# Language Evolution - Fourward Programming Language

## 1. Initial Design Phase

- Core language concepts focused on beginner accessibility
- Syntax designed for simplicity and clarity
- Feature set selected based on practical learning needs

## 2. Development Milestones

### Version 0.1 - Foundation

- Implemented the basic interpreter structure
- Core language features:
  - Variable declarations with `let`
  - Arithmetic operations (+, -, *, /, %)
  - Built-in functions: `print`, `input`
  - Control structures: `if`, `else`, `while`
  - Support for comments using `#`

### Version 0.2 - Parser Refinement

- Improved expression parsing
- Enhanced error reporting and exception handling
- Refactored AST structure and environment logic
- Clarified syntax rules and grammar edge cases

### Version 0.3 - Current State

- Modular and robust interpreter
- Tokenizer, parser, and interpreter fully integrated
- Complete documentation set (spec, white paper, examples)
- Example-driven testing of all core language features

## 3. Key Design Decisions

### 3.1 Syntax Design

- Clear, minimal syntax
- C-style block structure with `{}` braces
- Explicit variable declarations using `let`
- Statements terminated with semicolons (`;`)
- Control structure conditions enclosed in parentheses

### 3.2 Feature Selection

- Focused on essential programming concepts
- Prioritized clarity and educational value
- Omitted complex features like classes or user-defined functions
- Designed for easy parsing and debugging

---

## 4. Current Features

### 4.1 Core Language Features

- Variable declarations and assignments
- Arithmetic and string operations
- Control flow: `if`, `else`, `while`
- Comments with `#`
- Built-in `print` and `input` functions

### 4.2 Implementation Details

- **Lexer**: Token stream generation using regex
- **Parser**: Abstract Syntax Tree (AST) construction
- **Interpreter**: AST traversal and evaluation
- **Environment**: Scoped variable tracking

---

## 5. Future Evolution

### 5.1 Planned Features

- Support for additional data types (e.g., arrays, booleans, null)
- File I/O capabilities
- User-defined functions and parameters
- Basic standard library utilities

### 5.2 Potential Improvements

- Performance enhancements and optimization
- Better runtime error messages
- Debugging and trace logging tools
- Interactive REPL or web-based execution environment

---

## 6. Lessons Learned

- Simple syntax makes a big difference in usability
- Manual testing is essential in early interpreter phases
- Good error messages are crucial for debugging
- Every new feature requires updates to lexer, parser, and interpreter

---