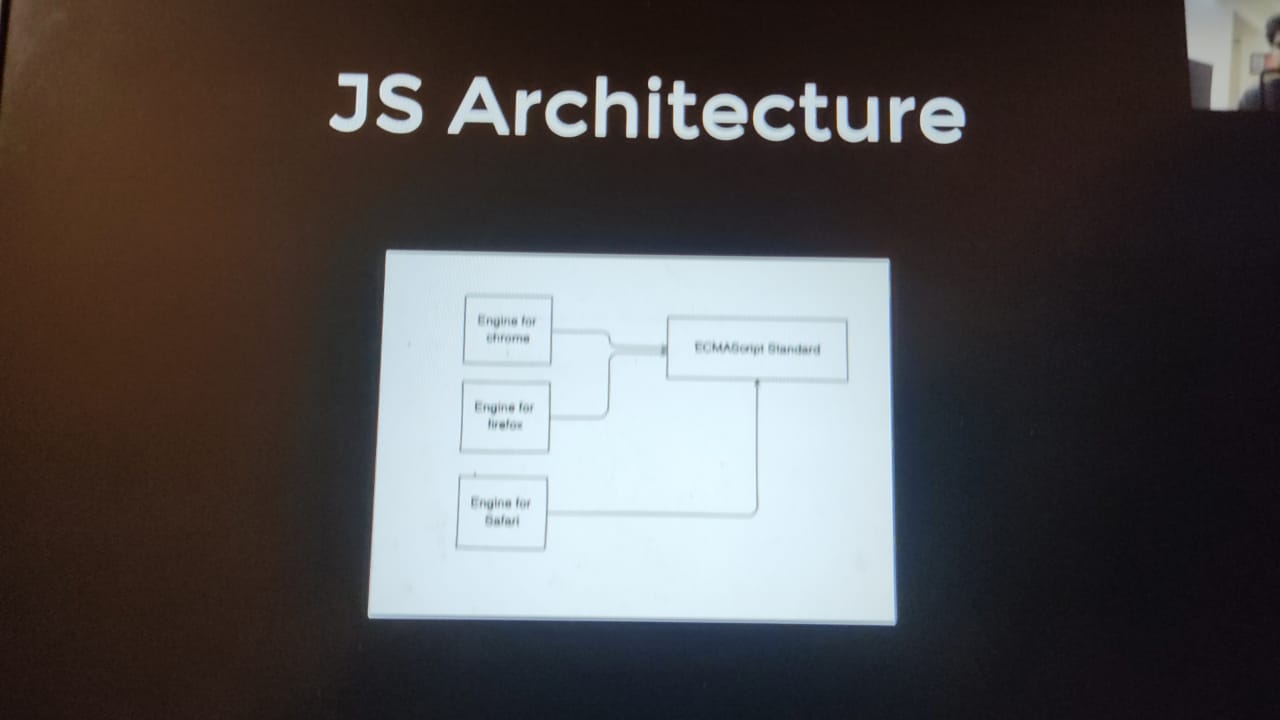
WEEK 0 : 09-06-23

->High level languages was introduced to write our code in a English like languages which can be easily converted to 0’s and 1’s by the compiler on compilation.



Great questions! Let me explain high-level languages, assembly language, and the concept of 0's and 1's to you.

Q. What is a high level language and what are assembly languages and what does the concept of 0’s and 1’s indicate?

A high-level language is like a special code that humans can understand and use to write programs. It's designed to be easier for us to read and write compared to machine language, which is a language computers understand directly. High-level languages are closer to how we speak and think, which makes programming more efficient and less prone to errors.

Imagine you want to tell a robot how to make a peanut butter and jelly sandwich. In a high-level language, you could say something like:

spreadPeanutButter();

spreadJelly();

putSlicesTogether();

cutIntoHalves();

See, it's easy to understand what the instructions mean. High-level languages provide these types of commands and structures that make programming more intuitive and readable.

Now, let's talk about assembly language. Assembly language is a low-level language that's a step closer to the machine language. It uses words and symbols to represent the instructions that a computer can understand directly.

If we go back to our sandwich-making example, assembly language would be like giving instructions to a robot using a combination of pictures, symbols, and simple words. For example:

MOV R1, PB ; Move peanut butter to R1

MOV R2, J ; Move jelly to R2

SPREAD R1 ; Spread peanut butter

SPREAD R2 ; Spread jelly

PUT\_SLICES\_TG ; Put slices together

CUT\_HALVES ; Cut into halves

In assembly language, we use these instructions to control the computer's hardware directly. While it's a bit more complicated than high-level languages, it provides more control and optimization.

Now, let's talk about 0's and 1's, also known as binary code. Computers work with electronic circuits that can be in two states: on and off. We represent these states using 0's and 1's. Think of 0 as off and 1 as on.

Imagine you have a light bulb. When it's off, we represent it as 0, and when it's on, we represent it as 1. Computers use this binary system to represent all the information, instructions, and data.

So when you write a program in a high-level language or assembly language, it eventually gets translated into a series of 0's and 1's, which is known as machine code. This machine code is what the computer's hardware understands and executes.

To summarize, a high-level language is a more human-readable language that makes programming easier, assembly language is a low-level language closer to machine code, and 0's and 1's represent the on and off states of electronic circuits that computers use to process information.

Q. what are scripting languages and what is Ecma Script standard?

Scripting languages are a type of programming language that are often used for scripting or automating tasks. They are designed to be easy to write and understand, allowing you to quickly create small programs or scripts to perform specific actions.

Imagine you have a magic wand that can make things happen. A scripting language is like a set of spells or instructions that you can use to make your computer or software do things for you. It's a convenient way to automate repetitive tasks or add functionality to existing programs.

For example, let's say you want to create a script to automatically resize and rename all the photos in a folder. With a scripting language, you can write a few lines of code to tell your computer how to do that, and it will perform the task for you without manually resizing and renaming each photo.

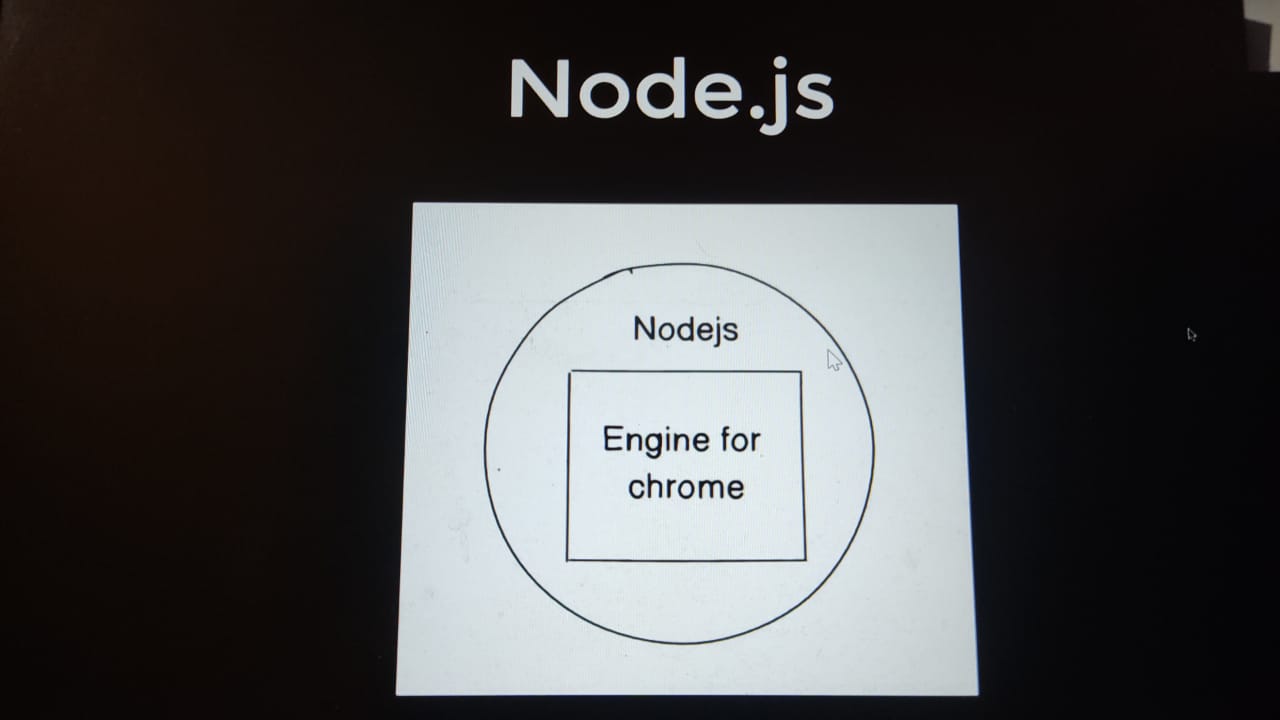
Now, let's talk about ECMAScript (ES) standard. ECMAScript is the official specification for scripting languages, most notably JavaScript. It defines the rules, syntax, and features that a scripting language should follow.

Think of ECMAScript as a set of guidelines or a rulebook that programming languages like JavaScript use. It ensures that different implementations of the language, such as browsers or JavaScript engines, work consistently and understand the code in the same way.

The ECMAScript standard is maintained by the Ecma International organization, and they release new versions periodically to introduce new features and improvements to the scripting language. These versions are commonly referred to as ECMAScript editions, such as ECMAScript 5, ECMAScript 6, ECMAScript 2015, ECMAScript 2018, and so on.

For example, if you're writing JavaScript code, you're essentially using the ECMAScript standard. JavaScript is the most popular implementation of ECMAScript and is supported by web browsers, allowing you to write interactive and dynamic scripts for websites.

To summarize, scripting languages are used for automating tasks or adding functionality to programs, and ECMAScript is the official standard for scripting languages like JavaScript, providing rules and guidelines for their implementation.



Q. what is Node.js?

-> it is a runtime environment not a programming language.

Node.js is a powerful open-source runtime environment that allows you to run JavaScript code outside of a web browser. It's like having a JavaScript engine on your computer/server that can execute JavaScript code directly. Node.js enables you to build server-side applications, command-line tools, and even desktop applications using JavaScript.

Traditionally, JavaScript was mainly used for client-side scripting in web browsers to make websites interactive. However, with Node.js, JavaScript can also be used on the server-side, which means you can use it to create the backend (server) part of web applications.

Think of Node.js as a backstage worker behind a theater play. It handles all the behind-the-scenes tasks necessary to make a performance run smoothly. It can process requests, handle databases, read and write files, and perform other server-side operations.

One of the key advantages of Node.js is its event-driven, non-blocking architecture. This means that it can handle many concurrent connections without getting blocked, allowing it to efficiently handle a large number of requests. This makes Node.js well-suited for applications that require high scalability and real-time communication.

Node.js also has a vast ecosystem of modules and packages available through the npm (Node Package Manager) registry. These modules provide ready-to-use functionality, such as connecting to databases, handling web requests, and implementing security features, saving developers time and effort.

To summarize, Node.js is a runtime environment that allows you to run JavaScript on the server-side, enabling the development of scalable and efficient web applications. It expands the use of JavaScript beyond the browser and provides access to a rich ecosystem of modules and packages.

Q. Popular backend programming language?

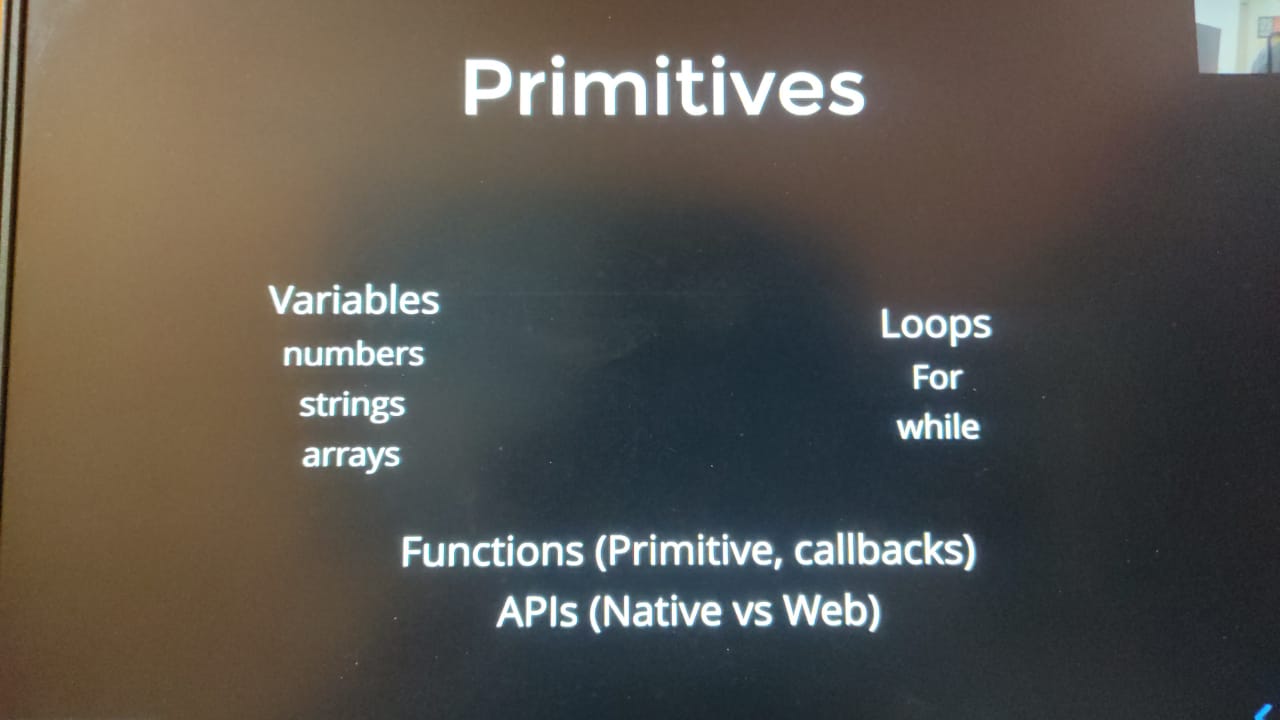
C++

Java

Javascript

Golang

Rust



Q. What are primitives in javascript?

In JavaScript, primitives are the most basic and fundamental data types. They are immutable, which means their values cannot be changed once they are created. JavaScript has six primitive data types:

1. **Number**: Represents numeric values, such as 42 or 3.14. Numbers can be integers or floating-point values.
2. **String**: Represents a sequence of characters enclosed in single quotes ('') or double quotes (""). For example, "Hello, world!" is a string.
3. **Boolean**: Represents a logical value of either true or false. Booleans are often used for making decisions in programming.
4. **Null**: Represents the intentional absence of any object value. It is a special value that indicates the absence of an assigned value.
5. **Undefined**: Represents a variable that has been declared but not assigned a value. It is the default value of variables that have not been initialized.
6. **Symbol**: Introduced in ECMAScript 2015, symbols are unique and immutable values that can be used as property keys in objects. Symbols are often used to create non-enumerable properties or to define well-known symbols.

These primitive data types in JavaScript are considered atomic, meaning they are indivisible and do not have any methods or properties of their own. However, JavaScript provides object wrappers (**Number**, **String**, **Boolean**) that temporarily wrap the primitive values and provide additional methods and properties when needed.

For example, you can use methods like **toUpperCase()** or **length** on a string value by temporarily wrapping it with a **String** object.

Here's an example to demonstrate the primitive types:

let numberValue = 42;

let stringValue = "Hello, world!";

let booleanValue = true;

let nullValue = null;

let undefinedValue = undefined;

let symbolValue = Symbol("mySymbol");

console.log(typeof numberValue); // "number"

console.log(typeof stringValue); // "string"

console.log(typeof booleanValue); // "boolean"

console.log(typeof nullValue); // "object" (a known quirk in JavaScript)

console.log(typeof undefinedValue); // "undefined"

console.log(typeof symbolValue); // "symbol"

Remember, primitives are the building blocks of JavaScript and are used to represent basic data in the language.

Q. what are objects in javascript?

Objects are basically known as key value pair.

In JavaScript, objects are complex data types that allow you to store and organize related data and functionality. Objects are a fundamental part of the language and are used extensively in JavaScript programming.

An object in JavaScript is like a container that can hold multiple values, called properties, and functions, called methods. Properties represent characteristics or attributes of an object, while methods define the actions or behaviors that the object can perform.

To understand objects, let's use an analogy: think of an object as a car. A car has various properties such as its color, model, and number of doors. It also has methods like starting the engine, accelerating, and braking.

In JavaScript, you can create an object using object literal notation, which involves wrapping the properties and methods within curly braces {}. Here's an example:

let car = {

color: "blue",

model: "sedan",

doors: 4,

startEngine: function() {

console.log("Engine started!");

},

accelerate: function() {

console.log("The car is accelerating.");

},

brake: function() {

console.log("Brakes applied!");

}

};

// Accessing object properties

console.log(car.color); // "blue"

console.log(car.model); // "sedan"

console.log(car.doors); // 4

// Calling object methods

car.startEngine(); // "Engine started!"

car.accelerate(); // "The car is accelerating."

car.brake(); // "Brakes applied!"

In this example, the **car** object has properties such as **color**, **model**, and **doors**, which can be accessed using dot notation (**car.color**, **car.model**, etc.). It also has methods like **startEngine**, **accelerate**, and **brake**, which can be invoked by using parentheses after the method name (**car.startEngine()**).

JavaScript objects provide a powerful way to model real-world entities, organize data, and encapsulate related functionality. They are extensively used in JavaScript applications, including web development, to represent and manipulate complex data structures.

It's important to note that JavaScript objects are dynamic, which means you can add, modify, or delete properties and methods even after the object is created. This flexibility allows for great flexibility and adaptability in your code.