]
todas = pequenas + grandes
print(todas)
```

```
Oi, tudo bem?
['elétron', 'nêutron', 'próton', 'átomo', 'molécula', 'polímero']
```

Brincando com listas

Estas criam e devolvem uma nova lista:

```
print("Oi, " + "tudo bem?")
pequenas = ["elétron", "nêutron", "próton"]
grandes = ["átomo", "molécula", "polímero"]
todas = pequenas + grandes
print(todas)
print(pequenas, grandes)
```

```
Oi, tudo bem?
['elétron', 'nêutron', 'próton', 'átomo', 'molécula', 'polímero']
```

Brincando com listas

Estas criam e devolvem uma nova lista:

```
print("Oi, " + "tudo bem?")
pequenas = ["elétron", "nêutron", "próton"]
grandes = ["átomo", "molécula", "polímero"]
todas = pequenas + grandes
print(todas)
print(pequenas, grandes)
```

```
Oi, tudo bem?
['elétron', 'nêutron', 'próton', 'átomo', 'molécula', 'polímero']
['elétron', 'nêutron', 'próton'] ['átomo', 'molécula', 'polímero']
```

Brincando com listas

Estas criam e devolvem uma nova lista:

```
print("Oi, " + "tudo bem?")
pequenas = ["elétron", "nêutron", "próton"]
grandes = ["átomo", "molécula", "polímero"]
todas = pequenas + grandes
print(todas)
print(pequenas, grandes)
print([""]*5)
```

```
Oi, tudo bem?
['elétron', 'nêutron', 'próton', 'átomo', 'molécula', 'polímero']
['elétron', 'nêutron', 'próton'] ['átomo', 'molécula', 'polímero']
```

Brincando com listas

Estas criam e devolvem uma nova lista:

```
print("Oi, " + "tudo bem?")
pequenas = ["elétron", "nêutron", "próton"]
grandes = ["átomo", "molécula", "polímero"]
todas = pequenas + grandes
print(todas)
print(pequenas, grandes)
print([""]*5)
```

```
Oi, tudo bem?
['elétron', 'nêutron', 'próton', 'átomo', 'molécula', 'polímero']
['elétron', 'nêutron', 'próton'] ['átomo', 'molécula', 'polímero']
['', '', '', '', '']
```

Brincando com listas

Estas criam e devolvem uma nova lista:

```
print("Oi, " + "tudo bem?")
pequenas = ["elétron", "nêutron", "próton"]
grandes = ["átomo", "molécula", "polímero"]
todas = pequenas + grandes
print(todas)
print(pequenas, grandes)
print([""]*5)
print(grandes[1:3])
```

```
Oi, tudo bem?
['elétron', 'nêutron', 'próton', 'átomo', 'molécula', 'polímero']
['elétron', 'nêutron', 'próton'] ['átomo', 'molécula', 'polímero']
['', '', '', '', '']
```

Brincando com listas

Estas criam e devolvem uma nova lista:

```
print("Oi, " + "tudo bem?")
pequenas = ["elétron", "nêutron", "próton"]
grandes = ["átomo", "molécula", "polímero"]
todas = pequenas + grandes
print(todas)
print(pequenas, grandes)
print([""]*5)
print(grandes[1:3])
```

```
Oi, tudo bem?
['elétron', 'nêutron', 'próton', 'átomo', 'molécula', 'polímero']
['elétron', 'nêutron', 'próton'] ['átomo', 'molécula', 'polímero']
['', '', '', '', '']
['molécula', 'polímero']
```

Brincando com listas

Estas modificam a própria lista:

Brincando com listas

Estas modificam a própria lista:

```
guloseimas = ["tofu", "quiabo"]
```

Brincando com listas

Estas modificam a própria lista:

```
guloseimas = ["tofu", "quiabo"]
guloseimas.append("jiló")
```

Brincando com listas

Estas modificam a própria lista:

```
guloseimas = ["tofu", "quiabo"]
guloseimas.append("jiló")
gostosuras = ["mostarda", "rúcula"]
```

Brincando com listas

Estas modificam a própria lista:

```
guloseimas = ["tofu", "quiabo"]
guloseimas.append("jiló")
gostosuras = ["mostarda", "rúcula"]
guloseimas.extend(gostosuras)
print(guloseimas)
```

Brincando com listas

Estas modificam a própria lista:

```
guloseimas = ["tofu", "quiabo"]
guloseimas.append("jiló")
gostosuras = ["mostarda", "rúcula"]
guloseimas.extend(gostosuras)
print(guloseimas)
```

```
['tofu', 'quiabo', 'jiló', 'mostarda', 'rúcula']
```

Brincando com listas

Estas modificam a própria lista:

```
guloseimas = ["tofu", "quiabo"]
guloseimas.append("jiló")
gostosuras = ["mostarda", "rúcula"]
guloseimas.extend(gostosuras)
print(guloseimas)
del gostosuras[1:2]
print(gostosuras)
```

```
['tofu', 'quiabo', 'jiló', 'mostarda', 'rúcula']
```

Brincando com listas

Estas modificam a própria lista:

```
guloseimas = ["tofu", "quiabo"]
guloseimas.append("jiló")
gostosuras = ["mostarda", "rúcula"]
guloseimas.extend(gostosuras)
print(guloseimas)
del gostosuras[1:2]
print(gostosuras)
```

```
['tofu', 'quiabo', 'jiló', 'mostarda', 'rúcula']
['mostarda']
```

Brincando com listas

Estas modificam a própria lista:

```
guloseimas = ["tofu", "quiabo"]
guloseimas.append("jiló")
gostosuras = ["mostarda", "rúcula"]
guloseimas.extend(gostosuras)
print(guloseimas)
del gostosuras[1:2]
print(gostosuras)
print(guloseimas)
```

```
['tofu', 'quiabo', 'jiló', 'mostarda', 'rúcula']
['mostarda']
```

Brincando com listas

Estas modificam a própria lista:

```
guloseimas = ["tofu", "quiabo"]
guloseimas.append("jiló")
gostosuras = ["mostarda", "rúcula"]
guloseimas.extend(gostosuras)
print(guloseimas)
del gostosuras[1:2]
print(gostosuras)
print(guloseimas)
```

```
['tofu', 'quiabo', 'jiló', 'mostarda', 'rúcula']
['mostarda']
['tofu', 'quiabo', 'jiló', 'mostarda', 'rúcula']
```

Brincando com listas

Estas modificam a própria lista:

```
guloseimas = ["tofu", "quiabo"]
guloseimas.append("jiló")
gostosuras = ["mostarda", "rúcula"]
guloseimas.extend(gostosuras)
print(guloseimas)
del gostosuras[1:2]
print(gostosuras)
print(guloseimas)
guloseimas.sort()
print(guloseimas)
```

```
['tofu', 'quiabo', 'jiló', 'mostarda', 'rúcula']
['mostarda']
['tofu', 'quiabo', 'jiló', 'mostarda', 'rúcula']
```

Brincando com listas

Estas modificam a própria lista:

```
guloseimas = ["tofu", "quiabo"]
guloseimas.append("jiló")
gostosuras = ["mostarda", "rúcula"]
guloseimas.extend(gostosuras)
print(guloseimas)
del gostosuras[1:2]
print(gostosuras)
print(guloseimas)
guloseimas.sort()
print(guloseimas)
```

```
['tofu', 'quiabo', 'jiló', 'mostarda', 'rúcula']
['mostarda']
['tofu', 'quiabo', 'jiló', 'mostarda', 'rúcula']
['jiló', 'mostarda', 'quiabo', 'rúcula', 'tofu']
```

Intermezzo

Métodos e funções

- **Funções em geral representam "ações" que esperamos que o computador realize**

Métodos e funções

- Funções em geral representam "ações" que esperamos que o computador realize
- Existem dois tipos de função em python: *funções* e *métodos*

Métodos e funções

- Funções em geral representam "ações" que esperamos que o computador realize
- Existem dois tipos de função em python: *funções* e *métodos*
- Funções seguem o formato *verbo + objeto*:

Métodos e funções

- **Funções em geral representam "ações" que esperamos que o computador realize**
- **Existem dois tipos de função em python: *funções* e *métodos***
- **Funções seguem o formato *verbo + objeto*:**
  - `print(blah)`

Métodos e funções

- **Funções em geral representam "ações" que esperamos que o computador realize**
- **Existem dois tipos de função em python: *funções* e *métodos***
- **Funções seguem o formato *verbo + objeto*:**
  - `print(blah)`
  - `éPrimo(x)`

Métodos e funções

- **Funções em geral representam "ações" que esperamos que o computador realize**
- **Existem dois tipos de função em python: *funções* e *métodos***
- **Funções seguem o formato *verbo + objeto*:**
  - `print(blah)`
  - `éPrimo(x)`
  - ...

Métodos e funções

- **Funções em geral representam "ações" que esperamos que o computador realize**
- **Existem dois tipos de função em python: *funções* e *métodos***
- **Funções seguem o formato *verbo + objeto*:**
  - `print(blah)`
  - `éPrimo(x)`
  - ...
- **Métodos seguem o formato *objeto + verbo***

Métodos e funções

- **Funções em geral representam "ações" que esperamos que o computador realize**
- **Existem dois tipos de função em python: *funções* e *métodos***
- **Funções seguem o formato *verbo + objeto*:**
  - `print(blah)`
  - `éPrimo(x)`
  - ...
- **Métodos seguem o formato *objeto + verbo***
  - `lista.append()`

Métodos e funções

- **Funções em geral representam "ações" que esperamos que o computador realize**
- **Existem dois tipos de função em python: *funções* e *métodos***
- **Funções seguem o formato *verbo + objeto*:**
  - `print(blah)`
  - `éPrimo(x)`
  - ...
- **Métodos seguem o formato *objeto + verbo***
  - `lista.append()`
  - ...

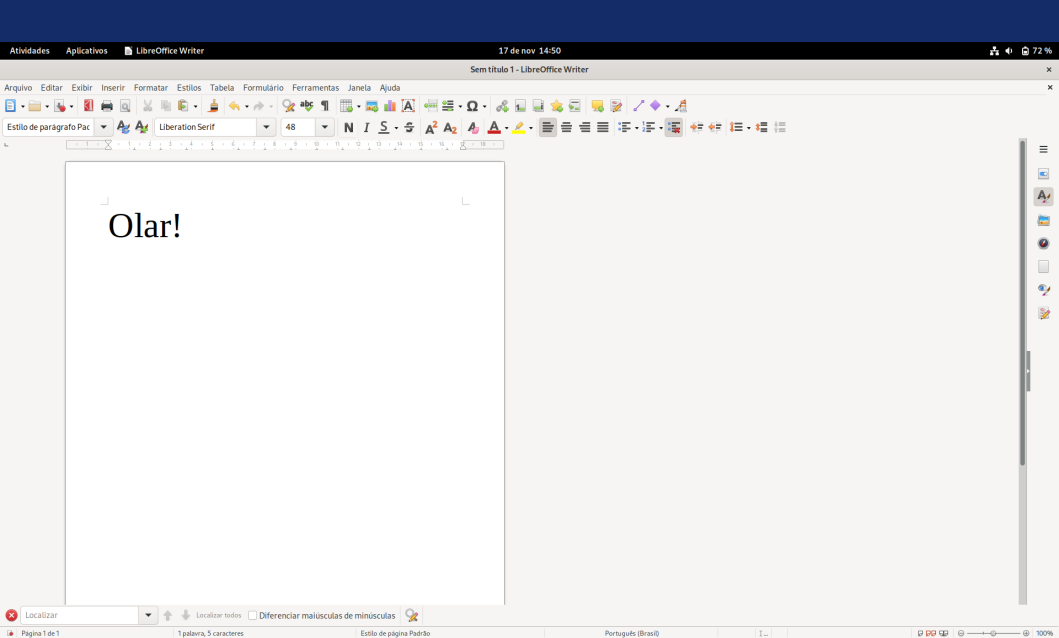
Sem título 1 - LibreOffice Writer

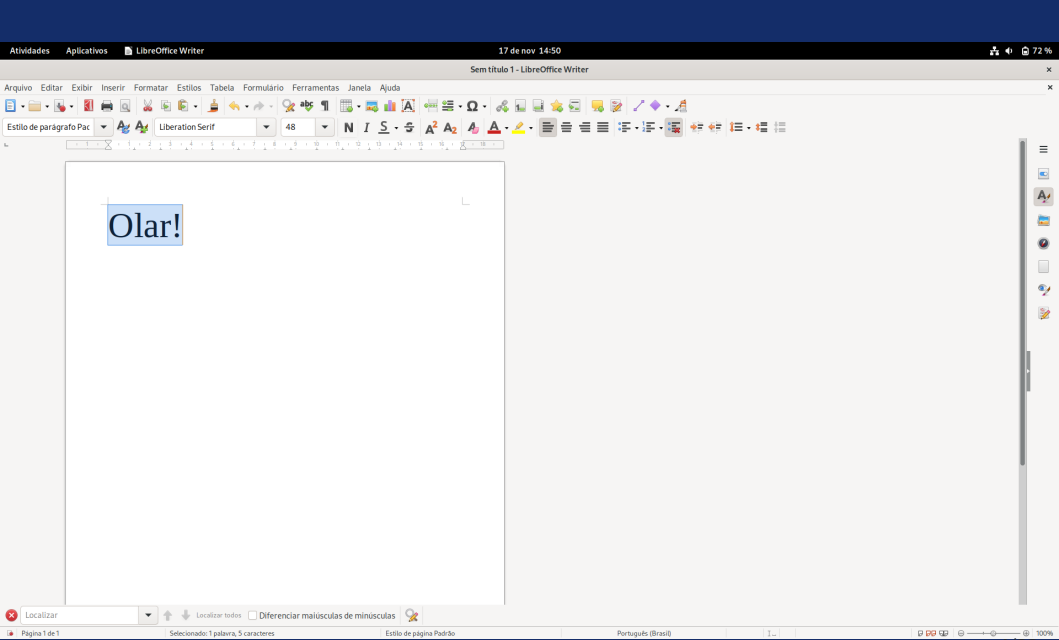
Arquivo Editar Exibir Inserir Formatar Estilos Tabela Formulário Ferramentas Janela Ajuda

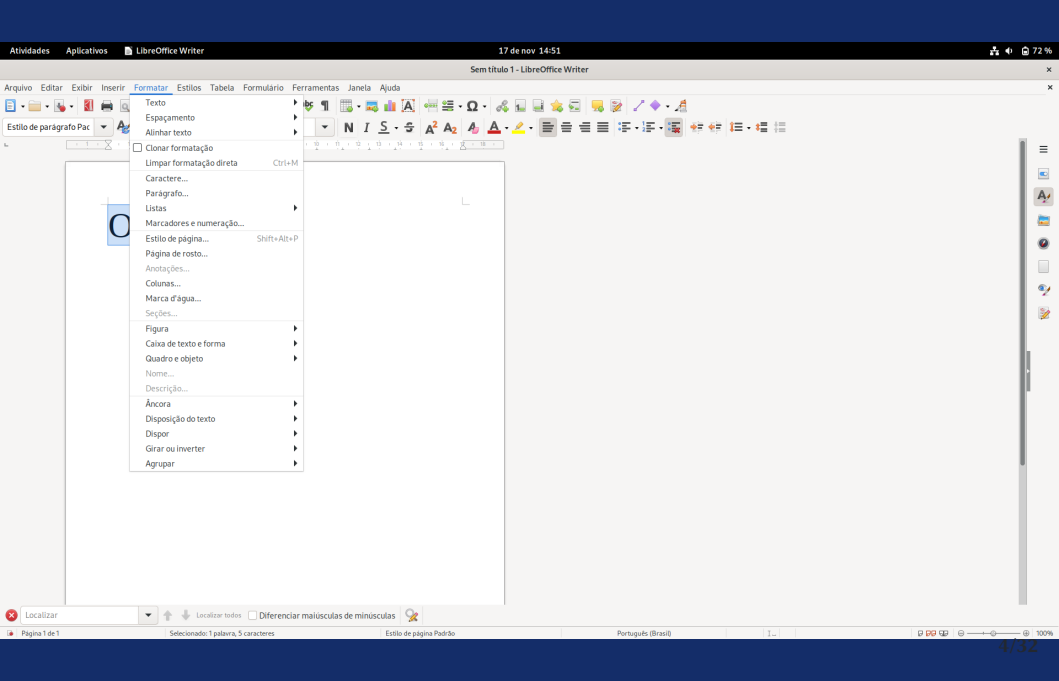
Estilo de parágrafo Pad Liberation Serif 48

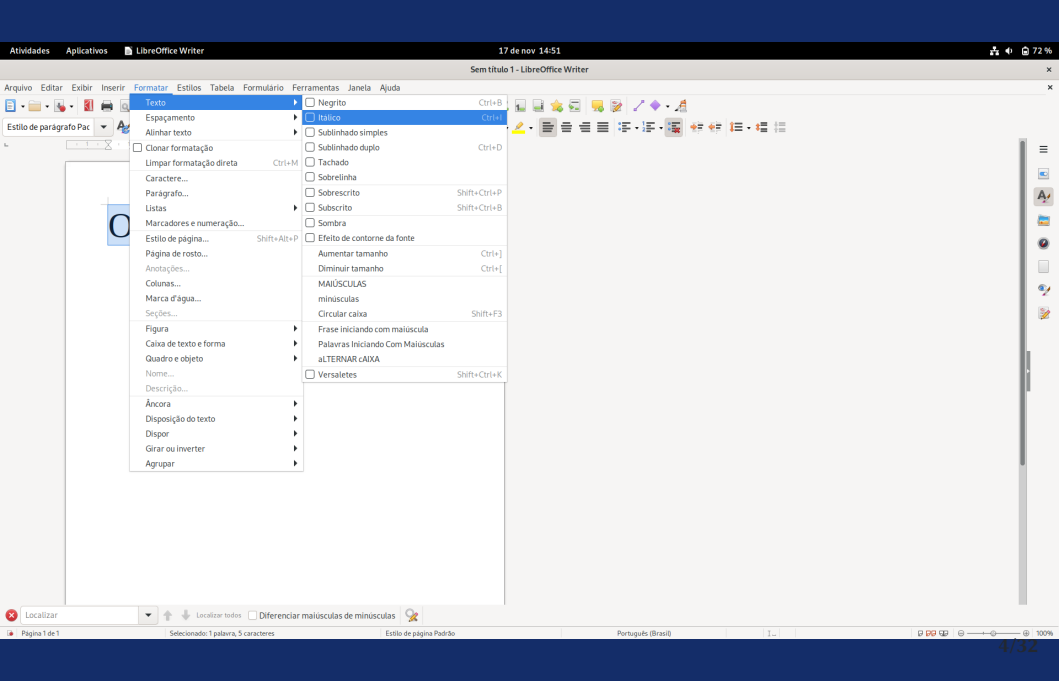
Localizar Localizar todos Diferenciar maiúsculas de minúsculas

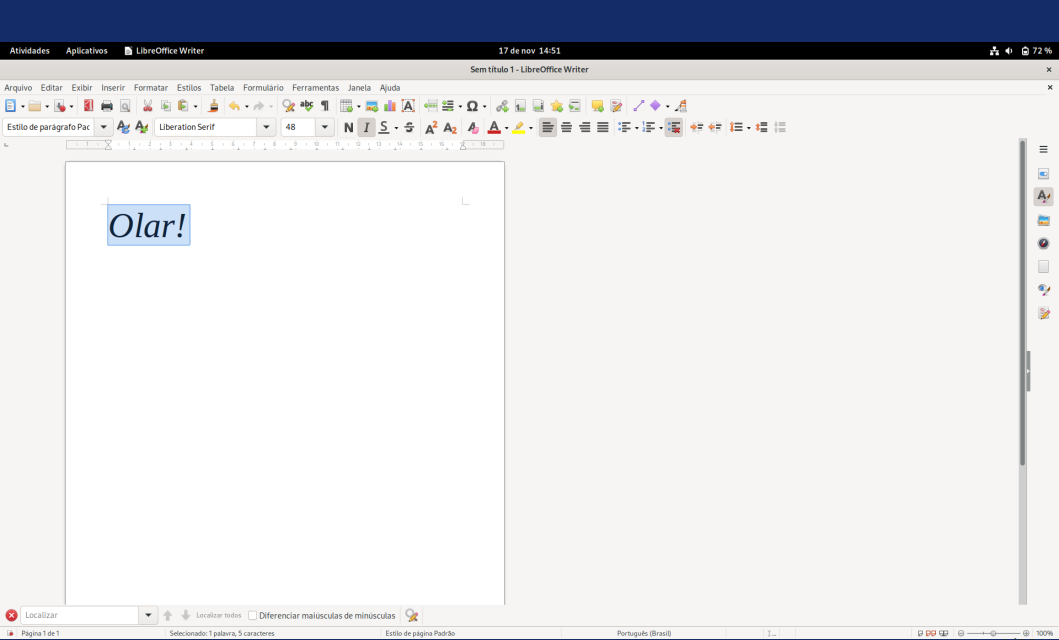
Página 1 de 1 0 palavra, 0 caractere Estilo de página Padrão Português (Brasil) 100%


Atividades
Aplicativos
LibreOffice Writer
17 de nov 14:50
72 %
Sem título 1 - LibreOffice Writer
Arquivo Editar Exibir Inserir Formatar Estilos Tabela Formulário Ferramentas Janela Ajuda
Estilo de parágrafo Pad
Liberation Serif
48
Olar!
Localizar
Localizar todos
Diferenciar maiúsculas de minúsculas
Página 1 de 1
1 palavra, 5 caracteres
Estilo de página Padrão
Português (Brasil)
100%


Atividades
Aplicativos
LibreOffice Writer
17 de nov 14:50
72 %
Sem título 1 - LibreOffice Writer
Arquivo Editar Exibir Inserir Formatar Estilos Tabela Formulário Ferramentas Janela Ajuda
Estilo de parágrafo Pad
Liberation Serif
48
Olar!
Localizar
Localizar todos
Diferenciar maiúsculas de minúsculas
Página 1 de 1
Selecionado: 1 palavra, 5 caracteres
Estilo de página Padrão
Português (Brasil)
100%


Atividades
Aplicativos
LibreOffice Writer
17 de nov 14:51
72 %
Sem título 1 - LibreOffice Writer
Arquivo
Editar
Exibir
Inserir
Formatar
Estilos
Tabela
Formulário
Ferramentas
Janela
Ajuda
Texto
Espaçamento
Alinhar texto
Clonar formatação
Limpar formatação direta
Ctrl+M
Caractere...
Parágrafo...
Listas
Marcadores e numeração...
Estilo de página...
Shift+Alt+P
Página de rosto...
Anotações...
Colunas...
Marca d'água...
Seções...
Figura
Caixa de texto e forma
Quadro e objeto
Nome...
Descrição...
Âncora
Disposição do texto
Dispor
Girar ou inverter
Agrupar
Estilo de parágrafo Pad
Localizar
Localizar todos
Diferenciar maiúsculas de minúsculas
Página 1 de 1
Selecionado: 1 palavra, 5 caracteres
Estilo de página Padrão
Português (Brasil)
100%


Atividades
Aplicativos
LibreOffice Writer
17 de nov 14:51
72 %
Sem título 1 - LibreOffice Writer
Arquivo
Editar
Exibir
Inserir
Formatar
Estilos
Tabela
Formulário
Ferramentas
Janela
Ajuda
Texto
Espaçamento
Alinhar texto
Clonar formatação
Limpar formatação direta
Ctrl+M
Caractere...
Parágrafo...
Listas
Marcadores e numeração...
Estilo de página...
Shift+Alt+P
Página de rosto...
Anotações...
Colunas...
Marca d'água...
Seções...
Figura
Caixa de texto e forma
Quadro e objeto
Nome...
Descrição...
Âncora
Disposição do texto
Dispor
Girar ou inverter
Agrupar
Negrito
Ctrl+B
Itálico
Ctrl+I
Sublinhado simples
Sublinhado duplo
Ctrl+D
Tachado
Sobrelinha
Sobrescrito
Shift+Ctrl+P
Subscrito
Shift+Ctrl+B
Sombra
Efeito de contorno da fonte
Aumentar tamanho
Ctrl+]
Diminuir tamanho
Ctrl+[
MAIÚSCULAS
minúsculas
Circular caixa
Shift+F3
Frase iniciando com maiúscula
Palavras Iniciando Com Maiúsculas
aLTERNAR cAIXA
Versaletes
Shift+Ctrl+K
Estilo de parágrafo Pad
Localizar
Localizar todos
Diferenciar maiúsculas de minúsculas
Página 1 de 1
Selecionado: 1 palavra, 5 caracteres
Estilo de página Padrão
Português (Brasil)
100%


Atividades
Aplicativos
LibreOffice Writer
17 de nov 14:51
72 %
Sem título 1 - LibreOffice Writer
Arquivo Editar Exibir Inserir Formatar Estilos Tabela Formulário Ferramentas Janela Ajuda
Estilo de parágrafo Pad
Liberation Serif
48
Olar!
Localizar
Localizar todos
Diferenciar maiúsculas de minúsculas
Página 1 de 1
Selecionado: 1 palavra, 5 caracteres
Estilo de página Padrão
Português (Brasil)
100%

Os quatro sinais — templo tibetano em Bodhgaya
foto de Anandajoti Bhikkhu
www.flickr.com/photos/anandajoti/9225063815/

Nomes e memória

- **Em python, só existe quem tem nome:**

Nomes e memória

- **Em python, só existe quem tem nome:**

```
saudação = "Olá!"
saudação = "Oi!"
```

Nomes e memória

- **Em python, só existe quem tem nome:**

```
saudação = "Olá!"
saudação = "Oi!"
```

- **O texto `"Olá!"` não existe mais em nenhum lugar da memória após a segunda linha ser executada!**

Nomes e memória

- **Em python, só existe quem tem nome:**

```
saudação = "Olá!"
saudação = "Oi!"
```

- **O texto `"Olá!"` não existe mais em nenhum lugar da memória após a segunda linha ser executada!**

```
x = 1
x += 1
```

Nomes e memória

- **Em python, só existe quem tem nome:**

```
saudação = "Olá!"
saudação = "Oi!"
```

- **O texto `"Olá!"` não existe mais em nenhum lugar da memória após a segunda linha ser executada!**

```
x = 1
x += 1
```

- **O número `1` não existe mais em nenhum lugar da memória após a segunda linha ser executada!**

Nomes e memória

- **Em python, só existe quem tem nome:**

Nomes e memória

- **Em python, só existe quem tem nome:**

```
x = 1
y = x
x += 1
```

Nomes e memória

- **Em python, só existe quem tem nome:**

```
x = 1
y = x
x += 1
```

- **O número `1` *continua existindo* na memória após a terceira linha ser executada**

Nomes e memória

- **Em python, só existe quem tem nome:**

```
x = 1
y = x
x += 1
```

- **O número `1` *continua existindo* na memória após a terceira linha ser executada**
  - Mas o que acontece na segunda linha?!?!?

Nomes e memória

- **Em python, só existe quem tem nome:**

```
x = 1
y = x
x += 1
```

- **O número `1` *continua existindo* na memória após a terceira linha ser executada**
  - Mas o que acontece na segunda linha?!?!?
    - » *Temos duas "cópias" do número `1` na memória?*

Nomes e memória

- **Em python, só existe quem tem nome:**

```
x = 1
y = x
x += 1
```

- **O número `1` *continua existindo* na memória após a terceira linha ser executada**
  - Mas o que acontece na segunda linha?!?!?
    - » *Temos duas "cópias" do número `1` na memória?*
- **DEPENDE**

Nomes e memória

- **Em python, só existe quem tem nome:**

```
x = 1
y = x
x += 1
```

- **O número `1` *continua existindo* na memória após a terceira linha ser executada**
  - Mas o que acontece na segunda linha?!?!?
    - » *Temos duas "cópias" do número `1` na memória?*
- **DEPENDE**
  - Em python, não!

Nomes e memória

- **Em python, só existe quem tem nome:**

```
x = 1
y = x
x += 1
```

- **O número `1` *continua existindo* na memória após a terceira linha ser executada**
  - Mas o que acontece na segunda linha?!?!?
    - » *Temos duas "cópias" do número `1` na memória?*
- **DEPENDE**
  - Em python, não!
    - » *O número `1` simplesmente tem dois nomes (`x` e `y`)*

Nomes e memória

- **Em python, só existe quem tem nome:**

```
x = 1
y = x
x += 1
```

- **O número `1` *continua existindo* na memória após a terceira linha ser executada**
  - Mas o que acontece na segunda linha?!?!?
    - » *Temos duas "cópias" do número `1` na memória?*
- **DEPENDE**
  - Em python, não!
    - » *O número `1` simplesmente tem dois nomes (`x` e `y`)*
    - » *Na terceira linha, ele volta a ter apenas um nome (`y`)*

O que são “variáveis”?

O que são “variáveis”?

- **Nomes que usamos para acessar dados armazenados em algum lugar na memória do computador?**

O que são “variáveis”?

- **Nomes que usamos para acessar dados armazenados em algum lugar na memória do computador?**
- **Conceitos abstratos que representam uma informação relevante para a computação que estamos realizando?**

O que são “variáveis”?

- **Nomes que usamos para acessar dados armazenados em algum lugar na memória do computador?**
- **Conceitos abstratos que representam uma informação relevante para a computação que estamos realizando?**

Ambos

O que são “variáveis”?

- **Nomes que usamos para acessar dados armazenados em algum lugar na memória do computador?**
- **Conceitos abstratos que representam uma informação relevante para a computação que estamos realizando?**

Ambos

- **Mas cada linguagem dá mais ênfase a este ou àquele ponto de vista**

igualdade e identidade

- **Em python**
  - `x += 1` não altera o número `1` armazenado na memória, mas sim o valor associado à variável `x` (o número `2` "nasce" e recebe o nome `x`, enquanto o número `1` é "jogado fora")

igualdade e identidade

- **Em python**
  - `x += 1` não altera o número `1` armazenado na memória, mas sim o valor associado à variável `x` (o número `2` "nasce" e recebe o nome `x`, enquanto o número `1` é "jogado fora")
    - » *Pensando em variáveis como conceitos, isso faz sentido: o valor da variável é o conceito, e ele foi modificado; o número armazenado é* ***outro*** *número*

igualdade e identidade

- **Em python**
  - `x += 1` não altera o número `1` armazenado na memória, mas sim o valor associado à variável `x` (o número `2` "nasce" e recebe o nome `x`, enquanto o número `1` é "jogado fora")
    - » *Pensando em variáveis como conceitos, isso faz sentido: o valor da variável é o conceito, e ele foi modificado; o número armazenado é* ***outro*** *número*
- **Em outras linguagens, como C, as coisas são diferentes:**

igualdade e identidade

- **Em python**
  - `x += 1` não altera o número `1` armazenado na memória, mas sim o valor associado à variável `x` (o número `2` "nasce" e recebe o nome `x`, enquanto o número `1` é "jogado fora")
    - » *Pensando em variáveis como conceitos, isso faz sentido: o valor da variável é o conceito, e ele foi modificado; o número armazenado é* ***outro*** *número*
- **Em outras linguagens, como C, as coisas são diferentes:**
  - `x = y` guarda duas "cópias" do mesmo número em lugares diferentes da memória

igualdade e identidade

- **Em python**
  - `x += 1` não altera o número `1` armazenado na memória, mas sim o valor associado à variável `x` (o número `2` "nasce" e recebe o nome `x`, enquanto o número `1` é "jogado fora")
    - » *Pensando em variáveis como conceitos, isso faz sentido: o valor da variável é o conceito, e ele foi modificado; o número armazenado é* ***outro*** *número*
- **Em outras linguagens, como C, as coisas são diferentes:**
  - `x = y` guarda duas "cópias" do mesmo número em lugares diferentes da memória
  - `x += 1` procura o lugar na memória em que a variável `x` está armazenada e altera o valor que está ali

igualdade e identidade

- **Em python**
  - `x += 1` não altera o número `1` armazenado na memória, mas sim o valor associado à variável `x` (o número `2` "nasce" e recebe o nome `x`, enquanto o número `1` é "jogado fora")
    - » *Pensando em variáveis como conceitos, isso faz sentido: o valor da variável é o conceito, e ele foi modificado; o número armazenado é* ***outro*** *número*
- **Em outras linguagens, como C, as coisas são diferentes:**
  - `x = y` guarda duas "cópias" do mesmo número em lugares diferentes da memória
  - `x += 1` procura o lugar na memória em que a variável `x` está armazenada e altera o valor que está ali
    - » *Pensando em variáveis como nomes para um dado na memória, isso faz sentido: o conteúdo daquele espaço de memória foi modificado*

igualdade e identidade

E por que eu preciso me preocupar com isso?

igualdade e identidade

E por que eu preciso me preocupar com isso?

```
def metade(x):
    x = x / 2
    return x
val = 10
print("A metade de {} é {}".format(val, metade(val))
```

igualdade e identidade

E por que eu preciso me preocupar com isso?

```
def metade(x):
    x = x / 2
    return x
val = 10
print("A metade de {} é {}".format(val, metade(val))
```

- **Na chamada de função, o número `10` passa a ter (temporariamente) dois nomes: `val` e `x`**

igualdade e identidade

E por que eu preciso me preocupar com isso?

```
def metade(x):
    x = x / 2
    return x
val = 10
print("A metade de {} é {}".format(val, metade(val))
```

- **Na chamada de função, o número `10` passa a ter (temporariamente) dois nomes: `val` e `x`**
  - Mas isso não tem nenhuma consequência relevante neste caso

igualdade e identidade

E por que eu preciso me preocupar com isso?

```
def metade(x):
    x = x / 2
    return x
val = 10
print("A metade de {} é {}".format(val, metade(val))
```

- **Na chamada de função, o número `10` passa a ter (temporariamente) dois nomes: `val` e `x`**
  - Mas isso não tem nenhuma consequência relevante neste caso
  - Dentro da função `metade()`, a nova atribuição a `x` associa esse nome (variável) a um novo valor dentro da função, sem alterar a associação de `val`

igualdade e identidade

E por que eu preciso me preocupar com isso?

igualdade e identidade

E por que eu preciso me preocupar com isso?

```
def adiciona(l):
    x = input("Qual item você quer adicionar à lista? ")
    l.append(x)
```

igualdade e identidade

E por que eu preciso me preocupar com isso?

```
def adiciona(l):
    x = input("Qual item você quer adicionar à lista? ")
    l.append(x)
compras = ["lampião de gás", "lenha"]
adiciona(compras)
```

igualdade e identidade

E por que eu preciso me preocupar com isso?

```
def adiciona(l):
    x = input("Qual item você quer adicionar à lista? ")
    l.append(x)
compras = ["lampião de gás", "lenha"]
adiciona(compras)
```

Qual item você quer adicionar à lista?

igualdade e identidade

E por que eu preciso me preocupar com isso?

```
def adiciona(l):
    x = input("Qual item você quer adicionar à lista? ")
    l.append(x)
compras = ["lampião de gás", "lenha"]
adiciona(compras)
```

Qual item você quer adicionar à lista? querosene

igualdade e identidade

E por que eu preciso me preocupar com isso?

```
def adiciona(l):
    x = input("Qual item você quer adicionar à lista? ")
    l.append(x)
compras = ["lampião de gás", "lenha"]
adiciona(compras)
print(compras)
```

```
Qual item você quer adicionar à lista? querosene
```

igualdade e identidade

E por que eu preciso me preocupar com isso?

```
def adiciona(l):
    x = input("Qual item você quer adicionar à lista? ")
    l.append(x)
compras = ["lampião de gás", "lenha"]
adiciona(compras)
print(compras)
```

```
Qual item você quer adicionar à lista? querosene
['lampião de gás', 'lenha', 'querosene']
```

igualdade e identidade

E por que eu preciso me preocupar com isso?

```
def adiciona(l):
    x = input("Qual item você quer adicionar à lista? ")
    l.append(x)
compras = ["lampião de gás", "lenha"]
adiciona(compras)
print(compras)
```

```
Qual item você quer adicionar à lista? querosene
['lampião de gás', 'lenha', 'querosene']
```

- **Na chamada de função, a lista `compras` passa a ter (temporariamente) dois nomes: `compras` e `l`**

igualdade e identidade

E por que eu preciso me preocupar com isso?

```
def adiciona(l):
    x = input("Qual item você quer adicionar à lista? ")
    l.append(x)
compras = ["lampião de gás", "lenha"]
adiciona(compras)
print(compras)
```

```
Qual item você quer adicionar à lista? querosene
['lampião de gás', 'lenha', 'querosene']
```

- **Na chamada de função, a lista `compras` passa a ter (temporariamente) dois nomes: `compras` e `l`**
  - Como `append()` modifica a lista, o que acontece dentro da função se reflete fora da função

igualdade e identidade

- Não existem funções ou métodos que modifiquem diretamente um número da maneira que `append()` modifica uma lista

igualdade e identidade

- **Não existem funções ou métodos que modifiquem diretamente um número da maneira que `append()` modifica uma lista**
  - Números são *imutáveis* em python

igualdade e identidade

- **Não existem funções ou métodos que modifiquem diretamente um número da maneira que `append()` modifica uma lista**
  - Números são *imutáveis* em python
- **Por conta disso, o que acontece dentro de uma função com um número nunca afeta o que acontece fora de seu escopo**

igualdade e identidade

- **Não existem funções ou métodos que modifiquem diretamente um número da maneira que `append()` modifica uma lista**
  - Números são *imutáveis* em python
- **Por conta disso, o que acontece dentro de uma função com um número nunca afeta o que acontece fora de seu escopo**
- **Listas são *mutáveis* em python**

igualdade e identidade

- **Não existem funções ou métodos que modifiquem diretamente um número da maneira que `append()` modifica uma lista**
  - Números são *imutáveis* em python
- **Por conta disso, o que acontece dentro de uma função com um número nunca afeta o que acontece fora de seu escopo**

- **Listas são *mutáveis* em python**
- **Por conta disso, o que acontece dentro de uma função *pode ou não* afetar o que acontece fora de seu escopo**

igualdade e identidade

igualdade e identidade

```
def adiciona(l):
    novos = []
    item = input("Qual item você quer adicionar à lista? ")
    while item != "":
        novos.append(item)
        item = input("Qual item você quer adicionar à lista? ")
    l = l + novos
```

igualdade e identidade

```
def adiciona(l):
    novos = []
    item = input("Qual item você quer adicionar à lista? ")
    while item != "":
        novos.append(item)
        item = input("Qual item você quer adicionar à lista? ")
    l = l + novos
compras = ["lampião de gás", "lenha"]
adiciona(compras)
```

igualdade e identidade

```
def adiciona(l):
    novos = []
    item = input("Qual item você quer adicionar à lista? ")
    while item != "":
        novos.append(item)
        item = input("Qual item você quer adicionar à lista? ")
    l = l + novos
compras = ["lampião de gás", "lenha"]
adiciona(compras)
```

Qual item você quer adicionar à lista?

igualdade e identidade

```
def adiciona(l):
    novos = []
    item = input("Qual item você quer adicionar à lista? ")
    while item != "":
        novos.append(item)
        item = input("Qual item você quer adicionar à lista? ")
    l = l + novos
compras = ["lampião de gás", "lenha"]
adiciona(compras)
```

Qual item você quer adicionar à lista? querosene

igualdade e identidade

```
def adiciona(l):
    novos = []
    item = input("Qual item você quer adicionar à lista? ")
    while item != "":
        novos.append(item)
        item = input("Qual item você quer adicionar à lista? ")
    l = l + novos
compras = ["lampião de gás", "lenha"]
adiciona(compras)
```

```
Qual item você quer adicionar à lista? querosene
Qual item você quer adicionar à lista?
```

igualdade e identidade

```
def adiciona(l):
    novos = []
    item = input("Qual item você quer adicionar à lista? ")
    while item != "":
        novos.append(item)
        item = input("Qual item você quer adicionar à lista? ")
    l = l + novos
compras = ["lampião de gás", "lenha"]
adiciona(compras)
```

Qual item você quer adicionar à lista? querosene
Qual item você quer adicionar à lista? água

igualdade e identidade

```
def adiciona(l):
    novos = []
    item = input("Qual item você quer adicionar à lista? ")
    while item != "":
        novos.append(item)
        item = input("Qual item você quer adicionar à lista? ")
    l = l + novos
compras = ["lampião de gás", "lenha"]
adiciona(compras)
print(compras)
```

Qual item você quer adicionar à lista? querosene
Qual item você quer adicionar à lista? água

igualdade e identidade

```
def adiciona(l):
    novos = []
    item = input("Qual item você quer adicionar à lista? ")
    while item != "":
        novos.append(item)
        item = input("Qual item você quer adicionar à lista? ")
    l = l + novos
compras = ["lampião de gás", "lenha"]
adiciona(compras)
print(compras)
```

```
Qual item você quer adicionar à lista? querosene
Qual item você quer adicionar à lista? água
['lampião de gás', 'lenha']
```

igualdade e identidade

```
def adiciona(l):
    novos = []
    item = input("Qual item você quer adicionar à lista? ")
    while item != "":
        novos.append(item)
        item = input("Qual item você quer adicionar à lista? ")
    l = l + novos
compras = ["lampião de gás", "lenha"]
adiciona(compras)
print(compras)
```

```
Qual item você quer adicionar à lista? querosene
Qual item você quer adicionar à lista? água
['lampião de gás', 'lenha']
```

AAAAAHHHH!!!!

igualdade e identidade

```
def adiciona(l):
    novos = []
    item = input("Qual item você quer adicionar à lista? ")
    while item != "":
        novos.append(item)
        item = input("Qual item você quer adicionar à lista? ")
    l = l + novos
compras = ["lampião de gás", "lenha"]
adiciona(compras)
print(compras)
```

```
Qual item você quer adicionar à lista? querosene
Qual item você quer adicionar à lista? água
['lampião de gás', 'lenha']
```

AAAAAHHHH!!!!

igualdade e identidade

- Se você altera o *conteúdo* de uma variável (por exemplo, com `append()`), essa alteração afeta todos os nomes (variáveis) que se referem a aquele conteúdo

igualdade e identidade

- Se você altera o *conteúdo* de uma variável (por exemplo, com `append()`), essa alteração afeta todos os nomes (variáveis) que se referem a aquele conteúdo
- Se você altera o *valor* associado a uma variável (com o operador de atribuição `=`), essa alteração só afeta esse nome (variável) específico

igualdade e identidade

- Se você altera o *conteúdo* de uma variável (por exemplo, com `append()`), essa alteração afeta todos os nomes (variáveis) que se referem a aquele conteúdo
- Se você altera o *valor* associado a uma variável (com o operador de atribuição `=`), essa alteração só afeta esse nome (variável) específico
- Em outras linguagens a lógica é diferente!

igualdade e identidade

- **Se você altera o *conteúdo* de uma variável (por exemplo, com `append()`), essa alteração afeta todos os nomes (variáveis) que se referem a aquele conteúdo**
- **Se você altera o *valor* associado a uma variável (com o operador de atribuição `=`), essa alteração só afeta esse nome (variável) específico**
- **Em outras linguagens a lógica é diferente!**
  - Em muitos casos, o resultado na prática é equivalente, mas nem sempre

igualdade e identidade

- **Na prática:**

igualdade e identidade

- **Na prática:**
  - Alterações feitas pelo operador de atribuição (=) nunca se “propagam” para fora do escopo atual

igualdade e identidade

- **Na prática:**
  - Alterações feitas pelo operador de atribuição (=) nunca se “propagam” para fora do escopo atual
    - » *Mas, com listas,* `+=` *é equivalente a* `lista.extend()` 🤦

igualdade e identidade

- **Na prática:**
  - Alterações feitas pelo operador de atribuição (`=`) nunca se "propagam" para fora do escopo atual
    - » *Mas, com listas,* `+=` *é equivalente a* `lista.extend()` 🤦
  - Alterações feitas por chamadas de métodos (`variável.método()`) geralmente se "propagam" para fora do escopo atual

igualdade e identidade

- **Na prática:**
  - Alterações feitas pelo operador de atribuição (`=`) nunca se "propagam" para fora do escopo atual
    - » *Mas, com listas,* `+=` *é equivalente a* `lista.extend()` 🤦
  - Alterações feitas por chamadas de métodos (`variável.método()`) geralmente se "propagam" para fora do escopo atual
    - » *Mas não com tipos imutáveis, como strings* 🙄

igualdade e identidade

- **Na prática:**
  - Alterações feitas pelo operador de atribuição (`=`) nunca se "propagam" para fora do escopo atual
    - » *Mas, com listas,* `+=` *é equivalente a* `lista.extend()` 🤦
  - Alterações feitas por chamadas de métodos (`variável.método()`) geralmente se "propagam" para fora do escopo atual
    - » *Mas não com tipos imutáveis, como strings* 🙄
  - Alterações feitas com alguns (poucos) outros operadores específicos, como **`del`**, se "propagam" para fora do escopo atual

igualdade e identidade

- **Na prática:**
  - Alterações feitas pelo operador de atribuição (`=`) nunca se "propagam" para fora do escopo atual
    - » *Mas, com listas,* `+=` *é equivalente a* `lista.extend()` 🤦
  - Alterações feitas por chamadas de métodos (`variável.método()`) geralmente se "propagam" para fora do escopo atual
    - » *Mas não com tipos imutáveis, como strings* 🙄
  - Alterações feitas com alguns (poucos) outros operadores específicos, como **`del`**, se "propagam" para fora do escopo atual

igualdade e identidade

O que acontece aqui?

igualdade e identidade

O que acontece aqui?

```
lista1 = ["Beethoven", "Bach", "Berio"]
lista2 = lista1
lista1[0] = "Beatles"
print(lista2)
```

igualdade e identidade

O que acontece aqui?

```
lista1 = ["Beethoven", "Bach", "Berio"]
lista2 = lista1
lista1[0] = "Beatles"
print(lista2)
```

['Beatles', 'Bach', 'Berio']

igualdade e identidade

O que acontece aqui?

```
lista1 = ["Beethoven", "Bach", "Berio"]
lista2 = lista1
lista1[0] = "Beatles"
print(lista2)
```

['Beatles', 'Bach', 'Berio']

Embora tenhamos usado o operador de atribuição (=), ele foi usado para alterar *um elemento* da lista, ou seja, o conteúdo da lista, e não a lista em si.

igualdade e identidade

E se eu quiser fazer uma cópia “de verdade” (um *clone*)?

igualdade e identidade

E se eu quiser fazer uma cópia “de verdade” (um *clone*)?

```
def clone(l1):
    l2 = []
    for item in l1:
        l2.append(item)
    return l2
```

igualdade e identidade

E se eu quiser fazer uma cópia “de verdade” (um *clone*)?

```
def clone(l1):
    l2 = []
    for item in l1:
        l2.append(item)
    return l2
```

```
l2 = l1[:]
```

Oncotô?

Oncotô?

- **Dividir o programa em partes com *nomes* — funções**

Oncotô?

- **Dividir o programa em partes com *nomes* — funções**
  - Cada parte funciona de maneira independente, pois as variáveis “dentro” de cada uma normalmente não são visíveis para as outras — escopo

Oncotô?

- **Dividir o programa em partes com *nomes* — funções**
  - Cada parte funciona de maneira independente, pois as variáveis “dentro” de cada uma normalmente não são visíveis para as outras — escopo
- **Essas partes também podem interagir com o “mundo”**

Oncotô?

- **Dividir o programa em partes com *nomes* — funções**
  - Cada parte funciona de maneira independente, pois as variáveis "dentro" de cada uma normalmente não são visíveis para as outras — escopo
- **Essas partes também podem interagir com o "mundo"**
  - `input()` e `print()`

Oncotô?

- **Dividir o programa em partes com *nomes* — funções**
  - Cada parte funciona de maneira independente, pois as variáveis "dentro" de cada uma normalmente não são visíveis para as outras — escopo
- **Essas partes também podem interagir com o "mundo"**
  - `input()` e `print()`
- **Ou podem interagir com as outras partes do programa**

Oncotô?

- **Dividir o programa em partes com *nomes* — funções**
  - Cada parte funciona de maneira independente, pois as variáveis "dentro" de cada uma normalmente não são visíveis para as outras — escopo
- **Essas partes também podem interagir com o "mundo"**
  - `input()` e `print()`
- **Ou podem interagir com as outras partes do programa**
  - *recebendo* e *devolvendo* dados — parâmetros e **`return`**

Oncotô?

- **Dividir o programa em partes com *nomes* — funções**
  - Cada parte funciona de maneira independente, pois as variáveis “dentro” de cada uma normalmente não são visíveis para as outras — escopo
- **Essas partes também podem interagir com o “mundo”**
  - `input()` e `print()`
- **Ou podem interagir com as outras partes do programa**
  - *recebendo* e *devolvendo* dados — parâmetros e **`return`**

```
def média(a, b):
    return (a + b) / 2
```

Oncotô?

- **Dividir o programa em partes com *nomes* — funções**
  - Cada parte funciona de maneira independente, pois as variáveis "dentro" de cada uma normalmente não são visíveis para as outras — escopo
- **Essas partes também podem interagir com o "mundo"**
  - `input()` e `print()`
- **Ou podem interagir com as outras partes do programa**
  - *recebendo* e *devolvendo* dados — parâmetros e **`return`**

```
def média(a, b):
    return (a + b) / 2
```

  - Modificando uma variável global — **`global`**

Oncotô?

- **Dividir o programa em partes com *nomes* — funções**
  - Cada parte funciona de maneira independente, pois as variáveis "dentro" de cada uma normalmente não são visíveis para as outras — escopo
- **Essas partes também podem interagir com o "mundo"**
  - `input()` e `print()`
- **Ou podem interagir com as outras partes do programa**
  - *recebendo* e *devolvendo* dados — parâmetros e **`return`**

```
def média(a, b):
    return (a + b) / 2
```

  - Modificando uma variável global — **`global`**
  - **Modificando o conteúdo de uma variável (mutável) recebida como parâmetro**

Exercícios

Invertendo uma lista

Escreva uma função que recebe uma lista e devolve outra lista, igual à primeira, mas na ordem inversa

Invertendo uma lista

Escreva uma função que recebe uma lista e devolve outra lista, igual à primeira, mas na ordem inversa

```
def inverte_lista(l):
```

Invertendo uma lista

Escreva uma função que recebe uma lista e devolve outra lista, igual à primeira, mas na ordem inversa

```
def inverte_lista(l):
    l2 = []
    n = len(l) -1
    while n >= 0:
        l2.append(l[n])
        n -= 1
    return l2
```

Invertendo uma lista

Escreva uma função que recebe uma lista e devolve outra lista, igual à primeira, mas na ordem inversa

```
def inverte_lista(l):
```

Invertendo uma lista

Escreva uma função que recebe uma lista e devolve outra lista, igual à primeira, mas na ordem inversa

```
def inverte_lista(l):
    l2 = []
    for n in range(len(l) -1, -1, -1):
        l2.append(l[n])
    return l2
```

Invertendo uma lista

Escreva uma função que recebe uma lista e devolve outra lista, igual à primeira, mas na ordem inversa

```
def inverte_lista(l):
```

Invertendo uma lista

Escreva uma função que recebe uma lista e devolve outra lista, igual à primeira, mas na ordem inversa

```
def inverte_lista(l):
    l.reverse()
    return l
```

Invertendo uma lista

Escreva uma função que recebe uma lista e devolve outra lista, igual à primeira, mas na ordem inversa

```
def inverte_lista(l):
    l.reverse()
    return l
```

Invertendo uma lista

Escreva uma função que recebe uma lista e devolve outra lista, igual à primeira, mas na ordem inversa

```
def inverte_lista(l):
```

Invertendo uma lista

Escreva uma função que recebe uma lista e devolve outra lista, igual à primeira, mas na ordem inversa

```
def inverte_lista(l):
    l = l[:]
    l.reverse()
    return l
```

Invertendo uma lista

Escreva uma função que recebe uma lista e devolve outra lista, igual à primeira, mas na ordem inversa

```
def inverte_lista(l):
```

Invertendo uma lista

Escreva uma função que recebe uma lista e devolve outra lista, igual à primeira, mas na ordem inversa

```
def inverte_lista(l):
    l = l[-1::-1]
    return l
```

Coleções de coleções

- Os itens de uma coleção/lista podem ser de qualquer tipo

Coleções de coleções

- **Os itens de uma coleção/lista podem ser de qualquer tipo**
  - (E, em python, podem inclusive ser de tipos diferentes)

Coleções de coleções

- **Os itens de uma coleção/lista podem ser de qualquer tipo**
  - (E, em python, podem inclusive ser de tipos diferentes)
- **Um item pode, inclusive, ser uma lista!**

Coleções de coleções

- **Os itens de uma coleção/lista podem ser de qualquer tipo**
  - (E, em python, podem inclusive ser de tipos diferentes)
- **Um item pode, inclusive, ser uma lista!**

Coleções de coleções

- **Os itens de uma coleção/lista podem ser de qualquer tipo**
  - (E, em python, podem inclusive ser de tipos diferentes)
- **Um item pode, inclusive, ser uma lista!**

```
lista = [1, 2, 3]
```

Coleções de coleções

- **Os itens de uma coleção/lista podem ser de qualquer tipo**
  - (E, em python, podem inclusive ser de tipos diferentes)
- **Um item pode, inclusive, ser uma lista!**

```
lista = [1, 2, 3]
outra_lista = [4, 5, 6]
```

Coleções de coleções

- **Os itens de uma coleção/lista podem ser de qualquer tipo**
  - (E, em python, podem inclusive ser de tipos diferentes)
- **Um item pode, inclusive, ser uma lista!**

```
lista = [1, 2, 3]
outra_lista = [4, 5, 6]
lista.extend(outra_lista)
```

Coleções de coleções

- **Os itens de uma coleção/lista podem ser de qualquer tipo**
  - (E, em python, podem inclusive ser de tipos diferentes)
- **Um item pode, inclusive, ser uma lista!**

```
lista = [1, 2, 3]
outra_lista = [4, 5, 6]
lista.extend(outra_lista)
print(lista)
```

Coleções de coleções

- **Os itens de uma coleção/lista podem ser de qualquer tipo**
  - (E, em python, podem inclusive ser de tipos diferentes)
- **Um item pode, inclusive, ser uma lista!**

```
lista = [1, 2, 3]
outra_lista = [4, 5, 6]
lista.extend(outra_lista)
print(lista)
```

[1, 2, 3, 4, 5, 6]

Coleções de coleções

- **Os itens de uma coleção/lista podem ser de qualquer tipo**
  - (E, em python, podem inclusive ser de tipos diferentes)
- **Um item pode, inclusive, ser uma lista!**

```
lista = [1, 2, 3]
outra_lista = [4, 5, 6]
lista.extend(outra_lista)
print(lista)
lista = [1, 2, 3]
```

[1, 2, 3, 4, 5, 6]

Coleções de coleções

- **Os itens de uma coleção/lista podem ser de qualquer tipo**
  - (E, em python, podem inclusive ser de tipos diferentes)
- **Um item pode, inclusive, ser uma lista!**

```
lista = [1, 2, 3]
outra_lista = [4, 5, 6]
lista.extend(outra_lista)
print(lista)
lista = [1, 2, 3]
lista.append(outra_lista)
```

[1, 2, 3, 4, 5, 6]

Coleções de coleções

- **Os itens de uma coleção/lista podem ser de qualquer tipo**
  - (E, em python, podem inclusive ser de tipos diferentes)
- **Um item pode, inclusive, ser uma lista!**

```
lista = [1, 2, 3]
outra_lista = [4, 5, 6]
lista.extend(outra_lista)
print(lista)
lista = [1, 2, 3]
lista.append(outra_lista)
print(lista)
```

[1, 2, 3, 4, 5, 6]

Coleções de coleções

- **Os itens de uma coleção/lista podem ser de qualquer tipo**
  - (E, em python, podem inclusive ser de tipos diferentes)
- **Um item pode, inclusive, ser uma lista!**

```
lista = [1, 2, 3]
outra_lista = [4, 5, 6]
lista.extend(outra_lista)
print(lista)
lista = [1, 2, 3]
lista.append(outra_lista)
print(lista)
```

[1, 2, 3, 4, 5, 6]

[1, 2, 3, [4, 5, 6]]

Coleções de coleções

Coleções de coleções

```
lista1 = [1, 2, 3]
lista2 = [4, 5, 6]
```

Coleções de coleções

```
lista1 = [1, 2, 3]
lista2 = [4, 5, 6]
juntas = [lista1, lista2]
```

Coleções de coleções

```
lista1 = [1, 2, 3]
lista2 = [4, 5, 6]
juntas = [lista1, lista2]
print(juntas)
```

Coleções de coleções

```
lista1 = [1, 2, 3]
lista2 = [4, 5, 6]
juntas = [lista1, lista2]
print(juntas)
```

[[1, 2, 3], [4, 5, 6]]

Coleções de coleções

```
lista1 = [1, 2, 3]
lista2 = [4, 5, 6]
juntas = [lista1, lista2]
print(juntas)
lista1.append(lista2)
```

[[1, 2, 3], [4, 5, 6]]

Coleções de coleções

```
lista1 = [1, 2, 3]
lista2 = [4, 5, 6]
juntas = [lista1, lista2]
print(juntas)
lista1.append(lista2)
lista2.append("olá")
```

[[1, 2, 3], [4, 5, 6]]

Coleções de coleções

```
lista1 = [1, 2, 3]
lista2 = [4, 5, 6]
juntas = [lista1, lista2]
print(juntas)
lista1.append(lista2)
lista2.append("olá")
print(lista2)
print(lista1)
print(juntas)
```

[[1, 2, 3], [4, 5, 6]]

Coleções de coleções

```
lista1 = [1, 2, 3]
lista2 = [4, 5, 6]
juntas = [lista1, lista2]
print(juntas)
lista1.append(lista2)
lista2.append("olá")
print(lista2)
print(lista1)
print(juntas)
```

[[1, 2, 3], [4, 5, 6]]

[4, 5, 6, 'olá']

Coleções de coleções

```
lista1 = [1, 2, 3]
lista2 = [4, 5, 6]
juntas = [lista1, lista2]
print(juntas)
lista1.append(lista2)
lista2.append("olá")
print(lista2)
print(lista1)
print(juntas)
```

[[1, 2, 3], [4, 5, 6]]

[4, 5, 6, 'olá']

[1, 2, 3, [4, 5, 6, 'olá']]

Coleções de coleções

```
lista1 = [1, 2, 3]
lista2 = [4, 5, 6]
juntas = [lista1, lista2]
print(juntas)
lista1.append(lista2)
lista2.append("olá")
print(lista2)
print(lista1)
print(juntas)
```

[[1, 2, 3], [4, 5, 6]]

[4, 5, 6, 'olá']

[1, 2, 3, [4, 5, 6, 'olá']]

[[1, 2, 3, [4, 5, 6, 'olá']], [4, 5, 6, 'olá']]

Mas para que fazer um rolo desses?!?

Mas para que fazer um rolo desses?!?

ABSTRAÇÕES

Abstrações

- As linguagens de programação oferecem “peças” que usamos para construir nossos programas

Abstrações

- **As linguagens de programação oferecem "peças" que usamos para construir nossos programas**
  - variáveis

Abstrações

- **As linguagens de programação oferecem "peças" que usamos para construir nossos programas**
  - variáveis
  - funções

Abstrações

- **As linguagens de programação oferecem "peças" que usamos para construir nossos programas**
  - variáveis
  - funções
  - coleções

Abstrações

- **As linguagens de programação oferecem "peças" que usamos para construir nossos programas**
  - variáveis
  - funções
  - coleções
  - ...

Abstrações

- **As linguagens de programação oferecem "peças" que usamos para construir nossos programas**
  - variáveis
  - funções
  - coleções
  - ...
- **Com essas "peças", construímos abstrações que se aproximam dos problemas reais que queremos solucionar**

Abstrações

- **As linguagens de programação oferecem "peças" que usamos para construir nossos programas**
  - variáveis
  - funções
  - coleções
  - ...
- **Com essas "peças", construímos abstrações que se aproximam dos problemas reais que queremos solucionar**
  - Assim como as peças de um Lego podem ser usadas para construir formas diversas

Abstrações

- **As linguagens de programação oferecem "peças" que usamos para construir nossos programas**
  - variáveis
  - funções
  - coleções
  - ...
- **Com essas "peças", construímos abstrações que se aproximam dos problemas reais que queremos solucionar**
  - Assim como as peças de um Lego podem ser usadas para construir formas diversas
    - » *Por exemplo, nosso exercício com polinômios*

Abstrações

- **As linguagens de programação oferecem "peças" que usamos para construir nossos programas**
  - variáveis
  - funções
  - coleções
  - ...
- **Com essas "peças", construímos abstrações que se aproximam dos problemas reais que queremos solucionar**
  - Assim como as peças de um Lego podem ser usadas para construir formas diversas
    - » *Por exemplo, nosso exercício com polinômios*
- **Podemos usar uma coleção de coleções para criar uma nova abstração:**

Abstrações

- **As linguagens de programação oferecem "peças" que usamos para construir nossos programas**
  - variáveis
  - funções
  - coleções
  - ...
- **Com essas "peças", construímos abstrações que se aproximam dos problemas reais que queremos solucionar**
  - Assim como as peças de um Lego podem ser usadas para construir formas diversas
    - » *Por exemplo, nosso exercício com polinômios*
- **Podemos usar uma coleção de coleções para criar uma nova abstração: Matrizes**

Matrizes

```
tabela_de_preços = [["item", "à vista", "a prazo"]]
```

Matrizes

```
tabela_de_preços = [["item", "à vista", "a prazo"]]
tabela_de_preços.append(["inhame", 10, 11])
```

Matrizes

```
tabela_de_preços = [["item", "à vista", "a prazo"]]
tabela_de_preços.append(["inhame", 10, 11])
tabela_de_preços.append(["batata", 13, 15])
tabela_de_preços.append(["aipim", 12, 14])
```

Matrizes

```
tabela_de_preços = [["item", "à vista", "a prazo"]]
tabela_de_preços.append(["inhame", 10, 11])
tabela_de_preços.append(["batata", 13, 15])
tabela_de_preços.append(["aipim", 12, 14])
print(tabela_de_preços)
```

Matrizes

```
tabela_de_preços = [["item", "à vista", "a prazo"]]
tabela_de_preços.append(["inhame", 10, 11])
tabela_de_preços.append(["batata", 13, 15])
tabela_de_preços.append(["aipim", 12, 14])
print(tabela_de_preços)
```

[['item', 'à vista', 'a prazo'], ['inhame', 10, 11], ['batata', 13, 15], ['aipim', 12, 14]]

Matrizes

```
tabela_de_preços = [["item", "à vista", "a prazo"]]
tabela_de_preços.append(["inhame", 10, 11])
tabela_de_preços.append(["batata", 13, 15])
tabela_de_preços.append(["aipim", 12, 14])
```

Matrizes

```
tabela_de_preços = [["item", "à vista", "a prazo"]]
tabela_de_preços.append(["inhame", 10, 11])
tabela_de_preços.append(["batata", 13, 15])
tabela_de_preços.append(["aipim", 12, 14])
for line in tabela_de_preços:
    for item in line:
        print(item, end="\t")
    print()
```

Matrizes

```
tabela_de_preços = [["item", "à vista", "a prazo"]]
tabela_de_preços.append(["inhame", 10, 11])
tabela_de_preços.append(["batata", 13, 15])
tabela_de_preços.append(["aipim", 12, 14])
for line in tabela_de_preços:
    for item in line:
        print(item, end="\t")
    print()
```

```
item    à vista a prazo
inhame  10      11
batata  13      15
aipim   12      14
```

Matrizes

```
tabela_de_preços = [["item", "à vista", "a prazo"]]
tabela_de_preços.append(["inhame", 10, 11])
tabela_de_preços.append(["batata", 13, 15])
tabela_de_preços.append(["aipim", 12, 14])
for line in tabela_de_preços:
    for item in line:
        print(item, end="\t")
    print()
```

```
item    à vista a prazo
inhame  10      11
batata  13      15
aipim   12      14
```

Matrizes

```
tabela_de_preços = [["item", "à vista", "a prazo"]]
tabela_de_preços.append(["inhame", 10, 11])
tabela_de_preços.append(["batata", 13, 15])
tabela_de_preços.append(["aipim", 12, 14])
for line in tabela_de_preços:
    for item in line:
        print(item, end="\t")
    print()
```

```
item    à vista a prazo
inhame  10      11
batata  13      15
aipim   12      14
```

Matrizes

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix} = (a_{i,j})_{1 \leq i \leq m, 1 \leq j \leq n}$$

Matrizes

$$A = \begin{bmatrix} a_{0,0} & a_{0,1} & \cdots & a_{1,n-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m-1,1} & a_{m-1,2} & \cdots & a_{m-1,n-1} \end{bmatrix} = (a_{i,j})_{0 \leq i < m, 0 \leq j < n}$$

Matrizes

```
tabela_de_preços = [["item", "à vista", "a prazo"]]
tabela_de_preços.append(["inhame", 10, 11])
tabela_de_preços.append(["batata", 13, 15])
tabela_de_preços.append(["aipim", 12, 14])
```

Matrizes

```
tabela_de_preços = [["item", "à vista", "a prazo"]]
tabela_de_preços.append(["inhame", 10, 11])
tabela_de_preços.append(["batata", 13, 15])
tabela_de_preços.append(["aipim", 12, 14])
l2 = tabela_de_preços[2]
```

Matrizes

```
tabela_de_preços = [["item", "à vista", "a prazo"]]
tabela_de_preços.append(["inhame", 10, 11])
tabela_de_preços.append(["batata", 13, 15])
tabela_de_preços.append(["aipim", 12, 14])
l2 = tabela_de_preços[2]
print(l2[0])
```

Matrizes

```
tabela_de_preços = [["item", "à vista", "a prazo"]]
tabela_de_preços.append(["inhame", 10, 11])
tabela_de_preços.append(["batata", 13, 15])
tabela_de_preços.append(["aipim", 12, 14])
l2 = tabela_de_preços[2]
print(l2[0])
```

```
batata
```

Matrizes

```
tabela_de_preços = [["item", "à vista", "a prazo"]]
tabela_de_preços.append(["inhame", 10, 11])
tabela_de_preços.append(["batata", 13, 15])
tabela_de_preços.append(["aipim", 12, 14])
l2 = tabela_de_preços[2]
print(l2[0])
print(tabela_de_preços[2][0])
```

```
batata
```

Matrizes

```
tabela_de_preços = [["item", "à vista", "a prazo"]]
tabela_de_preços.append(["inhame", 10, 11])
tabela_de_preços.append(["batata", 13, 15])
tabela_de_preços.append(["aipim", 12, 14])
l2 = tabela_de_preços[2]
print(l2[0])
print(tabela_de_preços[2][0])
```

```
batata
batata
```

Matrizes

```
tabela_de_preços = [["item", "à vista", "a prazo"]]
tabela_de_preços.append(["inhame", 10, 11])
tabela_de_preços.append(["batata", 13, 15])
tabela_de_preços.append(["aipim", 12, 14])
l2 = tabela_de_preços[2]
print(l2[0])
print(tabela_de_preços[2][0])
```

```
batata
batata
```

Nada obriga que todas as linhas tenham o mesmo número de elementos

Nada obriga que todas as linhas tenham o mesmo número de elementos

Matrizes desse tipo funcionam por *convenção*

Matrizes

Também podemos criar uma matriz assim:

```
MATRIZ = [[4, 1, 8, 3],
          [2, 5, 7, 0],
          [6, 9, 0, 3]]
```

Matrizes

E para criar uma matriz de zeros, digamos, 5x3?

Matrizes

E para criar uma matriz de zeros, digamos, 5x3?

```
linha = [0]*3
```

Matrizes

E para criar uma matriz de zeros, digamos, 5x3?

```
linha = [0]*3
matriz = [linha]*5
```

Matrizes

E para criar uma matriz de zeros, digamos, 5x3?

```
linha = [0]*3
matriz = [linha]*5
print(matriz[2][2])
```

Matrizes

E para criar uma matriz de zeros, digamos, 5x3?

```
linha = [0]*3
matriz = [linha]*5
print(matriz[2][2])
```

0

Matrizes

E para criar uma matriz de zeros, digamos, 5x3?

```
linha = [0]*3
matriz = [linha]*5
print(matriz[2][2])
```

0

Matrizes

E para criar uma matriz de zeros, digamos, 5x3?

```
linha = [0]*3
matriz = [linha]*5
print(matriz[2][2])
matriz[1][1] = 3
```

0

Matrizes

E para criar uma matriz de zeros, digamos, 5x3?

```
linha = [0]*3
matriz = [linha]*5
print(matriz[2][2])
matriz[1][1] = 3
print(matriz[1][1])
```

```
0
```

Matrizes

E para criar uma matriz de zeros, digamos, 5x3?

```
linha = [0]*3
matriz = [linha]*5
print(matriz[2][2])
matriz[1][1] = 3
print(matriz[1][1])
```

0

3

Matrizes

E para criar uma matriz de zeros, digamos, 5x3?

```
linha = [0]*3
matriz = [linha]*5
print(matriz[2][2])
matriz[1][1] = 3
print(matriz[1][1])
```

```
0
3
```

Matrizes

E para criar uma matriz de zeros, digamos, 5x3?

```
linha = [0]*3
matriz = [linha]*5
print(matriz[2][2])
matriz[1][1] = 3
print(matriz[1][1])
print(matriz[2][1])
```

```
0
3
```

Matrizes

E para criar uma matriz de zeros, digamos, 5x3?

```
linha = [0]*3
matriz = [linha]*5
print(matriz[2][2])
matriz[1][1] = 3
print(matriz[1][1])
print(matriz[2][1])
```

0

3

3

Matrizes

E para criar uma matriz de zeros, digamos, 5x3?

```
linha = [0]*3
matriz = [linha]*5
print(matriz[2][2])
matriz[1][1] = 3
print(matriz[1][1])
print(matriz[2][1])
```

```
0
3
3
```

Matrizes

E para criar uma matriz de zeros, digamos, 5x3?

```
linha = [0]*3
matriz = [linha]*5
```

Matrizes

E para criar uma matriz de zeros, digamos, 5x3?

```
linha = [0]*3
matriz = [linha]*5
matriz[1][1] = 3
```

Matrizes

E para criar uma matriz de zeros, digamos, 5x3?

```
linha = [0]*3
matriz = [linha]*5
matriz[1][1] = 3
print(matriz)
```

Matrizes

E para criar uma matriz de zeros, digamos, 5x3?

```
linha = [0]*3
matriz = [linha]*5
matriz[1][1] = 3
print(matriz)
```

[[0, 3, 0], [0, 3, 0], [0, 3, 0], [0, 3, 0], [0, 3, 0]]

Matrizes

E para criar uma matriz de zeros, digamos, 5x3?

```
linha = [0]*3
matriz = [linha]*5
matriz[1][1] = 3
print(matriz)
```

[[0, 3, 0], [0, 3, 0], [0, 3, 0], [0, 3, 0], [0, 3, 0]]

Matrizes

E para criar uma matriz de zeros, digamos, 5x3?

```
linha = [0]*3
matriz = [linha]*5
matriz[1][1] = 3
print(matriz)
```

[[0, 3, 0], [0, 3, 0], [0, 3, 0], [0, 3, 0], [0, 3, 0]]

Matrizes

E para criar uma matriz de zeros, digamos, 5x3?

```
linha = [0]*3
matriz = []
for i in range(5):
    matriz.append(linha[:])
```

Matrizes

E para criar uma matriz de zeros, digamos, 5x3?

```
linha = [0]*3
matriz = []
for i in range(5):
    matriz.append(linha[:])
matriz[1][1] = 3
```

Matrizes

E para criar uma matriz de zeros, digamos, 5x3?

```
linha = [0]*3
matriz = []
for i in range(5):
    matriz.append(linha[:])
matriz[1][1] = 3
print(matriz[1][1])
```

Matrizes

E para criar uma matriz de zeros, digamos, 5x3?

```
linha = [0]*3
matriz = []
for i in range(5):
    matriz.append(linha[:])
matriz[1][1] = 3
print(matriz[1][1])
```

3

Matrizes

E para criar uma matriz de zeros, digamos, 5x3?

```
linha = [0]*3
matriz = []
for i in range(5):
    matriz.append(linha[:])
matriz[1][1] = 3
print(matriz[1][1])
print(matriz[2][1])
```

```
3
```

Matrizes

E para criar uma matriz de zeros, digamos, 5x3?

```
linha = [0]*3
matriz = []
for i in range(5):
    matriz.append(linha[:])
matriz[1][1] = 3
print(matriz[1][1])
print(matriz[2][1])
```

```
3
0
```

Matrizes

E para criar uma matriz de zeros, digamos, 5x3?

```
linha = [0]*3
matriz = []
for i in range(5):
    matriz.append(linha[:])
matriz[1][1] = 3
print(matriz[1][1])
print(matriz[2][1])
```

```
3
0
```

Matrizes

```
def cria_matriz(linhas, colunas, valor_inicial):
```

Matrizes

```
def cria_matriz(linhas, colunas, valor_inicial):
    umalinha = [valor_inicial] *colunas
```

Matrizes

```
def cria_matriz(linhas, colunas, valor_inicial):
    umalinha = [valor_inicial] *colunas
    matriz = []
```

Matrizes

```
def cria_matriz(linhas, colunas, valor_inicial):
    umalinha = [valor_inicial] *colunas
    matriz = []
    for l in range(linhas):
```

Matrizes

```
def cria_matriz(linhas, colunas, valor_inicial):
    umalinha = [valor_inicial] *colunas
    matriz = []
    for l in range(linhas):
        matriz.append(umalinha[:])
```

Matrizes

```
def cria_matriz(linhas, colunas, valor_inicial):
    umalinha = [valor_inicial] *colunas
    matriz = []
    for l in range(linhas):
        matriz.append(umalinha[:])
    return matriz
```

Matrizes

```
def cria_matriz(linhas, colunas, valor_inicial):

    matriz = []
    for l in range(linhas):
        matriz.append([valor_inicial]*colunas)
    return matriz
```

Matrizes

```
def cria_matriz(linhas, colunas, valor_inicial):

    matriz = []
    for l in range(linhas):
        matriz.append([valor_inicial]*colunas)
    return matriz
```

Matrizes

```python
def cria_matriz(linhas, colunas, valor_inicial):

    matriz = []
    for l in range(linhas):
        matriz.append([valor_inicial]*colunas)
    return matriz
```

```python
def cria_matriz(linhas, colunas, valor_inicial):
    matriz = []
    for i in range(linhas):
        linha = []
        for j in range(colunas):
            linha.append(valor_inicial)
        matriz.append(linha)
    return matriz
```

Matrizes

```python
def cria_matriz(linhas, colunas, valor_inicial):

    matriz = []
    for l in range(linhas):
        matriz.append([valor_inicial]*colunas)
    return matriz
```

```python
def cria_matriz(linhas, colunas, valor_inicial):
    matriz = []
    for i in range(linhas):
        linha = []
        for j in range(colunas):
            linha.append(valor_inicial)
        matriz.append(linha)
    return matriz
```

Impressão de matrizes

```
def imprime_matriz(A):
```

Impressão de matrizes

```
def imprime_matriz(A):
    for linha in A:
```

Impressão de matrizes

```
def imprime_matriz(A):
    for linha in A:
        for col in linha:
```

Impressão de matrizes

```
def imprime_matriz(A):
    for linha in A:
        for col in linha:
            print("{:4}".format(col), end="")
```

Impressão de matrizes

```
def imprime_matriz(A):
    for linha in A:
        for col in linha:
            print("{:4}".format(col), end="")
        print()
```

Soma de matrizes

$$\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} \\ a_{1,0} & a_{1,1} & a_{1,2} \end{bmatrix} + \begin{bmatrix} b_{0,0} & b_{0,1} & b_{0,2} \\ b_{1,0} & b_{1,1} & b_{1,2} \end{bmatrix} = \begin{bmatrix} a_{0,0} + b_{0,0} & a_{0,1} + b_{0,1} & a_{0,2} + b_{0,2} \\ a_{1,0} + b_{1,0} & a_{1,1} + b_{1,1} & a_{1,2} + b_{1,2} \end{bmatrix}$$

Soma de matrizes

$$\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} \\ a_{1,0} & a_{1,1} & a_{1,2} \end{bmatrix} + \begin{bmatrix} b_{0,0} & b_{0,1} & b_{0,2} \\ b_{1,0} & b_{1,1} & b_{1,2} \end{bmatrix} = \begin{bmatrix} a_{0,0} + b_{0,0} & a_{0,1} + b_{0,1} & a_{0,2} + b_{0,2} \\ a_{1,0} + b_{1,0} & a_{1,1} + b_{1,1} & a_{1,2} + b_{1,2} \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 100 & 200 & 300 \\ 400 & 500 & 600 \end{bmatrix} = \begin{bmatrix} 101 & 202 & 303 \\ 404 & 505 & 606 \end{bmatrix}$$

Soma de matrizes

```
def soma_matrizes(A, B):




    return C
```

Soma de matrizes

```
def soma_matrizes(A, B):
    nlinhas = len(A)









    return C
```

Soma de matrizes

```
def soma_matrizes(A, B):
    nlinhas = len(A)
    ncols = len(A[0])




    return C
```

Soma de matrizes

```
def soma_matrizes(A, B):
    nlinhas = len(A)
    ncols = len(A[0])
    C = cria_matriz(nlinhas, ncols, 0)




    return C
```

Soma de matrizes

```
def soma_matrizes(A, B):
    nlinhas = len(A)
    ncols = len(A[0])
    C = cria_matriz(nlinhas, ncols, 0)
    for l in range(nlinhas):



    return C
```

Soma de matrizes

```
def soma_matrizes(A, B):
    nlinhas = len(A)
    ncols = len(A[0])
    C = cria_matriz(nlinhas, ncols, 0)
    for l in range(nlinhas):
        for c in range(ncols):

    return C
```

Soma de matrizes

```
def soma_matrizes(A, B):
    nlinhas = len(A)
    ncols = len(A[0])
    C = cria_matriz(nlinhas, ncols, 0)
    for l in range(nlinhas):
        for c in range(ncols):
            C[l][c] = A[l][c] + B[l][c]
    return C
```

Soma de matrizes

Soma de matrizes

```
A = [[1, 2, 3],
     [4, 5, 6]]
```

Soma de matrizes

```
A = [[1, 2, 3],
     [4, 5, 6]]

B = [[100, 200, 300],
     [400, 500, 600]]
```

Soma de matrizes

```
A = [[1, 2, 3],
     [4, 5, 6]]

B = [[100, 200, 300],
     [400, 500, 600]]

C = soma_matrizes(A, B)
```

Soma de matrizes

```
A = [[1, 2, 3],
     [4, 5, 6]]

B = [[100, 200, 300],
     [400, 500, 600]]

C = soma_matrizes(A, B)
imprime_matriz(C)
```

Soma de matrizes

```
A = [[1, 2, 3],
     [4, 5, 6]]

B = [[100, 200, 300],
     [400, 500, 600]]

C = soma_matrizes(A, B)
imprime_matriz(C)
```

---

101 202 303

404 505 606

Matrizes

Este é um assunto novo?

Matrizes

Este é um assunto novo?

Não!

Matrizes

Este é um assunto novo?

Não! — Listas e laços encaixados

Matrizes

Este é um assunto novo?

Não! — Listas e laços encaixados

Sim!

Matrizes

Este é um assunto novo?

Não! — Listas e laços encaixados

Sim! — Matrizes são uma abstração diferente