NOOBIE2 Programming Language
Complete Documentation

Luca Tarantola

June 2025

Contents

1 Introduction

NOOBIE is a simple, intuitive programming language specifically designed for beginners who are taking their first steps into the world of programming. Unlike complex languages that can overwhelm newcomers, NOOBIE focuses on readability, simplicity(is case insensitive), and educational value.

The language emphasizes natural language constructs, making it easy for students to understand programming concepts without getting bogged down in complex syntax. NOOBIE version 2.2 supports fundamental programming constructs including variables, data types, user input/output, conditionals, and basic operations.

This documentation is divided into two main sections: **Simple NOOBIE** covers the basic concepts and commands needed to write your first programs, while **Advanced NOOBIE** explores more sophisticated features for users ready to tackle complex programming challenges.

2 Simple NOOBIE

This section covers the fundamental concepts and commands that every NOOBIE2 programmer should master first.

2.1 Data Types

NOOBIE supports five basic data types that cover most programming needs:

- **INT**: Integer numbers for whole values (e.g., 42, -10, 0)
- **FLOAT**: Decimal numbers for precise calculations (e.g., 3.14, -0.5, 2.0)
- **STR**: Text strings for words and sentences (e.g., "Hello", 'World')
- **CHAR**: Single characters for individual letters or symbols (e.g., 'A', '5', '!')
- **BOOL**: Boolean values for true/false logic (true, false, null)

2.2 Basic Commands

2.2.1 SAY - Display Output

The `SAY` command displays text or variable values to the user.

Syntax: `SAY "message"` or `SAY variable_name`

```
SAY "Hello, World!"
SAY name
SAY "Your age is: @age"
SAY "{5+5}"
```

2.2.2 CREATE - Variable Declaration

The `CREATE` command declares new variables with optional initial values.
Syntax: `CREATE [CONST] <type> <name> [value]`

```
CREATE INT age 25
CREATE STR name "John"
CREATE STR surname Mark
CREATE CONST FLOAT pi 3.14159
CREATE BOOL is_student
```

If the value of the variable is not specified, it will be automatically set as: **0** for INT, **0.0** for FLOAT, "" for STR, **'\0'** for CHAR, **null** for BOOL

2.2.3 LISTEN - User Input

The `LISTEN` command gets input from the user and stores it in a variable.
Syntax: `LISTEN <type> [variable_name] "prompt"`

```
LISTEN STR name "Enter your name: "
LISTEN INT age "How old are you? "
LISTEN BOOL "Are you a student? (true/false): "
LISTEN INT num prompt_variable
```

2.2.4 CHANGE - Modify Variables

The `CHANGE` command updates the value of existing variables.
Syntax: `CHANGE <variable> <new_value>`

```
CHANGE age 26
CHANGE message "Updated text"
CHANGE is_active true
CHANGE name @new_name
CHANGE sum {current_value + sum}
```

2.2.5 WHILE - Loop Execution

The `WHILE` command repeats a block of code as long as a condition remains true.
Syntax: `WHILE <condition> DO <code> ENDO`

```
CREATE INT counter 1
WHILE counter <= 5 DO
    SAY "Count: " counter
    INCREMENT counter
```

```
ENDO
```

2.2.6 EXIT - Program Termination

The `EXIT` command immediately terminates the program execution. This is useful for ending programs early based on certain conditions or user input.

Syntax: `EXIT` or `EXIT "message"`

```
# Simple exit
EXIT

# Exit with message
EXIT "Program terminated by user"

# Conditional exit
CREATE STR user_choice
LISTEN STR user_choice "Do you want to continue? (yes/no): "
IF user_choice == "no" DO
    EXIT "Goodbye!"
ENDO
```

2.3 Basic Conditionals

Conditional statements allow your program to make decisions based on different conditions.

Syntax:

```
IF <condition> DO
    <statements>
ELSE
    <statements>
ENDO
```

Example:

```
CREATE INT age
LISTEN INT age "Enter your age: "
IF age >= 18 DO
    SAY "You are an adult!" end
ELSE
    SAY "You are a minor." end
```

```
ENDO
```

2.4 Loops with WHILE

The `WHILE` command allows you to repeat code as long as a condition is true, making it perfect for creating interactive programs and repetitive tasks.

Syntax:

```
WHILE <condition> DO
    <statements>
ENDO
```

Example - Simple Counter:

```
CREATE INT count 1
WHILE count <= 3 DO
    SAY "This is iteration " count end
    INCREMENT count
ENDO
SAY "Loop finished!"
```

Example - User Input Loop:

```
CREATE STR user_input ""
WHILE user_input != "quit" DO
    LISTEN STR user_input "Enter a word (or 'quit'
  to exit): "
    IF user_input != "quit" DO
        SAY "You entered: " user_input end
    ENDO
ENDO
SAY "Goodbye!"
```

2.5 Simple Examples

2.5.1 Hello World Program

```
# My first NOOBIE2 program
SAY "Hello, World!" end
SAY "Welcome to NOOBIE2 programming!" end
```

2.5.2 Personal Greeting

```
# Personal greeting program
CREATE STR name
CREATE INT age

LISTEN STR name "What's your name? "
LISTEN INT age "How old are you? "

SAY "Hello, " name "!" end
SAY "You are " age " years old." end
```

2.5.3 Simple Calculator

```
# Basic addition calculator
CREATE INT num1
CREATE INT num2

LISTEN INT num1 "Enter first number: "
LISTEN INT num2 "Enter second number: "

SAY "The sum is: {num1 + num2}" end
```

2.5.4 Counting Loop

```
# Count from 1 to 10
CREATE INT number 1

SAY "Counting from 1 to 10:" end
WHILE number <= 10 DO
    SAY number end
    INCREMENT number
ENDO
SAY "Finished counting!"
```

2.5.5 Program with User Control

```
# Program that lets user exit early
CREATE STR response

SAY "Welcome to the greeting program!"
LISTEN STR response "Press Enter to continue or type
    'exit' to quit: "

IF response == "exit" DO
    EXIT "Thanks for visiting!"
ENDO

SAY "Hello! Nice to meet you!" end
SAY "Program completed successfully." end
```

3 Advanced NOOBIE

This section covers more sophisticated features for users who have mastered the basics and are ready for complex programming challenges.

3.1 Advanced Variable Operations

3.1.1 Type Conversion

Convert variables between different data types using the `CONVERT` command.

Syntax: `CONVERT <variable> <new_type>`

```
CREATE INT number 42
CONVERT number STR
SAY "Number as string: " number

CREATE STR text "123"
CONVERT text INT
SAY "Text as integer: " text
```

3.1.2 Variable Manipulation

Advanced commands for manipulating variable values:

```
# Increment and decrement
CREATE INT counter 5
```

```
INCREMENT counter    # counter becomes 6
DECREMENT counter    # counter becomes 5

# Swap variables
CREATE STR first "Hello"
CREATE STR second "World"
SWAP first second    # first="World", second="Hello"

# Reset and delete
RESET counter        # Sets to default value for
  type
DEL unused_variable  # Removes variable from memory
```

3.2 Mathematical Operations

NOOBIE2 supports different operators:

- **Arithmetic Operators:** +, -, *, /, %, **
- **Relational Operators** <, >, !=, ==, >=, <=
- **Logical Operators** and, or, not, xor
- **Value Operators** @ (expand the value of a variable), ? (expand the type of a variable, only in strings), {} (evalute expression in strings)
- **Precision Control:** ROUND for floating-point precision

```
CREATE FLOAT pi 3.14159265
CREATE INT base 2
CREATE INT exponent 8

SAY "Pi rounded to 2 places: "
ROUND pi 2
SAY pi

SAY "2 to the power of 8:  {base ** exponent} " end
SAY "17 divided by 5: {17 / 5} " end
SAY "17 modulo 5: {17 % 5} " end
```

3.3 String Operations

Advanced string manipulation capabilities:

```
CREATE STR message "Hello World"

# String transformations
UPPERCASE message    # "HELLO WORLD"
SAY "Uppercase: " message

LOWERCASE message    # "hello world"
SAY "Lowercase: " message

REVERSE message      # "dlrow olleh"
SAY "Reversed: " message
```

3.4 Random Number Generation

Generate random values for games, simulations, and testing:

Syntax: `RANDOM <type> <min> <max> [variable]`

```
# Generate random integer between 1 and 100
RANDOM INT 1 100 dice_roll
SAY "You rolled: " dice_roll end

# Generate random float between 0.0 and 1.0
RANDOM FLOAT 0.0 1.0 probability
SAY "Random probability: " probability end

# Generate random boolean
RANDOM BOOL 1 2 coin_flip
IF coin_flip DO
    SAY "Heads!" end
ELSE
    SAY "Tails!" end
ENDO
```

3.5 Advanced Program Control

3.5.1 Conditional Program Termination

The `EXIT` command can be used strategically in complex programs to handle error conditions or user-requested termination:

```
# Error handling with EXIT
CREATE INT divisor
LISTEN INT divisor "Enter a divisor: "

IF divisor == 0 DO
    SAY "Error: Cannot divide by zero!" end
    EXIT "Program terminated due to invalid input"
ENDO

SAY "Result: {100 / divisor}" end
```

3.5.2 Menu-Driven Programs with EXIT

```
# Interactive menu program
CREATE STR choice ""

WHILE choice != "4" DO
    SAY "=== Main Menu ==="
    SAY "1. Say Hello" end
    SAY "2. Calculate Sum" end
    SAY "3. Show Random Number" end
    SAY "4. Exit Program" end

    LISTEN STR choice "Choose an option: "

    IF choice == "1" DO
        SAY "Hello, User!" end
    ELSE
        IF choice == "2" DO
            CREATE INT a
            CREATE INT b
            LISTEN INT a "First number: "
```

```
            LISTEN INT b "Second number: "
            SAY "Sum: {a + b}" end
        ELSE
            IF choice == "3" DO
                RANDOM INT 1 10 rand_num
                SAY "Random number: " rand_num end
            ELSE
                IF choice == "4" DO
                    EXIT "Thank you for using the program!"
                ELSE
                    SAY "Invalid choice. Please try again." end
                ENDO
            ENDO
        ENDO
    ENDO
ENDO
```

3.6 Advanced Syntax Features

3.6.1 Special Syntax Elements

- **Direct Variable Use:** Simply use the variable name in expressions and output
- **String Variable Output:** Use quotes with variable names for formatted output
- **Mathematical Expressions:** {expression} (evaluates math in strings)
- **Special Variables:** @end (represents newline character)

3.6.2 Comments

NOOBIE2 supports both single-line and multi-line comments:

```
# This is a single-line comment

## This is a
   multi-line comment
   that spans several lines ##

```

```
CREATE INT x 10  # Inline comment
```

3.7 Complex Examples

3.7.1 Advanced Calculator

```
# Advanced calculator with multiple operations and loop
CREATE FLOAT num1
CREATE FLOAT num2
CREATE STR operation
CREATE STR continue_calc "yes"

SAY "=== Advanced Calculator ===" end

WHILE continue_calc == "yes" DO
    LISTEN FLOAT num1 "Enter first number: "
    LISTEN STR operation "Enter operation (+, -, *, /, **, %): "
    LISTEN INT num2 "Enter second number: "

    IF operation == "+" DO
        SAY "Result: " {num1 + num2} end
    ELSE
        IF operation == "-" DO
            SAY "Result: " {num1 - num2} end
        ELSE
            IF operation == "*" DO
                SAY "Result: " {num1 * num2} end
            ELSE
                IF operation == "/" DO
                    IF num2 != 0 DO
                        SAY "Result: " {num1 / num2} end
                    ELSE
                        SAY "Error: Division by zero !" end
```

```
                        EXIT "Calculator terminated due to error"
                    ENDO
                ELSE
                    SAY "Unknown operation!" end
                ENDO
            ENDO
        ENDO
    ENDO

    LISTEN STR continue_calc "Continue? (yes/no): "
    IF continue_calc == "quit" DO
        EXIT "Calculator session ended by user"
    ENDO
ENDO

SAY "Thanks for using the calculator!"
```

3.7.2 Number Guessing Game

```
# Number guessing game with loops
CREATE INT secret_number
CREATE INT guess
CREATE INT attempts 0
CREATE BOOL game_won false

RANDOM INT 1 100 secret_number
SAY "=== Number Guessing Game ===" end
SAY "I'm thinking of a number between 1 and 100!" end
SAY "Type -1 to quit at any time." end

# Game loop
WHILE game_won == false DO
    LISTEN INT guess "Enter your guess: "

```

```
16      IF guess == -1 DO
17          EXIT "Thanks for playing! The number was "
    secret_number
18      ENDO
19
20      INCREMENT attempts
21
22      IF guess == secret_number DO
23          CHANGE game_won true
24          SAY "Congratulations! You guessed it in "
    attempts " attempt(s)!" end
25      ELSE
26          IF guess < secret_number DO
27              SAY "Too low! Try again." end
28          ELSE
29              SAY "Too high! Try again." end
30          ENDO
31      ENDO
32  ENDO
33
34  SAY "Thanks for playing!"
```

3.8 Best Practices

1. **Use descriptive variable names:** Choose names that clearly indicate the variable's purpose
2. **Comment your code:** Explain complex logic and important sections
3. **Initialize variables:** Always give variables initial values when possible
4. **Handle edge cases:** Check for division by zero, invalid input, etc.
5. **Use constants for fixed values:** CREATE CONST for values that shouldn't change
6. **Organize your code:** Group related operations together
7. **Test thoroughly:** Try different inputs to ensure your program works correctly
8. **Use EXIT strategically:** Provide clean program termination with informative messages
9. **Handle errors gracefully:** Use EXIT to prevent crashes from invalid operations

4 Quick Reference

4.1 Command Summary

- `SAY` - Display output
- `CREATE` - Declare variables
- `LISTEN` - Get user input
- `CHANGE` - Modify variables
- `CONVERT` - Change variable types
- `IF/ELSE/ENDO` - Conditional execution
- `WHILE/ENDO` - Loop execution
- `RANDOM` - Generate random values
- `EXIT` - Terminate program
- `INCREMENT/DECREMENT` - Modify numeric values
- `UPPERCASE/LOWERCASE/REVERSE` - String operations
- `SWAP` - Exchange variable values
- `DEL` - Delete variables
- `RESET` - Reset to default values
- `ROUND` - Round floating-point numbers

4.2 Tips for Success

- Variables are case-insensitive in NOOBIE2
- Use {} for mathematical expressions within strings
- The @end variable represents a newline character
- Always test your programs with different inputs
- Start simple and gradually add complexity
- No @ symbol needed for variable access in most contexts
- Use EXIT to provide clean program termination
- Include helpful messages with EXIT for better user experience

5 Conclusion

NOOBIE2 provides a gentle introduction to programming concepts while offering enough power for meaningful projects. By mastering the Simple NOOBIE concepts first, then progressing to Advanced NOOBIE features, students can build a solid foundation in programming logic and problem-solving.

The language's natural syntax and comprehensive error handling make it an ideal choice for educational environments, coding bootcamps, and self-directed learning. As students become comfortable with NOOBIE2, they'll find the transition to more complex programming languages much smoother.

Remember: every expert programmer started as a beginner. NOOBIE2 is designed to make that journey as enjoyable and productive as possible. Happy coding!