Okay, excellent! This is a really solid breakdown of the core DAX functions you'll need. You've got the aggregations, the crucial context modification with CALCULATE and the filter functions (FILTER, ALL, ALLEXCEPT), the row-by-row power of iterators, and table creation with SUMMARIZE. DIVIDE and IF are essential utilities.

Let's add GROUPBY and ORDERBY to your notes. GROUPBY is another powerful way to create summary tables, often considered more flexible than SUMMARIZE, and ORDERBY is how you control the sorting of tables returned by DAX functions.

Here are the new sections you can add:

**9. Another Way to Summarize: GROUPBY (Modern Table Summarization)**

While SUMMARIZE is great, GROUPBY is a more modern function often used alongside SUMMARIZECOLUMNS (another table function not covered here, but good to know exists). GROUPBY is particularly useful because it can group by expressions, not just physical columns, and it often works well when you need to define aggregations *after* grouping. Like SUMMARIZE, it creates a new table.

How I write it:

Code snippet

GROUPBY(

Which\_Table\_To\_Group,

Column\_To\_Group\_By1 [, Column\_To\_Group\_By2]...,

"MyNewColumnName1", My\_Aggregation\_Expression1 [,

"MyNewColumnName2", My\_Aggregation\_Expression2]...

)

* Which\_Table\_To\_Group: The table you're starting with.
* Column\_To\_Group\_ByN: The columns (or expressions) you want to group by.
* "MyNewColumnNameN": The name you give to the new column holding the aggregated result.
* My\_Aggregation\_ExpressionN: The calculation to perform for each group. This often uses functions like CURRENTGROUP() to refer to the rows within the current group.

Example (for a new calculated table):

Code snippet

SalesByProductCategoryAndColor =

GROUPBY(

SalesData,

SalesData[ProductCategoryColumn],

SalesData[ProductColorColumn],

"TotalQuantitySold", SUMX(CURRENTGROUP(), SalesData[QuantityColumn]), -- Use SUMX with CURRENTGROUP()

"AverageSaleAmount", AVERAGEX(CURRENTGROUP(), SalesData[AmountColumn]) -- Use AVERAGEX with CURRENTGROUP()

)

*(Notice the use of SUMX(CURRENTGROUP(), ...) and AVERAGEX(CURRENTGROUP(), ...) inside GROUPBY. CURRENTGROUP() refers to the temporary table of rows belonging to the current group being processed. This is a key difference from SUMMARIZE where you just use aggregate functions like SUM() directly on the column.)*

**10. Controlling Table Order: ORDERBY (For Sorting Results)**

When you use DAX functions that return a *table* (like SUMMARIZE, SUMMARIZECOLUMNS, GROUPBY, ADDCOLUMNS, FILTER, etc.), you might want to specify the order in which the rows in that resulting table are arranged. ORDERBY is the function you use *within* these table functions to achieve that.

How I write it (used inside another table function):

Code snippet

ORDERBY(

Expression\_To\_Sort\_By1 [, Sort\_Direction1 [, Expression\_To\_Sort\_By2 [, Sort\_Direction2]]]...

)

* Expression\_To\_Sort\_ByN: The column or expression you want to sort the table by.
* Sort\_DirectionN: Specify ASC for ascending (A-Z, 0-9, earliest to latest) or DESC for descending (Z-A, 9-0, latest to earliest). If you don't specify, ASC is the default.

Example (combining with SUMMARIZECOLUMNS, a common pattern):

(Note: SUMMARIZECOLUMNS is often used with ORDERBY. It's another table function similar to SUMMARIZE but with some different behaviors regarding context transitions.)

Code snippet

Top5SalesByCity =

TOPN( -- TOPN is another table function to get the top N rows

5, -- Get the top 5 rows

SUMMARIZECOLUMNS(

SalesData[City], -- Group by City

"TotalCitySales", SUM(SalesData[AmountColumn]) -- Calculate total sales per city

),

[TotalCitySales], DESC -- ORDERBY is implicitly handled by TOPN here, sorting by TotalCitySales DESC

)

Example using ORDERBY explicitly within ADDCOLUMNS (adds columns to an existing table):

Code snippet

ProductsRankedBySales =

ORDERBY( -- Applying ORDERBY to the result of ADDCOLUMNS

ADDCOLUMNS( -- Add new calculated columns to the ProductsData table

ProductsData,

"TotalProductSales", CALCULATE(SUM(SalesData[AmountColumn]), FILTER(SalesData, SalesData[ProductID] = ProductsData[ProductID])) -- Calculate total sales for each product

),

[TotalProductSales], DESC, -- Sort primarily by TotalProductSales descending

ProductsData[ProductName], ASC -- Then sort by ProductName ascending for ties

)

*(In this ADDCOLUMNS example, ORDERBY wraps the ADDCOLUMNS function to sort the table returned by ADDCOLUMNS.)*

**Sorting Queries (Clarification):**

When you talk about "sort queries" in DAX, you're generally referring to sorting the *output table* of a DAX expression. The primary way to do this within the DAX language itself is using the ORDERBY function, usually in conjunction with other table-returning functions like SUMMARIZECOLUMNS, GROUPBY, ADDCOLUMNS, or by wrapping expressions that return tables (like in the last ADDCOLUMNS example above). Sorting of visuals in a report is typically handled by the visual's properties pane, but if you define a *calculated table* using DAX, you can use ORDERBY to define its default sort order.

Add these sections to your notes. They introduce GROUPBY as a modern alternative/complement to SUMMARIZE and clearly explain how ORDERBY is used to control the sorting of the tables created by your DAX expressions. Good luck with your assignments!