BOTMS Bridge™ (Draft Architecture)

BOTMS Bridge™ is the core **translation layer** of BOTMS Genesis. It is responsible for converting human-friendly BOTMS code into executable Python code. This layer is essential to achieve the main goal: making bot creation easy and accessible, while keeping the underlying mechanics hidden from the user.

Key Responsibilities

1. **Parsing BOTMS Code:**
    - The Bridge receives BOTMS code written in natural language.
    - A parser (built using a tool such as Lark or ANTLR) tokenizes the code and constructs an **Abstract Syntax Tree (AST)**.
    - The AST represents the hierarchical structure of the code, including all commands, conditions, loops, and expressions.
2. **Translating to Python:**
    - The AST is traversed and each BOTMS construct is mapped to its equivalent Python construct.
    - For example:
        - `Ask "Enter your name"` becomes `input("Enter your name")`
        - `Reply hello` is converted to `print("hello")`
        - Functions and control structures are translated into Python's `def`, `if`, `for`, and other statements.
    - This translation ensures that complex BOTMS code is automatically converted into efficient and readable Python code.
3. **Executing the Translated Code:**
    - Once translated, the Python code is executed using Python's runtime environment.
    - The execution can be done by writing the code to a temporary file and running it, or dynamically via Python's `exec()` function.
    - Error handling and debugging information is also integrated at this stage, so that any issues during execution provide meaningful feedback to the user.

Process Flow Diagram

1. **Input Capture:**
   The user writes BOTMS code in a text editor using natural language commands.
2. **Lexical Analysis:**
   The parser scans the input and breaks it into tokens (keywords, variables, literals).
3. **Syntax Analysis (Parsing):**
   The tokens are organized into an AST representing the program structure.
4. **Semantic Analysis:**
   The AST is checked for correctness (e.g., variable definitions, data types, proper block structures).

5. **Code Generation (Translation):**
   The AST is converted into Python code by mapping each BOTMS command to its corresponding Python statement.
6. **Execution:**
   The generated Python code is executed and results are returned to the user.
7. **Feedback Loop:**
   Any errors detected during parsing, translation, or execution are reported back with user-friendly messages.

Syntax Clarifications: Inline vs. Multiline Blocks

To ensure clarity and prevent ambiguity in BOTMS code, a clear distinction is made between inline commands and multiline blocks. This allows users to write both simple one-liners and more complex, indented code blocks.

Inline Commands

- **Definition:**
  Inline commands are written on a single line and are used for simple tasks or short statements.
- **Usage:**
  They do not require any additional symbols (like colons) to indicate the end of the command. The end of the line is sufficient to signal the end of the inline command.

Example:
botms
CopyEdit
`Reply hello`

- This command outputs the word "hello" as a single line.

Multiline Blocks

- **Definition:**
  Multiline blocks are used for more complex commands that span several lines, such as multiple outputs or a series of actions that need to be executed sequentially.
- **Indicator:**
  A colon (`:`) at the end of the initiating command indicates the start of a multiline block.
- **Structure:**
  Once a multiline block is initiated, all subsequent indented lines are considered part of that block until the indent level decreases, which signals the end of the block.

Example:
botms

CopyEdit

```
Reply:
    hello
    hi there
```

This command outputs:

nginx

CopyEdit

```
hello
hi there
```

-

Key Differences

Aspect	Inline Commands	Multiline Blocks
Terminator	End-of-line (no colon)	Colon (`:`) at the end of the initiating command
Use Case	Single, simple outputs	Complex outputs, multi-line statements
Visual Structure	One-liner	Indented block structure

Example	`Reply hello`	`Reply:`
		`  hello`
		`  hi there`

Error Handling in Syntax

- **Missing Colon for Multiline Blocks:**
  If a multiline block is intended but the colon is omitted, BOTMS will throw a syntax error indicating an "Unclear Statement."
- **Unexpected Colon in Inline Commands:**
  Conversely, if a colon is used when it is not needed (i.e., when the command is meant to be inline), BOTMS will interpret it as the start of a multiline block, potentially causing a block error if no indented content follows.

By defining these clear rules and distinctions, BOTMS ensures that the code remains both human-friendly and structurally sound. This makes it easier for users to write and read BOTMS scripts while keeping the underlying parser robust and predictable.