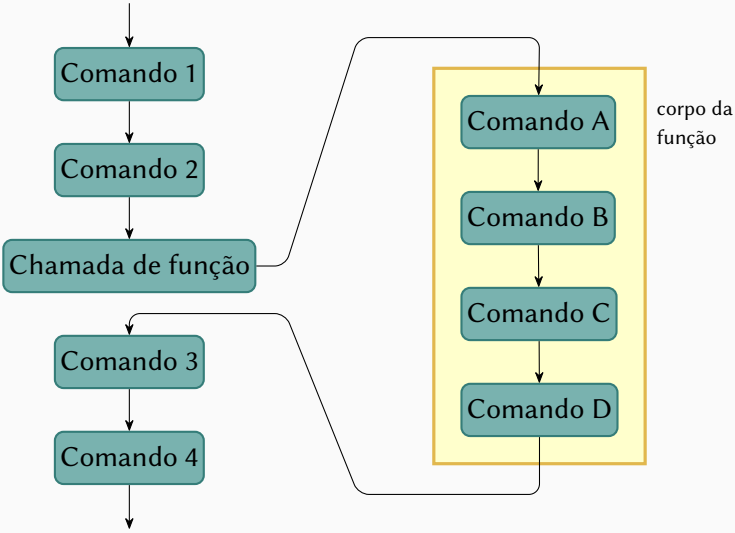
MAC 110 — Introdução à Ciência da Computação

Aula 18

Nelson Lago

BMAC – 2024

Funções são “desvios”



Funções – escopo

- Funções são **nomes** dados a trechos de código que representam uma **ideia** bem definida e auto-contida

Funções – escopo

- Funções são **nomes** dados a trechos de código que representam uma **ideia** bem definida e auto-contida
- Uma das vantagens é que elas podem ser usadas em contextos diferentes para realizar a computação correspondente àquela ideia

Funções – escopo

- Funções são **nomes** dados a trechos de código que representam uma **ideia** bem definida e auto-contida
- Uma das vantagens é que elas podem ser usadas em contextos diferentes para realizar a computação correspondente àquela ideia
- Portanto...

Funções – escopo

- Funções são **nomes** dados a trechos de código que representam uma **ideia** bem definida e auto-contida
- Uma das vantagens é que elas podem ser usadas em contextos diferentes para realizar a computação correspondente àquela ideia
- Portanto...
- Elas precisam ser capazes de funcionar de maneira independente do contexto

Cada um com seu cada um e ninguém se mete no cada um dos outros

Funções – escopo

```
def fatorial(n):
    fat = 1
    while n >= 2:
        fat *= n
        n -= 1
    return fat
fatorial(4)
print(fat)
```

Funções – escopo

```
def fatorial(n):
    fat = 1
    while n >= 2:
        fat *= n
        n -= 1
    return fat
print(fatorial(4))
```

Funções – escopo

```
n = 4
def fatorial(n):
    fat = 1
    while n >= 2:
        fat *= n
        n -= 1
    return fat
print(fatorial())
```

Funções – escopo

```
n = 5
def fatorial(n):
    fat = 1
    while n >= 2:
        fat *= n
        n -= 1
    return fat
print(fatorial(4))
print(n)
```

Funções – escopo

```
def fatorial_enviesado(n):
    fat = 1
    viés = 5
    while n >= 2:
        fat *= n
        n -= 1
    return fat + viés
print(fatorial_enviesado(4))
print(viés)
```

Funções – escopo

```
viés = 3
def fatorial_enviesado(n):
    fat = 1
    viés = 5
    while n >= 2:
        fat *= n
        n -= 1
    return fat + viés
print(fatorial_enviesado(4))
print(viés)
```

Funções – escopo

```
viés = 3
def fatorial_enviesado(n):
    fat = 1

    while n >= 2:
        fat *= n
        n -= 1
    return fat + viés
print(fatorial_enviesado(4))
```

Funções – escopo

```
viés = 3
def fatorial_enviesado(n):
    fat = 1

    while n >= 2:
        fat *= n
        n -= 1
    return fat + viés
print(fatorial_enviesado(4))
```

AAAAHHHH!!!!

Funções – escopo

```
viés = 3
def fatorial_enviesado(n):
    fat = 1
    print("Viés anterior:", viés)
    viés = 5
    while n >= 2:
        fat *= n
        n -= 1
    return fat + viés
print(fatorial_enviesado(4))
print(viés)
```

Funções – escopo

```
viés = 3
def fatorial_enviesado(n):
    fat = 1
    print("Viés anterior:", viés)
    viés = 5
    while n >= 2:
        fat *= n
        n -= 1
    return fat + viés
print(fatorial_enviesado(4))
print(viés)
```

AAAAHHHH!!!!

Funções – escopo

```
viés = 3
def fatorial_enviesado(n):
    global viés
    fat = 1
    print("Viés anterior:", viés)
    viés = 5
    while n >= 2:
        fat *= n
        n -= 1
    return fat + viés
print(fatorial_enviesado(4))
print(viés)
```

Cada um com seu cada um e ninguém se mete no cada um dos outros

Cada um com seu cada um e ninguém se mete no cada um dos outros

(só às vezes 🥲)

Funções – escopo

- **Às vezes significa:**

Funções – escopo

- **Às vezes significa:**
  - Quando usamos a palavra-chave **`global`**

Funções – escopo

- **Às vezes significa:**
  - Quando usamos a palavra-chave `global`
  - Quando a variável não existe no contexto da função, mas existe no contexto global **E** é usada na função apenas para leitura

Funções – escopo

- **Às vezes significa:**
  - Quando usamos a palavra-chave `global`
  - Quando a variável não existe no contexto da função, mas existe no contexto global **E** é usada na função apenas para leitura
  - Quando modificamos um dado mutável recebido como parâmetro

Funções – `main()`

```
print(fatorial(4))
def fatorial(n):
    fat = 1
    while n >= 2:
        fat *= n
        n -= 1
    return fat
```

Funções – `main()`

```
def main():
    x = int(input("Digite um inteiro positivo: "))
    print(fatorial(x))

def fatorial(n):
    fat = 1
    while n >= 2:
        fat *= n
        n -= 1
    return fat

main()
```

Funções – `main()`

```
def fatorial(n):
    fat = 1
    while n >= 2:
        fat *= n
        n -= 1
    return fat

def main():
    x = int(input("Digite um inteiro positivo: "))
    print(fatorial(x))

main()
```

Embora não seja obrigatório, em geral é uma boa ideia usar uma função `main()`

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")
```

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")
```

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

Tipos de laços diferentes “combinam melhor” com sintaxes diferentes

Tipos de repetição

- Quando a quantidade de repetições é desconhecida, `while` é uma boa escolha:

Tipos de repetição

- **Quando a quantidade de repetições é desconhecida, `while` é uma boa escolha:**
  - `while not achei:`

Tipos de repetição

- **Quando a quantidade de repetições é desconhecida, `while` é uma boa escolha:**
  - `while not achei:`
  - `while encontrados < 10:`

Tipos de repetição

- **Quando a quantidade de repetições é desconhecida, `while` é uma boa escolha:**
  - `while not achei:`
  - `while encontrados < 10:`
  - `while usuárioQuerJogar:`

Tipos de repetição

- **Quando a quantidade de repetições é desconhecida, `while` é uma boa escolha:**
  - `while not achei:`
  - `while encontrados < 10:`
  - `while usuárioQuerJogar:`
- **Quando queremos manipular os elementos de uma coleção, `for`...`in` é uma boa escolha:**

Tipos de repetição

- **Quando a quantidade de repetições é desconhecida, `while` é uma boa escolha:**
  - `while not achei:`
  - `while encontrados < 10:`
  - `while usuárioQuerJogar:`
- **Quando queremos manipular os elementos de uma coleção, `for`...`in` é uma boa escolha:**
  - `for p in primos:`

Tipos de repetição

- **Quando a quantidade de repetições é desconhecida, `while` é uma boa escolha:**
  - `while not achei:`
  - `while encontrados < 10:`
  - `while usuárioQuerJogar:`
- **Quando queremos manipular os elementos de uma coleção, `for`...`in` é uma boa escolha:**
  - `for p in primos:`
  - `for canção in canções:`

Tipos de repetição

- **Quando a quantidade de repetições é desconhecida, `while` é uma boa escolha:**
  - `while not achei:`
  - `while encontrados < 10:`
  - `while usuárioQuerJogar:`
- **Quando queremos manipular os elementos de uma coleção, `for`...`in` é uma boa escolha:**
  - `for p in primos:`
  - `for canção in canções:`
- **Quando queremos manipular uma lista pré-definida de números *ou* os *índices* dos elementos de uma coleção, `for`...`in` `range()` é uma boa escolha:**

Tipos de repetição

- **Quando a quantidade de repetições é desconhecida, `while` é uma boa escolha:**
  - `while not achei:`
  - `while encontrados < 10:`
  - `while usuárioQuerJogar:`
- **Quando queremos manipular os elementos de uma coleção, `for`...`in` é uma boa escolha:**
  - `for p in primos:`
  - `for canção in canções:`
- **Quando queremos manipular uma lista pré-definida de números *ou* os *índices* dos elementos de uma coleção, `for`...`in` `range()` é uma boa escolha:**
  - `for n in range(10):`

Tipos de repetição

- **Quando a quantidade de repetições é desconhecida, `while` é uma boa escolha:**
  - `while not achei:`
  - `while encontrados < 10:`
  - `while usuárioQuerJogar:`
- **Quando queremos manipular os elementos de uma coleção, `for`...`in` é uma boa escolha:**
  - `for p in primos:`
  - `for canção in canções:`
- **Quando queremos manipular uma lista pré-definida de números *ou* os *índices* dos elementos de uma coleção, `for`...`in` `range()` é uma boa escolha:**
  - `for n in range(10):`
  - `for n in range(len(minha_lista)):`

Tipos de repetição

- **Quando a quantidade de repetições é desconhecida, `while` é uma boa escolha:**
  - `while not achei:`
  - `while encontrados < 10:`
  - `while usuárioQuerJogar:`
- **Quando queremos manipular os elementos de uma coleção, `for`...`in` é uma boa escolha:**
  - `for p in primos:`
  - `for canção in canções:`
- **Quando queremos manipular uma lista pré-definida de números *ou* os *índices* dos elementos de uma coleção, `for`...`in` `range()` é uma boa escolha:**
  - `for n in range(10):`
  - `for n in range(len(minha_lista)):`
  - `for i in range(início,final):`

Uma “coisa” pode ter mais de um nome

Uma “coisa” pode ter mais de um nome

```
L1 = [1, 2, 3]
```

Uma “coisa” pode ter mais de um nome

```
L1 = [1, 2, 3]
```

L1

Uma “coisa” pode ter mais de um nome

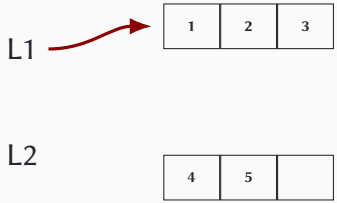
```
L1 = [1, 2, 3]
```

Uma “coisa” pode ter mais de um nome

```
L1 = [1, 2, 3]
L2 = [4, 5, L1]
```

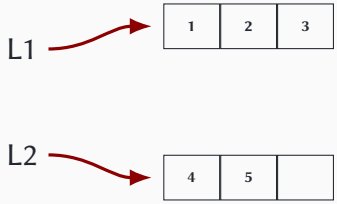
Uma “coisa” pode ter mais de um nome

```
L1 = [1, 2, 3]
L2 = [4, 5, L1]
```



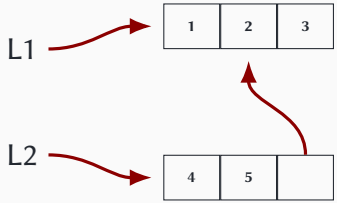
Uma “coisa” pode ter mais de um nome

```
L1 = [1, 2, 3]
L2 = [4, 5, L1]
```



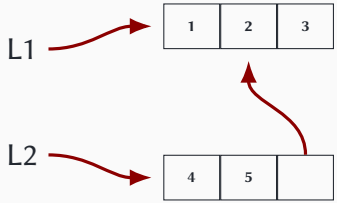
Uma “coisa” pode ter mais de um nome

```
L1 = [1, 2, 3]
L2 = [4, 5, L1]
```



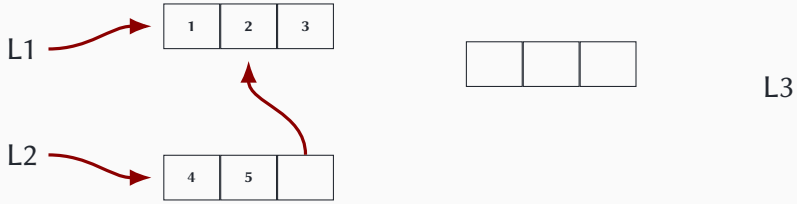
Uma “coisa” pode ter mais de um nome

```
L1 = [1, 2, 3]
L2 = [4, 5, L1]
L3 = [L1] * 3
```



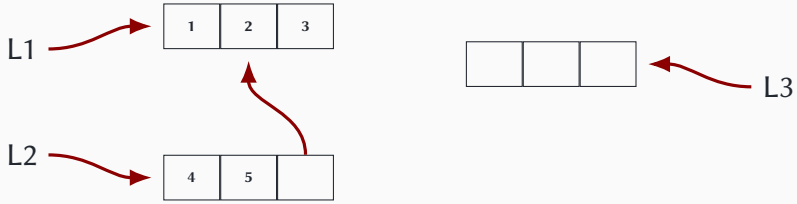
Uma “coisa” pode ter mais de um nome

```
L1 = [1, 2, 3]
L2 = [4, 5, L1]
L3 = [L1] * 3
```



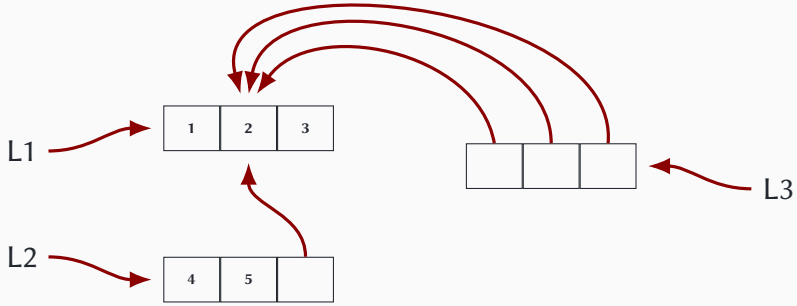
Uma “coisa” pode ter mais de um nome

```
L1 = [1, 2, 3]
L2 = [4, 5, L1]
L3 = [L1] * 3
```



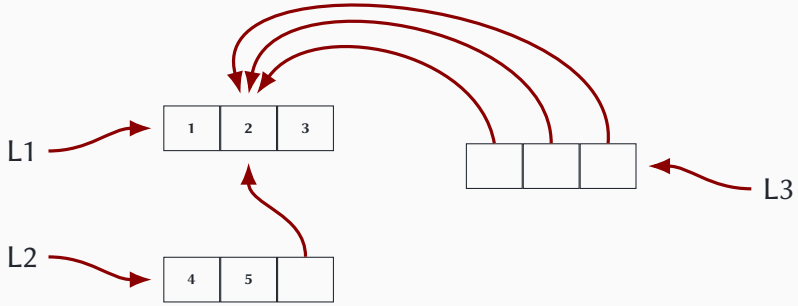
Uma “coisa” pode ter mais de um nome

```
L1 = [1, 2, 3]
L2 = [4, 5, L1]
L3 = [L1] * 3
```



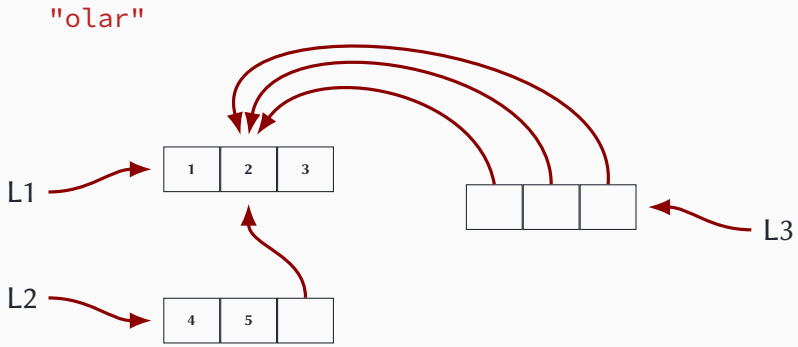
Uma “coisa” pode ter mais de um nome

```
L1 = [1, 2, 3]
L2 = [4, 5, L1]
L3 = [L1] * 3
L1 = "olar"
```



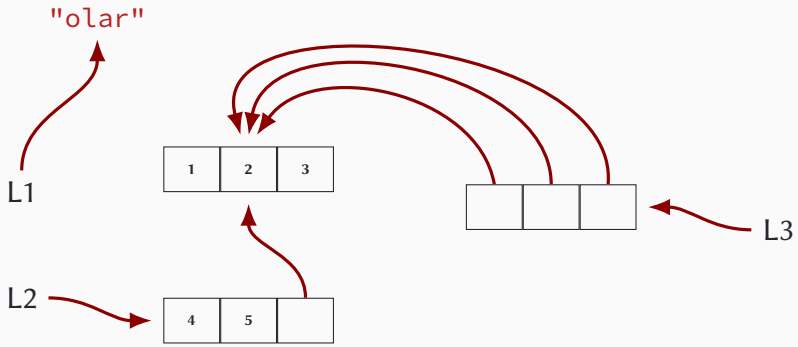
Uma “coisa” pode ter mais de um nome

```
L1 = [1, 2, 3]
L2 = [4, 5, L1]
L3 = [L1] * 3
L1 = "olar"
```



Uma “coisa” pode ter mais de um nome

```
L1 = [1, 2, 3]
L2 = [4, 5, L1]
L3 = [L1] * 3
L1 = "olar"
```



Igualdade e identidade

Igualdade e identidade

```
L1 = ["olá", "alô", "oi"]
```

Igualdade e identidade

```
L1 = ["olá", "alô", "oi"]
L2 = ["olá", "alô", "oi"]
```

Igualdade e identidade

```
L1 = ["olá", "alô", "oi"]
L2 = ["olá", "alô", "oi"]
print(L1 == L2)
```

Igualdade e identidade

```
L1 = ["olá", "alô", "oi"]
L2 = ["olá", "alô", "oi"]
print(L1 == L2)
```

True

Igualdade e identidade

```
L1 = ["olá", "alô", "oi"]
L2 = ["olá", "alô", "oi"]
print(L1 == L2)
print(L1 is L2)
```

True

Igualdade e identidade

```
L1 = ["olá", "alô", "oi"]
L2 = ["olá", "alô", "oi"]
print(L1 == L2)
print(L1 is L2)
```

True

False

Igualdade e identidade

```
L1 = ["olá", "alô", "oi"]
L2 = ["olá", "alô", "oi"]
print(L1 == L2)
print(L1 is L2)
L2 = L1
```

True

False

Igualdade e identidade

```
L1 = ["olá", "alô", "oi"]
L2 = ["olá", "alô", "oi"]
print(L1 == L2)
print(L1 is L2)
L2 = L1
print(L1 == L2)
```

True

False

Igualdade e identidade

```
L1 = ["olá", "alô", "oi"]
L2 = ["olá", "alô", "oi"]
print(L1 == L2)
print(L1 is L2)
L2 = L1
print(L1 == L2)
```

True

False

True

Igualdade e identidade

```
L1 = ["olá", "alô", "oi"]
L2 = ["olá", "alô", "oi"]
print(L1 == L2)
print(L1 is L2)
L2 = L1
print(L1 == L2)
print(L1 is L2)
```

True

False

True

Igualdade e identidade

```
L1 = ["olá", "alô", "oi"]
L2 = ["olá", "alô", "oi"]
print(L1 == L2)
print(L1 is L2)
L2 = L1
print(L1 == L2)
print(L1 is L2)
```

True

False

True

True

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

- **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**
  - Isso também é um efeito colateral!

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()` (efeitos colaterais)

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()` (efeitos colaterais)
- **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()` (efeitos colaterais)
- **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()` (efeitos colaterais)
- **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()` (efeitos colaterais)
- **Ou podem interagir com as outras partes do programa**
  - *recebendo* e *devolvendo* dados — parâmetros e **`return`**

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

  - Lendo ou modificando uma variável **`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()` (efeitos colaterais)
- **Ou podem interagir com as outras partes do programa**
  - *recebendo* e *devolvendo* dados — parâmetros e **`return`**

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

  - Lendo ou modificando uma variável **`global`** (efeito colateral)

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()` (efeitos colaterais)
- **Ou podem interagir com as outras partes do programa**
  - *recebendo* e *devolvendo* dados — parâmetros e **`return`**

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

  - Lendo ou modificando uma variável **`global`** (efeito colateral)
  - **Modificando o conteúdo de uma variável (mutável) recebida como parâmetro**

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()` (efeitos colaterais)
- **Ou podem interagir com as outras partes do programa**
  - *recebendo* e *devolvendo* dados — parâmetros e **`return`**

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

  - Lendo ou modificando uma variável **`global`** (efeito colateral)
  - **Modificando o conteúdo de uma variável (mutável) recebida como parâmetro** (efeito colateral)

Efeitos colaterais

- **Isso tudo é muito chato!**

Efeitos colaterais

- **Isso tudo é muito chato!**
- **Não dá para fazer uma linguagem totalmente sem efeitos colaterais?**

Efeitos colaterais

- **Isso tudo é muito chato!**
- **Não dá para fazer uma linguagem totalmente sem efeitos colaterais?**
- **Não** 😢

Efeitos colaterais

- Isso tudo é muito chato!
- Não dá para fazer uma linguagem totalmente sem efeitos colaterais?
- Não 😢
- As linguagens funcionais têm muito menos casos em que há efeitos colaterais

Efeitos colaterais

- **Isso tudo é muito chato!**
- **Não dá para fazer uma linguagem totalmente sem efeitos colaterais?**
- **Não** 😢
- **As linguagens funcionais têm muito menos casos em que há efeitos colaterais**
  - Isso tem diversas vantagens mas, como tudo, tem desvantagens também

Efeitos colaterais

- **Isso tudo é muito chato!**
- **Não dá para fazer uma linguagem totalmente sem efeitos colaterais?**
- **Não** 😢
- **As linguagens funcionais têm muito menos casos em que há efeitos colaterais**
  - Isso tem diversas vantagens mas, como tudo, tem desvantagens também
- **As linguagens orientadas a objetos, entre outras coisas, facilitam a tarefa de cuidar dos efeitos colaterais**

Efeitos colaterais

- **Isso tudo é muito chato!**
- **Não dá para fazer uma linguagem totalmente sem efeitos colaterais?**
- **Não** 😢
- **As linguagens funcionais têm muito menos casos em que há efeitos colaterais**
  - Isso tem diversas vantagens mas, como tudo, tem desvantagens também
- **As linguagens orientadas a objetos, entre outras coisas, facilitam a tarefa de cuidar dos efeitos colaterais**
  - A maior parte dos efeitos colaterais fica "contida"

Comandos básicos com listas

- `lista = [a, b, c]`
- `len(lista)`
- `lista.append(blah)`
- `lista[X]`
- `for item in lista:`
- `if item in lista:`
- `penúltimo = lista[-2]`
  - (de trás para a frente não começa do zero!)

- `pedaço = lista[2:5]`
  - mesmo formato de `range()`: `lista[início:final:passo]`
- `del l[3]`, `del l[2:5]`
- `lista.pop()`, `lista.pop(4)`
- `lista_com_tudo = lista1 + lista2`
- `lista.extend(outralista)`
- `váriosAs = ['A'] * 5` ⟶ `['A', 'A', 'A', 'A', 'A']`

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

Strings vs Listas

Strings vs listas

- Strings são bastante similares a listas

Strings vs listas

- **Strings são bastante similares a listas**
  - `len()`, `string[]`, **`for`** etc. funcionam como esperado

Strings vs listas

- **Strings são bastante similares a listas**
  - `len()`, `string[]`, **`for`** etc. funcionam como esperado
- **MAS!**

Strings vs listas

- **Strings são bastante similares a listas**
  - `len()`, `string[]`, **`for`** etc. funcionam como esperado
- **MAS!**
- **Strings são imutáveis**

Strings vs listas

- **Strings são bastante similares a listas**
  - `len()`, `string[]`, **`for`** etc. funcionam como esperado
- **MAS!**
- **Strings são imutáveis**
  - `append()`, `string[x] = "a"` etc. não existem ou não funcionam como esperado

Strings vs listas

- Strings têm vários métodos úteis

Strings vs listas

- **Strings têm vários métodos úteis**
  - `.format()`

Strings vs listas

- **Strings têm vários métodos úteis**
  - `.format()`
  - `.strip()`

Strings vs listas

- **Strings têm vários métodos úteis**
  - `.format()`
  - `.strip()`
  - `.upper()` / `.lower()` / `.casefold()`

Strings vs listas

- **Strings têm vários métodos úteis**
  - `.format()`
  - `.strip()`
  - `.upper()` / `.lower()` / `.casefold()`
  - `.find()` / `.replace()`

Strings vs listas

- **Strings têm vários métodos úteis**
  - `.format()`
  - `.strip()`
  - `.upper()` / `.lower()` / `.casefold()`
  - `.find()` / `.replace()`
  - `.split()` / `.join()`

Strings vs listas

- **Strings têm vários métodos úteis**
  - `.format()`
  - `.strip()`
  - `.upper()` / `.lower()` / `.casefold()`
  - `.find()` / `.replace()`
  - `.split()` / `.join()`

https://docs.python.org/3/library/stdtypes.html#string-methods

Strings vs listas

- **Strings têm vários métodos úteis**
  - `.format()`
  - `.strip()`
  - `.upper()` / `.lower()` / `.casefold()`
  - `.find()` / `.replace()`
  - `.split()` / `.join()`

https://docs.python.org/3/library/stdtypes.html#string-methods

Como strings são imutáveis, todos esses devolvem uma nova string

(exceto `.split()`, que devolve uma lista)

Strings e caracteres

- Uma string é uma sequência de caracteres e, por isso, “se assemelha” a uma lista

Strings e caracteres

- Uma string é uma sequência de caracteres e, por isso, "se assemelha" a uma lista
- Todos os caracteres são representados internamente por um número

Strings e caracteres

- **Uma string é uma sequência de caracteres e, por isso, "se assemelha" a uma lista**
- **Todos os caracteres são representados internamente por um número**
  - `ord("a")` → 97

Strings e caracteres

- **Uma string é uma sequência de caracteres e, por isso, "se assemelha" a uma lista**
- **Todos os caracteres são representados internamente por um número**

- `ord("a")` → 97
- `chr(97)` → "a"

Strings e caracteres

- **Uma string é uma sequência de caracteres e, por isso, "se assemelha" a uma lista**
- **Todos os caracteres são representados internamente por um número**

  - `ord("a")` → 97
  - `ord("β")` → 946
  - `chr(97)` → "a"

Strings e caracteres

- **Uma string é uma sequência de caracteres e, por isso, "se assemelha" a uma lista**
- **Todos os caracteres são representados internamente por um número**

- `ord("a")` → 97
- `ord("β")` → 946
- `chr(97)` → "a"
- `chr(946)` → "β"

Strings e caracteres

- **Uma string é uma sequência de caracteres e, por isso, "se assemelha" a uma lista**
- **Todos os caracteres são representados internamente por um número**

- `ord("a")` → 97
- `ord("β")` → 946
- `ord("3")` → 51

- `chr(97)` → "a"
- `chr(946)` → "β"

Strings e caracteres

- **Uma string é uma sequência de caracteres e, por isso, "se assemelha" a uma lista**
- **Todos os caracteres são representados internamente por um número**

- `ord("a")` → 97
- `ord("β")` → 946
- `ord("3")` → 51

- `chr(97)` → "a"
- `chr(946)` → "β"
- `chr(51)` → "3"

Strings e caracteres

- **Uma string é uma sequência de caracteres e, por isso, "se assemelha" a uma lista**
- **Todos os caracteres são representados internamente por um número**

- `ord("a")` → 97 (0x61)
- `ord("β")` → 946 (0x3b2)
- `ord("3")` → 51 (0x33)

- `chr(97)` → "a"
- `chr(946)` → "β"
- `chr(51)` → "3"

Strings e caracteres

- **Uma string é uma sequência de caracteres e, por isso, "se assemelha" a uma lista**
- **Todos os caracteres são representados internamente por um número**

  - `ord("a")` → 97 (0x61)
  - `ord("β")` → 946 (0x3b2)
  - `ord("3")` → 51 (0x33)
  - `chr(97)` → "a"
  - `chr(946)` → "β"
  - `chr(51)` → "3"

- **Do ponto de vista do usuário da linguagem, não existem caracteres "sozinhos"; um caracter é simplesmente uma string de comprimento um**

Strings e caracteres

- **Uma string é uma sequência de caracteres e, por isso, "se assemelha" a uma lista**
- **Todos os caracteres são representados internamente por um número**

  - `ord("a")` → 97 (0x61)
  - `ord("β")` → 946 (0x3b2)
  - `ord("3")` → 51 (0x33)
  - `chr(97)` → "a"
  - `chr(946)` → "β"
  - `chr(51)` → "3"

- **Do ponto de vista do usuário da linguagem, não existem caracteres "sozinhos"; um caracter é simplesmente uma string de comprimento um**

https://en.wikipedia.org/wiki/List_of_Unicode_characters

Ordem de strings

Ordem de strings

```
print("Brasil" > "Argentina")
```

Ordem de strings

```
print("Brasil" > "Argentina")
```

True

Ordem de strings

```
print("Brasil" > "Argentina")
print("Brasil" > "Rússia")
```

True

Ordem de strings

```
print("Brasil" > "Argentina")
print("Brasil" > "Rússia")
```

True

False

Ordem de strings

```
print("Brasil" > "Argentina")
print("Brasil" > "Rússia")
```

```
True
False
```

- **Os números correspondentes a cada letra estão “convenientemente” em ordem**

Ordem de strings

```
print("Brasil" > "Argentina")
print("Brasil" > "Rússia")
```

```
True
False
```

- **Os números correspondentes a cada letra estão “convenientemente” em ordem**
- **< e > funcionam para ordem alfabética 😀**

Ordem de strings

```
print("Brasil" > "Argentina")
print("Brasil" > "Rússia")
```

```
True
False
```

- **Os números correspondentes a cada letra estão “convenientemente” em ordem**
- **< e > funcionam para ordem alfabética 😀**
- **Mas com acentos a coisa não se mantém 😢**

Exercícios

Checando o conteúdo da lista

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

Checando o conteúdo da lista

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):
```

Checando o conteúdo da lista

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):
    tem = False
```

Checando o conteúdo da lista

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):
    tem = False




    if not tem:
        l.append(n)
```

Checando o conteúdo da lista

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):
    tem = False
    for i in l:



    if not tem:
        l.append(n)
```

Checando o conteúdo da lista

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):
    tem = False
    for i in l:
        if i == n:
            tem = True
    if not tem:
        l.append(n)
```

Checando o conteúdo da lista

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):
```

Checando o conteúdo da lista

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:
```

Checando o conteúdo da lista

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)
```

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 = 0

    for n in 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




    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 n in range(i:f):

    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 n in range(i:f):
        soma += l[n]
    return soma
```

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):
    nova = []



    return nova
```

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):
    nova = []
    for n in l:



    return nova
```

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):
    nova = []
    for n in l:
        if n >= 0:
            nova.append(n)
    return nova
```

Removendo indesejáveis

Escreva uma função que recebe uma lista de inteiros e remove dessa lista os elementos menores que zero

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

Removendo indesejáveis

Escreva uma função que recebe uma lista de inteiros e remove dessa lista os elementos menores que zero

```
def remove_negativos(l):
    i = 0
```

Removendo indesejáveis

Escreva uma função que recebe uma lista de inteiros e remove dessa lista os elementos menores que zero

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




        i += 1
```

Removendo indesejáveis

Escreva uma função que recebe uma lista de inteiros e remove dessa lista os elementos menores que zero

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

        i += 1
```

Removendo indesejáveis

Escreva uma função que recebe uma lista de inteiros e remove dessa lista os elementos menores que zero

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

        i += 1
l = [1, 2, -1, -2, 3, -4]
remove_negativos(l)
print(l)
```

Removendo indesejáveis

Escreva uma função que recebe uma lista de inteiros e remove dessa lista os elementos menores que zero

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

        i += 1
l = [1, 2, -1, -2, 3, -4]
remove_negativos(l)
print(l)
```

```
[1, 2, -2, 3]
```

Removendo indesejáveis

Escreva uma função que recebe uma lista de inteiros e remove dessa lista os elementos menores que zero

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

        i += 1
l = [1, 2, -1, -2, 3, -4]
remove_negativos(l)
print(l)
```

```
[1, 2, -2, 3]
```

Removendo indesejáveis

Escreva uma função que recebe uma lista de inteiros e remove dessa lista os elementos menores que zero

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

        i += 1
```

Removendo indesejáveis

Escreva uma função que recebe uma lista de inteiros e remove dessa lista os elementos menores que zero

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

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):
    limite = -1



        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:
            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:]
```

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

```
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:

            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):

        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):

        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):

        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):

        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
```

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
```

Exercício

Escreva uma função equivalente a `.split()`

Exercício

Escreva uma função equivalente a `.split()`

```
def esplita(s):
```

Exercício

Escreva uma função equivalente a `.split()`

```
def esplita(s):
    l = []




    return l
```

Exercício

Escreva uma função equivalente a `.split()`

```
def esplita(s):
    l = []
    palavra = ""
    for char in s:
        if char != " ":
            palavra += char









    return l
```

Exercício

Escreva uma função equivalente a `.split()`

```
def esplita(s):
    l = []
    palavra = ""
    for char in s:
        if char != " ":
            palavra += char
        elif palavra != "":
            l.append(palavra)




    return l
```

Exercício

Escreva uma função equivalente a `.split()`

```
def esplita(s):
    l = []
    palavra = ""
    for char in s:
        if char != " ":
            palavra += char
        elif palavra != "":
            l.append(palavra)
            palavra = ""


    return l
```

Exercício

Escreva uma função equivalente a `.split()`

```
def esplita(s):
    l = []
    palavra = ""
    for char in s:
        if char != " ":
            palavra += char
        elif palavra != "":
            l.append(palavra)
            palavra = ""
    if palavra != "":
        l.append(palavra)
    return l
```

Exercício

Escreva uma função equivalente a `.split()`

```
def esplita(s):
    l = []




    return l
```

Exercício

Escreva uma função equivalente a `.split()`

```
def esplita(s):
    l = []
    i = 0
    for j in range(len(s)):









    return l
```

Exercício

Escreva uma função equivalente a `.split()`

```
def esplita(s):
    l = []
    i = 0
    for j in range(len(s)):
        if s[j] == " ":









    return l
```

Exercício

Escreva uma função equivalente a `.split()`

```
def esplita(s):
    l = []
    i = 0
    for j in range(len(s)):
        if s[j] == " ":

                l.append(s[i:j])



    return l
```

Exercício

Escreva uma função equivalente a `.split()`

```
def esplita(s):
    l = []
    i = 0
    for j in range(len(s)):
        if s[j] == " ":
            if j > i:
                l.append(s[i:j])




    return l
```

Exercício

Escreva uma função equivalente a `.split()`

```
def esplita(s):
    l = []
    i = 0
    for j in range(len(s)):
        if s[j] == " ":
            if j > i:
                l.append(s[i:j])
            i = j +1


    return l
```

Exercício

Escreva uma função equivalente a `.split()`

```
def esplita(s):
    l = []
    i = 0
    for j in range(len(s)):
        if s[j] == " ":
            if j > i:
                l.append(s[i:j])
            i = j +1
    if i < len(s):
            l.append(s[i:])
    return l
```