Reading Input and Writing Output

One of the most important skills I developed while working with MATLAB was the ability to read data from files and write results back to files. File input and output (I/O) is a fundamental aspect of data analysis, as it allows me to access external datasets, process them, and save meaningful results. Whether dealing with text, images, videos, or numeric data, MATLAB offers a comprehensive set of tools to handle file I/O efficiently.

When I started working with file I/O, I quickly realized the importance of understanding the structure of the data I was working with. Each file has its own format, whether it's a plain text file with delimiters, a spreadsheet, or a more complex binary format. Before diving into a file, I learned to ask questions such as: What type of data does the file contain? Are there specific delimiters or patterns to consider? Are there missing values or inconsistencies in the data? These questions helped me choose the most appropriate MATLAB functions for reading and processing the data.

Reading data from files often began with the `fopen` function, which opened the file for further processing. Paired with `textscan`, it gave me precise control over how data was read. For example, I could specify the format of each column, handle delimiters, and even skip header lines. This flexibility proved invaluable when working with files containing a mix of numeric and textual data. The process was straightforward but powerful, allowing me to handle both small and large datasets with ease.

Once the data was imported, the next step was processing it. This often involved parsing strings, handling numeric arrays, and calculating derived metrics. For example, when working with text data, I frequently used functions like `strsplit` to break strings into components or `regexprep` to clean and modify text patterns. When handling numeric data, MATLAB's array operations made it simple to compute metrics such as averages, sums, or statistical measures.

Writing data back to files was equally important. MATLAB's `fprintf` function allowed me to create custom-formatted text files, perfect for summarizing results or creating reports. On the other hand, for saving large datasets or preserving MATLAB variables, the `save` function was indispensable. It allowed me to store data in `.mat` files, ensuring that all variable structures were maintained for future use. This was especially helpful in projects that required iterative analysis or when sharing data with collaborators.

Through my experience, I developed a few key best practices for file I/O. Always closing files with `fclose` became second nature, as it prevents resource leaks and ensures smooth operation. I also learned to handle potential errors gracefully using try-catch blocks, especially when working with external files that might be missing or corrupted. Documenting the file structure and any assumptions I made about the data helped maintain clarity and reproducibility in my workflows.

In conclusion, mastering file I/O in MATLAB has been a transformative experience. It enabled me to seamlessly integrate MATLAB with external data sources, process complex datasets, and produce meaningful outputs. By understanding the nuances of reading and writing files, I've been able to handle diverse projects more efficiently and confidently. File I/O is more than just a technical skill—it's a gateway to unlocking the full potential of MATLAB in data-driven problem-solving.

Reading Data from Files:

Using `fopen` and `textscan`

```
% Creating Fictional Data
% I create a fictional dataset to simulate reading a file.
data = {
    'Apple', 100, 20, '$1.50';
    'Banana', 200, 50, '$0.75';
    'Orange', 150, 30, '$1.25';
    'Grapes', 120, 10, '$2.00';
};

% Writing the fictional data to a temporary file for demonstration.
inputFile = 'temp_test.txt'; % Temporary input file path.
fileID = fopen(inputFile, 'w'); % Open the file in write mode.

% Write the header and data to the file.
fprintf(fileID, 'Fruit,Total Units,Units Left,Price Per Unit\n');
```

```
Error using fprintf
Invalid file identifier. Use fopen to generate a valid file identifier.
```

```
for i = 1:size(data, 1)
    fprintf(fileID, '%s,%d,%d,%s\n', data{i, :});
end
fclose(fileID); % Close the input file.

% Reading Data from Files
% I open the file containing my data.
fileID = fopen(inputFile, 'r'); % The 'r' flag indicates reading mode.
if fileID == -1
    error('Failed to open file. Please check the file path: %s', inputFile);
else
    disp('File opened successfully.');
end

% I use textscan to parse the file. The format specifies data types:
% %s for strings, %f for floating-point numbers, and ',' as the delimiter.
C = textscan(fileID, '%s %f %f %s', 'Delimiter', ',', 'HeaderLines', 1);
fclose(fileID); % Close the input file.
disp('Data loaded into cell array C:');
disp(C); % Display the parsed data.
```

Processing Imported DataMATLAB Code: Calculating Units Sold

```
% Processing Imported Data: Calculating Units Sold
% I calculate the number of units sold by subtracting the third column (units left) from the second column (total units).
sold = C{2} - C{3}; % Element-wise subtraction to get the units sold.
disp('Units sold:');
disp(sold); % Display the vector of units sold.
```

Converting Data Types:

Parsing Currency Values

```
% Converting Data Types: Parsing Currency Values
% The price per unit contains a "$" symbol. I remove this symbol and convert the remaining strings to numeric values.
D = cellfun(@(x) str2double(regexprep(x, '\$', '')), C{4}, 'UniformOutput', false);
disp('Prices after removing "$":');
disp(D); % Display the cleaned price data.

% I convert the cleaned cell array of numbers into a numeric matrix.
E = cell2mat(D); % Converts cell array to matrix.
disp('Numeric matrix of prices:');
disp(E); % Display the matrix of numeric prices.
```

Calculating Revenue

```
% Calculating Revenue
% I calculate revenue by multiplying units sold by the cost per unit.
revenue = sold .* E; % Element-wise multiplication.
disp('Revenue for each fruit:');
disp(revenue); % Display the revenue vector.

% Summing up the revenue to find the total revenue.
totalRevenue = sum(revenue); % Total revenue.
disp(['Total Revenue: $', num2str(totalRevenue)]); % Display the total revenue.
```

Writing Output to FilesMATLAB Code: Saving Results

```
% Writing Output to Files
% I save the revenue data to a new file for reporting.
outputFile = 'revenue_report.txt'; % Define the output file path.
fileID_out = fopen(outputFile, 'w'); % Open the file in write mode.
fprintf(fileID_out, 'Fruit\tUnits Sold\tRevenue\n'); % Write the header.

% Write the data row by row.
for i = 1:length(C{1})
    fprintf(fileID_out, '%s\t%d\t%.2f\n', C{1}{i}, sold(i), revenue(i));
end

fclose(fileID_out); % Close the output file.
disp(['Results saved to ', outputFile]);
```