Khamseena Programming Language

Core Concept

Language Name: Khamseena
Tagline: "Code it simple, brew it smooth!"

Theme:
Khamseena is a beginner-friendly programming language inspired by the Egyptian "khamseena tea" - small, simple, and effective. The language uses everyday English words related to making tea and simple actions to make programming intuitive and fun for kids and beginners.

Keywords

Function	Khamseena Keyword	Standard Equivalent
Program start	brew	main / start
Function definition	recipe	function / def
Integer variable	count	int
Float/Double variable	measure	float / double
String variable	label	string
Boolean variable	switch	boolean
Output/Print	serve	print / cout
Input/Read	pour	input / cin
If condition	taste	if
Else	refill	else
While loop	stir	while
For loop	mix	for
Return value	deliver	return
Include library	fetch	include / import
Single line comment	# note:	// comment
True value	hot	true
False value	cold	false

Data Types

Type	Meaning	Example Syntax
count	Integer number	count score = 10;
measure	Decimal number	measure price = 25.5;
label	Text string	label name = "Ahmed";
switch	Boolean value	switch ready = hot;

Code Examples

🎯 Example 1: Hello World

```
fetch basics

recipe brew() {
    serve "Welcome to Khamseena!";
    deliver 0;
}
```

📝 Example 2: Variables and Input/Output

```
fetch basics

recipe brew() {
    label name;
    count age;

    serve "What's your name? ";
    pour name;

    serve "How old are you? ";
    pour age;

    serve name + " is " + age + " years old";

    deliver 0;
}
```

🔍 Example 3: Conditions (taste and refill)

```
fetch basics

recipe brew() {
    count grade = 85;

    taste (grade >= 50) {
        serve "Congratulations! You passed";
    }
    refill {
        serve "Try again next time";
    }

    deliver 0;
}
```

🔁 Example 4: While Loop (stir)

```
fetch basics

recipe brew() {
    count counter = 0;

    stir (counter < 5) {
        serve "Count: " + counter;
        counter = counter + 1;
    }

    deliver 0;
}
```

🔄 Example 5: For Loop (mix)

```
fetch basics

recipe brew() {
    mix (count i = 1; i <= 3; i = i + 1) {
        serve "Round number " + i;
    }

    deliver 0;
}
```

⚙️ Example 6: Custom Function

```
fetch basics

recipe add(count a, count b) {
    count result = a + b;
    deliver result;
}

recipe brew() {
    count x = 10;
    count y = 20;
    count sum = add(x, y);

    serve "Sum = " + sum;

    deliver 0;
}
```

🎲 Example 7: Complete Program (Guessing Game)

```
fetch basics

recipe brew() {
    count secret = 7;
    count guess;
    switch playing = hot;

    serve "** Guessing Game **";
    serve "Guess a number from 1 to 10";

    stir (playing == hot) {
        serve "Enter a number: ";
        pour guess;

        taste (guess == secret) {
            serve "Correct! You won!";
            playing = cold;
        }
        refill {
            taste (guess < secret) {
                serve "Too low! Try again";
            }
            refill {
                serve "Too high! Try again";
            }
        }
    }
    serve "Game over. Thanks for playing!";
    deliver 0;
}
```

Report Notes

🎯 Language Purpose:

Khamseena is an educational programming language designed for children and beginners. It uses tea-making metaphors and simple everyday English words to make programming approachable and non-intimidating.

🧩 Design Goals:

1. **Simplicity:** Use familiar words from daily life
2. **Clarity:** Each command has an obvious, memorable meaning
3. **Compatibility:** Structure similar to C/Python for easy transition later
4. **Fun:** Make programming enjoyable and stress-free

💡 Why "Khamseena"?

The name comes from the Egyptian "khamseena tea" - a small, simple cup of tea that's quick and effective. The language is designed to be just as simple and straightforward!

🔧 Additional Features:

- **Easy Learning:** Keywords based on tea-making and simple actions
- **Clear Metaphors:** "brew" starts programs, "serve" outputs results
- **Memorable:** Each keyword connects to a real-world action
- **Consistent:** All keywords follow the same simple pattern

---

Syntax Rules

1. Line Endings:

- Every statement ends with a semicolon ;

2. Brackets:

- Round brackets () for functions and conditions
- Curly brackets {} for code blocks

3. Comments:

```
# note: This is a single-line comment
```

4. Strings:

- Text is enclosed in double quotes ""

5. Arithmetic Operators:

- Addition: +
- Subtraction: -
- Multiplication: *

- Division: /
- Modulo: %

6. Comparison Operators:

- Equal to: ==
- Not equal: !=
- Greater than: >
- Less than: <
- Greater or equal: >=
- Less or equal: <=

7. Logical Operators:

- AND: &&
- OR: ||
- NOT: !

Comprehensive Example: Grade Calculator

```
fetch basics

recipe calculate_average(count sub1, count sub2, count sub3) {
    measure total = sub1 + sub2 + sub3;
    measure avg = total / 3;
    deliver avg;
}

recipe brew() {
    label student_name;
    count score1, score2, score3;

    serve "==== Grade Calculator ====";
    serve "Student name: ";
    pour student_name;

    serve "First subject score: ";
    pour score1;

    serve "Second subject score: ";
    pour score2;

    serve "Third subject score: ";
    pour score3;

    measure final_avg = calculate_average(score1, score2, score3);

    serve "Student: " + student_name;
    serve "Average: " + final_avg;

    taste (final_avg >= 90) {
        serve "Grade: Excellent";
    }
    refill taste (final_avg >= 75) {
        serve "Grade: Very Good";
    }
    refill taste (final_avg >= 60) {
        serve "Grade: Good";
    }
    refill {
        serve "Grade: Pass";
    }

    deliver 0;
}
```

More Examples

📊 Example: Simple Calculator

```
fetch basics

recipe calculate(count num1, count num2, label operation) {
    count result = 0;

    taste (operation == "add") {
        result = num1 + num2;
    }
    refill taste (operation == "subtract") {
        result = num1 - num2;
    }
    refill taste (operation == "multiply") {
        result = num1 * num2;
    }
    refill taste (operation == "divide") {
        result = num1 / num2;
    }

    deliver result;
}

recipe brew() {
    count a, b;
    label op;

    serve "Enter first number: ";
    pour a;

    serve "Enter second number: ";
    pour b;

    serve "Operation (add/subtract/multiply/divide): ";
    pour op;

    count answer = calculate(a, b, op);
    serve "Result: " + answer;

    deliver 0;
}
```

🔢 Example: Multiplication Table

```
fetch basics

recipe brew() {
    count number;
    serve "Enter a number for multiplication table: ";
    pour number;

    serve "Multiplication Table for " + number;

    mix (count i = 1; i <= 10; i = i + 1) {
        count result = number * i;
        serve number + " x " + i + " = " + result;
    }

    deliver 0;
}
```

🎯 Example: Even or Odd Checker

```
fetch basics

recipe check_even_odd(count num) {
    taste (num % 2 == 0) {
        serve num + " is even";
    }
    refill {
        serve num + " is odd";
    }
}

recipe brew() {
    count number;

    serve "Enter a number: ";
    pour number;

    check_even_odd(number);

    deliver 0;
}
```

💰 Example: Shopping Cart Total

```
fetch basics

recipe brew() {
    count items = 0;
    measure total = 0.0;
    measure price;
    switch shopping = hot;
    label answer;

    serve "==== Shopping Cart ====";

    stir (shopping == hot) {
        serve "Enter item price: ";
        pour price;

        total = total + price;
        items = items + 1;

        serve "Add another item? (yes/no): ";
        pour answer;

        taste (answer == "no") {
            shopping = cold;
        }
    }

    serve "Total items: " + items;
    serve "Total price: " + total;

    deliver 0;
}
```

🌟 Example: Countdown Timer

```
fetch basics

recipe brew() {
    count seconds;

    serve "Enter countdown seconds: ";
    pour seconds;

    serve "Starting countdown...";

    stir (seconds > 0) {
        serve seconds + " seconds remaining";
        seconds = seconds - 1;
    }

    serve "Time's up!";

    deliver 0;
}
```

Language Features Summary

✨ Key Advantages:

1. **Memorable Keywords:** Tea-making metaphors make commands easy to remember
    - `brew` = start cooking/programming
    - `serve` = present/output result
    - `pour` = add/input ingredient
    - `taste` = check/test condition
    - `stir` = keep mixing/looping
    - `mix` = combine/iterate
2. **Beginner-Friendly:** No complex syntax or confusing symbols
3. **Natural Flow:** Reading code feels like following a recipe
4. **Clear Structure:** Same organization as mainstream languages (C/C++/Java)
5. **Educational:** Teaches fundamental programming concepts with familiar vocabulary

---