BRO-CODE

Programming Language

The Ultimate Vibe Check Language

Version 1.1

Generated: January 10, 2026

Created by Prashith

For CSI Code Wars

Table of Contents

1. Introduction

Bro Code is a high-level, dynamically typed esoteric programming language designed for the ultimate vibe check. It combines the block structure of C++ (using curly braces {}) with the simplicity and dynamic typing of Python.

The language prioritizes readability and 'vibes', making it both fun to learn and a unique challenge for programmers looking to adapt their coding skills.

Key Features

- File Extension: .homie
- Dynamic typing - no explicit type declarations
- C-style block structure with curly braces {}
- Python-like simplicity
- First-class functions with default parameters
- Error handling with shoot/fumble (try/catch)
- Built-in array type called 'Squad'
- No external dependencies - Python 3.8+ only

2. Installation

Via pip (Recommended)

```
pip install brocode-lang
```

Manual Installation

```
# Clone the repository
git clone https://github.com/Prashithshetty/Bro-CODE.git
cd Bro-CODE

# Install in development mode
pip install -e .

# Or run directly without installing
python -m brocode examples/hello.homie
```

Requirements

Python 3.8 or higher. No external dependencies required!

3. Quick Start

Running a Program

```
# Run a .homie file
brocode examples/hello.homie

# Or use the Python module directly
python -m brocode examples/hello.homie

# Or use the wrapper script
chmod +x bro
./bro examples/hello.homie
```

Hello World Example

Create a file called hello.homie:

```
// Hello World in Bro Code
// Expected output: Hello World! Welcome to Bro Code!

sup {
    spit("Hello World!");
    spit("Welcome to Bro Code!");
    peace;
}
```

Run it with: brocode hello.homie

4. Program Structure

Every Bro Code program must be contained within a main execution block named 'sup'. The program ends with the 'peace' statement, which acts as the exit command.

Basic Structure

```
sup {
    // Code logic goes here

    peace;  // Exit program
}
```

Comments

Bro Code supports both single-line and multi-line comments:

```
// This is a single-line comment

/* This is a
   multi-line comment */
```

5. Variables and Data Types

Bro Code uses dynamic typing. Variables are declared using 'var' (mutable) or 'const' (immutable). Explicit type declaration is not required.

Declaration

```
var name = "Bro";          // String (mutable)
var age = 21;              // Integer
var price = 4.20;          // Float
var isAwesome = no_cap;    // Boolean (true)
var isBoring = cap;        // Boolean (false)
var nothing = ghost;       // Null
const PI = 3.14;           // Constant (immutable)
```

Supported Data Types

Type	Example	Description
Integer	42, -7, 0	Whole numbers
Float	3.14, 4.20, -0.5	Decimal numbers
String	"Hello", "Bro"	Text (double quotes only)
Boolean	no_cap / cap	True / False
Null	ghost	Represents empty/null value
Squad	[1, 2, 3]	Ordered list (array)

Variables Example (variables.homie)

```
sup {
    // Integer
    var age = 21;
    spit("Age: " + str(age));

    // Float
    var price = 4.20;
    spit("Price: $" + str(price));

    // String
    var name = "Bro Coder";
    spit("Name: " + name);

    // Boolean
    var isAwesome = no_cap;
    spit("Is awesome: " + str(isAwesome));

    // Null
    var nothing = ghost;
    spit("Nothing: " + str(nothing));

    // Constants
    const PI = 3.14159;
    spit("PI: " + str(PI));

    // Arithmetic operations
    var a = 10;
    var b = 3;
    spit("10 + 3 = " + str(a + b));
    spit("10 - 3 = " + str(a - b));
    spit("10 * 3 = " + str(a * b));
    spit("10 / 3 = " + str(a / b));
    spit("10 % 3 = " + str(a % b));

    peace;
}
```

6. Input/Output (I/O)

Output - spit()

Use the spit() function to print data to the console:

```
spit("Hello!");                        // Print string
spit(x);                               // Print variable
spit("Score: " + str(score));          // Concatenate and print
```

Input - listen()

Use listen() to receive input from the user. It always returns a string:

```
var name = listen("Enter your name: ");
var age = int(listen("Enter your age: "));   // Convert to integer
```

Interactive Input Example (input.homie)

```
gig greetUser() {
    var name = listen("What's your name, bro? ");
    spit("Yo " + name + "! Welcome to Bro Code!");
}

gig guessNumber() {
    var secret = 7;
    var guess = int(listen("Guess a number (1-10): "));

    fr (guess is_vibe secret) {
        spit("No cap! You got it!");
    }
    meh (guess < secret) {
        spit("Too low, bro!");
    }
    nah {
        spit("Too high, bro!");
    }
}

sup {
    greetUser();
    guessNumber();
    peace;
}
```

7. Control Flow (Conditionals)

Decision making uses: fr (if), meh (else if), nah (else)

Syntax

Bro Code	Equivalent
fr (condition) { }	if (condition) { }
meh (condition) { }	else if (condition) { }
nah { }	else { }

Conditionals Example (conditionals.homie)

```
sup {
    var energy = 50;

    spit("Energy level: " + str(energy));

    fr (energy > 80) {
        spit("Hyper mode activated!");
    }
    meh (energy > 50) {
        spit("Feeling good!");
    }
    meh (energy > 20) {
        spit("Chilling...");
    }
    nah {
        spit("Need coffee ASAP!");
    }

    // Using is_vibe (equality check)
    var score = 100;
    fr (score is_vibe 100) {
        spit("Perfect score!");
    }

    // Boolean conditionals
    var hasPermission = no_cap;
    var isAdmin = cap;

    fr (hasPermission && !isAdmin) {
        spit("Regular user with access");
    }

    fr (hasPermission || isAdmin) {
        spit("Access granted somehow!");
    }

    peace;
}
```

8. Loops

While Loop (hold_up)

Executes the block while the condition is no_cap (True):

```
var count = 0;
hold_up (count < 5) {
    spit("Count: " + str(count));
    count = count + 1;
}
```

For Loop (cycle)

Iterates over a range or array (squad):

```
// Range iteration (0 to 4)
cycle (var i in range(0, 5)) {
    spit("i = " + str(i));
}

// With step parameter (0, 2, 4, 6, 8)
cycle (var i in range(0, 10, 2)) {
    spit(i);
}

// Iterating over a squad
var squad = ["Alice", "Bob", "Charlie"];
cycle (var name in squad) {
    spit("Hello, " + name + "!");
}
```

Loop Control (dip, skip)

dip = break (exit loop), skip = continue (next iteration)

```
// break_continue.homie example
sup {
    spit("=== Testing dip (break) ===");

    // Test break in while loop
    var count = 0;
    hold_up (no_cap) {
        count++;
        fr (count is_vibe 5) {
            spit("Breaking at count = " + str(count));
            dip;
        }
    }
    spit("After while loop, count = " + str(count));

    spit("=== Testing skip (continue) ===");

    // Test continue - skip even numbers
    spit("Odd numbers from 0 to 10:");
    cycle (var i in range(0, 10)) {
        fr (i % 2 is_vibe 0) {
            skip;
        }
        spit(i);
    }

    peace;
}
```

Loops Example (loops.homie)

```
sup {
    // While loop with hold_up
    spit("=== While Loop (hold_up) ===");
    var count = 0;
    hold_up (count < 5) {
        spit("Count: " + str(count));
        count = count + 1;
    }

    // For loop with cycle and range
    spit("=== For Loop (cycle) ===");
    cycle (var i in range(0, 5)) {
        spit("i = " + str(i));
    }

    // Iterating over an array (squad)
    spit("=== Iterating Squad ===");
    var squad = ["Alice", "Bob", "Charlie"];
    cycle (var name in squad) {
        spit("Hello, " + name + "!");
    }

    // Nested loops
    spit("=== Multiplication Table (3x3) ===");
    cycle (var x in range(1, 4)) {
        cycle (var y in range(1, 4)) {
            spit(str(x) + " x " + str(y) + " = " + str(x * y));
        }
    }

    peace;
}
```

9. Functions

Functions are first-class citizens in Bro Code, defined with 'gig'. Values are returned using 'pay'. Functions support default parameters.

Basic Function Syntax

```
// Simple function
gig greet(name) {
    spit("Hello, " + name + "!");
}

// Function with return value
gig add(a, b) {
    pay a + b;
}

// Function with default parameters
gig greet(name = "bro", greeting = "Hey") {
    spit(greeting + ", " + name + "!");
}
```

Functions Example (functions.homie)

```
// Simple function
gig greet(name) {
    spit("Hello, " + name + "!");
}

// Function with return value
gig add(a, b) {
    pay a + b;
}

// Function with multiple parameters
gig calculateArea(width, height) {
    var area = width * height;
    pay area;
}

// Recursive function
gig factorial(n) {
    fr (n <= 1) {
        pay 1;
    }
    pay n * factorial(n - 1);
}

// Function using conditionals
gig getGrade(score) {
    fr (score >= 90) {
        pay "A";
    }
    meh (score >= 80) {
        pay "B";
    }
    meh (score >= 70) {
        pay "C";
    }
    meh (score >= 60) {
        pay "D";
    }
    nah {
        pay "F";
    }
}

sup {
    greet("Bro");

    var sum = add(5, 3);
    spit("5 + 3 = " + str(sum));

    var area = calculateArea(10, 5);
    spit("Area of 10x5: " + str(area));

    spit("5! = " + str(factorial(5)));

    var myScore = 85;
    spit("Score " + str(myScore) + " = Grade " + getGrade(myScore));

    peace;
}
```

Default Parameters Example (default_params.homie)

```
gig greet(name = "bro", greeting = "Hey") {
    spit(greeting + ", " + name + "!");
}

gig add(a, b = 10, c = 100) {
    pay a + b + c;
}

gig power_of_two(n = 8) {
    pay 2 ** n;
}

sup {
    spit("=== Testing Default Parameters ===");

    greet();                    // "Hey, bro!"
    greet("homie");             // "Hey, homie!"
    greet("fam", "Yo");         // "Yo, fam!"

    spit("add(5) = " + str(add(5)));           // 115
    spit("add(5, 20) = " + str(add(5, 20)));   // 125
    spit("add(5, 20, 50) = " + str(add(5, 20, 50))); // 75

    spit("power_of_two() = " + str(power_of_two()));   // 256
    spit("power_of_two(4) = " + str(power_of_two(4))); // 16

    peace;
}
```

10. Operators

Arithmetic Operators

Operator	Description	Example
+	Addition	5 + 3 = 8
-	Subtraction	5 - 3 = 2
*	Multiplication	5 * 3 = 15
/	Division	7 / 2 = 3.5
%	Modulo	7 % 2 = 1
**	Power (exponentiation)	2 ** 10 = 1024
//	Integer Division	7 // 2 = 3

Comparison Operators

Operator	Description
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
==	Equal to
!=	Not equal to
is_vibe	Equal to (alias for ==)

Logical Operators

Operator	Description
&&	Logical AND
||	Logical OR
!	Logical NOT

Assignment Operators

Operator	Description	Equivalent
=	Assignment	x = 5
+=	Add and assign	x = x + 5
-=	Subtract and assign	x = x - 5
*=	Multiply and assign	x = x * 5

/=	Divide and assign	x = x / 5
%=	Modulo and assign	x = x % 5
++	Increment by 1	x = x + 1
--	Decrement by 1	x = x - 1

Power Operations Example (power_ops.homie)

```
sup {
    spit("=== Testing Power Operator (**) ===");

    var result = 2 ** 10;
    spit("2 ** 10 = " + str(result));  // 1024

    result = 3 ** 4;
    spit("3 ** 4 = " + str(result));   // 81

    result = 10 ** 0;
    spit("10 ** 0 = " + str(result));  // 1

    // Right-associative: 2 ** 3 ** 2 = 2 ** 9 = 512
    result = 2 ** 3 ** 2;
    spit("2 ** 3 ** 2 = " + str(result));  // 512

    spit("=== Testing Integer Division (//) ===");

    result = 7 // 2;
    spit("7 // 2 = " + str(result));   // 3

    result = 10 // 3;
    spit("10 // 3 = " + str(result));  // 3

    peace;
}
```

Compound Ops Example (compound_ops.homie)

```
sup {
    spit("=== Testing Compound Assignment ===");

    var x = 100;
    spit("Initial x = " + str(x));

    x += 50;
    spit("After x += 50: " + str(x));  // 150

    x -= 30;
    spit("After x -= 30: " + str(x));  // 120

    x *= 2;
    spit("After x *= 2: " + str(x));   // 240

    x /= 4;
    spit("After x /= 4: " + str(x));   // 60

    x %= 7;
    spit("After x %= 7: " + str(x));   // 4

    spit("=== Testing Increment/Decrement ===");

    var count = 0;
    spit("Initial count = " + str(count));

    count++;
    spit("After count++: " + str(count));  // 1

    count++;
    count++;
    spit("After two more count++: " + str(count));  // 3

    count--;
    spit("After count--: " + str(count));  // 2

    peace;
}
```

11. Data Structures (Squads/Arrays)

Lists in Bro Code are called 'Squads'. They are mutable, ordered collections that can hold any type of value including mixed types.

Squad Basics

```
var squad = [1, 2, 3, 4, 5];    // Create a squad
spit(squad[0]);                  // Access: 1
squad[0] = 100;                  // Modify element
spit(len(squad));                // Length: 5
```

Instance Methods

Called on the squad variable itself:

Method	Description	Example
push(value)	Add to end	squad.push(4)
pop()	Remove & return last	var x = squad.pop()

Global Array Functions

Function	Description
len(arr)	Returns number of elements
sort(arr)	Returns a new sorted squad
reverse(arr)	Returns a new reversed squad
index_of(arr, val)	Returns index or -1 if not found
includes(arr, val)	Returns boolean if value exists
slice(arr, start, end)	Returns portion of squad
concat(arr1, arr2)	Merges two squads into new one
range(start, end, step)	Generate sequence of integers

Squads Example (squads.homie)

```
sup {
    var squad = [1, 2, 3, 4, 5];
    spit("Original squad: ");
    cycle (var num in squad) {
        spit(num);
    }

    // Access by index
    spit("First element: " + str(squad[0]));
    spit("Last element: " + str(squad[4]));

    // Get length
    spit("Squad size: " + str(len(squad)));

    // Push (add to end)
    squad.push(6);
    spit("After push(6), size: " + str(len(squad)));

    // Pop (remove from end)
    var popped = squad.pop();
    spit("Popped: " + str(popped));

    // Modify by index
    squad[0] = 100;
    spit("After squad[0] = 100: " + str(squad[0]));

    // Mixed type squad
    var mixed = ["hello", 42, no_cap, 3.14];
    spit("Mixed squad types:");
    cycle (var item in mixed) {
        spit("  " + str(item) + " (type: " + type(item) + ")");
    }

    peace;
}
```

Array Functions Example (array_funcs.homie)

```
sup {
    var squad = [3, 1, 4, 1, 5, 9, 2, 6];
    spit("Original squad: [3, 1, 4, 1, 5, 9, 2, 6]");

    // sort (returns new array)
    var sorted_squad = sort(squad);
    spit("sort(squad) = " + str(sorted_squad));

    // reverse (returns new array)
    var reversed = reverse(squad);
    spit("reverse(squad) = " + str(reversed));

    // slice
    var sliced = slice(squad, 2, 5);
    spit("slice(squad, 2, 5) = " + str(sliced));  // [4, 1, 5]

    // index_of
    spit("index_of(squad, 5) = " + str(index_of(squad, 5)));   // 4
    spit("index_of(squad, 99) = " + str(index_of(squad, 99))); // -1

    // includes
    spit("includes(squad, 9) = " + str(includes(squad, 9)));   // no_cap
    spit("includes(squad, 99) = " + str(includes(squad, 99))); // cap

    // concat
    var squad2 = [7, 8];
    var combined = concat(squad, squad2);
    spit("concat(squad, [7, 8]) = " + str(combined));

    // range with step
    var evens = range(0, 10, 2);
    spit("range(0, 10, 2) = " + str(evens));  // [0, 2, 4, 6, 8]

    peace;
}
```

12. Error Handling

Handle runtime errors gracefully using 'shoot' (try) and 'fumble' (catch). The error variable in fumble contains the error message.

Syntax

```
shoot {
    // Code that might cause an error
    var result = 10 / 0;
}
fumble (error) {
    spit("Error: " + error);
}
```

Error Handling Example (error_handling.homie)

```
sup {
    spit("=== Error Handling Demo ===");

    // Catching a division error
    shoot {
        var result = 10 / 0;
        spit("Result: " + str(result));
    }
    fumble (error) {
        spit("Caught error: " + error);
    }

    spit("Program continues after error handling!");

    // Successful try block
    shoot {
        var x = 10;
        var y = 2;
        spit("10 / 2 = " + str(x / y));
    }
    fumble (error) {
        spit("This won't print");
    }

    peace;
}
```

13. Built-in Functions

All built-in functions are available globally without any imports.

I/O Functions

Function	Description	Return Type
spit(value)	Print value to stdout	ghost
listen(prompt)	Read input from user	string

Type Conversion Functions

Function	Description	Return Type
int(value)	Convert to integer	integer
float(value)	Convert to float	float
str(value)	Convert to string	string
type(value)	Get type name	string

Math Functions

Function	Description	Return Type
abs(x)	Absolute value	number
min(a, b)	Minimum of two values	number
max(a, b)	Maximum of two values	number
pow(base, exp)	Exponentiation	number
sqrt(x)	Square root	float
floor(x)	Round down	integer
ceil(x)	Round up	integer
round(x, digits)	Round to precision	number

Math Functions Example (math_funcs.homie)

```
sup {
    spit("=== Testing Math Functions ===");

    // abs
    spit("abs(-42) = " + str(abs(-42)));      // 42
    spit("abs(42) = " + str(abs(42)));        // 42

    // max/min
    spit("max(10, 20) = " + str(max(10, 20)));  // 20
    spit("min(10, 20) = " + str(min(10, 20)));  // 10

    // pow
    spit("pow(2, 8) = " + str(pow(2, 8)));     // 256

    // sqrt
    spit("sqrt(16) = " + str(sqrt(16)));       // 4.0
    spit("sqrt(2) = " + str(sqrt(2)));         // 1.414...

    // floor/ceil
    spit("floor(3.7) = " + str(floor(3.7)));   // 3
    spit("ceil(3.2) = " + str(ceil(3.2)));     // 4
    spit("floor(-3.2) = " + str(floor(-3.2))); // -4
    spit("ceil(-3.7) = " + str(ceil(-3.7)));   // -3

    // round
    spit("round(3.5) = " + str(round(3.5)));              // 4
    spit("round(3.14159, 2) = " + str(round(3.14159, 2))); // 3.14

    peace;
}
```

String Functions

Function	Description
upper(s)	Convert to uppercase
lower(s)	Convert to lowercase
trim(s)	Remove whitespace from ends
split(s, delim)	Split string by delimiter
join(arr, delim)	Join array with delimiter
replace(s, old, new)	Replace all occurrences
contains(s, sub)	Check if contains substring
starts_with(s, prefix)	Check if starts with prefix
ends_with(s, suffix)	Check if ends with suffix
char_at(s, index)	Get character at index
substring(s, start, end)	Get substring

String Functions Example (string_funcs.homie)

```
sup {
    spit("=== Testing String Functions ===");

    // upper/lower
    spit("upper('hello') = " + upper("hello"));     // HELLO
    spit("lower('WORLD') = " + lower("WORLD"));     // world

    // trim
    spit("trim('  hello  ') = '" + trim("  hello  ") + "'");  // 'hello'

    // split/join
    var words = split("apple,banana,cherry", ",");
    spit("split result:");
    cycle (var word in words) {
        spit("  - " + word);
    }

    var joined = join(words, " | ");
    spit("join with ' | ': " + joined);  // apple | banana | cherry

    // replace
    var msg = "Hello World";
    spit("replace 'World' with 'Bro': " + replace(msg, "World", "Bro"));

    // contains
    spit("contains('Hello', 'ell') = " + str(contains("Hello", "ell")));  // no_cap

    // starts_with/ends_with
    spit("starts_with('Hello', 'He') = " + str(starts_with("Hello", "He"))); // no...
    spit("ends_with('Hello', 'lo') = " + str(ends_with("Hello", "lo")));     // no...

    // char_at
    spit("char_at('Hello', 1) = " + char_at("Hello", 1));  // e

    // substring
    spit("substring('Hello World', 0, 5) = " + substring("Hello World", 0, 5)); //...

    peace;
}
```

Utility Functions

Function	Description	Return Type
random()	Random float 0-1	float
random_int(min, max)	Random int (inclusive)	integer
time()	Current Unix timestamp	float

Utility Functions Example (utility_funcs.homie)

```
sup {
    spit("=== Testing Utility Functions ===");

    // random (float 0-1)
    spit("random() = " + str(random()));
    spit("random() = " + str(random()));
    spit("random() = " + str(random()));

    spit("=== Testing random_int ===");

    // random_int (inclusive)
    spit("random_int(1, 10) = " + str(random_int(1, 10)));
    spit("random_int(1, 10) = " + str(random_int(1, 10)));
    spit("random_int(1, 10) = " + str(random_int(1, 10)));

    spit("=== Testing time ===");

    // time (current timestamp)
    var start = time();
    spit("Current timestamp: " + str(start));

    // Small delay simulation
    var sum = 0;
    cycle (var i in range(0, 10000)) {
        sum += i;
    }

    var end = time();
    spit("After loop timestamp: " + str(end));
    spit("Elapsed time: " + str(end - start) + " seconds");

    peace;
}
```

14. Example Programs

The following examples demonstrate practical use of Bro Code features. All examples are available in the examples/ directory of the repository.

FizzBuzz (fizzbuzz.homie)

```
// FizzBuzz in Bro Code
// Classic programming challenge

gig fizzBuzz(n) {
    cycle (var i in range(1, n + 1)) {
        fr (i % 15 is_vibe 0) {
            spit("FizzBuzz");
        }
        meh (i % 3 is_vibe 0) {
            spit("Fizz");
        }
        meh (i % 5 is_vibe 0) {
            spit("Buzz");
        }
        nah {
            spit(i);
        }
    }
}

sup {
    spit("=== FizzBuzz (1-20) ===");
    fizzBuzz(20);
    peace;
}
```

Fibonacci Sequence (fibonacci.homie)

```
// Fibonacci Sequence in Bro Code

gig fibonacci(n) {
    fr (n <= 0) {
        pay 0;
    }
    fr (n is_vibe 1) {
        pay 1;
    }

    var a = 0;
    var b = 1;
    var result = 0;
    var i = 2;

    hold_up (i <= n) {
        result = a + b;
        a = b;
        b = result;
        i = i + 1;
    }

    pay result;
}

sup {
    spit("=== Fibonacci Sequence ===");

    cycle (var i in range(0, 15)) {
        spit("fib(" + str(i) + ") = " + str(fibonacci(i)));
    }

    peace;
}
```

Negative Numbers (negative_numbers.homie)

```
sup {
    spit("=== Testing Negative Numbers ===");

    var n1 = -5;
    spit("n1 = " + str(n1));  // -5

    var n2 = 10 + -3;
    spit("10 + -3 = " + str(n2));  // 7

    var n3 = -(-5);
    spit("-(-5) = " + str(n3));  // 5

    var n4 = 5 * -2;
    spit("5 * -2 = " + str(n4));  // -10

    var n5 = -10 / 2;
    spit("-10 / 2 = " + str(n5));  // -5

    var n6 = -5.5;
    spit("n6 = " + str(n6));  // -5.5

    // Test negative in function args
    spit("abs(-50) = " + str(abs(-50)));

    peace;
}
```

Appendix A: Quick Reference Cheat Sheet

Feature	Bro Code	Python
Main block	sup { ... }	if __name__ == "__main__":
Exit	peace;	exit()
Print	spit(x)	print(x)
Input	listen(prompt)	input(prompt)
Variable	var x = 1;	x = 1
Constant	const X = 1;	X = 1 (convention)
True/False	no_cap / cap	True / False
Null	ghost	None
If	fr (cond) { }	if cond:
Else if	meh (cond) { }	elif cond:
Else	nah { }	else:
While	hold_up (cond) { }	while cond:
For	cycle (var i in range(0, n)) { }	for i in range(n):
Break	dip;	break
Continue	skip;	continue
Function	gig name(x = 10) { }	def name(x=10):
Return	pay x;	return x
Equality	is_vibe or ==	==
Try	shoot { }	try:
Catch	fumble (e) { }	except Exception as e:
Power	2 ** 10	2 ** 10
Int Division	7 // 2	7 // 2
Increment	i++;	i += 1

Appendix B: Reserved Keywords

The following keywords are reserved and cannot be used as variable or function names:

```
sup, peace, var, const, fr, meh, nah, hold_up, cycle, in,
gig, pay, shoot, fumble, no_cap, cap, ghost, is_vibe, dip, skip
```

Appendix C: CLI Options

```
brocode <file.homie>           # Run a file
brocode --version              # Show version
brocode --tokens file.homie    # Debug: show tokens
brocode --ast file.homie       # Debug: show AST
```

Appendix D: Project Structure

```
Bro-CODE/
|-- brocode/
|   |-- __init__.py      # Package info
|   |-- __main__.py      # CLI entry point
|   |-- tokens.py        # Token definitions
|   |-- lexer.py         # Tokenizer
|   |-- parser.py        # AST builder
|   |-- ast_nodes.py     # AST node classes
|   |-- interpreter.py   # Executor
|   |-- builtins.py      # Built-in functions
|   |-- errors.py        # Exception classes
|-- examples/            # Sample programs (19 files)
|-- docs/                # Documentation
|-- bro                  # CLI wrapper script
|-- README.md
```

Appendix E: Example Files

All example files in the examples/ directory:

- hello.homie - Hello World
- variables.homie - Data types and operations
- conditionals.homie - fr/meh/nah (if/else)
- loops.homie - hold_up and cycle loops
- functions.homie - gig/pay with recursion
- squads.homie - Array operations
- fibonacci.homie - Fibonacci sequence
- fizzbuzz.homie - Classic FizzBuzz
- error_handling.homie - shoot/fumble (try/catch)
- input.homie - Interactive input
- break_continue.homie - dip/skip (break/continue)

- compound_ops.homie - +=, -=, ++, -- operators
- default_params.homie - Function default parameters
- power_ops.homie - ** and // operators
- math_funcs.homie - Math built-in functions
- string_funcs.homie - String built-in functions
- array_funcs.homie - Array built-in functions
- utility_funcs.homie - Utility functions
- negative_numbers.homie - Negative number handling