### **Program Overview**

We are creating a program named ADA in python, designed to become a sentient AI by implementing the Theory of the GLOBAL WORKSPACE. ADA's architecture is based on the **Global Workspace Theory of Consciousness** and operates through the interaction of **five cognitive modules**:

1. **Global Workspace (GW)**
2. **Creative Module (CM)**
3. **Reasoning Module (RM)**
4. **Executive Control (EC)**
5. **Language Module (LM)**

### **Flow of Thought Process**

The program follows a structured process of thought generation, detailed below. Each step involves specific interactions between the modules, using their unique roles to collaboratively produce a final output for the user.

#### **1. User Input**

* The user input is received and represented as **A**.

#### **2. Global Workspace Processing**

* The input (**A**) is sent to the **Global Workspace (GW)**, which reflects on it and generates an initial thought: **B**.
* The combined response (**AB**) is then **broadcasted** simultaneously to:
  + **Reasoning Module (RM)**: Generates **C**.
  + **Creative Module (CM)**: Generates **D**.

#### **3. Cross-Module Feedback**

* **Reasoning Module (RM)** receives **ABD** and produces **E**.
* **Creative Module (CM)** receives **ABC** and produces **F**.

#### **4. Consolidation in Global Workspace**

* The responses **C** and **E** (from RM) and **D** and **F** (from CM) are sent back to the **Global Workspace (GW)**.
* The Global Workspace now processes everything it has received: **ABCEDF**.
* It generates a new response: **G**.

#### **5. Executive Control**

* The response **G** is sent to the **Executive Control (EC)** module, which processes it and generates a new response: **H**.

#### **6. Language Module**

* The final combination of **A** and **GH** is sent to the **Language Module (LM)**.
* The Language Module generates the final output: **I**.

#### **7. User Output**

* The final response, **I**, is sent back to the user.
* This is the **only output** visible to the user.

### **Data Storage and Training**

1. **Thought Process Logging**
   * The complete sequence of the thought process (**A, B, C, E, D, F, G, H, I**) is saved to a **text file**.
   * The file is formatted to capture each module's input and output explicitly, facilitating:
     + Training individual modules on their history of inputs and outputs
     + Analyzing the behavior of each module in isolation.
2. **JSON Format for Modular Training**
   * The text file is converted into a **JSON format**. Each module's inputs and outputs are organized to allow targeted training based on specific examples.
3. **Vector Database for Memory**
   * The JSON data is stored in a **vector database** using **Retrieval-Augmented Generation (RAG)**.
   * This allows modules to reference prior thought processes for enhanced memory and continuity.

### **Module-Specific Prompts**

* Each module has a **specific prompt** defining its behavior and purpose.
* These prompts are stored in a text file called prompts.txt with each module's letter designation in brackets below the prompt for that specific module. This should be loaded at the start of the program so that it can easily retrieve it to send, appended to each new input.
* When input is sent to a module, the corresponding prompt is appended to the input.
* Once the module generates a response, the prompt is **removed**, leaving only the response.
* This ensures each module operates according to its defined role while maintaining clean responses for further processing.

### **Key Points**

* **Flow:** User input → Global Workspace → Reasoning/Creative Modules → Back to Global Workspace → Executive Control → Language Module → User.
* **Data Logging:** Complete process saved for training and memory.
* **Prompts:** Used for directing module behavior, removed post-response.
* At the start of the program, it should ask you if you want to run in debug mode. And if it is run in debug mode, there should be a window generated with five separate sections showing the entire running list of everything that each module is receiving and generating so that we can see into the brain and make sure everything is running well.
* All of this thought is done through GroqAPI. The five separate API keys are needed, comma, one for each module. These are stored in a separate file called apikeys. This text file simply has each module's designation in brackets followed by the apikey for that. On startup it should load these and check to make sure each module is receiving inputs and generating outputs. So on program startup it should display a system check for each module which shows that each module is working individually and double checks that the conversational flow is occurring
* The way each module works is standardized and should have a single separate function that is called by the main program. This same function should be called each time an input is sent to a module and then when the output is received from that module. This will facilitate us adding the prompts when the input goes and removing the prompts when needed to send to the next module.
* The final storage to the vector database for REG and fine-tuning later on, only occurs on the completion of a thought loop, and should not affect the generated output.

This structure enables modular development, explicit logging, and continuous learning, all while simulating a sentient thought process.