

# *Witch Cooking*

Formatação Multilíngue e Personalizada de Código-Fonte via o Sistema  
*Tree-Sitter*

Átila Gama Silva

19 de novembro de 2023

# Motivações

# Motivações

- ▶ Surgiram das dificuldades ao estudar estilos de formatação de código-fonte em diversas linguagens de programação

# Motivações

- ▶ Surgiram das dificuldades ao estudar estilos de formatação de código-fonte em diversas linguagens de programação
  - ▶ Durante a análise dos estilos convencionais de formatação

# Motivações

- ▶ Surgiram das dificuldades ao estudar estilos de formatação de código-fonte em diversas linguagens de programação
  - ▶ Durante a análise dos estilos convencionais de formatação
    - ▶ Era imprescindível recorrer a diferentes *prettyprinters*

# Motivações

- ▶ Surgiram das dificuldades ao estudar estilos de formatação de código-fonte em diversas linguagens de programação
  - ▶ Durante a análise dos estilos convencionais de formatação
    - ▶ Era imprescindível recorrer a diferentes *prettyprinters*
    - ▶ Cada um com suas próprias configurações e níveis de suporte para esses estilos

# Motivações

- ▶ Surgiram das dificuldades ao estudar estilos de formatação de código-fonte em diversas linguagens de programação
  - ▶ Durante a análise dos estilos convencionais de formatação
    - ▶ Era imprescindível recorrer a diferentes *prettyprinters*
    - ▶ Cada um com suas próprias configurações e níveis de suporte para esses estilos
  - ▶ Durante a análise dos estilos não convencionais

# Motivações

- ▶ Surgiram das dificuldades ao estudar estilos de formatação de código-fonte em diversas linguagens de programação
  - ▶ Durante a análise dos estilos convencionais de formatação
    - ▶ Era imprescindível recorrer a diferentes *prettyprinters*
    - ▶ Cada um com suas próprias configurações e níveis de suporte para esses estilos
  - ▶ Durante a análise dos estilos não convencionais
    - ▶ A aplicação manual era inevitável



# Motivações

- ▶ Surgiram das dificuldades ao estudar estilos de formatação de código-fonte em diversas linguagens de programação
  - ▶ Durante a análise dos estilos convencionais de formatação
    - ▶ Era imprescindível recorrer a diferentes *prettyprinters*
    - ▶ Cada um com suas próprias configurações e níveis de suporte para esses estilos
  - ▶ Durante a análise dos estilos não convencionais
    - ▶ A aplicação manual era inevitável, consumindo consideravelmente tempo e esforço

# Propósitos

# Propósitos

- ▶ Desenvolver um software de linha de comando

# Propósitos

- ▶ Desenvolver um software de linha de comando — de natureza prototípica —

# Propósitos

- ▶ Desenvolver um software de linha de comando — de natureza prototípica — para a formatação de código-fonte

# Propósitos

- ▶ Desenvolver um software de linha de comando — de natureza prototípica — para a formatação de código-fonte, tendo como objetivos

# Propósitos

- ▶ Desenvolver um software de linha de comando — de natureza prototípica — para a formatação de código-fonte, tendo como objetivos
  - ▶ Abranger uma gama de linguagens de programação

# Propósitos

- ▶ Desenvolver um software de linha de comando — de natureza prototípica — para a formatação de código-fonte, tendo como objetivos
  - ▶ Abranger uma gama de linguagens de programação
  - ▶ proporcionar a formatação personalizada via a linguagem de consulta do *Tree-Sitter*



# A Formatação de Código-Fonte

# A Formatação de Código-Fonte

- ▶ Desde os primórdios da computação, métodos foram desenvolvidos para garantir que a saída impressa fosse formatada de maneira esteticamente agradável (HARRIS, 1956 apud YELLAND, 2015, p. 1)

# A Formatação de Código-Fonte

- ▶ Desde os primórdios da computação, métodos foram desenvolvidos para garantir que a saída impressa fosse formatada de maneira esteticamente agradável (HARRIS, 1956 apud YELLAND, 2015, p. 1)
- ▶ Esses métodos ganharam popularidade sob o termo “*prettyprinting*”

# A Formatação de Código-Fonte

- ▶ Desde os primórdios da computação, métodos foram desenvolvidos para garantir que a saída impressa fosse formatada de maneira esteticamente agradável (HARRIS, 1956 apud YELLAND, 2015, p. 1)
- ▶ Esses métodos ganharam popularidade sob o termo “*prettyprinting*”
- ▶ No desenvolvimento de software, o *prettyprinting* é conhecido como formatação de código-fonte

## A Formatação de Código-Fonte

- ▶ Durante as décadas de 60 e 70, a linguagem de programação LISP proporcionou condições favoráveis para o avanço da formatação de código (YELLAND, 2015, p. 2)

## A Formatação de Código-Fonte

- ▶ Durante as décadas de 60 e 70, a linguagem de programação LISP proporcionou condições favoráveis para o avanço da formatação de código (YELLAND, 2015, p. 2)
- ▶ Em 1967, Bill Gosper desenvolveu o *GRINDEF*

## A Formatação de Código-Fonte

- ▶ Durante as décadas de 60 e 70, a linguagem de programação LISP proporcionou condições favoráveis para o avanço da formatação de código (YELLAND, 2015, p. 2)
- ▶ Em 1967, Bill Gosper desenvolveu o *GRINDEF*: considerado o primeiro *prettyprinter* a mensurar o tamanho das linhas e ter ciência de sua localização no arquivo (GOSPER, 2023; GRIESEMER, 2022)

# A Formatação de Código-Fonte

- ▶ Durante as décadas de 60 e 70, a linguagem de programação LISP proporcionou condições favoráveis para o avanço da formatação de código (YELLAND, 2015, p. 2)
- ▶ Em 1967, Bill Gosper desenvolveu o *GRINDEF*: considerado o primeiro *prettyprinter* a mensurar o tamanho das linhas e ter ciência de sua localização no arquivo (GOSPER, 2023; GRIESEMER, 2022)
- ▶ Posteriormente, Oppen (1980) apresentou algoritmo inovador de formatação de código-fonte



## A Formatação de Código-Fonte

- ▶ Durante as décadas de 60 e 70, a linguagem de programação LISP proporcionou condições favoráveis para o avanço da formatação de código (YELLAND, 2015, p. 2)
- ▶ Em 1967, Bill Gosper desenvolveu o *GRINDEF*: considerado o primeiro *prettyprinter* a mensurar o tamanho das linhas e ter ciência de sua localização no arquivo (GOSPER, 2023; GRIESEMER, 2022)
- ▶ Posteriormente, Oppen (1980) apresentou algoritmo inovador de formatação de código-fonte: destacava-se por sua capacidade de formatar código derivado de qualquer linguagem de programação

# A Formatação de Código-Fonte

- ▶ Durante as décadas de 60 e 70, a linguagem de programação LISP proporcionou condições favoráveis para o avanço da formatação de código (YELLAND, 2015, p. 2)
- ▶ Em 1967, Bill Gosper desenvolveu o *GRINDEF*: considerado o primeiro *prettyprinter* a mensurar o tamanho das linhas e ter ciência de sua localização no arquivo (GOSPER, 2023; GRIESEMER, 2022)
- ▶ Posteriormente, Oppen (1980) apresentou algoritmo inovador de formatação de código-fonte: destacava-se por sua capacidade de formatar código derivado de qualquer linguagem de programação
  - ▶ O algoritmo necessitava que o código fosse anotado com espaços em branco e delimitadores especiais para marcar o início e fim de blocos

# A Formatação de Código-Fonte

- ▶ Durante as décadas de 60 e 70, a linguagem de programação LISP proporcionou condições favoráveis para o avanço da formatação de código (YELLAND, 2015, p. 2)
- ▶ Em 1967, Bill Gosper desenvolveu o *GRINDEF*: considerado o primeiro *prettyprinter* a mensurar o tamanho das linhas e ter ciência de sua localização no arquivo (GOSPER, 2023; GRIESEMER, 2022)
- ▶ Posteriormente, Oppen (1980) apresentou algoritmo inovador de formatação de código-fonte: destacava-se por sua capacidade de formatar código derivado de qualquer linguagem de programação
  - ▶ O algoritmo necessitava que o código fosse anotado com espaços em branco e delimitadores especiais para marcar o início e fim de blocos
  - ▶ Assim, o código a ser fornecido ao algoritmo precisaria ser processado por uma ferramenta intermediária capaz de compreender a sintaxe da linguagem e fornecer um código anotado de forma adequada, permitindo que o algoritmo realizasse a formatação apropriada

# A Formatação de Código-Fonte

- ▶ Recentemente, [Yelland \(2015\)](#) descreveu um algoritmo que visa otimizar o layout do código em relação a uma noção intuitiva de custo de layout

# A Formatação de Código-Fonte

- ▶ Recentemente, [Yelland \(2015\)](#) descreveu um algoritmo que visa otimizar o layout do código em relação a uma noção intuitiva de custo de layout
  - ▶ Notavelmente, entre as abstrações de programação empregadas para facilitar sua aplicação em diversas linguagens e políticas de layout de código, destacam-se os *combinators*

# A Formatação de Código-Fonte

- ▶ Recentemente, [Yelland \(2015\)](#) descreveu um algoritmo que visa otimizar o layout do código em relação a uma noção intuitiva de custo de layout
  - ▶ Notavelmente, entre as abstrações de programação empregadas para facilitar sua aplicação em diversas linguagens e políticas de layout de código, destacam-se os *combinators*: funções geradoras que descrevem layouts alternativos para o código-fonte

# Referências



GOSPER, Ralph William. **Twubblesome Twelve**. Disponível em: <http://gospers.org/bill.html>. Acesso em: 21 mai. 2023.



GRIESEMER, Robert. **The Cultural Evolution of gofmt**. Google Research. 2022. Disponível em: <https://go.dev/talks/2015/gofmt-en.slide>. Acesso em: 20 mai. 2023.



HARRIS, R. W. Keyboard Standardization. **Western Union Technical Review**, v. 10, n. 1, p. 37–42, 1956.



OPPEN, Derek C. Prettyprinting. **ACM Transactions on Programming Languages and Systems**, v. 2, n. 4, p. 465–483, out. 1980. DOI: 10.1145/357114.357115.



YELLAND, Phillip M. A New Approach to Optimal Code Formatting. **Google Research**, 2015. Disponível em: <https://research.google.com/pubs/archive/44667.pdf>. Acesso em: 7 abr. 2023.