Universidade do Minho
Escola de Engenharia
Mestrado em Engenharia Informática

Unidade Curricular de Engenharia Gramatical 2025/2026

Beautifiers

Guilherme Pinto Pinho pg60263
Rodrigo Sousa pg60302

Outubro, 2025

EG

Data da Receção	
Responsável	
Avaliação	
Observações	

Beautifiers

Guilherme Pinto Pinho pg60263
Rodrigo Sousa pg60302

Outubro, 2025

Índice

Lista de Figuras

Lista de Tabelas

1. Introdução

A **análise estática** de código-fonte consiste na inspeção do código sem a sua execução, com o objetivo de identificar problemas, aprimorar a qualidade e assegurar a conformidade com boas práticas de programação.

Dentro das diversas categorias de ferramentas de análise estática, destacam-se os **Beautifiers**, cuja principal função é formatar o código automaticamente, tornando-o mais legível, padronizado e fácil de manter.

O presente relatório tem como finalidade estudar o funcionamento das ferramentas **Beautifiers** e compreender a forma como se relacionam com os conteúdos abordados na disciplina de **Engenharia Gramatical**. Pretende-se analisar a sua relevância no processo de desenvolvimento de software e apresentar exemplos práticos da sua aplicação.

2. Conceito de Beautifiers

Beautifiers (ou code formatters) são ferramentas que analisam o código-fonte e aplicam automaticamente regras de formatação definidas, sem alterar a sua lógica. O seu objetivo é uniformizar o estilo do código, melhorando a legibilidade e facilitando o trabalho em equipa.

Principais funcionalidades:

- Ajuste automático de indentação e espaçamento.
- Colocação consistente de chavetas {} e parênteses.
- Quebra de linhas longa (line wrapping).
- Remoção de espaços ou linhas em branco desnecessárias.
- Aplicação de convenções de estilo (ex: nomes, alinhamento).

Benefícios:

- Melhoria da legibilidade e consistência do código.
- Redução de discussões de estilo entre programadores.
- Facilita revisões de código (code review).
- Automatiza tarefas repetitivas, poupando tempo.

3. Funcionamento Interno

Um Beautifier (como o **Prettier** e o **Black**) é uma ferramenta de formatação automática que atua sobre a **estrutura sintática** do código, e não apenas sobre regras textuais. O seu funcionamento segue um **pipeline típico da Engenharia Gramatical**, composto por três fases principais:

1. **Parsing(Análise Sintática)**: O Beautifier utiliza um **parser específico** para a linguagem em questão. Este parser lê o código-fonte e transforma-o numa **árvore sintática abstrata (AST)**, que representa a estrutura gramatical do programa de acordo com as regras formais da linguagem. Este processo é análogo ao que é aplicado em ferramentas como o Lark, trabalhada na cadeira de Engenharia Gramatical, onde uma gramática definida em EBNF é usada para definir um parser capaz de construir uma árvore hierárquica que representa o programa.
2. **Transformação Estrutural**: Após a criação da AST, o Beautifier percorre a árvore e aplica as **devidas transformações estruturais**, que **não alteram o significado semântico do código**, mas reorganizam a sua forma lexical e estrutural. São aplicadas regras de formatação, garantindo assim algo **consistente e independente do estilo pessoal do programador.**
3. **Gerar Código modificado**: Finalmente, o Beautifier **gera novamente o texto do programa** a partir da AST transformada.O resultado é um código-fonte semanticamente igual ao original, mas visualmente mais legível e padronizado de acordo com as convenções da ferramenta.

O processo completo de funcionamento de um Beautifier pode ser representado da seguinte forma:

Código Fonte → Parser (AST) → Transformer (formatação estrutural) → Código Formatado

Por outro lado Beautifiers como o Autopep8 não realizam uma análise sintática completa nem constroem uma árvore sintática abstrata (AST). Em vez disso, eles baseiam-se na ferramenta pycodestyle (ou pep8), que analisa o código para detetar violação das regras da pep8 — o guia oficial de estilo do Python.

1. **Análise e deteção de erros de estilo**: O Autopep8 lê o código-fonte linha a linha e identifica onde há:
   - indentação incorreta,
   - linhas demasiado longas,
   - espaços em falta ou a mais,
   - operadores mal posicionados,
   - outras inconsistências estilísticas.
2. **Correção Estrutural (Transformação Superficial)**: Após identificar as violações, o Autopep8 aplica transformações diretas no texto do código — ou seja, corrige apenas as partes que estão em desacordo com as regras pep8, sem reescrever completamente o programa. Isto distingue-o do Blacke do Prettier: enquanto as outras duas ferramentas reconstroem todo o código com base numa AST, o Autopep8 corrige pontualmente o que está errado, tentando alterar o mínimo possível.
3. **Código Corrigido**: Finalmente, o Autopep8 grava novamente o código com as correções aplicadas. O resultado é um programa com estilo conforme à pep8, mas cuja estrutura e espaçamento permanecem semelhantes ao original. Esta abordagem é útil quando se pretende corrigir automaticamente pequenas falhas de estilo, sem reformular completamente a estrutura visual do código.

O processo completo de funcionamento da Autopep8 pode ser representado da seguinte forma.

Código Fonte → Analisador de Estilo (pep8) → Correções Locais → Código Formatado

3.1. Exemplo de Ferramentas Beautifiers

Ferramenta	Linguagem	Descrição
Prettier	JavaScript , TypeScript, HTML e CSS	Formata o código com base numa árvore sintática abstrata (AST). Aplica regras consistentes de indentação, espaçamento e estrutura. Foca-se na legibilidade e uniformização do estilo.
Black	Python	Reescreve todo o código segundo um conjunto fixo de regras definidas pela PEP 8. Garante consistência total e elimina decisões de estilo do programador.
Autopep8	Python	Analisa o código e corrige automaticamente erros de estilo que violam a PEP 8. Atua mais como um corretor do que como reformulador estrutural.

Tabela 1: Exemplos de Ferramentas Beautifiers

4. Exemplos práticos de formatação:

4.1. Black

1. **Exemplo 1 - Execução do Black Formatter**: A imagem seguinte mostra o código Python antes de ser formatado pelo Black Formatter. Nota-se a presença de espaços desnecessários, falta de indentação e estilo inconsistente.

Antes:

```
def   media (a,b):return (a+b)/2
print ( media (10 ,5) )
```

Figura 1: Exemplo 1 Black-Antes

Depois:

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


print(media(10, 5))
```

Figura 2: Exemplo 1 Black-Depois

Após a execução do Black Formatter, o código é automaticamente reestruturado para seguir as convenções da PEP 8(o guia oficial de estilo do Python que define regras para manter o código legível, coerente e bem estruturado). O resultado apresenta indentação adequada, espaçamento uniforme e melhor legibilidade.

2. **Exemplo 2 - Execução do Black Formatter**: O código abaixo representa uma função Python com formatação incorreta. Podemos observar problemas como a ausência de indentação adequada, espaços desnecessários e má organização das linhas.

Antes:

```
def  calcular_media (lista ):
 total=0
 for numero in lista: total+=numero
 return total /len(lista)
print (calcular_media([ 10,20,30,40]))
```

Figura 3: Exemplo 2 Black-Antes

Depois:

```python
def calcular_media(lista):
    total = 0
    for numero in lista:
        total += numero
    return total / len(lista)


print(calcular_media([10, 20, 30, 40]))
```

Figura 4: Exemplo 2 Black-Depois

Após a execução do Black Formatter, o código é automaticamente reestruturado para seguir as convenções da PEP 8, o guia oficial de estilo do Python que define regras para manter o código legível, coerente e bem estruturado. O resultado apresenta indentação adequada, espaçamento uniforme e melhor legibilidade.

4.2. Prettier

1. **Exemplo 1 - Execução do Prettier Formatter**: A imagem seguinte apresenta um exemplo simples de código JavaScript antes de ser formatado pelo Prettier. Podem ser observadas inconsistências de espaços, indentação incorreta e uso desigual de aspas. O código, embora funcional, está pouco legível.

Antes:

```javascript
function apresentar ( nome,idade )
{if(idade>17){console.log('Bem vindo '+nome+"!")
}else{console.log( "Olá "+nome+", ainda és menor de idade." )}}
```

Figura 5: Exemplo 1 Prettier-Antes

Depois:

```javascript
function apresentar(nome, idade) {
  if (idade > 17) {
    console.log("Bem vindo " + nome + "!");
  } else {
    console.log("Olá " + nome + ", ainda és menor de idade.");
  }
}
```

Figura 6: Exemplo 1 Prettier-Depois

Nesta imagem está presente o mesmo código após ser formatado automaticamente pelo Prettier. O resultado apresenta indentação consistente, espaços padronizados e uso uniforme de aspas duplas, tornando o código mais limpo e fácil de ler.

2. **Exemplo 2 - Execução do Prettier Formatter**: A imagem seguinte mostra um pequeno trecho de código HTML antes de ser formatado pelo Prettier. O código apresenta erros comuns de estilo, como quebras de linha irregulares entre outros, o que dificulta a leitura e manutenção.

Antes:

```html
<!doctype html><html><head><meta charset="utf-8"><title>
Teste</title><style>body{background:#fff;color:black}
</style></head><body><div class="container">
<h1>Olá Mundo</h1><p>Este é um parágrafo <strong>
importante</strong>.<img src="img.png" alt="img">
</p><ul><li>Item1</li><li>Item2</li></ul></div></body></html
```

Figura 7: Exemplo 2 Prettier-Antes

Depois:

Figura 8: Exemplo 2 Prettier-Depois

A imagem mostra o mesmo código após ser formatado automaticamente pelo Prettier. Agora o código está bem estruturado, com indentação e espaçamento consistentes, posicionamento correto das chavetas e quebras de linha uniformes, evidenciando a legibilidade e organização que o Prettier proporciona.

4.3. Autopep8

1. **Exemplo 1 - Execução do Autopep8 Formatter**: A imagem seguinte mostra o código Python antes de ser formatado pelo autopep8. É possível observar a presença de espaços desnecessários, falta de indentação adequada e inconsistência no estilo..

Antes:

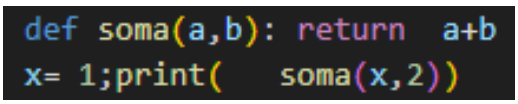

Figura 9: Exemplo 1 Autopep8-Antes

Depois:

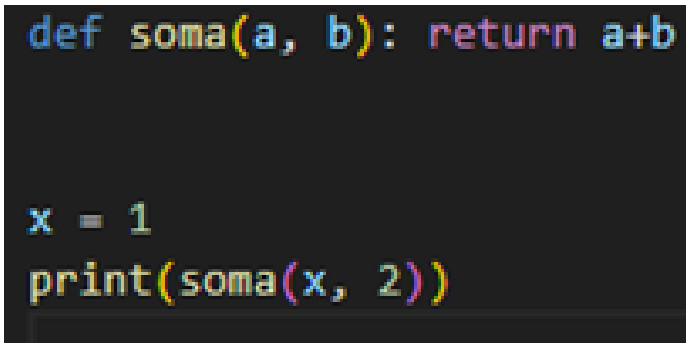

Figura 10: Exemplo 1 Autopep8-Depois

Após a execução do autopep8, o código é automaticamente reestruturado de acordo com as normas da PEP 8, o que resulta numa melhor legibilidade, indentação correta e espaçamento uniforme. O autopep8 assegura que o código segue as boas práticas definidas pela comunidade Python.

2. **Exemplo 2 - Execução do Autopep8 Formatter**: A imagem seguinte apresenta um exemplo de código Python com instruções condicionais mal formatadas. Nota-se a ausência de indentação adequada, falta de espaçamento entre operadores e inconsistência na organização das linhas de código.

Antes:

```
def avaliar(nota):
 if nota>=9: print("Excelente")
 elif nota>=7 and nota<9:  print("Bom")
 elif nota>=5 and nota<7: print("Suficiente")
 else:  print("Insuficiente")
avaliar (8)
```

Figura 11: Exemplo 2 Autopep8-Antes

Depois:

```
def avaliar(nota):
    if nota >= 9:
        print("Excelente")
    elif nota >= 7 and nota < 9:
        print("Bom")
    elif nota >= 5 and nota < 7:
        print("Suficiente")
    else:
        print("Insuficiente")


avaliar(8)
```

Figura 12: Exemplo 2 Autopep8-Depois

Após a execução do autopep8, o código é automaticamente ajustado de acordo com as normas da PEP 8. A indentação é corrigida, o espaçamento entre operadores e vírgulas é uniformizado e as linhas são devidamente organizadas, resultando num código mais claro e legível.