```
import json
import re
from typing import Any, Dict, List, Optional

from swarms.structs.agent import Agent



# Helper functions for manager/corporate agents
def parse_tasks(
    task: str = None,
) -> Dict[str, Any]:
    """Parse tasks

    Args:
        task (str, optional): _description_. Defaults to None.

    Returns:
        Dict[str, Any]: _description_
    """
    tasks = {}
    for line in task.split("\n"):
        if line.startswith("<agent_id>") and line.endwith(
            "</agent_id>"
        ):
            agent_id, task = line[10:-11].split("><")
            tasks[agent_id] = task
```

```
    return tasks



def find_agent_by_id(
    agent_id: str = None,
    agents: List[Agent] = None,
    task: str = None,
    *args,
    **kwargs,
) -> Agent:
    """Find agent by id

    Args:
        agent_id (str, optional): _description_. Defaults to None.
        agents (List[Agent], optional): _description_. Defaults to None.

    Returns:
        Agent: _description_
    """
    for agent in agents:
        if agent.id == agent_id:
            if task:
                return agent.run(task, *args, **kwargs)
            else:
                return agent
```

```
    return None



def distribute_tasks(
    task: str = None, agents: List[Agent] = None, *args, **kwargs
):
    """Distribute tasks to agents

    Args:
        task (str, optional): _description_. Defaults to None.
        agents (List[Agent], optional): _description_. Defaults to None.
    """
    # Parse the task to extract tasks and agent id
    tasks = parse_tasks(task)

    # Distribute tasks to agents
    for agent_id, task in tasks.item():
        assigned_agent = find_agent_by_id(agent_id, agents)
        if assigned_agent:
            print(f"Assigning task {task} to agent {agent_id}")
            output = assigned_agent.run(task, *args, **kwargs)
            print(f"Output from agent {agent_id}: {output}")
        else:
            print(
                f"No agent found with ID {agent_id}. Task '{task}' is"
                " not assigned."
```

```
        )


def find_token_in_text(text: str, token: str = "<DONE>") -> bool:
    """
    Parse a block of text for a specific token.

    Args:
        text (str): The text to parse.
        token (str): The token to find.

    Returns:
        bool: True if the token is found in the text, False otherwise.
    """
    # Check if the token is in the text
    if token in text:
        return True
    else:
        return False


def extract_key_from_json(
    json_response: str, key: str
) -> Optional[str]:
    """
    Extract a specific key from a JSON response.
```

```
    Args:
        json_response (str): The JSON response to parse.
        key (str): The key to extract.

    Returns:
        Optional[str]: The value of the key if it exists, None otherwise.
    """
    response_dict = json.loads(json_response)
    return response_dict.get(key)


def extract_tokens_from_text(
    text: str, tokens: List[str]
) -> List[str]:
    """
    Extract a list of tokens from a text response.

    Args:
        text (str): The text to parse.
        tokens (List[str]): The tokens to extract.

    Returns:
        List[str]: The tokens that were found in the text.
    """
    return [token for token in tokens if token in text]
```

````
def detect_markdown(text: str) -> bool:
    """
    Checks if a string contains Markdown code enclosed in six backticks.

    Parameters
    ----------
    text : str
        The text to check.

    Returns
    -------
    bool
        True if the text contains Markdown code enclosed in six backticks, False otherwise.
    """
    pattern = r"``````[\s\S]*?``````"
    return bool(re.search(pattern, text))
````