



Ferramentas e Técnicas de ***Coding Standards***

Investigação sobre ferramentas de qualidade de *software* e/ou testes de *software*

ÍNDICE DE CONTEÚDO

TÓPICOS A ABORDAR AO LONGO DA APRESENTAÇÃO

1 INDICADORES DE QUALIDADE DE *SOFTWARE*

Qualidade do *Software*/Código • Características de Qualidade de *Software* • Métricas de Qualidade de *Software*

CODING STANDARDS 2

Definição • Propósito/Funcionalidade • Práticas Recomendadas

3 FERRAMENTAS DE *CODING STANDARDS*

Linguagens de Programação mais usadas • Ferramenta *Java* e *Python* • Ferramenta Multi Linguagem

1 INDICADORES DE QUALIDADE DE SOFTWARE

- 1.1. Qualidade do *Software*/Código
- 1.2. Características de Qualidade de *Software*
- 1.3. Métricas de Qualidade de *Software*

QUALIDADE DO *SOFTWARE*/CÓDIGO

DIFICULDADE EM DEFINIR DE FORMA JUSTA A QUALIDADE *SOFTWARE*/CÓDIGO



Necessidade definir
uma Qualidade De
Software/Código



Diferentes perspectivas de
vários programadores
originam diferentes opiniões



Necessidade da existência
de uma conformidade na
Avaliação Final

QUALIDADE DO *SOFTWARE*/CÓDIGO

MEDIDAS PENSADAS PARA AVALIAR/MEDIR A QUALIDADE DE UM PRODUTO DE *SOFTWARE*

O padrão ISO/IEC 25010 surge no sentido de criar um modelo de qualidade para um produto de *software*, estabelecendo todas as características de qualidade de um *software*. No entanto, conforme se verá mais à frente, apresenta lacunas/entraves.



- Características de Qualidade de *Software* consequentemente
- Métricas de Qualidade de *Software*

CARACTERÍSTICAS DE QUALIDADE DE *SOFTWARE*

ISO/IEC 25010 E AS OITO CARACTERÍSTICAS DE QUALIDADE DE UM PRODUTO DE *SOFTWARE*

A ISO 25000 mencionada anteriormente surge no sentido de definir um modelo de qualidade para um produto de *software*. Este padrão ISO é então nomeado de ISO/IEC 25010 e estipula oito características principais de qualidade:

- | | | | |
|----------|-------------------------------|----------|------------------------|
| 1 | <i>Functional Suitability</i> | 5 | <i>Reliability</i> |
| 2 | <i>Performance Efficiency</i> | 6 | <i>Security</i> |
| 3 | <i>Compatibility</i> | 7 | <i>Maintainability</i> |
| 4 | <i>Usability</i> | 8 | <i>Portability</i> |

CARACTERÍSTICAS DE QUALIDADE DE *SOFTWARE*

ENTRAVES DAS OITO CARACTERÍSTICAS DE QUALIDADE DEFINIDOS PELA ISO/IEC 25010

Portability

O padrão ISO/IEC 25010 não define formas de medir as características de qualidade, muito menos a unidade em si.



Como se poderia medir a adaptabilidade de um produto e qual a unidade de medida a usar?



Efficiency

As características de qualidade podem ter diferentes significados dependendo do contexto.

Uma resposta de 1ms pode ser ideal para um programa e má para outro. Como medir isso?

MÉTRICAS DE QUALIDADE DE *SOFTWARE*

AS OITO MÉTRICAS DE QUALIDADE DE CÓDIGO DE *SOFTWARE*

De forma a se obter uma maneira mais sistemática de medir e qualificar as características faladas até então, mitigando toda a abordagem feita pela ISO 25000, surgem oito métricas de qualidade de código de *software*:

1 *Code coverage*

2 *Abstract interpretation*

3 *Cyclomatic complexity*

4 *Compiler warnings*

5 *Coding standards*

6 *Code duplication*

7 *Fan out*

8 *Security*

2 CODING STANDARDS

2.1. Definição

2.2. Propósito/Funcionalidade

2.3. Práticas Recomendadas

DEFINIÇÃO

O QUE SÃO CODING STANDARDS?



“Coding standards are a set of industry-recognized best practices that provide a variety of guidelines for developing software code. There is evidence to suggest that compliance to coding standards in software development can enhance team communication, reduce program errors and improve code quality.”

Com isto se entende que os **Coding Standards** consistem num conjunto de regras e diretrizes que determinam o estilo, procedimentos e métodos de programação para uma determinada linguagem de programação.

PROPÓSITO/ FUNCIONALIDADE

PORQUE SÃO NECESSÁRIOS *CODING STANDARDS*?

A ideia de escrever código sem convenções e padrões pode facilmente tornar-se algo confuso e insustentável, principalmente quando se está perante uma grande base de código pertencente a um produto de *software*.

Para existir um código bem estruturado e limpo, os *developers* precisam de estar a par dos padrões/convenções usados pela sua equipa no geral, dado que dessa forma estão a facilitar a compreensão do mesmo por parte de *developers* que estejam a entrar pela primeira vez na base de código.

necessidade
extrema



Cláusulas
pré-definidas
para as
várias
linguagens

PROPÓSITO/ FUNCIONALIDADE

QUAIS SÃO AS VANTAGENS DA PRESENÇA DE *CODING STANDARDS*?
E AS DESVANTAGENS DA AUSÊNCIA DE *CODING STANDARDS*?

Vantagens da presença de <i>Coding Standards</i>	Desvantagens da ausência de <i>Coding Standards</i>
<ul style="list-style-type: none">• Aumento da eficiência na qualidade do código;• Aumento da facilidade de manutenção;• Aumento da facilidade de detecção/correção de <i>bugs</i>;• Redução nos riscos de falhas;• Redução da complexidade do código;• Redução custo do desenvolvimento de <i>software</i>.	<ul style="list-style-type: none">• Problemas ao nível da segurança do <i>software</i>;• Motivação reduzida por parte de toda a equipa;• Aumento do tempo de desenvolvimento;• Estrutura complexa da base do código.

PRÁTICAS RECOMENDADAS

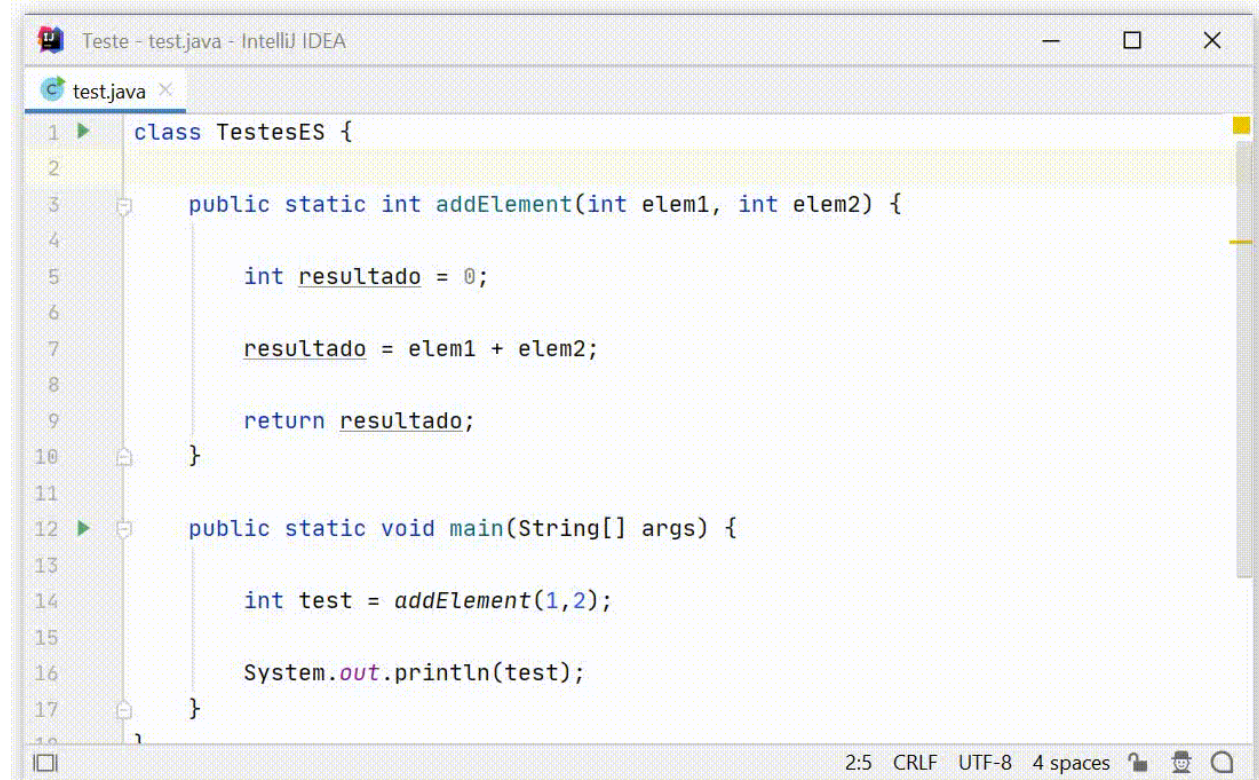
QUAIS SÃO AS PRÁTICAS UNIVERSAIS DE GRANDE PARTE DAS LINGUAGENS DE PROGRAMAÇÃO?

1 PRÁTICA

Documentação adequada do código

Comentar o código é uma boa prática e facilita o entendimento entre os vários *developers*.

Neste caso, com o IntelliJ IDEA consegue-se criar de imediato uma documentação automática, sendo apenas necessário alterar os parâmetros.



```
1  class TestesES {
2
3      public static int addElement(int elem1, int elem2) {
4
5          int resultado = 0;
6
7          resultado = elem1 + elem2;
8
9          return resultado;
10     }
11
12     public static void main(String[] args) {
13
14         int test = addElement(1,2);
15
16         System.out.println(test);
17     }
18 }
```

PRÁTICAS RECOMENDADAS

QUAIS SÃO AS PRÁTICAS UNIVERSAIS DE GRANDE PARTE DAS LINGUAGENS DE PROGRAMAÇÃO?

2 PRÁTICA

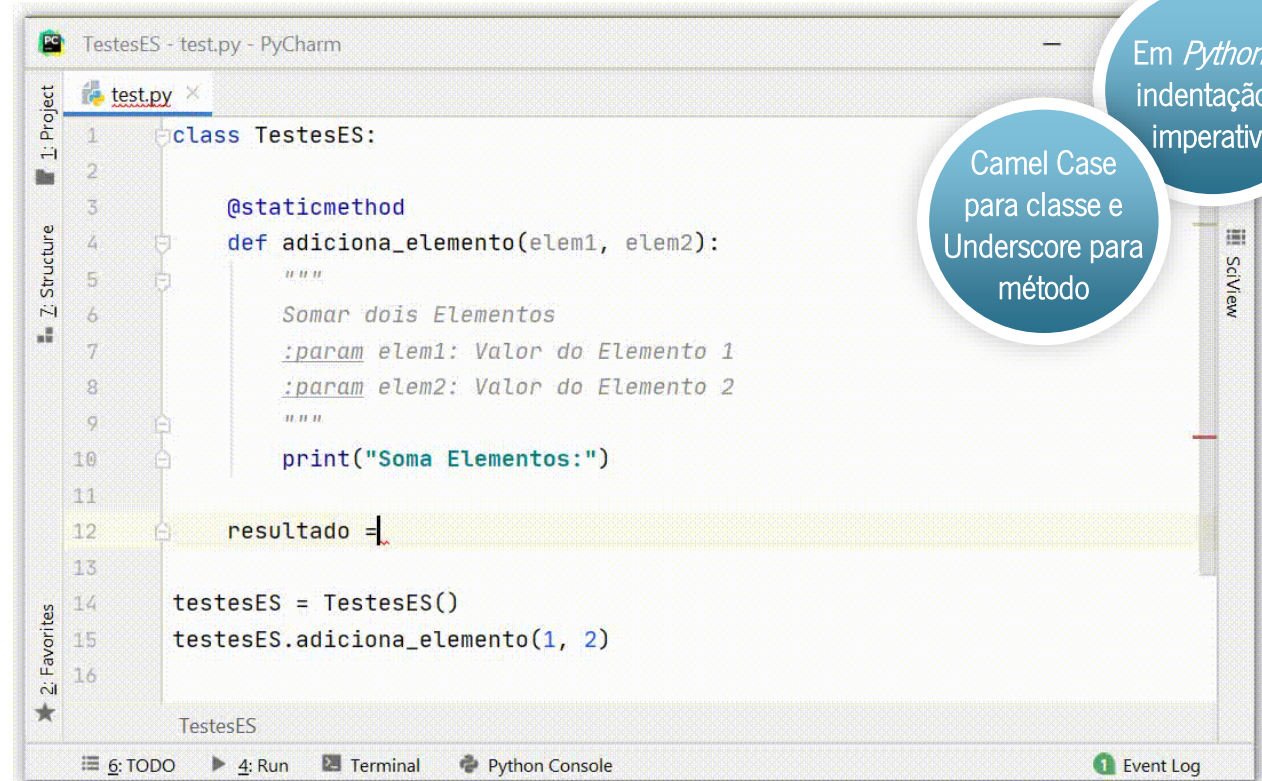
Indentação/Formatação Legível

Um bom código deve apresentar um formato e indentação padronizados.

3 PRÁTICA

Esquema de Nomenclatura

Usar uma nomenclatura para nomear os vários métodos, classes, variáveis, etc, é das melhores formas de ajudar na identificação dos vários nomes.



PRÁTICAS RECOMENDADAS

QUAIS SÃO AS PRÁTICAS UNIVERSAIS DE GRANDE PARTE DAS LINGUAGENS DE PROGRAMAÇÃO?

4 PRÁTICA

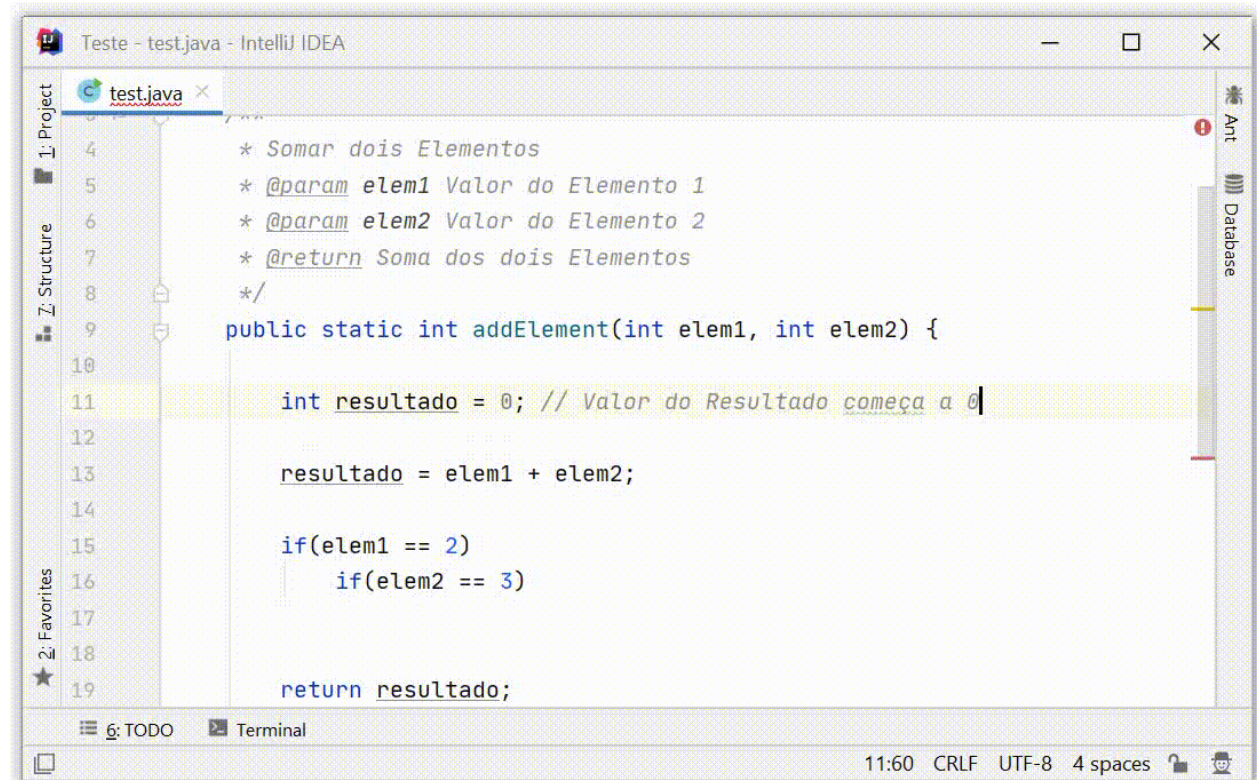
Comentários Desnecessários

Evitar comentar linhas desnecessárias que ocupem espaço visual.

5 PRÁTICA

Condições/Estruturas Aninhadas

A existência de estruturas/condições aninhadas pode complicar a interpretação do algoritmo em si e em muitas situações pode ser facilmente evitada.



PRÁTICAS RECOMENDADAS

QUAIS SÃO AS PRÁTICAS UNIVERSAIS DE GRANDE PARTE DAS LINGUAGENS DE PROGRAMAÇÃO?

6 PRÁTICA

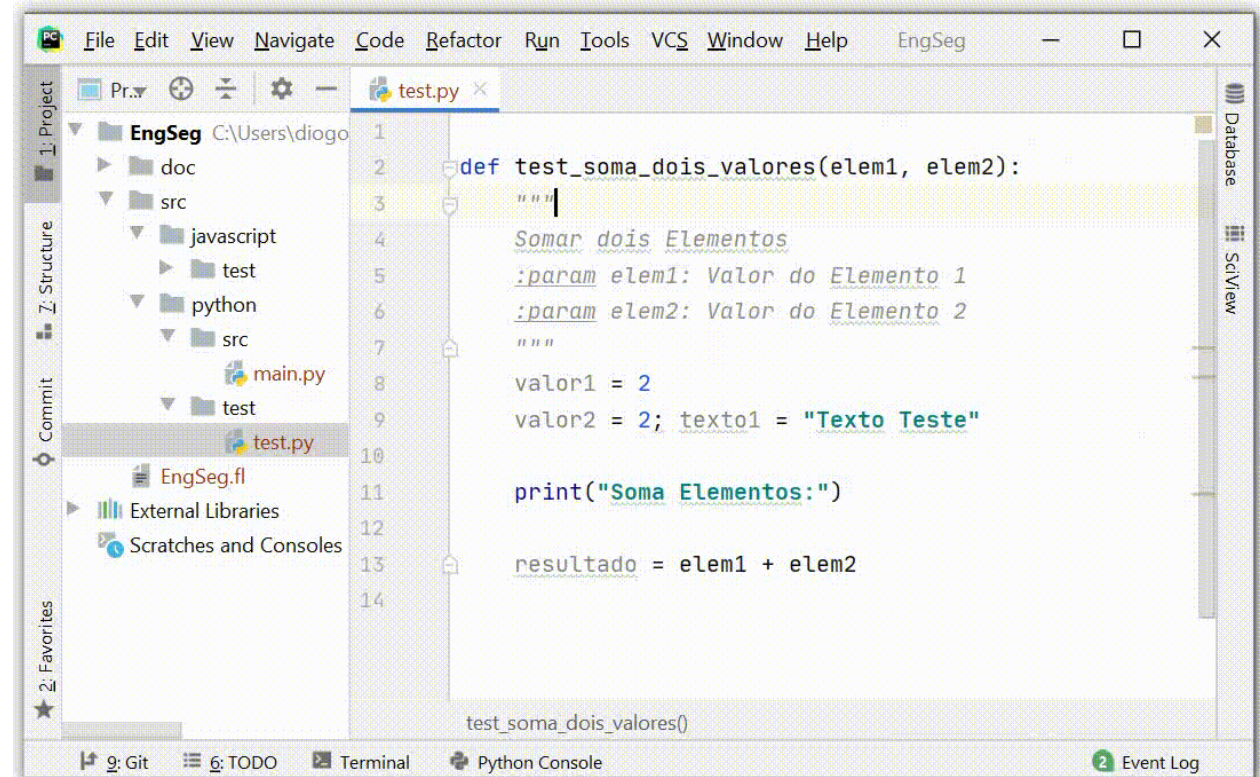
Linhas/Funções de comprimento curto

É preferível criar funções mais pequenas, dividindo tarefas em outras funções auxiliares.

7 PRÁTICA

Organização das várias pastas e arquivos

Organizar todo o projeto, organizando-o em diferentes pastas e ficheiros indo ao encontro da funcionalidade do *software*.

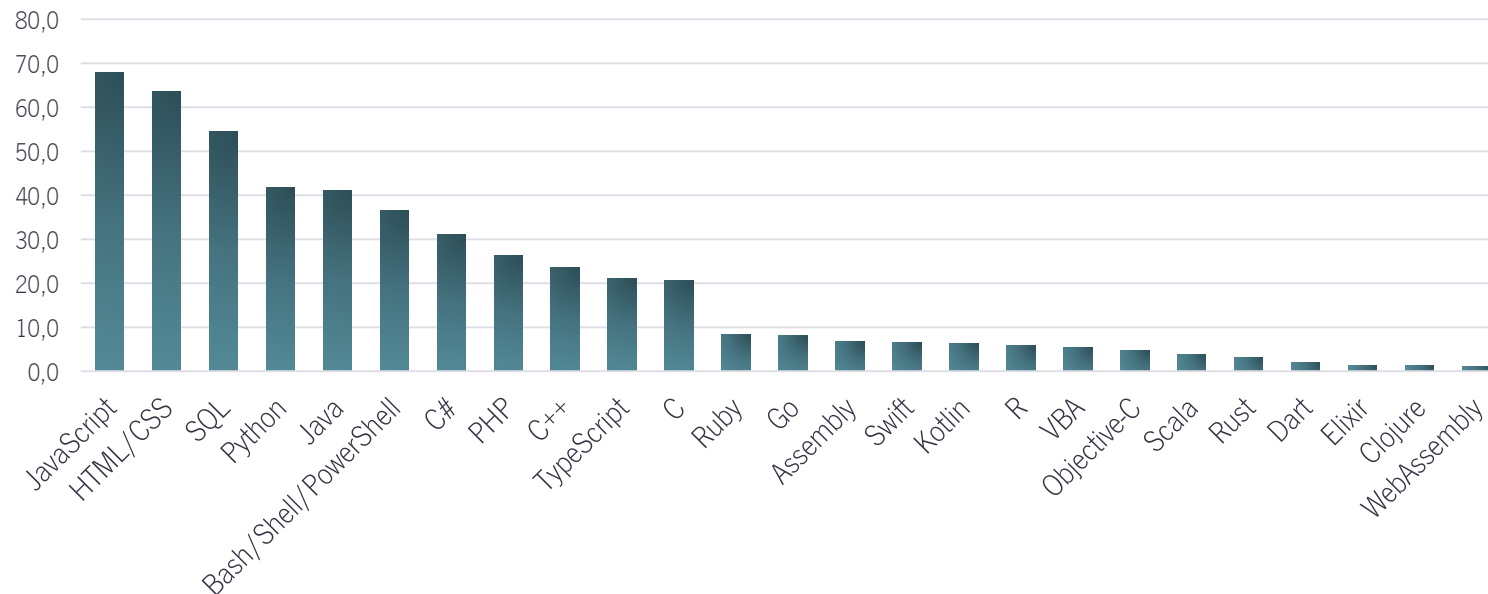


3 FERRAMENTAS DE *CODING STANDARDS*

- 3.1. Linguagens de Programação mais usadas
- 3.2. Ferramenta *Java* e *Python*
- 3.3. Ferramenta Multi Linguagem

LINGUAGENS DE PROGRAMAÇÃO MAIS USADAS

GRÁFICO *STACK OVERFLOW'S ANNUAL DEVELOPER SURVEY* – DADOS 2019



LINGUAGENS DE PROGRAMAÇÃO MAIS USADAS

ESCOLHA DE DUAS LINGUAGENS PARA O ESTUDO DOS *STANDARDS* E FERRAMENTAS

Python tem na sua génese uma filosofia nuclear que consiste em simplificar, explicitar e embelezar.

Linguagem que desde a sua conceção tinha em ideia existir um *Coding Standard* universal - PEP8 - que se tornou a base para a convenção universal de qualquer *developer*.



Java, na sua génese não seguia um *code guideline* que fosse universal e ditado pelo desenvolvedor da mesma.

Com a sua enorme popularidade e crescimento, principalmente na partilha de código e importação de bibliotecas, houve a necessidade de estandardizar algumas regras.

LINGUAGEM *PYTHON*

STANDARDS ESPECÍFICOS PARA *PYTHON*

PEP 8 é a convenção universal proposta pela fundação *Python* e apresenta, entre outros, os mais variados *Coding Standards*:

- Existe convenções de nomes dos objetos, como a não utilização das letras “O,l,l” que podem ser confundidas com os valores de 0 e 1, respetivamente;
- Os nomes das funções, variáveis, métodos devem ser todos com letra minúscula separando palavras com o carácter *Underscore*;
- As classes devem seguir o estilo de *Camel Case* de forma a se distinguir facilmente;
- As constantes devem ser sempre com caracteres maiúsculos.

“Código é lido muito mais vezes que é escrito”

Guido van Rossum,
criador do *Python*



LINGUAGEM *PYTHON*

STANDARDS ESPECÍFICOS PARA *PYTHON*

- **Linha 2** - Múltiplas importações na mesma linha;
- **Linha 3** - Menos do que 2 linhas a separar o código anterior do novo método;
- **Linha 4** - O nome do método não segue a convenção *snake_case*, ou seja, letras minúsculas e palavras separadas por *underscore*;
- **Linha 4 e 5** - O nome das variáveis não segue a mesma convenção supracitada;
- **Linha 5** - A indentação de espaços na declaração do método e os seus argumentos está errada confundindo-se com corpo de código.

Code Editor | teste.py

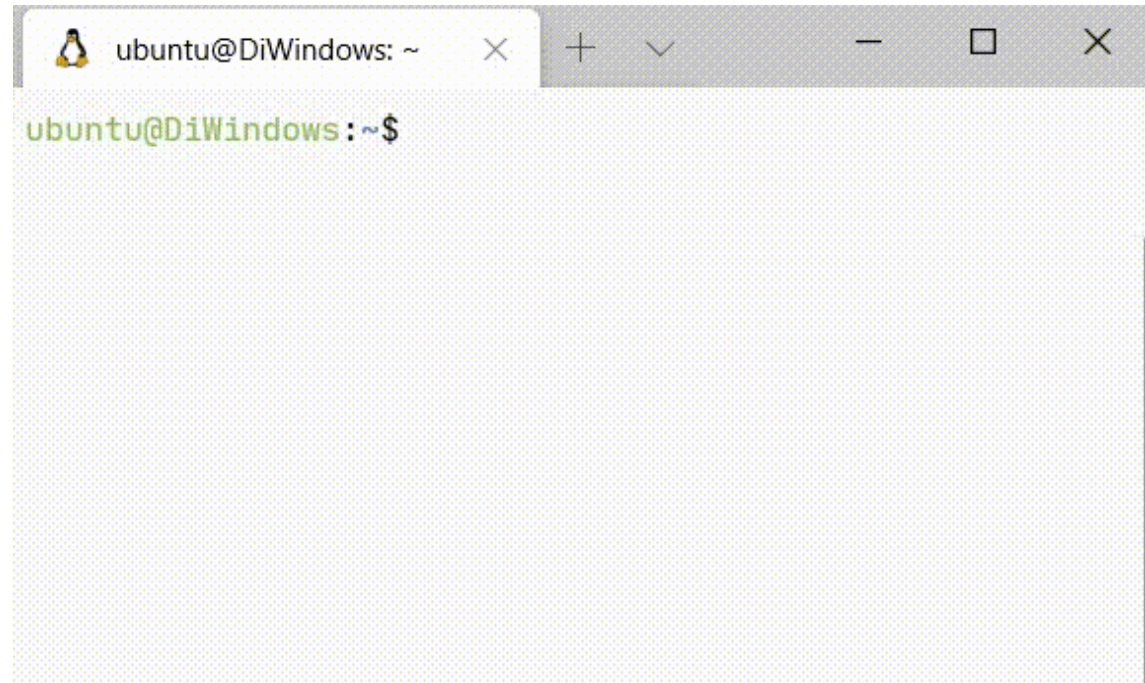
```
1
2  import os, sys
3
4  def FuncaoErrada(VARIAVEL1,
5                  VARIAVEL2):
6
7      if VARIAVEL is int:
8          print("É um inteiro")
9
```

LINGUAGEM *PYTHON*

FERRAMENTA *OPEN-SOURCE* PARA VERIFICAÇÃO AUTOMÁTICA

Uma ferramenta recomendada pela TIOBE é utilizada amplamente pela grande parte dos programadores *Python* é o *PyLint*.

Este *linter* instalado de maneira simples em qualquer instância *Python* e suportado pela maioria dos editores de texto utilizados, tornou-se uma ferramenta obrigatória para cuidar do standard do código, bem como a continua análise do mesmo fornecendo ajudas visuais e avisos para as falhas que o código possa ter.



LINGUAGEM JAVA

STANDARDS ESPECÍFICOS PARA *JAVA*

Apesar de não existir uma convenção universal proposta pela própria fundação *Java*, existe um conjunto de convenções que devem ser tomadas em conta:

- Devem ser declarados os *packages* primeiro e só depois todo os *imports*;
- A existência de comentários (análogos à documentação);
- Declarar as variáveis do mesmo tipo na mesma linha, por forma a evitar linhas desnecessárias;
- As classes em Java devem seguir regras de espaçamento pelos vários parênteses;
- Os métodos devem ser separados por uma linha em branco;
- O uso de *Camel Case* como prática para escrever os nomes dos métodos, variáveis, classes, packages e constantes.

LINGUAGEM JAVA

ANÁLISE DE EXTRATO DE CÓDIGO COM FALHAS CONSOANTE AS CONVENÇÕES ESPECÍFICAS

- **Linha 1** - Fazer o *import* de todo o package `java.lang` mesmo sem ser necessário para o código em causa;
- **Linha 5** - Uso de *Dollar Sign* no nome do método;
- **Linha 9** - Criação de um ciclo *for* para definir uma ideia de continuidade de uma condição quando poderia ser usado um ciclo *while*;
- **Linha 10** - Uso de uma negação na parte do *if* com a cláusula *e/se*.

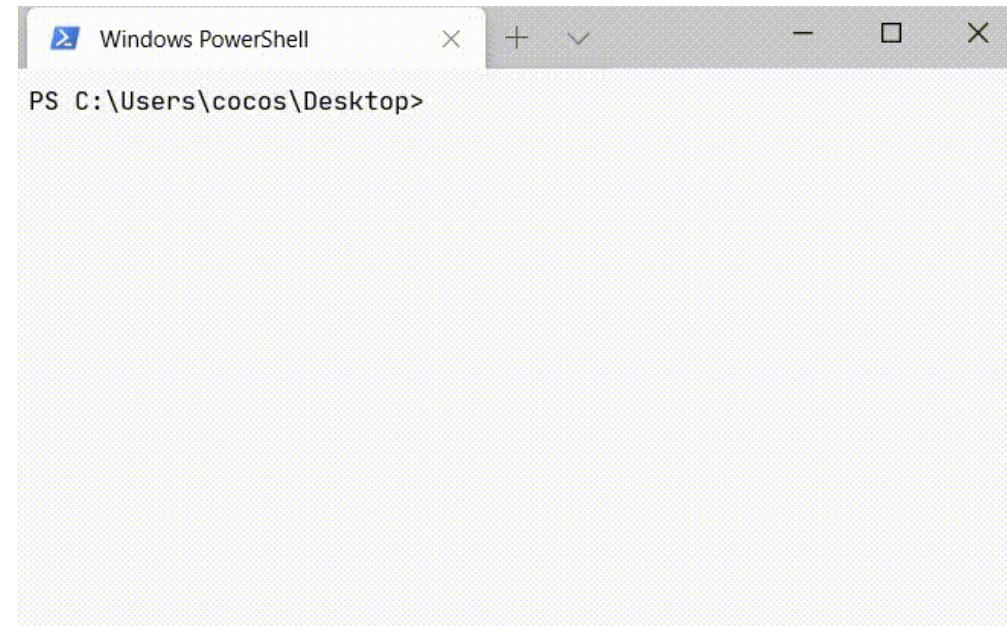
Code Editor | teste.java

```
1 import java.lang.*;
2
3 public class Main {
4
5     private static void teste$codigo(int x) {
6
7         int y = 4;
8
9         for(;x==3;){
10             if (x != y) System.out.println("Dif");
11             else System.out.println("Iguais");
12         }
13         (...)
```


LINGUAGEM JAVA

FERRAMENTA *OPEN-SOURCE* PARA VERIFICAÇÃO AUTOMÁTICA

Uma das razões por se ter escolhido *Java* como uma das linguagens para estudo de ferramentas de *Coding Standards* foi o facto da mesma não apresentar um *guideline* pensado pela própria fundação. Isto nota-se pela diferença de resultados, mas ao utilizar a ferramenta **PMD** recomendada pelo TIOBE conseguimos verificar na figura abaixo que todas as “falhas” do *guideline* seguido aparecem na análise final verificando assim o poder da ferramenta *open-source*.



FERRAMENTA MULTI LINGUAGEM

FERRAMENTA *CLOSED-SOURCE* PARA VERIFICAÇÃO AUTOMÁTICA

