

Hochschule  
für Technik  
Stuttgart

## HOWDYLANG SYNTAX GUIDE



---

# Concept of Programming Language - HowdyLang Syntax Guide

---

### AUTHOR

Ulugbek Yarkinov: 42yaul1mst@hft-stuttgart.de

Hung Vinh Dang: 42dahu1mst@hft-stuttgart.de

Kang Chun-Yu: 42kach1mst@hft-stuttgart.de

Matthew Claflin: 42clma1mst@hft-stuttgart.de

2025-06-16

# Contents

<b>1</b>	<b>Formal language Definition</b>	<b>1</b>
1.1	Lexical Tokens . . . . .	1
1.2	Program Structure . . . . .	1
1.3	Statement Types . . . . .	2
1.4	Grammar Snippet in JavaCC . . . . .	3
<b>2</b>	<b>Language Design Decisions</b>	<b>4</b>
2.1	Team Contribution for the Project . . . . .	4
<b>3</b>	<b>Code Example &amp; Coverage</b>	<b>5</b>
3.1	Parsing Tree . . . . .	6
3.2	Result . . . . .	7
<b>4</b>	<b>Implementation &amp; Presentation</b>	<b>8</b>
<b>5</b>	<b>Conclusion</b>	<b>9</b>
	<b>List of Figures</b>	<b>I</b>
	<b>References</b>	<b>II</b>

# 1 Formal language Definition

HowdyLang is a string-only programming language with a cowboy-themed syntax. It uses JavaCC to define the lexical tokens and grammar rules. The language aims to be humorous and approachable, while still demonstrating fundamental programming constructs.

## 1.1 Lexical Tokens

HowdyLang defines the following tokens:

- HOLLER: print statement
- IF\_YALL, ELSE\_YALL: conditional statements
- WHILE\_YALL, DO\_YALL: loop constructs
- MAKE\_PROGRAMMING, GREAT\_AGAIN: delimit program start and end
- ASSIGN: includes '=', '<-', 'is', 'are', 'contains', 'contain'
- EQ: includes '==', 'equals', 'equal'
- STRING\_LITERAL, IDENTIFIER, SEMI, LPAR, RPAR, LBRA, RBRA

## 1.2 Program Structure

A HowdyLang program starts with MAKE\_PROGRAMMING and ends with GREAT\_AGAIN. Statements are executed in sequence and can be nested using block syntax.

Example:

```
MAKE_PROGRAMMING

username = "Cowboy";
HOLLER "Howdy, " username;

GREAT_AGAIN
```

## 1.3 Statement Types

HowdyLang supports the following kinds of statements:

- **Variable Assignment:** Assign values using one of six cowboy-styled assignment keywords. Only string literals or existing variables may be assigned.
  - Example: `name = "Buck";` or `boots is "snakeskin";`
- **Equality Comparison:** Compare variable values using `'=='`, `'equals'`, or `'equal'`.
- **Printing:** The `HOLLER` keyword is used to print one or more values to the console. You can combine strings and variable values.
  - Example: `HOLLER "Howdy partner!";`, `HOLLER "Name: " name;`
- **Conditional Statements:** Use `IF_YALL` to branch logic with optional `ELSE_YALL`. Statements can be enclosed in curly braces or written inline.
  - Example: `IF_YALL (name == "Buck") HOLLER "That's Buck!";`
- **Loops:** Two types of loops are supported:
  - `WHILE_YALL (condition)` — Repeats while the condition is true.
  - `DO_YALL ("N") TIMES` — Executes a block N times (where N is a string literal).
  - Example: `DO_YALL ("3") TIMES HOLLER "Yeehaw!";`
- **Block Scoping:** Curly braces `{}` are used to group multiple statements under a single control structure, such as an `IF_YALL` or `WHILE_YALL`.
  - Example:

```
IF_YALL (role == "rancher") {
    HOLLER "Howdy, partner!";
    HOLLER "You've got work to do!";
}
```

## 1.4 Grammar Snippet in JavaCC

This grammar below covers all required syntax elements, including assignments, conditions, loops, and print statements. Optional blocks and syntactic sugar (like multiple assignment forms) are included to improve expression capabilities:

```
void Script() : {  
    Statements()  
}  
  
void Statements() : {  
    ( Statement() )+  
}  
  
void Statement() : {  
    Assignment()  
    | Print()  
    | If()  
    | While()  
    | Block()  
    | DoTimes()  
}  
  
void Assignment() : {  
    Value()  
}  
  
void Condition() : {  
    Value()  
}  
  
void Value() : {  
    <STRING_LITERAL> |  
}
```

## 2 Language Design Decisions

HowdyLang was developed to make the learning process fun, engaging, and intuitive through a themed, readable syntax. The design of the language reflects our effort to reduce the entry barrier for new programmers while allowing experienced users to appreciate its structure. Hence, the following decisions were made as part of our design process:

- **No explicit typing:** All values in HowdyLang are treated as strings. This simplifies the syntax and parsing, making it easier for learners to grasp the basic mechanics of a programming language without worrying about types.
- **Expressive keywords:** Inspired by cowboy lingo, keywords like `HOLLER`, `IF_YALL`, and `DO_YALL` were chosen to inject personality into the language. This encourages users to connect emotionally with the code and remember syntax through Texas cultural language.
- **Java and JavaCC:** We selected JavaCC as our parser generator and Java as the execution backend. This allowed us to practice working with abstract syntax trees (ASTs), parser construction, and code translation.
- **Whitespace flexibility:** HowdyLang does not impose strict formatting rules, allowing users to write clean or free-form code as they wish.

Overall, the development of HowdyLang was driven by our initiative to combine storytelling, syntax, and software engineering principles in a cohesive, user-friendly and interesting cultural experience.

### 2.1 Team Contribution for the Project

Our team began by brainstorming several thematic ideas for the language, including **Texas**, **Morse code**, **Satire**, and **Brainrot**. After evaluating readability, user engagement, and clarity, we unanimously decided to pursue the Texas-inspired theme. It provided a unique balance of humor and readability.

Throughout development, each team member contributed individually—some focused on enhancing the syntax with expressive keywords and optional emoji functionalities, while others restructured the grammar to reduce its resemblance to conventional Java syntax. Together, we designed and executed a list of test scripts to ensure full coverage of all required language features, such as loops, conditionals, string comparison, and printing. Finally, we merged the strengths of each idea and contribution into a cohesive final version of HowdyLang, balancing creativity with technical completeness.

### 3 Code Example & Coverage

The following HowdyLang code demonstrates all required coverage points: Strings, Variables, Assignments, String Equality, Conditional Logic, Loops, and Printing.

Listing 1: HowdyLang Sample Script

```
MAKE_PROGRAMMING

name = "Billy Joe";           // Assign string to variable using '='
role <- "drifter";            // Assign string using '<-'
bootsOn is "false";           // Assign string using 'is'
rounds contains "1";          // Assign string using 'contains'
loopCount = "0";              // Initialize loop counter

HOLLER "YEEHAW!";             // Print a cowboy greeting
HOLLER "Howdy, " name "!";    // Print a message with a variable

IF_YALL (role == "drifter")   // Conditional check using '==' without {}
HOLLER "Ridin' solo. Grab yer hat."; // If true, print message
ELSE_YALL
HOLLER "Sheriff's back in town!"; // Else branch

WHILE_YALL (bootsOn equals "false") { // While loop using 'equals'
HOLLER "Get them boots on!";         // Loop body prints and updates
bootsOn = "true";                     // Change loop condition
  IF_YALL (role == "drifter") {       // Nested conditional inside loop
    HOLLER "Walkin' into town without backup...";
  }
}

HOLLER "Boots on? " bootsOn; // Print current value of bootsOn

DO_YALL (rounds) TIMES          // Fixed repetition loop using string count
HOLLER "Wranglin' time!";      // Loop body prints message

HOLLER "All done, partner!";    // Final output

GREAT_AGAIN
```

### 3.1 Parsing Tree

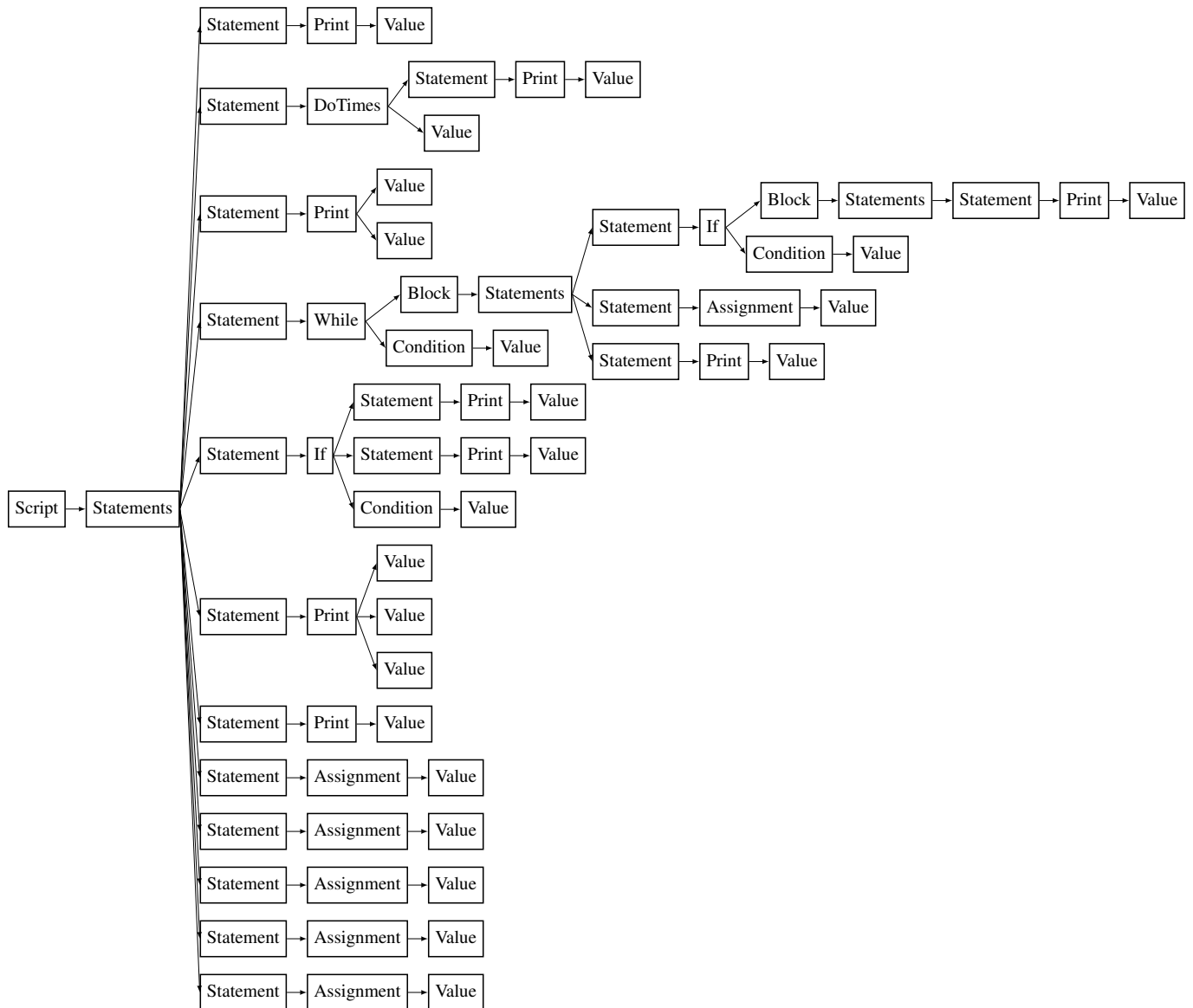
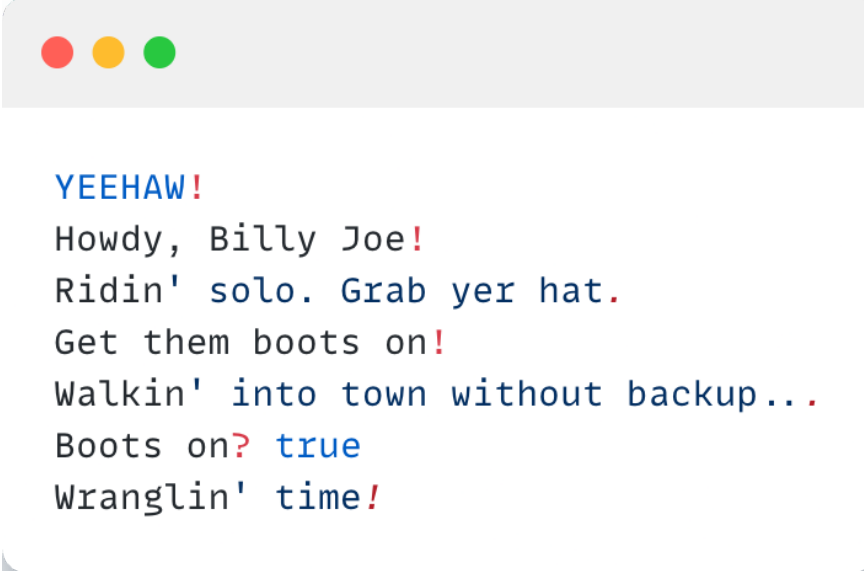


Figure 1: HowdyLang Abstract Syntax Tree (AST)



## 3.2 Result

Hence, the result of the code is shown in Figure 2:

A terminal window with a light gray title bar containing three colored circles (red, yellow, green). The output text is displayed in a monospaced font with syntax highlighting: 'YEEHAW!' in blue, 'Howdy, Billy Joe!' in black, 'Ridin' solo. Grab yer hat.' in black, 'Get them boots on!' in black, 'Walkin' into town without backup...' in black, 'Boots on? true' in black, and 'Wranglin' time!' in black.

```
YEEHAW!  
Howdy, Billy Joe!  
Ridin' solo. Grab yer hat.  
Get them boots on!  
Walkin' into town without backup...  
Boots on? true  
Wranglin' time!
```

Figure 2: HowdyLang Execution Output

## 4 Implementation & Presentation

Our implementation is structured around three key components that enable parsing, interpretation, and testing of HowdyLang:

- **HowdyLangParser.jjt:** This file defines the formal grammar of the language using JavaCC's JJTree syntax. It specifies all token definitions (keywords, identifiers, string literals) and the parsing rules for assignments, conditionals, loops, print statements, and blocks. It is responsible for generating the Abstract Syntax Tree (AST).
- **Interpreter.java:** This Java class traverses the generated AST and executes the program logic directly. It handles variable assignments, evaluates conditions, and controls flow structures like 'IF\_YALL', 'WHILE\_YALL', and 'DO\_YALL'.
- **presentation.hl:** This is a sample HowdyLang source file that contains a complete demonstration of all major language features. It is used to test both parsing and execution during development and presentation.

These three files work together to demonstrate a full pipeline: from reading HowdyLang code, parsing it into a structured tree, and finally interpreting the logic as runnable behavior. Hence, In our project demonstration, we will:

- Step 1: Present the HowdyLang script (Chapter 3)
- Step 2: Parse the script to generate an Abstract Syntax Tree
- Step 3: Traverse the AST and output translated Java code
- Step 4: Compile and execute the generated Java code
- Step 5: Showcase the console output

## 5 Conclusion

Afterall, HowdyLang is a creative and expressive language designed to teach core programming concepts using a thematic syntax. Through this project, we successfully:

- Designed a formal grammar and implemented a custom parser in JavaCC
- Built an AST-based interpreter and translator to Java
- Demonstrated all required language features with themed syntax
- Presented a full development and execution pipeline

HowdyLang brings fun to foundational language design, making it a great tool for both learning and teaching programming language theory. In addition, to run the code can find the source code and documentation on GitHub [1].

## List of Figures

1	HowdyLang Abstract Syntax Tree (AST) . . . . .	6
2	HowdyLang Execution Output . . . . .	7

## References

- [1] Ulugbek Yarkinov and David King. Howdylang - cowboy-themed educational language. <https://github.com/UlugbekYarkinov/howdy>, 2025. Accessed: 2025-06-15.