```
import re


def extract_code_blocks_with_language(markdown_text: str):
    """
    Extracts all code blocks from Markdown text along with their languages.

    Args:
        markdown_text (str): The input Markdown text.

    Returns:
        list[dict]: A list of dictionaries, each containing:
                    - 'language': The detected language (or 'plaintext' if none specified).
                    - 'content': The content of the code block.
    """
    # Regex pattern to match code blocks and optional language specifiers
    pattern = r"```(\w+)?\n(.*?)```"

    # Find all matches (language and content)
    matches = re.findall(pattern, markdown_text, re.DOTALL)

    # Parse results
    code_blocks = []
    for language, content in matches:
        language = (
            language.strip() if language else "plaintext"
```

```
        )  # Default to 'plaintext'
        code_blocks.append(
            {"language": language, "content": content.strip()}
        )

    return code_blocks


def extract_code_from_markdown(
    markdown_text: str, language: str = None
):
    """
    Extracts content of code blocks for a specific language or all blocks if no language specified.

    Args:
        markdown_text (str): The input Markdown text.
        language (str, optional): The language to filter by (e.g., 'yaml', 'python').

    Returns:
        str: The concatenated content of matched code blocks or an empty string if none found.
    """
    # Get all code blocks with detected languages
    code_blocks = extract_code_blocks_with_language(markdown_text)

    # Filter by language if specified
    if language:
```

```
    code_blocks = [
        block["content"]
        for block in code_blocks
        if block["language"] == language
    ]
else:
    code_blocks = [
        block["content"] for block in code_blocks
    ]  # Include all blocks

# Return concatenated content
return "\n\n".join(code_blocks) if code_blocks else ""
```