# Data Massage i.e. Data manipulation

## Introduction:

- Data manipulation is the process of changing or transforming data in order to make it more useful for our needs or more presentable. It's a crucial step in data usage, data analysis and visualization as it allows you to extract insights from raw data.
- Many times, the raw data you will get from a server will be structured in a complicated way, with deep nesting arrays and objects, and a lot of unnecessary data, and your job will be to extract meaningful and useful data for your specific needs.
- The types of data sets that will be covered in this presentation are objects, arrays, nested objects and nested arrays. These data structures are commonly used in JavaScript and are essential for working with complicated data sets.

# Extracting existing data

- One of the most basic operations in data manipulation is extracting data from objects. In JavaScript, you can access properties of an object using **dot notation** (object.property) or **bracket notation** (object['property']). This allows you to retrieve the value of a specific property within the object while iterating over the object's items or when you need to extract or assign data based on variables you hold.

- Arrays are also a fundamental data structure in JavaScript. To access an element in an array, you can use **indexing** (array[index]). You can also use loops such as **for loops** or the **forEach** method to iterate over the array and access each element. You can use the **filter** and the **find** methods to extract the data you need. You can also combine some of those methods. You can also use **logical operators** or the **includes** method to evaluate whether it is the data you need.

- Nested data structures, such as objects within objects or arrays within arrays, can be more complex to work with. However, you can use a combination of the above methods to access and extract data from nested structures. For example, if you have an object with a property that is an array, you can use **bracket notation** to access the property in the object and then **indexing** to access an element within the array.

# Creating new computed data

- Once you have extracted the data you need, you can use JavaScript's built-in array and object methods to create new data from existing data sets. For example, the **map()** method allows you to create a new array with the results of calling a provided function on every element in the original array. The **filter()** method allows you to create a new array with all elements that pass the test implemented by the provided function. The **reduce()** method allows you to iterate over an array and reduce it to a single value, such as a sum or an average. The **slice()** method allows you to get a specific item you need from the array
- JavaScript also has built-in math functions that can be used to compute new data. For example, **Math.max()** and **Math.min()** can be used to find the maximum and minimum value in an array, respectively. You can also use **date** methods to extract dates and calculate time periods etc. You can use the **sort()** method to sort the data as you need. You can use **bracket notation** in new objects you create and populate them with computed values. Use the **.push()** method to add new element to arrays you create etc.

# Practical approach

- When working with data manipulation, it's important to have a clear and organized approach. This includes **understanding the source data set** you are working with. It is a good practice to **split up the window** in the code editor and have the data set on one side and your code on the other side, so you can see the referred data all the time.
- It is also important to use **meaningful variable names**, to **comment your code** while you work, and **breaking complex tasks into smaller ones**. It's also important to keep in mind that data manipulation is an iterative work process, so you can start with basic for loops and many basic methods, and than refactor the code and make it cleaner and more efficient after you are done.
- There are common problems that can arise during data manipulation such as dealing with missing values or inconsistent data types. It's important to be able to **debug and troubleshoot** these issues and find where you made the mistake. You can use **break points**, the **debugger** and **console.logs** to trace the origin of the bug.

# Conclusion

In this presentation, we covered the basics of data manipulation in JavaScript including extracting existing data, creating new computed data, and a practical approach to working with complicated data sets.