## Introduction

This Python script is a **code documentation generator** designed to analyze `.py` files and produce readable documentation using the **Abstract Syntax Tree (AST)** module. By parsing the structure of the uploaded Python file, it identifies key components such as imports, classes, functions, assignments, loops, and function calls.

The application also uses **Gradio**, a web-based interface library, allowing users to easily upload Python files and view both the original source code and the auto-generated documentation side by side in a clean, interactive interface.

This tool is especially useful for:

- Developers exploring unfamiliar code
- Students learning Python code structures
- Teams documenting legacy codebases

---

## Line-by-Line Code Explanation

### 1. **Importing Required Libraries**

```
import ast
import gradio as gr
```

- `ast`: Used to parse Python code into a structured tree that can be analyzed.
- `gradio`: Creates the interactive user interface for uploading and analyzing Python files.

---

### 2. **CodeDocGenerator Class**

```
class CodeDocGenerator(ast.NodeVisitor):
```

- Inherits from `ast.NodeVisitor`, which allows traversal through the AST nodes of the parsed code.

Constructor

```
def __init__(self):
    self.docs = []
    self.indent_level = 0
```

- Initializes:
  - `self.docs`: List to hold Markdown documentation lines.
  - `self.indent_level`: Tracks indentation for nested structures (like functions inside classes).

Indentation Helper

```python
def indent(self):
    return "  " * self.indent_level
```

- Returns spaces based on the current indentation level for formatting nested sections.

---

### 3. **AST Node Visit Methods**

These methods are called automatically when visiting different parts of the parsed Python code:

visit_Module

```python
def visit_Module(self, node):
```

- Entry point of the AST.
- Adds a header and calls `generic_visit` to continue traversing the AST.

visit_Import and visit_ImportFrom

```python
def visit_Import(self, node):
def visit_ImportFrom(self, node):
```

- Detect and document `import` and `from module import ...` statements.

visit_ClassDef

```python
def visit_ClassDef(self, node):
```

- Documents class definitions, including their line number and docstring.
- Increases indentation to handle nested items like methods.

visit_FunctionDef

```python
def visit_FunctionDef(self, node):
```

- Documents function names, arguments, line numbers, and docstrings.
- Handles nested code using `visit_body_elements`.

visit_Assign

```python
def visit_Assign(self, node):
```

- Captures variable assignments (`a = 10`) and adds them to the docs.

visit_For

```python
def visit_For(self, node):
```

- Captures `for` loops and documents their structure (`for x in y`).

visit_Call

```python
def visit_Call(self, node):
```

- Documents function/method calls like `print()`, `self.method()`.

visit_Return

```python
def visit_Return(self, node):
```

- Captures and documents `return` statements in functions.

Helper to Traverse Bodies

```python
def visit_body_elements(self, body):
    for elem in body:
        self.visit(elem)
```

- Iterates through function or class bodies and visits each element recursively.

---

## 4. **analyze_code Function**

```python
def analyze_code(file):
```

- Accepts a file input (Python file).
- Reads and parses the file using `ast.parse()`.
- Uses `CodeDocGenerator` to analyze and return the generated documentation and the original code.
- Handles missing file or parsing errors gracefully.

---

## 5. **clear_all Function**

```python
def clear_all():
    return None, "", ""
```

- Resets the Gradio interface when the user clicks "Clear All".

---

## 6. **info_text String**

```python
info_text = """
### 🛈 General Information
...
"""
```

- Contains Markdown-formatted user instructions:
  - Overview of the tool
  - Usage instructions

- Technologies used
- Target audience

---

## 7. **Gradio Interface (UI) Setup**

```
with gr.Blocks(title="CodeExplain", theme=gr.themes.Base()) as iface:
```

Info Accordion

```
with gr.Accordion("ℹ About / Info", open=False):
    gr.Markdown(info_text)
```

- Collapsible section displaying general information about the tool.

UI Layout

```
with gr.Row():
    ...
```

- Organized in two columns:
    - **Left**: File upload, Clear button, and documentation output.
    - **Right**: Original source code display.

Event Triggers

```
file_input.change(fn=analyze_code, inputs=file_input,
outputs=[doc_output, code_output])
clear_btn.click(fn=clear_all, inputs=None, outputs=[file_input,
doc_output, code_output])
```

- Connects the file input and buttons to the corresponding backend functions.

---

## 8. **Application Launch**

```
if __name__ == "__main__":
    iface.launch()
```

- Runs the Gradio app when the script is executed directly.

---

## Conclusion

This script effectively combines Python's **AST parsing power** with a **user-friendly Gradio interface** to create a lightweight and accessible code documentation tool. By uploading any `.py` file, users can immediately visualize the structure of the code, making it especially helpful for:

- Quick overviews of large or legacy projects
- Students learning code organization
- Teams needing fast documentation for shared code

The auto-generated Markdown output can be copied, reused, or even exported for further documentation efforts.