Class 1 - Introduction to Javascript

Why JS?

1. Static vs dynamic websites
2. JS was initially created to make web pages alive / interactive
3. why we need
    a. Interactivity
    b. Dynamic
    c. Adding logic
4. using html / css we would be able to create website which are static - same for all users.
5. Initially JS was meant to be run only in browsers but that is not the case now

The Role of ECMAScript Standardization:

a. JavaScript is governed by specifications called as ECMAScript
b. Standardization: JavaScript's standardization as ECMAScript by the ECMA International organization provided a consistent guideline for browser vendors to implement JavaScript support, ensuring interoperability across different browsers.
c. https://github.com/tc39/ecma262

How is the JS code run ?

1. For any programming language to be able to be understood by machines there is another program that coverts it into machine understandable code
2. JavaScript can execute not only in browser but on any device that has a JavaScript engine
3. JS engine converts the JS code into machine understandable code.
4. Some popular engines
    a. V8 used in Chrome (https://github.com/v8/v8)
    b. Chakra for IE
    c. SpiderMonkey in Firefox

Running JS code

1. Create intro.html
2. just like style tag which goes in head, we have script tag for JS which goes at the end of body tag.
3. Why
    a. Short answer - It helps in reducing the perceived load time of the page because the visual content of the page (HTML and CSS) is displayed to the user before the JavaScript starts executing.
4. The last thing befire closing body tag is generally the script tag where the JS code goes

```
<body>

   <script>
```

```
       console.log('Hello World')
   </script>

</body>
```

5. Just like we had external css, we also have external script
6. Create script.js file

```
<body>

   <script>
       console.log('Hello World')
   </script>
   <script src="./index.js"></script>

</body>
```

Running JS in VS code

1. Now the limitation of JS was that it could run only in browser
2. Basically it is the JS engine that runs the script
3. So a guy named Ryan Dahl said that if JS is so powerful, why limit it to only browser. So he took the V8 JS engine, wrapped it in a an environment and created Node Js
4. So now we could install node js in our local system and do JS development
5. Download nodejs
6. Verify by opening the terminal
7. Node –version

8. Video if facing an issue esp windows users - https://www.youtube.com/watch?v=sQXWVrb52kw
9. Type node in terminal
    a. It opens up the node run time environment
    b. Type in terminal
    c. A = 5, b= 10, console.log(a + b)
10. Running from VS code
    a. We have terminal integrated in VS Code as well
    b. Cd into the folder
    c. node index.js to run the js file

Variables

a. Variables are named storage for data
b. var, let , const
c. let message = ‘Hello!’
d. String hello is saved in memory and we reference it using variable
e.

f. Declare variables

```
console.log("hello from external file")
var myFirstVariable = "Hello from my first variable";
console.log(myFirstVariable);
```

g. As you can see, we dont specify the type of variable beforehand. It is figured out by JS at run time
h. We can even assign the same variable to another data type

```
console.log("hello from external file")
var myFirstVariable = "Hello from my first variable";
console.log(myFirstVariable);
myFirstVariable = true
console.log(myFirstVariable);
```

i. This behavior of JS where type of variables changes dynamically is why it is called as dynamically typed language

Redeclaration in JS

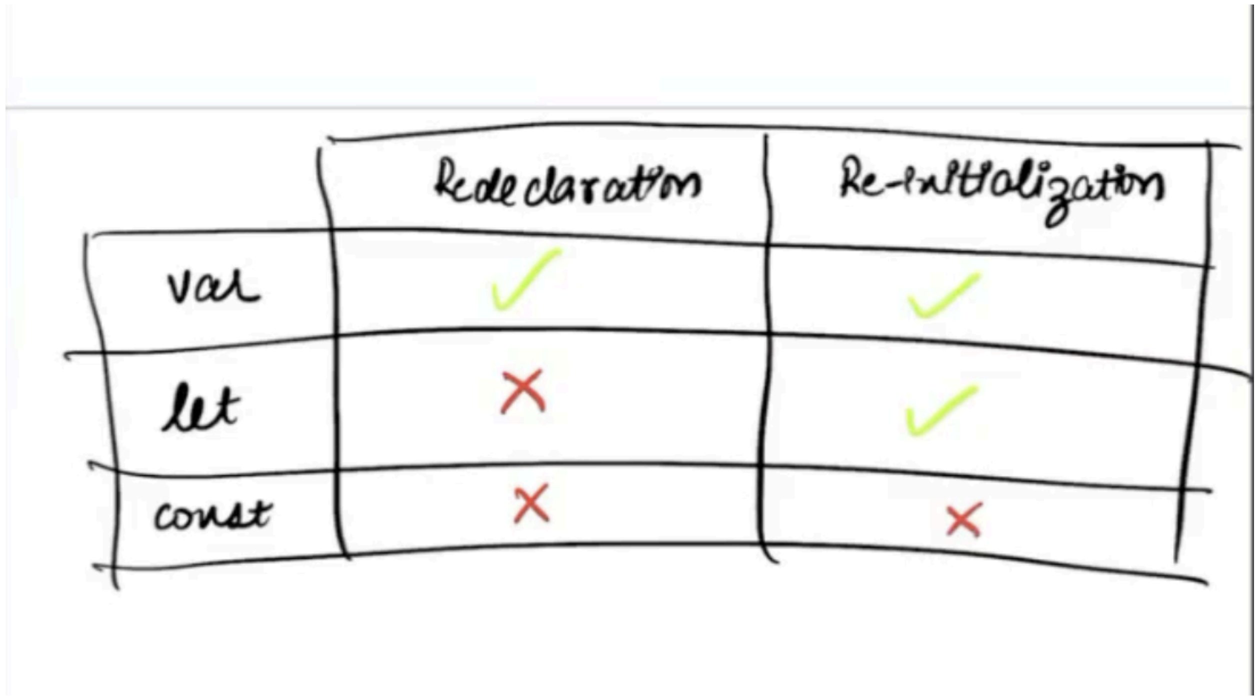
1. Declare variable of the same name and try to log

```
console.log("hello from external file")
var myFirstVariable = "Hello from my first variable";
console.log(myFirstVariable);
myFirstVariable = true
console.log(myFirstVariable);
var myFirstVariable = "another valriable"
console.log(myFirstVariable);
```

2. Summary - var keyword allows to redeclare and reinitialise the variables

Let and const

1. In the ES6 versions, we introduces two another ways to declare variables - let and const
2. This was done to contain some of the issues that happened because of too much flexibility with var
3. There is an issue related to scope behavior of var keywords which we will see later
4. All of this was tried to be addressed using let and const



	Redeclaration	Re-initialization
var	✓	✓
let	✗	✓
const	✗	✗

5. Var examples

```
var a // delcare a variable
a = 5 // initialize a variable
var b = 5 // declare and initialize a variable
var a = "hello" // re-declare a variable
```

6. Let examples

```
let val = "hello"
```

```
val = "bye" // re-assign a variable allowed
console.log(val)
let val = "some other val" // re-declare a variable not
allowed
```

7. Const behavior

```
const pi = 3.14
pi = 3.15 // re-assign a variable not allowed
   console.log(pi)
```

   a. For const variables, declarations and assignment needs to be done in one line

Data Types

1. There are 8 basic data types in JavaScript:
   a. number
   b. bigint
   c. string
   d. boolean
   e. null
   f. undefined
   g. symbol
   h. Object

Dynamically Typed

1. Can put any type in a variable
2. A variable can initially hold a number then later a string and then something else
3. let value = ‘hello’
4. value = 1234
5. Variables are not bound to any data type
6. JavaScript infers the value of variable

Number Type

1. let number1 = 12
2. number2 = 12.345
3. Special numeric values: Infinity, NaN
4. console.log(11 / 0) - Infinity
5. console.log (“string” / 100) - NaN
6. Ambani’s net worth ~100bn
```
console.log(100000000000)
console.log(100_000_000_000)
console.log(100e9)
console.log(1e11)
console.log(1e11 + 120000)
```
7.
8. 1 millionth of a second - 1e-6
9. Converting types on the fly

a.

```
console.log("123"+10)// how much money
console.log(+"123")
console.log(+"123"+10)
console.log(+"123$") // NaN
```

BigInt

1. Numbers have 64 bit representation in memory
2. Values can be between -(2^53-1) to +(2^53-1)
3. For values beyond this range BigInt was introduced
4. Represented by appending n towards end 1n 2n

String

1. Values surrounded by quotes
2. 3 types of quotes:
3. Single - ‘Hello’
4. Double - “Hello”
5. Backticks: `Hello`
6. Hotstar cricket audience count
   a. Backticks allow us to embed variables and expressions in string
   b. let activeUsers = 100
   c. const message = `There are ${activeUsers} users online`

7. No special type for single character like in Java or C

Boolean

1. Boolean type has two values
2. true and false
3. Boolean values are obtained as part of comparisons

```
console.log(1 < 2)
console.log(1 > 2)

const val = "hello"
if(val){
   console.log("val is present")
}else{
   console.log("val is not present")
}
```

Null and Undefined

1. Null is a special value that represents ‘nothing’, ‘empty’ or value unknown
    a. null is an assignment value that represents a deliberate non-value (or an empty value). It indicates the absence of any value or object.
2. Undefined: represents value is not assigned
    a. It is a default initial value for unassigned variables.
    b. Typical Use Cases:

i. A variable declared but not assigned any value.
ii. A function that doesn't explicitly return a value will return undefined.
iii. Accessing a non-existing property of an object or an element of an array.

c.

```
var a
console.log(a)
a = 10
```

d. While both null and undefined represent "no value", the choice between them is not interchangeable. undefined is used when a value has not been assigned and is the default state, while null is used to explicitly denote a null or "empty" value.

Reference data types

1. In JavaScript, reference data types refer to objects that store references to the actual data, not the data itself.
2. Examples include objects, arrays, and functions.

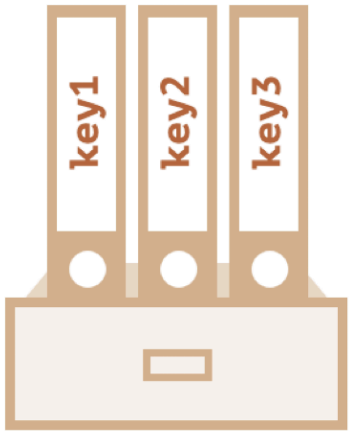
Characteristics of reference data types (extra)

1. Memory Allocation:

    a. Data for reference types is stored in the heap, a larger memory area separate from the stack where primitive data types are stored.
    b. The variable in the stack stores a reference (memory address) to the actual data in the heap.
2. Mutability:
    a. Reference data types are mutable, meaning the data they reference can be modified without changing the reference itself.
    b. For example, adding elements to an array doesn't change the array's reference in memory.
3. Comparison:
    a. When comparing reference types, JavaScript compares the memory address (the reference), not the actual data.
    b. Two objects with identical properties are not equal because their references are different.
4.

Objects

1. Objects are used to store collection of data or more complex entities like user data, product specifications, etc.



2.

```
let user = new Object( ) // object constructor syntax
let anotherUser = { } // object literal syntax
console.log(user)
console.log(anotherUser)
```

3. Can directly create objects by specifying properties

```
let user = {
   name:"Kohli",
   age:34
}

// Accessing the properties of an object
console.log(user.name)
console.log(user.age)

// adding a property to an object
user.isTopranked = true
console.log(user)
```

4. Removing a key and computed keys

```
// removing a property from an object
delete user.age

// adding a multi word property
user["man of match"] = true

// For multi word or computed keys dot notation does not
work
console.log(user["man of match"])
const key = 'man of match'
console.log("Computed key: ", user[key])
```

Typeof operator

1. How to check the type of a value ?

```
console.log(typeof undefined)
console.log(typeof 0)
console.log(typeof true)
console.log(typeof '0')
console.log(typeof Symbol('just me'))
console.log(typeof function() {})
console.log(typeof null)
```

Arrays

```
const arr = [1,2,3,4,5]
const arr2 = [1,function(){},"hello",true,"unity in diversity",{}]
```

1. see array by indices
2. array .length
3. What will below print

```
const obj = {
   name:"ayuhs",
   age:34
}
const arr2 = [obj,1,function(){},"hello",true,"unity in
diversity",{}]
console.log(arr2[0].age)
```

Functions

1. What are functions
   a. Set of code that you can execute again and afghan

```
function myFirstFunction(){
   console.log("Hello from my first function")
}

myFirstFunction()
```

2. see function name, arguments, function body, invoking the function
3. Passing arguments

```
function myFirstFunction(name){
   console.log("Hi",name, "Hello from my first function")
}

myFirstFunction("ayush")
```