JAVASCRIPT

History:

There was not much of interactivity. For that, livescirpt was first introduced which was later coupled with web technologies (HTML & CSS). Livescript had limited features. Over the time it got improved.

JavaScript is a dynamic computer programming language. It is lightweight and most commonly used as a part of web pages, whose implementations allow client-side script to interact with the user and make dynamic pages. It is an interpreted programming language with object-oriented capabilities.

Javascript has multiple versions and there are certain flavors in javascript as well. To run python, we need to have a python environment. Similarly, we need to provide a run time environment. Server-side environment as Node.js but the main starting point is the browser itself. Every engine is basically running javascript.

JavaScript was first known as **LiveScript,** but Netscape changed its name to JavaScript, possibly because of the excitement being generated by Java. JavaScript made its first appearance in Netscape 2.0 in 1995 with the name **LiveScript**. The general-purpose core of the language has been embedded in Netscape, Internet Explorer, and other web browsers.

The ECMA-262 Specification defined a standard version of the core JavaScript language.

- JavaScript is a lightweight, interpreted programming language.
- Designed for creating network-centric applications.
- Complementary to and integrated with Java.
- Complementary to and integrated with HTML.
- Open and cross-platform

Advantages of JavaScript

The merits of using JavaScript are −

- **Less server interaction** − You can validate user input before sending the page off to the server. This saves server traffic, which means less load on your server.

- **Immediate feedback to the visitors** − They don't have to wait for a page reload to see if they have forgotten to enter something.
- **Increased interactivity** − You can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.
- **Richer interfaces** − You can use JavaScript to include such items as drag-and-drop components and sliders to give a Rich Interface to your site visitors.

Limitations of JavaScript

We cannot treat JavaScript as a full-fledged programming language. It lacks the following important features −

- Client-side JavaScript does not allow the reading or writing of files. This has been kept for security reason.
- JavaScript cannot be used for networking applications because there is no such support available.
- JavaScript doesn't have any multi-threading or multiprocessor capabilities.

Once again, JavaScript is a lightweight, interpreted programming language that allows you to build interactivity into otherwise static HTML pages.

To run javascript code, we have to put in our html and open that in the browser. In python we use idle. In javascript, the same thing is available as console in browser.

You can write your proper javascript code and the browser will run it, but not for long-time code.

Two ways of integrating javascript in HTML:

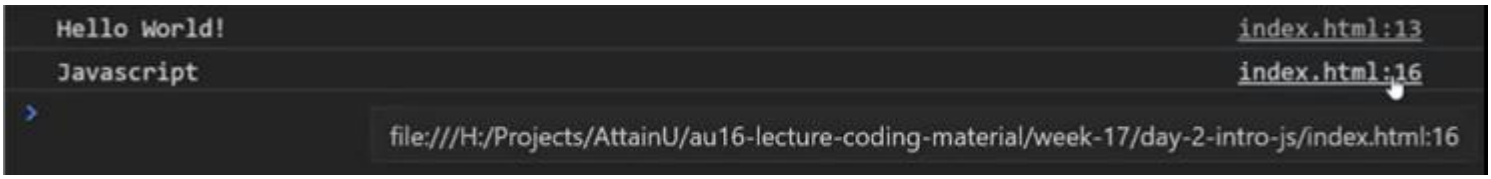
We put the style tag in the head element of the html page and the CSS changes would be visible in the HTML page. Similarly, using the script tag in the head of the HTML, we can run javascript.

Other way, we can put in an external javascript sheet and link it in the html page.

```
<script type="text/javascript">
    console.log('hello world!')
</script>
```

Output:

The output is not visible on the web page for now, but it can be seen on the console tab in the inspect (Ctrl + Shift + I)



When we hover over the index.html:16 it shows the link address as well.

Linking:

If we have multiple pages and multiple functionalities, then we have to copy and paste this in every page will be a tedious task. Instead, we can link. We can link using the script tag.

```
<script type="text/javascript" src="script.js"></script>
```

Now if we put anything in the ***console*.log()** in the script.js file then you it will be visible on the Console tab of the Inspect sheet. If we are using two pages of javascript linked into HTML page, then based on the order you put the files, it would be executed.

Variables:

Variables can be thought of as named containers. You can place data into these containers and then refer to the data simply by naming the container.

Before you use a variable in a JavaScript program, you must declare it. Variables are declared with the **var** keyword as follows.

```
var money = 12.2;
var name = "a";
```

You can also declare multiple variables with the same var keyword as follows −

```
var money, name;
```

Storing a value in a variable is called variable initialization. You can do variable initialization at the time of variable creation or at a later point in time when you need that variable.

For instance, you might create a variable named money and assign the value 2000.50 to it later. For another variable, you can assign a value at the time of initialization as follows.

```
var name = "Attainu";
var money;
money = 2000.50;
```

NOTE − Use the var keyword only for declaration or initialization, once for the life of any variable name in a document. You should not re-declare same variable twice.

JavaScript is untyped language. This means that a JavaScript variable can hold a value of any data type. Unlike many other languages, you don't have to tell JavaScript during variable declaration what type of value the variable will hold. The value type of a variable can change during the execution of a program and JavaScript takes care of it automatically.

JavaScript allows you to work with three **primitive data types** −

- **Numbers**, e.g., 123, 120.50 etc.
- **Strings** of text e.g., "This text string" etc.
- **Boolean** e.g., true or false.

JavaScript also defines two trivial data types, null and undefined, each of which defines only a single value. In addition to these primitive data types, JavaScript supports a composite data type known as **object**. We will cover objects in detail in a separate chapter.

Note − JavaScript does not make a distinction between integer values and floating-point values. All numbers in JavaScript are represented as floating-point values

There is one more, **bigint**, which will be covered in future sessions.

Operators:

Let us take a simple expression 4 + 5 is equal to 9. Here 4 and 5 are called operands and '+' is called the operator. JavaScript supports the following types of operators.

- Arithmetic Operators
- Comparison Operators
- Logical (or Relational) Operators
- Assignment Operators
- Conditional (or ternary) Operators

Arithmetic Operators

JavaScript supports the following arithmetic operators −

Assume variable A holds 10 and variable B holds 20, then −

Sr.No.	Operator & Description
1	+ (Addition) – Adds two operands Ex: A + B will give 30
2	- (Subtraction) – Subtracts the second operand from the first Ex: A - B will give -10
3	* (Multiplication) – Multiply both operands Ex: A * B will give 200
4	/ (Division) – Divide the numerator by the denominator Ex: B / A will give 2
5	% (Modulus) – Outputs the remainder of an integer division Ex: B % A will give 0
6	++ (Increment) – Increases an integer value by one Ex: A++ will give 11
7	-- (Decrement) – Decreases an integer value by one Ex: A-- will give 9

Note − Addition operator (+) works for Numeric as well as Strings. e.g. "a" + 10 will give "a10".

Comparison Operators

JavaScript supports the following comparison operators −

Assume variable A holds 10 and variable B holds 20, then −

Sr.No.	Operator & Description
1	= = (Equal) Checks if the value of two operands are equal or not, if yes, then the condition becomes true. Ex: (A == B) is not true.
2	!= (Not Equal) Checks if the value of two operands are equal or not, if the values are not equal, then the condition becomes true. Ex: (A != B) is true.
3	> (Greater than) Checks if the value of the left operand is greater than the value of the right operand, if yes, then the condition becomes true. Ex: (A > B) is not true.
4	< (Less than) Checks if the value of the left operand is less than the value of the right operand, if yes, then the condition becomes true. Ex: (A < B) is true.
5	>= (Greater than or Equal to) Checks if the value of the left operand is greater than or equal to the value of the right operand, if yes, then the condition becomes true. Ex: (A >= B) is not true.
6	<= (Less than or Equal to) Checks if the value of the left operand is less than or equal to the value of the right operand, if yes, then the condition becomes true. Ex: (A <= B) is true.

Logical Operators

JavaScript supports the following logical operators −

Assume variable A holds 10 and variable B holds 20, then −

Sr.No.	Operator & Description
1	&& (Logical AND) If both the operands are non-zero, then the condition becomes true. Ex: (A && B) is true.
2	|| (Logical OR) If any of the two operands are non-zero, then the condition becomes true. Ex: (A || B) is true.
3	! (Logical NOT) Reverses the logical state of its operand. If a condition is true, then the Logical NOT operator will make it false. Ex: ! (A && B) is false.

Miscellaneous Operator

We will discuss two operators here that are quite useful in JavaScript: the conditional operator (? :) and the typeof operator. Conditional Operator (? :)

The conditional operator first evaluates an expression for a true or false value and then executes one of the two given statements depending upon the result of the evaluation.

Sr.No.	Operator and Description
1	? : (Conditional) If Condition is true? Then value X : Otherwise value Y

Bitwise Operators

JavaScript supports the following bitwise operators −

Assume variable A holds 2 and variable B holds 3, then −

Sr.No.	Operator & Description
1	& (Bitwise AND) It performs a Boolean AND operation on each bit of its integer arguments – Ex: (A & B) is 2.
2	| (BitWise OR) It performs a Boolean OR operation on each bit of its integer arguments – Ex: (A | B) is 3.
3	^ (Bitwise XOR) It performs a Boolean exclusive OR operation on each bit of its integer arguments. Exclusive OR means that either operand one is true or operand two is true, but not both – Ex: (A ^ B) is 1
4	~ (Bitwise Not) It is a unary operator and operates by reversing all the bits in the operand – Ex: (~B) is -4.
5	<< (Left Shift) It moves all the bits in its first operand to the left by the number of places specified in the second operand. New bits are filled with zeros. Shifting a value left by one position is equivalent to multiplying it by 2, shifting two positions is equivalent to multiplying by 4, and so on – Ex: (A << 1) is 4.
6	>> (Right Shift) Binary Right Shift Operator. The left operand's value is moved right by the number of bits specified by the right operand – Ex: (A >> 1) is 1
7	>>> (Right shift with Zero) This operator is just like the >> operator, except that the bits shifted in on the left are always zero. Ex: (A >>> 1) is 1.

Assignment Operators

JavaScript supports the following assignment operators −

Sr.No.	Operator & Description
1	= (Simple Assignment) Assigns values from the right side operand to the left side operand Ex: C = A + B will assign the value of A + B into C
2	+= (Add and Assignment) It adds the right operand to the left operand and assigns the result to the left operand. Ex: C += A is equivalent to C = C + A
3	−= (Subtract and Assignment) It subtracts the right operand from the left operand and assigns the result to the left operand. Ex: C -= A is equivalent to C = C - A
4	*= (Multiply and Assignment) It multiplies the right operand with the left operand and assigns the result to the left operand. Ex: C *= A is equivalent to C = C * A
5	/= (Divide and Assignment) It divides the left operand with the right operand and assigns the result to the left operand. Ex: C /= A is equivalent to C = C / A
6	%= (Modules and Assignment) It takes modulus using two operands and assigns the result to the left operand. Ex: C %= A is equivalent to C = C % A

Note − Same logic applies to Bitwise operators so they will become like <<=, >>=, >>=, &=, |= and ^=.

‘***typeof***’ Operator

The ***typeof*** operator is a unary operator that is placed before its single operand, which can be of any type. Its value is a string indicating the data type of the operand.

The ***typeof*** operator evaluates to "number", "string", or "boolean" if its operand is a number, string, or boolean value and returns true or false based on the evaluation.

Here is a list of the return values for the ***typeof*** Operator.

Type	String Returned by ***typeof***
Number	"number"
String	"string"
Boolean	"boolean"
Object	"object"
Function	"function"
Undefined	"undefined"
Null	"object"

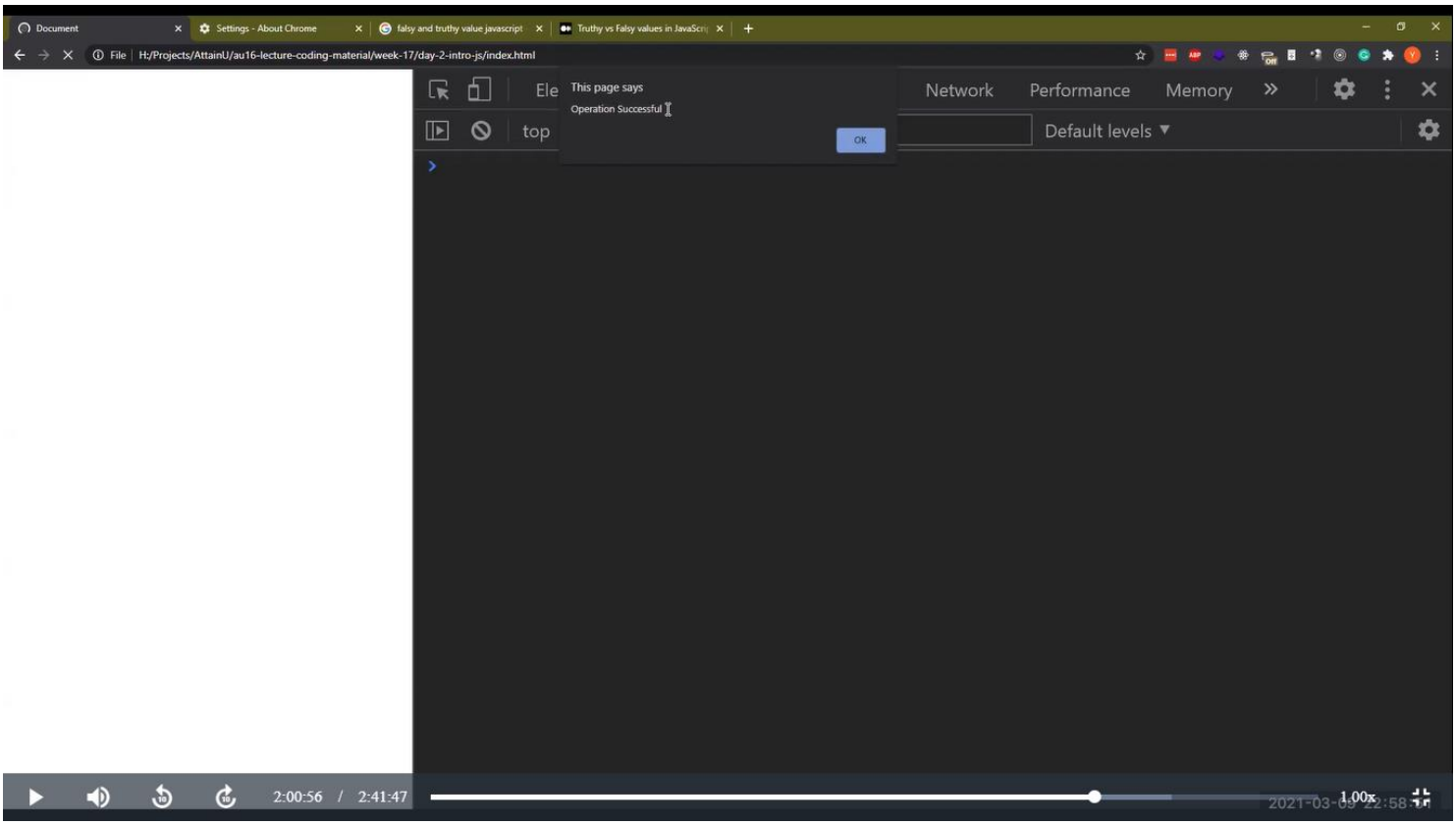
Function Calls:

Alert Boxes:

There is an alert() function, which javascript provides us.

```
alert('operation successful')
```

this will get executed immediately.



`confirm()`

When the program is executed, it would first execute the alert() function first and then the confirm() function.

3 function calls:

- **`alert()`**
- **`confirm()`**
- **`prompt()`**