

# *Witch Cooking*

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

Átila Gama Silva

20 de novembro de 2023

# Motivações

- ▶ As 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

# Objetivos Gerais

- ▶ 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*

# Objetivos Específicos

- ▶ Desenvolver um algoritmo de formatação fundamentado no *Tree-Sitter*
- ▶ Definir configurações de estilização para o predicado `set!`, nativo da linguagem de consulta do *Tree-Sitter*
- ▶ Estender os predicados embutidos da linguagem de consulta, proporcionando predicados basais para a formatação.

# 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)
- ▶ 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 um algoritmo inovador capaz de formatar código-fonte derivado de qualquer linguagem de programação
  - ▶ O algoritmo necessitava que o código fosse anotado — por uma ferramenta intermediária — com espaços em branco e delimitadores especiais para marcar o início e fim de blocos

# 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

## O Sistema *Tree-Sitter*

- ▶ O *Tree-Sitter* ([TREE-SITTER... , 2023](#)) é um sistema multilíngue de análise sintática para ferramentas de programação, desenvolvido como uma tentativa de solucionar problemas presentes nas ferramentas de análise sintática da época, tendo como objetivos
  - ▶ Produzir árvores de sintaxe a partir da análise de códigos escritos em várias linguagens
  - ▶ Implementar a análise incremental, permitindo a atualização da árvore de sintaxe em tempo real
  - ▶ Expor através da árvore de sintaxe os nós representando suas construções gramaticais no código (e.g., classes, funções, declarações, etc.)
  - ▶ Ser livre de dependências, assim beneficiando sua adoção e aplicabilidade



## O Sistema *Tree-Sitter*

- ▶ Além disso, o *Tree-Sitter* oferece uma pequena linguagem de consulta declarativa que é capaz de expressar padrões da árvore sintática por meio de *S-expressions* e buscar correspondências
- ▶ Essa linguagem suporta operadores que permitem
  - ▶ A captura de nós
  - ▶ A quantificação de nós, análoga às expressões regulares
  - ▶ O agrupamento de nós
  - ▶ As alternâncias de nós
  - ▶ O uso de *wildcards*
  - ▶ A ordenação de nós

## O Sistema *Tree-Sitter*

- ▶ Adicionalmente, a linguagem de consulta permite o uso de predicados — funções arbitrárias geralmente utilizadas para filtrar nós ou realizar verificações mais complexas durante a busca de padrões —, sejam eles *builtins* ou estendidos por meio de uma API

# Referências



GOSPER, Ralph William. **Twubblesome Twelve**. Disponível em: <http://gosper.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.



TREE-SITTER: a parsing system for programming tools. Versão 0.20.8. Tree-Sitter. Disponível em: <https://tree-sitter.github.io/>. Acesso em: 6 abr. 2023.

# Referências



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.