Client Side Scripting

1. JavaScript is a dynamic computer programming language.
2. It is used to create client side **dynamic** web pages/applications.

1. JavaScript was invented by Brendan Eich in 1995.
2. JavaScript was originally named Mocha and changed to Livescript but ultimately became

3. 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.
4. It is an interpreted programming language with object-oriented capabilities.
5. JavaScript made its first appearance in Netscape 2.0 in 1995 with the name LiveScript.
6. Open and cross-platform.
7. It is an interpreted language. i.e. The code executes without preliminary compilation.
8. All major web browsers such as Chrome, Mozilla Firefox, safari, internet explorer, edge, etc, supports javascript.
9. JavaScript is/can be used to create sophisticated user interfaces(**UI**) and increase the user experience (**UX**) of a web page/application.

JavaScript.

3. It's important to note that JavaScript came before ECMAscript.
4. It was developed for Netscape 2, and became the ECMA-262 standard in 1997.
5. After Netscape handed JavaScript over to

ECMA, the Mozilla foundation continued to develop JavaScript for the Firefox browser. Mozilla's latest version was 1.8.5. (Identical to ES5).

6. ECMAScript is the standard for JavaScript.

1. **An interpreted language:**
JavaScript is an interpreted language, which requires no compilation steps. This provides an easy development process. The syntax is completely interpreted by the browser line by line.
2. **Cross browser support:**
JavaScript codes are supported by number of popular web browsers such as google chrome, mozilla firefox, Safari, Microsoft edge, etc. 3. **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.

4. **Immediate feedback to the visitors:** They don't have to wait for a page reload to see if they have forgotten to enter something.

5. **Increased interactivity:**
You can create interfaces that react when the user hovers over them with a mouse or
activates them via the keyboard.
6. **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.
7. **Easy to Learn:**
By learning just a few commands and simple rules of syntax, complete applications can be built using JavaScript.
8. **Designed for programming user events:** JavaScript supports Object/Event based programming. JavaScript recognizes different events such as click, mouse hover. It can handle such events and perform user defined

tasks when that event occurs.

1. **Client-side Security:**
Since the JavaScript code is viewable to the user, others may use it for malicious purposes. These practices may include using the source code without authentication. Also, it is very easy to place some code into the site that compromises the security of data over the website.
2. **Browser Support:**
The browser interprets JavaScript differently in different browsers. Thus, the code must be run on various platforms before publishing. The older browsers don't support some new functions and we need to check them as well.
3. **Lack of Debugging Facility:**
Though some HTML editors support debugging, it is not as efficient as other editors like C/C++ editors. Also, as the browser doesn't show any error, it is difficult for the developer to detect the problem. 4. **No support for multi-threading:**
JavaScript doesn't have any multithreading or multiprocessor capabilities.
5. **Rendering Stopped:**
A single code error can stop the rendering of the entire JavaScript code on the website. To the user, it looks as if JavaScript was not present.

Coding conventions are the guidelines for

programming. They typically cover:

1. Naming and declaration rules for variables and functions.
2. Rules for the use of white space, indentation, and comments.
3. Programming practices and principles.

Coding conventions secure quality:

1. Improves code readability.
2. Make code maintenance easier

Coding conventions can be documented rules for teams to follow, or just be your individual coding practice.

Variable Names

1. User **camelCase** for identifier (variable names and functions).
2. All names start with a letter.
3. Global variables should be in **UPPER_CASE.**

Example:

```
[redacted] "John"

[redacted] "Doe"

price = 19.90

[redacted] 0.20

[redacted]
```

[redacted] Spaces Around Operators

- Always put spaces around operators (= + - * /), and after commas.
  i.e. `let` [redacted]

File Extensions:

1. HTML files should have a **.html** extension (.htm is allowed).
2. CSS files should have a **.css** extension. 3. JavaScript files should have a **.js** extension.

Use Lower Case File Names:

1. Most web servers (Apache, Unix) are **case sensitive** about file names:

london.jpg cannot be accessed as London.jpg. 2. Other web servers (Microsoft, IIS) are not case sensitive:

london.jpg can be accessed as London.jpg or london.jpg.

Loading JavaScript in HTML

1. JavaScript code can be written inside the <script> tag within the html file.
2. Use simple syntax for loading external scripts (the type attribute is not necessary):

```
<script src="myscript.js"></script>
```

```
<!DOCTYPE html>

<html>

 <head>

<title>Index file</title>  <script>

alert('Hello World!');  </script>

</head>

<body>

</body>

</html>
```

There are multiple ways of adding JavaScript content in the HTML.

The <script> Tag

In HTML, JavaScript code is inserted between **<script>** and **</script>** tags. Example:

```
<script>

document.getElementById("demo").innerHTML =  "My
First JavaScript";

</script>
```

**Note: Old JavaScript examples may use a type attribute: <script type="text/javascript">.*

```
<!DOCTYPE html>

<html>

<head>

<script>

function myFunction() {

 document.getElementById( "demo").innerHTML =
"Paragraph changed." ;

}

</script>

</head>

<body>

<h2>Demo JavaScript in Head </h2>

<p id="demo">A Paragraph</p>

<button type="button"
```

```
onclick="myFunction()"> Try it</button>

</body>

</html>
```

```
<!DOCTYPE html>

<html>

<head>

</head>

<body>

<h2>Demo JavaScript in Head </h2>

<p id="demo">A Paragraph</p>

<button type="button"
```

```
onclick="myFunction()"> Try it</button>

<script>

function myFunction() {

 document.getElementById( "demo").innerHTML =
"Paragraph changed." ;

}

</script>

</body>

</html>
```

Scripts can also be placed in external files:
External file: app.js

```
function
```

```
[redacted]"demo"
"Paragraph changed." [redacted]
```

And inside HTML file we embed as:

```
<!DOCTYPE html>

<html>

<head>

      <script src="app.js"></script>

</head>

<body>

<h2>Demo JavaScript in Head </h2>

<p id="demo">A Paragraph</p>

<button type="button"

onclick="myFunction()"> Try it</button>

</body>

</html>
```

Placing scripts in external files has some advantages:

1. It separates HTML and code.
2. It makes HTML and JavaScript easier to read and maintain.
3. Cached JavaScript files can speed up page loads

*Note: To add several script files to one page - use several script tags:

```
<script src="menu.js"></script>

<script src="body.js"></script>
```

Single Line Comments:

1. Single line comments start with //.
2. Any text between // and the end of the line will be ignored by JavaScript (will not be executed).

3. This example uses a single-line comment before each code line:

```
// Change heading:

                         "myH"                   "My
First Page"
```

Multi-line Comments:

1. Multi-line comments start with /* and end with */. 2. Any text between /* and */ will be ignored by JavaScript. 3. This example uses a multi-line comment (a comment block) to explain the code:

```
/*

The code below will change

the heading with id = "myH"

and the paragraph with id = "myP"

in my web page:

*/

                         "myH"                   "My
First Page"

                         "myP"                   "My
first paragraph."
```

The **<noscript>** tag defines an alternate content to be displayed to users that have disabled scripts in their browser or have a browser that doesn't support script.

The **<noscript>** element can be used in both **<head>**

and **<body>**.

Example:

```
<script>

    document.write("Hello World!")

</script>

 <noscript>[redacted]
[redacted]</noscript>
```

JavaScript operators are used to assign values, compare values, perform arithmetic operations, and more.

JavaScript Arithmetic Operators:

Arithmetic operators are used to perform arithmetic on numbers:

Operator Description

+ Addition

- Subtraction

* Multiplication

** Exponentiation (ES2016) / Division

% Modulus (Division Remainder) ++ Increment

-- Decrement

JavaScript Assignment Operators:

Assignment operators assign values to JavaScript variables.

[redacted]

JavaScript String Operators:

The + operator can also be used to add (concatenate) strings.

Operator

+

Example text1 text2 text3 text3 = text1 + text2 “Good” “Morning” “Good Morning”

+= text1 += text2 “Good Morning” “Morning” “”

JavaScript Comparison Operators:

Comparison operators are used in logical statements to define equality and difference between variables or values. Given that x = 5, the table below explains the comparison operators:

Operator
==

Description Comparing Returns

equal toa x == 8 x == 5 false true

=== equal value and equal type x === ‘5’ false != not equal x != 8 true !== not equal

value or not equal type x !== ‘5’ true > greater than x > 8 false < less than x < 8 true >=

greater than or equal to x >= 4 true <= less than or equal to x <= 4 false

JavaScript Logical Operators:

Logical operators are used to determine the logic between variables and values. Given that x = 6 and y = 3, the table below explains the logical operators:

Operator Description Example

&& logical and (x < 7 && y >= 3) is true || logical or (x == 5 || y == 4) is false ! logical

not !(x == 6) is false

The typeof Operator:

You can use the typeof operator to find the data type of a JavaScript variable

```
typeof "John" // Returns "string"
typeof 3.14 // Returns "number"
typeof NaN // Returns "number"
typeof false // Returns "boolean"
typeof [1,2,3,4] // Returns "object"
typeof {name:'John', age:34} // Returns "object"
typeof new Date() // Returns "object"
typeof function () {} // Returns "function"
typeof myCar // Returns "undefined" * typeof null //
Returns "object".
```

Control structures actually controls the flow of execution of a program. Following are different control structures used in JavaScript:

1. if…else block
2. switch case
3. do while loop
4. while loop
5. for loop

1. if else block:

Conditional statements are used to perform different actions based on different conditions.

```
if
// true condition block
else
// false condition block
```

2. switch case:

The switch statement is used to perform different actions based on different conditions.

Example:

```
switch  new

 default

                  "Looking forward to the Weekend"
```

```
break
case 6
        "Today is Saturday"
break
case 0
             "Today is Sunday"
}
```

3. do-while loop:

The do...while statements creates a loop (around a code block), executes the block once, and repeat code block as long as a condition is true.

The do...while is used when you want to run a code block at least one time.

```
       ""
     0
do
                 "<br>"

}
while      5
```

4. while loop:

The while loop loops through a block of code as long as a specified condition is true.

i = 0;

```
while        10
```

```
         "The number is "
```

5. for Loop:

for loop is similar to other loops and is used to iterate over certain number of times.

```
for       0      5
```

```
<!DOCTYPE html>
<html>
```

```
         "The number is "          "<br>"
```

1. A JavaScript function is a block of code designed to perform a particular task.
2. A JavaScript function is executed when "something" invokes it (calls it).

Syntax of a function:

```
function

 // code to be executed
```

Example:

```
<body>
 <h2>[illegible]</h2>
 <p>[illegible]
[illegible]</p>
 <p id "demo"></p>
 <script>
 function product num1  num2
 return num1   num2

 document getElementById "demo"  innerHTML   product 4   3
</script>
</body>
</html>
<!DOCTYPE html>
<html>
<body>
```

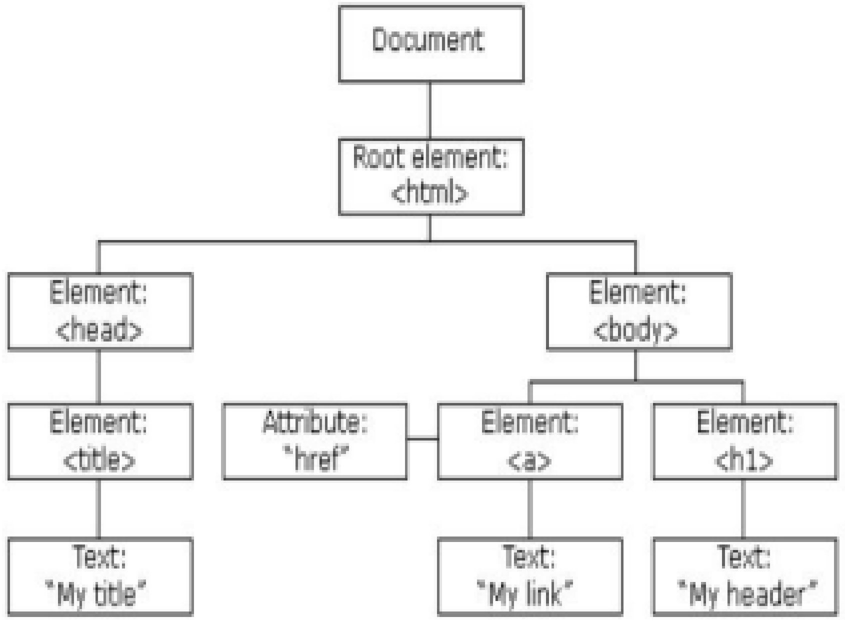
```
<h2>                    </h2>
<p>                                        </p> <button
onclick "product(4,3)">            </button> <p
id "demo"></p>
<script>
function product num1  num2
             total = num1   num2
    document getElementById "demo"  innerHTML   total
</script>
</body>
</html>
```

1. The DOM is a W3C (World Wide Web Consortium) standard.
2. The DOM defines a standard for accessing documents: *"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."*
3. When a web page is loaded, the browser creates a Document Object Model of the page.

4. The HTML DOM model is constructed as a tree of Objects.
5. A document object represents the HTML document that is displayed in that window. The document object has various properties that refer to other objects which will allow access to modify document content.
6. Window object:- Top of the hierarchy. It is the outmost element of the object hierarchy.
7. Document object:- Each HTML document that gets loaded into a window becomes a document object. The document contains the content of the page.
8. Form object:- Everything enclosed in the <form>...</form> tags sets the form object.

The HTML DOM model is constructed as a tree of Objects:

The HTML DOM Tree of Objects



The HTML DOM is a standard object model and programming interface for HTML. It defines:

1. The HTML elements as objects
2. The properties of all HTML elements 3. The methods to access all HTML elements 4. The events for all HTML elements

In other words: **The HTML DOM is a standard for how to get, change, add, or delete HTML elements.**

Why DOM is required in JavaScript?

1. JavaScript can change all the HTML elements in the page
2. JavaScript can change all the HTML attributes in the page
3. JavaScript can change all the CSS styles in the page
4. JavaScript can remove existing HTML elements and attributes
5. JavaScript can add new HTML elements and attributes
6. JavaScript can react to all existing HTML events in the page
7. JavaScript can create new HTML events in the page

The DOM Programming Interface

1. The HTML DOM can be accessed with JavaScript (and with other programming languages).
2. In the DOM, all HTML elements are defined as objects.

3. The programming interface is the properties and methods of each object.
4. A property is a value that you can get or set (like changing the content of an HTML element). 5. A method is an action you can do (like add or deleting an HTML element).

Example

The following example changes the content (the **innerHTML**) of the **<p>** element with **id="demo"**:

```
<!DOCTYPE html>
<html>
<body>
 <p id "demo"></p>
 <script>
 document getElementById "demo"  innerHTML   "Hello World!"
</script>
</body>
</html>
```

In the example above, **getElementById** is a method, while **innerHTML** is a property.

Finding HTML elements:

Method Description document.getElementById('id') Find an element by an id

document.getElementsByTagName('tag-name') Find elements by tag name

document.getElementsByClassName('class-name') Find elements by class name
Changing HTML elements

Property Description

element.innerHTML Changes the inner HTML of an element

element.attribute Change the attribute value of an HTML element

Element.style.property = new style Change the style of an HTML element

element.setAttribute('attributeName', 'attribute-value') Change the attribute value of an HTML element.

Adding and Deleting elements

Method Description document.createElement(element) Create an HTML

element document.removeChild(element) Remove an HTMl element

document.appendChild(element) Add an HTML element

document.replaceChild(new, old) Replace an HTML element

document.write(text) Write into HTML output stream

1. HTML events are "**things**" that happen to HTML elements.
2. When JavaScript is used in HTML pages, JavaScript can "react" on these events. 3. An HTML event can be something a browser does or an event does.

4. Here are some examples of HTML events: a. A HTML webpage has finished loading. b. A HTML input field was changed.
c. An HTML button was clicked.
d. Mouse over to HTML components

Event handlers can be used to handle and verify user input, user actions, and browser actions:

- Things that should be done every time a page loads
- Things that should be done when the page is closed
- Action that should be performed when a user clicks a button
- Content that should be verified when a user inputs data
- And more ...

Event	**Description**
onchange	An HTML element has been changed
onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element
onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page

JavaScript has several built-in or core language objects. These built-in objects are available regardless of window content and operate independently of whatever page your browser has loaded.

The different types of built-in objects available in JavaScript are as follows:

1. Array
2. Date
3. String
4. Math
5. Boolean
6. Number
7. RegExp

1. Multiple values are stored in a single value. 2. In JavaScript, an array can hold different type of data types in a single slot, which implies that an array can have a string or a number or an object in a single slot.

An array object can be created using following ways:

Using array constructors

Creating an empty array:

```
myArray = new [redacted]
```

Creating an array of given size:

```
myArray = new [redacted] size [redacted]
```

Creating array with elements:

```
myArray = new [redacted] 'Manoj' [redacted] 'Ritush' [redacted]
'Suraj' [redacted] 'Surakshya' [redacted]
```

Using array literal notation: To create an empty array:

```
myArray = [];
```

To create an array with elements:

```
myArray = ['Aayush'  'Niraj'  'Ekin'
'Samita'  'Reena'
```

Sr.No.	Method & Description
1	concat() Returns a new array comprised of this array joined with other array(s) and/or value(s).
2	every()

Returns true if every element in this array satisfies the provided testing function.

3 filter()

Creates a new array with all of the elements of this array for which the provided filtering function returns true.

4 forEach()

Calls a function for each element in the array.

5
6

indexOf()

Returns the first (least) index of an element within the array equal to the specified value, or -1 if none is found.

join()

Joins all elements of an array into a string.

7 lastIndexOf()

Returns the last (greatest) index of an element within the array equal to the specified value, or -1 if none is found.

8 map()

Creates a new array with the results of calling a provided function on every element in this array.

9 pop()

Removes the last element from an array and returns that element.

10 push()

Adds one or more elements to the end of an array and returns the new length of the array.

11 reduce()

Apply a function simultaneously against two values of the array (from left-to-right) as to reduce it to a single value.

12 reduceRight()

Apply a function simultaneously against two values of the array (from right-to-left) as to reduce it to a single value.

13 reverse()

Reverses the order of the elements of an array -- the first becomes the last, and the last becomes the first.

14	shift() Removes the first element from an array and returns that element.
15	slice() Extracts a section of an array and

returns a new array.

16 some()

Returns true if at least one element in this array satisfies the provided testing function.

17 toSource()

Represents the source code of an object

18 sort()

Sorts the elements of an array

19 splice()

Adds and/or removes
elements from an array.

20 toString()

Returns a string representing the array and its elements.

21 unshift()

Adds one or more elements to the front of an array and returns the new length of the array.

1. The Date object is a datatype built into the JavaScript language.
2. Date objects are created with the new Date( ) as shown below.

```
        new
```

3. Once a Date object is created, a number of methods allow you to operate on it.
4. Most methods simply allow you to get and set the year, month, day, hour, minute, second, and millisecond fields of the object, using either local time or UTC (universal, or GMT) time.

Date Methods:

Method Description

getFullYear() Get the year as a four digit number (yyyy) getMonth()

Get the month as a number (0-11) getDate() Get the day as a number (1-31)

getHours() Get the hour (0-23)

getMinutes() Get the minute (0-59)

getSeconds() Get the second (0-59)

getMilliseconds() Get the millisecond (0-999)

getTime() Get the time (milliseconds since January 1, 1970)

getDay() Get the weekday as a number (0-6) Date.now() Get

the time. ECMAScript 5.

Method Description

setDate() Set the day as a number (1-31) setFullYear() Set the year (optionally

month and day) setHours() Set the hour (0-23)

setMilliseconds() Set the milliseconds (0-999) setMinutes() Set the minutes (0-59) setMonth() Set the month (0-11)

setSeconds() Set the seconds (0-59)

setTime() Set the time (milliseconds since January 1, 1970)

https://www.tutorialspoint.com/javascript/javascript_date_object.htm

A JavaScript string stores a series of characters. A string can be any text inside double or single quotes:

```
let [redacted] "Volvo XC60"

let [redacted] 'Volvo XC60'
```

New string can also be created using the string object:

```
let carName3 = new String("BMW");
```

String Methods

Method
[redacted]

[redacted]
[redacted]
[redacted]

Description

Returns the character at a specified index

(position) Returns the Unicode of the

character at a specified index Returns two

or more joined strings

Returns if a string ends with a specified value

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

Returns a new string with a number of copies of a string

Searches a string for a value, or a regular expression, and returns a string where the values are replaced

Searches a string for a value, or regular expression, and returns the index (position) of the match

Extracts a part of a string and returns a new string

Splits a string into an array of substrings

[redacted]

[redacted]

[redacted]
[redacted]
[redacted]

Returns a string or a string object as a

string Returns a string converted to uppercase letters Returns a string with removed whitespaces Returns the primitive value of a string or a string object

A regular expression is an object that describes a pattern of characters.

The JavaScript RegExp class represents regular expressions, and both String and RegExp define methods that use regular expressions to perform powerful pattern-matching and search-and-replace functions on text.

Syntax:

```
var [redacted] new [redacted]
var [redacted]
```

Example:

```
var [redacted] /achscollege/i
```

Explanation:

`/achscollege/i` is a regular expression where achscollege is a pattern(to be used in a search) and i is a modifier(modifies the search to be case-insensitive).

Note: search(), replace(), match(), test() methods can be used with string.

Modifiers are used to perform case-insensitive and global searches:

Modifier	Description
g	Perform a global match (find all matches rather than stopping after the first match)
i	Perform case-insensitive matching
m	Perform multiline matching

Brackets are used to find a range of characters:

Expression	Description
[abc]	Find any character between the brackets
[^abc]	Find any character NOT between the brackets
[0-9]	Find any character between the brackets (any digit)

[^0-9] Find any character NOT between the brackets (any non-digit) (x|y) Find any of the alternatives specified

Metacharacters are characters with a special meaning:

Metacharacter	Description
.	Find a single character, except newline or line terminator
\w	Find a word character
\W	Find a non-word character
\d	Find a digit

\D Find a non-digit character

\s Find a whitespace character

\S Find a non-whitespace character

\b Find a match at the beginning/end of a word, beginning like this: \bHI, end like this: HI\b

\B Find a match, but not at the beginning/end of a word \0 Find a NULL

character

Find a new line character

Find a form feed character

Find a carriage return

character Find a tab

character

Find a vertical tab
character

\n
\f
\r
\t
\v

\xxx Find the character specified by an octal number xxx \xdd Find the character

specified by a hexadecimal number dd \udddd Find the Unicode character specified by a

hexadecimal number dddd

Quantifiers

Quantifier
`n+`
`n*`
`n?`
`n{X}`

Matches any string that contains at least one *n*

Matches any string that contains zero or more

occurrences of *n* Matches any string that

contains zero or one occurrences of *n* Matches

any string that contains a sequence of *X* *n*'s

Description

`n{X,Y}` Matches any string that contains a sequence of X to Y *n*'s `n{X,}` Matches any

string that contains a sequence of at least X *n*'s `n$` Matches any string with *n* at the

end of it `^n` Matches any string with *n* at the beginning of it `?=n` Matches any string

that is followed by a specific string *n* `?!n` Matches any string that is not followed by a

specific string *n*

The JavaScript Math object allows you to perform mathematical tasks on numbers.

The syntax for any Math property is : **Math.property**

Math Methods:

round() - It returns the nearest integer. Math.round(9.9)

= 10

Math.round(9.2) = 9

Math.round(9.5) = 10

sign()- It returns if the given number is negative null or positive.

Math.sign(5) = 1

Math.sign(0) = 0

Math.sign(-5) = -1

abs() - It returns the absolute value of given number (returns positive value).

Math.abs(-7) = 7

Math.abs(7) = 7

min() - Math.min() can be used to find the lowest value in a list of arguments.

Math.min(234,52,6,8,-8,-55) = -55

max() - Math.max() can be used to find the highest value in a list of arguments.

Math.max(243,54,123,54,-43,4-,34) = 243

random() - returns a random number between 0(inclusive) and 1 (exclusive)

Math.random() = 0.030564546456

pow() - Math.pow(x,y) returns the value of x to the power of y.

Math.pow(9,2) = 81

sqrt() - Math.sqrt(x) returns the square root of x

Math.sqrt(81) = 9

JavaScript eval() function

1. Evaluates the

expression.

2. Evaluates the string as JavaScript code and executes it.
3. Returns the result of the expression as number

eg: eval(“2+3”) = 5
eval(2+3) = 5

Form validation is the process of ensuring that the user input is clean, correct, and useful. Typical validation tasks are:

1. Has the user filled in all required fields? 2. Has the user entered a valid date?
3. Has the user entered text in a numeric field? 4. And many more..

Most often, the purpose of data validation is to ensure correct user input.

Validation can be defined by many different methods, and deployed in many different ways.

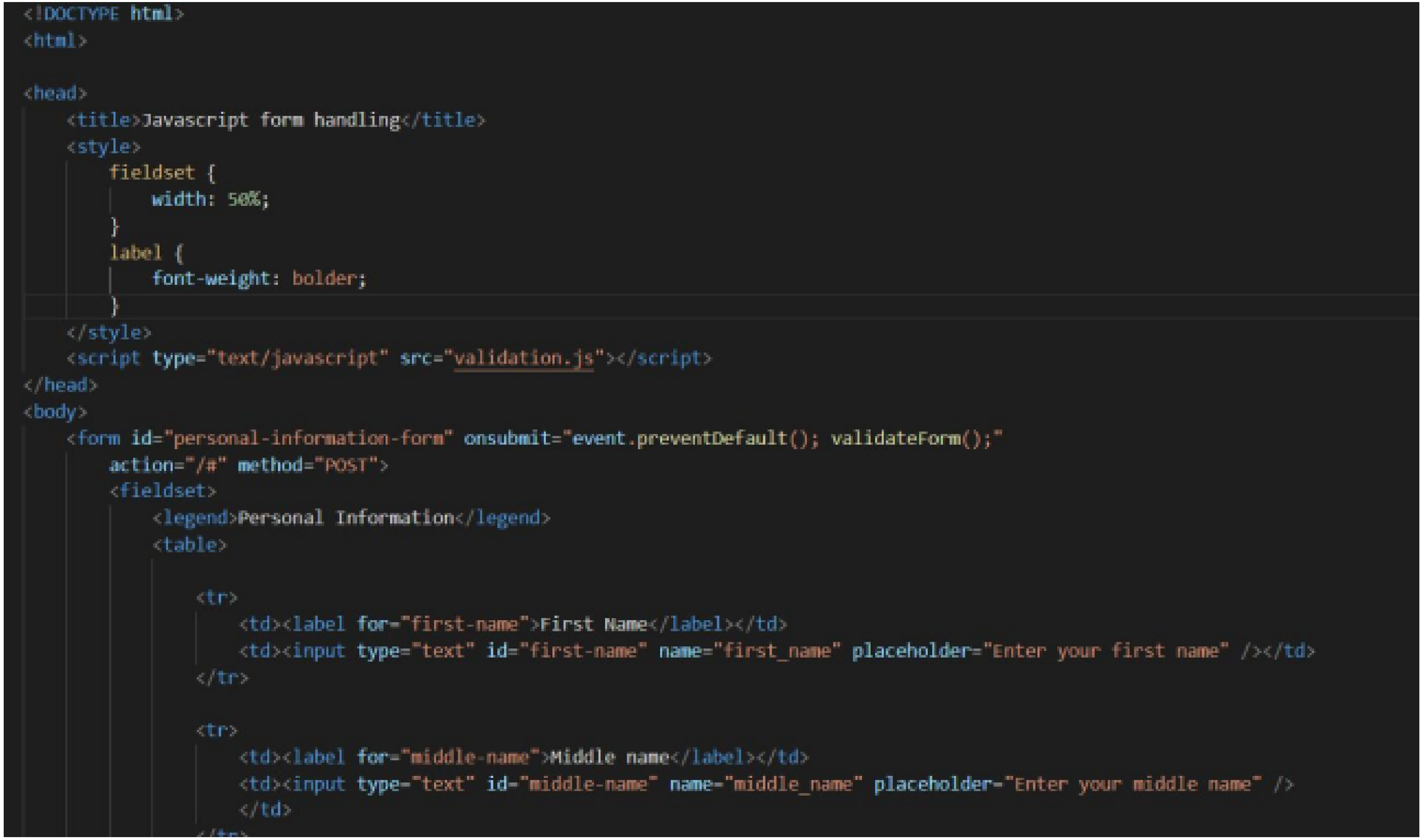
It is important to validate the form submitted by the user because it can have inappropriate values. So, validation is must to authenticate user.

JavaScript provides facility to validate the form on the client-side so data processing will be faster than server-side validation. Most of the the developers prefer JavaScript form validation.

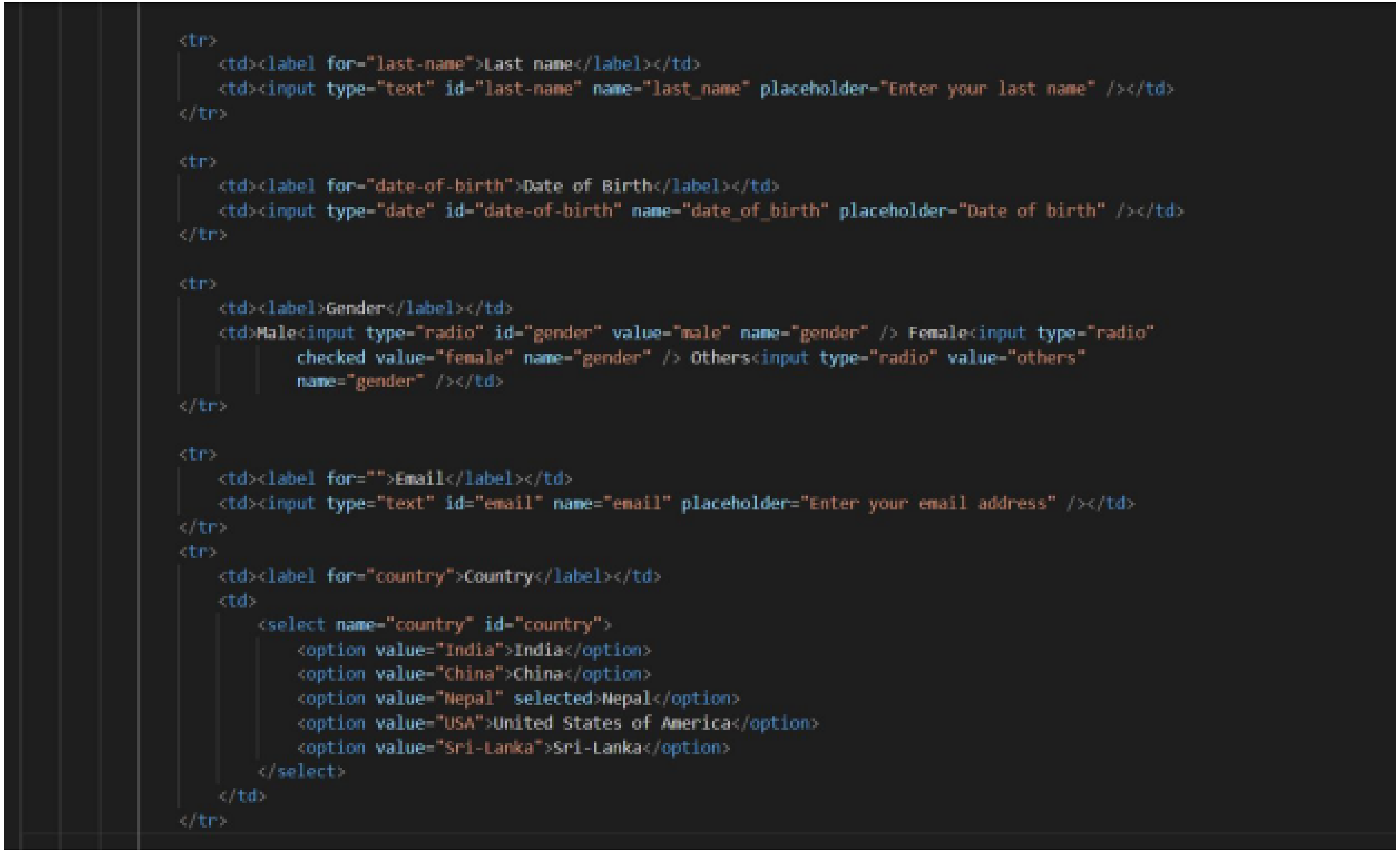
There are generally two types form validation done. They are:

1. Server Side Validation:
Validation is done by the web server after the input has been sent to the web server.
2. Client Side Validation:

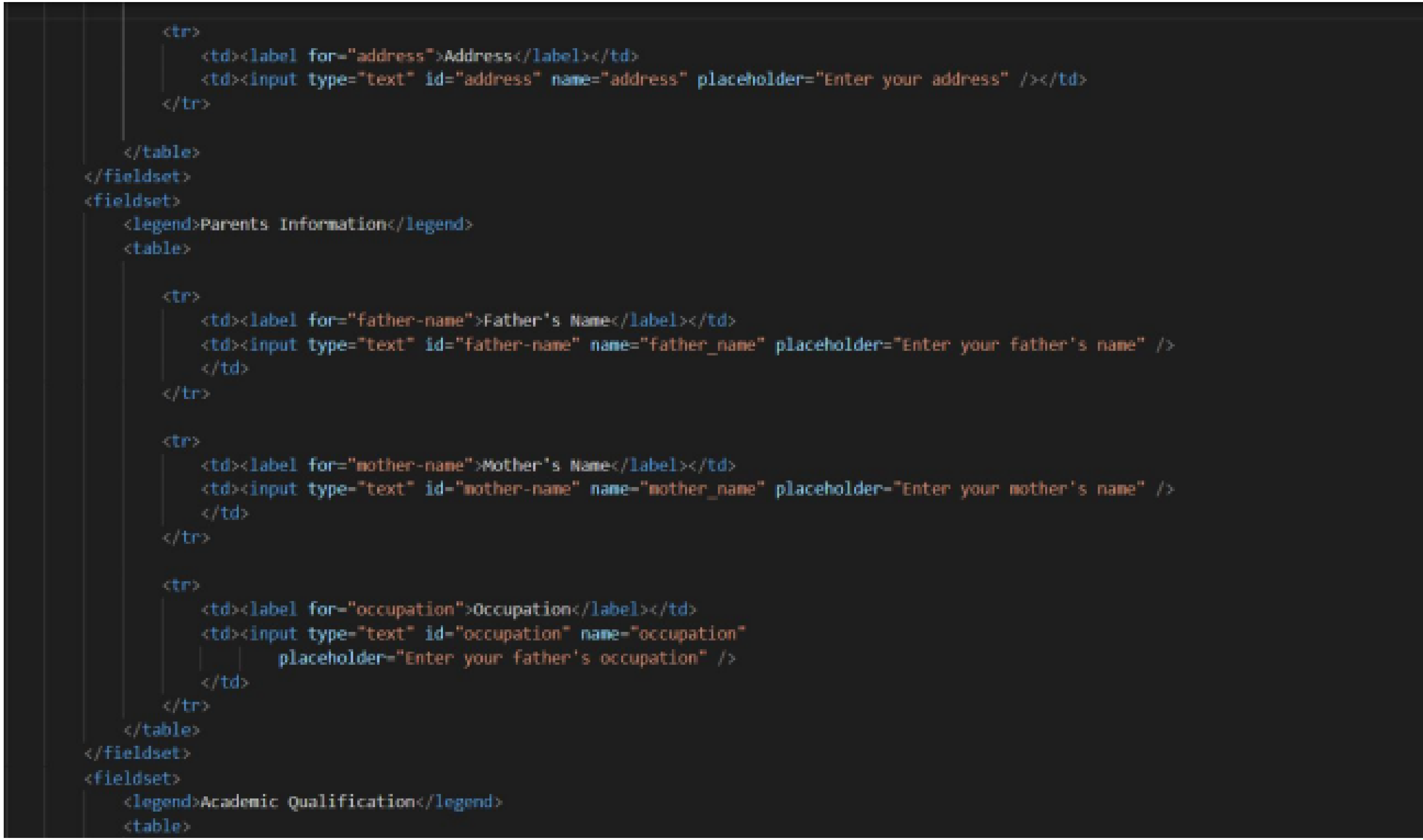
3. Validation is done by a web browser before the input is sent to the web server.



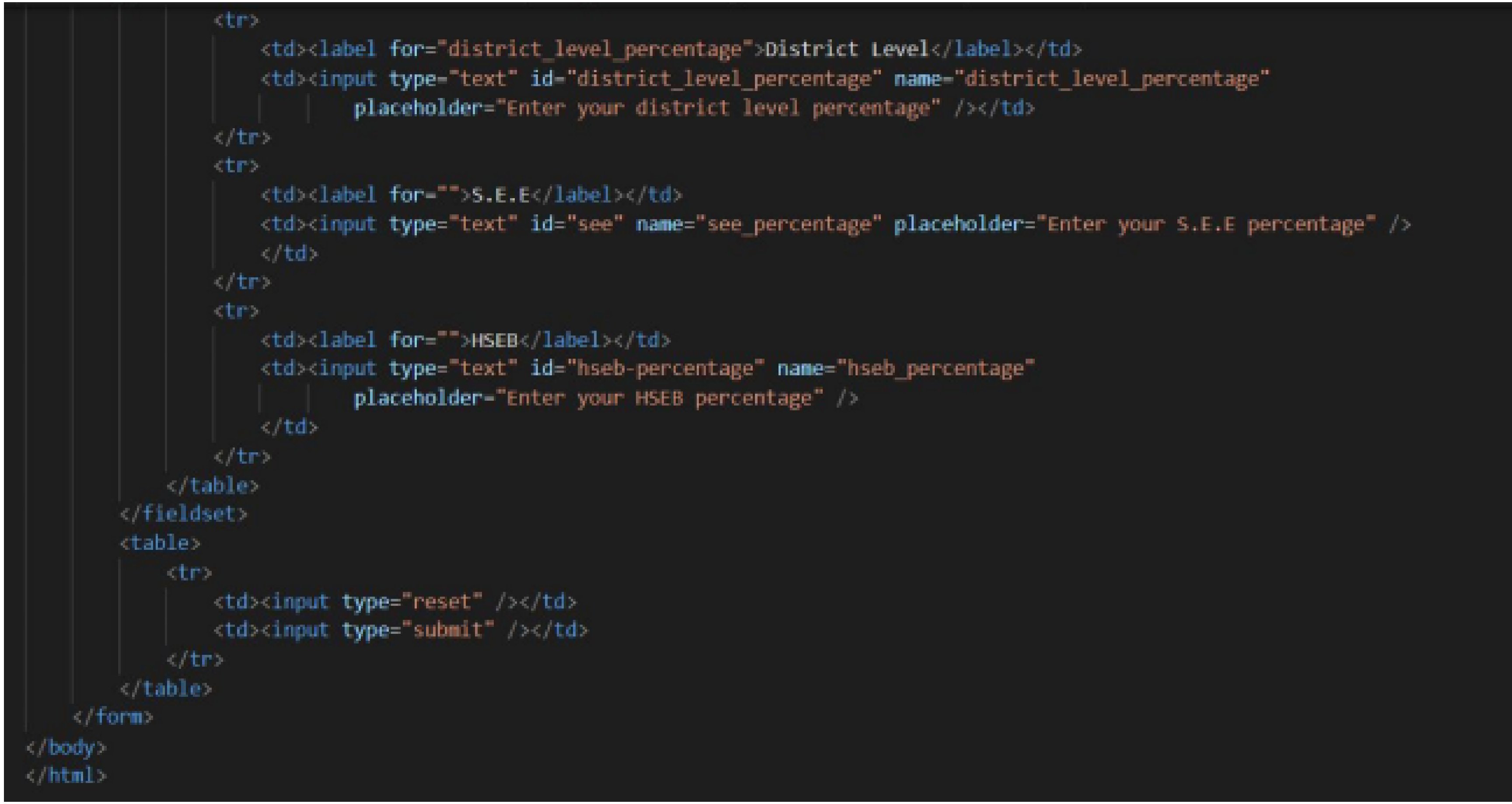
```
<!DOCTYPE html>
<html>

<head>
    <title>Javascript form handling</title>
    <style>
        fieldset {
            width: 50%;
        }
        label {
            font-weight: bolder;
        }
    </style>
    <script type="text/javascript" src="validation.js"></script>
</head>
<body>
    <form id="personal-information-form" onsubmit="event.preventDefault(); validateForm();"
        action="/#" method="POST">
        <fieldset>
            <legend>Personal Information</legend>
            <table>

                <tr>
                    <td><label for="first-name">First Name</label></td>
                    <td><input type="text" id="first-name" name="first_name" placeholder="Enter your first name" /></td>
                </tr>

                <tr>
                    <td><label for="middle-name">Middle name</label></td>
                    <td><input type="text" id="middle-name" name="middle_name" placeholder="Enter your middle name" />
                    </td>
```



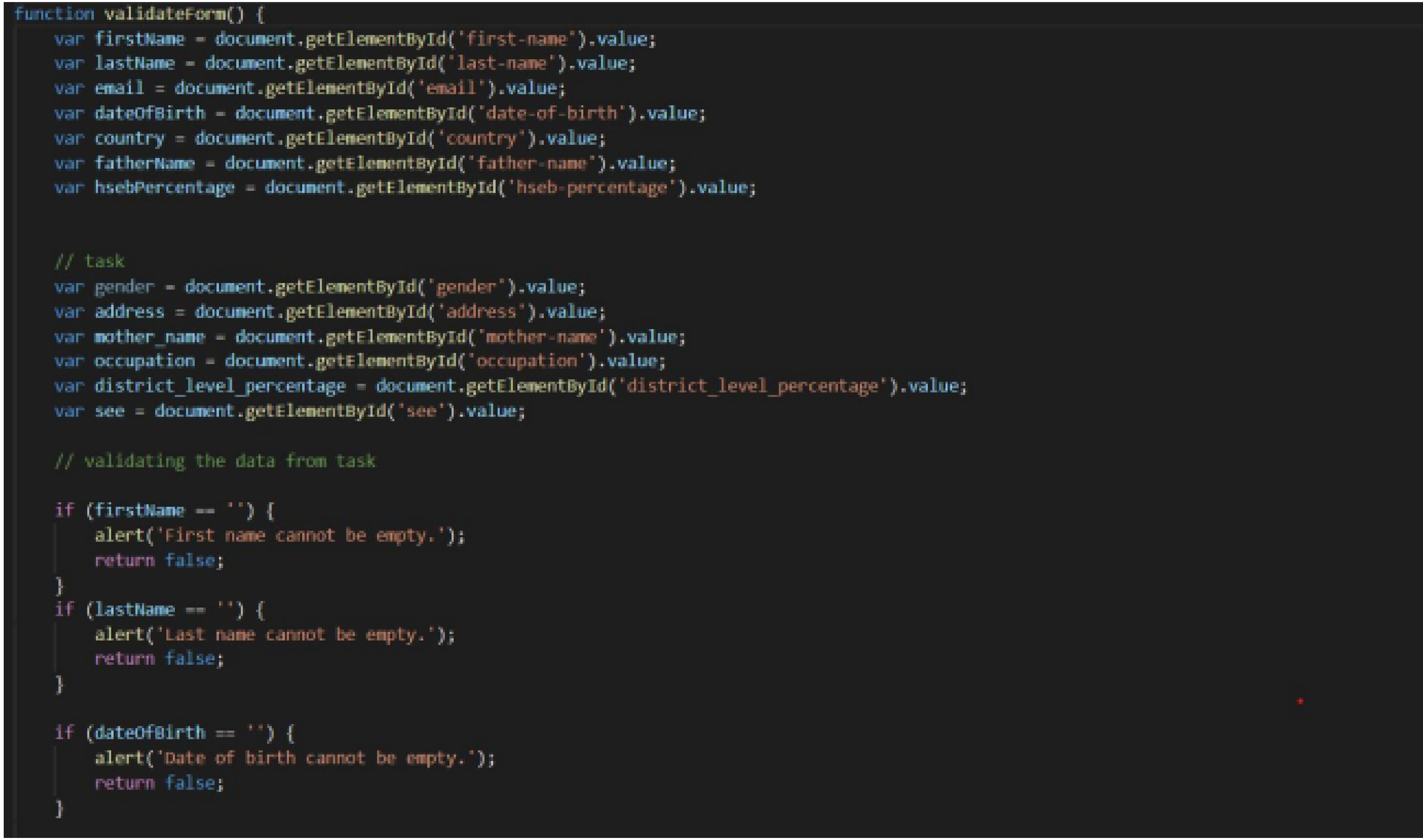
```html
<tr>
    <td><label for="last-name">Last name</label></td>
    <td><input type="text" id="last-name" name="last_name" placeholder="Enter your last name" /></td>
</tr>

<tr>
    <td><label for="date-of-birth">Date of Birth</label></td>
    <td><input type="date" id="date-of-birth" name="date_of_birth" placeholder="Date of birth" /></td>
</tr>

<tr>
    <td><label>Gender</label></td>
    <td>Male<input type="radio" id="gender" value="male" name="gender" /> Female<input type="radio"
            checked value="female" name="gender" /> Others<input type="radio" value="others"
            name="gender" /></td>
</tr>

<tr>
    <td><label for="">Email</label></td>
    <td><input type="text" id="email" name="email" placeholder="Enter your email address" /></td>
</tr>
<tr>
    <td><label for="country">Country</label></td>
    <td>
        <select name="country" id="country">
            <option value="India">India</option>
            <option value="China">China</option>
            <option value="Nepal" selected>Nepal</option>
            <option value="USA">United States of America</option>
            <option value="Sri-Lanka">Sri-Lanka</option>
        </select>
    </td>
</tr>
```



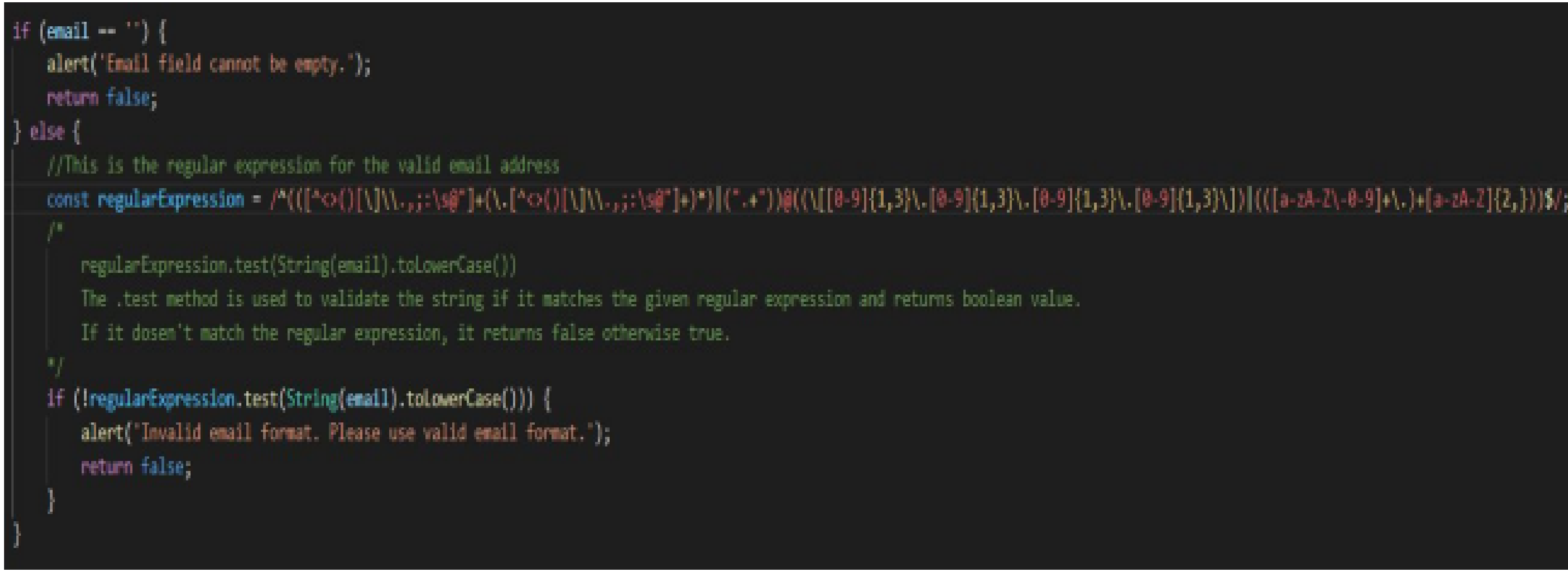
```
                <tr>
                    <td><label for="address">Address</label></td>
                    <td><input type="text" id="address" name="address" placeholder="Enter your address" /></td>
                </tr>

            </table>
        </fieldset>
        <fieldset>
            <legend>Parents Information</legend>
            <table>

                <tr>
                    <td><label for="father-name">Father's Name</label></td>
                    <td><input type="text" id="father-name" name="father_name" placeholder="Enter your father's name" />
                    </td>
                </tr>

                <tr>
                    <td><label for="mother-name">Mother's Name</label></td>
                    <td><input type="text" id="mother-name" name="mother_name" placeholder="Enter your mother's name" />
                    </td>
                </tr>

                <tr>
                    <td><label for="occupation">Occupation</label></td>
                    <td><input type="text" id="occupation" name="occupation"
                            placeholder="Enter your father's occupation" />
                    </td>
                </tr>
            </table>
        </fieldset>
        <fieldset>
            <legend>Academic Qualification</legend>
            <table>
```



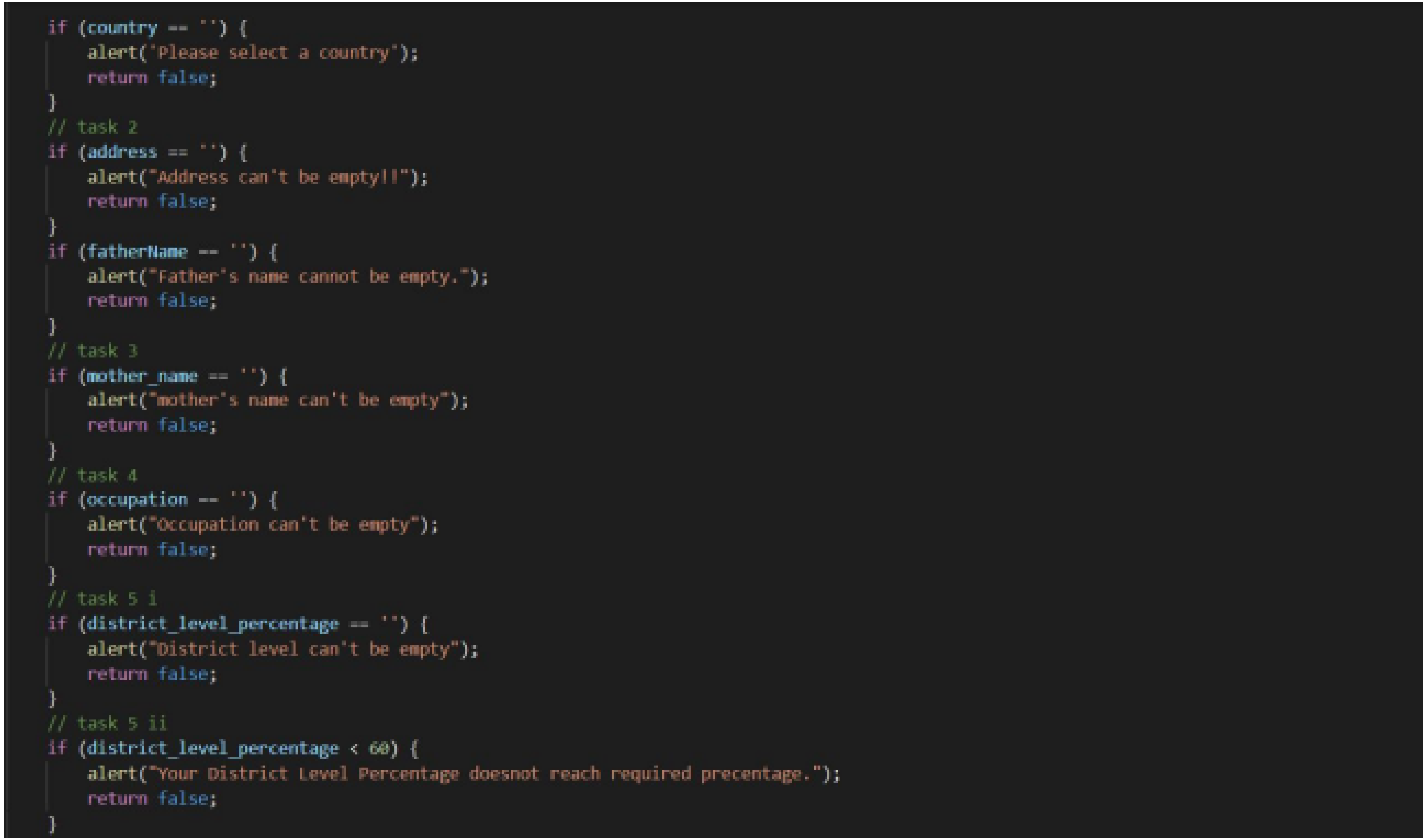
```
                <tr>
                    <td><label for="district_level_percentage">District Level</label></td>
                    <td><input type="text" id="district_level_percentage" name="district_level_percentage"
                            placeholder="Enter your district level percentage" /></td>
                </tr>
                <tr>
                    <td><label for="">S.E.E</label></td>
                    <td><input type="text" id="see" name="see_percentage" placeholder="Enter your S.E.E percentage" />
                    </td>
                </tr>
                <tr>
                    <td><label for="">HSEB</label></td>
                    <td><input type="text" id="hseb-percentage" name="hseb_percentage"
                            placeholder="Enter your HSEB percentage" />
                    </td>
                </tr>
            </table>
        </fieldset>
        <table>
            <tr>
                <td><input type="reset" /></td>
                <td><input type="submit" /></td>
            </tr>
        </table>
    </form>
</body>
</html>
```



```
function validateForm() {
    var firstName = document.getElementById('first-name').value;
    var lastName = document.getElementById('last-name').value;
    var email = document.getElementById('email').value;
    var dateOfBirth = document.getElementById('date-of-birth').value;
    var country = document.getElementById('country').value;
    var fatherName = document.getElementById('father-name').value;
    var hsebPercentage = document.getElementById('hseb-percentage').value;


    // task
    var gender = document.getElementById('gender').value;
    var address = document.getElementById('address').value;
    var mother_name = document.getElementById('mother-name').value;
    var occupation = document.getElementById('occupation').value;
    var district_level_percentage = document.getElementById('district_level_percentage').value;
    var see = document.getElementById('see').value;

    // validating the data from task

    if (firstName == '') {
        alert('First name cannot be empty.');
        return false;
    }
    if (lastName == '') {
        alert('Last name cannot be empty.');
        return false;
    }

    if (dateOfBirth == '') {
        alert('Date of birth cannot be empty.');
        return false;
    }
```



```
if (email == '') {
    alert('Email field cannot be empty.');
    return false;
} else {
    //This is the regular expression for the valid email address
    const regularExpression = /^(([^<>()[\]\\.,;:\s@"]+(\.[^<>()[\]\\.,;:\s@"]+)*)|(".+"))@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\])|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))$/;
    /*
        regularExpression.test(String(email).toLowerCase())
        The .test method is used to validate the string if it matches the given regular expression and returns boolean value.
        If it dosen't match the regular expression, it returns false otherwise true.
    */
    if (!regularExpression.test(String(email).toLowerCase())) {
        alert('Invalid email format. Please use valid email format.');
        return false;
    }
}
```



```
if (country == '') {
    alert('Please select a country');
    return false;
}
// task 2
if (address == '') {
    alert("Address can't be empty!!");
    return false;
}
if (fatherName == '') {
    alert("Father's name cannot be empty.");
    return false;
}
// task 3
if (mother_name == '') {
    alert("mother's name can't be empty");
    return false;
}
// task 4
if (occupation == '') {
    alert("Occupation can't be empty");
    return false;
}
// task 5 i
if (district_level_percentage == '') {
    alert("District level can't be empty");
    return false;
}
// task 5 ii
if (district_level_percentage < 60) {
    alert("Your District Level Percentage doesnot reach required precentage.");
    return false;
}
```