****

**CENTRO UNIVERSITÁRIO NEWTON PAIVA**

Relatório de atividades de Extensão

Compilo Java

Alunos:

Aaron Lopes

Jonathan Oldani

Marcos Antônio Gabriel

Victor Pacheco

Walisson Macedo

Orientador:

Lucas Ferreira

# BELO HORIZONTE 2025

# SUMÁRIO

1. INTRODUÇÃO ..................................................................... [pág. X]
2. OBJETIVO ............................................................................ [pág. X]
3. JUSTIFICATIVA .................................................................... [pág. X]
4. SOBRE O COMPILADOR ..................................................... [pág. X]
5. PROCESSO DE DESENVOLVIMENTO ................................ [pág. X]
6. RESULTADOS ALCANÇADOS ............................................. [pág. X]
7. CONCLUSÃO ........................................................................ [pág. X]

# INTRODUÇÃO

No campo da Ciência da Computação, o estudo e o desenvolvimento de compiladores representam um pilar fundamental, pois conectam a linguagem humana de programação à linguagem de máquina. Compreender seu funcionamento interno é essencial para a formação de profissionais da área. Neste contexto, o presente trabalho apresenta o projeto Compilo Python, uma ferramenta de propósito didático cujo objetivo é a implementação de um compilador funcional e de uma máquina virtual simplificada, utilizando a linguagem Python como plataforma de desenvolvimento.

O Compilo Python foi projetado para analisar, validar e traduzir um código-fonte escrito em uma linguagem específica, deliberadamente limitada a um conjunto de até 30 comandos. Sua arquitetura foi concebida de forma modular, segmentando o processo nas três fases canônicas da compilação: análise léxica, sintática e semântica. Essa estrutura não apenas organiza o desenvolvimento, mas também garante a correta interpretação do código e culmina na geração de um código intermediário, que por sua vez, serve de entrada para a máquina virtual.

Complementar ao compilador, a máquina virtual simplificada foi desenvolvida para executar o referido código intermediário. Ela é capaz de processar um conjunto de instruções fundamentais, incluindo atribuições de variáveis e estruturas de controle de fluxo. Para a execução de operações aritméticas e cálculos matemáticos avançados, suas funcionalidades foram implementadas utilizando o módulo math do Python. Isso permite que a linguagem do Compilo Python suporte um vasto conjunto de funções, como o cálculo de raiz quadrada (math.sqrt), potenciação (math.pow), funções trigonométricas (math.sin, math.cos) e operações de arredondamento (math.ceil, math.floor). A interação do usuário com o sistema se dará por meio de uma interface de linha de comando (terminal), onde o código-fonte será inserido para compilação.

Embora a limitação do léxico possa parecer restritiva, essa decisão, combinada com a integração de uma biblioteca matemática robusta, constitui um elemento central do propósito didático do projeto. Ao reduzir a complexidade da linguagem-fonte, o Compilo Python permite uma imersão aprofundada nos mecanismos essenciais da compilação, como a construção de tabelas de símbolos, a manipulação da Árvore de Sintaxe Abstrata (AST) e o mapeamento de comandos para chamadas de bibliotecas de sistema. Dessa forma, o projeto se estabelece como uma valiosa ferramenta de estudo prático para estudantes das disciplinas de automação de linguagens, teoria da computação e engenharia de software.

# OBJETIVO

O objetivo central deste trabalho é a concepção e implementação de um ecossistema de compilação didático, composto por um compilador e uma máquina virtual, ambos desenvolvidos em Python. A finalidade é criar uma ferramenta prática que não apenas demonstre as etapas fundamentais do processo de compilação, mas que também ilustre como uma linguagem de programação customizada pode fornecer acesso a funcionalidades de bibliotecas de sistema robustas.

Para alcançar este propósito, o projeto se concentra em dois componentes essenciais e interdependentes:

Objetivo Central

O objetivo central deste trabalho é a concepção e implementação de um ecossistema de compilação didático, composto por um compilador e uma máquina virtual, ambos desenvolvidos em Python. A finalidade é criar uma ferramenta prática que não apenas demonstre as etapas fundamentais do processo de compilação, mas que também ilustre como uma linguagem de programação customizada pode fornecer acesso a funcionalidades de bibliotecas de sistema robustas.

Para alcançar este propósito, o projeto se concentra em dois componentes essenciais e interdependentes:

Componentes do Projeto

O Compilador

O primeiro componente é um compilador com arquitetura modular (análise léxica, sintática e semântica), cuja função é analisar, validar e traduzir um código-fonte. Este código será escrito em uma linguagem própria, com um léxico limitado a aproximadamente 30 comandos pré-definidos. O principal objetivo do compilador é atuar como um tradutor, convertendo os comandos de alto nível da linguagem-fonte em um código de instruções de baixo nível (bytecode), que será compreensível para a Máquina Virtual.

A Máquina Virtual (VM)

O segundo pilar do projeto, a VM, serve como o ambiente de execução para o código intermediário gerado pelo compilador. O seu principal diferencial reside na implementação de um poderoso motor de cálculo, que se apoia diretamente no módulo math da biblioteca padrão do Python. Esta integração permite que a VM execute um amplo espectro de operações matemáticas, que vão muito além da aritmética básica. O objetivo é que a linguagem-fonte possa suportar comandos que sejam mapeados para funções como:

Operações Aritméticas Avançadas: Comandos para cálculos de raiz quadrada (math.sqrt), cúbica (math.cbrt) e potenciação (math.pow), permitindo expressões matemáticas complexas.

Funções Trigonométricas: Suporte a comandos para cálculos de seno (math.sin), cosseno (math.cos) e tangente (math.tan), com o apoio de métodos de conversão angular como math.radians para facilitar a manipulação de ângulos.

Manipulação e Comparação de Dados: Comandos que utilizam métodos de arredondamento (math.ceil, math.floor), comparação para obtenção de valores máximo (math.max) e mínimo (math.min), e cálculo de valor absoluto (math.fabs).

Capacidades Adicionais: Implementação de comandos para geração de números pseudoaleatórios (random.random) e o uso de constantes fundamentais como Pi (math.pi) em cálculos.

Dessa forma, o objetivo se materializa na criação de uma ferramenta completa onde o processo de compilação não apenas valida a sintaxe, mas efetivamente traduz comandos expressivos em operações matemáticas poderosas, que são finalmente executadas pela Máquina Virtual.

# JUSTIFICATIVA

O estudo e a implementação de compiladores são etapas cruciais na formação de profissionais de Ciência da Computação, pois revelam o mecanismo subjacente que converte a lógica de programação de alto nível em instruções executáveis pela máquina. Embora o mercado de trabalho raramente exija a criação de um compilador do zero, a compreensão de seus princípios fundamentais, como as fases de análise léxica, sintática e semântica, é essencial para a escrita de código mais eficiente, a depuração de erros complexos e o desenvolvimento de ferramentas de automação de linguagem. O projeto Compilo Python se justifica, portanto, como uma ferramenta didática valiosa para preencher essa lacuna de conhecimento, proporcionando uma experiência prática e aprofundada.

Ao contrário de compiladores de linguagens comerciais, que apresentam grande complexidade e uma curva de aprendizado íngreme, o Compilo Python adota uma abordagem pedagógica. A linguagem-fonte foi deliberadamente simplificada, limitando seu léxico a um conjunto de aproximadamente 30 comandos. Essa restrição estratégica não diminui o valor do projeto; pelo contrário, ela permite que os estudantes se concentrem nos mecanismos essenciais da compilação, como a construção de tabelas de símbolos, o gerenciamento da Árvore de Sintaxe Abstrata (AST) e o mapeamento de instruções para a máquina virtual. Essa abordagem focada garante que o aprendizado seja profundo e relevante, sem a distração de lidar com a complexidade de uma linguagem real, como Python ou C++.

Além da simplificação, outro pilar fundamental da justificativa do projeto é a integração com a biblioteca padrão do Python. A máquina virtual, em vez de re-implementar operações matemáticas complexas, utiliza diretamente o módulo math. Essa escolha de design não apenas demonstra como uma linguagem de programação customizada pode interagir com bibliotecas de sistema, mas também ilustra o conceito de reuso de código. Esse método oferece uma maneira elegante de expandir as capacidades da linguagem-fonte de forma poderosa e eficiente, permitindo que a VM execute funções avançadas como cálculos trigonométricos e logarítmicos, que seriam inviáveis de serem implementados manualmente em um projeto com foco didático.

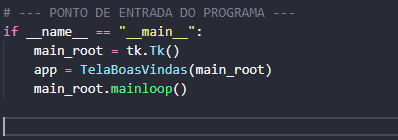
Dessa forma, o Compilo Python se justifica como uma ferramenta completa e estrategicamente concebida para o ensino. Ele não apenas desmistifica o processo de compilação, mas também serve como uma ponte entre a teoria acadêmica e a aplicação prática, validando-se como um recurso inestimável para estudantes das áreas de Teoria da Computação, Engenharia de Software e Automação de Linguagens. O projeto oferece uma experiência de aprendizado hands-on que reforça conceitos teóricos de maneira concreta, preparando os futuros profissionais para desafios mais complexos no campo da tecnologia.

1. **SOBRE O COMPILADOR**
2. **PROCESSO DE DESENVOLVIMENTO**

O desenvolvimento do projeto foi realizado na linguagem Python, utilizando a biblioteca tkinter para a construção da interface gráfica do usuário (GUI). A arquitetura do software foi concebida de forma modular, separando a lógica da tela de apresentação da lógica do interpretador principal. A seguir, o código é fracionado e explicado em suas partes constituintes.

#### 5.1 Estrutura Geral e Ponto de Entrada

O programa é inicializado a partir de um ponto de entrada padrão em Python, que verifica se o script está sendo executado diretamente. Uma vez confirmado, ele instancia a janela principal (main\_root) e a classe TelaBoasVindas, que serve como a interface inicial para o usuário.



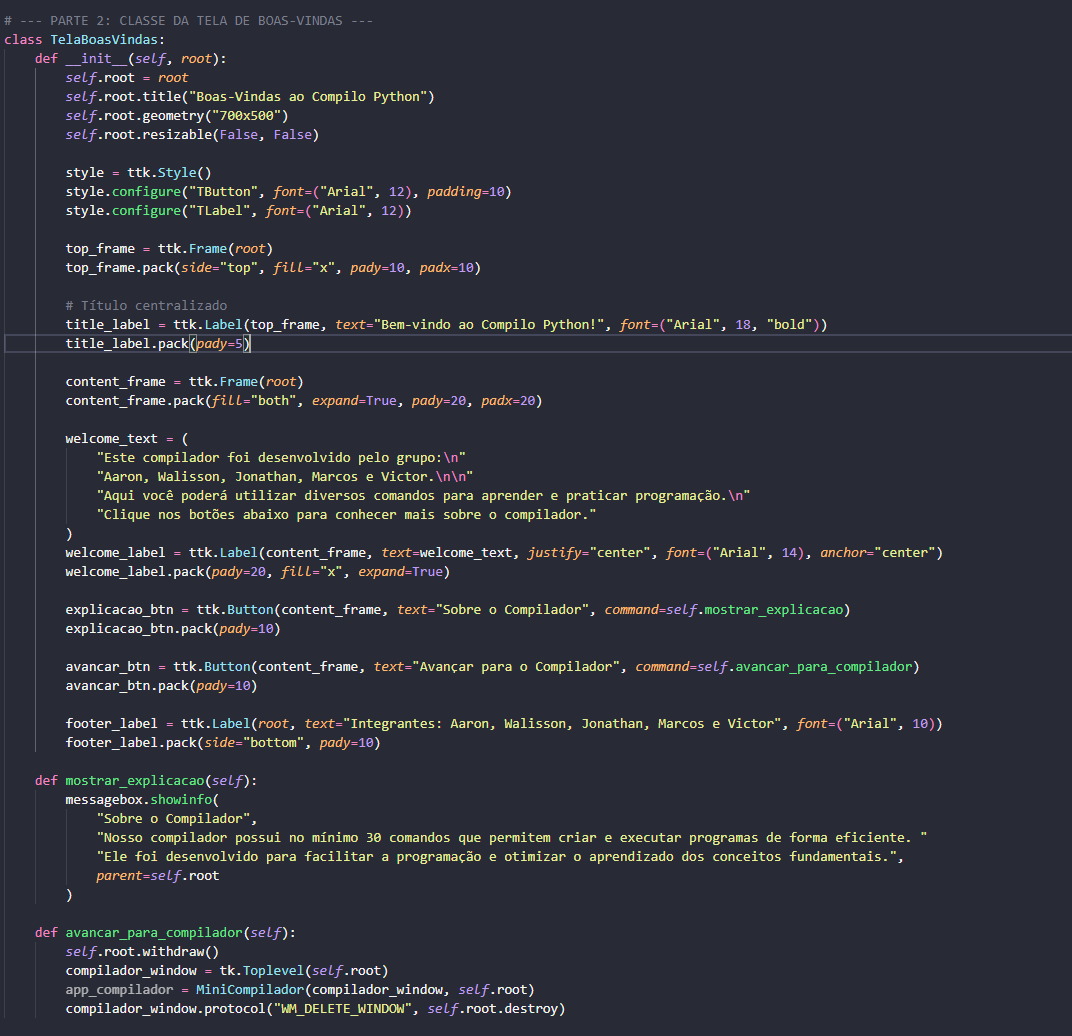
Este bloco de código é responsável por lançar a aplicação, começando pela tela de apresentação.

#### 

#### 5.2 Módulo de Apresentação (TelaBoasVindas)

A classe TelaBoasVindas gerencia a primeira janela que o usuário visualiza. Sua finalidade é apresentar o projeto, os desenvolvedores e fornecer as opções de navegação.

* Componentes Visuais: A tela é composta por um título, um texto de boas-vindas e dois botões principais: "Sobre o Compilador" e "Avançar para o Compilador".
* Funcionalidades:
  + O botão "Sobre o Compilador" aciona a função mostrar\_explicacao, que exibe uma caixa de diálogo (messagebox) com informações sobre o projeto.
  + O botão "Avançar para o Compilador" chama o método avancar\_para\_compilador. Este método é crucial para a transição entre as telas: ele esconde a janela de boas-vindas (self.root.withdraw()) e cria uma nova janela (Toplevel) onde a classe MiniCompilador é instanciada, efetivamente iniciando a ferramenta principal.



#### 

#### 

#### 

#### 5.3 Módulo do Interpretador (MiniCompilador)

Esta é a classe central do projeto, onde toda a lógica de interpretação e execução dos comandos ocorre.

##### 5.3.1 Interface do Interpretador

A interface do MiniCompilador é dividida em três áreas principais:

1. Área de Código (codigo\_area): Um campo de texto tk.Text onde o usuário insere os comandos da linguagem-fonte. O campo já vem pré-preenchido com um código de exemplo.
2. Painel de Controle: Contém os botões "Executar" e "Voltar", que permitem ao usuário processar o código ou retornar à tela de boas-vindas.
3. Área de Saída (saida\_area): Um campo de texto somente leitura que exibe os resultados da execução dos comandos ou eventuais mensagens de erro.

##### 5.3.2 Fluxo de Execução

Ao clicar no botão "Executar", o método iniciar\_execucao é chamado. Para melhorar a experiência do usuário, o processo foi dividido em duas etapas:

1. Exibição de Carregamento: Uma pequena janela pop-up "Aguarde" é exibida para informar ao usuário que o processamento está em andamento. Isso evita a impressão de que o programa travou durante a execução de comandos.
2. Processamento Assíncrono: A execução da lógica principal é agendada para ocorrer alguns milissegundos depois (root.after(200, ...)), garantindo que a janela de carregamento seja renderizada antes que o processamento intensivo comece.

##### 5.3.3 Lógica de Interpretação

O método processar\_codigo contém o núcleo do interpretador:

1. Leitura e Limpeza: O código-fonte é lido da codigo\_area e a saida\_area é limpa.
2. Análise Léxica Simplificada: O texto é dividido em linhas, e cada linha é dividida em "tokens" (palavras). O primeiro token é considerado o comando (ex: print, sqrt), e os tokens subsequentes são os argumentos.
3. Análise Sintática e Semântica (Estrutura de Controle): Uma estrutura condicional (if/elif/else) percorre os comandos. Para cada comando reconhecido, a lógica correspondente é executada. Se um comando não é reconhecido, uma mensagem de erro é exibida.
4. Execução: Os argumentos são convertidos para os tipos de dados necessários (geralmente float) e passados para os métodos correspondentes (ex: self.sqrt(x)). O resultado é então inserido na saida\_area.



Esta abordagem, embora simples, simula efetivamente as etapas de um interpretador, validando e executando cada instrução sequencialmente.

1. **RESULTADOS ALCANÇADOS**
2. **CONCLUSÃO**

**Link Github:**