MC102 — Comandos de Repetição

Rafael C. S. Schouery
rafael@ic.unicamp.br

Universidade Estadual de Campinas

Atualizado em: 2023-04-04 11:30

Imprimindo números

Queremos imprimir 3 números consecutivos

Imprimindo números

Queremos imprimir 3 números consecutivos

```
1 n = int(input("Entre com n: "))
2 print(n)
3 print(n + 1)
4 print(n + 2)
```

Imprimindo números

Queremos imprimir 3 números consecutivos

```
1 n = int(input("Entre com n: "))
2 print(n)
3 print(n + 1)
4 print(n + 2)
```

E se quisermos 5 números consecutivos?

Imprimindo números

Queremos imprimir 3 números consecutivos

```
1 n = int(input("Entre com n: "))
2 print(n)
3 print(n + 1)
4 print(n + 2)
```

E se quisermos 5 números consecutivos?

```
1 n = int(input("Entre com n: "))
2 print(n)
3 print(n + 1)
4 print(n + 2)
5 print(n + 3)
6 print(n + 4)
```

Imprimindo números

Queremos imprimir 3 números consecutivos

```
1 n = int(input("Entre com n: "))
2 print(n)
3 print(n + 1)
4 print(n + 2)
```

E se quisermos 5 números consecutivos?

```
1 n = int(input("Entre com n: "))
2 print(n)
3 print(n + 1)
4 print(n + 2)
5 print(n + 3)
6 print(n + 4)
```

E se quisermos 100 números consecutivos?

100 números consecutivos

```
1  n = int(input("Entre com n: "))
2  print(n)
3  print(n + 1)
4  print(n + 2)
5  print(n + 3)
6  print(n + 4)
7  print(n + 5)
8  print(n + 6)
9  print(n + 7)
10 print(n + 8)
11 print(n + 9)
12 ... # o ... não é um código Python válido
13 print(n + 99)
```

100 números consecutivos

```
1  n = int(input("Entre com n: "))
2  print(n)
3  print(n + 1)
4  print(n + 2)
5  print(n + 3)
6  print(n + 4)
7  print(n + 5)
8  print(n + 6)
9  print(n + 7)
10 print(n + 8)
11 print(n + 9)
12 ... # o ... não é um código Python válido
13 print(n + 99)
```

Dois problemas:

100 números consecutivos

```
1  n = int(input("Entre com n: "))
2  print(n)
3  print(n + 1)
4  print(n + 2)
5  print(n + 3)
6  print(n + 4)
7  print(n + 5)
8  print(n + 6)
9  print(n + 7)
10 print(n + 8)
11 print(n + 9)
12 ... # o ... não é um código Python válido
13 print(n + 99)
```

Dois problemas:

- Código é repetitivo

100 números consecutivos

```
1  n = int(input("Entre com n: "))
2  print(n)
3  print(n + 1)
4  print(n + 2)
5  print(n + 3)
6  print(n + 4)
7  print(n + 5)
8  print(n + 6)
9  print(n + 7)
10 print(n + 8)
11 print(n + 9)
12 ... # o ... não é um código Python válido
13 print(n + 99)
```

Dois problemas:

- Código é repetitivo
  - DRY (Don't repeat yourself)

100 números consecutivos

```
1  n = int(input("Entre com n: "))
2  print(n)
3  print(n + 1)
4  print(n + 2)
5  print(n + 3)
6  print(n + 4)
7  print(n + 5)
8  print(n + 6)
9  print(n + 7)
10 print(n + 8)
11 print(n + 9)
12 ... # o ... não é um código Python válido
13 print(n + 99)
```

Dois problemas:

- Código é repetitivo
  - DRY (Don't repeat yourself)
- Imprimi um número fixo de números consecutivos

100 números consecutivos

```
1  n = int(input("Entre com n: "))
2  print(n)
3  print(n + 1)
4  print(n + 2)
5  print(n + 3)
6  print(n + 4)
7  print(n + 5)
8  print(n + 6)
9  print(n + 7)
10 print(n + 8)
11 print(n + 9)
12 ... # o ... não é um código Python válido
13 print(n + 99)
```

Dois problemas:

- Código é repetitivo
  - DRY (Don't repeat yourself)
- Imprimi um número fixo de números consecutivos
  - O usuário não pode entrar com a quantidade

Menor número

Dado 2 números, queremos saber qual é o menor

Menor número

Dado 2 números, queremos saber qual é o menor

```
1 a = int(input("Entre com o número 1: "))
2 b = int(input("Entre com o número 2: "))
3 if a <= b:
4     print(a)
5 else:
6     print(b)
```

Menor número

Dado 2 números, queremos saber qual é o menor

```
1 a = int(input("Entre com o número 1: "))
2 b = int(input("Entre com o número 2: "))
3 if a <= b:
4     print(a)
5 else:
6     print(b)
```

Dado 3 números, queremos saber qual é o menor

Menor número

Dado 2 números, queremos saber qual é o menor

```
1 a = int(input("Entre com o número 1: "))
2 b = int(input("Entre com o número 2: "))
3 if a <= b:
4     print(a)
5 else:
6     print(b)
```

Dado 3 números, queremos saber qual é o menor

```
1 a = int(input("Entre com o número 1: "))
2 b = int(input("Entre com o número 2: "))
3 c = int(input("Entre com o número 3: "))
4 if a <= b and a <= c:
5     print(a)
6 elif b < a and b <= c:
7     print(b)
8 else
9     print(c)
```

Menor número

Dado 2 números, queremos saber qual é o menor

```
1 a = int(input("Entre com o número 1: "))
2 b = int(input("Entre com o número 2: "))
3 if a <= b:
4     print(a)
5 else:
6     print(b)
```

Dado 3 números, queremos saber qual é o menor

```
1 a = int(input("Entre com o número 1: "))
2 b = int(input("Entre com o número 2: "))
3 c = int(input("Entre com o número 3: "))
4 if a <= b and a <= c:
5     print(a)
6 elif b < a and b <= c:
7     print(b)
8 else
9     print(c)
```

Dado 100 números, queremos saber qual é o menor

Menor número

Dado 2 números, queremos saber qual é o menor

```
1 a = int(input("Entre com o número 1: "))
2 b = int(input("Entre com o número 2: "))
3 if a <= b:
4     print(a)
5 else:
6     print(b)
```

Dado 3 números, queremos saber qual é o menor

```
1 a = int(input("Entre com o número 1: "))
2 b = int(input("Entre com o número 2: "))
3 c = int(input("Entre com o número 3: "))
4 if a <= b and a <= c:
5     print(a)
6 elif b < a and b <= c:
7     print(b)
8 else
9     print(c)
```

Dado 100 números, queremos saber qual é o menor

- Seguindo o mesmo padrão, o código seria bem longo...

Comandos de Repetição

Felizmente, podemos usar um comando de repetição!

Comandos de Repetição

Felizmente, podemos usar um comando de repetição!

- Repete o mesmo código

Comandos de Repetição

Felizmente, podemos usar um comando de repetição!

- Repete o mesmo código
- Até que algo aconteça!

Pseudocódigo

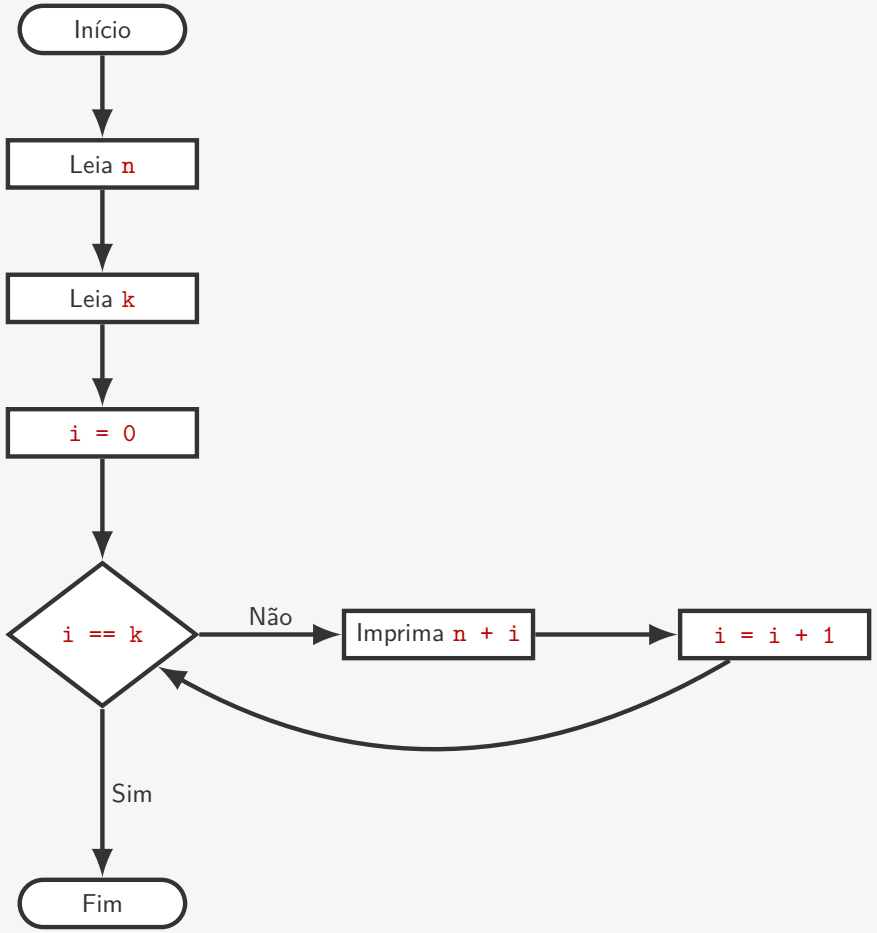
Dado n e k, queremos imprimir os k números consecutivos começando em n

Pseudocódigo

Dado n e k, queremos imprimir os k números consecutivos começando em n

```
1 Leia n
2 Leia k
3 i = 0
4 Enquanto i < k faça
5     Imprima n + i
6     i = i + 1
```

Fluxograma



Em Python

```
1 n = int(input("Entre com n: "))
2 k = int(input("Entre com k: "))
3 i = 0
4 while i < k:
5     print(n + i)
6     i += 1
```

Em Python

```
1 n = int(input("Entre com n: "))
2 k = int(input("Entre com k: "))
3 i = 0
4 while i < k:
5     print(n + i)
6     i += 1
```

- A sintaxe do `while` é similar a do `if`

Em Python

```
1 n = int(input("Entre com n: "))
2 k = int(input("Entre com k: "))
3 i = 0
4 while i < k:
5     print(n + i)
6     i += 1
```

- A sintaxe do `while` é similar a do `if`
  - O `:` indica o começo do bloco

Em Python

```
1 n = int(input("Entre com n: "))
2 k = int(input("Entre com k: "))
3 i = 0
4 while i < k:
5     print(n + i)
6     i += 1
```

- A sintaxe do `while` é similar a do `if`
  - O `:` indica o começo do bloco
  - Que tem que ser indentado

Em Python

```
1 n = int(input("Entre com n: "))
2 k = int(input("Entre com k: "))
3 i = 0
4 while i < k:
5     print(n + i)
6     i += 1
```

- A sintaxe do `while` é similar a do `if`
  - O `:` indica o começo do bloco
  - Que tem que ser indentado
- `x += y` é o mesmo que `x = x + y`

Em Python

```
1 n = int(input("Entre com n: "))
2 k = int(input("Entre com k: "))
3 i = 0
4 while i < k:
5     print(n + i)
6     i += 1
```

- A sintaxe do `while` é similar a do `if`
  - O `:` indica o começo do bloco
  - Que tem que ser indentado
- `x += y` é o mesmo que `x = x + y`
- Existe também

Em Python

```
1 n = int(input("Entre com n: "))
2 k = int(input("Entre com k: "))
3 i = 0
4 while i < k:
5     print(n + i)
6     i += 1
```

- A sintaxe do `while` é similar a do `if`
  - O `:` indica o começo do bloco
  - Que tem que ser indentado
- `x += y` é o mesmo que `x = x + y`
- Existe também
  - `x -= y`,

Em Python

```
1 n = int(input("Entre com n: "))
2 k = int(input("Entre com k: "))
3 i = 0
4 while i < k:
5     print(n + i)
6     i += 1
```

- A sintaxe do `while` é similar a do `if`
  - O `:` indica o começo do bloco
  - Que tem que ser indentado
- `x += y` é o mesmo que `x = x + y`
- Existe também
  - `x -= y`,
  - `x *= y`,

Em Python

```
1 n = int(input("Entre com n: "))
2 k = int(input("Entre com k: "))
3 i = 0
4 while i < k:
5     print(n + i)
6     i += 1
```

- A sintaxe do `while` é similar a do `if`
  - O `:` indica o começo do bloco
  - Que tem que ser indentado
- `x += y` é o mesmo que `x = x + y`
- Existe também
  - `x -= y`,
  - `x *= y`,
  - `x /= y`,

Em Python

```
1 n = int(input("Entre com n: "))
2 k = int(input("Entre com k: "))
3 i = 0
4 while i < k:
5     print(n + i)
6     i += 1
```

- A sintaxe do `while` é similar a do `if`
  - O `:` indica o começo do bloco
  - Que tem que ser indentado
- `x += y` é o mesmo que `x = x + y`
- Existe também
  - `x -= y`,
  - `x *= y`,
  - `x /= y`,
  - `x //= y`,

Em Python

```
1 n = int(input("Entre com n: "))
2 k = int(input("Entre com k: "))
3 i = 0
4 while i < k:
5     print(n + i)
6     i += 1
```

- A sintaxe do `while` é similar a do `if`
  - O `:` indica o começo do bloco
  - Que tem que ser indentado
- `x += y` é o mesmo que `x = x + y`
- Existe também
  - `x -= y`,
  - `x *= y`,
  - `x /= y`,
  - `x //= y`,
  - `x %= y`,

Em Python

```
1 n = int(input("Entre com n: "))
2 k = int(input("Entre com k: "))
3 i = 0
4 while i < k:
5     print(n + i)
6     i += 1
```

- A sintaxe do `while` é similar a do `if`
  - O `:` indica o começo do bloco
  - Que tem que ser indentado
- `x += y` é o mesmo que `x = x + y`
- Existe também
  - `x -= y`,
  - `x *= y`,
  - `x /= y`,
  - `x //= y`,
  - `x %= y`,
  - `x **= y`

Em Python

```
1 n = int(input("Entre com n: "))
2 k = int(input("Entre com k: "))
3 i = 0
4 while i < k:
5     print(n + i)
6     i += 1
```

- A sintaxe do `while` é similar a do `if`
  - O `:` indica o começo do bloco
  - Que tem que ser indentado
- `x += y` é o mesmo que `x = x + y`
- Existe também
  - `x -= y`,
  - `x *= y`,
  - `x /= y`,
  - `x //= y`,
  - `x %= y`,
  - `x **= y`
  - E mais alguns outros...

Bugs

Como saber se o programa está certo?

Bugs

Como saber se o programa está certo?

- Que ele não tem um *bug*?

Bugs

Como saber se o programa está certo?

- Que ele não tem um *bug*?

Podemos executar o programa para alguns valores de entrada...

Bugs

Como saber se o programa está certo?

- Que ele não tem um *bug*?

Podemos executar o programa para alguns valores de entrada...

E se tiver bug, o que fazer?

Livrando-se de bugs: teste de mesa

Uma forma prática é fazer um *teste de mesa*

Livrando-se de bugs: teste de mesa

Uma forma prática é fazer um *teste de mesa*

- simular o programa usando papel e lápis

Livrando-se de bugs: teste de mesa

Uma forma prática é fazer um *teste de mesa*

- simular o programa usando papel e lápis
- para alguns valores de entrada

Livrando-se de bugs: teste de mesa

Uma forma prática é fazer um *teste de mesa*

- simular o programa usando papel e lápis
- para alguns valores de entrada

Podemos olhar também para alguns casos mais críticos:

Livrando-se de bugs: teste de mesa

Uma forma prática é fazer um *teste de mesa*

- simular o programa usando papel e lápis
- para alguns valores de entrada

Podemos olhar também para alguns casos mais críticos:

- E se k ou n for zero?

Livrando-se de bugs: teste de mesa

Uma forma prática é fazer um *teste de mesa*

- simular o programa usando papel e lápis
- para alguns valores de entrada

Podemos olhar também para alguns casos mais críticos:

- E se k ou n for zero?
- E se k ou n for negativo?

Exemplo de teste de mesa

Exemplo de teste de mesa

```
1 n = int(input("Entre com n: "))
2 k = int(input("Entre com k: "))
3 i = 0
4 while i < k:
5     print(n + i)
6     i += 1
```

Exemplo de teste de mesa

```
1 n = int(input("Entre com n: "))
2 k = int(input("Entre com k: "))
3 i = 0
4 while i < k:
5     print(n + i)
6     i += 1
```

Exemplo de teste de mesa

linha	n	k	i	obs

```
1 n = int(input("Entre com n: "))
2 k = int(input("Entre com k: "))
3 i = 0
4 while i < k:
5     print(n + i)
6     i += 1
```

Exemplo de teste de mesa

linha	n	k	i	obs
1	2	?	?	

```
1 n = int(input("Entre com n: "))
2 k = int(input("Entre com k: "))
3 i = 0
4 while i < k:
5     print(n + i)
6     i += 1
```

Exemplo de teste de mesa

linha	n	k	i	obs
1	2	?	?	
2	2	3	?	

```
1 n = int(input("Entre com n: "))
2 k = int(input("Entre com k: "))
3 i = 0
4 while i < k:
5     print(n + i)
6     i += 1
```

Exemplo de teste de mesa

linha	n	k	i	obs
1	2	?	?	
2	2	3	?	
3	2	3	0	

```
1 n = int(input("Entre com n: "))
2 k = int(input("Entre com k: "))
3 i = 0
4 while i < k:
5     print(n + i)
6     i += 1
```

Exemplo de teste de mesa

```
1 n = int(input("Entre com n: "))
2 k = int(input("Entre com k: "))
3 i = 0
4 while i < k:
5     print(n + i)
6     i += 1
```

linha	n	k	i	obs
1	2	?	?	
2	2	3	?	
3	2	3	0	
4	2	3	0	$0 < 3$

Exemplo de teste de mesa

```
1 n = int(input("Entre com n: "))
2 k = int(input("Entre com k: "))
3 i = 0
4 while i < k:
5     print(n + i)
6     i += 1
```

linha	n	k	i	obs
1	2	?	?	
2	2	3	?	
3	2	3	0	
4	2	3	0	$0 < 3$
5	2	3	0	imprimi 2

Exemplo de teste de mesa

```
1 n = int(input("Entre com n: "))
2 k = int(input("Entre com k: "))
3 i = 0
4 while i < k:
5     print(n + i)
6     i += 1
```

linha	n	k	i	obs
1	2	?	?	
2	2	3	?	
3	2	3	0	
4	2	3	0	$0 < 3$
5	2	3	0	imprimi 2
6	2	3	1	

Exemplo de teste de mesa

```
1 n = int(input("Entre com n: "))
2 k = int(input("Entre com k: "))
3 i = 0
4 while i < k:
5     print(n + i)
6     i += 1
```

linha	n	k	i	obs
1	2	?	?	
2	2	3	?	
3	2	3	0	
4	2	3	0	$0 < 3$
5	2	3	0	imprimi 2
6	2	3	1	
4	2	3	1	$1 < 3$

Exemplo de teste de mesa

```
1 n = int(input("Entre com n: "))
2 k = int(input("Entre com k: "))
3 i = 0
4 while i < k:
5     print(n + i)
6     i += 1
```

linha	n	k	i	obs
1	2	?	?	
2	2	3	?	
3	2	3	0	
4	2	3	0	$0 < 3$
5	2	3	0	imprimi 2
6	2	3	1	
4	2	3	1	$1 < 3$
5	2	3	1	imprimi 3

Exemplo de teste de mesa

```
1 n = int(input("Entre com n: "))
2 k = int(input("Entre com k: "))
3 i = 0
4 while i < k:
5     print(n + i)
6     i += 1
```

linha	n	k	i	obs
1	2	?	?	
2	2	3	?	
3	2	3	0	
4	2	3	0	$0 < 3$
5	2	3	0	imprimi 2
6	2	3	1	
4	2	3	1	$1 < 3$
5	2	3	1	imprimi 3
6	2	3	2	

Exemplo de teste de mesa

```
1 n = int(input("Entre com n: "))
2 k = int(input("Entre com k: "))
3 i = 0
4 while i < k:
5     print(n + i)
6     i += 1
```

linha	n	k	i	obs
1	2	?	?	
2	2	3	?	
3	2	3	0	
4	2	3	0	$0 < 3$
5	2	3	0	imprimi 2
6	2	3	1	
4	2	3	1	$1 < 3$
5	2	3	1	imprimi 3
6	2	3	2	
4	2	3	2	$2 < 3$

Exemplo de teste de mesa

```
1 n = int(input("Entre com n: "))
2 k = int(input("Entre com k: "))
3 i = 0
4 while i < k:
5     print(n + i)
6     i += 1
```

linha	n	k	i	obs
1	2	?	?	
2	2	3	?	
3	2	3	0	
4	2	3	0	$0 < 3$
5	2	3	0	imprimi 2
6	2	3	1	
4	2	3	1	$1 < 3$
5	2	3	1	imprimi 3
6	2	3	2	
4	2	3	2	$2 < 3$
5	2	3	2	imprimi 4

Exemplo de teste de mesa

```
1 n = int(input("Entre com n: "))
2 k = int(input("Entre com k: "))
3 i = 0
4 while i < k:
5     print(n + i)
6     i += 1
```

linha	n	k	i	obs
1	2	?	?	
2	2	3	?	
3	2	3	0	
4	2	3	0	$0 < 3$
5	2	3	0	imprimi 2
6	2	3	1	
4	2	3	1	$1 < 3$
5	2	3	1	imprimi 3
6	2	3	2	
4	2	3	2	$2 < 3$
5	2	3	2	imprimi 4
6	2	3	3	

Exemplo de teste de mesa

```
1 n = int(input("Entre com n: "))
2 k = int(input("Entre com k: "))
3 i = 0
4 while i < k:
5     print(n + i)
6     i += 1
```

linha	n	k	i	obs
1	2	?	?	
2	2	3	?	
3	2	3	0	
4	2	3	0	$0 < 3$
5	2	3	0	imprimi 2
6	2	3	1	
4	2	3	1	$1 < 3$
5	2	3	1	imprimi 3
6	2	3	2	
4	2	3	2	$2 < 3$
5	2	3	2	imprimi 4
6	2	3	3	
4	2	3	3	$3 == 3$

Livrando-se de bugs: depurador

Uma maneira parecida de fazer isso é usando um *debugger*

Livrando-se de bugs: depurador

Uma maneira parecida de fazer isso é usando um *debugger*

- Um programa que permite testar o seu programa!

Livrando-se de bugs: depurador

Uma maneira parecida de fazer isso é usando um *debugger*

- Um programa que permite testar o seu programa!
- Executa o programa passo a passo

Livrando-se de bugs: depurador

Uma maneira parecida de fazer isso é usando um *debugger*

- Um programa que permite testar o seu programa!
- Executa o programa passo a passo
- E mostra os valores das variáveis

Livrando-se de bugs: depurador

Uma maneira parecida de fazer isso é usando um *debugger*

- Um programa que permite testar o seu programa!
- Executa o programa passo a passo
- E mostra os valores das variáveis
- Entre várias outras funcionalidades

Livrando-se de bugs: depurador

Uma maneira parecida de fazer isso é usando um *debugger*

- Um programa que permite testar o seu programa!
- Executa o programa passo a passo
- E mostra os valores das variáveis
- Entre várias outras funcionalidades

Exemplo no VSCode!

Livrando-se de bugs: lendo o código

Outra forma de se livrar de bugs é lendo o código

Livrando-se de bugs: lendo o código

Outra forma de se livrar de bugs é lendo o código

- Ler seu código com atenção

Livrando-se de bugs: lendo o código

Outra forma de se livrar de bugs é lendo o código

- Ler seu código com atenção
- Pensando o que cada coisa faz

Livrando-se de bugs: lendo o código

Outra forma de se livrar de bugs é lendo o código

- Ler seu código com atenção
- Pensando o que cada coisa faz

Costuma ser mais rápido do que os outros métodos

Livrando-se de bugs: lendo o código

Outra forma de se livrar de bugs é lendo o código

- Ler seu código com atenção
- Pensando o que cada coisa faz

Costuma ser mais rápido do que os outros métodos

- Quando o bug é claro, pelo menos...

Voltando ao código

Vimos esse código:

Voltando ao código

Vimos esse código:

```
1 n = int(input("Entre com n: "))
2 k = int(input("Entre com k: "))
3 i = 0
4 while i < k:
5     print(n + i)
6     i += 1
```

Voltando ao código

Vimos esse código:

```
1 n = int(input("Entre com n: "))
2 k = int(input("Entre com k: "))
3 i = 0
4 while i < k:
5     print(n + i)
6     i += 1
```

Mas existem outras formas de fazer!

Voltando ao código

Vimos esse código:

```
1 n = int(input("Entre com n: "))
2 k = int(input("Entre com k: "))
3 i = 0
4 while i < k:
5     print(n + i)
6     i += 1
```

Mas existem outras formas de fazer!

```
1 n = int(input("Entre com n: "))
2 k = int(input("Entre com k: "))
3 atual = n
4 while atual < n + k:
5     print(atual)
6     atual += 1
```

Exercício

Relembrando: uma Progressão Aritmética (PA)

Exercício

Relembrando: uma Progressão Aritmética (PA)

- é uma sequência de números $(a_1, a_2, \ldots, a_n)$

Exercício

Relembrando: uma Progressão Aritmética (PA)

- é uma sequência de números $(a_1, a_2, \ldots, a_n)$
- onde existe um número r tal que

Exercício

Relembrando: uma Progressão Aritmética (PA)

- é uma sequência de números $(a_1, a_2, \ldots, a_n)$
- onde existe um número r tal que
- $a_{i+1} = a_i + r$

Exercício

Relembrando: uma Progressão Aritmética (PA)

- é uma sequência de números $(a_1, a_2, \ldots, a_n)$
- onde existe um número r tal que
- $a_{i+1} = a_i + r$
- para todo $1 \leq i < n$

Exercício

Relembrando: uma Progressão Aritmética (PA)

- é uma sequência de números $(a_1, a_2, \ldots, a_n)$
- onde existe um número r tal que
- $a_{i+1} = a_i + r$
- para todo $1 \leq i < n$

A soma S da progressão aritmética é

Exercício

Relembrando: uma Progressão Aritmética (PA)

- é uma sequência de números $(a_1, a_2, \ldots, a_n)$
- onde existe um número r tal que
- $a_{i+1} = a_i + r$
- para todo $1 \leq i < n$

A soma S da progressão aritmética é

$$S = \frac{n(a_1 + a_n)}{2} = a_1 n + \frac{n(n-1)r}{2}$$

Exercício

Relembrando: uma Progressão Aritmética (PA)

- é uma sequência de números $(a_1, a_2, \ldots, a_n)$
- onde existe um número r tal que
- $a_{i+1} = a_i + r$
- para todo $1 \leq i < n$

A soma S da progressão aritmética é

$$S = \frac{n(a_1 + a_n)}{2} = a_1 n + \frac{n(n-1)r}{2}$$

Ou é o que dizem!

Exercício

Relembrando: uma Progressão Aritmética (PA)

- é uma sequência de números $(a_1, a_2, \ldots, a_n)$
- onde existe um número r tal que
- $a_{i+1} = a_i + r$
- para todo $1 \leq i < n$

A soma S da progressão aritmética é

$$S = \frac{n(a_1 + a_n)}{2} = a_1 n + \frac{n(n-1)r}{2}$$

Ou é o que dizem!

- Vamos fazer um código para ver se fórmula está correta!

Exercício

Relembrando: uma Progressão Aritmética (PA)

- é uma sequência de números $(a_1, a_2, \ldots, a_n)$
- onde existe um número r tal que
- $a_{i+1} = a_i + r$
- para todo $1 \leq i < n$

A soma S da progressão aritmética é

$$S = \frac{n(a_1 + a_n)}{2} = a_1 n + \frac{n(n-1)r}{2}$$

Ou é o que dizem!

- Vamos fazer um código para ver se fórmula está correta!
- E vamos simular esse programa no debbuger!

Possível solução

```
1  a_1 = int(input("Entre com a_1: "))
2  n = int(input("Entre com n: "))
3  r = int(input("Entre com r: "))
4
5  esperado = a_1 * n + n * (n - 1) * r / 2
6
7  i = 1
8  soma = 0
9  atual = a_1
10 while i <= n:
11     soma += atual
12     atual += r
13     i = i + 1
14
15 if soma != esperado:
16     print("Fórmula incorreta!")
17     print("Esperado: ", esperado)
18     print("Obtido: ", soma)
19 else:
20     print("Fórmula correta!")
```

Possível solução

```
1  a_1 = int(input("Entre com a_1: "))
2  n = int(input("Entre com n: "))
3  r = int(input("Entre com r: "))
4
5  esperado = a_1 * n + n * (n - 1) * r / 2
6
7  i = 1
8  soma = 0
9  atual = a_1
10 while i <= n:
11     soma += atual
12     atual += r
13     i = i + 1
14
15 if soma != esperado:
16     print("Fórmula incorreta!")
17     print("Esperado: ", esperado)
18     print("Obtido: ", soma)
19 else:
20     print("Fórmula correta!")
```

- A variável `i` está *contando* até `n`

Possível solução

```
1  a_1 = int(input("Entre com a_1: "))
2  n = int(input("Entre com n: "))
3  r = int(input("Entre com r: "))
4
5  esperado = a_1 * n + n * (n - 1) * r / 2
6
7  i = 1
8  soma = 0
9  atual = a_1
10 while i <= n:
11     soma += atual
12     atual += r
13     i = i + 1
14
15 if soma != esperado:
16     print("Fórmula incorreta!")
17     print("Esperado: ", esperado)
18     print("Obtido: ", soma)
19 else:
20     print("Fórmula correta!")
```

- A variável `i` está *contando* até `n`
- A variável `soma` está *acumulando* o resultado

Exercício

Relembrando: um número inteiro positivo p é primo se seus divisores positivos são apenas 1 e p

Exercício

Relembrando: um número inteiro positivo p é primo se seus divisores positivos são apenas 1 e p

Escreva um programa que, dado p, diz se p é primo ou não.

Possível solução

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

Possível solução

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

- `primo` *indica* se `p` é primo ou não

Possível solução

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

- `primo` *indica* se `p` é primo ou não

Vamos depurar no VSCode

Possível solução

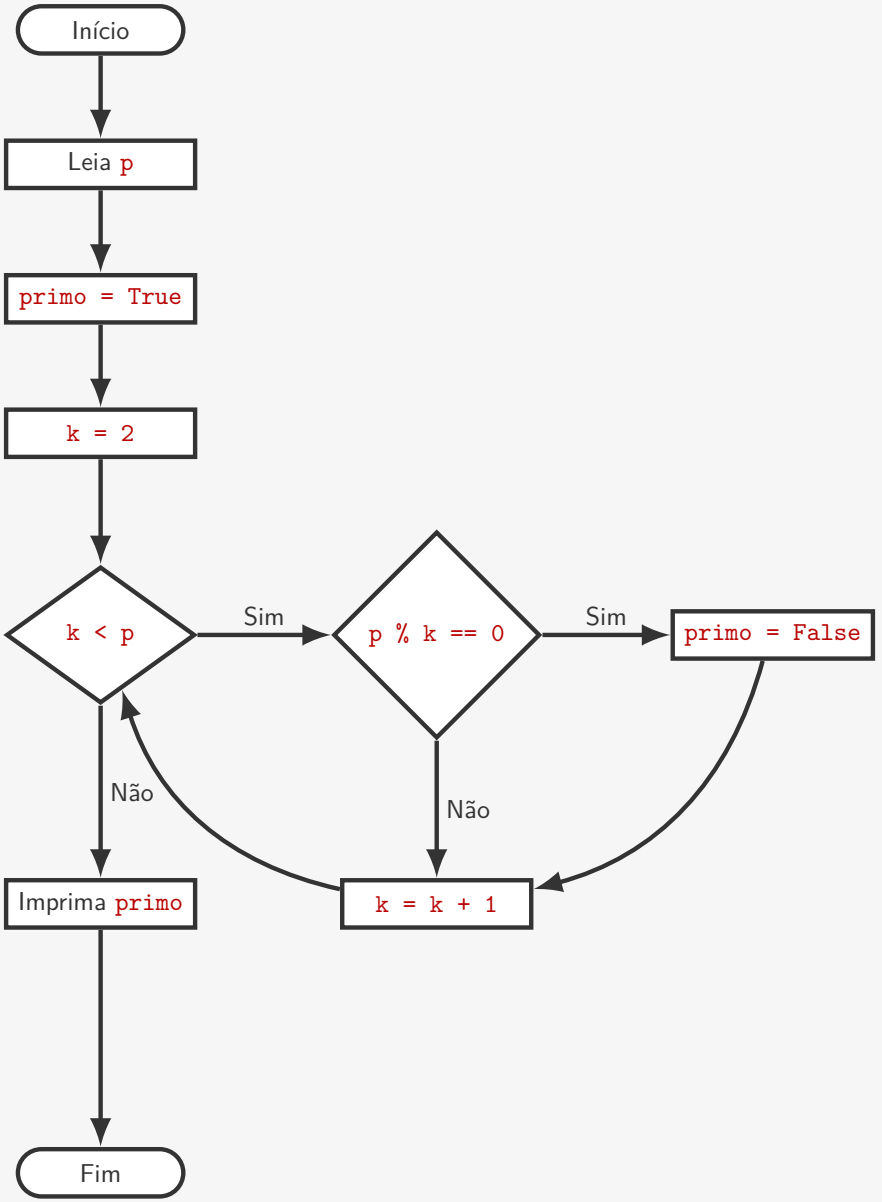
```
1 p = int(input("Entre com p: "))
2 primo = True
3 k = 2
4 while k < p:
5     if p % k == 0:
6         primo = False
7     k += 1
8 print(p >= 2 and primo)
```

- `primo` *indica* se `p` é primo ou não

Vamos depurar no VSCode

- tente também fazer um teste de mesa!

Fluxograma


Início
Leia p
primo = True
k = 2
k < p
Sim
p % k == 0
Sim
primo = False
Não
Não
Imprima primo
k = k + 1
Fim

Otimizando

Podemos melhor o código:

Otimizando

Podemos melhor o código:

- Testando k apenas até $\sqrt{\texttt{p}}$

Otimizando

Podemos melhor o código:

- Testando k apenas até $\sqrt{\texttt{p}}$
  - Porque se p tem um divisor maior ou igual a $\sqrt{p}$,

Otimizando

Podemos melhor o código:

- Testando k apenas até $\sqrt{\texttt{p}}$
  - Porque se p tem um divisor maior ou igual a $\sqrt{p}$,
  - então p tem um divisor menor ou igual a $\sqrt{p}$

Otimizando

Podemos melhor o código:

- Testando k apenas até $\sqrt{\texttt{p}}$
  - Porque se p tem um divisor maior ou igual a $\sqrt{p}$,
  - então p tem um divisor menor ou igual a $\sqrt{p}$
- Saindo do laço quando achamos um divisor

Otimizando

Podemos melhor o código:

- Testando k apenas até $\sqrt{\texttt{p}}$
  - Porque se p tem um divisor maior ou igual a $\sqrt{p}$,
  - então p tem um divisor menor ou igual a $\sqrt{p}$
- Saindo do laço quando achamos um divisor
  - Porque já sabemos que o número não é primo

Otimizando

Podemos melhor o código:

- Testando k apenas até $\sqrt{\mathrm{p}}$
  - Porque se p tem um divisor maior ou igual a $\sqrt{p}$,
  - então p tem um divisor menor ou igual a $\sqrt{p}$
- Saindo do laço quando achamos um divisor
  - Porque já sabemos que o número não é primo

```
1 p = int(input("Entre com p: "))
2 primo = True
3 k = 2
4 # é melhor testar k * k <= p do que k <= p ** (1/2)
5 while k * k <= p and primo:
6     if p % k == 0:
7         primo = False
8     k += 1
9 print(p >= 2 and primo)
```

Otimizando

Podemos melhor o código:

- Testando k apenas até $\sqrt{\mathrm{p}}$
  - Porque se p tem um divisor maior ou igual a $\sqrt{p}$,
  - então p tem um divisor menor ou igual a $\sqrt{p}$
- Saindo do laço quando achamos um divisor
  - Porque já sabemos que o número não é primo

```
1 p = int(input("Entre com p: "))
2 primo = True
3 k = 2
4 # é melhor testar k * k <= p do que k <= p ** (1/2)
5 while k * k <= p and primo:
6     if p % k == 0:
7         primo = False
8     k += 1
9 print(p >= 2 and primo)
```

Faz diferença?

Otimizando

Podemos melhor o código:

- Testando k apenas até $\sqrt{\mathrm{p}}$
  - Porque se p tem um divisor maior ou igual a $\sqrt{p}$,
  - então p tem um divisor menor ou igual a $\sqrt{p}$
- Saindo do laço quando achamos um divisor
  - Porque já sabemos que o número não é primo

```
1 p = int(input("Entre com p: "))
2 primo = True
3 k = 2
4 # é melhor testar k * k <= p do que k <= p ** (1/2)
5 while k * k <= p and primo:
6     if p % k == 0:
7         primo = False
8     k += 1
9 print(p >= 2 and primo)
```

Faz diferença?

- $p = 27644437$: 5.32s vs. 0.06s (e é primo)

Exercícios

1. Leia uma sequência de números e imprima a soma

Exercícios

1. Leia uma sequência de números e imprima a soma
2. Leia uma sequência de números e imprima o menor

Exercícios

1. Leia uma sequência de números e imprima a soma
2. Leia uma sequência de números e imprima o menor
3. Leia uma sequência de números e veja se todos são pares

Exercícios

1. Leia uma sequência de números e imprima a soma
2. Leia uma sequência de números e imprima o menor
3. Leia uma sequência de números e veja se todos são pares
4. Leia uma sequência de números e conte quantos são pares

Exercícios

1. Leia uma sequência de números e imprima a soma
2. Leia uma sequência de números e imprima o menor
3. Leia uma sequência de números e veja se todos são pares
4. Leia uma sequência de números e conte quantos são pares
5. Dado um número, imprima a sua decomposição em números primos

Solução

Dado um número, imprima a sua decomposição em números primos

Solução

Dado um número, imprima a sua decomposição em números primos

```
1 n = int(input("Entre com o n: "))
```

Solução

Dado um número, imprima a sua decomposição em números primos

```
1 n = int(input("Entre com o n: "))
```

Solução

Dado um número, imprima a sua decomposição em números primos

```
1 n = int(input("Entre com o n: "))
2 d = 2
```

Solução

Dado um número, imprima a sua decomposição em números primos

```
1 n = int(input("Entre com o n: "))
2 d = 2
3 while n != 1:
```

Solução

Dado um número, imprima a sua decomposição em números primos

```
1 n = int(input("Entre com o n: "))
2 d = 2
3 while n != 1:
4     if n % d == 0:
```

Solução

Dado um número, imprima a sua decomposição em números primos

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

Solução

Dado um número, imprima a sua decomposição em números primos

```
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)
```

Solução

Dado um número, imprima a sua decomposição em números primos

```
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:
```

Solução

Dado um número, imprima a sua decomposição em números primos

```
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
```

Solução

Dado um número, imprima a sua decomposição em números primos

```
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
```

Vamos simular!

Armazenando dados

O que queremos:

Armazenando dados

O que queremos:

- Ter fácil acesso aos dados

Armazenando dados

O que queremos:

- Ter fácil acesso aos dados
- Para não ter tantas variáveis

Armazenando dados

O que queremos:

- Ter fácil acesso aos dados
- Para não ter tantas variáveis
- Para ter um código mais simples

Armazenando dados

O que queremos:

- Ter fácil acesso aos dados
- Para não ter tantas variáveis
- Para ter um código mais simples

Para tanto, usaremos listas

Armazenando dados

O que queremos:

- Ter fácil acesso aos dados
- Para não ter tantas variáveis
- Para ter um código mais simples

Para tanto, usaremos listas

- E, futuramente, outras formas de acesso

Listas

Lista (`list`) é um tipo do Python

Listas

Lista (`list`) é um tipo do Python

- Permite armazenar uma grande quantidade de dados

Listas

Lista (`list`) é um tipo do Python

- Permite armazenar uma grande quantidade de dados
  - Tanto quanto você queira...

Listas

Lista (`list`) é um tipo do Python

- Permite armazenar uma grande quantidade de dados
  - Tanto quanto você queira...
  - Claro, até o limite de memória do computador

Listas

Lista (`list`) é um tipo do Python

- Permite armazenar uma grande quantidade de dados
  - Tanto quanto você queira...
  - Claro, até o limite de memória do computador

- Permite acessar os dados usando um *índice*

Listas

Lista (`list`) é um tipo do Python

- Permite armazenar uma grande quantidade de dados
  - Tanto quanto você queira...
  - Claro, até o limite de memória do computador

- Permite acessar os dados usando um *índice*
  - Ex: `lista[0]`, `lista[1]`, ...

Listas

Lista (`list`) é um tipo do Python

- Permite armazenar uma grande quantidade de dados
  - Tanto quanto você queira...
  - Claro, até o limite de memória do computador

- Permite acessar os dados usando um *índice*
  - Ex: `lista[0]`, `lista[1]`, ...
  - Tanto para escrita quanto para leitura

Listas

Lista (`list`) é um tipo do Python

- Permite armazenar uma grande quantidade de dados
  - Tanto quanto você queira...
  - Claro, até o limite de memória do computador

- Permite acessar os dados usando um *índice*
  - Ex: `lista[0]`, `lista[1]`, ...
  - Tanto para escrita quanto para leitura
  - O índice começa em zero

Criando Listas e Adicionando Itens

Criando uma lista vazia (sem nada armazenado):

Criando Listas e Adicionando Itens

Criando uma lista vazia (sem nada armazenado):

- `lista = []` (mais usado)

Criando Listas e Adicionando Itens

Criando uma lista vazia (sem nada armazenado):

- `lista = []` (mais usado)
- `lista = list()`

Criando Listas e Adicionando Itens

Criando uma lista vazia (sem nada armazenado):

- `lista = []` (mais usado)
- `lista = list()`

Criando uma lista com conteúdo:

Criando Listas e Adicionando Itens

Criando uma lista vazia (sem nada armazenado):

- `lista = []` (mais usado)
- `lista = list()`

Criando uma lista com conteúdo:

- `lista = [1, 7, 2, 2, 15]`

Criando Listas e Adicionando Itens

Criando uma lista vazia (sem nada armazenado):

- `lista = []` (mais usado)
- `lista = list()`

Criando uma lista com conteúdo:

- `lista = [1, 7, 2, 2, 15]`
- `lista = ["ana", "joão", "pedro"]`

Criando Listas e Adicionando Itens

Criando uma lista vazia (sem nada armazenado):

- `lista = []` (mais usado)
- `lista = list()`

Criando uma lista com conteúdo:

- `lista = [1, 7, 2, 2, 15]`
- `lista = ["ana", "joão", "pedro"]`
- `lista = [1.3, 7.5, -2.1]`

Criando Listas e Adicionando Itens

Criando uma lista vazia (sem nada armazenado):

- `lista = []` (mais usado)
- `lista = list()`

Criando uma lista com conteúdo:

- `lista = [1, 7, 2, 2, 15]`
- `lista = ["ana", "joão", "pedro"]`
- `lista = [1.3, 7.5, -2.1]`
- `lista = [x, y, z]` ⟵ Não é uma lista de variáveis!

Criando Listas e Adicionando Itens

Criando uma lista vazia (sem nada armazenado):

- `lista = []` (mais usado)
- `lista = list()`

Criando uma lista com conteúdo:

- `lista = [1, 7, 2, 2, 15]`
- `lista = ["ana", "joão", "pedro"]`
- `lista = [1.3, 7.5, -2.1]`
- `lista = [x, y, z]` ⟵ Não é uma lista de variáveis!
- `lista = [1, "mc102", 3.7]`

Criando Listas e Adicionando Itens

Criando uma lista vazia (sem nada armazenado):

- `lista = []` (mais usado)
- `lista = list()`

Criando uma lista com conteúdo:

- `lista = [1, 7, 2, 2, 15]`
- `lista = ["ana", "joão", "pedro"]`
- `lista = [1.3, 7.5, -2.1]`
- `lista = [x, y, z]` ⟵ Não é uma lista de variáveis!
- `lista = [1, "mc102", 3.7]`

`lista.append(x)`

Criando Listas e Adicionando Itens

Criando uma lista vazia (sem nada armazenado):

- `lista = []` (mais usado)
- `lista = list()`

Criando uma lista com conteúdo:

- `lista = [1, 7, 2, 2, 15]`
- `lista = ["ana", "joão", "pedro"]`
- `lista = [1.3, 7.5, -2.1]`
- `lista = [x, y, z]` ⟵ Não é uma lista de variáveis!
- `lista = [1, "mc102", 3.7]`

`lista.append(x)`

- Insere o valor `x` no final da lista

Criando Listas e Adicionando Itens

Criando uma lista vazia (sem nada armazenado):

- `lista = []` (mais usado)
- `lista = list()`

Criando uma lista com conteúdo:

- `lista = [1, 7, 2, 2, 15]`
- `lista = ["ana", "joão", "pedro"]`
- `lista = [1.3, 7.5, -2.1]`
- `lista = [x, y, z]` ⟵ Não é uma lista de variáveis!
- `lista = [1, "mc102", 3.7]`

`lista.append(x)`

- Insere o valor `x` no final da lista
  - `x` pode ser variável, constante ou uma expressão

Exemplo

```
1 >>> l = []
```

Exemplo

```
1 >>> l = []
2 >>> l
```

Exemplo

```
1 >>> l = []
2 >>> l
3 []
```

Exemplo

```
1 >>> l = []
2 >>> l
3 []
4 >>> type(l)
```

Exemplo

```
1 >>> l = []
2 >>> l
3 []
4 >>> type(l)
5 <class 'list'>
```

Exemplo

```
1 >>> l = []
2 >>> l
3 []
4 >>> type(l)
5 <class 'list'>
6 >>> l.append("Ana")
```

Exemplo

```
1 >>> l = []
2 >>> l
3 []
4 >>> type(l)
5 <class 'list'>
6 >>> l.append("Ana")
7 >>> l.append("Beto")
```

Exemplo

```
>>> l = []
>>> l
[]
>>> type(l)
<class 'list'>
>>> l.append("Ana")
>>> l.append("Beto")
>>> l.append("Carlos")
```

Exemplo

```
1 >>> l = []
2 >>> l
3 []
4 >>> type(l)
5 <class 'list'>
6 >>> l.append("Ana")
7 >>> l.append("Beto")
8 >>> l.append("Carlos")
9 >>> l
```

Exemplo

```
>>> l = []
>>> l
[]
>>> type(l)
<class 'list'>
>>> l.append("Ana")
>>> l.append("Beto")
>>> l.append("Carlos")
>>> l
['Ana', 'Beto', 'Carlos']
```

Exemplo

```
>>> l = []
>>> l
[]
>>> type(l)
<class 'list'>
>>> l.append("Ana")
>>> l.append("Beto")
>>> l.append("Carlos")
>>> l
['Ana', 'Beto', 'Carlos']
>>> l[0]
```

Exemplo

```
>>> l = []
>>> l
[]
>>> type(l)
<class 'list'>
>>> l.append("Ana")
>>> l.append("Beto")
>>> l.append("Carlos")
>>> l
['Ana', 'Beto', 'Carlos']
>>> l[0]
'Ana'
```

Exemplo

```
1  >>> l = []
2  >>> l
3  []
4  >>> type(l)
5  <class 'list'>
6  >>> l.append("Ana")
7  >>> l.append("Beto")
8  >>> l.append("Carlos")
9  >>> l
10 ['Ana', 'Beto', 'Carlos']
11 >>> l[0]
12 'Ana'
13 >>> l[1]
```

Exemplo

```
>>> l = []
>>> l
[]
>>> type(l)
<class 'list'>
>>> l.append("Ana")
>>> l.append("Beto")
>>> l.append("Carlos")
>>> l
['Ana', 'Beto', 'Carlos']
>>> l[0]
'Ana'
>>> l[1]
'Beto'
```

Exemplo

```
>>> l = []
>>> l
[]
>>> type(l)
<class 'list'>
>>> l.append("Ana")
>>> l.append("Beto")
>>> l.append("Carlos")
>>> l
['Ana', 'Beto', 'Carlos']
>>> l[0]
'Ana'
>>> l[1]
'Beto'
>>> l[2]
```

Exemplo

```
>>> l = []
>>> l
[]
>>> type(l)
<class 'list'>
>>> l.append("Ana")
>>> l.append("Beto")
>>> l.append("Carlos")
>>> l
['Ana', 'Beto', 'Carlos']
>>> l[0]
'Ana'
>>> l[1]
'Beto'
>>> l[2]
'Carlos'
```

Exemplo

```
>>> l = []
>>> l
[]
>>> type(l)
<class 'list'>
>>> l.append("Ana")
>>> l.append("Beto")
>>> l.append("Carlos")
>>> l
['Ana', 'Beto', 'Carlos']
>>> l[0]
'Ana'
>>> l[1]
'Beto'
>>> l[2]
'Carlos'
>>> l[3]
```

Exemplo

```
>>> l = []
>>> l
[]
>>> type(l)
<class 'list'>
>>> l.append("Ana")
>>> l.append("Beto")
>>> l.append("Carlos")
>>> l
['Ana', 'Beto', 'Carlos']
>>> l[0]
'Ana'
>>> l[1]
'Beto'
>>> l[2]
'Carlos'
>>> l[3]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

Exemplo

```
>>> l = []
>>> l
[]
>>> type(l)
<class 'list'>
>>> l.append("Ana")
>>> l.append("Beto")
>>> l.append("Carlos")
>>> l
['Ana', 'Beto', 'Carlos']
>>> l[0]
'Ana'
>>> l[1]
'Beto'
>>> l[2]
'Carlos'
>>> l[3]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
>>> l[1] = 'Roberto'
```

Exemplo

```
>>> l = []
>>> l
[]
>>> type(l)
<class 'list'>
>>> l.append("Ana")
>>> l.append("Beto")
>>> l.append("Carlos")
>>> l
['Ana', 'Beto', 'Carlos']
>>> l[0]
'Ana'
>>> l[1]
'Beto'
>>> l[2]
'Carlos'
>>> l[3]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
>>> l[1] = 'Roberto'
>>> l
```

Exemplo

```
>>> l = []
>>> l
[]
>>> type(l)
<class 'list'>
>>> l.append("Ana")
>>> l.append("Beto")
>>> l.append("Carlos")
>>> l
['Ana', 'Beto', 'Carlos']
>>> l[0]
'Ana'
>>> l[1]
'Beto'
>>> l[2]
'Carlos'
>>> l[3]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
>>> l[1] = 'Roberto'
>>> l
['Ana', 'Roberto', 'Carlos']
```

Lendo uma lista de nomes

Se tivermos a quantidade `n` de nomes, podemos fazer:

Lendo uma lista de nomes

Se tivermos a quantidade `n` de nomes, podemos fazer:

```
1 n = int(input("Entre com a quantidade de nomes: "))
```

Lendo uma lista de nomes

Se tivermos a quantidade `n` de nomes, podemos fazer:

```
1 n = int(input("Entre com a quantidade de nomes: "))
```

Lendo uma lista de nomes

Se tivermos a quantidade `n` de nomes, podemos fazer:

```
1 n = int(input("Entre com a quantidade de nomes: "))
2 l = []
```

Lendo uma lista de nomes

Se tivermos a quantidade `n` de nomes, podemos fazer:

```
1 n = int(input("Entre com a quantidade de nomes: "))
2 l = []
3 i = 0
```

Lendo uma lista de nomes

Se tivermos a quantidade `n` de nomes, podemos fazer:

```
1 n = int(input("Entre com a quantidade de nomes: "))
2 l = []
3 i = 0
4 while i < n:
```

Lendo uma lista de nomes

Se tivermos a quantidade `n` de nomes, podemos fazer:

```
1 n = int(input("Entre com a quantidade de nomes: "))
2 l = []
3 i = 0
4 while i < n:
5     nome = input("Entre com o nome: ")
```

Lendo uma lista de nomes

Se tivermos a quantidade `n` de nomes, podemos fazer:

```
1 n = int(input("Entre com a quantidade de nomes: "))
2 l = []
3 i = 0
4 while i < n:
5     nome = input("Entre com o nome: ")
6     l.append(nome)
```

Lendo uma lista de nomes

Se tivermos a quantidade `n` de nomes, podemos fazer:

```
1 n = int(input("Entre com a quantidade de nomes: "))
2 l = []
3 i = 0
4 while i < n:
5     nome = input("Entre com o nome: ")
6     l.append(nome)
7     i += 1
```

Lendo uma lista de nomes

Se tivermos a quantidade `n` de nomes, podemos fazer:

```
1 n = int(input("Entre com a quantidade de nomes: "))
2 l = []
3 i = 0
4 while i < n:
5     nome = input("Entre com o nome: ")
6     l.append(nome)
7     i += 1
8
9 print("Nomes digitados:")
```

Lendo uma lista de nomes

Se tivermos a quantidade `n` de nomes, podemos fazer:

```
1  n = int(input("Entre com a quantidade de nomes: "))
2  l = []
3  i = 0
4  while i < n:
5      nome = input("Entre com o nome: ")
6      l.append(nome)
7      i += 1
8
9  print("Nomes digitados:")
10 i = 0
```

Lendo uma lista de nomes

Se tivermos a quantidade `n` de nomes, podemos fazer:

```
1  n = int(input("Entre com a quantidade de nomes: "))
2  l = []
3  i = 0
4  while i < n:
5      nome = input("Entre com o nome: ")
6      l.append(nome)
7      i += 1
8
9  print("Nomes digitados:")
10 i = 0
11 while i < n:
```

Lendo uma lista de nomes

Se tivermos a quantidade `n` de nomes, podemos fazer:

```
1  n = int(input("Entre com a quantidade de nomes: "))
2  l = []
3  i = 0
4  while i < n:
5      nome = input("Entre com o nome: ")
6      l.append(nome)
7      i += 1
8
9  print("Nomes digitados:")
10 i = 0
11 while i < n:
12     print(l[i])
```

Lendo uma lista de nomes

Se tivermos a quantidade `n` de nomes, podemos fazer:

```
1  n = int(input("Entre com a quantidade de nomes: "))
2  l = []
3  i = 0
4  while i < n:
5      nome = input("Entre com o nome: ")
6      l.append(nome)
7      i += 1
8
9  print("Nomes digitados:")
10 i = 0
11 while i < n:
12     print(l[i])
13     i += 1
```

Lendo uma lista de nomes

Se tivermos a quantidade `n` de nomes, podemos fazer:

```
1  n = int(input("Entre com a quantidade de nomes: "))
2  l = []
3  i = 0
4  while i < n:
5      nome = input("Entre com o nome: ")
6      l.append(nome)
7      i += 1
8
9  print("Nomes digitados:")
10 i = 0
11 while i < n:
12     print(l[i])
13     i += 1
```

Exercício: faça uma versão onde a quantidade de nomes não é conhecida de antemão

Lendo uma lista de nomes

Se tivermos a quantidade `n` de nomes, podemos fazer:

```
1  n = int(input("Entre com a quantidade de nomes: "))
2  l = []
3  i = 0
4  while i < n:
5      nome = input("Entre com o nome: ")
6      l.append(nome)
7      i += 1
8
9  print("Nomes digitados:")
10 i = 0
11 while i < n:
12     print(l[i])
13     i += 1
```

Exercício: faça uma versão onde a quantidade de nomes não é conhecida de antemão

- Considere que o usuário pressiona enter quando não quer dar mais nomes

Elemento está na lista?

Dada uma sequência de números e um número k queremos saber se k está na sequência

Elemento está na lista?

Dada uma sequência de números e um número k queremos saber se k está na sequência

- Se desse k antes, não precisava armazenar a sequência

Elemento está na lista?

Dada uma sequência de números e um número k queremos saber se k está na sequência

- Se desse k antes, não precisava armazenar a sequência
- Mas como não sabemos o k antes, precisamos guardá-la!

Elemento está na lista?

Dada uma sequência de números e um número k queremos saber se k está na sequência

- Se desse k antes, não precisava armazenar a sequência
- Mas como não sabemos o k antes, precisamos guardá-la!

```
1 n = int(input("Número de elementos: "))
```

Elemento está na lista?

Dada uma sequência de números e um número k queremos saber se k está na sequência

- Se desse k antes, não precisava armazenar a sequência
- Mas como não sabemos o k antes, precisamos guardá-la!

```
1 n = int(input("Número de elementos: "))
```

Elemento está na lista?

Dada uma sequência de números e um número k queremos saber se k está na sequência

- Se desse k antes, não precisava armazenar a sequência
- Mas como não sabemos o k antes, precisamos guardá-la!

```
1 n = int(input("Número de elementos: "))
2 l = []
3 i = 0
4 while i < n:
5     l.append(int(input("Entre com o número: ")))
6     i = i + 1
```

Elemento está na lista?

Dada uma sequência de números e um número k queremos saber se k está na sequência

- Se desse k antes, não precisava armazenar a sequência
- Mas como não sabemos o k antes, precisamos guardá-la!

```
1 n = int(input("Número de elementos: "))
2 l = []
3 i = 0
4 while i < n:
5     l.append(int(input("Entre com o número: ")))
6     i = i + 1
7
8 k = int(input("k: "))
```

Elemento está na lista?

Dada uma sequência de números e um número k queremos saber se k está na sequência

- Se desse k antes, não precisava armazenar a sequência
- Mas como não sabemos o k antes, precisamos guardá-la!

```
1  n = int(input("Número de elementos: "))
2  l = []
3  i = 0
4  while i < n:
5      l.append(int(input("Entre com o número: ")))
6      i = i + 1
7
8  k = int(input("k: "))
9
10 i = 0
```

Elemento está na lista?

Dada uma sequência de números e um número k queremos saber se k está na sequência

- Se desse k antes, não precisava armazenar a sequência
- Mas como não sabemos o k antes, precisamos guardá-la!

```
1  n = int(input("Número de elementos: "))
2  l = []
3  i = 0
4  while i < n:
5      l.append(int(input("Entre com o número: ")))
6      i = i + 1
7
8  k = int(input("k: "))
9
10 i = 0
11 encontrou = False
```

Elemento está na lista?

Dada uma sequência de números e um número k queremos saber se k está na sequência

- Se desse k antes, não precisava armazenar a sequência
- Mas como não sabemos o k antes, precisamos guardá-la!

```
1  n = int(input("Número de elementos: "))
2  l = []
3  i = 0
4  while i < n:
5      l.append(int(input("Entre com o número: ")))
6      i = i + 1
7
8  k = int(input("k: "))
9
10 i = 0
11 encontrou = False
12 while i < n:
```

Elemento está na lista?

Dada uma sequência de números e um número k queremos saber se k está na sequência

- Se desse k antes, não precisava armazenar a sequência
- Mas como não sabemos o k antes, precisamos guardá-la!

```
1  n = int(input("Número de elementos: "))
2  l = []
3  i = 0
4  while i < n:
5      l.append(int(input("Entre com o número: ")))
6      i = i + 1
7
8  k = int(input("k: "))
9
10 i = 0
11 encontrou = False
12 while i < n:
13     if l[i] == k:
```

Elemento está na lista?

Dada uma sequência de números e um número k queremos saber se k está na sequência

- Se desse k antes, não precisava armazenar a sequência
- Mas como não sabemos o k antes, precisamos guardá-la!

```
1  n = int(input("Número de elementos: "))
2  l = []
3  i = 0
4  while i < n:
5      l.append(int(input("Entre com o número: ")))
6      i = i + 1
7
8  k = int(input("k: "))
9
10 i = 0
11 encontrou = False
12 while i < n:
13     if l[i] == k:
14         encontrou = True
```

Elemento está na lista?

Dada uma sequência de números e um número k queremos saber se k está na sequência

- Se desse k antes, não precisava armazenar a sequência
- Mas como não sabemos o k antes, precisamos guardá-la!

```
1  n = int(input("Número de elementos: "))
2  l = []
3  i = 0
4  while i < n:
5      l.append(int(input("Entre com o número: ")))
6      i = i + 1
7
8  k = int(input("k: "))
9
10 i = 0
11 encontrou = False
12 while i < n:
13     if l[i] == k:
14         encontrou = True
15     i += 1
```

Elemento está na lista?

Dada uma sequência de números e um número k queremos saber se k está na sequência

- Se desse k antes, não precisava armazenar a sequência
- Mas como não sabemos o k antes, precisamos guardá-la!

```
1  n = int(input("Número de elementos: "))
2  l = []
3  i = 0
4  while i < n:
5      l.append(int(input("Entre com o número: ")))
6      i = i + 1
7
8  k = int(input("k: "))
9
10 i = 0
11 encontrou = False
12 while i < n:
13     if l[i] == k:
14         encontrou = True
15     i += 1
16 print(encontrou)
```

Elemento está na lista?

Dada uma sequência de números e um número k queremos saber se k está na sequência

- Se desse k antes, não precisava armazenar a sequência
- Mas como não sabemos o k antes, precisamos guardá-la!

```
1  n = int(input("Número de elementos: "))
2  l = []
3  i = 0
4  while i < n:
5      l.append(int(input("Entre com o número: ")))
6      i = i + 1
7
8  k = int(input("k: "))
9
10 i = 0
11 encontrou = False
12 while i < n:
13     if l[i] == k:
14         encontrou = True
15     i += 1
16 print(encontrou)
```

É tão comum querermos saber se um elemento está na lista que o Python nos dá o comando `in`

Comando in

```
1 n = int(input("Número de elementos: "))
2 l = []
3 i = 0
4 while i < n:
5     l.append(int(input("Entre com o número: ")))
6     i += 1
7 k = int(input("k: "))
8
9 print(k in l)
```

Comando in

```
1 n = int(input("Número de elementos: "))
2 l = []
3 i = 0
4 while i < n:
5     l.append(int(input("Entre com o número: ")))
6     i += 1
7 k = int(input("k: "))
8
9 print(k in l)
```

`k in l` é:

Comando `in`

```
1 n = int(input("Número de elementos: "))
2 l = []
3 i = 0
4 while i < n:
5     l.append(int(input("Entre com o número: ")))
6     i += 1
7 k = int(input("k: "))
8
9 print(k in l)
```

`k in l` é:

- `True` se `k` está em `l`

Comando `in`

```
1 n = int(input("Número de elementos: "))
2 l = []
3 i = 0
4 while i < n:
5     l.append(int(input("Entre com o número: ")))
6     i += 1
7 k = int(input("k: "))
8
9 print(k in l)
```

`k in l` é:

- `True` se `k` está em `l`
- `False` se `k` não está em `l`

Contando as repetições

Dada uma sequência de números e um número k queremos saber quantas vezes k está na sequência

Contando as repetições

Dada uma sequência de números e um número k queremos saber quantas vezes k está na sequência

- Poderíamos contar usando `while`

Contando as repetições

Dada uma sequência de números e um número k queremos saber quantas vezes k está na sequência

- Poderíamos contar usando `while`

É tão comum percorrer uma lista, que temos algo mais simples:

Contando as repetições

Dada uma sequência de números e um número k queremos saber quantas vezes k está na sequência

- Poderíamos contar usando `while`

É tão comum percorrer uma lista, que temos algo mais simples:

```
1 n = int(input("Número de elementos: "))
2 l = []
3 i = 0
4 while i < n:
5     l.append(int(input("Entre com o número: ")))
6     i += 1
7 k = int(input("k: "))
```

Contando as repetições

Dada uma sequência de números e um número k queremos saber quantas vezes k está na sequência

- Poderíamos contar usando `while`

É tão comum percorrer uma lista, que temos algo mais simples:

```
1 n = int(input("Número de elementos: "))
2 l = []
3 i = 0
4 while i < n:
5     l.append(int(input("Entre com o número: ")))
6     i += 1
7 k = int(input("k: "))
```

Contando as repetições

Dada uma sequência de números e um número k queremos saber quantas vezes k está na sequência

- Poderíamos contar usando `while`

É tão comum percorrer uma lista, que temos algo mais simples:

```
1 n = int(input("Número de elementos: "))
2 l = []
3 i = 0
4 while i < n:
5     l.append(int(input("Entre com o número: ")))
6     i += 1
7 k = int(input("k: "))
8
9 conta = 0
```

Contando as repetições

Dada uma sequência de números e um número k queremos saber quantas vezes k está na sequência

- Poderíamos contar usando `while`

É tão comum percorrer uma lista, que temos algo mais simples:

```
1  n = int(input("Número de elementos: "))
2  l = []
3  i = 0
4  while i < n:
5      l.append(int(input("Entre com o número: ")))
6      i += 1
7  k = int(input("k: "))
8
9  conta = 0
10 for x in l: # para cada elemento x da lista l faça
```

Contando as repetições

Dada uma sequência de números e um número k queremos saber quantas vezes k está na sequência

- Poderíamos contar usando `while`

É tão comum percorrer uma lista, que temos algo mais simples:

```
1  n = int(input("Número de elementos: "))
2  l = []
3  i = 0
4  while i < n:
5      l.append(int(input("Entre com o número: ")))
6      i += 1
7  k = int(input("k: "))
8
9  conta = 0
10 for x in l: # para cada elemento x da lista l faça
11     if x == k:
```

Contando as repetições

Dada uma sequência de números e um número k queremos saber quantas vezes k está na sequência

- Poderíamos contar usando `while`

É tão comum percorrer uma lista, que temos algo mais simples:

```
1  n = int(input("Número de elementos: "))
2  l = []
3  i = 0
4  while i < n:
5      l.append(int(input("Entre com o número: ")))
6      i += 1
7  k = int(input("k: "))
8
9  conta = 0
10 for x in l: # para cada elemento x da lista l faça
11     if x == k:
12         conta += 1
```

Contando as repetições

Dada uma sequência de números e um número k queremos saber quantas vezes k está na sequência

- Poderíamos contar usando `while`

É tão comum percorrer uma lista, que temos algo mais simples:

```
1  n = int(input("Número de elementos: "))
2  l = []
3  i = 0
4  while i < n:
5      l.append(int(input("Entre com o número: ")))
6      i += 1
7  k = int(input("k: "))
8
9  conta = 0
10 for x in l: # para cada elemento x da lista l faça
11     if x == k:
12         conta += 1
13 print(conta)
```

Note que o significado de `in` nesse caso é levemente diferente...

Padrão de contagem

É comum contarmos de um número até outro em um laço

Padrão de contagem

É comum contarmos de um número até outro em um laço

Ex:

```
1 i = 0
2 while i < n:
3     l.append(int(input("Entre com o número: ")))
4     i += 1
```

Padrão de contagem

É comum contarmos de um número até outro em um laço

Ex:

```
1 i = 0
2 while i < n:
3     l.append(int(input("Entre com o número: ")))
4     i += 1
```

Há uma estrutura que fica sempre aparecendo:

Padrão de contagem

É comum contarmos de um número até outro em um laço

Ex:

```
1 i = 0
2 while i < n:
3     l.append(int(input("Entre com o número: ")))
4     i += 1
```

Há uma estrutura que fica sempre aparecendo:

- Inicializa a variável com zero (linha 1)

Padrão de contagem

É comum contarmos de um número até outro em um laço

Ex:

```
1 i = 0
2 while i < n:
3     l.append(int(input("Entre com o número: ")))
4     i += 1
```

Há uma estrutura que fica sempre aparecendo:

- Inicializa a variável com zero (linha 1)
- Executa um laço até chegar em um valor (linha 2)

Padrão de contagem

É comum contarmos de um número até outro em um laço

Ex:

```
1 i = 0
2 while i < n:
3     l.append(int(input("Entre com o número: ")))
4     i += 1
```

Há uma estrutura que fica sempre aparecendo:

- Inicializa a variável com zero (linha 1)
- Executa um laço até chegar em um valor (linha 2)
- Faz alguma coisa (linha 3)

Padrão de contagem

É comum contarmos de um número até outro em um laço

Ex:

```
1 i = 0
2 while i < n:
3     l.append(int(input("Entre com o número: ")))
4     i += 1
```

Há uma estrutura que fica sempre aparecendo:

- Inicializa a variável com zero (linha 1)
- Executa um laço até chegar em um valor (linha 2)
- Faz alguma coisa (linha 3)
- Soma um na variável ao final do laço (linha 4)

Reescrevendo usando `for` e `range`

Ao invés de

```
1 i = 0
2 while i < n:
3     l.append(int(input("Entre com o número: ")))
4     i += 1
```

Reescrevendo usando `for` e `range`

Ao invés de

```
1 i = 0
2 while i < n:
3     l.append(int(input("Entre com o número: ")))
4     i += 1
```

Podemos escrever

```
1 for i in range(n):
2     l.append(int(input("Entre com o número: ")))
```

Reescrevendo usando `for` e `range`

Ao invés de

```
1 i = 0
2 while i < n:
3     l.append(int(input("Entre com o número: ")))
4     i += 1
```

Podemos escrever

```
1 for i in range(n):
2     l.append(int(input("Entre com o número: ")))
```

O `range(n)` é como se fosse a lista `[0, 1, ..., n - 1]`

Reescrevendo usando `for` e `range`

Ao invés de

```
1 i = 0
2 while i < n:
3     l.append(int(input("Entre com o número: ")))
4     i += 1
```

Podemos escrever

```
1 for i in range(n):
2     l.append(int(input("Entre com o número: ")))
```

O `range(n)` é como se fosse a lista `[0, 1, ..., n - 1]`

- Mas não é uma lista

Reescrevendo usando `for` e `range`

Ao invés de

```
1 i = 0
2 while i < n:
3     l.append(int(input("Entre com o número: ")))
4     i += 1
```

Podemos escrever

```
1 for i in range(n):
2     l.append(int(input("Entre com o número: ")))
```

O `range(n)` é como se fosse a lista `[0, 1, ..., n - 1]`

- Mas não é uma lista
- Seu tipo é `range`

Reescrevendo usando `for` e `range`

Ao invés de

```
1 i = 0
2 while i < n:
3     l.append(int(input("Entre com o número: ")))
4     i += 1
```

Podemos escrever

```
1 for i in range(n):
2     l.append(int(input("Entre com o número: ")))
```

O `range(n)` é como se fosse a lista `[0, 1, ..., n - 1]`

- Mas não é uma lista
- Seu tipo é `range`
- Pode ser convertido em lista usando `list(range(n))`

Revisitando soluções

Imprimindo os próximos k números

```
1 n = int(input("Entre com n: "))
2 k = int(input("Entre com k: "))
3 for i in range(k):
4     print(n + i)
```

Revisitando soluções

Soma da Progressão Aritmética

```
1  a_1 = int(input("Entre com a_1: "))
2  n = int(input("Entre com n: "))
3  r = int(input("Entre com r: "))
4
5  esperado = a_1*n + n * (n - 1) * r / 2
6
7  soma = 0
8  atual = a_1
9  for i in range(n):
10     soma += atual
11     atual += r
12
13 if soma != esperado:
14     print("Fórmula incorreta!")
15     print("Esperado: ", esperado)
16     print("Obtido: ", soma)
17 else:
18     print("Fórmula correta!")
```

Revisitando soluções

Lendo e imprimindo uma lista

```
1 n = int(input("Entre com a quantidade de nomes: "))
2 l = []
3 for i in range(n):
4     nome = input("Entre com o nome: ")
5     l.append(nome)
6
7 print("Nomes digitados:")
8 for x in l:
9     print(x)
```

Versões do `range`

Temos três versões:

Versões do `range`

Temos três versões:

- `range(fim)`

Versões do `range`

Temos três versões:

- `range(fim)`
  - Intervalo de `0` (incluído) a `fim` (excluído)

Versões do `range`

Temos três versões:

- `range(fim)`
  - Intervalo de `0` (incluído) a `fim` (excluído)
  - Ex: `range(10)` é $0, 1, \ldots, 9$

Versões do `range`

Temos três versões:

- `range(fim)`
  - Intervalo de `0` (incluído) a `fim` (excluído)
  - Ex: `range(10)` é $0, 1, \ldots, 9$
- `range(inicio, fim)`

Versões do `range`

Temos três versões:

- `range(fim)`
  - Intervalo de `0` (incluído) a `fim` (excluído)
  - Ex: `range(10)` é $0, 1, \ldots, 9$
- `range(inicio, fim)`
  - Intervalo de `inicio` (incluído) a `fim` (excluído)

Versões do `range`

Temos três versões:

- `range(fim)`
  - Intervalo de `0` (incluído) a `fim` (excluído)
  - Ex: `range(10)` é $0, 1, \ldots, 9$
- `range(inicio, fim)`
  - Intervalo de `inicio` (incluído) a `fim` (excluído)
  - Ex: `range(2, 10)` é $2, 3, \ldots, 9$

Versões do `range`

Temos três versões:

- `range(fim)`
  - Intervalo de `0` (incluído) a `fim` (excluído)
  - Ex: `range(10)` é $0, 1, \ldots, 9$
- `range(inicio, fim)`
  - Intervalo de `inicio` (incluído) a `fim` (excluído)
  - Ex: `range(2, 10)` é $2, 3, \ldots, 9$
- `range(inicio, fim, passo)`

Versões do `range`

Temos três versões:

- `range(fim)`
  - Intervalo de `0` (incluído) a `fim` (excluído)
  - Ex: `range(10)` é $0, 1, \ldots, 9$
- `range(inicio, fim)`
  - Intervalo de `inicio` (incluído) a `fim` (excluído)
  - Ex: `range(2, 10)` é $2, 3, \ldots, 9$
- `range(inicio, fim, passo)`
  - Intervalo de `inicio` (incluído) a `fim` (excluído), pulando de `passo` em `passo`

Versões do `range`

Temos três versões:

- `range(fim)`
  - Intervalo de `0` (incluído) a `fim` (excluído)
  - Ex: `range(10)` é $0, 1, \ldots, 9$
- `range(inicio, fim)`
  - Intervalo de `inicio` (incluído) a `fim` (excluído)
  - Ex: `range(2, 10)` é $2, 3, \ldots, 9$
- `range(inicio, fim, passo)`
  - Intervalo de `inicio` (incluído) a `fim` (excluído), pulando de `passo` em `passo`
  - Ex: `range(3, 10, 2)` é $3, 5, 7, 9$

Versões do `range`

Temos três versões:

- `range(fim)`
  - Intervalo de `0` (incluído) a `fim` (excluído)
  - Ex: `range(10)` é $0, 1, \ldots, 9$
- `range(inicio, fim)`
  - Intervalo de `inicio` (incluído) a `fim` (excluído)
  - Ex: `range(2, 10)` é $2, 3, \ldots, 9$
- `range(inicio, fim, passo)`
  - Intervalo de `inicio` (incluído) a `fim` (excluído), pulando de `passo` em `passo`
  - Ex: `range(3, 10, 2)` é $3, 5, 7, 9$
  - Ex: `range(10, 3, -2)` é $10, 8, 6, 4$

Revisitando soluções

Primo

```
1 p = int(input("Entre com p: "))
2 primo = True
3 for k in range(2, p):
4     if p % k == 0:
5         primo = False
6         break  # para a execução do laço
7 print(p >= 2 and primo)
```

Note que agora estamos indo até $p - 1$ e não $\sqrt{p}$...

- Daria para calcular $\sqrt{p}$ antes

Sempre dá para usar?

Como usar `for ... in range(...)` nesse código?

Sempre dá para usar?

Como usar `for ... in range(...)` nesse código?

```
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
9 print(lista)
```

Sempre dá para usar?

Como usar `for ... in range(...)` nesse código?

```
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
9 print(lista)
```

Esse é um `while` que foge do padrão!

Exercícios

1. Dado n, imprima todos os números ímpares menores do que n

Exercícios

1. Dado n, imprima todos os números ímpares menores do que n
2. Imprima os n primeiros números naturais em ordem inversa

Imprimindo primos

Já sabemos verificar se um número é primo

```
1 k = 2
2 # é melhor testar k * k <= p do que k <= p ** (1/2)
3 while k * k <= p and primo:
4     if p % k == 0:
5         primo = False
6     k += 1
7 print(p >= 2 and primo)
```

Imprimindo primos

Já sabemos verificar se um número é primo

```
1 k = 2
2 # é melhor testar k * k <= p do que k <= p ** (1/2)
3 while k * k <= p and primo:
4     if p % k == 0:
5         primo = False
6     k += 1
7 print(p >= 2 and primo)
```

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

Imprimindo primos

Já sabemos verificar se um número é primo

```
1 k = 2
2 # é melhor testar k * k <= p do que k <= p ** (1/2)
3 while k * k <= p and primo:
4     if p % k == 0:
5         primo = False
6     k += 1
7 print(p >= 2 and primo)
```

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

- Precisaríamos executar o código acima várias vezes...

Imprimindo primos

Já sabemos verificar se um número é primo

```
1 k = 2
2 # é melhor testar k * k <= p do que k <= p ** (1/2)
3 while k * k <= p and primo:
4     if p % k == 0:
5         primo = False
6     k += 1
7 print(p >= 2 and primo)
```

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

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

Laços encaixados

Podemos colocar um laço dentro de outro!

Laços encaixados

Podemos colocar um laço dentro de outro!

```
1 n = int(input("Entre com n: "))
```

Laços encaixados

Podemos colocar um laço dentro de outro!

```
1 n = int(input("Entre com n: "))
```

Laços encaixados

Podemos colocar um laço dentro de outro!

```
1 n = int(input("Entre com n: "))
2
3 # n também é considerado
4 for p in range(2, n + 1):
```

Laços encaixados

Podemos colocar um laço dentro de outro!

```
1 n = int(input("Entre com n: "))
2
3 # n também é considerado
4 for p in range(2, n + 1):
5     primo = True
```

Laços encaixados

Podemos colocar um laço dentro de outro!

```
1 n = int(input("Entre com n: "))
2
3 # n também é considerado
4 for p in range(2, n + 1):
5     primo = True
6     k = 2
```

Laços encaixados

Podemos colocar um laço dentro de outro!

```
1 n = int(input("Entre com n: "))
2
3 # n também é considerado
4 for p in range(2, n + 1):
5     primo = True
6     k = 2
7     while k * k <= p and primo:
```

Laços encaixados

Podemos colocar um laço dentro de outro!

```
1 n = int(input("Entre com n: "))
2
3 # n também é considerado
4 for p in range(2, n + 1):
5     primo = True
6     k = 2
7     while k * k <= p and primo:
8         # verifica se k divide p
```

Laços encaixados

Podemos colocar um laço dentro de outro!

```
1 n = int(input("Entre com n: "))
2
3 # n também é considerado
4 for p in range(2, n + 1):
5     primo = True
6     k = 2
7     while k * k <= p and primo:
8         # verifica se k divide p
9         if p % k == 0:
```

Laços encaixados

Podemos colocar um laço dentro de outro!

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

Laços encaixados

Podemos colocar um laço dentro de outro!

```
1  n = int(input("Entre com n: "))
2
3  # n também é considerado
4  for p in range(2, n + 1):
5      primo = True
6      k = 2
7      while k * k <= p and primo:
8          # verifica se k divide p
9          if p % k == 0:
10             primo = False
11         k += 1
```

Laços encaixados

Podemos colocar um laço dentro de outro!

```
1  n = int(input("Entre com n: "))
2
3  # n também é considerado
4  for p in range(2, n + 1):
5      primo = True
6      k = 2
7      while k * k <= p and primo:
8          # verifica se k divide p
9          if p % k == 0:
10             primo = False
11         k += 1
12     if primo:
```

Laços encaixados

Podemos colocar um laço dentro de outro!

```
1  n = int(input("Entre com n: "))
2
3  # n também é considerado
4  for p in range(2, n + 1):
5      primo = True
6      k = 2
7      while k * k <= p and primo:
8          # verifica se k divide p
9          if p % k == 0:
10             primo = False
11         k += 1
12     if primo:
13         print(p)
```

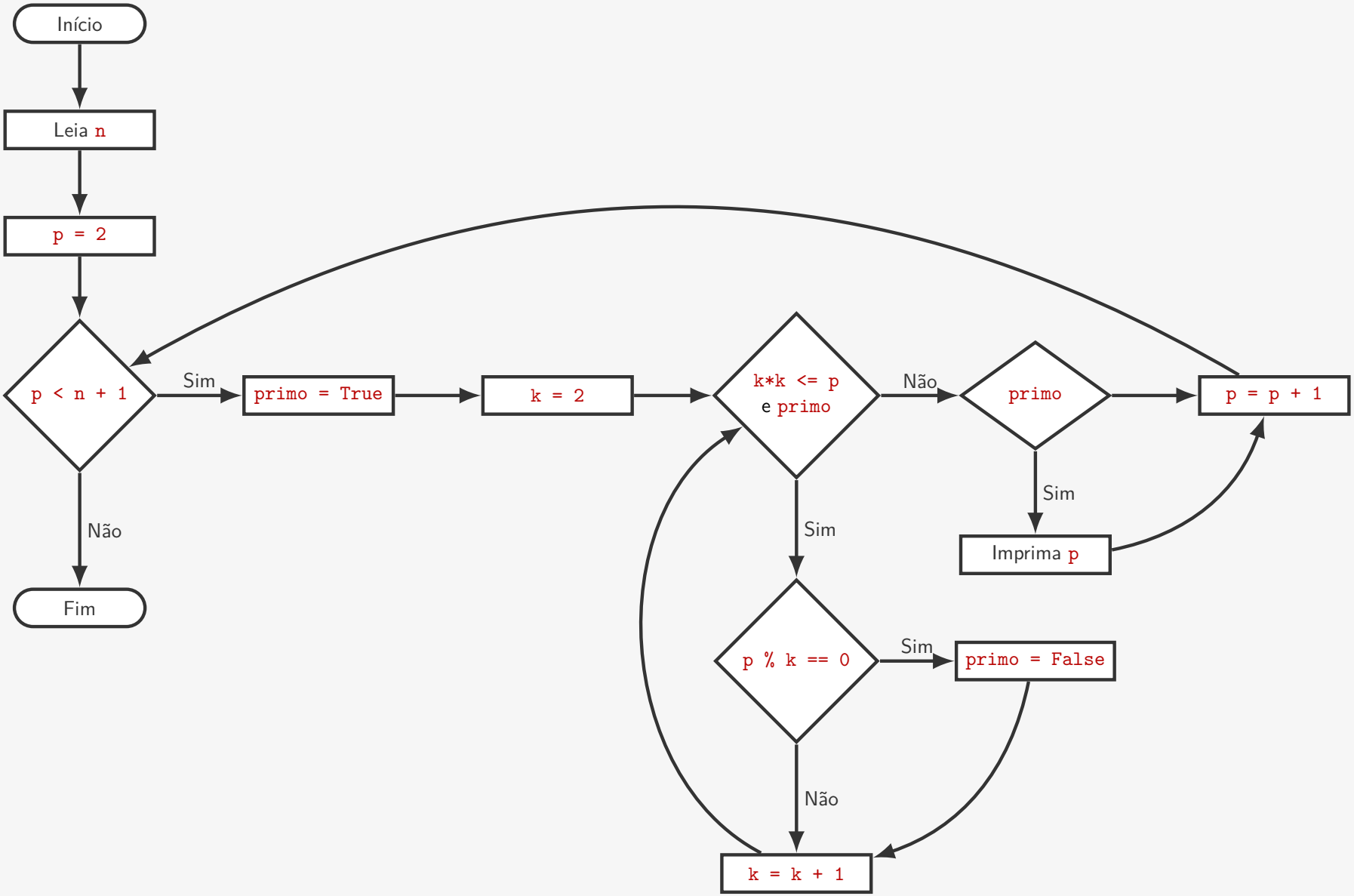
Laços encaixados

Podemos colocar um laço dentro de outro!

```
1  n = int(input("Entre com n: "))
2
3  # n também é considerado
4  for p in range(2, n + 1):
5      primo = True
6      k = 2
7      while k * k <= p and primo:
8          # verifica se k divide p
9          if p % k == 0:
10             primo = False
11         k += 1
12     if primo:
13         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
Não
Sim
primo
Sim
Imprima p
p = p + 1
p % k == 0
Sim
primo = False
Não
k = k + 1

Versão com break

```
1  n = int(input("Entre com n: "))
2
3  # n também é considerado
4  for p in range(2, n + 1):
5      primo = True
6      k = 2
7      while k * k <= p:
8          # verifica se k divide p
9          if p % k == 0:
10             primo = False
11             break
12         k += 1
13     if primo:
14         print(p)
```

Versão com break

```
1  n = int(input("Entre com n: "))
2
3  # n também é considerado
4  for p in range(2, n + 1):
5      primo = True
6      k = 2
7      while k * k <= p:
8          # verifica se k divide p
9          if p % k == 0:
10             primo = False
11             break
12         k += 1
13     if primo:
14         print(p)
```

- O `break` interrompe o laço mais interno apenas

Versão com break

```
1  n = int(input("Entre com n: "))
2
3  # n também é considerado
4  for p in range(2, n + 1):
5      primo = True
6      k = 2
7      while k * k <= p:
8          # verifica se k divide p
9          if p % k == 0:
10             primo = False
11             break
12         k += 1
13     if primo:
14         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 n: "))
2
3  # n também é considerado
4  for p in range(2, n + 1):
5      primo = True
6      k = 2
7      while k * k <= p:
8          # verifica se k divide p
9          if p % k == 0:
10             primo = False
11             break
12         k += 1
13     if primo:
14         print(p)
```

- O `break` interrompe o laço mais interno apenas
- Ele reduz um pouco a legibilidade do código
  - Mas é necessário no `for` pois não há condição booleana

Versão com break

```
1  n = int(input("Entre com n: "))
2
3  # n também é considerado
4  for p in range(2, n + 1):
5      primo = True
6      k = 2
7      while k * k <= p:
8          # verifica se k divide p
9          if p % k == 0:
10             primo = False
11             break
12         k += 1
13     if primo:
14         print(p)
```

- O `break` interrompe o laço mais interno apenas
- Ele reduz um pouco a legibilidade do código
  - Mas é necessário no `for` pois não há condição booleana
- Existe também um comando chamado `continue`

Versão com break

```
1  n = int(input("Entre com n: "))
2
3  # n também é considerado
4  for p in range(2, n + 1):
5      primo = True
6      k = 2
7      while k * k <= p:
8          # verifica se k divide p
9          if p % k == 0:
10             primo = False
11             break
12         k += 1
13     if primo:
14         print(p)
```

- O `break` interrompe o laço mais interno apenas
- Ele reduz um pouco a legibilidade do código
  - Mas é necessário no `for` pois não há 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 n: "))
2
3  # n também é considerado
4  for p in range(2, n + 1):
5      primo = True
6      k = 2
7      while k * k <= p:
8          # verifica se k divide p
9          if p % k == 0:
10             primo = False
11             break
12         k += 1
13     if primo:
14         print(p)
```

- O `break` interrompe o laço mais interno apenas
- Ele reduz um pouco a legibilidade do código
  - Mas é necessário no `for` pois não há 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...

Crivo de Eratóstenes

A ideia anterior é a base do algoritmo de Erastóstenes

Crivo de Eratóstenes

A ideia anterior é a base do algoritmo de Erastóstenes

- Matemático grego (276 AEC - 194 AEC)

Crivo de Eratóstenes

A ideia anterior é a base do algoritmo de Erastóstenes

- Matemático grego (276 AEC - 194 AEC)
- Foi bibliotecário chefe na Biblioteca de Alexandria

Crivo de Eratóstenes

A ideia anterior é a base do algoritmo de Erastóstenes

- Matemático grego (276 AEC - 194 AEC)
- Foi bibliotecário chefe na Biblioteca de Alexandria

Pseudocódigo:

```
1 Leia n
2 Marque todo número de 2 a n como primo
3 Para cada número p de 2 a n faça
4     Se p está marcado como primo
5         Imprima p
6         Para cada múltiplo k de p menor ou igual a n faça
7             Marque k como não-primo
```

Crivo de Eratóstenes

A ideia anterior é a base do algoritmo de Erastóstenes

- Matemático grego (276 AEC - 194 AEC)
- Foi bibliotecário chefe na Biblioteca de Alexandria

Pseudocódigo:

```
1 Leia n
2 Marque todo número de 2 a n como primo
3 Para cada número p de 2 a n faça
4     Se p está marcado como primo
5         Imprima p
6         Para cada múltiplo k de p menor ou igual a n faça
7             Marque k como não-primo
```

Exercício: Vamos simular na lousa!

Crivo de Eratóstenes

A ideia anterior é a base do algoritmo de Erastóstenes

- Matemático grego (276 AEC - 194 AEC)
- Foi bibliotecário chefe na Biblioteca de Alexandria

Pseudocódigo:

```
1 Leia n
2 Marque todo número de 2 a n como primo
3 Para cada número p de 2 a n faça
4     Se p está marcado como primo
5         Imprima p
6         Para cada múltiplo k de p menor ou igual a n faça
7             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

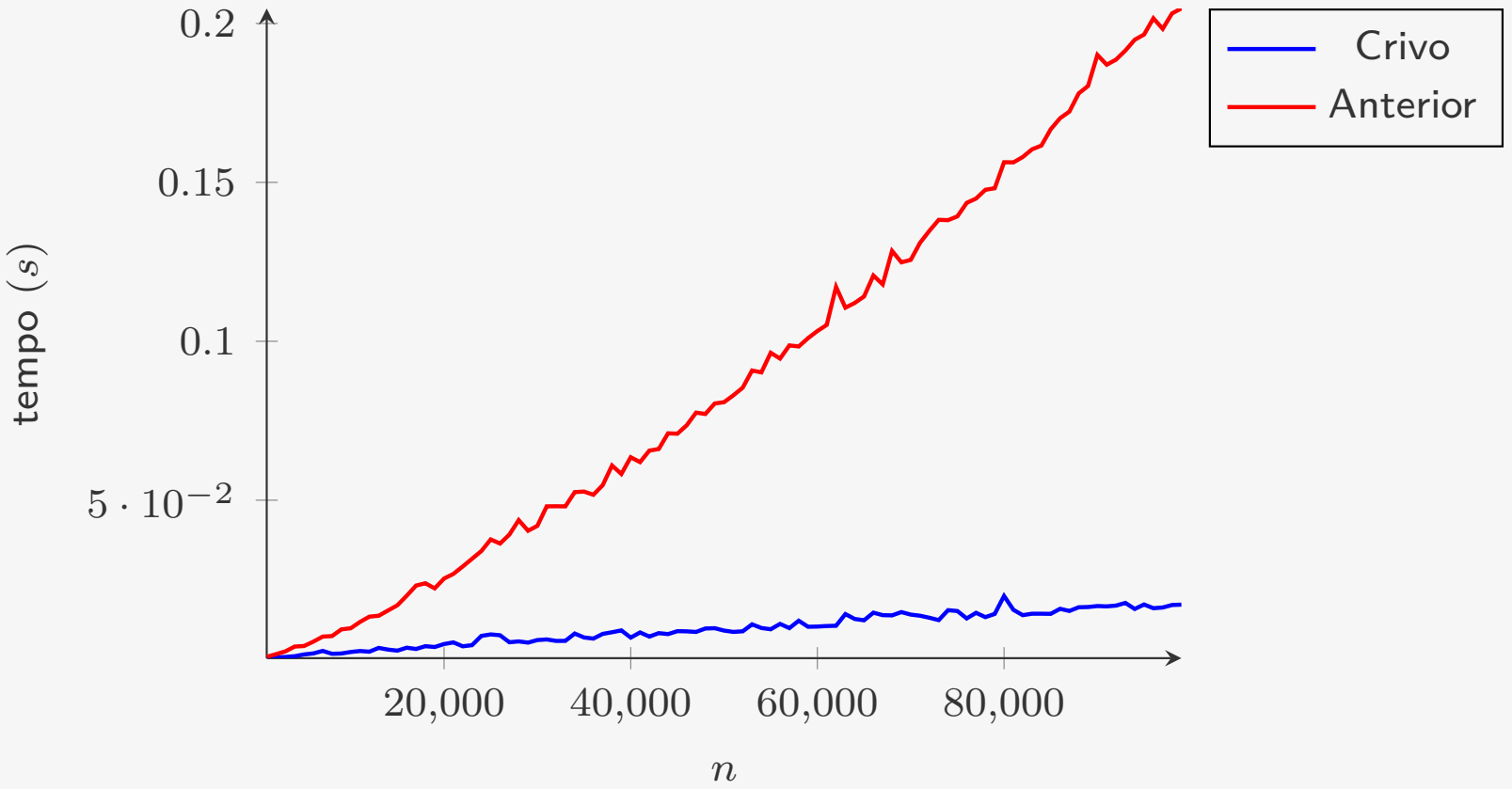
```
1  n = int(input("Entre com n: "))
2  # cria uma lista de n + 1 elementos todos True
3  eh_primo = []
4  for i in range(n + 1):
5      eh_primo.append(True)
6
7  eh_primo[0] = eh_primo[1] = False
8
9  for p in range(2, n + 1):
10     if eh_primo[p]:
11         print(p)
12         # podemos começar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

Vale a pena usar o Crivo?

Comparação de tempo de execução dos dois algoritmos

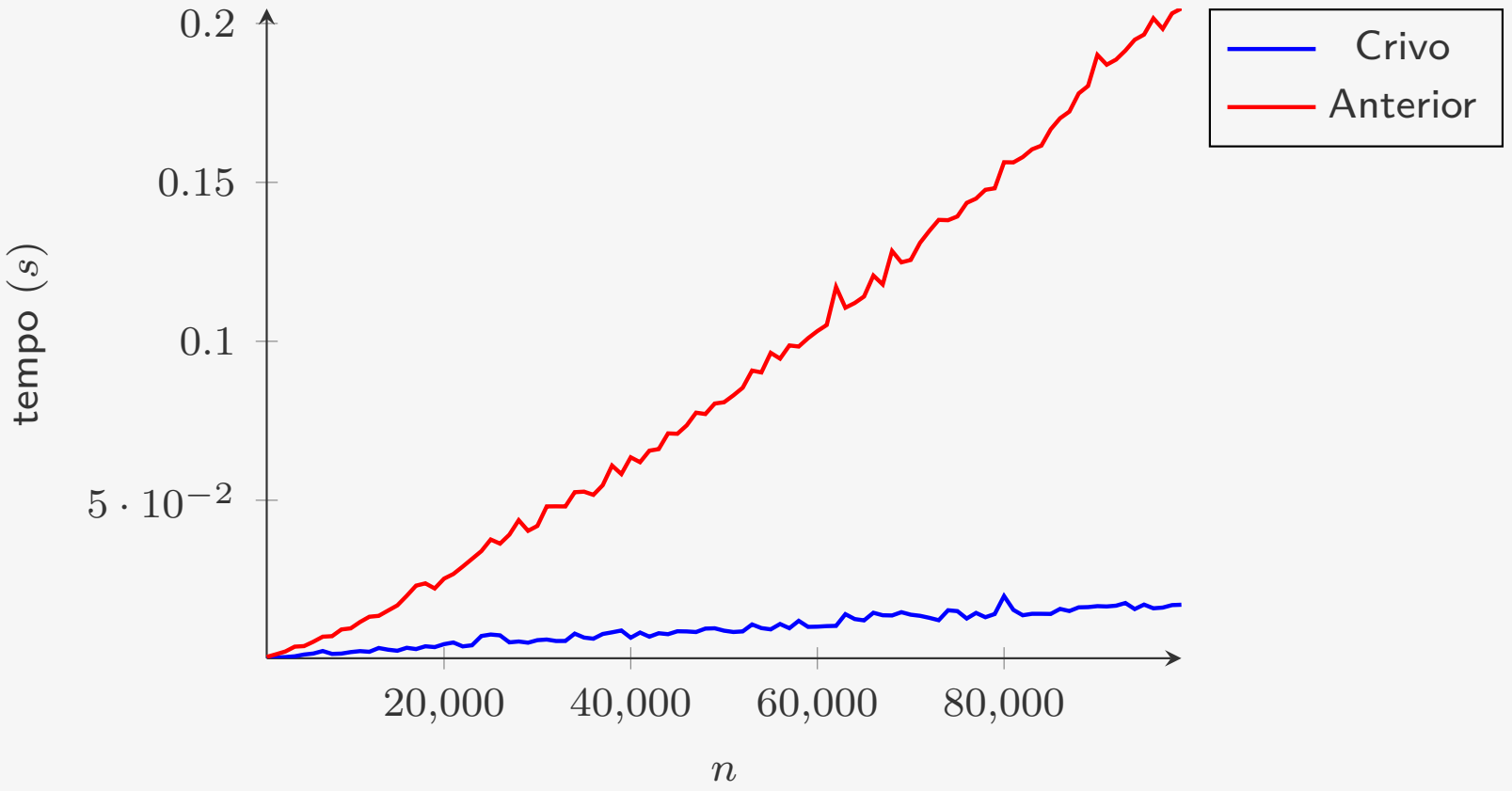
Vale a pena usar o Crivo?

Comparação de tempo de execução dos dois algoritmos



Vale a pena usar o Crivo?

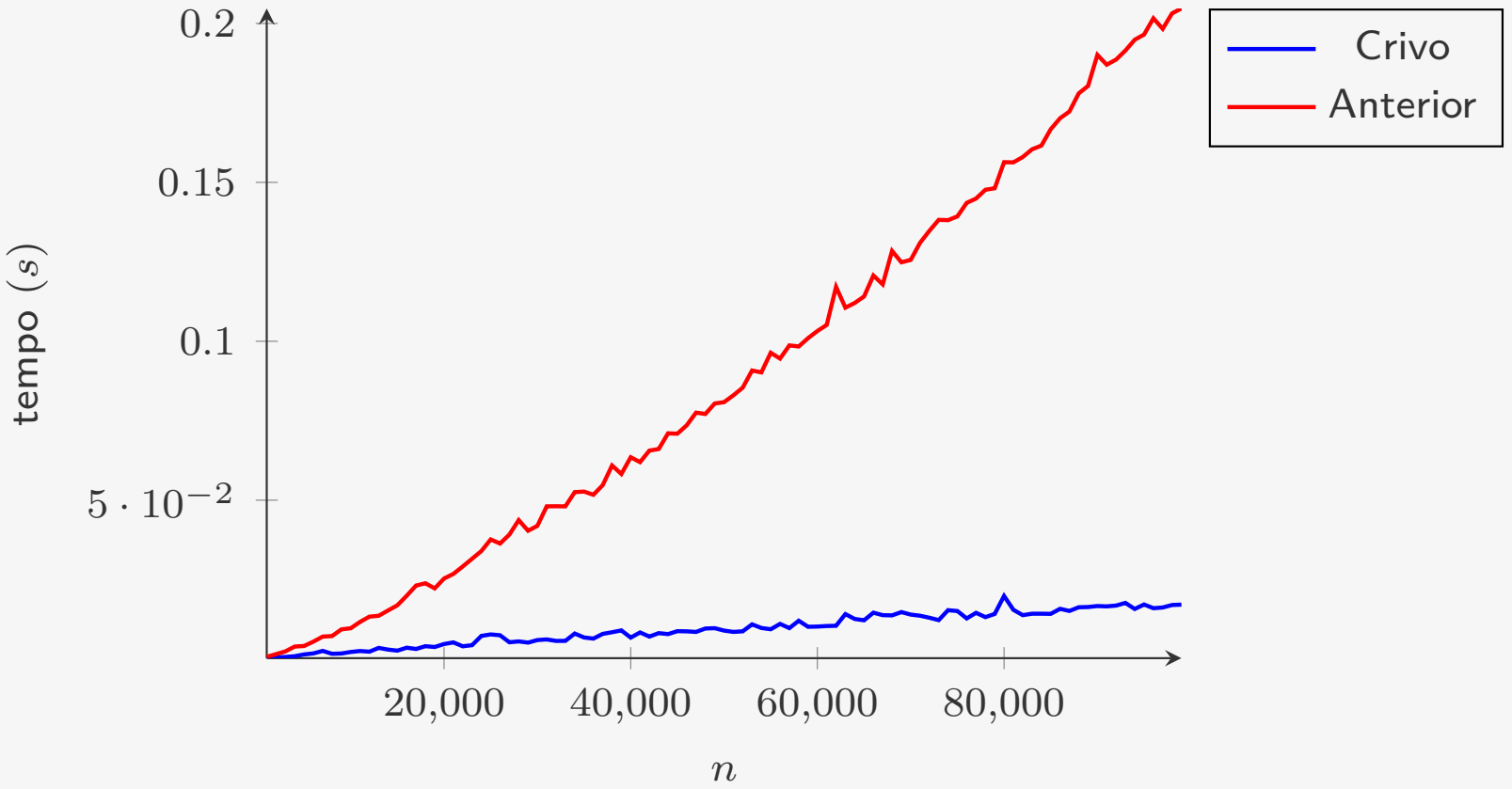
Comparação de tempo de execução dos dois algoritmos



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!

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.