🟢Retro Python Lab – Code Walkthrough (Beginner-Friendly)

This document explains **how your Streamlit app works**, in **simple language**, file by file. You can keep this as a **personal reference** or paste it into **Notion**.

Big Picture (How everything fits together)

Your project has **4 main files**, each with a clear responsibility:

File	Purpose
`app.py`	Main controller (decides what runs)
`calculator.py`	Calculator logic
`text_analyzer.py`	Text analysis logic
`retro.css`	Styling (retro look)

Think of it like: - `app.py` → **Brain / Director** - Other `.py` files → **Tools / Features** - `retro.css` → **Visual design**

1️⃣ `app.py` – Main Controller

Importing libraries and tools

```
import streamlit as st
from calculator import calculator
from text_analyzer import text_analyzer
```

What this means: - We import Streamlit and rename it to `st` (shorter and standard). - We import functions from other Python files. - This allows us to keep code **clean and modular**.

Loading custom CSS

```
with open("retro.css") as f:
    st.markdown(f"<style>{f.read()}</style>", unsafe_allow_html=True)
```

In simple words: - Open the `retro.css` file - Read all styling rules - Inject them into the Streamlit app

`unsafe_allow_html=True` is required because CSS is injected as HTML.

App title

```
st.title("🟢 Retro Python Lab ")
```

Displays the title at the top of the page.

Sidebar menu

```
menu = st.sidebar.selectbox(
    "Select Tool",
    ["Calculator", "Text Analyzer"]
)
```

What happens: - Creates a dropdown in the sidebar - Stores the selected option in the variable `menu`

Example values: - `menu = "Calculator"` - `menu = "Text Analyzer"`

Routing logic

```
if menu == "Calculator":
    calculator()
else:
    text_analyzer()
```

Meaning: - If user selects Calculator → run calculator tool - Otherwise → run text analyzer

This is how **one Streamlit app hosts multiple tools**.

2️⃣ `calculator.py` – Calculator Tool

Import Streamlit

```
import streamlit as st
```

Each file that uses Streamlit must import it.

Function definition

```
def calculator():
```

Defines a function named `calculator()`. Nothing runs until this function is **called from** `app.py`.

Number inputs

```
num1 = st.number_input("Enter first number")
num2 = st.number_input("Enter second number")
```

- Displays two input boxes
- Stores user input in variables `num1` and `num2`

Operation selector

```
operation = st.selectbox(
    "Choose operation",
    ["Add", "Subtract", "Multiply", "Divide"]
)
```

User selects an operation and it gets stored in `operation`.

Button logic

```
if st.button("Calculate"):
```

- Code inside runs **only when button is clicked**
- Streamlit reruns the script on every interaction

Calculation logic

```
if operation == "Add":
    result = num1 + num2
```

Basic Python conditional logic for math operations.

Error handling

```
if num2 == 0:
    st.error("Division by zero not allowed")
    return
```

Prevents app crash and exits the function safely.

Display result

```
st.success(f"Result → {result}")
```

Shows output in a green success box.

3️⃣ `text_analyzer.py` – Text Processing Tool

Text input

```
text = st.text_area("Enter text")
```

Stores user input as a **string**.

Empty input check

```
if not text.strip():
```

- `strip()` removes spaces
- Prevents running logic on empty input

Splitting text

```
words = text.split()
```

Converts sentence into a list of words.

Example:

```
"Hello world" → ["Hello", "world"]
```

Unique word count

```
unique_words = len(set(words))
```

- `set()` removes duplicates
- `len()` counts remaining items

Display results

```
st.write(f" 🔹 Words: {len(words)}")
```

Outputs analysis results to the UI.

4️⃣ `retro.css` – Styling Layer

Example rule

```
body {
    background-color: #0f172a;
    font-family: "Courier New", monospace;
}
```

Controls: - Background color - Font - Overall retro feel

CSS overrides Streamlit's default design.

🔁How Streamlit Works Internally

- Streamlit reruns the script **top to bottom**
- Every user interaction triggers a rerun
- UI state is handled automatically

This is why Streamlit feels simple but powerful.

🔝Key Concepts You Learned

- Modular Python programming
- Functions and imports
- Conditional logic
- Input validation
- UI-driven Python execution
- CSS injection
- Real-world app structure

Mental Model to Remember

> Streamlit = Python script + UI reruns

No frontend/backend separation required.

🚳Why this project is portfolio-worthy

- Clean structure
- Real deployment
- Understandable logic
- Custom UI
- Extendable architecture

You can now confidently explain:

- How your app works
- Why files are separated
- How Streamlit executes code
- How UI interacts with Python

This is **real understanding**, not tutorial memorization.