Hey there! Let's break down this project's code step-by-step, as if you're a beginner. Think of this project like a small, digital office where different people have different jobs.

`calculator.py` - The Smart Accountant

Imagine this file is a super-smart accountant. Its job is to handle all the difficult calculations and crunch the numbers. It doesn't care about what the final report looks like; it just focuses on getting the math right.

This file uses a concept called **Object-Oriented Programming (OOP)**. Don't let the big name scare you! It's just a way of organizing code into "objects" that represent real-world things. In this case, the things are `Trade`, `MatchedTrade`, and the main `InvestorCalculator`.

1. The `Trade` Class This is the simplest object. Think of it as a single, neat receipt for one stock transaction.

```
class Trade:
    """Represents a single trade transaction"""

    def __init__(self, date, trade_type, stock, qty, price, brokerage, dividend=0):
        # When a new "receipt" is created, this function runs.
        # It takes all the info from a row in your CSV file and saves it.
        # It's like writing down the details of a transaction in your ledger.
        self.date = date
        self.trade_type = trade_type.upper()  # 'BUY' or 'SELL'
        self.stock = stock
        self.qty = qty
        self.price = price
        self.brokerage = brokerage
        self.dividend = dividend
        self.remaining_qty = qty  # This is crucial for FIFO! It keeps track of how much st
```

2. The `MatchedTrade` Class This is where the magic happens. A `MatchedTrade` object represents a perfect pair: one `BUY` trade and one `SELL` trade. It calculates the profit, loss, and GST for that specific pair.

```
class MatchedTrade:
    """Represents a matched buy-sell pair for capital gains calculation"""

    def __init__(self, buy_trade, sell_trade, matched_qty):
        # This function takes a buy trade and a sell trade, and figures out the details.

        # It calculates the total brokerage for this pair.
        self.total_brokerage = ...
```

```
        # It calculates the profit or loss (the "gain").
        self.gain = (self.sell_price * matched_qty) - (self.buy_price * matched_qty) - self.

        # It checks how long the stock was held to see if it's STCG or LTCG.
        days_held = (self.sell_date - self.buy_date).days
        self.is_ltcg = days_held > 365
        self.gain_type = "LTCG" if self.is_ltcg else "STCG"

        # It calculates GST at 18%.
        self.gst_on_brokerage = self.total_brokerage * 0.18

    def to_dict(self):
        # This function neatly organizes all the calculated info into a dictionary.
        # The main app will use this to create the final report table.
        return { 'Buy Date': self.buy_date, 'Sell Date': self.sell_date, ... }
```

3. The `InvestorCalculator` Class This is the head of the accounting firm. It manages all the `Trade` and `MatchedTrade` objects and puts them to work.

```
class InvestorCalculator:
    """Main calculator class for processing trades and calculating taxes"""

    def __init__(self):
        # Sets up empty lists to hold all our "receipts" and "matched pairs".
        self.trades = []
        self.matched_trades = []

    def load_csv_data(self, csv_file):
        # This function reads your CSV file, checks if it has the right columns,
        # and then creates a `Trade` object for each row, storing them in a list.
        ...

    def calculate_fifo_matching(self):
        # This is the core logic. It looks at all your trades, stock by stock.
        # It takes the first (oldest) "BUY" trade and tries to match it with a "SELL" trade.
        # This is the "First In, First Out" (FIFO) method.
        # It updates the `remaining_qty` on the `Trade` objects as it goes.
        # It creates a `MatchedTrade` object for every successful pair.
        ...

    def calculate_summary(self):
        # After all the matching is done, this function totals everything up.
        # It sums up all the STCG, LTCG, GST, etc., to give you the final numbers.
        ...
```

```
    def process_portfolio(self, csv_file):
        # This is the main button you "press" to get things done.
        # It calls the other functions in the right order:
        # 1. load_csv_data()
        # 2. calculate_fifo_matching()
        # 3. calculate_summary()
        # Finally, it returns the detailed report and the final summary.
        ...
```

`app.py` - The Presentation Designer

This file is a presentation specialist. It doesn't do any calculations itself. Instead, it takes the results from `calculator.py` and displays them in a beautiful, easy-to-read web page using the **Streamlit** library.

The `main()` function is the only part that runs. It's like a script for building the web page.

```
# Imports Streamlit and the calculator logic
import streamlit as st
from calculator import InvestorCalculator

def main():
    # --- The Setup ---
    # st.set_page_config() configures the page title and layout.
    st.title(" Investor ITR & GST Calculator")
    st.markdown(...) # This adds the main title and description.

    # --- The Sidebar (on the left) ---
    with st.sidebar:
        st.header(" Upload Your Portfolio")
        uploaded_file = st.file_uploader(...) # This creates the button for uploading your C
        st.subheader(" CSV Format Required:")
        st.dataframe(...) # This shows you what the CSV should look like.

    # --- The Main Content Area ---
    if uploaded_file is not None:
        # This code only runs AFTER a file has been uploaded.

        # It creates an instance of our "accountant" class.
        calculator = InvestorCalculator()

        # It calls the main "process_portfolio" method from the accountant.
        # This is where all the calculations from `calculator.py` happen!
        results_df, summary = calculator.process_portfolio(uploaded_file)
```

```
        # Now, it uses Streamlit to display the results.
        st.subheader("  Tax Summary")
        col1, col2, col3, col4 = st.columns(4) # Divides the page into 4 columns.
        with col1:
            st.metric(...) # Displays a small box with the total STCG.

        # ... and so on for all the other metrics and tables.

        # It uses st.dataframe() to show the detailed results table.
        # It uses st.download_button() to let you download the reports as CSVs.

        # It even creates simple charts using st.bar_chart() to visualize your data!
        ...

    else:
        # This part runs when no file has been uploaded yet.
        # It shows the user the introductory message and instructions.
        st.subheader("  Get Started")
        st.markdown(...)
```

In short, the two files work together like a team:

- `calculator.py` is the behind-the-scenes expert that handles all the hard work and complex financial logic.
- `app.py` is the front-facing "face" of the project, taking the expert's work and presenting it beautifully to the user on a web page.