MC102 — Código Bom

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

Universidade Estadual de Campinas

Atualizado em: 2023-04-21 17:02

O que é um código bom?

Talvez é mais fácil olhar primeiro o que não é...

O que é um código bom?

Talvez é mais fácil olhar primeiro o que não é...

```
1  def imprime(n):
2      for i in range(3,n,2):
3          é=True
4          k=3
5          while k<i and é:
6
7
8              é=i%k!=0
9
10
11
12             k+=1
13         if é: print(i)
```

O que é um código bom?

Talvez é mais fácil olhar primeiro o que não é...

```
1  def imprime(n):
2      for i in range(3,n,2):
3          é=True
4          k=3
5          while k<i and é:
6
7
8              é=i%k!=0
9
10
11
12             k+=1
13         if é: print(i)
```

O que essa função faz?

O que é um código bom?

Talvez é mais fácil olhar primeiro o que não é...

```
1  def imprime(n):
2      for i in range(3,n,2):
3          é=True
4          k=3
5          while k<i and é:
6
7
8              é=i%k!=0
9
10
11
12             k+=1
13         if é: print(i)
```

O que essa função faz?

- Tem que pensar um bom tempo para perceber

Estilo

Quando programamos é importante ter um estilo de escrita claro e consistente

Estilo

Quando programamos é importante ter um estilo de escrita claro e consistente

- E de acordo com o que o resto da equipe usa

Estilo

Quando programamos é importante ter um estilo de escrita claro e consistente

- E de acordo com o que o resto da equipe usa

É comum usarmos um linter para verificar regras de estilo

Estilo

Quando programamos é importante ter um estilo de escrita claro e consistente

- E de acordo com o que o resto da equipe usa

É comum usarmos um linter para verificar regras de estilo

- O linter lê o seu código e indica problemas de legibilidade

Estilo

Quando programamos é importante ter um estilo de escrita claro e consistente

- E de acordo com o que o resto da equipe usa

É comum usarmos um linter para verificar regras de estilo

- O linter lê o seu código e indica problemas de legibilidade

Em Python, uma opção é usar o Flake8

Estilo

Quando programamos é importante ter um estilo de escrita claro e consistente

- E de acordo com o que o resto da equipe usa

É comum usarmos um linter para verificar regras de estilo

- O linter lê o seu código e indica problemas de legibilidade

Em Python, uma opção é usar o Flake8

- Vamos ver o que ele diz do nosso código original

Flake8 no nosso código

```
ruim1.py:2:21: E231 missing whitespace after ','
ruim1.py:2:23: E231 missing whitespace after ','
ruim1.py:3:10: E225 missing whitespace around operator
ruim1.py:4:10: E225 missing whitespace around operator
ruim1.py:5:16: E225 missing whitespace around operator
ruim1.py:6:1: W293 blank line contains whitespace
ruim1.py:7:1: W293 blank line contains whitespace
ruim1.py:8:13: E303 too many blank lines (2)
ruim1.py:8:14: E225 missing whitespace around operator
ruim1.py:8:16: E228 missing whitespace around modulo operator
ruim1.py:8:18: E225 missing whitespace around operator
ruim1.py:9:1: W293 blank line contains whitespace
ruim1.py:10:1: W293 blank line contains whitespace
ruim1.py:11:1: W293 blank line contains whitespace
ruim1.py:12:13: E303 too many blank lines (3)
ruim1.py:12:14: E225 missing whitespace around operator
ruim1.py:12:17: W291 trailing whitespace
ruim1.py:13:13: E701 multiple statements on one line (colon)
ruim1.py:13:23: W291 trailing whitespace
ruim1.py:16:12: W292 no newline at end of file
```

Flake8 no nosso código

```
ruim1.py:2:21: E231 missing whitespace after ','
ruim1.py:2:23: E231 missing whitespace after ','
ruim1.py:3:10: E225 missing whitespace around operator
ruim1.py:4:10: E225 missing whitespace around operator
ruim1.py:5:16: E225 missing whitespace around operator
ruim1.py:6:1: W293 blank line contains whitespace
ruim1.py:7:1: W293 blank line contains whitespace
ruim1.py:8:13: E303 too many blank lines (2)
ruim1.py:8:14: E225 missing whitespace around operator
ruim1.py:8:16: E228 missing whitespace around modulo operator
ruim1.py:8:18: E225 missing whitespace around operator
ruim1.py:9:1: W293 blank line contains whitespace
ruim1.py:10:1: W293 blank line contains whitespace
ruim1.py:11:1: W293 blank line contains whitespace
ruim1.py:12:13: E303 too many blank lines (3)
ruim1.py:12:14: E225 missing whitespace around operator
ruim1.py:12:17: W291 trailing whitespace
ruim1.py:13:13: E701 multiple statements on one line (colon)
ruim1.py:13:23: W291 trailing whitespace
ruim1.py:16:12: W292 no newline at end of file
```

Temos a indicação do nome do arquivo, da linha, da coluna e do erro/aviso com o código e descrição

Nova versão

```
1 def imprime(n):
2     for i in range(3, n, 2):
3         é = True
4         k = 3
5         while k < i and é:
6             é = (i % k != 0)
7             k += 1
8         if é:
9             print(i)
```

Nova versão

```
1 def imprime(n):
2     for i in range(3, n, 2):
3         é = True
4         k = 3
5         while k < i and é:
6             é = (i % k != 0)
7             k += 1
8         if é:
9             print(i)
```

- Não há quebras de linhas excessivas

Nova versão

```
1 def imprime(n):
2     for i in range(3, n, 2):
3         é = True
4         k = 3
5         while k < i and é:
6             é = (i % k != 0)
7             k += 1
8         if é:
9             print(i)
```

- Não há quebras de linhas excessivas
- Os operadores binários estão cercados por espaços

Nova versão

```
1 def imprime(n):
2     for i in range(3, n, 2):
3         é = True
4         k = 3
5         while k < i and é:
6             é = (i % k != 0)
7             k += 1
8         if é:
9             print(i)
```

- Não há quebras de linhas excessivas
- Os operadores binários estão cercados por espaços
- Temos espaço após as vírgulas

Nova versão

```
1 def imprime(n):
2     for i in range(3, n, 2):
3         é = True
4         k = 3
5         while k < i and é:
6             é = (i % k != 0)
7             k += 1
8         if é:
9             print(i)
```

- Não há quebras de linhas excessivas
- Os operadores binários estão cercados por espaços
- Temos espaço após as vírgulas
- Não tenho o corpo do `if` na mesma linha

Nova versão

```
1 def imprime(n):
2     for i in range(3, n, 2):
3         é = True
4         k = 3
5         while k < i and é:
6             é = (i % k != 0)
7             k += 1
8         if é:
9             print(i)
```

- Não há quebras de linhas excessivas
- Os operadores binários estão cercados por espaços
- Temos espaço após as vírgulas
- Não tenho o corpo do `if` na mesma linha
- Usei um parêntese na linha 6 para deixar mais claro

Nova versão

```
1 def imprime(n):
2     for i in range(3, n, 2):
3         é = True
4         k = 3
5         while k < i and é:
6             é = (i % k != 0)
7             k += 1
8         if é:
9             print(i)
```

- Não há quebras de linhas excessivas
- Os operadores binários estão cercados por espaços
- Temos espaço após as vírgulas
- Não tenho o corpo do `if` na mesma linha
- Usei um parêntese na linha 6 para deixar mais claro
- Flake8 não aponta mais erros

Nova versão

```
1 def imprime(n):
2     for i in range(3, n, 2):
3         é = True
4         k = 3
5         while k < i and é:
6             é = (i % k != 0)
7             k += 1
8         if é:
9             print(i)
```

- Não há quebras de linhas excessivas
- Os operadores binários estão cercados por espaços
- Temos espaço após as vírgulas
- Não tenho o corpo do `if` na mesma linha
- Usei um parêntese na linha 6 para deixar mais claro
- Flake8 não aponta mais erros
- Mas dá para ser melhor...

Outra versão

```
1 def imprime_primos(n):
2     for numero in range(3, n, 2):
3         é_primo = True
4         divisor = 3
5         while divisor < numero and é_primo:
6             é_primo = (numero % divisor != 0)
7             divisor += 1
8         if é_primo:
9             print(numero)
```

Outra versão

```
1 def imprime_primos(n):
2     for numero in range(3, n, 2):
3         é_primo = True
4         divisor = 3
5         while divisor < numero and é_primo:
6             é_primo = (numero % divisor != 0)
7             divisor += 1
8         if é_primo:
9             print(numero)
```

- Nome da função dá uma pista do que ela faz

Outra versão

```
1 def imprime_primos(n):
2     for numero in range(3, n, 2):
3         é_primo = True
4         divisor = 3
5         while divisor < numero and é_primo:
6             é_primo = (numero % divisor != 0)
7             divisor += 1
8         if é_primo:
9             print(numero)
```

- Nome da função dá uma pista do que ela faz
- Nomes das variáveis dão pistas do que são

Outra versão

```
1 def imprime_primos(n):
2     for numero in range(3, n, 2):
3         é_primo = True
4         divisor = 3
5         while divisor < numero and é_primo:
6             é_primo = (numero % divisor != 0)
7             divisor += 1
8         if é_primo:
9             print(numero)
```

- Nome da função dá uma pista do que ela faz
- Nomes das variáveis dão pistas do que são
- Linha 6 é mais fácil de ler

Outra versão

```
1 def imprime_primos(n):
2     for numero in range(3, n, 2):
3         é_primo = True
4         divisor = 3
5         while divisor < numero and é_primo:
6             é_primo = (numero % divisor != 0)
7             divisor += 1
8         if é_primo:
9             print(numero)
```

- Nome da função dá uma pista do que ela faz
- Nomes das variáveis dão pistas do que são
- Linha 6 é mais fácil de ler
  - Pode ficar ainda mais fácil com um `if`

Outra versão

```
1 def imprime_primos(n):
2     for numero in range(3, n, 2):
3         é_primo = True
4         divisor = 3
5         while divisor < numero and é_primo:
6             é_primo = (numero % divisor != 0)
7             divisor += 1
8         if é_primo:
9             print(numero)
```

- Nome da função dá uma pista do que ela faz
- Nomes das variáveis dão pistas do que são
- Linha 6 é mais fácil de ler
  - Pode ficar ainda mais fácil com um `if`
- Bons nomes já ajudam a documentar o código

Outra versão

```
1 def imprime_primos(n):
2     for numero in range(3, n, 2):
3         é_primo = True
4         divisor = 3
5         while divisor < numero and é_primo:
6             é_primo = (numero % divisor != 0)
7             divisor += 1
8         if é_primo:
9             print(numero)
```

- Nome da função dá uma pista do que ela faz
- Nomes das variáveis dão pistas do que são
- Linha 6 é mais fácil de ler
  - Pode ficar ainda mais fácil com um `if`
- Bons nomes já ajudam a documentar o código
- Mas poderia ser melhor ainda...

E mais outra versão

```
1  def é_primo(numero):
2      ''' Devolve se o numero dado é primo ou não.'''
3      divisor = 3
4      while divisor < numero:
5          if numero % divisor == 0:
6              return False
7          divisor += 1
8      return True
9
10
11 def imprime_primos(n):
12     ''' Imprime os primos menores ou iguais a n.'''
13     for numero in range(3, n, 2):
14         if é_primo(numero):
15             print(numero)
```

E mais outra versão

```
1  def é_primo(numero):
2      ''' Devolve se o numero dado é primo ou não.'''
3      divisor = 3
4      while divisor < numero:
5          if numero % divisor == 0:
6              return False
7          divisor += 1
8      return True
9
10
11 def imprime_primos(n):
12     ''' Imprime os primos menores ou iguais a n.'''
13     for numero in range(3, n, 2):
14         if é_primo(numero):
15             print(numero)
```

- Temos uma função para saber se é primo

E mais outra versão

```
1  def é_primo(numero):
2      ''' Devolve se o numero dado é primo ou não.'''
3      divisor = 3
4      while divisor < numero:
5          if numero % divisor == 0:
6              return False
7          divisor += 1
8      return True
9
10
11 def imprime_primos(n):
12     ''' Imprime os primos menores ou iguais a n.'''
13     for numero in range(3, n, 2):
14         if é_primo(numero):
15             print(numero)
```

- Temos uma função para saber se é primo
- `imprime_primos` fica mais fácil de entender

E mais outra versão

```
1  def é_primo(numero):
2      ''' Devolve se o numero dado é primo ou não.'''
3      divisor = 3
4      while divisor < numero:
5          if numero % divisor == 0:
6              return False
7          divisor += 1
8      return True
9
10
11 def imprime_primos(n):
12     ''' Imprime os primos menores ou iguais a n.'''
13     for numero in range(3, n, 2):
14         if é_primo(numero):
15             print(numero)
```

- Temos uma função para saber se é primo
- `imprime_primos` fica mais fácil de entender
- A docstring também ajuda

E mais outra versão

```
1  def é_primo(numero):
2      ''' Devolve se o numero dado é primo ou não.'''
3      divisor = 3
4      while divisor < numero:
5          if numero % divisor == 0:
6              return False
7          divisor += 1
8      return True
9
10
11 def imprime_primos(n):
12     ''' Imprime os primos menores ou iguais a n.'''
13     for numero in range(3, n, 2):
14         if é_primo(numero):
15             print(numero)
```

- Temos uma função para saber se é primo
- `imprime_primos` fica mais fácil de entender
- A docstring também ajuda
  - Não são os `n` primeiros primos

E mais outra versão

```
1  def é_primo(numero):
2      ''' Devolve se o numero dado é primo ou não.'''
3      divisor = 3
4      while divisor < numero:
5          if numero % divisor == 0:
6              return False
7          divisor += 1
8      return True
9
10
11 def imprime_primos(n):
12     ''' Imprime os primos menores ou iguais a n.'''
13     for numero in range(3, n, 2):
14         if é_primo(numero):
15             print(numero)
```

- Temos uma função para saber se é primo
- `imprime_primos` fica mais fácil de entender
- A docstring também ajuda
  - Não são os `n` primeiros primos
  - Mas sim os menores ou iguais a `n`

Tipagem Dinâmica

Python é uma linguagem de tipagem dinâmica

Tipagem Dinâmica

Python é uma linguagem de tipagem dinâmica

- Uma variável pode guardar qualquer tipo

Tipagem Dinâmica

Python é uma linguagem de tipagem dinâmica

- Uma variável pode guardar qualquer tipo

Ex:

```
1 x = 10
2 x = str(x) + '!'
3 print(x)          # imprime 10!
```

Tipagem Dinâmica

Python é uma linguagem de tipagem dinâmica

- Uma variável pode guardar qualquer tipo

Ex:

```
1 x = 10
2 x = str(x) + '!'
3 print(x)          # imprime 10!
```

Isso permite algumas coisas interessantes

```
1 def soma(x, y):
2     return x + y
3
4
5 print(soma(2, 3))       # imprime 5
6 print(soma('a', 'b'))   # imprime ab
```

Tipagem Dinâmica

Python é uma linguagem de tipagem dinâmica

- Uma variável pode guardar qualquer tipo

Ex:

```
1 x = 10
2 x = str(x) + '!'
3 print(x)          # imprime 10!
```

Isso permite algumas coisas interessantes

```
1 def soma(x, y):
2     return x + y
3
4
5 print(soma(2, 3))       # imprime 5
6 print(soma('a', 'b'))   # imprime ab
```

Porém, algumas checagens de bugs são perdidas com isso

Tipagem Dinâmica

Python é uma linguagem de tipagem dinâmica

- Uma variável pode guardar qualquer tipo

Ex:

```
1 x = 10
2 x = str(x) + '!'
3 print(x)          # imprime 10!
```

Isso permite algumas coisas interessantes

```
1 def soma(x, y):
2     return x + y
3
4
5 print(soma(2, 3))       # imprime 5
6 print(soma('a', 'b'))   # imprime ab
```

Porém, algumas checagens de bugs são perdidas com isso

- E o código pode ser mais difícil de entender

Dica de Tipo — Type Hinting

Veja o código abaixo

```
1  def é_primo(numero: int) -> bool:
2      ''' Devolve se o numero dado é primo ou não.'''
3      divisor: int = 3
4      while divisor < numero:
5          if numero % divisor == 0:
6              return False
7          divisor += 1
8      return True
9
10
11 def imprime_primos(n: int) -> None:
12     ''' Imprime os primos menores ou iguais a n.'''
13     for numero in range(3, n, 2):
14         if é_primo(numero):
15             print(numero)
```

Dica de Tipo — Type Hinting

Veja o código abaixo

```
1  def é_primo(numero: int) -> bool:
2      ''' Devolve se o numero dado é primo ou não.'''
3      divisor: int = 3
4      while divisor < numero:
5          if numero % divisor == 0:
6              return False
7          divisor += 1
8      return True
9
10
11 def imprime_primos(n: int) -> None:
12     ''' Imprime os primos menores ou iguais a n.'''
13     for numero in range(3, n, 2):
14         if é_primo(numero):
15             print(numero)
```

Damos algumas dicas dos tipos esperados das variáveis

Dica de Tipo — Type Hinting

Veja o código abaixo

```
1  def é_primo(numero: int) -> bool:
2      ''' Devolve se o numero dado é primo ou não.'''
3      divisor: int = 3
4      while divisor < numero:
5          if numero % divisor == 0:
6              return False
7          divisor += 1
8      return True
9
10
11 def imprime_primos(n: int) -> None:
12     ''' Imprime os primos menores ou iguais a n.'''
13     for numero in range(3, n, 2):
14         if é_primo(numero):
15             print(numero)
```

Damos algumas dicas dos tipos esperados das variáveis

- O `:` nos dá a dica do tipo esperado

Dica de Tipo — Type Hinting

Veja o código abaixo

```
1  def é_primo(numero: int) -> bool:
2      ''' Devolve se o numero dado é primo ou não.'''
3      divisor: int = 3
4      while divisor < numero:
5          if numero % divisor == 0:
6              return False
7          divisor += 1
8      return True
9
10
11 def imprime_primos(n: int) -> None:
12     ''' Imprime os primos menores ou iguais a n.'''
13     for numero in range(3, n, 2):
14         if é_primo(numero):
15             print(numero)
```

Damos algumas dicas dos tipos esperados das variáveis

- O `:` nos dá a dica do tipo esperado
- O `->` diz o tipo de retorno da função

Dica de Tipo — Type Hinting

Veja o código abaixo

```
1  def é_primo(numero: int) -> bool:
2      ''' Devolve se o numero dado é primo ou não.'''
3      divisor: int = 3
4      while divisor < numero:
5          if numero % divisor == 0:
6              return False
7          divisor += 1
8      return True
9
10
11 def imprime_primos(n: int) -> None:
12     ''' Imprime os primos menores ou iguais a n.'''
13     for numero in range(3, n, 2):
14         if é_primo(numero):
15             print(numero)
```

Damos algumas dicas dos tipos esperados das variáveis

- O `:` nos dá a dica do tipo esperado
- O `->` diz o tipo de retorno da função

E o que fazer com isso?

Erro de Execução por causa de tipo

Olhe esse código:

Erro de Execução por causa de tipo

Olhe esse código:

```
1 from datetime import datetime
2
3 ...
4
5 if datetime.now().hour == 3:
6     imprime_primos(9.5)
7 else:
8     imprime_primos(9)
```

Erro de Execução por causa de tipo

Olhe esse código:

```
1 from datetime import datetime
2
3 ...
4
5 if datetime.now().hour == 3:
6     imprime_primos(9.5)
7 else:
8     imprime_primos(9)
```

Se não for 3 da manhã, ele roda sem erros...

Erro de Execução por causa de tipo

Olhe esse código:

```
1 from datetime import datetime
2
3 ...
4
5 if datetime.now().hour == 3:
6     imprime_primos(9.5)
7 else:
8     imprime_primos(9)
```

Se não for 3 da manhã, ele roda sem erros...

Mas às 3 da manhã ele dá o seguinte erro:

Erro de Execução por causa de tipo

Olhe esse código:

```
1 from datetime import datetime
2
3 ...
4
5 if datetime.now().hour == 3:
6     imprime_primos(9.5)
7 else:
8     imprime_primos(9)
```

Se não for 3 da manhã, ele roda sem erros...

Mas às 3 da manhã ele dá o seguinte erro:

```
Traceback (most recent call last):
  File "/Users/schouery/ruim2.py", line 22, in <module>
    imprime_primos(9.5)
  File "/Users/schouery/ruim2.py", line 16, in imprime_primos
    for numero in range(3, n, 2):
TypeError: 'float' object cannot be interpreted as an integer
```

Erros de Execução

Dependendo do caminho percorrido no código podemos ter um erro de execução

Erros de Execução

Dependendo do caminho percorrido no código podemos ter um erro de execução

- O difícil é que alguns caminhos podem ser raros

Erros de Execução

Dependendo do caminho percorrido no código podemos ter um erro de execução

- O difícil é que alguns caminhos podem ser raros

Linguagens estaticamente tipadas conseguem evitar alguns desses erros

Erros de Execução

Dependendo do caminho percorrido no código podemos ter um erro de execução

- O difícil é que alguns caminhos podem ser raros

Linguagens estaticamente tipadas conseguem evitar alguns desses erros

- Nós sabemos exatamente o tipo da variável

Erros de Execução

Dependendo do caminho percorrido no código podemos ter um erro de execução

- O difícil é que alguns caminhos podem ser raros

Linguagens estaticamente tipadas conseguem evitar alguns desses erros

- Nós sabemos exatamente o tipo da variável
- Então sabemos que ela não pode assumir certos valores

Erros de Execução

Dependendo do caminho percorrido no código podemos ter um erro de execução

- O difícil é que alguns caminhos podem ser raros

Linguagens estaticamente tipadas conseguem evitar alguns desses erros

- Nós sabemos exatamente o tipo da variável
- Então sabemos que ela não pode assumir certos valores

Na linguagem C, eu detectaria esse erro muito antes

Erros de Execução

Dependendo do caminho percorrido no código podemos ter um erro de execução

- O difícil é que alguns caminhos podem ser raros

Linguagens estaticamente tipadas conseguem evitar alguns desses erros

- Nós sabemos exatamente o tipo da variável
- Então sabemos que ela não pode assumir certos valores

Na linguagem C, eu detectaria esse erro muito antes

- Um `int` não pode assumir o valor `9.5`

Erros de Execução

Dependendo do caminho percorrido no código podemos ter um erro de execução

- O difícil é que alguns caminhos podem ser raros

Linguagens estaticamente tipadas conseguem evitar alguns desses erros

- Nós sabemos exatamente o tipo da variável
- Então sabemos que ela não pode assumir certos valores

Na linguagem C, eu detectaria esse erro muito antes

- Um `int` não pode assumir o valor `9.5`
- Eu teria um erro de compilação

Erros de Execução

Dependendo do caminho percorrido no código podemos ter um erro de execução

- O difícil é que alguns caminhos podem ser raros

Linguagens estaticamente tipadas conseguem evitar alguns desses erros

- Nós sabemos exatamente o tipo da variável
- Então sabemos que ela não pode assumir certos valores

Na linguagem C, eu detectaria esse erro muito antes

- Um `int` não pode assumir o valor `9.5`
- Eu teria um erro de compilação
- Independente do caminho a ser percorrido na execução

MyPy

O MyPy é um programa que faz checagem de tipo para Python

MyPy

O MyPy é um programa que faz checagem de tipo para Python

- Baseada nas dicas de tipo dadas

MyPy

O MyPy é um programa que faz checagem de tipo para Python

- Baseada nas dicas de tipo dadas

```
ruim2.py:22: error: Argument 1 to "imprime_primos" has
    incompatible type "float"; expected "int"
Found 1 error in 1 file (checked 1 source file)
```

MyPy

O MyPy é um programa que faz checagem de tipo para Python

- Baseada nas dicas de tipo dadas

```
ruim2.py:22: error: Argument 1 to "imprime_primos" has
    incompatible type "float"; expected "int"
Found 1 error in 1 file (checked 1 source file)
```

Esse erro encontrado não depende da execução do código

MyPy

Vejamos o seguinte código

```
1 def soma_inteiros(x: int, y: int) -> int:
2     return x + y
3
4
5 print(soma_inteiros(2, 3))
6 print(soma_inteiros('a', 'b'))
```

MyPy

Vejamos o seguinte código

```
1 def soma_inteiros(x: int, y: int) -> int:
2     return x + y
3
4
5 print(soma_inteiros(2, 3))
6 print(soma_inteiros('a', 'b'))
```

O MyPy encontra o seguinte erro:

```
mypy1.py:6: error: Argument 1 to "soma_inteiros" has
    incompatible type "str"; expected "int"
mypy1.py:6: error: Argument 2 to "soma_inteiros" has
    incompatible type "str"; expected "int"
Found 2 errors in 1 file (checked 1 source file)
```

MyPy

Vejamos o seguinte código

```
1 def soma_inteiros(x: int, y: int) -> int:
2     return x + y
3
4
5 print(soma_inteiros(2, 3))
6 print(soma_inteiros('a', 'b'))
```

O MyPy encontra o seguinte erro:

```
mypy1.py:6: error: Argument 1 to "soma_inteiros" has
    incompatible type "str"; expected "int"
mypy1.py:6: error: Argument 2 to "soma_inteiros" has
    incompatible type "str"; expected "int"
Found 2 errors in 1 file (checked 1 source file)
```

Mas o código roda no Python sem erros!

MyPy

Vejamos o seguinte código

```
1 def soma_inteiros(x: int, y: int) -> int:
2     return x + y
3
4
5 print(soma_inteiros(2, 3))
6 print(soma_inteiros('a', 'b'))
```

O MyPy encontra o seguinte erro:

```
mypy1.py:6: error: Argument 1 to "soma_inteiros" has
    incompatible type "str"; expected "int"
mypy1.py:6: error: Argument 2 to "soma_inteiros" has
    incompatible type "str"; expected "int"
Found 2 errors in 1 file (checked 1 source file)
```

Mas o código roda no Python sem erros!

- A dica de tipo é só uma dica...

MyPy

Vejamos o seguinte código

```
1 def soma_inteiros(x: int, y: int) -> int:
2     return x + y
3
4
5 print(soma_inteiros(2, 3))
6 print(soma_inteiros('a', 'b'))
```

O MyPy encontra o seguinte erro:

```
mypy1.py:6: error: Argument 1 to "soma_inteiros" has
    incompatible type "str"; expected "int"
mypy1.py:6: error: Argument 2 to "soma_inteiros" has
    incompatible type "str"; expected "int"
Found 2 errors in 1 file (checked 1 source file)
```

Mas o código roda no Python sem erros!

- A dica de tipo é só uma dica...
- O Python não deixa de ser dinamicamente tipado

O que eu não estou te contando?

A biblioteca `typing` dá suporte as dicas

O que eu não estou te contando?

A biblioteca `typing` dá suporte as dicas

- Tem várias coisas para aprender lá

O que eu não estou te contando?

A biblioteca `typing` dá suporte as dicas

- Tem várias coisas para aprender lá
- E depende da versão do Python

O que eu não estou te contando?

A biblioteca `typing` dá suporte as dicas

- Tem várias coisas para aprender lá
- E depende da versão do Python

Você precisa instalar o MyPy:

O que eu não estou te contando?

A biblioteca `typing` dá suporte as dicas

- Tem várias coisas para aprender lá
- E depende da versão do Python

Você precisa instalar o MyPy:

- `pip install mypy`

O que eu não estou te contando?

A biblioteca `typing` dá suporte as dicas

- Tem várias coisas para aprender lá
- E depende da versão do Python

Você precisa instalar o MyPy:

- `pip install mypy`

Você pode executar o MyPy fazendo:

O que eu não estou te contando?

A biblioteca `typing` dá suporte as dicas

- Tem várias coisas para aprender lá
- E depende da versão do Python

Você precisa instalar o MyPy:

- `pip install mypy`

Você pode executar o MyPy fazendo:

- `mypy meuarquivo.py`

O que eu não estou te contando?

A biblioteca `typing` dá suporte as dicas

- Tem várias coisas para aprender lá
- E depende da versão do Python

Você precisa instalar o MyPy:

- `pip install mypy`

Você pode executar o MyPy fazendo:

- `mypy meuarquivo.py`
- `mypy minhapasta`

O que eu não estou te contando?

A biblioteca `typing` dá suporte as dicas

- Tem várias coisas para aprender lá
- E depende da versão do Python

Você precisa instalar o MyPy:

- `pip install mypy`

Você pode executar o MyPy fazendo:

- `mypy meuarquivo.py`
- `mypy minhapasta`
- `mypy .`

É bonito, mas faz o que promete?

```
1  def é_primo(numero: int) -> bool:
2      ''' Devolve se o numero dado é primo ou não.'''
3      divisor: int = 3
4      while divisor < numero:
5          if numero % divisor == 0:
6              return False
7          divisor += 1
8      return True
9
10
11 def imprime_primos(n: int) -> None:
12     ''' Imprime os primos menores ou iguais a n.'''
13     for numero in range(3, n, 2):
14         if é_primo(numero):
15             print(numero)
```

É bonito, mas faz o que promete?

```
1  def é_primo(numero: int) -> bool:
2      ''' Devolve se o numero dado é primo ou não.'''
3      divisor: int = 3
4      while divisor < numero:
5          if numero % divisor == 0:
6              return False
7          divisor += 1
8      return True
9
10
11 def imprime_primos(n: int) -> None:
12     ''' Imprime os primos menores ou iguais a n.'''
13     for numero in range(3, n, 2):
14         if é_primo(numero):
15             print(numero)
```

Meu código está certo?

É bonito, mas faz o que promete?

```
1  def é_primo(numero: int) -> bool:
2      ''' Devolve se o numero dado é primo ou não.'''
3      divisor: int = 3
4      while divisor < numero:
5          if numero % divisor == 0:
6              return False
7          divisor += 1
8      return True
9
10
11 def imprime_primos(n: int) -> None:
12     ''' Imprime os primos menores ou iguais a n.'''
13     for numero in range(3, n, 2):
14         if é_primo(numero):
15             print(numero)
```

Meu código está certo?

- Preciso saber o algoritmo

É bonito, mas faz o que promete?

```
1  def é_primo(numero: int) -> bool:
2      ''' Devolve se o numero dado é primo ou não.'''
3      divisor: int = 3
4      while divisor < numero:
5          if numero % divisor == 0:
6              return False
7          divisor += 1
8      return True
9
10
11 def imprime_primos(n: int) -> None:
12     ''' Imprime os primos menores ou iguais a n.'''
13     for numero in range(3, n, 2):
14         if é_primo(numero):
15             print(numero)
```

Meu código está certo?

- Preciso saber o algoritmo
  - O que é divisor, primo, como verifica primalidade, etc.

É bonito, mas faz o que promete?

```
1  def é_primo(numero: int) -> bool:
2      ''' Devolve se o numero dado é primo ou não.'''
3      divisor: int = 3
4      while divisor < numero:
5          if numero % divisor == 0:
6              return False
7          divisor += 1
8      return True
9
10
11 def imprime_primos(n: int) -> None:
12     ''' Imprime os primos menores ou iguais a n.'''
13     for numero in range(3, n, 2):
14         if é_primo(numero):
15             print(numero)
```

Meu código está certo?

- Preciso saber o algoritmo
  - O que é divisor, primo, como verifica primalidade, etc.
- Depois preciso saber se o código implementa o algoritmo

Testando

Testar aumenta a confiança que o código está correto

Testando

Testar aumenta a confiança que o código está correto

```
1  def é_primo(numero: int) -> bool:
2      ''' Devolve se o numero dado é primo ou não.'''
3      divisor: int = 3  # errado propositalmente
4      while divisor < numero:
5          if numero % divisor == 0:
6              return False
7          divisor += 1
8      return True
9
10
11 assert é_primo(2) is True
12 assert é_primo(3) is True
13 assert é_primo(4) is False
14 assert é_primo(17) is True
15 assert é_primo(42) is False
```

Testando

Testar aumenta a confiança que o código está correto

```
1  def é_primo(numero: int) -> bool:
2      ''' Devolve se o numero dado é primo ou não.'''
3      divisor: int = 3  # errado propositalmente
4      while divisor < numero:
5          if numero % divisor == 0:
6              return False
7          divisor += 1
8      return True
9
10
11 assert é_primo(2) is True
12 assert é_primo(3) is True
13 assert é_primo(4) is False
14 assert é_primo(17) is True
15 assert é_primo(42) is False
```

O resultado me ajuda a achar o erro:

Testando

Testar aumenta a confiança que o código está correto

```
1  def é_primo(numero: int) -> bool:
2      ''' Devolve se o numero dado é primo ou não.'''
3      divisor: int = 3  # errado propositalmente
4      while divisor < numero:
5          if numero % divisor == 0:
6              return False
7          divisor += 1
8      return True
9
10
11 assert é_primo(2) is True
12 assert é_primo(3) is True
13 assert é_primo(4) is False
14 assert é_primo(17) is True
15 assert é_primo(42) is False
```

O resultado me ajuda a achar o erro:

```
Traceback (most recent call last):
  File "/Users/schouery/test1.py", line 13, in <module>
    assert é_primo(4) is False
AssertionError
```

Testes de Unidade

O ruim de usar `assert` é que paramos no primeiro erro

Testes de Unidade

O ruim de usar `assert` é que paramos no primeiro erro

- Nosso código poderia ser grande

Testes de Unidade

O ruim de usar `assert` é que paramos no primeiro erro

- Nosso código poderia ser grande
- E estar em vários arquivos

Testes de Unidade

O ruim de usar `assert` é que paramos no primeiro erro

- Nosso código poderia ser grande
- E estar em vários arquivos
- E ter vários erros...

Testes de Unidade

O ruim de usar `assert` é que paramos no primeiro erro

- Nosso código poderia ser grande
- E estar em vários arquivos
- E ter vários erros...

Vamos usar algo melhor — `pytest`

Nosso primeiro teste

Esse é o arquivo `test_primo.py`

```
1  from primos import é_primo
2
3
4  def test_é_primo():
5      primos = [2, 3, 5, 7, 11, 13, 17]
6      for p in primos:
7          assert é_primo(p) is True
8      compostos = [4, 6, 8, 9, 10, 12, 14, 15, 16]
9      for c in compostos:
10         assert é_primo(c) is False
```

Nosso primeiro teste

Esse é o arquivo `test_primo.py`

```
1  from primos import é_primo
2
3
4  def test_é_primo():
5      primos = [2, 3, 5, 7, 11, 13, 17]
6      for p in primos:
7          assert é_primo(p) is True
8      compostos = [4, 6, 8, 9, 10, 12, 14, 15, 16]
9      for c in compostos:
10         assert é_primo(c) is False
```

Para rodar o teste, executamos `pytest` no terminal

Nosso primeiro teste

Esse é o arquivo `test_primo.py`

```
1  from primos import é_primo
2
3
4  def test_é_primo():
5      primos = [2, 3, 5, 7, 11, 13, 17]
6      for p in primos:
7          assert é_primo(p) is True
8      compostos = [4, 6, 8, 9, 10, 12, 14, 15, 16]
9      for c in compostos:
10         assert é_primo(c) is False
```

Para rodar o teste, executamos `pytest` no terminal

- Executa todos os arquivos que comecem com `test_`

Nosso primeiro teste

Esse é o arquivo `test_primo.py`

```
1  from primos import é_primo
2
3
4  def test_é_primo():
5      primos = [2, 3, 5, 7, 11, 13, 17]
6      for p in primos:
7          assert é_primo(p) is True
8      compostos = [4, 6, 8, 9, 10, 12, 14, 15, 16]
9      for c in compostos:
10         assert é_primo(c) is False
```

Para rodar o teste, executamos `pytest` no terminal

- Executa todos os arquivos que comecem com `test_`
- Considera que cada função começando `test_` é um teste

pytest

Resultado:

```
============================= test session starts ==============================
platform darwin -- Python 3.9.10, pytest-6.2.5, py-1.11.0, pluggy-1.0.0
rootdir: /Users/schouery/repos/mc102/codigo_bom/codigos
plugins: Faker-8.10.1
collected 1 item

test_primo.py F                                                          [100%]

=================================== FAILURES ===================================
_______________________________ test_é_primo ________________________________

    def test_é_primo():
        primos = [2, 3, 5, 7, 11, 13, 17]
        for p in primos:
            assert é_primo(p) is True
        compostos = [4, 6, 8, 9, 10, 12, 14, 15, 16]
        for c in compostos:
>           assert é_primo(c) is False
E           assert True is False
E            +  where True = é_primo(4)

test_primo.py:10: AssertionError
=========================== short test summary info ============================
FAILED test_primo.py::test_é_primo - assert True is False
============================== 1 failed in 0.05s ===============================
```

pytest

Resultado:

```
============================= test session starts ==============================
platform darwin -- Python 3.9.10, pytest-6.2.5, py-1.11.0, pluggy-1.0.0
rootdir: /Users/schouery/repos/mc102/codigo_bom/codigos
plugins: Faker-8.10.1
collected 1 item

test_primo.py F                                                          [100%]

=================================== FAILURES ===================================
_______________________________ test_é_primo ________________________________

    def test_é_primo():
        primos = [2, 3, 5, 7, 11, 13, 17]
        for p in primos:
            assert é_primo(p) is True
        compostos = [4, 6, 8, 9, 10, 12, 14, 15, 16]
        for c in compostos:
>           assert é_primo(c) is False
E           assert True is False
E            +  where True = é_primo(4)

test_primo.py:10: AssertionError
=========================== short test summary info ============================
FAILED test_primo.py::test_é_primo - assert True is False
============================== 1 failed in 0.05s ===============================
```

Meu código está errado pois está dizendo que 4 é primo

Voltando ao código

```
1 def é_primo(numero: int) -> bool:
2     ''' Devolve se o numero dado é primo ou não.'''
3     divisor: int = 3
4     while divisor < numero:
5         if numero % divisor == 0:
6             return False
7         divisor += 1
8     return True
```

Voltando ao código

```
1 def é_primo(numero: int) -> bool:
2     ''' Devolve se o numero dado é primo ou não.'''
3     divisor: int = 3
4     while divisor < numero:
5         if numero % divisor == 0:
6             return False
7         divisor += 1
8     return True
```

O problema está na linha 3...

Voltando ao código

```
1 def é_primo(numero: int) -> bool:
2     ''' Devolve se o numero dado é primo ou não.'''
3     divisor: int = 3
4     while divisor < numero:
5         if numero % divisor == 0:
6             return False
7         divisor += 1
8     return True
```

O problema está na linha 3...

- Eu começo `divisor` com `3`

Voltando ao código

```
1 def é_primo(numero: int) -> bool:
2     ''' Devolve se o numero dado é primo ou não.'''
3     divisor: int = 3
4     while divisor < numero:
5         if numero % divisor == 0:
6             return False
7         divisor += 1
8     return True
```

O problema está na linha 3...

- Eu começo `divisor` com `3`
- Assim, eu devolvo `True` para toda potência de 2

Corrigindo

Também alterei a função de impressão

```
1  def é_primo(numero: int) -> bool:
2      ''' Devolve se o numero dado é primo ou não.'''
3      divisor: int = 2  # corrigido
4      while divisor < numero:
5          if numero % divisor == 0:
6              return False
7          divisor += 1
8      return True
9
10
11 def primos_até_n(n: int) -> list[int]:
12     ''' Devolve a lista dos primos menores ou iguais a n.'''
13     lista: list[int] = []
14     for numero in range(3, n, 2):
15         if é_primo(numero):
16             lista.append(numero)
17     return lista
18
19
20 def imprime_primos(n: int) -> None:
21     ''' Imprime os primos menores ou iguais a n.'''
22     for numero in primos_até_n(n):
23         print(numero)
```

Aumentando o teste

```
1  from primos import é_primo, primos_até_n
2
3
4  def test_é_primo():
5      primos = [2, 3, 5, 7, 11, 13, 17]
6      for p in primos:
7          assert é_primo(p) is True
8      compostos = [4, 6, 8, 9, 10, 12, 14, 15, 16]
9      for c in compostos:
10         assert é_primo(c) is False
11
12
13 def test_primos_até_n():
14     primos = [2, 3, 5, 7, 11, 13, 17]
15     assert primos_até_n(17) == primos
```

Resultado

```
============================= test session starts ==============================
platform darwin -- Python 3.9.10, pytest-6.2.5, py-1.11.0, pluggy-1.0.0
rootdir: /Users/schouery/repos/mc102/codigo_bom/codigos
plugins: Faker-8.10.1
collected 2 items

test_primo.py .F                                                         [100%]

=================================== FAILURES ===================================
_____________________________ test_primos_até_n ______________________________

    def test_primos_até_n():
        primos = [2, 3, 5, 7, 11, 13, 17]
>       assert primos_até_n(17) == primos
E       assert [3, 5, 7, 11, 13] == [2, 3, 5, 7, 11, 13, ...]
E         At index 0 diff: 3 != 2
E         Right contains 2 more items, first extra item: 13
E         Use -v to get the full diff

test_primo.py:15: AssertionError
=========================== short test summary info ============================
FAILED test_primo.py::test_primos_até_n - assert [3, 5, 7, 11, 13] == [2, 3, ...
========================= 1 failed, 1 passed in 0.06s ==========================
```

Resultado

```
============================= test session starts ==============================
platform darwin -- Python 3.9.10, pytest-6.2.5, py-1.11.0, pluggy-1.0.0
rootdir: /Users/schouery/repos/mc102/codigo_bom/codigos
plugins: Faker-8.10.1
collected 2 items

test_primo.py .F                                                         [100%]

=================================== FAILURES ===================================
_____________________________ test_primos_até_n ______________________________

    def test_primos_até_n():
        primos = [2, 3, 5, 7, 11, 13, 17]
>       assert primos_até_n(17) == primos
E       assert [3, 5, 7, 11, 13] == [2, 3, 5, 7, 11, 13, ...]
E         At index 0 diff: 3 != 2
E         Right contains 2 more items, first extra item: 13
E         Use -v to get the full diff

test_primo.py:15: AssertionError
=========================== short test summary info ============================
FAILED test_primo.py::test_primos_até_n - assert [3, 5, 7, 11, 13] == [2, 3, ...
========================= 1 failed, 1 passed in 0.06s ==========================
```

Ele não encontrou 2 e 17 na lista

Voltando ao código

```
1  def é_primo(numero: int) -> bool:
2      ''' Devolve se o numero dado é primo ou não.'''
3      divisor: int = 2
4      while divisor < numero:
5          if numero % divisor == 0:
6              return False
7          divisor += 1
8      return True
9
10
11 def primos_até_n(n: int) -> list[int]:
12     ''' Devolve a lista dos primos menores ou iguais a n.'''
13     lista: list[int] = []
14     for numero in range(2, n + 1):  # corrigido
15         if é_primo(numero):
16             lista.append(numero)
17     return lista
```

Voltando ao código

```
1  def é_primo(numero: int) -> bool:
2      ''' Devolve se o numero dado é primo ou não.'''
3      divisor: int = 2
4      while divisor < numero:
5          if numero % divisor == 0:
6              return False
7          divisor += 1
8      return True
9
10
11 def primos_até_n(n: int) -> list[int]:
12     ''' Devolve a lista dos primos menores ou iguais a n.'''
13     lista: list[int] = []
14     for numero in range(2, n + 1):  # corrigido
15         if é_primo(numero):
16             lista.append(numero)
17     return lista
```

Agora sim o código está correto!

Voltando ao código

```
1  def é_primo(numero: int) -> bool:
2      ''' Devolve se o numero dado é primo ou não.'''
3      divisor: int = 2
4      while divisor < numero:
5          if numero % divisor == 0:
6              return False
7          divisor += 1
8      return True
9
10
11 def primos_até_n(n: int) -> list[int]:
12     ''' Devolve a lista dos primos menores ou iguais a n.'''
13     lista: list[int] = []
14     for numero in range(2, n + 1):  # corrigido
15         if é_primo(numero):
16             lista.append(numero)
17     return lista
```

Agora sim o código está correto!

- Meus testes passam!

Voltando ao código

```
1  def é_primo(numero: int) -> bool:
2      ''' Devolve se o numero dado é primo ou não.'''
3      divisor: int = 2
4      while divisor < numero:
5          if numero % divisor == 0:
6              return False
7          divisor += 1
8      return True
9
10
11 def primos_até_n(n: int) -> list[int]:
12     ''' Devolve a lista dos primos menores ou iguais a n.'''
13     lista: list[int] = []
14     for numero in range(2, n + 1):  # corrigido
15         if é_primo(numero):
16             lista.append(numero)
17     return lista
```

Agora sim o código está correto!

- Meus testes passam!
- Mas será mesmo que está correto?

Voltando ao código

```
1  def é_primo(numero: int) -> bool:
2      ''' Devolve se o numero dado é primo ou não.'''
3      divisor: int = 2
4      while divisor < numero:
5          if numero % divisor == 0:
6              return False
7          divisor += 1
8      return True
9
10
11 def primos_até_n(n: int) -> list[int]:
12     ''' Devolve a lista dos primos menores ou iguais a n.'''
13     lista: list[int] = []
14     for numero in range(2, n + 1):  # corrigido
15         if é_primo(numero):
16             lista.append(numero)
17     return lista
```

Agora sim o código está correto!

- Meus testes passam!
- Mas será mesmo que está correto?
- Qual o resultado de `é_primo(1)`? e `é_primo(-4)`?

Versão Final

```python
def é_primo(numero: int) -> bool:
    ''' Devolve se o numero dado é primo ou não.'''
    if numero <= 1:  # negativos, 0 e 1 não são primos
        return False
    divisor: int = 2
    while divisor < numero:
        if numero % divisor == 0:
            return False
        divisor += 1
    return True


def primos_até_n(n: int) -> list[int]:
    ''' Devolve a lista dos primos menores ou iguais a n.'''
    lista: list[int] = []
    for numero in range(2, n + 1):
        if é_primo(numero):
            lista.append(numero)
    return lista
```

Versão Final

```python
def é_primo(numero: int) -> bool:
    ''' Devolve se o numero dado é primo ou não.'''
    if numero <= 1:  # negativos, 0 e 1 não são primos
        return False
    divisor: int = 2
    while divisor < numero:
        if numero % divisor == 0:
            return False
        divisor += 1
    return True


def primos_até_n(n: int) -> list[int]:
    ''' Devolve a lista dos primos menores ou iguais a n.'''
    lista: list[int] = []
    for numero in range(2, n + 1):
        if é_primo(numero):
            lista.append(numero)
    return lista
```

Finalmente um código bom!

Versão Final

```python
def é_primo(numero: int) -> bool:
    ''' Devolve se o numero dado é primo ou não.'''
    if numero <= 1:  # negativos, 0 e 1 não são primos
        return False
    divisor: int = 2
    while divisor < numero:
        if numero % divisor == 0:
            return False
        divisor += 1
    return True


def primos_até_n(n: int) -> list[int]:
    ''' Devolve a lista dos primos menores ou iguais a n.'''
    lista: list[int] = []
    for numero in range(2, n + 1):
        if é_primo(numero):
            lista.append(numero)
    return lista
```

Finalmente um código bom!

Versão Final

```python
def é_primo(numero: int) -> bool:
    ''' Devolve se o numero dado é primo ou não.'''
    if numero <= 1:  # negativos, 0 e 1 não são primos
        return False
    divisor: int = 2
    while divisor < numero:
        if numero % divisor == 0:
            return False
        divisor += 1
    return True


def primos_até_n(n: int) -> list[int]:
    ''' Devolve a lista dos primos menores ou iguais a n.'''
    lista: list[int] = []
    for numero in range(2, n + 1):
        if é_primo(numero):
            lista.append(numero)
    return lista
```

Finalmente um código bom! Será?

O que eu não estou te contando

Testes é uma disciplina

O que eu não estou te contando

Testes é uma disciplina

- Existem tipos diferentes de teste

O que eu não estou te contando

Testes é uma disciplina

- Existem tipos diferentes de teste
- O que vimos é basicamente teste de unidade

O que eu não estou te contando

Testes é uma disciplina

- Existem tipos diferentes de teste
- O que vimos é basicamente teste de unidade
  - Testamos pequenas partes do nosso código

O que eu não estou te contando

Testes é uma disciplina

- Existem tipos diferentes de teste
- O que vimos é basicamente teste de unidade
  - Testamos pequenas partes do nosso código
- Buscamos ter testes que cubra todo o código

O que eu não estou te contando

Testes é uma disciplina

- Existem tipos diferentes de teste
- O que vimos é basicamente teste de unidade
  - Testamos pequenas partes do nosso código
- Buscamos ter testes que cubra todo o código
  - Tem como verificar isso

O que eu não estou te contando

Testes é uma disciplina

- Existem tipos diferentes de teste
- O que vimos é basicamente teste de unidade
  - Testamos pequenas partes do nosso código
- Buscamos ter testes que cubra todo o código
  - Tem como verificar isso
- É preciso cuidado ao criar os testes

O que eu não estou te contando

Testes é uma disciplina

- Existem tipos diferentes de teste
- O que vimos é basicamente teste de unidade
  - Testamos pequenas partes do nosso código
- Buscamos ter testes que cubra todo o código
  - Tem como verificar isso
- É preciso cuidado ao criar os testes

Você precisa instalar o pytest

O que eu não estou te contando

Testes é uma disciplina

- Existem tipos diferentes de teste
- O que vimos é basicamente teste de unidade
  - Testamos pequenas partes do nosso código
- Buscamos ter testes que cubra todo o código
  - Tem como verificar isso
- É preciso cuidado ao criar os testes

Você precisa instalar o pytest

- `pip install pytest`

O que eu não estou te contando

Testes é uma disciplina

- Existem tipos diferentes de teste
- O que vimos é basicamente teste de unidade
  - Testamos pequenas partes do nosso código
- Buscamos ter testes que cubra todo o código
  - Tem como verificar isso
- É preciso cuidado ao criar os testes

Você precisa instalar o pytest

- `pip install pytest`

Ele tem outras funcionalidades

O que eu não estou te contando

Testes é uma disciplina

- Existem tipos diferentes de teste
- O que vimos é basicamente teste de unidade
  - Testamos pequenas partes do nosso código
- Buscamos ter testes que cubra todo o código
  - Tem como verificar isso
- É preciso cuidado ao criar os testes

Você precisa instalar o pytest

- `pip install pytest`

Ele tem outras funcionalidades

- E existem concorrentes como o `unittest`

O que eu não estou te contando

Testes é uma disciplina

- Existem tipos diferentes de teste
- O que vimos é basicamente teste de unidade
  - Testamos pequenas partes do nosso código
- Buscamos ter testes que cubra todo o código
  - Tem como verificar isso
- É preciso cuidado ao criar os testes

Você precisa instalar o pytest

- `pip install pytest`

Ele tem outras funcionalidades

- E existem concorrentes como o `unittest`

Em geral há uma estrutura de pastas a ser seguida

Tempo de Execução

Importamos `cProfile` e executamos

```
cProfile.run("primos_até_n(100000)")
```

Tempo de Execução

Importamos `cProfile` e executamos

`cProfile.run("primos_até_n(100000)")`

```
         109595 function calls in 25.820 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000   25.820   25.820 <string>:1(<module>)
        1    0.012    0.012   25.820   25.820 teste_tempo.py:15(primos_até_n)
    99999   25.807    0.000   25.807    0.000 teste_tempo.py:3(é_primo)
        1    0.000    0.000   25.820   25.820 {built-in method builtins.exec}
     9592    0.001    0.000    0.001    0.000 {method 'append' of 'list' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler'
      objects}
```

Tempo de Execução

Importamos `cProfile` e executamos
`cProfile.run("primos_até_n(100000)")`

```
         109595 function calls in 25.820 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000   25.820   25.820 <string>:1(<module>)
        1    0.012    0.012   25.820   25.820 teste_tempo.py:15(primos_até_n)
    99999   25.807    0.000   25.807    0.000 teste_tempo.py:3(é_primo)
        1    0.000    0.000   25.820   25.820 {built-in method builtins.exec}
     9592    0.001    0.000    0.001    0.000 {method 'append' of 'list' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler'
      objects}
```

Demorou 25 segundos para rodar

Tempo de Execução

Importamos `cProfile` e executamos
`cProfile.run("primos_até_n(100000)")`

```
         109595 function calls in 25.820 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000   25.820   25.820 <string>:1(<module>)
        1    0.012    0.012   25.820   25.820 teste_tempo.py:15(primos_até_n)
    99999   25.807    0.000   25.807    0.000 teste_tempo.py:3(é_primo)
        1    0.000    0.000   25.820   25.820 {built-in method builtins.exec}
     9592    0.001    0.000    0.001    0.000 {method 'append' of 'list' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler'
      objects}
```

Demorou 25 segundos para rodar

- Praticamente todo o tempo é executando `é_primo`

Tempo de Execução

Importamos `cProfile` e executamos
`cProfile.run("primos_até_n(100000)")`

```
         109595 function calls in 25.820 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000   25.820   25.820 <string>:1(<module>)
        1    0.012    0.012   25.820   25.820 teste_tempo.py:15(primos_até_n)
    99999   25.807    0.000   25.807    0.000 teste_tempo.py:3(é_primo)
        1    0.000    0.000   25.820   25.820 {built-in method builtins.exec}
     9592    0.001    0.000    0.001    0.000 {method 'append' of 'list' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler'
      objects}
```

Demorou 25 segundos para rodar

- Praticamente todo o tempo é executando `é_primo`
- Se melhorarmos `é_primo`, melhoramos muito o código

Novo código

```python
import cProfile


def é_primo(numero: int) -> bool:
    ''' Devolve se o numero dado é primo ou não.'''
    if numero <= 1:  # negativos, 0 e 1 não são primos
        return False
    divisor: int = 2
    while divisor * divisor <= numero:  # melhor
        if numero % divisor == 0:
            return False
        divisor += 1
    return True


def primos_até_n(n: int) -> list[int]:
    ''' Devolve a lista dos primos menores ou iguais a n.'''
    lista: list[int] = []
    for numero in range(2, n + 1):
        if é_primo(numero):
            lista.append(numero)
    return lista


cProfile.run("primos_até_n(100000)")
```

Tempo de Execução

```
         109595 function calls in 0.181 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.181    0.181 <string>:1(<module>)
        1    0.009    0.009    0.181    0.181 teste_tempo2.py:15(primos_até_n)
    99999    0.172    0.000    0.172    0.000 teste_tempo2.py:3(é_primo)
        1    0.000    0.000    0.181    0.181 {built-in method builtins.exec}
     9592    0.000    0.000    0.000    0.000 {method 'append' of 'list' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler'
       objects}
```

Tempo de Execução

```
         109595 function calls in 0.181 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.181    0.181 <string>:1(<module>)
        1    0.009    0.009    0.181    0.181 teste_tempo2.py:15(primos_até_n)
    99999    0.172    0.000    0.172    0.000 teste_tempo2.py:3(é_primo)
        1    0.000    0.000    0.181    0.181 {built-in method builtins.exec}
     9592    0.000    0.000    0.000    0.000 {method 'append' of 'list' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler'
      objects}
```

Rodamos em 0.181 segundos!

Tempo de Execução

```
         109595 function calls in 0.181 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.181    0.181 <string>:1(<module>)
        1    0.009    0.009    0.181    0.181 teste_tempo2.py:15(primos_até_n)
    99999    0.172    0.000    0.172    0.000 teste_tempo2.py:3(é_primo)
        1    0.000    0.000    0.181    0.181 {built-in method builtins.exec}
     9592    0.000    0.000    0.000    0.000 {method 'append' of 'list' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler'
      objects}
```

Rodamos em 0.181 segundos!

- Ao invés de 25.820 segundos!

Usando um algoritmo melhor

```
1  def primos_até_n(n: int) -> list[int]:
2      ''' Devolve a lista dos primos menores ou iguais a n.
3
4          Implementa o Crivo de Eratóstenes.
5      '''
6      primos: list[int] = []
7      é_primo: list[bool] = []
8      for _ in range(n + 1):  # _ é variável não usada
9          é_primo.append(True)
10     p = 2
11     while p <= n:
12         if é_primo[p]:
13             primos.append(p)
14             # se p é primo, então seus múltiplos não são
15             for k in range(p * p, n + 1, p):
16                 é_primo[k] = False
17         p += 1
18     return primos
```

Usando um algoritmo melhor

```
1  def primos_até_n(n: int) -> list[int]:
2      ''' Devolve a lista dos primos menores ou iguais a n.
3
4          Implementa o Crivo de Eratóstenes.
5      '''
6      primos: list[int] = []
7      é_primo: list[bool] = []
8      for _ in range(n + 1):  # _ é variável não usada
9          é_primo.append(True)
10     p = 2
11     while p <= n:
12         if é_primo[p]:
13             primos.append(p)
14             # se p é primo, então seus múltiplos não são
15             for k in range(p * p, n + 1, p):
16                 é_primo[k] = False
17         p += 1
18     return primos
```

O código ainda poderia ser mais “Pythonico”

Usando um algoritmo melhor

```
1  def primos_até_n(n: int) -> list[int]:
2      ''' Devolve a lista dos primos menores ou iguais a n.
3
4          Implementa o Crivo de Eratóstenes.
5      '''
6      primos: list[int] = []
7      é_primo: list[bool] = []
8      for _ in range(n + 1):  # _ é variável não usada
9          é_primo.append(True)
10     p = 2
11     while p <= n:
12         if é_primo[p]:
13             primos.append(p)
14             # se p é primo, então seus múltiplos não são
15             for k in range(p * p, n + 1, p):
16                 é_primo[k] = False
17         p += 1
18     return primos
```

O código ainda poderia ser mais "Pythonico"

- Mas precisamos aprender mais coisas para isso

Tempo de Execução

```
         109597 function calls in 0.033 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.033    0.033 <string>:1(<module>)
        1    0.028    0.028    0.032    0.032 ultima_versao.py:4(primos_até_n)
        1    0.000    0.000    0.033    0.033 {built-in method builtins.exec}
   109593    0.005    0.000    0.005    0.000 {method 'append' of 'list' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler'
       objects}
```

Tempo de Execução

```
         109597 function calls in 0.033 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.033    0.033 <string>:1(<module>)
        1    0.028    0.028    0.032    0.032 ultima_versao.py:4(primos_até_n)
        1    0.000    0.000    0.033    0.033 {built-in method builtins.exec}
   109593    0.005    0.000    0.005    0.000 {method 'append' of 'list' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler'
       objects}
```

Rodamos em 0.033 segundos!

Tempo de Execução

```
         109597 function calls in 0.033 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.033    0.033 <string>:1(<module>)
        1    0.028    0.028    0.032    0.032 ultima_versao.py:4(primos_até_n)
        1    0.000    0.000    0.033    0.033 {built-in method builtins.exec}
   109593    0.005    0.000    0.005    0.000 {method 'append' of 'list' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler'
       objects}
```

Rodamos em 0.033 segundos!

- Ao invés de 0.181 segundos!

Tempo de Execução

```
         109597 function calls in 0.033 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.033    0.033 <string>:1(<module>)
        1    0.028    0.028    0.032    0.032 ultima_versao.py:4(primos_até_n)
        1    0.000    0.000    0.033    0.033 {built-in method builtins.exec}
   109593    0.005    0.000    0.005    0.000 {method 'append' of 'list' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler'
       objects}
```

Rodamos em 0.033 segundos!

- Ao invés de 0.181 segundos!
- Porém estamos criando uma lista bem grande na memória

Tempo de Execução

```
         109597 function calls in 0.033 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.033    0.033 <string>:1(<module>)
        1    0.028    0.028    0.032    0.032 ultima_versao.py:4(primos_até_n)
        1    0.000    0.000    0.033    0.033 {built-in method builtins.exec}
   109593    0.005    0.000    0.005    0.000 {method 'append' of 'list' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler'
       objects}
```

Rodamos em 0.033 segundos!

- Ao invés de 0.181 segundos!
- Porém estamos criando uma lista bem grande na memória
  - Gastamos por volta de `8 * n` bytes

Tempo de Execução

```
         109597 function calls in 0.033 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.033    0.033 <string>:1(<module>)
        1    0.028    0.028    0.032    0.032 ultima_versao.py:4(primos_até_n)
        1    0.000    0.000    0.033    0.033 {built-in method builtins.exec}
   109593    0.005    0.000    0.005    0.000 {method 'append' of 'list' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler'
       objects}
```

Rodamos em 0.033 segundos!

- Ao invés de 0.181 segundos!
- Porém estamos criando uma lista bem grande na memória
  - Gastamos por volta de `8 * n` bytes
  - 1Mb a cada 125 mil

Tempo de Execução

```
         109597 function calls in 0.033 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.033    0.033 <string>:1(<module>)
        1    0.028    0.028    0.032    0.032 ultima_versao.py:4(primos_até_n)
        1    0.000    0.000    0.033    0.033 {built-in method builtins.exec}
   109593    0.005    0.000    0.005    0.000 {method 'append' of 'list' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler'
       objects}
```

Rodamos em 0.033 segundos!

- Ao invés de 0.181 segundos!
- Porém estamos criando uma lista bem grande na memória
  - Gastamos por volta de `8 * n` bytes
  - 1Mb a cada 125 mil
- Estamos trocando tempo por espaço

Então o que é um bom código?

Um bom código é:

Então o que é um bom código?

Um bom código é:

- Correto

Então o que é um bom código?

Um bom código é:

- Correto
  - Implementa o algoritmo corretamente

Então o que é um bom código?

Um bom código é:

- Correto
  - Implementa o algoritmo corretamente
  - Testes ajudam a acreditar nisso

Então o que é um bom código?

Um bom código é:

- Correto
  - Implementa o algoritmo corretamente
  - Testes ajudam a acreditar nisso
- Rápido

Então o que é um bom código?

Um bom código é:

- Correto
  - Implementa o algoritmo corretamente
  - Testes ajudam a acreditar nisso
- Rápido
  - Rápido é relativo, depende da necessidade

Então o que é um bom código?

Um bom código é:

- Correto
  - Implementa o algoritmo corretamente
  - Testes ajudam a acreditar nisso
- Rápido
  - Rápido é relativo, depende da necessidade
  - Melhor ser rápido onde importa

Então o que é um bom código?

Um bom código é:

- Correto
  - Implementa o algoritmo corretamente
  - Testes ajudam a acreditar nisso
- Rápido
  - Rápido é relativo, depende da necessidade
  - Melhor ser rápido onde importa
  - Profilling ajuda a identificar gargalos

Então o que é um bom código?

Um bom código é:

- Correto
  - Implementa o algoritmo corretamente
  - Testes ajudam a acreditar nisso
- Rápido
  - Rápido é relativo, depende da necessidade
  - Melhor ser rápido onde importa
  - Profilling ajuda a identificar gargalos
- Fácil de ler

Então o que é um bom código?

Um bom código é:

- Correto
  - Implementa o algoritmo corretamente
  - Testes ajudam a acreditar nisso
- Rápido
  - Rápido é relativo, depende da necessidade
  - Melhor ser rápido onde importa
  - Profilling ajuda a identificar gargalos
- Fácil de ler
  - Segue um estilo

Então o que é um bom código?

Um bom código é:

- Correto
  - Implementa o algoritmo corretamente
  - Testes ajudam a acreditar nisso
- Rápido
  - Rápido é relativo, depende da necessidade
  - Melhor ser rápido onde importa
  - Profilling ajuda a identificar gargalos
- Fácil de ler
  - Segue um estilo
  - Tem bons nomes de variáveis, funções, etc.

Então o que é um bom código?

Um bom código é:

- Correto
  - Implementa o algoritmo corretamente
  - Testes ajudam a acreditar nisso
- Rápido
  - Rápido é relativo, depende da necessidade
  - Melhor ser rápido onde importa
  - Profilling ajuda a identificar gargalos
- Fácil de ler
  - Segue um estilo
  - Tem bons nomes de variáveis, funções, etc.
  - Tem comentários úteis e focados

Então o que é um bom código?

Um bom código é:

- Correto
  - Implementa o algoritmo corretamente
  - Testes ajudam a acreditar nisso
- Rápido
  - Rápido é relativo, depende da necessidade
  - Melhor ser rápido onde importa
  - Profilling ajuda a identificar gargalos
- Fácil de ler
  - Segue um estilo
  - Tem bons nomes de variáveis, funções, etc.
  - Tem comentários úteis e focados
  - Tem uma boa documentação