MAC 115 — Introdução à Ciência da Computação

Aula 17

Nelson Lago

IF noturno – 2023

Previously on MAC 115...

Repetições com coleções

```
primos = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
n = 0
while n < len(primos):
  print("O número", primos[n], "é primo")
  n += 1
```

Repetições com coleções

```
primos = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
n = 0
while n < len(primos):
  print("O número", primos[n], "é primo")
  n += 1
```

```
primos = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
for p in primos:
  print("O número", p, "é primo")
```

A função range()

```
range(início, final, passo)
```

A função `range()`

```
range(início, final, passo)
```

- **O início pode ser omitido; o padrão é zero**

A função range()

```
range(início, final, passo)
```

- **O início pode ser omitido; o padrão é zero**
- **O passo pode ser omitido; o padrão é um**

A função `range()`

```
range(início, final, passo)
```

- **O início pode ser omitido; o padrão é zero**
- **O passo pode ser omitido; o padrão é um**
- **Se há dois parâmetros, eles são `início` e `final`**

A função `range()`

```
range(início, final, passo)
```

- **O início pode ser omitido; o padrão é zero**
- **O passo pode ser omitido; o padrão é um**
- **Se há dois parâmetros, eles são `início` e `final`**

O intervalo é sempre
fechado no início e aberto no final

A função range()

```
range(início, final, passo)
```

A função range()

```
range(início, final, passo)
```

O intervalo é sempre
fechado no início e aberto no final

A função range()

```
range(início, final, passo)
```

O intervalo é sempre

fechado no início e aberto no final

primeiro = início

A função `range()`

```
range(início, final, passo)
```

O intervalo é sempre

fechado no início e aberto no final

$$\text{primeiro} = \text{início}$$

$$|\text{último}| < |\text{final}|$$

A função range()

```
range(início, final, passo)
```

O intervalo é sempre

fechado no início e aberto no final

primeiro = início

|último| < |final|

A função range()

```
range(início, final, passo)
```

A função range()

```
range(início, final, passo)
```

O intervalo é sempre
fechado no início e aberto no final

A função `range()`

`range(início, final, passo)`

O intervalo é sempre

fechado no início e aberto no final

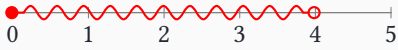
`range(4)` → `range(0, 4)` → de zero a três! (quatro elementos)

A função `range()`

`range(início, final, passo)`

O intervalo é sempre
fechado no início e aberto no final

`range(4)` → `range(0, 4)` → de zero a três! (quatro elementos)

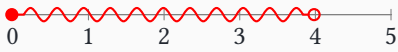


A função `range()`

`range(início, final, passo)`

O intervalo é sempre fechado no início e aberto no final

`range(4)` → `range(0, 4)` → de zero a três! (quatro elementos)



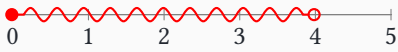
`range(2, 7)` → de dois a seis! (cinco elementos)

A função `range()`

`range(início, final, passo)`

O intervalo é sempre fechado no início e aberto no final

`range(4)` → `range(0, 4)` → de zero a três! (quatro elementos)



`range(2, 7)` → de dois a seis! (cinco elementos)

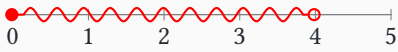
`range(1, 10, 2)` → de um a nove! (cinco elementos)

A função range()

`range(início, final, passo)`

O intervalo é sempre fechado no início e aberto no final

`range(4)` → `range(0, 4)` → de zero a três! (quatro elementos)



`range(2, 7)` → de dois a seis! (cinco elementos)

`range(1, 10, 2)` → de um a nove! (cinco elementos)

`range(1, 11, 2)` → de um a nove! (cinco elementos)

A função range()

```
range(início, final, passo)
```

A função range()

```
range(início, final, passo)
```

O intervalo é sempre

fechado no início e aberto no final

A função `range()`

`range(início, final, passo)`

O intervalo é sempre
fechado no início e aberto no final

O número total de elementos é $\left\lceil \frac{\text{final} - \text{início}}{\text{passo}} \right\rceil$

And now for something completely different

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']
```

Brincando com listas

Esta é só para facilitar:

Brincando com listas

Esta é só para facilitar:

```
achei = False
for iguaria in guloseimas:
    if iguaria == "espinafre":
        achei = True
if not achei:
    print("Precisa comprar espinafre!")
```

Brincando com listas

Esta é só para facilitar:

```
achei = False
for iguaria in guloseimas:
    if iguaria == "espinafre":
        achei = True
if not achei:
    print("Precisa comprar espinafre!")
```

```
if not "espinafre" in guloseimas:
    print("Precisa comprar espinafre!")
```

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

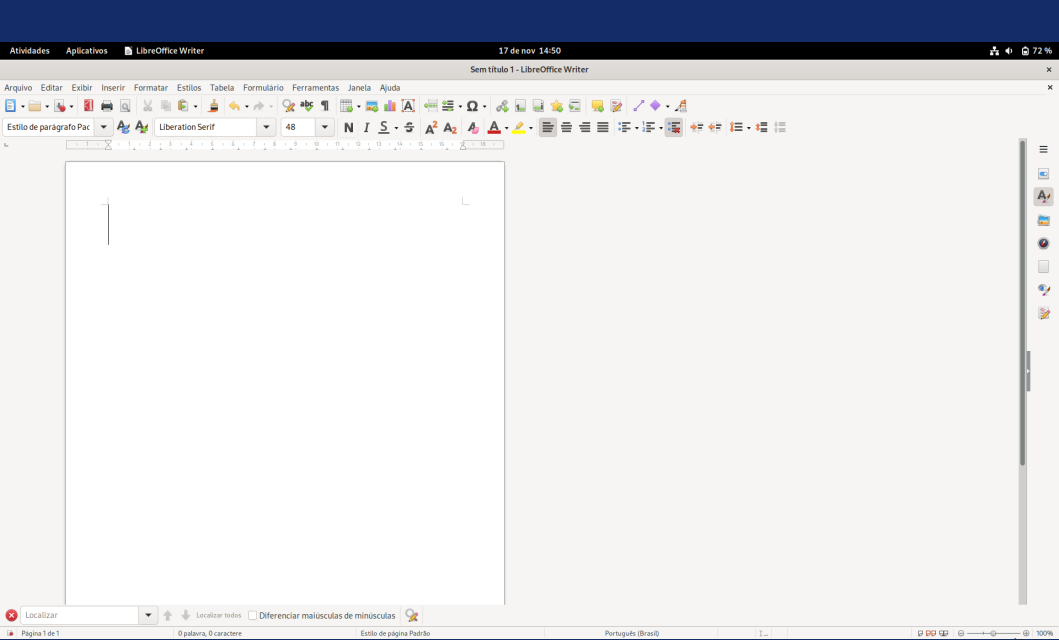
- **Funções em geral representam "ações" que esperamos que o computador realize**
- **Existem dois tipos de função em python: *funções* e *métodos***
- **Funções seguem o formato *verbo + objeto*:**
  - `print(blah)`
  - `éPrimo(x)`
  - ...
- **Métodos seguem o formato *objeto + verbo***

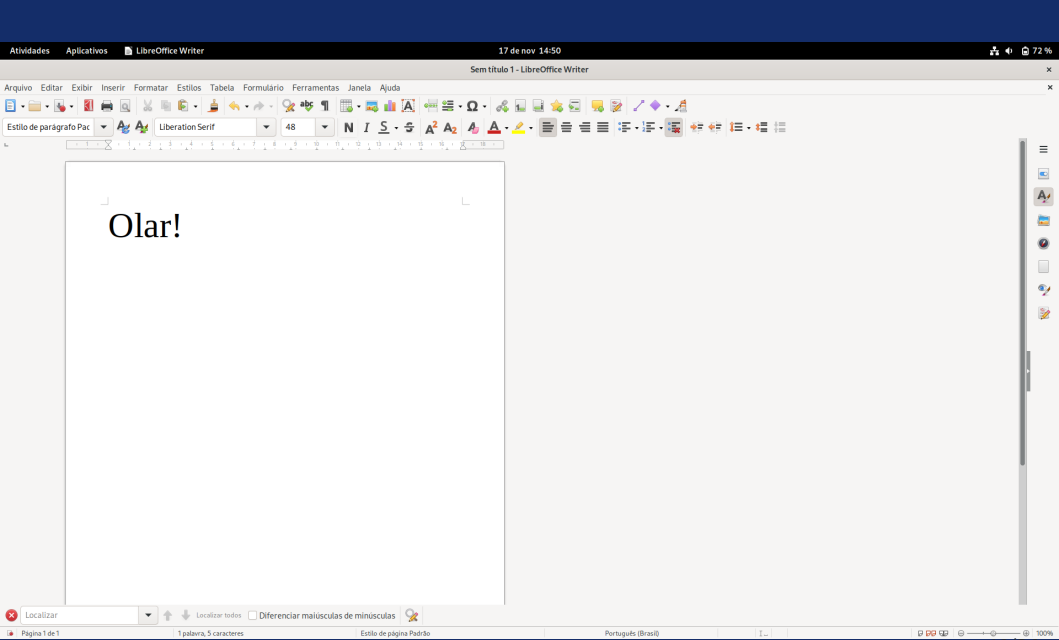
Métodos e funções

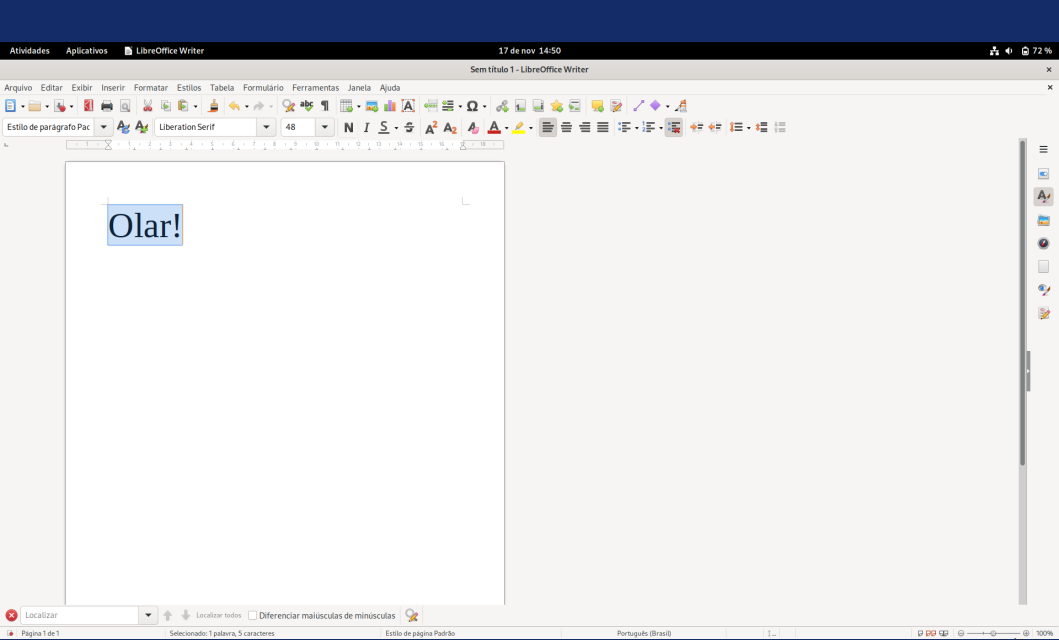
- **Funções em geral representam "ações" que esperamos que o computador realize**
- **Existem dois tipos de função em python: *funções* e *métodos***
- **Funções seguem o formato *verbo + objeto*:**
  - `print(blah)`
  - `éPrimo(x)`
  - ...
- **Métodos seguem o formato *objeto + verbo***
  - `lista.append()`

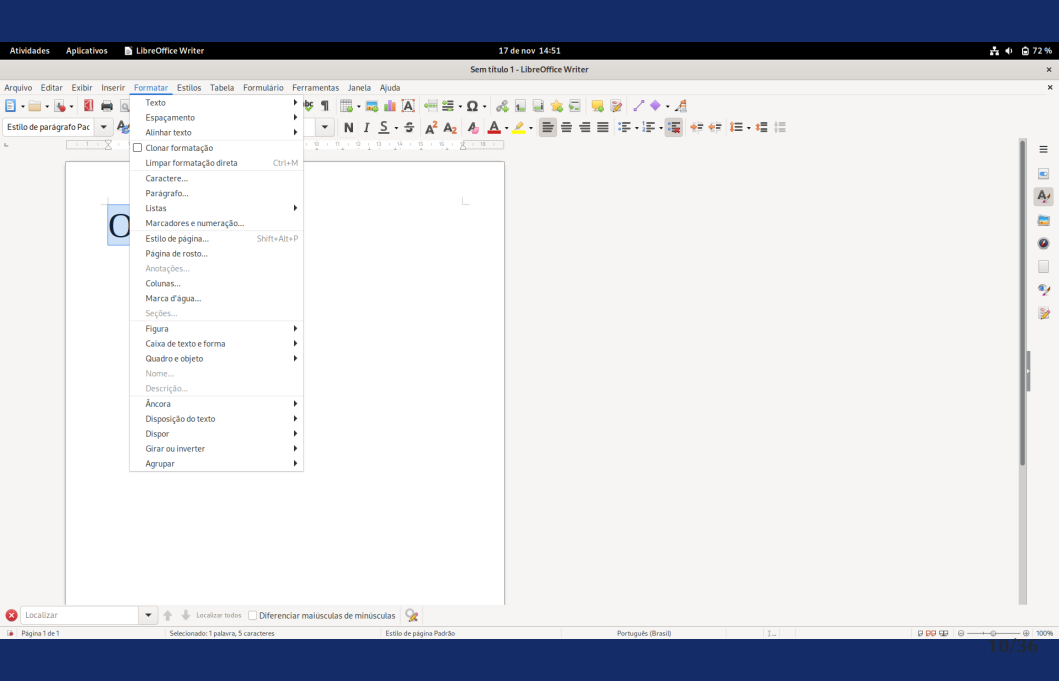
Métodos e funções

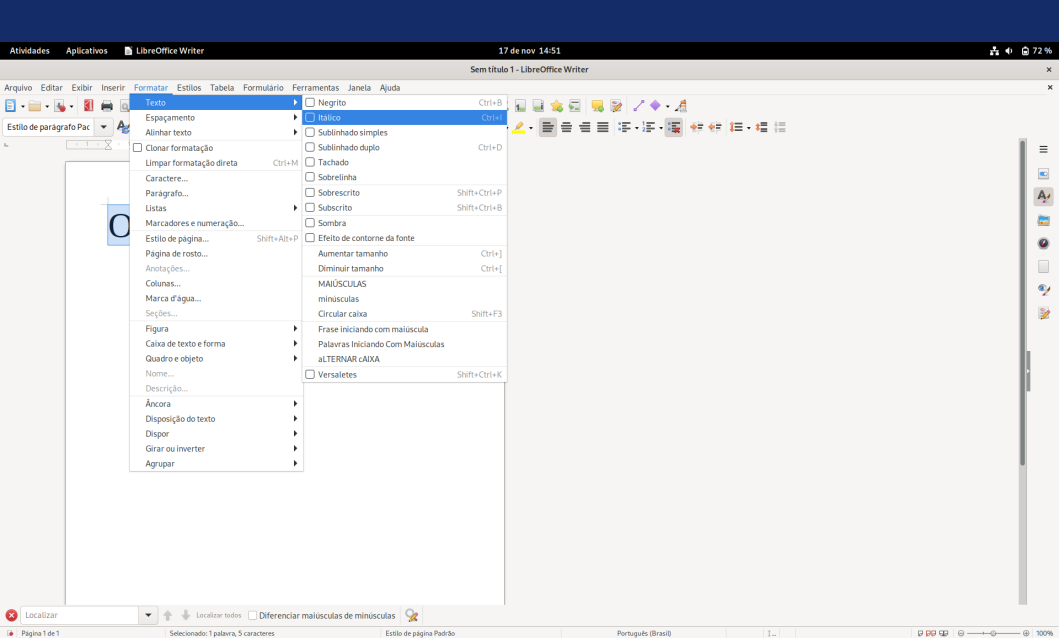
- **Funções em geral representam "ações" que esperamos que o computador realize**
- **Existem dois tipos de função em python: *funções* e *métodos***
- **Funções seguem o formato *verbo + objeto*:**
  - `print(blah)`
  - `éPrimo(x)`
  - ...
- **Métodos seguem o formato *objeto + verbo***
  - `lista.append()`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

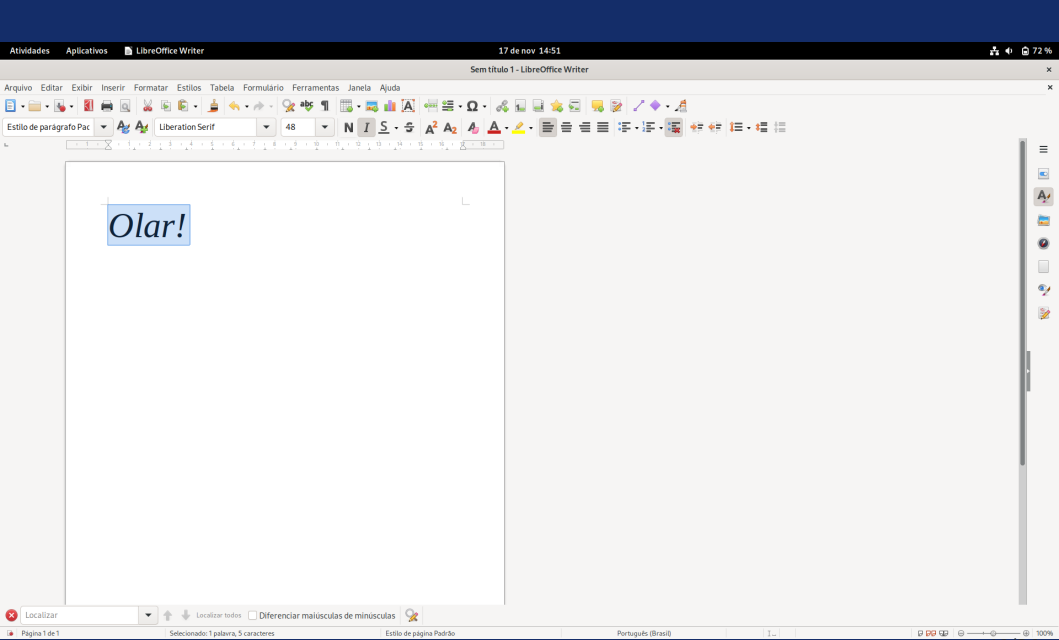

Atividades
Aplicativos
LibreOffice Writer
17 de nov 14:50
72 %
Sem título 1 - LibreOffice Writer
Arquivo
Editar
Exibir
Inserir
Formatar
Estilos
Tabela
Formulário
Ferramentas
Janela
Ajuda
Estilo de parágrafo Pad
Liberation Serif
48
Localizar
Localizar todos
Diferenciar maiúsculas de minúsculas
Página 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
Estilo de parágrafo Pad
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
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
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
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 aquele 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

- 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

Polinômio

Um polinômio de uma variável ($3x^4 + 2x^2 + x + 5$) pode ser representado pela lista de seus coeficientes: `polinômio = [5, 1, 2, 0, 3]`

Nesta representação:

- **O tamanho da lista é o grau do polinômio +1**
- **O número na posição k da lista é coeficiente do monômio de grau k**
- **O último item da lista é sempre diferente de zero**

Escreva uma função `calcula_polinômio(p, x)` em python que recebe um polinômio p como definido acima e um número real x e devolve $p(x)$.

Polinômio

Escreva uma função `calcula_polinômio(p, x)` em python que recebe um polinômio `p` como definido acima e um número real `x` e devolve $p(x)$.

Polinômio

Escreva uma função `calcula_polinômio(p, x)` em python que recebe um polinômio p como definido acima e um número real x e devolve $p(x)$.

```
def calcula_polinômio(p, x):
```

Polinômio

Escreva uma função `calcula_polinômio(p, x)` em python que recebe um polinômio p como definido acima e um número real x e devolve $p(x)$.

```
def calcula_polinômio(p, x):



    return resultado
```

Polinômio

Escreva uma função `calcula_polinômio(p, x)` em python que recebe um polinômio p como definido acima e um número real x e devolve $p(x)$.

```
def calcula_polinômio(p, x):
    resultado = 0


    return resultado
```

Polinômio

Escreva uma função `calcula_polinômio(p, x)` em python que recebe um polinômio p como definido acima e um número real x e devolve $p(x)$.

```
def calcula_polinômio(p, x):
    resultado = 0
    for i in range(len(p)):

    return resultado
```

Polinômio

Escreva uma função `calcula_polinômio(p, x)` em python que recebe um polinômio p como definido acima e um número real x e devolve $p(x)$.

```python
def calcula_polinômio(p, x):
    resultado = 0
    for i in range(len(p)):
        resultado += p[i] * x**i
    return resultado
```

Extremos

Escreva uma função que recebe uma lista de números ordenada e devolve o menor e o maior números da lista

Extremos

Escreva uma função que recebe uma lista de números ordenada e devolve o menor e o maior números da lista

```
def extremos_da_lista(l):

```

Extremos

Escreva uma função que recebe uma lista de números ordenada e devolve o menor e o maior números da lista

```
def extremos_da_lista(l):
    return l[0], l[-1]
```

Ímpares

Escreva uma função que recebe uma lista de números naturais e devolve uma lista com os números ímpares da lista recebida

Ímpares

Escreva uma função que recebe uma lista de números naturais e devolve uma lista com os números ímpares da lista recebida

```
def ímpares(l):
```

Ímpares

Escreva uma função que recebe uma lista de números naturais e devolve uma lista com os números ímpares da lista recebida

```
def ímpares(l):




    return imp
```

Ímpares

Escreva uma função que recebe uma lista de números naturais e devolve uma lista com os números ímpares da lista recebida

```
def ímpares(l):
    imp = []




    return imp
```

Ímpares

Escreva uma função que recebe uma lista de números naturais e devolve uma lista com os números ímpares da lista recebida

```
def ímpares(l):
    imp = []

        if n % 2 != 0:
            imp.append(n)
    return imp
```

Ímpares

Escreva uma função que recebe uma lista de números naturais e devolve uma lista com os números ímpares da lista recebida

```
def ímpares(l):
    imp = []
    for n in l:
        if n % 2 != 0:
            imp.append(n)
    return imp
```

Sequência de inteiros

Escreva uma função que recebe uma lista de números inteiros e devolve o pedaço da lista entre o primeiro e o último números estritamente positivos (inclusive)

Sequência de inteiros

Escreva uma função que recebe uma lista de números inteiros e devolve o pedaço da lista entre o primeiro e o último números estritamente positivos (inclusive)

```
def sequência_positivos(l):
```

Sequência de inteiros

Escreva uma função que recebe uma lista de números inteiros e devolve o pedaço da lista entre o primeiro e o último números estritamente positivos (inclusive)

```
def sequência_positivos(l):









        return l[primeiro_positivo:último_positivo +1]
```

Sequência de inteiros

Escreva uma função que recebe uma lista de números inteiros e devolve o pedaço da lista entre o primeiro e o último números estritamente positivos (inclusive)

```
def sequência_positivos(l):
    primeiro_positivo, último_positivo = -1, -1
    return l[primeiro_positivo:último_positivo +1]
```

Sequência de inteiros

Escreva uma função que recebe uma lista de números inteiros e devolve o pedaço da lista entre o primeiro e o último números estritamente positivos (inclusive)

```
def sequência_positivos(l):
    primeiro_positivo, último_positivo = -1, -1
    for i in range(len(l)):









        return l[primeiro_positivo:último_positivo +1]
```

Sequência de inteiros

Escreva uma função que recebe uma lista de números inteiros e devolve o pedaço da lista entre o primeiro e o último números estritamente positivos (inclusive)

```
def sequência_positivos(l):
    primeiro_positivo, último_positivo = -1, -1
    for i in range(len(l)):
        if l[i] > 0:









        return l[primeiro_positivo:último_positivo +1]
```

Sequência de inteiros

Escreva uma função que recebe uma lista de números inteiros e devolve o pedaço da lista entre o primeiro e o último números estritamente positivos (inclusive)

```
def sequência_positivos(l):
    primeiro_positivo, último_positivo = -1, -1
    for i in range(len(l)):
        if l[i] > 0:
            if primeiro_positivo < 0:
                primeiro_positivo = i




        return l[primeiro_positivo:último_positivo +1]
```

Sequência de inteiros

Escreva uma função que recebe uma lista de números inteiros e devolve o pedaço da lista entre o primeiro e o último números estritamente positivos (inclusive)

```
def sequência_positivos(l):
    primeiro_positivo, último_positivo = -1, -1
    for i in range(len(l)):
        if l[i] > 0:
            if primeiro_positivo < 0:
                primeiro_positivo = i
            último_positivo = i



        return l[primeiro_positivo:último_positivo +1]
```

Sequência de inteiros

Escreva uma função que recebe uma lista de números inteiros e devolve o pedaço da lista entre o primeiro e o último números estritamente positivos (inclusive)

```
def sequência_positivos(l):
    primeiro_positivo, último_positivo = -1, -1
    for i in range(len(l)):
        if l[i] > 0:
            if primeiro_positivo < 0:
                primeiro_positivo = i
            último_positivo = i
    if primeiro_positivo < 0:
        return []
    else:
        return l[primeiro_positivo:último_positivo +1]
```

Soma de um segmento

Escreva uma função que recebe uma lista de números inteiros, um índice inicial e um índice final e devolve a soma dos elementos no intervalo [início, fim)

Soma de um segmento

Escreva uma função que recebe uma lista de números inteiros, um índice inicial e um índice final e devolve a soma dos elementos no intervalo [início, fim)

```
def soma_segmento(l, i, f):
```

Soma de um segmento

Escreva uma função que recebe uma lista de números inteiros, um índice inicial e um índice final e devolve a soma dos elementos no intervalo [início, fim)

```
def soma_segmento(l, i, f):
    soma = 0
```

Soma de um segmento

Escreva uma função que recebe uma lista de números inteiros, um índice inicial e um índice final e devolve a soma dos elementos no intervalo [início, fim)

```
def soma_segmento(l, i, f):
    soma = 0




    return soma
```

Soma de um segmento

Escreva uma função que recebe uma lista de números inteiros, um índice inicial e um índice final e devolve a soma dos elementos no intervalo [início, fim)

```
def soma_segmento(l, i, f):
    soma = 0



        soma += n
    return soma
```

Soma de um segmento

Escreva uma função que recebe uma lista de números inteiros, um índice inicial e um índice final e devolve a soma dos elementos no intervalo [início, fim)

```
def soma_segmento(l, i, f):
    soma = 0
    l = l[i:f]

        soma += n
    return soma
```

Soma de um segmento

Escreva uma função que recebe uma lista de números inteiros, um índice inicial e um índice final e devolve a soma dos elementos no intervalo [início, fim)

```
def soma_segmento(l, i, f):
    soma = 0
    l = l[i:f]
    for n in l:
        soma += n
    return soma
```

Soma de um segmento

Escreva uma função que recebe uma lista de números inteiros,
um índice inicial e um índice final e devolve a soma dos elementos
no intervalo [início, fim)

```
def soma_segmento(l, i, f):
```

Soma de um segmento

Escreva uma função que recebe uma lista de números inteiros, um índice inicial e um índice final e devolve a soma dos elementos no intervalo [início, fim)

```
def soma_segmento(l, i, f):
    soma = 0
```

Soma de um segmento

Escreva uma função que recebe uma lista de números inteiros, um índice inicial e um índice final e devolve a soma dos elementos no intervalo [início, fim)

```
def soma_segmento(l, i, f):
    soma = 0



    return soma
```

Soma de um segmento

Escreva uma função que recebe uma lista de números inteiros, um índice inicial e um índice final e devolve a soma dos elementos no intervalo [início, fim)

```
def soma_segmento(l, i, f):
    soma = 0
    for i in range(i:f):
        soma += l(i)
    return soma
```

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
```

Removendo indesejáveis

Escreva uma função que recebe uma lista de inteiros e devolve outra lista, igual à primeira, mas sem os elementos menores que zero

Removendo indesejáveis

Escreva uma função que recebe uma lista de inteiros e devolve outra lista, igual à primeira, mas sem os elementos menores que zero

```
def remove_negativos(l):
```

Removendo indesejáveis

Escreva uma função que recebe uma lista de inteiros e devolve outra lista, igual à primeira, mas sem os elementos menores que zero

```
def remove_negativos(l):
    l = l[:]
    for n in l:
        if n < 0:
            l.remove(n)
    return l
```

Removendo indesejáveis

Escreva uma função que recebe uma lista de inteiros e devolve outra lista, igual à primeira, mas sem os elementos menores que zero

```
def remove_negativos(l):
    l = l[:]
    for n in l:
        if n < 0:
            l.remove(n)
    return l
def main():
    l = [1, 2, -1, -2, 3, -4]
    print(remove_negativos(l))
```

Removendo indesejáveis

Escreva uma função que recebe uma lista de inteiros e devolve outra lista, igual à primeira, mas sem os elementos menores que zero

```
def remove_negativos(l):
    l = l[:]
    for n in l:
        if n < 0:
            l.remove(n)
    return l
def main():
    l = [1, 2, -1, -2, 3, -4]
    print(remove_negativos(l))
```

[1, 2, -2, 3]

Removendo indesejáveis

Escreva uma função que recebe uma lista de inteiros e devolve outra lista, igual à primeira, mas sem os elementos menores que zero

```
def remove_negativos(l):
    l = l[:]
    for n in l:
        if n < 0:
            l.remove(n)
    return l
def main():
    l = [1, 2, -1, -2, 3, -4]
    print(remove_negativos(l))
```

[1, 2, -2, 3]

Removendo indesejáveis

Escreva uma função que recebe uma lista de inteiros e devolve outra lista, igual à primeira, mas sem os elementos menores que zero

Removendo indesejáveis

Escreva uma função que recebe uma lista de inteiros e devolve outra lista, igual à primeira, mas sem os elementos menores que zero

```
def remove_negativos(l):
```

Removendo indesejáveis

Escreva uma função que recebe uma lista de inteiros e devolve outra lista, igual à primeira, mas sem os elementos menores que zero

```
def remove_negativos(l):
    l = l[:]
```

Removendo indesejáveis

Escreva uma função que recebe uma lista de inteiros e devolve outra lista, igual à primeira, mas sem os elementos menores que zero

```
def remove_negativos(l):
    l = l[:]









    return l
```

Removendo indesejáveis

Escreva uma função que recebe uma lista de inteiros e devolve outra lista, igual à primeira, mas sem os elementos menores que zero

```
def remove_negativos(l):
    l = l[:]

    while i < len(l):




    return l
```

Removendo indesejáveis

Escreva uma função que recebe uma lista de inteiros e devolve outra lista, igual à primeira, mas sem os elementos menores que zero

```
def remove_negativos(l):
    l = l[:]
    i = 0
    while i < len(l):




        i += 1
    return l
```

Removendo indesejáveis

Escreva uma função que recebe uma lista de inteiros e devolve outra lista, igual à primeira, mas sem os elementos menores que zero

```
def remove_negativos(l):
    l = l[:]
    i = 0
    while i < len(l):
        if l[i] < 0:
            del l[i]

        i += 1
    return l
```

Removendo indesejáveis

Escreva uma função que recebe uma lista de inteiros e devolve outra lista, igual à primeira, mas sem os elementos menores que zero

```
def remove_negativos(l):
    l = l[:]
    i = 0
    while i < len(l):
        if l[i] < 0:
            del l[i]
        else:
            i += 1
    return l
```

Removendo indesejáveis II

Escreva uma função que recebe uma lista de inteiros em ordem crescente e um número inteiro e remove dessa lista os elementos maiores que o número dado

Removendo indesejáveis II

Escreva uma função que recebe uma lista de inteiros em ordem crescente e um número inteiro e remove dessa lista os elementos maiores que o número dado

```
def remove_grandes(l, n):
```

Removendo indesejáveis II

Escreva uma função que recebe uma lista de inteiros em ordem crescente e um número inteiro e remove dessa lista os elementos maiores que o número dado

```
def remove_grandes(l, n):


    while i < len(l)
```

Removendo indesejáveis II

Escreva uma função que recebe uma lista de inteiros em ordem crescente e um número inteiro e remove dessa lista os elementos maiores que o número dado

```
def remove_grandes(l, n):

    i = 0
    while i < len(l)
```

Removendo indesejáveis II

Escreva uma função que recebe uma lista de inteiros em ordem crescente e um número inteiro e remove dessa lista os elementos maiores que o número dado

```
def remove_grandes(l, n):

    i = 0
    while i < len(l)


        i += 1
```

Removendo indesejáveis II

Escreva uma função que recebe uma lista de inteiros em ordem crescente e um número inteiro e remove dessa lista os elementos maiores que o número dado

```
def remove_grandes(l, n):

    i = 0
    while i < len(l)


        i += 1

        del l[limite:]
```

Removendo indesejáveis II

Escreva uma função que recebe uma lista de inteiros em ordem crescente e um número inteiro e remove dessa lista os elementos maiores que o número dado

```
def remove_grandes(l, n):

    i = 0
    while i < len(l)
        if l[i] > n:
            limite = i
        i += 1

        del l[limite:]
```

Removendo indesejáveis II

Escreva uma função que recebe uma lista de inteiros em ordem crescente e um número inteiro e remove dessa lista os elementos maiores que o número dado

```
def remove_grandes(l, n):
    limite = -1
    i = 0
    while i < len(l)
        if l[i] > n:
            limite = i
        i += 1

        del l[limite:]
```

Removendo indesejáveis II

Escreva uma função que recebe uma lista de inteiros em ordem crescente e um número inteiro e remove dessa lista os elementos maiores que o número dado

```
def remove_grandes(l, n):
    limite = -1
    i = 0
    while i < len(l) and limite < 0:
        if l[i] > n:
            limite = i
        i += 1

        del l[limite:]
```

Removendo indesejáveis II

Escreva uma função que recebe uma lista de inteiros em ordem crescente e um número inteiro e remove dessa lista os elementos maiores que o número dado

```
def remove_grandes(l, n):
    limite = -1
    i = 0
    while i < len(l) and limite < 0:
        if l[i] > n:
            limite = i
        i += 1
    if limite >= 0:
        del l[limite:]
```

Removendo indesejáveis II

Escreva uma função que recebe uma lista de inteiros em ordem crescente e um número inteiro e remove dessa lista os elementos maiores que o número dado

```
def remove_grandes(l, n):
```

Removendo indesejáveis II

Escreva uma função que recebe uma lista de inteiros em ordem crescente e um número inteiro e remove dessa lista os elementos maiores que o número dado

```
def remove_grandes(l, n):
    limite = -1
```

Removendo indesejáveis II

Escreva uma função que recebe uma lista de inteiros em ordem crescente e um número inteiro e remove dessa lista os elementos maiores que o número dado

```
def remove_grandes(l, n):
    limite = -1




    if limite >= 0:
        del l[limite:]
```

Removendo indesejáveis II

Escreva uma função que recebe uma lista de inteiros em ordem crescente e um número inteiro e remove dessa lista os elementos maiores que o número dado

```
def remove_grandes(l, n):
    limite = -1
    for i in range(len(n)):



    if limite >= 0:
        del l[limite:]
```

Removendo indesejáveis II

Escreva uma função que recebe uma lista de inteiros em ordem crescente e um número inteiro e remove dessa lista os elementos maiores que o número dado

```
def remove_grandes(l, n):
    limite = -1
    for i in range(len(n)):
        if l[i] > n              :
            limite = i
    if limite >= 0:
        del l[limite:]
```

Removendo indesejáveis II

Escreva uma função que recebe uma lista de inteiros em ordem crescente e um número inteiro e remove dessa lista os elementos maiores que o número dado

```
def remove_grandes(l, n):
    limite = -1
    for i in range(len(n)):
        if l[i] > n and limit < 0:
            limite = i
    if limite >= 0:
        del l[limite:]
```

Removendo indesejáveis II

Escreva uma função que recebe uma lista e um valor e insere esse valor no final da lista, a menos que o valor já faça parte da lista

Removendo indesejáveis II

Escreva uma função que recebe uma lista e um valor e insere esse valor no final da lista, a menos que o valor já faça parte da lista

```
def insere_se_novo(l, n):
```

Removendo indesejáveis II

Escreva uma função que recebe uma lista e um valor e insere esse valor no final da lista, a menos que o valor já faça parte da lista

```
def insere_se_novo(l, n):

        l.append(n)
```

Removendo indesejáveis II

Escreva uma função que recebe uma lista e um valor e insere esse valor no final da lista, a menos que o valor já faça parte da lista

```
def insere_se_novo(l, n):
    if not n in l:
        l.append(n)
```

Coleções

Escreva uma função que recebe duas listas de números ordenadas e devolve uma lista com a união das duas listas, também ordenada

Coleções

Escreva uma função que recebe duas listas de números ordenadas e devolve uma lista com a união das duas listas, também ordenada

```
def mescla_listas(l1, l2):
```

Coleções

Escreva uma função que recebe duas listas de números ordenadas e devolve uma lista com a união das duas listas, também ordenada

```
def mescla_listas(l1, l2):
    lista = []









    return lista
```

Coleções

Escreva uma função que recebe duas listas de números ordenadas e devolve uma lista com a união das duas listas, também ordenada

```
def mescla_listas(l1, l2):
    lista = []
    i, j = 0, 0









    return lista
```

Coleções

Escreva uma função que recebe duas listas de números ordenadas e devolve uma lista com a união das duas listas, também ordenada

```
def mescla_listas(l1, l2):
    lista = []
    i, j = 0, 0
    while i < len(l1) and j < len(l2):









    return lista
```

Coleções

Escreva uma função que recebe duas listas de números ordenadas e devolve uma lista com a união das duas listas, também ordenada

```
def mescla_listas(l1, l2):
    lista = []
    i, j = 0, 0
    while i < len(l1) and j < len(l2):
        if l1[i] < l2[j]:
            lista.append(l1[i])
            i += 1









    return lista
```

Coleções

Escreva uma função que recebe duas listas de números ordenadas e devolve uma lista com a união das duas listas, também ordenada

```
def mescla_listas(l1, l2):
    lista = []
    i, j = 0, 0
    while i < len(l1) and j < len(l2):
        if l1[i] < l2[j]:
            lista.append(l1[i])
            i += 1
        else:









    return lista
```

Coleções

Escreva uma função que recebe duas listas de números ordenadas e devolve uma lista com a união das duas listas, também ordenada

```
def mescla_listas(l1, l2):
    lista = []
    i, j = 0, 0
    while i < len(l1) and j < len(l2):
        if l1[i] < l2[j]:
            lista.append(l1[i])
            i += 1
        else:
            lista.append(l2[j])
            j += 1









    return lista
```

Coleções

Escreva uma função que recebe duas listas de números ordenadas e devolve uma lista com a união das duas listas, também ordenada

```
def mescla_listas(l1, l2):
    lista = []
    i, j = 0, 0
    while i < len(l1) and j < len(l2):
        if l1[i] < l2[j]:
            lista.append(l1[i])
            i += 1
        else:
            lista.append(l2[j])
            j += 1
    while i < len(l1):
        lista.append(l1[i])
        i += 1



    return lista
```

Coleções

Escreva uma função que recebe duas listas de números ordenadas e devolve uma lista com a união das duas listas, também ordenada

```
def mescla_listas(l1, l2):
    lista = []
    i, j = 0, 0
    while i < len(l1) and j < len(l2):
        if l1[i] < l2[j]:
            lista.append(l1[i])
            i += 1
        else:
            lista.append(l2[j])
            j += 1
    while i < len(l1):
        lista.append(l1[i])
        i += 1
    while j < len(l2):
        lista.append(l2[j])
        j += 1
    return lista
```

Coleções

Modifique a função anterior para eliminar números repetidos

Coleções

Modifique a função anterior para eliminar números repetidos

```
def mescla_listas(l1, l2):
    lista = []
    i, j = 0, 0
    while i < len(l1) and j < len(l2):
        if l1[i] < l2[j]:

                lista.append(l1[i])
            i += 1
        else:

                lista.append(l2[j])
            j += 1
    while i < len(l1):

            lista.append(l1[i])
        i += 1
    while j < len(l2):

            lista.append(l2[j])
        j += 1
    return lista
```

Coleções

Modifique a função anterior para eliminar números repetidos

```
def mescla_listas(l1, l2):
    lista = []
    i, j = 0, 0
    while i < len(l1) and j < len(l2):
        if l1[i] < l2[j]:
            if len(lista) == 0 or lista[-1] < l1[i]:
                lista.append(l1[i])
            i += 1
        else:

                lista.append(l2[j])
            j += 1
    while i < len(l1):

            lista.append(l1[i])
        i += 1
    while j < len(l2):

            lista.append(l2[j])
        j += 1
    return lista
```

Coleções

Modifique a função anterior para eliminar números repetidos

```
def mescla_listas(l1, l2):
    lista = []
    i, j = 0, 0
    while i < len(l1) and j < len(l2):
        if l1[i] < l2[j]:
            if len(lista) == 0 or lista[-1] < l1[i]:
                lista.append(l1[i])
            i += 1
        else:
            if len(lista) == 0 or lista[-1] < l2[j]:
                lista.append(l2[j])
            j += 1
    while i < len(l1):
        if len(lista) == 0 or lista[-1] < l1[i]:
            lista.append(l1[i])
        i += 1
    while j < len(l2):
        if len(lista) == 0 or lista[-1] < l2[j]:
            lista.append(l2[j])
        j += 1
    return lista
```