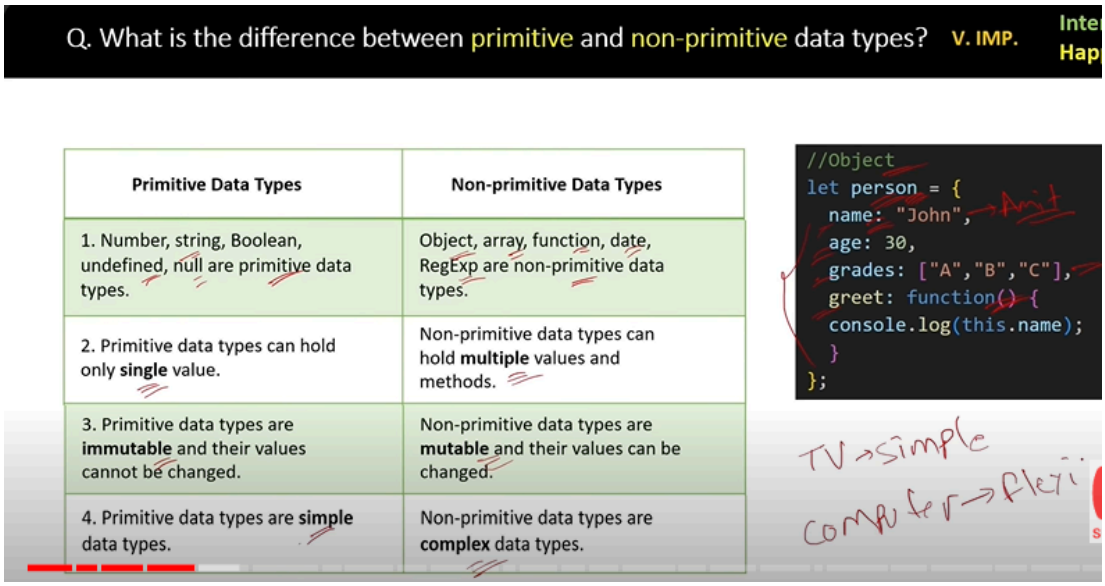
1) What is Javascript?

It is a programming language that is used for converting static web pages to interactive and dynamic web pages.

2) What are data types in JavaScript? What are primitive data types?

A datatype determines What type of values that can be stored in a variable. In primitive datatypes, the data assigned to a variable will never change. They can hold only single value.

3) primitive vs non primitive datatypes:



4) What is array, function and objects?

An array is a collection of values.
A function is a block of code that performs a specific task and returns a value
An object is a non primitive data type that can hold multiple values or a combination of values and functions.

5) What is scope?

Local scope: variable declared inside the body of a function
Global scope: variable declared outside the body of the function

6) difference between var, let and const?

var creates a function-scoped variable. Available inside the function.
let creates a block-scoped variable . available inside the block only.
Const can be defined only once and its value cannot be changed.

7) What is loop? What are the types of loops in JavaScript?

Set of conditions that run until the specified condition is met. Types of loops:
For, while, do-while, for-of, for-in

8) for-of vs for-in?

For-of:
used to loop through the values of an object like arrays, string.
Allows you to access each value directly without having to use an index.

For-in:
used to loop through the properties of an object.
Allows you to iterate over the keys of an object and access the values associated with those keys.

9) What is forEach method?

It is a method available for both arrays and objects that allows you to iterate over each element of the array and perform some action on the element.

reduce(): used when you want to derive a single element from multiple elements in an array

map() used to transform the elements.

10) What is the difference between == and === in JavaScript?

== performs type coercion and converts the string to a number and then compare.
=== does not perform type coercion and only set to true if value and types both are same.

11) What are named and anonymous functions in JavaScript?

Named: function with a name
Anonymous: function used without a name

12) What is function expression in javascript?

It is a way to define a function in JavaScript by assigning it to a variable.

13) What are arrow functions in JavaScript?

It is a concise syntax for definition functions in JavaScript.

14) What are callback functions?

It is function that is passed as an argument in another function. They are used in higher ordered functions, promises, iteration , event handling.

15) What is the use of event handling in JavaScript?

It is the process of responding to user actions in a web page.

16) What is higher order function in JavaScript?

It takes one or more functions as arguments, or returns a function as a result.

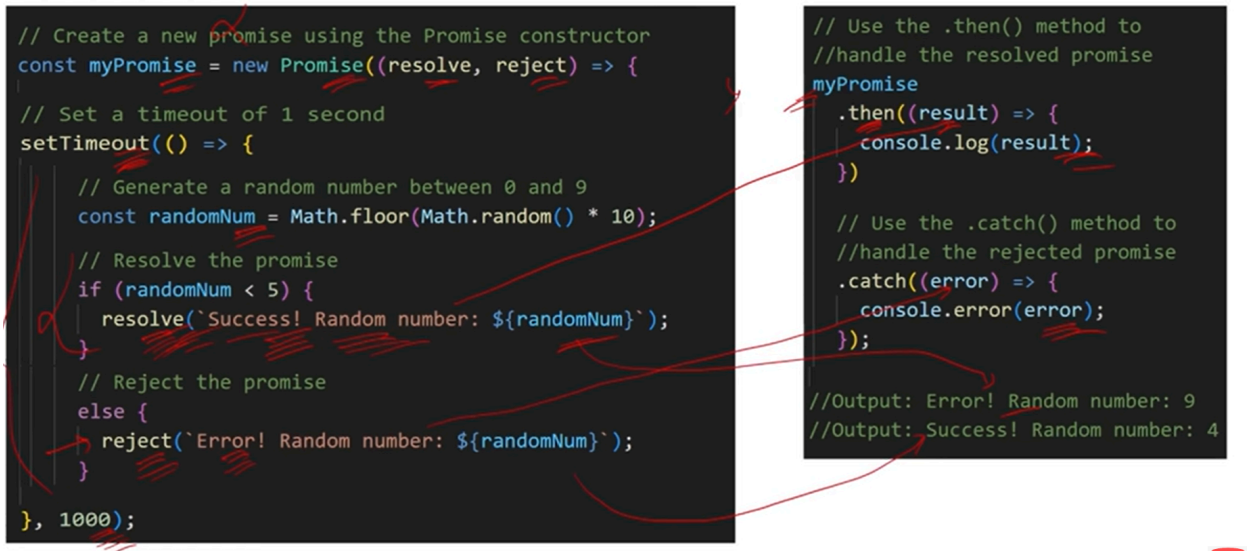
17) What are asynchronous operations in JavaScript?

They are the operations that do not block the execution of the code.
setTimeout is the asynchronous function which executes the callback function after a specific period of time. They can be used in uploading files, promises, animations and transitions etc.

18) What are promises in JavaScript?

They are a way to handle asynchronous operations.
They represent a value that may not be available yet but will be available at some point in the future.
Promises can be in one of the 3 states: pending, resolved or rejected

19) how to implement promises in JavaScript?



```
// Create a new promise using the Promise constructor
const myPromise = new Promise((resolve, reject) => {

// Set a timeout of 1 second
setTimeout(() => {
    // Generate a random number between 0 and 9
    const randomNum = Math.floor(Math.random() * 10);

    // Resolve the promise
    if (randomNum < 5) {
      resolve(`Success! Random number: ${randomNum}`);
    }
    // Reject the promise
    else {
      reject(`Error! Random number: ${randomNum}`);
    }

}, 1000);
```

```
// Use the .then() method to
//handle the resolved promise
myPromise
  .then((result) => {
    console.log(result);
  })

  // Use the .catch() method to
  //handle the rejected promise
  .catch((error) => {
    console.error(error);
  });

//Output: Error! Random number: 9
//Output: Success! Random number: 4
```

20) When to use promises?

Can be used in api calls, file handling, data fetching, event handling, animations and visual effects.

21) What are classes and objects in JavaScript?

Class is a template for creating objects with similar properties and methods.
Objects are the instances of the class that are created using the new keyword.

22) What is the purpose of this keyword in JavaScript?

This refers to the current context or scope in which code is being executed.

23) What is hoisting in JavaScript?

It is a JavaScript behavior where functions and variable declarations are moved to the top of their respective scopes during the compilation phase.

- **OOJS (Object-Oriented JavaScript):** OOJS is a programming paradigm that uses objects and classes to structure code. In JavaScript, objects are collections of key-value pairs, and classes (introduced in ES6) provide blueprints for creating objects with shared properties and methods.

- **Memory Management:** Memory management in JavaScript involves allocating memory for variables and objects and deallocating memory when they are no longer needed. JavaScript engines use techniques like garbage collection to reclaim memory occupied by objects that are no longer reachable.

- **Event Loop:** The event loop is a fundamental concept in JavaScript's concurrency model, responsible for managing asynchronous operations. It continuously checks the call stack and the callback queue, executing tasks in the queue when the call stack is empty, ensuring non-blocking behavior.

- **TDZ (Temporal Dead Zone):** The Temporal Dead Zone is a phase during variable instantiation where accessing the variable results in a ReferenceError. It occurs when trying to access a variable before it's declared with `let` or `const` within its lexical scope.

- **Closures:** Closures occur when a function accesses variables from its outer scope, even after the outer function has finished executing. They allow for maintaining state and creating private variables in JavaScript.

- **Inheritance (Prototypes):** In JavaScript, inheritance is achieved through prototypes, where objects can inherit properties and methods from other objects. When a property or method is accessed on an object, JavaScript looks up the prototype chain to find it.

- **Generator:** Generators in JavaScript are functions that can pause and resume their execution. They're defined using function* syntax and yield values one at a time, allowing for lazy evaluation and simplifying asynchronous code with async/await.

- **ES6-ES14:** ES6 (ECMAScript 2015) introduced significant enhancements to JavaScript, including arrow functions, classes, template literals, and destructuring. ES14 refers to the most recent ECMAScript version, which continues to evolve the language with new features and improvements.

- **Currying:** Currying is a functional programming technique where a function with multiple arguments is transformed into a series of functions, each taking a single argument. It enables partial application of functions and allows for creating more modular and reusable code.

- **DOM (Document Object Model):** The DOM is a programming interface for web documents, providing a structured representation of the document as a tree of nodes. It allows JavaScript to interact with HTML elements dynamically, modifying their content, structure, and styles.

- **BOM (Browser Object Model)**: The BOM is a set of objects provided by web browsers to interact with the browser environment. It includes objects like window, navigator, and location, which enable JavaScript to control browser behavior and access browser-related information.

- **Async/Await:** Async/await is a modern JavaScript feature for handling asynchronous code in a synchronous-like manner. It allows functions to pause execution until an asynchronous operation completes, making asynchronous code easier to read and write compared to traditional callback-based approaches.

- **Callback Hell:** Callback hell refers to the nesting of multiple asynchronous callback functions, leading to code that is hard to read, understand, and maintain. It often occurs in complex asynchronous operations and can be mitigated by using techniques like Promises, async/await, or modularization.