Project Report: Design Decisions, Challenges, and Solutions

# Introduction

The project aimed to develop a simple functional programming language interpreter that supports key features such as function definitions, conditional statements, and basic arithmetic operations. The primary goal was to create a system that could parse and evaluate expressions, while efficiently handling errors and optimizing performance through lazy evaluation techniques. This report will discuss the design decisions made during the development process, the challenges encountered, and the solutions implemented to address these challenges.

# Design Decisions

1. Objectives  
The project aimed to build an efficient interpreter supporting key functional programming features like function definitions, lambda expressions, and immutability.

2. Function Definition  
Functions are defined using the `func` keyword, enabling users to create reusable named functions with parameters, which are essential for complex operations.

3. Immutability  
The interpreter enforces immutability, ensuring that values, once assigned, cannot be modified, aligning with functional programming principles.

4. Lambda Expressions  
Lambda expressions, defined with the `Lambady` keyword, allow for the creation of anonymous functions, enhancing the interpreter's flexibility and expressiveness.

# Core Components

1. Interpreter  
The interpreter executes the Abstract Syntax Tree (AST) generated by the parser, handling function calls, conditions, and arithmetic, while enforcing immutability and lazy evaluation.

2. Lexer  
The lexer tokenizes the source code, converting it into a stream of tokens that represent keywords, operators, and identifiers, which the parser uses to structure the code.

3. Parser  
The parser transforms the tokens from the lexer into an Abstract Syntax Tree (AST), providing a structured representation of the code for the interpreter to execute.