On my 14th day of the internship at Surfboard Payments, I started my morning session by creating my own array and writing an algorithm to find values in the array using the binary search concept. My task was to develop a JavaScript application that implements a binary search algorithm with dynamic input values. As I worked on this task, I learned more about the binary search algorithm, how it works, and how to implement it in JavaScript. The implementation process was still ongoing, but I managed to understand the core concepts behind binary search. This included the idea of dividing a sorted array into two halves and repeatedly narrowing down the search range until the target value is found. Through this, I also gained a better understanding of JavaScript functions, loops, and conditional statements, which are essential for implementing algorithms. Although I was unable to complete the task within the day, I made significant progress and improved my problem-solving skills related to searching algorithms.

In the afternoon session, the internship class started, so we went downstairs and began by recalling what we had learned the previous day. We revised the topic of data types, followed by a review of tokens, symbols, keywords, and expressions. During the discussion, we focused more on strings and their properties. For example, when declaring a string in JavaScript, any value enclosed in quotation marks is considered a string. If we declare a variable as let a="10" and another as let b=16, the output of a+b would be "1016" instead of performing numerical addition, because "10" is treated as a string. This helped me understand how JavaScript handles string concatenation when combining different data types.

We then moved on to arrays, which are used to store multiple values of different data types within a single entity. For example, an array can contain numbers, strings, and boolean values in the same collection, such as a=[a,16,"Avinash",true]. Understanding arrays is important because they allow us to manage multiple pieces of data efficiently and perform various operations like searching, sorting, and modifying data.

After arrays, we discussed objects in JavaScript. Objects are defined using key-value pairs, where each key is unique, but values can be the same. For example, in an object declaration like a=10, b=20, c=16, the variables a, b, and c act as keys, and their respective values are stored in the object. This concept is widely used in JavaScript programming to represent real-world entities and manage data in an organized manner.

Next, we learned about functions in JavaScript. A function is a block of code that performs a specific task and can return a value. Every function has a return type, which defines what kind of value the function will return after execution. For example, a function to return a full name can be written as follows: "string fullname(fname,lname){return(fname,lname)}". However, declaring a function alone is not enough; it must be called in order to execute and display the output. If a function is not called, it remains inactive and does not perform any operations. This helped me understand the importance of function calls and how they control program execution.

After discussing functions, we moved on to the concept of typed and untyped languages. A typed language requires explicit declaration of data types before assigning values to variables. Examples of typed languages include TypeScript and Dart. For instance, in a typed language, declaring "a=10" without specifying the type will result in an error, whereas writing "int a=10" ensures that the variable is correctly defined with an integer data type. On the other hand, untyped languages like JavaScript do not require explicit type declarations. In JavaScript, we can simply write "a=10;b=18;c=97;" without specifying data types, and the program will still execute correctly. This makes JavaScript more flexible and easier to use for dynamic applications.

We also discussed how functions work in untyped languages. Unlike typed languages where functions require a specified return type, JavaScript functions can be created without explicitly defining a return type. For example, a simple addition function in JavaScript can be written as "void addition(a,b){a+b;}" and called using "addition(10,18);". Even though the function does not have a return type, it still performs the addition operation. This flexibility in JavaScript functions allows for a wide range of implementations and simplifies coding in many cases.

By the end of the session, we had covered multiple important topics related to JavaScript programming, including arrays, objects, functions, and the difference between typed and untyped languages. The discussion helped me strengthen my understanding of core programming concepts and how they are applied in real-world scenarios. With this, the session came to an end, and we went back upstairs and continue by task.