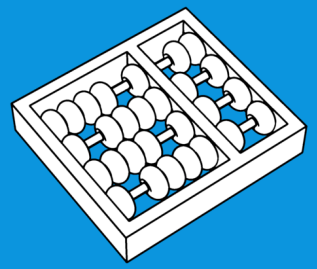
Listas e repetição. De novo?!

MC102 - Algoritmos e Programação de Computadores

Santiago Valdés Ravelo
https://ic.unicamp.br/~santiago/
ravelo@unicamp.br

03/25

7



"Os índices de listas deveriam começar com 0 *ou com* 1*? Minha proposta de* 0,5 *foi rejeitada [acredito que] sem uma consideração adequada."*

Stan Kelly-Bootle.

Dúvidas da aula anterior

Dúvidas selecionadas

- Não entendi como usar o for x in range(n).

Dúvidas selecionadas

- Não entendi como usar o for x in range(n).
- Como funciona a variável depois do “for”? Por exemplo: for “k”.

Dúvidas selecionadas

- Não entendi como usar o for x in range(n).
- Como funciona a variável depois do "for"? Por exemplo: for "k".
- O que acontece com range(-1), daria erro?

Dúvidas selecionadas

- Não entendi como usar o for x in range(n).
- Como funciona a variável depois do "for"? Por exemplo: for "k".
- O que acontece com range(-1), daria erro?
- Quando eu atribuo um valor de uma variável $X = 5$ em uma lista L (L.append(X)), caso eu mude o valor da variável posteriormente $X = 0$, a L[0] continuará 5 ou será 0?

Dúvidas selecionadas

- Não entendi como usar o for x in range(n).
- Como funciona a variável depois do "for"? Por exemplo: for "k".
- O que acontece com range(-1), daria erro?
- Quando eu atribuo um valor de uma variável $X = 5$ em uma lista L (L.append(X)), caso eu mude o valor da variável posteriormente $X = 0$, a L[0] continuará 5 ou será 0?
- Existem mais tipos de for além do in e do in range?

Dúvidas selecionadas

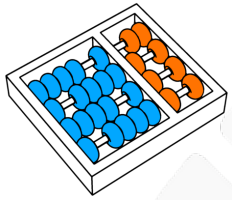
- Não entendi como usar o for x in range(n).
- Como funciona a variável depois do "for"? Por exemplo: for "k".
- O que acontece com range(-1), daria erro?
- Quando eu atribuo um valor de uma variável $X = 5$ em uma lista L (L.append(X)), caso eu mude o valor da variável posteriormente $X = 0$, a L[0] continuará 5 ou será 0?
- Existem mais tipos de for além do in e do in range?
- Qual é o limite de dados que podem ser armazenados em uma lista?

Dúvidas selecionadas

- Não entendi como usar o for x in range(n).
- Como funciona a variável depois do "for"? Por exemplo: for "k".
- O que acontece com range(-1), daria erro?
- Quando eu atribuo um valor de uma variável $X = 5$ em uma lista L (L.append(X)), caso eu mude o valor da variável posteriormente $X = 0$, a L[0] continuará 5 ou será 0?
- Existem mais tipos de for além do in e do in range?
- Qual é o limite de dados que podem ser armazenados em uma lista?
- O que acontece se a condição do while for falsa antes de entrar no bloco?

Dúvidas selecionadas

- Não entendi como usar o for x in range(n).
- Como funciona a variável depois do "for"? Por exemplo: for "k".
- O que acontece com range(-1), daria erro?
- Quando eu atribuo um valor de uma variável $X = 5$ em uma lista L (L.append(X)), caso eu mude o valor da variável posteriormente $X = 0$, a L[0] continuará 5 ou será 0?
- Existem mais tipos de for além do in e do in range?
- Qual é o limite de dados que podem ser armazenados em uma lista?
- O que acontece se a condição do while for falsa antes de entrar no bloco?
- No comando 'for', eu posso colocar uma conta em vez de uma variável? Por exemplo: for $(a + b)$ in range(n).



Dúvidas selecionadas

- Não entendi como usar o for x in range(n).
- Como funciona a variável depois do "for"? Por exemplo: for "k".
- O que acontece com range(-1), daria erro?
- Quando eu atribuo um valor de uma variável X = 5 em uma lista L (L.append(X)), caso eu mude o valor da variável posteriormente X = 0, a L[0] continuará 5 ou será 0?
- Existem mais tipos de for além do in e do in range?
- Qual é o limite de dados que podem ser armazenados em uma lista?
- O que acontece se a condição do while for falsa antes de entrar no bloco?
- No comando 'for', eu posso colocar uma conta em vez de uma variável? Por exemplo: for (a + b) in range(n).
- Posso colocar listas dentro de uma lista? Por exemplo, lista = [[var, var2], '5', 12] ?

Dúvidas selecionadas

- Não entendi como usar o for x in range(n).
- Como funciona a variável depois do "for"? Por exemplo: for "k".
- O que acontece com range(-1), daria erro?
- Quando eu atribuo um valor de uma variável X = 5 em uma lista L (L.append(X)), caso eu mude o valor da variável posteriormente X = 0, a L[0] continuará 5 ou será 0?
- Existem mais tipos de for além do in e do in range?
- Qual é o limite de dados que podem ser armazenados em uma lista?
- O que acontece se a condição do while for falsa antes de entrar no bloco?
- No comando 'for', eu posso colocar uma conta em vez de uma variável? Por exemplo: for (a + b) in range(n).
- Posso colocar listas dentro de uma lista? Por exemplo, lista = [[var, var2], '5', 12] ?
- Não entendi direito como adicionar coisas na lista, eu executo o comando ou só escrevo?

Dúvidas selecionadas

- Não entendi como usar o for x in range(n).
- Como funciona a variável depois do "for"? Por exemplo: for "k".
- O que acontece com range(-1), daria erro?
- Quando eu atribuo um valor de uma variável X = 5 em uma lista L (L.append(X)), caso eu mude o valor da variável posteriormente X = 0, a L[0] continuará 5 ou será 0?
- Existem mais tipos de for além do in e do in range?
- Qual é o limite de dados que podem ser armazenados em uma lista?
- O que acontece se a condição do while for falsa antes de entrar no bloco?
- No comando 'for', eu posso colocar uma conta em vez de uma variável? Por exemplo: for (a + b) in range(n).
- Posso colocar listas dentro de uma lista? Por exemplo, lista = [[var, var2], '5', 12] ?
- Não entendi direito como adicionar coisas na lista, eu executo o comando ou só escrevo?
- É possível utilizar um range dentro de outro range?

Dúvidas selecionadas

- Não entendi como usar o for x in range(n).
- Como funciona a variável depois do "for"? Por exemplo: for "k".
- O que acontece com range(-1), daria erro?
- Quando eu atribuo um valor de uma variável X = 5 em uma lista L (L.append(X)), caso eu mude o valor da variável posteriormente X = 0, a L[0] continuará 5 ou será 0?
- Existem mais tipos de for além do in e do in range?
- Qual é o limite de dados que podem ser armazenados em uma lista?
- O que acontece se a condição do while for falsa antes de entrar no bloco?
- No comando 'for', eu posso colocar uma conta em vez de uma variável? Por exemplo: for (a + b) in range(n).
- Posso colocar listas dentro de uma lista? Por exemplo, lista = [[var, var2], '5', 12] ?
- Não entendi direito como adicionar coisas na lista, eu executo o comando ou só escrevo?
- É possível utilizar um range dentro de outro range?
- Não é possível criar uma lista com variáveis que ainda não foram definidas, por exemplo que precisam receber um input?

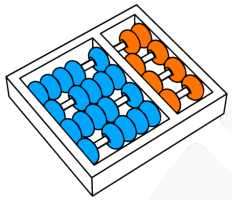
Dúvidas selecionadas

- Não entendi como usar o for x in range(n).
- Como funciona a variável depois do “for”? Por exemplo: for “k”.
- O que acontece com range(-1), daria erro?
- Quando eu atribuo um valor de uma variável X = 5 em uma lista L (L.append(X)), caso eu mude o valor da variável posteriormente X = 0, a L[0] continuará 5 ou será 0?
- Existem mais tipos de for além do in e do in range?
- Qual é o limite de dados que podem ser armazenados em uma lista?
- O que acontece se a condição do while for falsa antes de entrar no bloco?
- No comando ’for’, eu posso colocar uma conta em vez de uma variável? Por exemplo: for (a + b) in range(n).
- Posso colocar listas dentro de uma lista? Por exemplo, lista = [[var, var2], ’5’, 12] ?
- Não entendi direito como adicionar coisas na lista, eu executo o comando ou só escrevo?
- É possível utilizar um range dentro de outro range?
- Não é possível criar uma lista com variáveis que ainda não foram definidas, por exemplo que precisam receber um input?
- Além da praticidade, existe algo que o "for"faz e o "while"não consegue? e o contrário? Há alguma diferença de performance?

Divisores e primos

Divisores

Faça um programa que lê o número **n** e imprime todos os divisores.



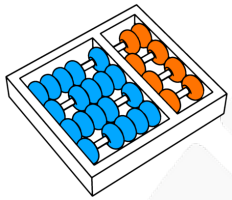
Divisores

Faça um programa que lê o número **n** e imprime todos os divisores.

```
1  n = int(input("Entre com o n: "))
2  d = 2
3  while n != 1:
4      if n % d == 0:
5          n //= d
6          print(d)
7      else:
8          d += 1
```

Imprimindo primos

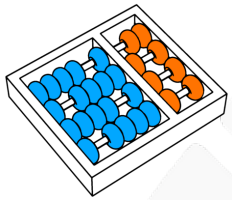
Já sabemos verificar se um número é primo.



Imprimindo primos

Já sabemos verificar se um número é primo.

```
1  p = int(input("Entre com o p: "))
2  primo = True
3  k = 2
4  while k * k <= p and primo:
5      if p % k == 0:
6          primo = False
7      k += 1
8  print(p >= 2 and primo)
```

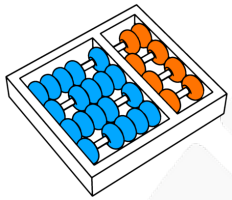


Imprimindo primos

Já sabemos verificar se um número é primo.

```
1  p = int(input("Entre com o p: "))
2  primo = True
3  k = 2
4  while k * k <= p and primo:
5      if p % k == 0:
6          primo = False
7      k += 1
8  print(p >= 2 and primo)
```

Mas, e se quisermos imprimir todos os números primos menores ou iguais a um inteiro n?



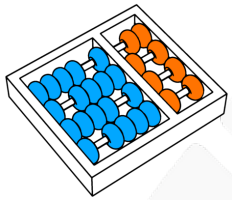
Imprimindo primos

Já sabemos verificar se um número é primo.

```
1  p = int(input("Entre com o p: "))
2  primo = True
3  k = 2
4  while k * k <= p and primo:
5      if p % k == 0:
6          primo = False
7      k += 1
8  print(p >= 2 and primo)
```

Mas, e se quisermos imprimir todos os números primos menores ou iguais a um inteiro n?

- Precisaríamos executar o código acima várias vezes...



Imprimindo primos

Já sabemos verificar se um número é primo.

```
1  p = int(input("Entre com o p: "))
2  primo = True
3  k = 2
4  while k * k <= p and primo:
5      if p % k == 0:
6          primo = False
7      k += 1
8  print(p >= 2 and primo)
```

Mas, e se quisermos imprimir todos os números primos menores ou iguais a um inteiro n?

- Precisaríamos executar o código acima várias vezes...
- Como fazer isso?

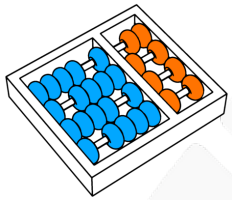
Pergunta

Como repetir o que já repetimos dentro de um laço?

Laços encaixados

Imprimindo primos

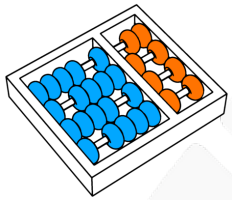
Podemos colocar um laço dentro de outro!



Imprimindo primos

Podemos colocar um laço dentro de outro!

```
1  n = int(input("Entre com o n: "))
2  for p in range(2, n + 1):
3      primo = True
4      k = 2
5      while k * k <= p and primo:
6          if p % k == 0:
7              primo = False
8          k += 1
9      if(primo):
10         print(p)
```



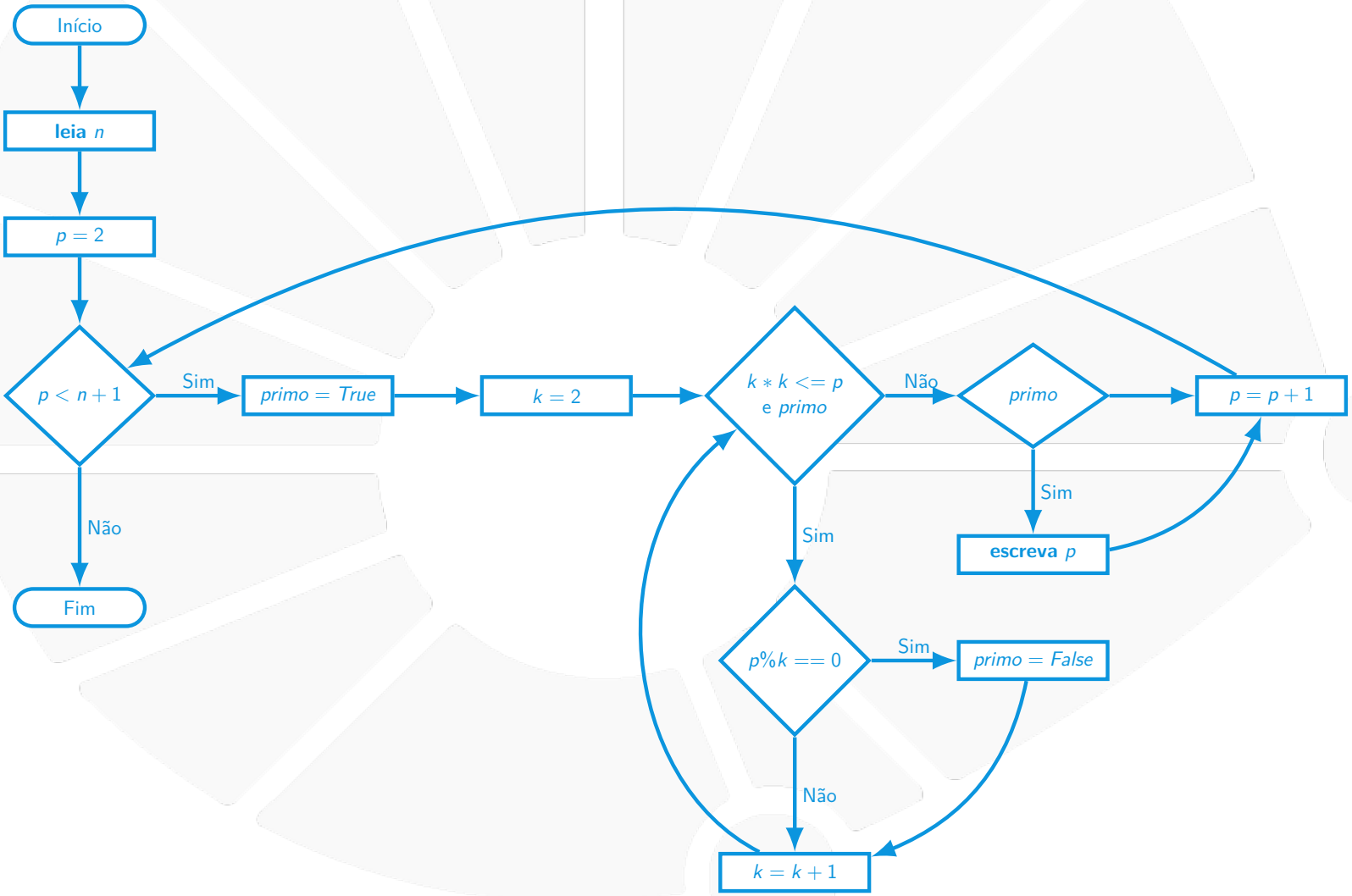
Imprimindo primos

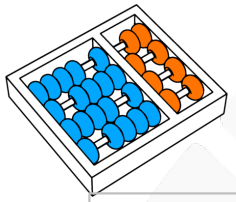
Podemos colocar um laço dentro de outro!

```
n = int(input("Entre com o n: "))
for p in range(2, n + 1):
    primo = True
    k = 2
    while k * k <= p and primo:
        if p % k == 0:
            primo = False
        k += 1
    if(primo):
        print(p)
```

- Vamos simular!

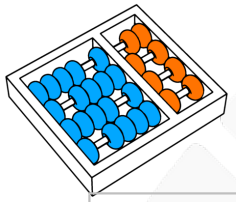
Fluxograma


Início
leia n
$p = 2$
$p < n + 1$
Sim
Não
Fim
$primo = True$
$k = 2$
$k * k <= p$ e $primo$
Sim
Não
$p\%k == 0$
Sim
Não
$primo = False$
$k = k + 1$
$primo$
Sim
escreva p
$p = p + 1$



Versão com **break**

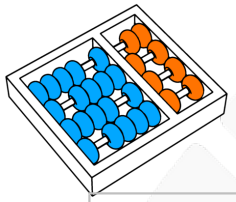
```
1  n = int(input("Entre com o n: "))
2  for p in range(2, n + 1):
3      primo = True
4      k = 2
5      while k * k <= p and primo:
6          if p % k == 0:
7              primo = False
8              break
9          k += 1
10     if(primo):
11         print(p)
```



Versão com **break**

```
1  n = int(input("Entre com o n: "))
2  for p in range(2, n + 1):
3      primo = True
4      k = 2
5      while k * k <= p and primo:
6          if p % k == 0:
7              primo = False
8              break
9          k += 1
10     if(primo):
11         print(p)
```

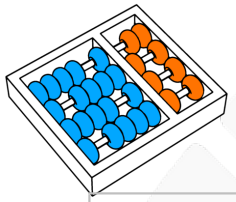
- O **break** interrompe o laço mais interno apenas.



Versão com **break**

```
1  n = int(input("Entre com o n: "))
2  for p in range(2, n + 1):
3      primo = True
4      k = 2
5      while k * k <= p and primo:
6          if p % k == 0:
7              primo = False
8              break
9          k += 1
10     if(primo):
11         print(p)
```

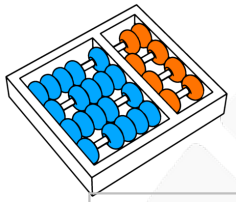
- O **break** interrompe o laço mais interno apenas.
- Ele reduz um pouco a legibilidade do código.



Versão com **break**

```
1  n = int(input("Entre com o n: "))
2  for p in range(2, n + 1):
3      primo = True
4      k = 2
5      while k * k <= p and primo:
6          if p % k == 0:
7              primo = False
8              break
9          k += 1
10     if(primo):
11         print(p)
```

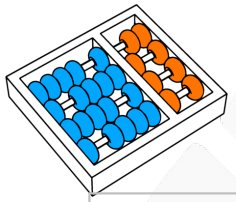
- O **break** interrompe o laço mais interno apenas.
- Ele reduz um pouco a legibilidade do código.
  - Mas é necessário em laços **for** que não possuem condição booleana.



Versão com **break**

```
1  n = int(input("Entre com o n: "))
2  for p in range(2, n + 1):
3      primo = True
4      k = 2
5      while k * k <= p and primo:
6          if p % k == 0:
7              primo = False
8              break
9          k += 1
10     if(primo):
11         print(p)
```

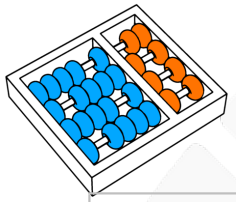
- O **break** interrompe o laço mais interno apenas.
- Ele reduz um pouco a legibilidade do código.
  - Mas é necessário em laços **for** que não possuem condição booleana.
- Existe também um comando chamado **continue**:



Versão com **break**

```
1  n = int(input("Entre com o n: "))
2  for p in range(2, n + 1):
3      primo = True
4      k = 2
5      while k * k <= p and primo:
6          if p % k == 0:
7              primo = False
8              break
9          k += 1
10     if(primo):
11         print(p)
```

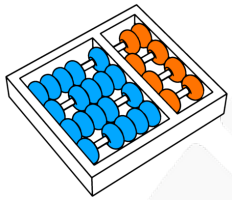
- O **break** interrompe o laço mais interno apenas.
- Ele reduz um pouco a legibilidade do código.
  - Mas é necessário em laços **for** que não possuem condição booleana.
- Existe também um comando chamado **continue**:
  - Ele vai para a próxima iteração do laço mais interno.



Versão com **break**

```
1  n = int(input("Entre com o n: "))
2  for p in range(2, n + 1):
3      primo = True
4      k = 2
5      while k * k <= p and primo:
6          if p % k == 0:
7              primo = False
8              break
9          k += 1
10     if(primo):
11         print(p)
```

- O **break** interrompe o laço mais interno apenas.
- Ele reduz um pouco a legibilidade do código.
  - Mas é necessário em laços **for** que não possuem condição booleana.
- Existe também um comando chamado **continue**:
  - Ele vai para a próxima iteração do laço mais interno.
  - Usado algumas vezes para melhorar legibilidade do código.



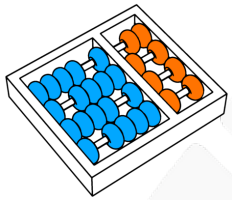
Uma outra forma de achar primos...

Nós sabemos que 2 é primo. . .

Uma outra forma de achar primos...

Nós sabemos que 2 é primo. . .

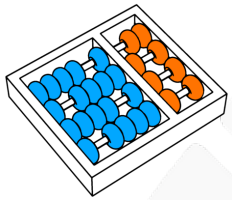
- Pois 2 não pode ter divisores diferentes de 1 e 2.



Uma outra forma de achar primos...

Nós sabemos que 2 é primo. . .

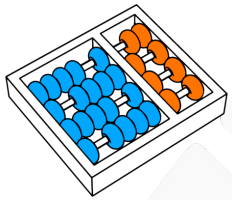
- Pois 2 não pode ter divisores diferentes de 1 e 2.
- Pois um divisor precisa ser menor ou igual a 2.



Uma outra forma de achar primos...

Nós sabemos que 2 é primo. . .

- Pois 2 não pode ter divisores diferentes de 1 e 2.
- Pois um divisor precisa ser menor ou igual a 2.
- E 2 é o segundo número natural.

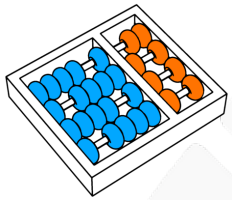


Uma outra forma de achar primos...

Nós sabemos que 2 é primo. . .

- Pois 2 não pode ter divisores diferentes de 1 e 2.
- Pois um divisor precisa ser menor ou igual a 2.
- E 2 é o segundo número natural.

O que ganhamos com a informação que 2 é primo?



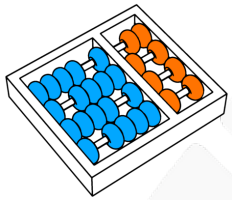
Uma outra forma de achar primos...

Nós sabemos que 2 é primo...

- Pois 2 não pode ter divisores diferentes de 1 e 2.
- Pois um divisor precisa ser menor ou igual a 2.
- E 2 é o segundo número natural.

O que ganhamos com a informação que 2 é primo?

- Que $4, 6, 8, 10, 12, \ldots$ não são primos.



Uma outra forma de achar primos...

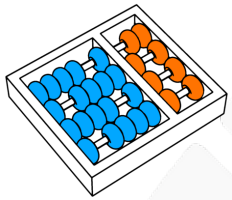
Nós sabemos que 2 é primo...

- Pois 2 não pode ter divisores diferentes de 1 e 2.
- Pois um divisor precisa ser menor ou igual a 2.
- E 2 é o segundo número natural.

O que ganhamos com a informação que 2 é primo?

- Que $4, 6, 8, 10, 12, \ldots$ não são primos.

E o 3?



Uma outra forma de achar primos...

Nós sabemos que 2 é primo...

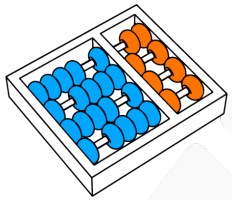
- Pois 2 não pode ter divisores diferentes de 1 e 2.
- Pois um divisor precisa ser menor ou igual a 2.
- E 2 é o segundo número natural.

O que ganhamos com a informação que 2 é primo?

- Que $4, 6, 8, 10, 12, \ldots$ não são primos.

E o 3?

- É primo, pois não é múltiplo dos primos anteriores.



Uma outra forma de achar primos...

Nós sabemos que 2 é primo...

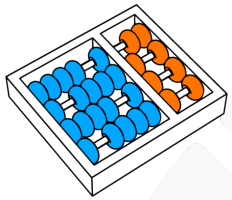
- Pois 2 não pode ter divisores diferentes de 1 e 2.
- Pois um divisor precisa ser menor ou igual a 2.
- E 2 é o segundo número natural.

O que ganhamos com a informação que 2 é primo?

- Que $4, 6, 8, 10, 12, \ldots$ não são primos.

E o 3?

- É primo, pois não é múltiplo dos primos anteriores.
- Portanto, $6, 9, 12, 15, \ldots$ não são primos.



Uma outra forma de achar primos...

Nós sabemos que 2 é primo...

- Pois 2 não pode ter divisores diferentes de 1 e 2.
- Pois um divisor precisa ser menor ou igual a 2.
- E 2 é o segundo número natural.

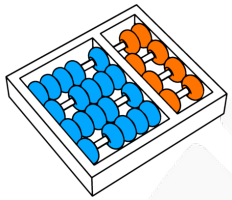
O que ganhamos com a informação que 2 é primo?

- Que $4, 6, 8, 10, 12, \ldots$ não são primos.

E o 3?

- É primo, pois não é múltiplo dos primos anteriores.
- Portanto, $6, 9, 12, 15, \ldots$ não são primos.

O 4 já sabemos que não é primo, mas e o 5?



Uma outra forma de achar primos...

Nós sabemos que 2 é primo...

- Pois 2 não pode ter divisores diferentes de 1 e 2.
- Pois um divisor precisa ser menor ou igual a 2.
- E 2 é o segundo número natural.

O que ganhamos com a informação que 2 é primo?

- Que $4, 6, 8, 10, 12, \ldots$ não são primos.

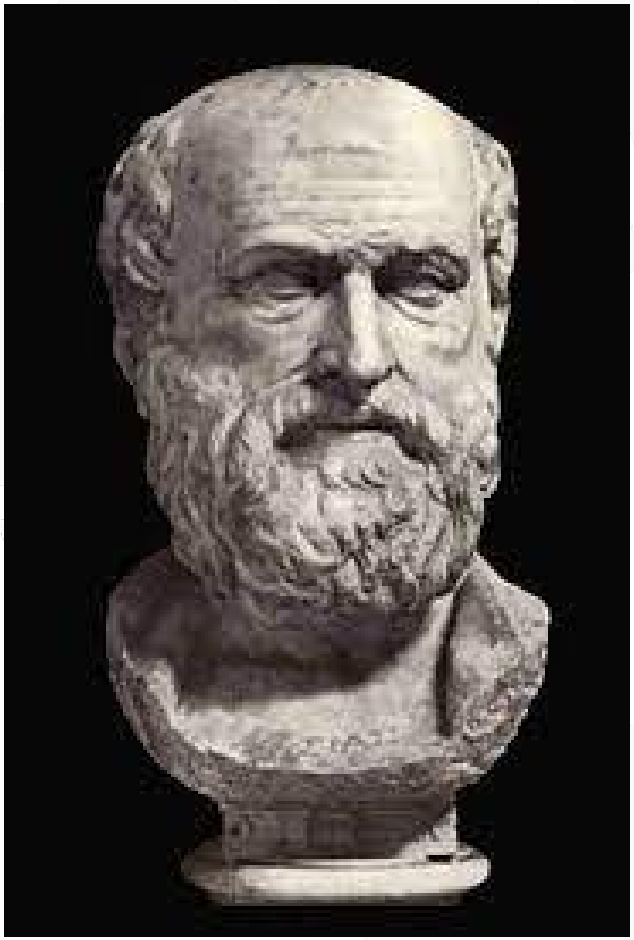
E o 3?

- É primo, pois não é múltiplo dos primos anteriores.
- Portanto, $6, 9, 12, 15, \ldots$ não são primos.

O 4 já sabemos que não é primo, mas e o 5?

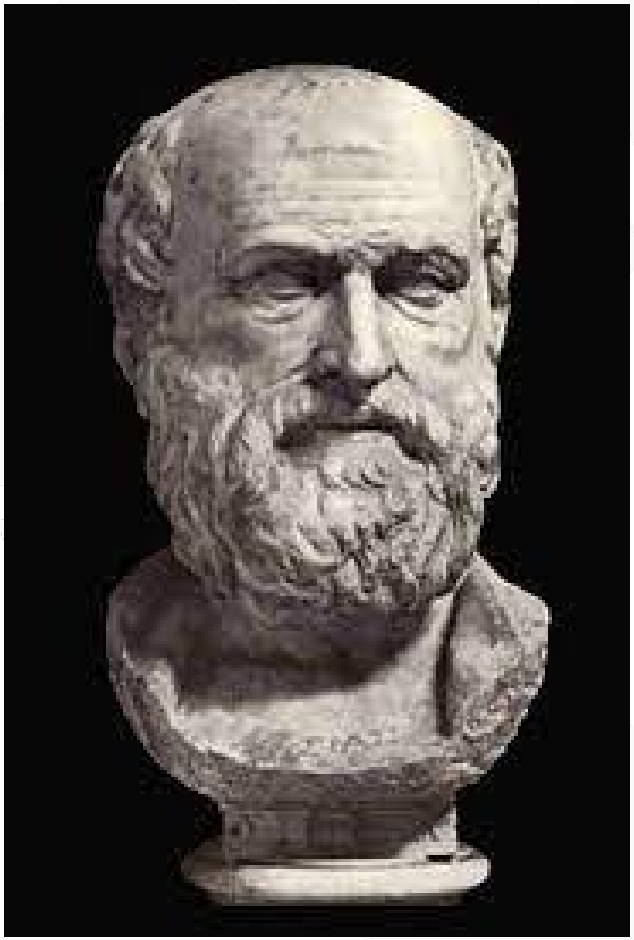
- Mesmo raciocínio do 3...

Eratóstenes de Cirene (276 *AEC* – 194 *AEC*)



Principais áreas de atuação:

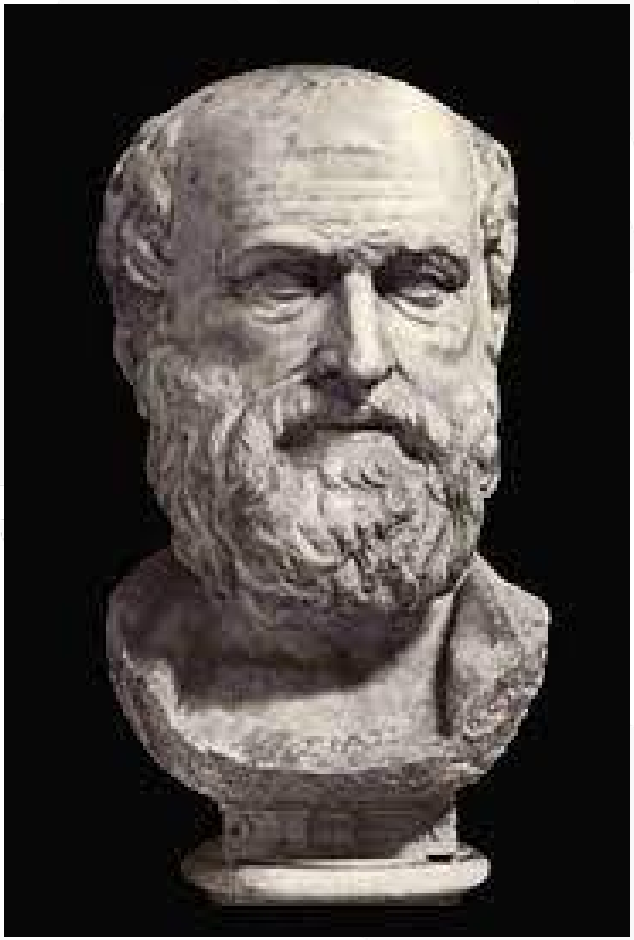
Eratóstenes de Cirene (276 *AEC* – 194 *AEC*)



Principais áreas de atuação:

- Filosofia.

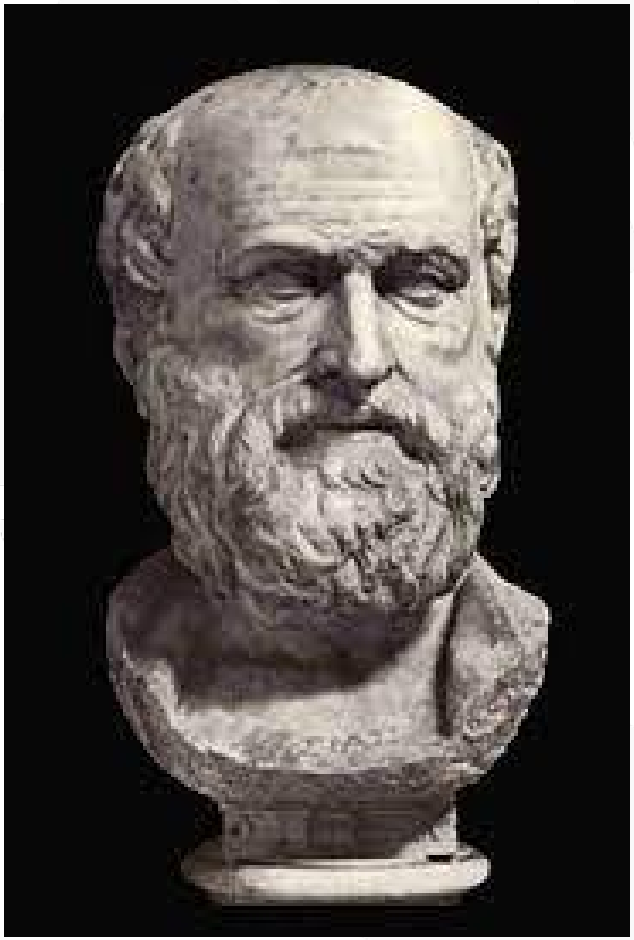
Eratóstenes de Cirene (276 *AEC* – 194 *AEC*)



Principais áreas de atuação:

- Filosofia.
- Matemática.

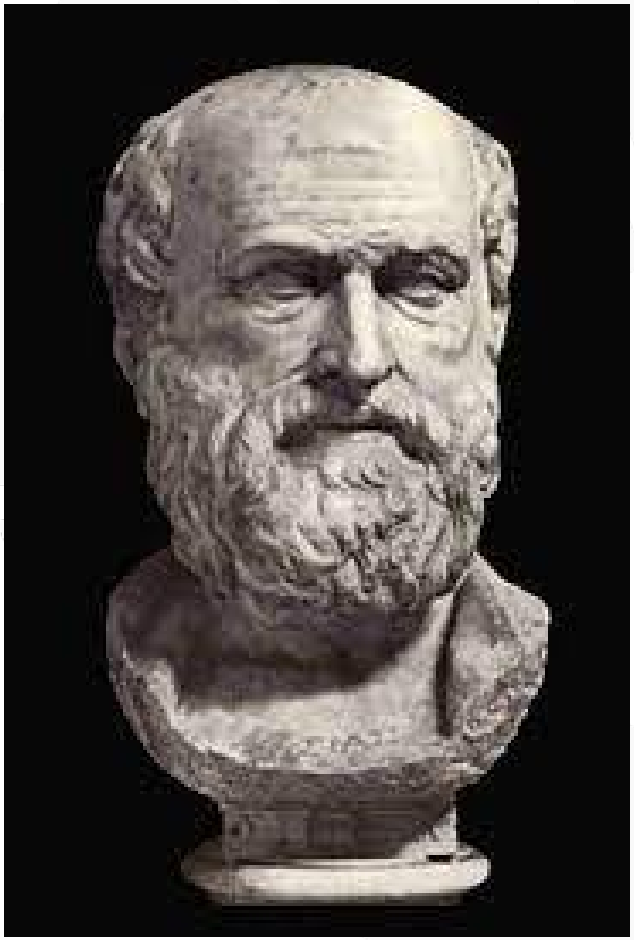
Eratóstenes de Cirene (276 *AEC* – 194 *AEC*)



Principais áreas de atuação:

- Filosofia.
- Matemática.
- Astronomia.

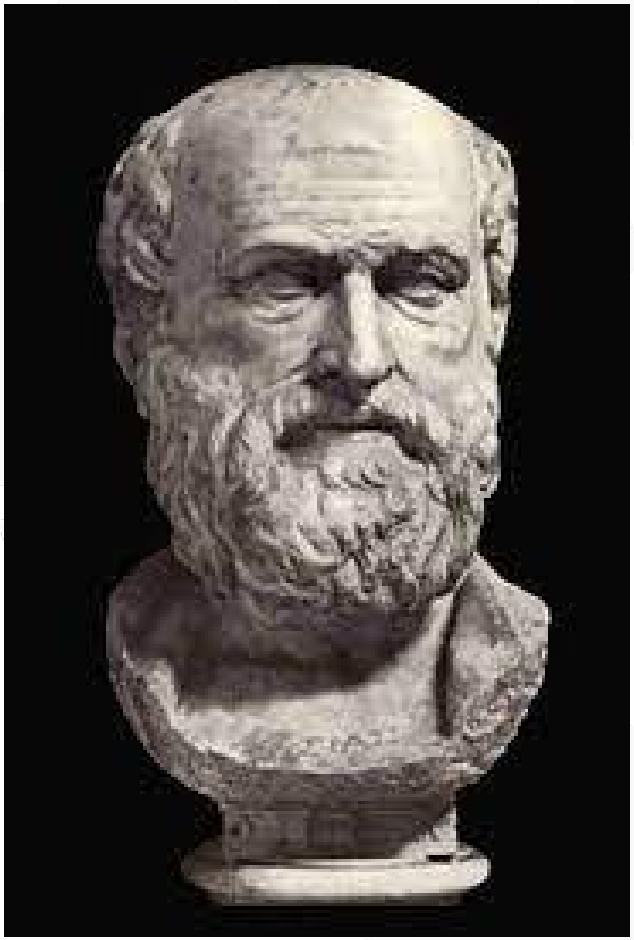
Eratóstenes de Cirene (276 *AEC* – 194 *AEC*)



Principais áreas de atuação:

- Filosofia.
- Matemática.
- Astronomia.
- Literatura.

Eratóstenes de Cirene (276 *AEC* – 194 *AEC*)



Principais áreas de atuação:

- Filosofia.
- Matemática.
- Astronomia.
- Literatura.
- Geografia.

Crivo de Eratóstenes

Pseudocódigo:

```
1 leia n
2 Marque todo número de 2 até n como primo
3 para p = 2 até n
4     se p está marcado como primo
5         escreva p
6         para cada k múltiplo de p menor ou igual que n
7             Marque k como não-primo
```

Crivo de Eratóstenes

Pseudocódigo:

```
leia n
Marque todo número de 2 até n como primo
para p = 2 até n
    se p está marcado como primo
        escreva p
        para cada k múltiplo de p menor ou igual que n
            Marque k como não-primo
```

Exercício: Vamos simular na lousa!

Crivo de Eratóstenes

Pseudocódigo:

```
leia n
Marque todo número de 2 até n como primo
para p = 2 até n
    se p está marcado como primo
        escreva p
        para cada k múltiplo de p menor ou igual que n
            Marque k como não-primo
```

Exercício: Vamos simular na lousa!

Exercício: Vamos fazer em Python!

Solução

Versão otimizada:

Solução

Versão otimizada:

- Para um primo p, já marcamos alguns múltiplos de p.

Solução

Versão otimizada:

- Para um primo p, já marcamos alguns múltiplos de p.
- $2p$ já foi marcado pelo 2, $3p$ por 3, $4p$ por 2, $5p$ por 5, ...

Solução

Versão otimizada:

- Para um primo p, já marcamos alguns múltiplos de p.
- $2p$ já foi marcado pelo 2, $3p$ por 3, $4p$ por 2, $5p$ por 5, ...
- $(p-1)p$ foi marcado por um divisor de $(p-1)$.

Solução

Versão otimizada:

- Para um primo p, já marcamos alguns múltiplos de p.
- $2p$ já foi marcado pelo 2, $3p$ por 3, $4p$ por 2, $5p$ por 5, ...
- $(p-1)p$ foi marcado por um divisor de $(p-1)$.
- O primeiro elemento a ser marcado é p^2.

Solução

Versão otimizada:

- Para um primo p, já marcamos alguns múltiplos de p.
- $2p$ já foi marcado pelo 2, $3p$ por 3, $4p$ por 2, $5p$ por 5, ...
- $(p-1)p$ foi marcado por um divisor de $(p-1)$.
- O primeiro elemento a ser marcado é p^2.

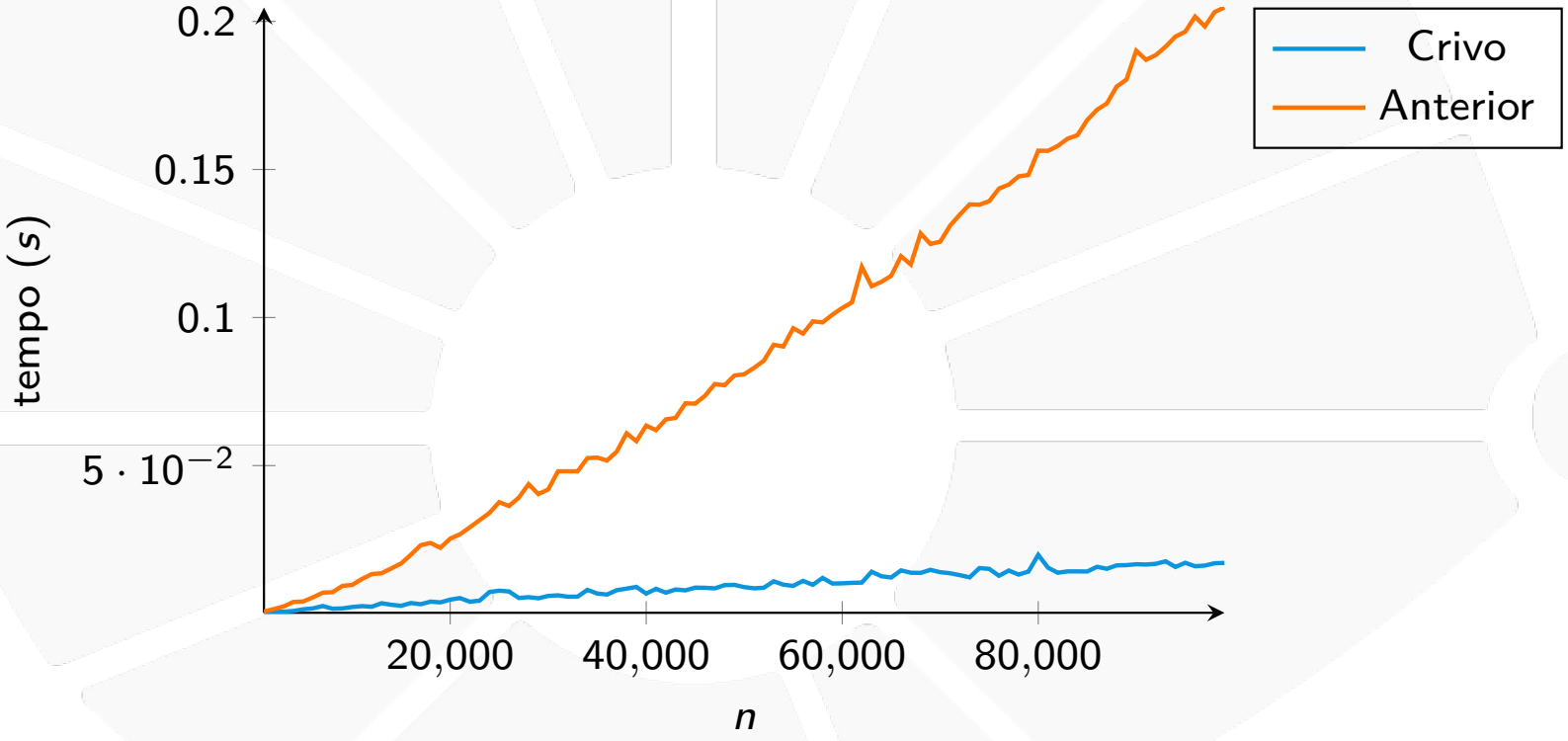
```
1  n = int(input("Entre com n: "))
2  # cria uma lista de n + 1 elementos todos True
3  eh_primo = []
4  for i in range(n + 1):
5      eh_primo.append(True)
6  eh_primo[0] = eh_primo[1] = False
7  for p in range(2, n + 1):
8      if eh_primo[p]:
9          print(p)
10         # podemos começar em p * p e pulamos de p em p
11         for k in range(p * p, n + 1, p):
12             eh_primo[k] = False
```

Vale a pena usar o Crivo?

Comparação de tempo de execução dos dois algoritmos

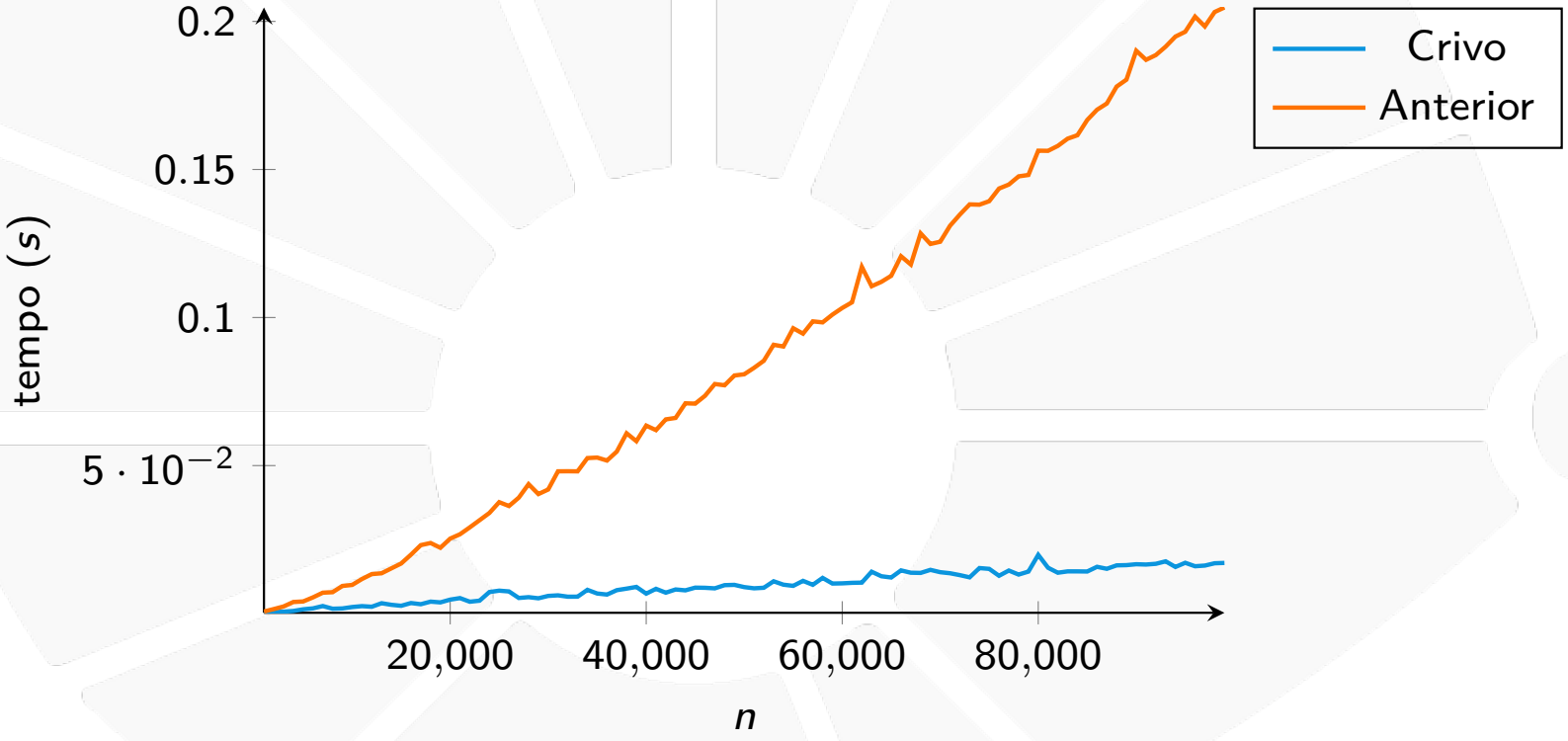
Vale a pena usar o Crivo?

Comparação de tempo de execução dos dois algoritmos



Vale a pena usar o Crivo?

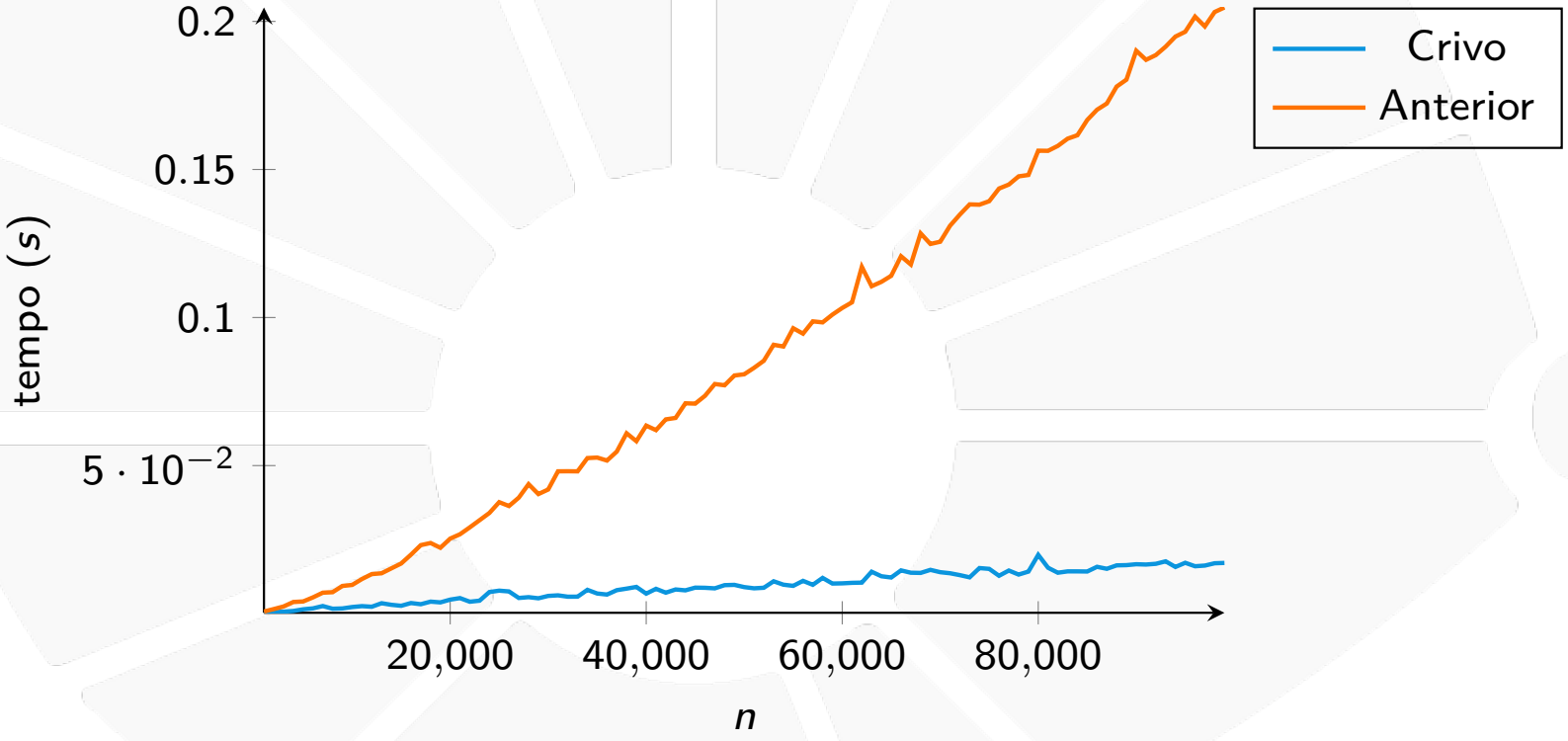
Comparação de tempo de execução dos dois algoritmos



Dois algoritmos que fazem exatamente a mesma coisa. . .

Vale a pena usar o Crivo?

Comparação de tempo de execução dos dois algoritmos



Dois algoritmos que fazem exatamente a mesma coisa...

- Mas um é muito mais rápido do que o outro!

Outras operações com listas

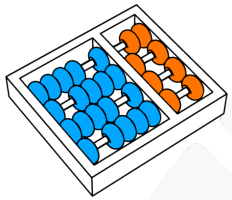
Operações

Existem várias operações que podemos fazer com listas:

Operações

Existem várias operações que podemos fazer com listas:

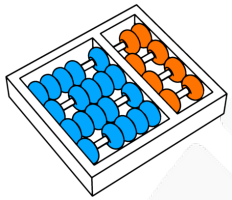
- **lista.clear()**: remove todos os elementos da **lista**.



Operações

Existem várias operações que podemos fazer com listas:

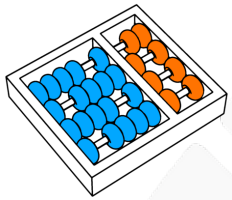
- **lista.clear()**: remove todos os elementos da **lista**.
- **len(lista)**: informa o número de elementos da **lista**.



Operações

Existem várias operações que podemos fazer com listas:

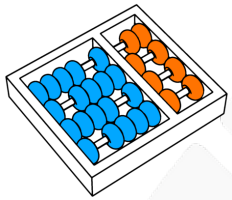
- **lista.clear()**: remove todos os elementos da **lista**.
- **len(lista)**: informa o número de elementos da **lista**.
- **lista.copy()**: devolve uma cópia da **lista**.



Operações

Existem várias operações que podemos fazer com listas:

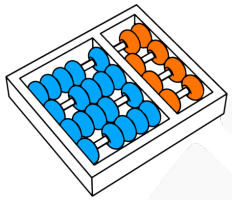
- **lista.clear()**: remove todos os elementos da **lista**.
- **len(lista)**: informa o número de elementos da **lista**.
- **lista.copy()**: devolve uma cópia da **lista**.
- **lista1.extend(lista2)**: adiciona os elementos de **lista2** em **lista1**.



Operações

Existem várias operações que podemos fazer com listas:

- **lista.clear()**: remove todos os elementos da **lista**.
- **len(lista)**: informa o número de elementos da **lista**.
- **lista.copy()**: devolve uma cópia da **lista**.
- **lista1.extend(lista2)**: adiciona os elementos de **lista2** em **lista1**.
- **lista.index(x)**: informa o primeiro índice da **lista** que contém o elemento **x** (lança um **ValueError** se não existir).



Operações

Existem várias operações que podemos fazer com listas:

- **lista.clear()**: remove todos os elementos da **lista**.
- **len(lista)**: informa o número de elementos da **lista**.
- **lista.copy()**: devolve uma cópia da **lista**.
- **lista1.extend(lista2)**: adiciona os elementos de **lista2** em **lista1**.
- **lista.index(x)**: informa o primeiro índice da **lista** que contém o elemento **x** (lança um **ValueError** se não existir).
- **lista.reverse()**: inverte a **lista**.

Mais operações

Outras operações:

Mais operações

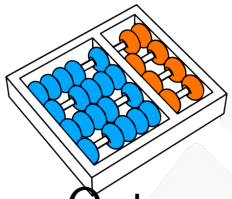
Outras operações:

- **lista.insert(i, x)**: insere **x** antes do índice **i**.

Mais operações

Outras operações:

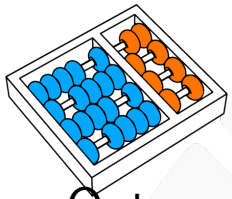
- **lista.insert(i, x)**: insere **x** antes do índice **i**.
  - **x** fica no índice **i**.



Mais operações

Outras operações:

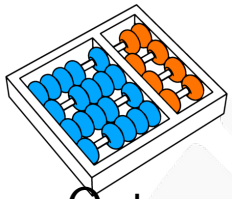
- **lista.insert(i, x)**: insere **x** antes do índice **i**.
  - **x** fica no índice **i**.
- **lista.pop(i)**: remove o elemento de índice **i** e o devolve.



Mais operações

Outras operações:

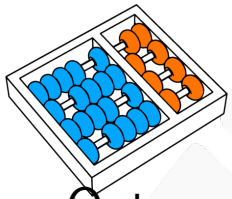
- **lista.insert(i, x)**: insere **x** antes do índice **i**.
  - **x** fica no índice **i**.
- **lista.pop(i)**: remove o elemento de índice **i** e o devolve.
  - Ou então: **del lista[i]**.



Mais operações

Outras operações:

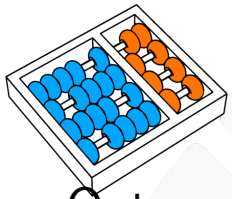
- **lista.insert(i, x)**: insere **x** antes do índice **i**.
  - **x** fica no índice **i**.
- **lista.pop(i)**: remove o elemento de índice **i** e o devolve.
  - Ou então: **del lista[i]**.
- **lista.pop()**: remove o último elemento da **lista** e o devolve.



Mais operações

Outras operações:

- **lista.insert(i, x)**: insere **x** antes do índice **i**.
  - **x** fica no índice **i**.
- **lista.pop(i)**: remove o elemento de índice **i** e o devolve.
  - Ou então: **del lista[i]**.
- **lista.pop()**: remove o último elemento da **lista** e o devolve.
- **lista.remove(x)**: remove a primeira ocorrência de **x** da **lista** (e lança um **ValueError** se não existir).

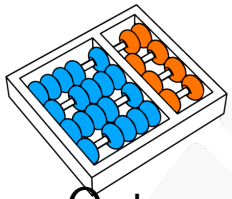


Mais operações

Outras operações:

- **lista.insert(i, x)**: insere **x** antes do índice **i**.
  - **x** fica no índice **i**.
- **lista.pop(i)**: remove o elemento de índice **i** e o devolve.
  - Ou então: **del lista[i]**.
- **lista.pop()**: remove o último elemento da **lista** e o devolve.
- **lista.remove(x)**: remove a primeira ocorrência de **x** da **lista** (e lança um **ValueError** se não existir).

Também é possível:



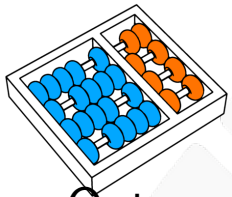
Mais operações

Outras operações:

- **lista.insert(i, x)**: insere **x** antes do índice **i**.
  - **x** fica no índice **i**.
- **lista.pop(i)**: remove o elemento de índice **i** e o devolve.
  - Ou então: **del lista[i]**.
- **lista.pop()**: remove o último elemento da **lista** e o devolve.
- **lista.remove(x)**: remove a primeira ocorrência de **x** da **lista** (e lança um **ValueError** se não existir).

Também é possível:

- Somar duas listas: **lista1 + lista2** ou **lista1 += lista2**.



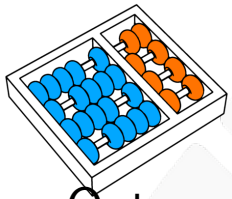
Mais operações

Outras operações:

- **lista.insert(i, x)**: insere **x** antes do índice **i**.
  - **x** fica no índice **i**.
- **lista.pop(i)**: remove o elemento de índice **i** e o devolve.
  - Ou então: **del lista[i]**.
- **lista.pop()**: remove o último elemento da **lista** e o devolve.
- **lista.remove(x)**: remove a primeira ocorrência de **x** da **lista** (e lança um **ValueError** se não existir).

Também é possível:

- Somar duas listas: **lista1 + lista2** ou **lista1 += lista2**.
  - Concatena **lista1** com **lista2**.



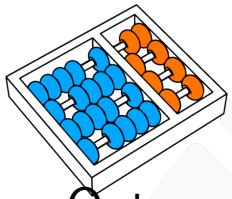
Mais operações

Outras operações:

- **lista.insert(i, x)**: insere **x** antes do índice **i**.
  - **x** fica no índice **i**.
- **lista.pop(i)**: remove o elemento de índice **i** e o devolve.
  - Ou então: **del lista[i]**.
- **lista.pop()**: remove o último elemento da **lista** e o devolve.
- **lista.remove(x)**: remove a primeira ocorrência de **x** da **lista** (e lança um **ValueError** se não existir).

Também é possível:

- Somar duas listas: **lista1 + lista2** ou **lista1 += lista2**.
  - Concatena **lista1** com **lista2**.
- Multiplicar uma lista por um inteiro: **lista * 2** ou **lista *= 2**.



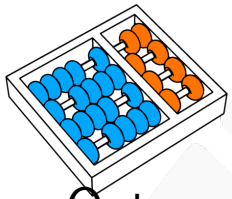
Mais operações

Outras operações:

- **lista.insert(i, x)**: insere **x** antes do índice **i**.
  - **x** fica no índice **i**.
- **lista.pop(i)**: remove o elemento de índice **i** e o devolve.
  - Ou então: **del lista[i]**.
- **lista.pop()**: remove o último elemento da **lista** e o devolve.
- **lista.remove(x)**: remove a primeira ocorrência de **x** da **lista** (e lança um **ValueError** se não existir).

Também é possível:

- Somar duas listas: **lista1 + lista2** ou **lista1 += lista2**.
  - Concatena **lista1** com **lista2**.
- Multiplicar uma lista por um inteiro: **lista * 2** ou **lista *= 2**.
  - Concatena **lista** com **lista** várias vezes.



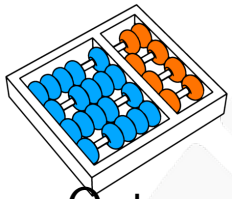
Mais operações

Outras operações:

- **lista.insert(i, x)**: insere **x** antes do índice **i**.
  - **x** fica no índice **i**.
- **lista.pop(i)**: remove o elemento de índice **i** e o devolve.
  - Ou então: **del lista[i]**.
- **lista.pop()**: remove o último elemento da **lista** e o devolve.
- **lista.remove(x)**: remove a primeira ocorrência de **x** da **lista** (e lança um **ValueError** se não existir).

Também é possível:

- Somar duas listas: **lista1 + lista2** ou **lista1 += lista2**.
  - Concatena **lista1** com **lista2**.
- Multiplicar uma lista por um inteiro: **lista * 2** ou **lista *= 2**.
  - Concatena **lista** com **lista** várias vezes.
- Na primeira opção, uma nova lista é criada.



Mais operações

Outras operações:

- **lista.insert(i, x)**: insere **x** antes do índice **i**.
  - **x** fica no índice **i**.
- **lista.pop(i)**: remove o elemento de índice **i** e o devolve.
  - Ou então: **del lista[i]**.
- **lista.pop()**: remove o último elemento da **lista** e o devolve.
- **lista.remove(x)**: remove a primeira ocorrência de **x** da **lista** (e lança um **ValueError** se não existir).

Também é possível:

- Somar duas listas: **lista1 + lista2** ou **lista1 += lista2**.
  - Concatena **lista1** com **lista2**.
- Multiplicar uma lista por um inteiro: **lista * 2** ou **lista *= 2**.
  - Concatena **lista** com **lista** várias vezes.
- Na primeira opção, uma nova lista é criada.
- Na segunda, a lista é alterada (como no **extend**).

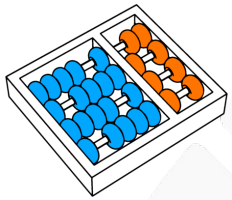
Operações específicas

Algumas operações são para listas de tipos específicos:

Operações específicas

Algumas operações são para listas de tipos específicos:

- **lista.sort()**: ordena os elementos da **lista** (é necessário que os lementos possam ser comparados).



Operações específicas

Algumas operações são para listas de tipos específicos:

- **lista.sort()**: ordena os elementos da **lista** (é necessário que os lementos possam ser comparados).
- **sum(lista)**: soma os elementos da **lista** e devolve o valor (os elementos precisam ser numéricos).

Índices

Os índices podem ser também negativos:

Índices

Os índices podem ser também negativos:

- **lista[-1]** é o último elemento,

Índices

Os índices podem ser também negativos:

- **lista[-1]** é o último elemento,
- **lista[-2]** é o penúltimo elemento,

Índices

Os índices podem ser também negativos:

- **lista[-1]** é o último elemento,
- **lista[-2]** é o penúltimo elemento,
- assim por diante, até

Índices

Os índices podem ser também negativos:

- **lista[-1]** é o último elemento,
- **lista[-2]** é o penúltimo elemento,
- assim por diante, até
- **lista[-len(lista)]** que é o primeiro elemento.

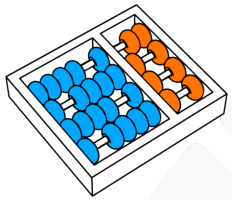
Slices

Podemos fatiar (***slice***) a lista:

Slices

Podemos fatiar (***slice***) a lista:

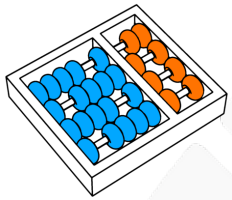
- **lista[i:f]** nos dá os elementos: **lista[i]**, **lista[i + 1]**, . . . , **lista[f - 1]**.



Slices

Podemos fatiar (***slice***) a lista:

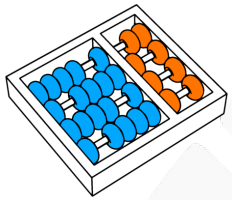
- **lista[i:f]** nos dá os elementos: **lista[i]**, **lista[i + 1]**, . . . , **lista[f - 1]**.
  - **lista[:f]** - até o elemento **f - 1** (omitindo **i**).



Slices

Podemos fatiar (***slice***) a lista:

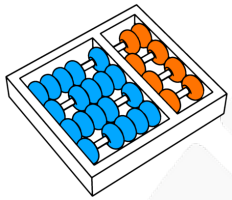
- **lista[i:f]** nos dá os elementos: **lista[i]**, **lista[i + 1]**, . . . , **lista[f - 1]**.
  - **lista[:f]** - até o elemento **f - 1** (omitindo **i**).
  - **lista[i:]** - todos os elementos a partir do **i** (omitindo **f**).



Slices

Podemos fatiar (***slice***) a lista:

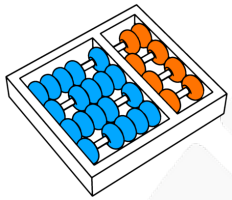
- **lista[i:f]** nos dá os elementos: **lista[i]**, **lista[i + 1]**, . . . , **lista[f - 1]**.
  - **lista[:f]** - até o elemento **f - 1** (omitindo **i**).
  - **lista[i:]** - todos os elementos a partir do **i** (omitindo **f**).
  - **lista[:]** - toda a lista (omitindo **i** e **f**).



Slices

Podemos fatiar (***slice***) a lista:

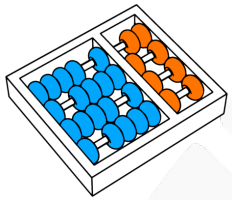
- **lista[i:f]** nos dá os elementos: **lista[i]**, **lista[i + 1]**, . . . , **lista[f - 1]**.
  - **lista[:f]** - até o elemento **f - 1** (omitindo **i**).
  - **lista[i:]** - todos os elementos a partir do **i** (omitindo **f**).
  - **lista[:]** - toda a lista (omitindo **i** e **f**).
  - Podemos usar valores negativos para os índices!



Slices

Podemos fatiar (***slice***) a lista:

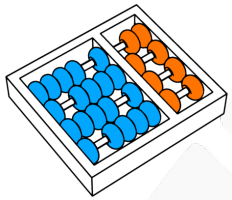
- **lista[i:f]** nos dá os elementos: **lista[i]**, **lista[i + 1]**, . . . , **lista[f - 1]**.
  - **lista[:f]** - até o elemento **f - 1** (omitindo **i**).
  - **lista[i:]** - todos os elementos a partir do **i** (omitindo **f**).
  - **lista[:]** - toda a lista (omitindo **i** e **f**).
  - Podemos usar valores negativos para os índices!
- **lista[i:f:p]** nos dá os elementos **lista[r]**, onde **r = i + k * p** com **k** inteiro não negativo e **r < f** (como no **range**).



Slices

Podemos fatiar (***slice***) a lista:

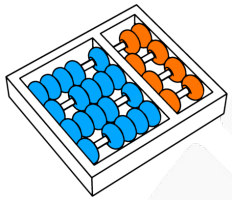
- **lista[i:f]** nos dá os elementos: **lista[i]**, **lista[i + 1]**, . . . , **lista[f - 1]**.
  - **lista[:f]** - até o elemento **f - 1** (omitindo **i**).
  - **lista[i:]** - todos os elementos a partir do **i** (omitindo **f**).
  - **lista[:]** - toda a lista (omitindo **i** e **f**).
  - Podemos usar valores negativos para os índices!
- **lista[i:f:p]** nos dá os elementos **lista[r]**, onde **r = i + k * p** com **k** inteiro não negativo e **r < f** (como no **range**).
- Você pode usar slice para:



Slices

Podemos fatiar (***slice***) a lista:

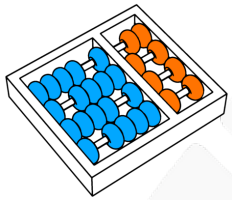
- **lista[i:f]** nos dá os elementos: **lista[i]**, **lista[i + 1]**, . . . , **lista[f - 1]**.
  - **lista[:f]** - até o elemento **f - 1** (omitindo **i**).
  - **lista[i:]** - todos os elementos a partir do **i** (omitindo **f**).
  - **lista[:]** - toda a lista (omitindo **i** e **f**).
  - Podemos usar valores negativos para os índices!
- **lista[i:f:p]** nos dá os elementos **lista[r]**, onde **r = i + k * p** com **k** inteiro não negativo e **r < f** (como no **range**).
- Você pode usar slice para:
  - **copiar** parte da lista (ex: **lista2 = lista1[::2]**).



Slices

Podemos fatiar (***slice***) a lista:

- **lista[i:f]** nos dá os elementos: **lista[i]**, **lista[i + 1]**, . . . , **lista[f - 1]**.
  - **lista[:f]** - até o elemento **f - 1** (omitindo **i**).
  - **lista[i:]** - todos os elementos a partir do **i** (omitindo **f**).
  - **lista[:]** - toda a lista (omitindo **i** e **f**).
  - Podemos usar valores negativos para os índices!
- **lista[i:f:p]** nos dá os elementos **lista[r]**, onde **r = i + k * p** com **k** inteiro não negativo e **r < f** (como no **range**).
- Você pode usar slice para:
  - **copiar** parte da lista (ex: **lista2 = lista1[::2]**).
  - **mudar** valores da lista (ex: **lista[1:10] = ['x']**).



Solução revisitada

```
1  n = int(input("Entre com n: "))
2  # cria uma lista de n + 1 elementos todos True
3  eh_primo = [True] * (n + 1)
4  eh_primo[0:2] = [False] * 2
5  for p in range(2, n + 1):
6      if eh_primo[p]:
7          print(p)
8          # podemos começar em p * p e pulamos de p em p
9          eh_primo[p * p::p] = [False] * len(eh_primo[p * p::p])
```

Exercícios

Listas e repetição

Vamos fazer alguns exercícios?

Exercícios

1. Faça um programa que dado uma lista de números, testa se todos são primos.

Exercícios

1. Faça um programa que dado uma lista de números, testa se todos são primos.
2. Faça um programa que conta quantos primos menores do que n existem.

Exercícios

1. Faça um programa que dado uma lista de números, testa se todos são primos.
2. Faça um programa que conta quantos primos menores do que n existem.
3. Faça um programa que, dado n, imprime um quadrado formado por *'s com n linhas e n colunas.

Exercícios

1. Faça um programa que dado uma lista de números, testa se todos são primos.
2. Faça um programa que conta quantos primos menores do que n existem.
3. Faça um programa que, dado n, imprime um quadrado formado por *'s com n linhas e n colunas.
   - Repita para retângulo.

Exercícios

1. Faça um programa que dado uma lista de números, testa se todos são primos.
2. Faça um programa que conta quantos primos menores do que n existem.
3. Faça um programa que, dado n, imprime um quadrado formado por *'s com n linhas e n colunas.
   - Repita para retângulo.
   - Repita para triângulo retângulo isósceles.

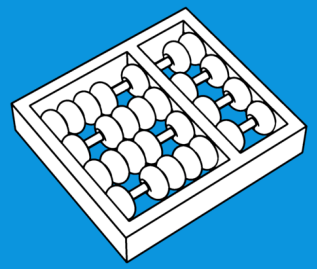
Exercícios

1. Faça um programa que dado uma lista de números, testa se todos são primos.
2. Faça um programa que conta quantos primos menores do que n existem.
3. Faça um programa que, dado n, imprime um quadrado formado por *'s com n linhas e n colunas.
   - Repita para retângulo.
   - Repita para triângulo retângulo isósceles.
4. Um número é perfeito se é igual a soma dos seus divisores (ex: $6 = 1 + 2 + 3$). Faça um programa que dado um número n, imprime todos os números perfeitos menores ou iguais a n.

Listas e repetição. De novo?!

MC102 - Algoritmos e Programação de Computadores

Santiago Valdés Ravelo
https://ic.unicamp.br/~santiago/
ravelo@unicamp.br

03/25

7