⚡ Lightning: A High-Level Programming Language

Author: Shanil Aziz Malik (aka **Cyber Code**)**Type:** HLPL — High-Level Programming Language

📌 1. Introduction

Lightning is a human-readable, minimalist high-level programming language (HLPL) that emphasizes **clarity**, **expression**, and **speed**. It is built for those who want to code in a way that mirrors human thinking and poetic flow.

Created with simplicity in mind, Lightning allows users to write expressive, readable, and symbolic code, ideal for education, automation, and creative applications.

📚 2. History

Lightning was invented in **2025** by **Shanil Aziz Malik**, known online as **Cyber Code**. The motivation was to design a **clean**, **interpretable**, and **human-first** programming language.

Inspired by Python's readability and pseudocode's approachability, Lightning focuses on **removing barriers** for new learners and making code more expressive for creative developers.

🌍 3. Where and How It Is Used

Lightning is mainly used in:

- **Learning Environments**: Teaching coding fundamentals without technical overhead.
- **Creative Coding**: For developers who want their code to feel like poetry.
- **Script Automation**: Lightweight tools and task automators.
- **Micro Apps**: Small logic-driven tools (e.g. bots, responders, planners).
- **Game Scripting**: Dialogue trees and logic-based behaviors.

The language runs on a lightweight interpreter called **FlashEngine**, designed specifically for Lightning's symbolic syntax.

⚙️ 4. How to Make It

To build Lightning as a real HLPL:

A. Define Syntax

Use a grammar that supports:

- `import#` statements
- Variable assignment with `tro={}`
- Output with `print (=value=)`
- Input simulation with `put(value)`
- Conditionals using `if#var==value:`
- Return using `return!=[value]`

B. Write the Interpreter

Use Python or JavaScript to:

1. Tokenize the Lightning script
2. Parse tokens into commands
3. Execute them using custom runtime logic

You can simulate this with a simple stack-based interpreter.

C. Optional Extensions

- Syntax highlighting
- Visual editors
- Online REPL

🧪 5. Applications

Domain	Usage
Education	Teach variables, conditionals, output
Creative Coding	Code poems, logic art, storytelling
Scripting	Chatbots, task automators, simulations
Game Dev	AI trees, dialogue scripting
Prototyping	Outline logic before implementing in full

⚖️ 6. Comparing It With Other HLPLs

Feature	Lightning	Python	Lua	Scratch
Syntax Simplicity	✅✅✅✅✅	✅✅✅✅	✅✅✅	✅✅✅✅✅
Learning Curve	✅✅✅✅✅	✅✅✅✅	✅✅✅	✅✅✅✅✅
Human-Readability	✅✅✅✅✅	✅✅✅✅	✅✅✅	✅✅✅✅✅
Execution Speed	✅✅	✅✅✅✅	✅✅✅✅	✅✅
Creative Flex	✅✅✅✅✅	✅✅✅	✅✅✅	✅✅✅✅✅

✅ 7. Examples

🔹 A. Hello World + Identity

```
import# Light
      # ning
({                                      imp-
      bort.hi-there+*
      tro={i-am}
      tro={lightning-}
   put(nothing)
        return!=[greeting]
     print (=nothing=)

                                        emp-
 }
)
```

Output:

```
hi there
i am lightning
greetings
```

🔹 B. Mood Responder App (⚡ Simple App Example)

A small interactive script that responds based on the user’s mood.

```
import# Light
      # ning
({

   # user input
   tro={mood}
   put(mood)
   print (=you-feel=)
   print (=mood=)
   # decision logic

   if#mood==happy:

         print (=that's-awesome!=)

         return!=[smile]

   elif#mood==sad:

         print (=sending-good-vibes=)

         return!=[hug]

   else:

         print (=i-feel-you=)

         return!=[okay]


   # goodbye

   tro={thanks-for-using-mood-responder}

   print (=thanks-for-using-mood-responder=)


})
```

Sample Output 1:

```
>> (User types: happy)

you feel

happy

that's awesome!

smile

thanks for using mood responder
```

Sample Output 2:

```
>> (User types: sad)

you feel

sad

sending good vibes

hug

thanks for using mood responder
```

🔮 8. Lightning Language Features (Experimental Ideas)

Feature	Description
`spark{}`	Function blocks
`bolt()`	Random choice expressions
`@flash`	Async event simulation
`glow.theme=dark`	Theme/visual code mode (for IDEs or REPLs)
`loop!`	Repetitive action keyword

✍️ 9. Design Philosophy

"Let the code **speak like a person**, **behave like a machine**, and **feel like lightning**."— **Shanil Aziz Malik** (*Cyber Code*)

Lightning is more than a programming language. It's an **experiment in symbolic communication**, designed to make logic **beautiful**, **simple**, and **expressive**.

⚡ Lightning HLPL — Symbolic Language Extensions

🧿 Symbolic Language Features

Symbol	Name	Meaning / Use Case
`tro={}`	Thought Declaration	Used to declare or assign a value (like `var`)
`put()`	Expression Drop	Executes or "places" a variable into logic
`print(=x=)`	Echo Output	Outputs with expressive format
`return!=[]`	Energy Return	Returns a response value (emotion or result)
`+*`	Initiate Action	Symbol for triggering an action, function, or logic start
`imp-`	Logic Open Mark	Marks the start of logic region
`emp-`	Logic Close Mark	Marks the end of logic region
`!@`	Energy Ping	Trigger an event or pulse
`@flash`	Asynchronous Signal	Declares async behavior
`!loop`	Lightning Loop	Start a repeated action
`<>`	Dynamic Evaluation	Used around expressions to be evaluated in real time
`bolt{}`	Random Selector Block	Randomly pick from items inside
`spark{}`	Function Block	Defines a function
`>>`	Prompt or Input Trigger	Marks user input
`--//-->`	Thought Trail	A comment or annotation
`glow.mode`	Theme/Color Selector	Used for visual mode selection in editor/repl

💡 Example Usage with Symbols

Example 1

```
import# App
```

```
      # LoginPage


({

   imp-                                          --//--> start of program


   print(=⚡ Welcome to Lightning App ⚡=)

   print(=please log in below=)


   # user enters login details

   tro={username}

   tro={password}

   print(=enter username: =)

   >> username

   print(=enter password: =)

   >> password


   # check login

   if#username==shanil:

        if#password==1234:

             print(=✅ login successful=)

             return!=[dashboard]                --//--> take user to dashboard

        else:

             print(=❌ wrong password=)

             return!=[try-again]

   else:

        print(=❌ username not found=)

        return!=[try-again]


   emp-                                          --//--> end of program

})
```

Example 2

```
import# App
```

```
      # MoodResponder
({                                  imp-


    tro={mood}
    print(=how-are-you-feeling-today?=)
    >> mood                           --//--> user input


    if#mood==happy:
        print(=⚡great-energy-today!=)
        return!=[!@positive-vibes]


    elif#mood==tired:
        print(=⚡take-a-breath=)
        print(=🌙rest-is-power=)
        return!=[!@calm-mode]


    elif#mood==angry:
        bolt{
            print(=deep-breath-in=)
            print(=count-1-2-3=)
            print(=you-are-safe=)
        }
        return!=[storm-cleared]


    else:
        print(=emotion-unreadable=)
        return!=[unknown-wave]


    !loop 3x:
        print(=<3> stay-strong <3>)


    tro={thank-you}
    print(=glow-on!=)

    return!=[thank-you]


                                          emp-

})
```

🔽 Output Samples

Input: happy

```
how are you feeling today?

>> happy

⚡great energy today!

positive vibes

<3> stay strong <3>

<3> stay strong <3>

<3> stay strong <3>

glow on!

thank you
```

Input: tired

```
how are you feeling today?

>> tired

⚡take a breath

🌙rest is power

calm mode

<3> stay strong <3>

<3> stay strong <3>

<3> stay strong <3>

glow on!

thank you
```

🧬 Why Add Symbolic Elements?

Symbolic syntax:

- **Enhances creativity** – code looks like visual poetry
- **Reduces verbosity** – less typing, more meaning
- [illegible]

- **Creates identity** – makes Lightning stand out from other HLPLs
- **Feels expressive** – encourages emotion and connection to logic

📦 Updated Example Library in Lightning HLPL

Example Name	Description	Symbols Introduced
Hello World	Basic greeting and identity message	`tro`, `put`, `print`, `return!=`
Mood Responder	Responds to emotional state	`if#`, `bolt{}`, `>>`, `!loop`
Energy Pinger	Triggers symbolic "events" using `!@`	`!@`, `spark{}`
Random Oracle	Gives random wisdom	`bolt{}`, `<>`, `print(=)=`
Loop Affirmations	Repeats a message multiple times	`!loop`, `<3>`

✍️ Final Quote by the Creator

“Let code breathe. Let symbols talk. Let thought become light.”— *Shanil Aziz Malik (Cyber Code)*