## Overview  
  
This file provides utility functions for statically analyzing Python code. It uses Python's built-in `ast` (Abstract Syntax Tree) module to parse code content and extract structural information about its components, such as functions and classes, without executing the code.  
  
## Functions  
  
### `get\_functions\_and\_classes()`  
  
Parses a string of Python code to identify and extract metadata about all defined functions (synchronous and asynchronous) and classes.  
  
The function walks through the Abstract Syntax Tree (AST) of the provided code and collects details for each function and class definition it finds. If the input code contains syntax errors, the function will gracefully handle the exception and return an empty dictionary to prevent breaking the analysis process.  
  
#### Parameters  
  
- `code\_content (str)`: A string containing the Python source code to be analyzed.  
  
#### Returns  
  
- `Dict[str, Dict[str, Any]]`: A dictionary where each key is the name of a discovered function or class. The corresponding value is another dictionary containing the following metadata:  
 - `type (str)`: The type of the object, either "Function" or "Class".  
 - `name (str)`: The name of the function or class.  
 - `start\_line (int)`: The line number where the definition of the object begins.  
 - `end\_line (int)`: The line number where the definition of the object ends.  
  
#### Usage Example  
  
```python  
import ast  
from typing import List, Tuple, Dict, Any  
  
def get\_functions\_and\_classes(code\_content: str) -> Dict[str, Dict[str, Any]]:  
 """  
 Parses Python code content to extract information about functions and classes.  
 """  
 try:  
 tree = ast.parse(code\_content)  
 except SyntaxError:  
 # If file has syntax errors, return empty so it doesn't break analysis  
 return {}  
 objects = {}  
  
 for node in ast.walk(tree):  
 if isinstance(node, (ast.FunctionDef, ast.ClassDef, ast.AsyncFunctionDef)):  
 obj\_type = "Class" if isinstance(node, ast.ClassDef) else "Function"  
 objects[node.name] = {  
 "type": obj\_type,  
 "name": node.name,  
 "start\_line": node.lineno,  
 "end\_line": node.end\_lineno  
 }  
 return objects  
  
# Example usage:  
code = """  
import os  
  
class MyClass:  
 def \_\_init\_\_(self):  
 pass  
  
 def my\_method(self):  
 return "Hello"  
  
def my\_function():  
 print("World")  
"""  
  
analysis\_result = get\_functions\_and\_classes(code)  
import json  
print(json.dumps(analysis\_result, indent=2))  
  
# Expected Output:  
# {  
# "MyClass": {  
# "type": "Class",  
# "name": "MyClass",  
# "start\_line": 3,  
# "end\_line": 9  
# },  
# "my\_function": {  
# "type": "Function",  
# "name": "my\_function",  
# "start\_line": 11,  
# "end\_line": 12  
# },  
# "\_\_init\_\_": {  
# "type": "Function",  
# "name": "\_\_init\_\_",  
# "start\_line": 4,  
# "end\_line": 6  
# },  
# "my\_method": {  
# "type": "Function",  
# "name": "my\_method",  
# "start\_line": 8,  
# "end\_line": 9  
# }  
# }

```