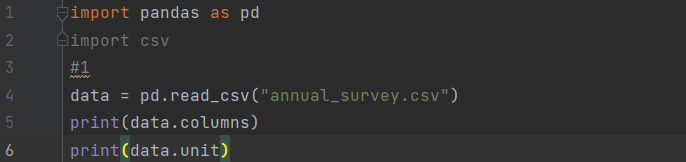
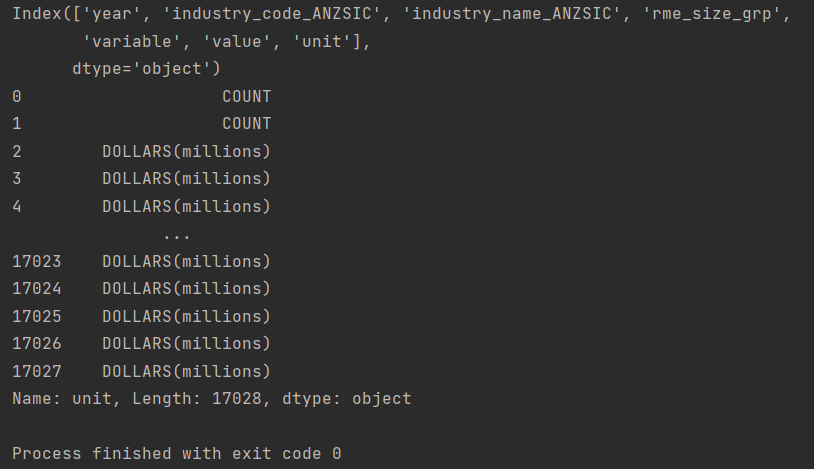
**Read data from csv files using pandas:**

This code uses a Python library called Pandas to work with data from a CSV file named "annual\_survey.csv." First, it brings in Pandas and gives it a shortcut name 'pd' for easier use. Pandas is known for making it easy to handle structured data like what you find in CSV files. The second line uses the Pandas `read\_csv` function to grab the information from the CSV file and put it into a Pandas DataFrame. Think of a DataFrame like a table with rows and columns, and now our data from "annual\_survey.csv" is stored in this DataFrame named `data`.The next part of the code prints out the column names in the DataFrame (`print(data.columns)`). Columns are like the different categories of information in the data. This line helps us see what kinds of data we have in our table. Finally, the last line tries to print the contents of a specific column named 'unit' (`print(data.unit)`). You should replace 'unit' with the actual name of a column in your dataset. This shows how to get and display data from a particular column in the DataFrame. To sum it up, the code demonstrates how Pandas makes it simple to load and explore data from a CSV file. It's a user-friendly way to handle data, especially for tasks like data analysis.

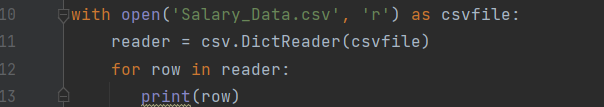


**Output:**



**DictReader:**

This code opens a file named 'Salary\_Data.csv' that contains data in a table format. It uses a special reader called `DictReader` to read the data, treating each row as if it were a dictionary. In this 'dictionary,' the keys are the names of the columns, and the values are the actual data in each row. The 'with' statement ensures that the file is closed properly, even if there's an issue during the reading process. Then, the code goes through each row in the file one by one and prints out the data in a structured way. Instead of just listing the values, it organizes them like a dictionary, with the column names as labels. In simpler terms, the code helps us look at data in a CSV file more neatly. It treats each row like a dictionary, making it easier to understand and work with the information.



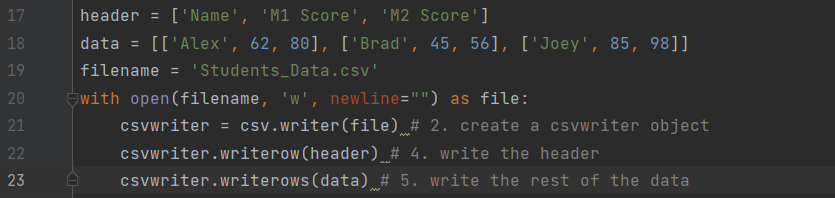
**Output:**



Because the file is empty.

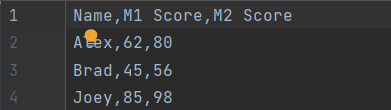
**Write CSV file Using csv.writer:**

This code makes a file called 'Students\_Data.csv' to save details about students. First, it decides what labels or columns the table should have, like 'Name', 'M1 Score', and 'M2 Score', and calls them 'header'. Then, it puts together the actual student data in the 'data' variable, where each student has a row with their name, score in the first subject ('M1 Score'), and score in the second subject ('M2 Score'). The file is named 'Students\_Data.csv' so the computer knows what to call it. The code opens this file in a way that handles errors properly, making sure everything is neat with the 'with' statement and ensuring correct line endings. A special tool called 'csvwriter' is used to help write the information in a way that works for CSV files. It writes the column names ('header') at the top of the page, acting like headers for the table. After that, the code writes down the actual student details from 'data' into the file. Each row in 'data' represents a student's information, making a structured table in the 'Students\_Data.csv' file. In simpler words, this code organizes student details into a table, specifies the column names, and writes it into a CSV file so we can easily understand and use the information.



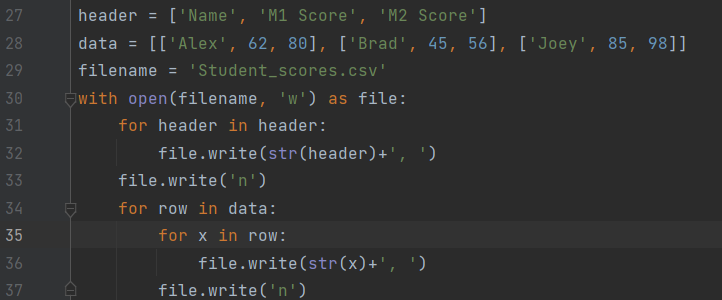
**Output:**

Csv file



**Write CSV File Using .writelines():**

This code is like making a table called 'Student\_scores.csv' and writing down student information directly. First, it decides what the column names will be—like 'Name', 'M1 Score', and 'M2 Score'. Then, it writes these names to the table, making sure to add a comma and space between each one. After that, it adds a newline character to start a new line for the actual student data. Next, for each student, it writes their details to the table. It puts their name, 'M1 Score', and 'M2 Score', adding commas and spaces in between, and then starts a new line for the next student. So, in simple words, this code manually creates a table of student scores, writing column names first and then adding each student's information row by row. While this method is straightforward, using the `csv` module can be more helpful for complicated tasks.



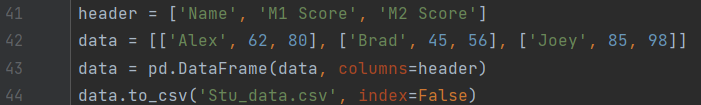
**Output:**

Csv file



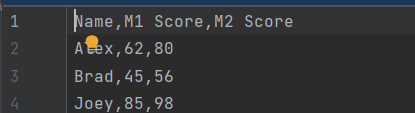
**Writing a CSV Using Pandas:**

First off, we decide on the names for our table columns with `header = ['Name', 'M1 Score', 'M2 Score']`. It's like saying, "Alright, our table will have columns for names, scores in M1, and scores in M2." Then, we list out the actual student details with `data = [['Alex', 62, 80], ['Brad', 45, 56], ['Joey', 85, 98]]`. Each little list inside the big list tells us about a student, including their name and scores. Now comes the magic of Pandas. `data = pd.DataFrame(data, columns=header)` is like putting all our student details into a neat table. Think of it as organizing your messy data into rows and columns, making it super easy to work with. Finally, `data.to\_csv('Stu\_data.csv', index=False)` is where Pandas saves the day. It turns our organized table into a CSV file named 'Stu\_data.csv'. The `index=False` part just means we don't want to include row numbers in our file; we like it clean and simple.



**Output:**

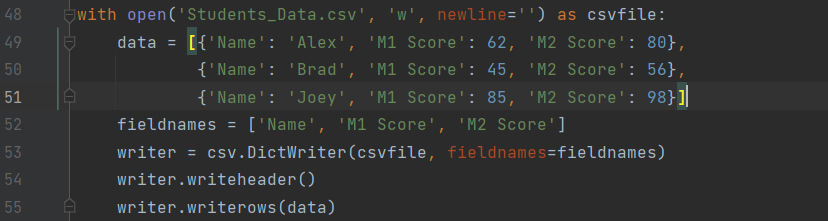
Csv file



**Writing a CSV File Using csv.DictWriter:**

The initial line, `with open('Students\_Data.csv', 'w', newline='') as csvfile:`, marks the beginning of the file-handling process. This 'with' statement ensures that the file is opened in write mode ('w'), facilitating the creation of a new file or overwriting an existing one. The 'newline=""' component ensures consistent line endings, and the 'with' statement guarantees proper closure of the file even in the presence of potential errors. The next step involves the creation of a list named `fieldnames`, encompassing the column names. This list serves as a reference for the subsequent writing process, indicating the headers for the CSV file. It essentially defines the structure of the data to be included in the table. The `csv.DictWriter` class is then invoked to instantiate a writer object named `writer`. This specialized class is designed for writing dictionaries into CSV files, aligning the provided dictionaries with the specified fieldnames. The `csvfile` parameter signifies the file object to which the data will be written. The line `writer.writeheader()` marks the insertion of the header row into the CSV file, effectively labeling each column with its corresponding name. This step establishes a clear and organized structure for the ensuing data rows.

Finally, the `writer.writerows(data)` line orchestrates the writing of the actual student data into the CSV file. The `DictWriter` seamlessly translates each dictionary in the 'data' list into a row, maintaining the association between keys and values in accordance with the specified column names. In essence, this code is a testament to the versatility of Python's `csv` module in handling structured data. It showcases a streamlined process of creating, labeling, and populating a CSV file with student information, emphasizing clarity and efficiency in data organization. This approach offers a pragmatic solution for scenarios where a lightweight and straightforward CSV format suffices for data storage and sharing.



**Output:**

Csv file

