INT219
Front End Web Developer

BY :KEDAR NATH SINGH
UID:29465

Unit I

Overview of HTML, CSS and JavaScript : Fundamentals of HTML, Creating Style Sheet, CSS Box Model- Border properties, Padding properties, Margin properties, Introduction to JavaScript, Working with Web Forms and validating user input, JavaScript functions and events, JavaScript Timing Events, JavaScript Image Slideshow, Recursive function in JavaScript, Error handling in JavaScript

HTML:

- HTML stands for Hyper Text Markup Language.
- HTML is the standard markup language for creating Web pages.
- HTML describes the structure of a Web page.
- HTML consists of a series of elements.
- HTML elements tell the browser how to display the content.
- HTML elements label pieces of content such as "this is a heading", "this is a paragraph", "this is a link", etc.

HTML(contd.)

- HTML was originally developed by Tim Berners-Lee in 1990. He is also known as the father of the web.
- In 1996, the World Wide Web Consortium (W3C) became the authority to maintain the HTML specifications. HTML also became an international standard (ISO) in 2000.
- HTML5 is the latest version of HTML. HTML5 provides a faster and more robust approach to web development.

Versions of HTML(contd.)

- HTML 1.0: released in 1991
- HTML 2.0: released in 1995
- HTML 3.2: released in 1997
- HTML 4.01: released in 1999
- HTML5: released in 2014

STRUCTURE:

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>

<h1>This is a Heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

The body tag is a container that contains all the elements or tags that will be displayed to the user.Tags used in Body tags are:Inside the body tag the parts are divided into;

(I) header (II) section (III) footer

- <h1>,<h2>(like this till <h6>),
- <p>,
- <div>,
- <header>,<section>,<footer>,
- <br>,<hr>
- <img>,
- <ul><li>
- <span>
- <form>
- <label> and many more…

Explanation of Structure :

- The <!DOCTYPE html> declaration defines that this document is an HTML5 document
- The <html> element is the root element of an HTML page
- The <head> element contains meta information about the HTML page
- The <title> element specifies a title for the HTML page (which is shown in the browser's title bar or in the page's tab)
- The <body> element defines the document's body, and is a container for all the visible contents, such as headings, paragraphs, images, hyperlinks, tables, lists, etc.
- The <h1> element defines a large heading
- The <p> element defines a paragraph

What is an HTML Element?

- An HTML element is defined by a start tag, some content, and an end tag:

<tagname> Content goes here... </tagname>

- The HTML element is everything from the start tag to the end tag:

<h1>My First Heading</h1>

<p>My first paragraph.</p>

Various Tags :

- **HTML Headings:**
  - HTML headings are defined with the <h1> to <h6> tags.
  - <h1> defines the most important heading. <h6> defines the least important heading.

- **HTML Paragraphs:**
  - HTML paragraphs are defined with the <p> tag.

- **HTML Links:**
  - HTML links are defined with the <a> tag.(Anchor Tag)

```
<a href="https://www.youtube.com">This is a link</a>
```

  - The link's destination is specified in the href attribute.

- **HTML Images:**
  - HTML images are defined with the <img> tag.
  - The source file (src), alternative text (alt), width, and height are provided as attributes.

```
<img src="abcd.jpg" alt="this is an image" width="104" height="142">.
```

- **Line Break:**
  - The <br> tag defines a line break, and is an empty element without a closing tag.
- **HTML Horizontal Rules:**
  - The <hr> tag defines a thematic break in an HTML page, and is most often displayed as a horizontal rule.
  - The <hr> element is used to separate content (or define a change) in an HTML page.
  - The <hr> tag is an empty tag, which means that it has no end tag.
- **HTML <pre> Element:**
  - The HTML <pre> element defines preformatted text.
  - The text inside a <pre> element is displayed in a fixed-width font (usually Courier), and it preserves both spaces and line breaks (eg poem format)

- **HTML Styles:**

- The HTML style attribute is used to add styles to an element, such as color, font, size, and more.
- Setting the style of an HTML element, can be done with the style attribute.
- The HTML style attribute has the following syntax:

```
<tagname style="property:value;">
```

- **Background Color:**

- The CSS background-color property defines the background color for an HTML element.

```
<body style="background-color:powderblue;">
<h1 style="background-color:powderblue;">This is a heading</h1>
```

- **Text Color:**
  - The CSS color property defines the text color for an HTML element:

```
<h1 style="color:blue;">This is a heading</h1>
```

- **Fonts:**
  - The CSS font-family property defines the font to be used for an HTML element:

```
<h1 style="font-family:verdana;">This is a heading</h1>
```

- **Text Size:**
  - The CSS font-size property defines the text size for an HTML element:

```
<h1 style="font-size:300%;">This is a heading</h1>
```

- **Text Alignment:**
  - The CSS text-align property defines the horizontal text alignment for an HTML element:

```
<h1 style="text-align:center;">Centered Heading</h1>
```

HTML Formatting Elements

- Formatting elements were designed to display special types of text:
- <b> - Bold text
- <strong> - Important text
- <i> - Italic text
- <em> - Emphasized text
- <mark> - Marked text
- <small> - Smaller text
- <del> - Deleted text
- <ins> - Inserted text
- <sub> - Subscript text
- <sup> - Superscript text

HTML Quotation

- HTML <q> for Short Quotations:
  - The HTML <q> tag defines a short quotation.

```
<p>WWF's goal is to: <q>Build a future where people live in harmony with nature.</q></p>
```

- HTML <abbr> for Abbreviations:
  - The HTML <abbr> tag defines an abbreviation or an acronym, like "HTML", "CSS", "Mr.", "Dr.", "ASAP", "ATM".

```
<p>The <abbr title="World Health Organization">WHO</abbr> was founded in 1948.</p>
```

- HTML <bdo> for Bi-Directional Override:
  - BDO stands for Bi-Directional Override.
  - The HTML <bdo> tag is used to override the current text direction:

```
<bdo dir="rtl">This text will be written from right to left</bdo>
```

- **HTML Comments:**
  - Notice that there is an exclamation point (!) in the start tag, but not in the end tag.
  - Comments are not displayed by the browser, but they can help document your HTML source code.

```
<!-- This is a comment -->
<p>This is a paragraph.</p>
<!-- Remember to add more information here -->
```

HTML Video

- The HTML <video> element is used to show a video on a web page.
- To show a video in HTML, use the <video> element:

```
<video width="320" height="240" controls>
  <source src="movie.mp4" type="video/mp4">
  <source src="movie.ogg" type="video/ogg">
Your browser does not support the video tag.
</video>
```

- The controls attribute adds video controls, like play, pause, and volume.
- It is a good idea to always include width and height attributes. If height and width are not set, the page might flicker while the video loads.
- The <source> element allows you to specify alternative video files which the browser may choose from. The browser will use the first recognized format.
- The text between the <video> and </video> tags will only be displayed in browsers that do not support the <video> element.

HTML Audio

- To play an audio file in HTML, use the <audio> element:The controls attribute adds audio controls, like play, pause, and volume.

```
<audio controls>
  <source src="horse.ogg" type="audio/ogg">
  <source src="horse.mp3" type="audio/mpeg">
Your browser does not support the audio element.
</audio>
```

- The <source> element allows you to specify alternative audio files which the browser may choose from. The browser will use the first recognized format.
- The text between the <audio> and </audio> tags will only be displayed in browsers that do not support the <audio> element.

HTML YouTube Videos

- The easiest way to play videos in HTML, is to use YouTube.
- Converting videos to different formats can be difficult and time-consuming.
- An easier solution is to let YouTube play the videos in your web page.
- An HTML iframe is used to display a web page within a web page.
- The HTML <iframe> tag specifies an inline frame.
- An inline frame is used to embed another document within the current HTML document.

```
<iframe src="url" title="description"></iframe>
```

- Example :

```
<iframe width="420" height="315"
src="https://www.youtube.com/embed/tgbNymZ7vqY">
</iframe>
```

CSS

- CSS stands for Cascading Style Sheets.
- CSS is the language we use to style an HTML document.
- CSS describes how HTML elements should be displayed.
- CSS can be added to HTML documents in 3 ways:
  - Inline - by using the style attribute inside HTML elements.
  - Internal - by using a <style> element in the <head> section.
  - External - by using a <link> element to link to an external CSS file.

CSS Syntax

- A CSS rule consists of a selector and a declaration block.
- The selector points to the HTML element you want to style.
- The declaration block contains one or more declarations separated by semicolons.
- Each declaration includes a CSS property name and a value, separated by a colon.
- Multiple CSS declarations are separated with semicolons, and declaration blocks are surrounded by curly braces.

- selector { Declaration;Declaration}

```
p {
  color: red;
  text-align: center;
}
```

- p is a selector in CSS (it points to the HTML element you want to style: <p>).
- color is a property, and red is the property value
- text-align is a property, and center is the property value

Inline CSS

- An inline CSS is used to apply a unique style to a single HTML element.
- An inline CSS uses the style attribute of an HTML element.
- The following example sets the text color of the <h1> element to blue, and the text color of the <p> element to red:
  - <h1 style="color:blue;">A Blue Heading</h1>
  - <p style="color:red;">A red paragraph.</p>

Internal CSS

- An internal CSS is used to define a style for a single HTML page.
- An internal CSS is defined in the <head> section of an HTML page, within a <style> element.
- The following example sets the text color of ALL the <h1> elements (on that page) to blue, and the text color of ALL the <p> elements to red. In addition, the page will be displayed with a "powderblue" background color:

```
<!DOCTYPE html>    <html>
<head>
<style>
body {background-color: powderblue;}
h1   {color: blue;}
p    {color: red;}
</style>          </head>
<body>
<h1>This is a heading</h1>
<p>This is a paragraph.</p>
</body></html>
```

External CSS

- An external style sheet is used to define the style for many HTML pages.
- To use an external style sheet, add a link to it in the <head> section of each HTML page:

```
<!DOCTYPE html>
<html>
<head>
 <link rel="stylesheet" href="styles.css">
</head>
<body>
<h1>This is a heading</h1>
<p>This is a paragraph.</p>
</body>
</html>
```

- The external style sheet can be written in any text editor. The file must not contain any HTML code, and must be saved with a .css extension.
- Here is what the "styles.css" file looks like:

```
body {
  background-color: powderblue;
}
h1 {
  color: blue;
}
p {
  color: red;
}
```

CSS Backgrounds

- The CSS background properties are used to add background effects for elements.
- CSS background-color:
  - The background-color property specifies the background color of an element :

```
body {
  background-color: lightblue;
}
```

- Opacity / Transparency:
  - The opacity property specifies the opacity/transparency of an element. It can take a value from 0.0 - 1.0. The lower value, the more transparent:

```
div {
  background-color: green;
  opacity: 0.3;
}
```

- CSS background-image:
- The background-image property specifies an image to use as the background of an element.

```
body {
  background-image: url("bgdesert.jpg");
  background-position: right top;
  background-attachment: fixed;
  background-attachment: scroll;
}
```

CSS Box Model

- In CSS, the term "box model" is used when talking about design and layout.
- The CSS box model is essentially a box that wraps around every HTML element. It consists of: margins, borders, padding, and the actual content.

- Explanation of the different parts:
- Content - The content of the box, where text and images appear
- Padding - Clears an area around the content. The padding is transparent
- Border - A border that goes around the padding and content
- Margin - Clears an area outside the border. The margin is transparent

CSS Box Model

```
div {
  width: 300px;
  border: 15px solid green;
  padding: 50px;
  margin: 20px;
}
```

JavaScript

- JavaScript was initially created to “make web pages alive”.
- The programs in this language are called scripts. They can be written right in a web page’s HTML and run automatically as the page loads.
- Scripts are provided and executed as plain text. They don’t need special preparation or compilation to run.
- In this aspect, JavaScript is very different from another language called Java.
- Today, JavaScript can execute not only in the browser, but also on the server, or actually on any device that has a special program called the JavaScript engine.
- The browser has an embedded engine sometimes called a “JavaScript virtual machine”.

JAVASCRIPT SYNTAX

- JavaScript programs can be inserted almost anywhere into an HTML document (HEAD & BODY)using the <script> tag.
- The type and language attributes are not required.
- A script in an external file can be inserted with <script src="path/to/script.js"></script>.
- EXAMPLE:

```
<!DOCTYPE HTML>
<html>
<body>
<p>Before the script...</p>
<script>
        alert( 'Hello, world!' );
</script>
<p>...After the script.</p>
</body>
</html>
```

JavaScript Output

- JavaScript can "display" data in different ways:
  - Writing into an HTML element, using innerHTML.
  - Writing into the HTML output using document.write().
  - Writing into an alert box, using alert().
  - Writing into the browser console, using console.log().

Using innerHTML

- To access an HTML element, JavaScript can use the document.getElementById(id) method.
- The id attribute defines the HTML element. The innerHTML property defines the HTML content:

```
<!DOCTYPE html>
<html>
<body>
<h1>My First Web Page</h1>
<p>My First Paragraph</p>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = 5 + 6;
</script>
</body>
</html>
```

Using document.write()

```
<!DOCTYPE html>
<html>
<body>
<h1>My First Web Page</h1>
<p>My first paragraph.</p>
<script>
document.write(5 + 6);
</script>
</body>
</html>
```

Task :

- Create a button named : “Click Me” and on clicking the button should disappear and the following content should be displayed “ This is the information i got after clicking on button “

Solution:

```
<!DOCTYPE html>
<html>
<body>
<button type="button" onclick='document.write("This is the information i got after clicking on button")'>Click Me</button>
</body>
</html>
```

JavaScript Print

- JavaScript does not have any print object or print methods.
- You cannot access output devices from JavaScript.
- The only exception is that you can call the window.print() method in the browser to print the content of the current window.

```
<!DOCTYPE html>
<html>
<body>
<button onclick="window.print()">Print this
page</button>
</body>
</html>
```

Functions

- Functions are the main “building blocks” of the program. They allow the code to be called many times without repetition.
- We’ve already seen examples of built-in functions, like alert(message), prompt(message, default) and confirm(question). But we can create functions of our own as well.

❑ **Function Declaration**

- To create a function we can use a function declaration.

```
function showMessage() {
    alert( 'Hello everyone!' );
}
```

- The function keyword goes first, then goes the name of the function, then a list of parameters between the parentheses (comma-separated, empty in the example) and finally the code of the function, also named “the function body”, between curly braces.

VARIABLES

- Most of the time, a JavaScript application needs to work with information. Here are two examples:
  - An online shop – the information might include goods being sold and a shopping cart.
  - A chat application – the information might include users, messages, and much more.
- Variables are used to store this information.
- A variable is a "named storage" for data. We can use variables to store goodies, visitors, and other data.
- To create a variable in JavaScript, use the let keyword.
- The statement below creates (in other words: declares) a variable with the name "message":

```
let message;
```

Constants:

- To declare a constant (unchanging) variable, use const instead of let:

```
const myBirthday = '18.04.1982';
```

- Variables declared using const are called “constants”. They cannot be reassigned. An attempt to do so would cause an error:

```
const myBirthday = '18.04.1982';
myBirthday = '01.01.2001'; // error, can't reassign the constant!
```

- When a programmer is sure that a variable will never change, they can declare it with const to guarantee and clearly communicate that fact to everyone.

SCOPE

- Scope determines the accessibility (visibility) of variables.
- Before ES6 (2015), JavaScript had only Global Scope and Function Scope(local scope)
- ES6 introduced two important new JavaScript keywords: let and const.
- These two keywords provide Block Scope in JavaScript.

- JavaScript has 3 types of scope:
  - Block scope
  - Function scope
  - Global scope
- 1. Block Scope:
  - Variables declared inside a { } block cannot be accessed from outside the block:
  - Variables declared with the var keyword can NOT have block scope.
  - Variables declared inside a { } block can be accessed from outside the block.

EXAMPLE:

```
{
var x = 2;
}
// x CAN be used here
```

- 2. Local Scope
  - Variables declared within a JavaScript function, become LOCAL to the function , i.e it can only be accessed within a function.
  - Example:

```
let a = "hello";
function greet()
                    {
let b = "World"
          console.log(a + b);
                    }
                    greet();
console.log(a + b); // error
```

- In the above program, variable a is a global variable and variable b is a local variable. The variable b can be accessed only inside the function greet. Hence, when we try to access variable b outside of the function, an error occurs.

- **let is Block Scoped:**
  - The let keyword is block-scoped (variable can be accessed only in the immediate block).
  - Example:

```
                    // global variable
                    let a = 'Hello';
                    function greet() {
                    let b = 'World';                    // local variable
                    console.log(a + ' ' + b);
                    if (b == 'World') {
                      let c = 'hello';                       // block-scoped
variable
                      console.log(a + ' ' + b + ' ' + c);
                           }
   console.log(a + ' ' + b + ' ' + c);    // variable c cannot be accessed here
                          }
                          greet();
```

- In the above program, variable
  - a is a global variable. It can be accessed anywhere in the program.
  - b is a local variable. It can be accessed only inside the function greet.
  - c is a block-scoped variable. It can be accessed only inside the if statement block.
- Hence, in the above program, the first two console.log() work without any issue.
- However, we are trying to access the block-scoped variable c outside of the block in the third console.log(). This will throw an error.

- **3. GLOBAL SCOPE**
  - A variable declared at the top of a program or outside of a function is considered a global scope variable.
  - Example:

```
let a = "hello";
function greet () {
console.log(a);
        }
greet(); // hello
```

  - In the above program, variable a is declared at the top of a program and is a global variable. It means the variable a can be used anywhere in the program.

- **Note: It is a good practice to avoid using global variables because the value of a global variable can change in different areas in the program. It can introduce unknown results in the program.**
- In JavaScript, a variable can also be used without declaring it. If a variable is used without declaring it, that variable automatically becomes a global variable.

JAVASCRIPT DATATYPES

- A value in JavaScript is always of a certain type. For example, a string or a number.
- There are eight basic data types in JavaScript.
- We can put any type in a variable. For example, a variable can at one moment be a string and then store a number:

```
let message = "hello";
message = 123456;
```

- Programming languages that allow such things, such as JavaScript, are called “dynamically typed”, meaning that there exist data types, but variables are not bound to any of them.

Seven primitive data types:

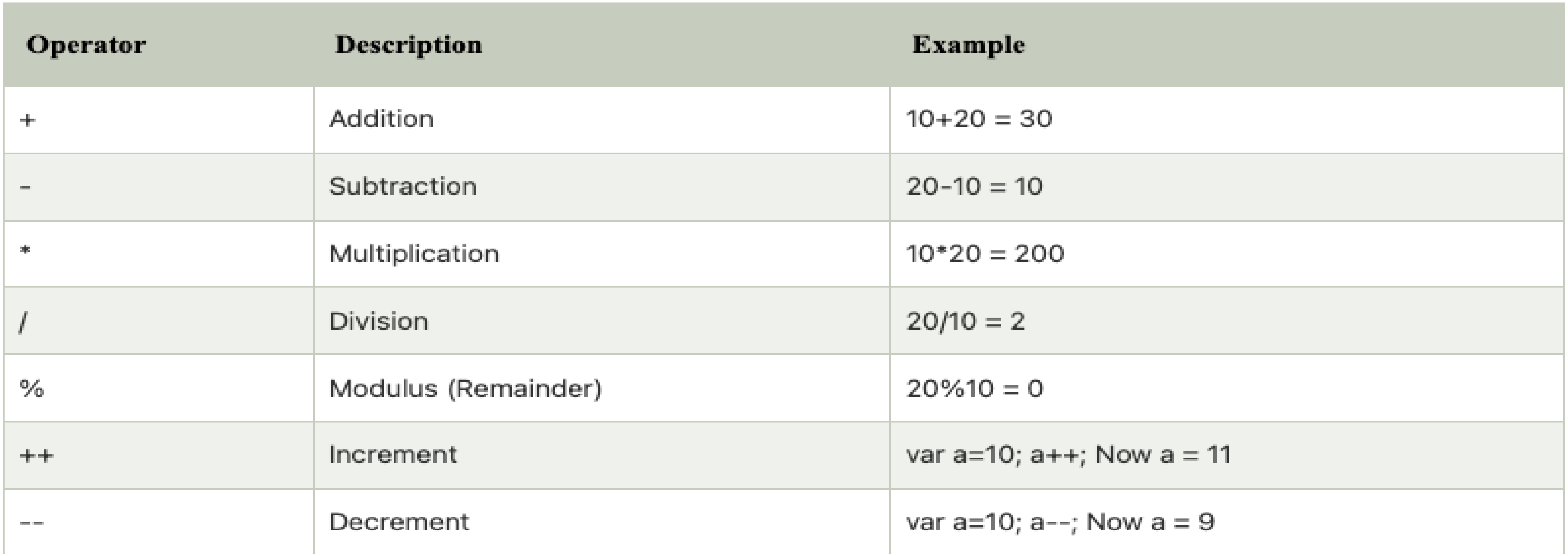
- number for numbers of any kind: integer or floating-point, integers are limited by ±(253-1).
- bigint for integer numbers of arbitrary length.
- string for strings. A string may have zero or more characters, there’s no separate single-character type.
- boolean for true/false.
- null for unknown values – a standalone type that has a single value null.
- undefined for unassigned values – a standalone type that has a single value undefined.
- symbol for unique identifiers.

And one non-primitive data type:

- object for more complex data structures.

OPERATORS

- An operand – is what operators are applied to. For instance, in the multiplication of 5 * 2 there are two operands: the left operand is 5 and the right operand is 2. Sometimes, people call these “arguments” instead of “operands”.
- ARITHEMATIC OPERATORS:



Operator	Description	Example
+	Addition	10+20 = 30
-	Subtraction	20-10 = 10
*	Multiplication	10*20 = 200
/	Division	20/10 = 2
%	Modulus (Remainder)	20%10 = 0
++	Increment	var a=10; a++; Now a = 11
--	Decrement	var a=10; a--; Now a = 9

Arithematic

- <!DOCTYPE html>
- <html>
- <body>
- <p id="demo"></p>
- <script>
- let x = 100 + 50;
- document.getElementById("demo").innerHTML = x;
- </script>
- </body>
- </html>

JavaScript Comparison Operators

The JavaScript comparison operator compares the two operands. The comparison operators are as follows:

Operator	Description	Example
==	Is equal to	10==20 = false
===	Identical (equal and of same type)	10==20 = false
!=	Not equal to	10!=20 = true
!==	Not Identical	20!==20 = false
>	Greater than	20>10 = true
>=	Greater than or equal to	20>=10 = true
<	Less than	20<10 = false
<=	Less than or equal to	20<=10 = false

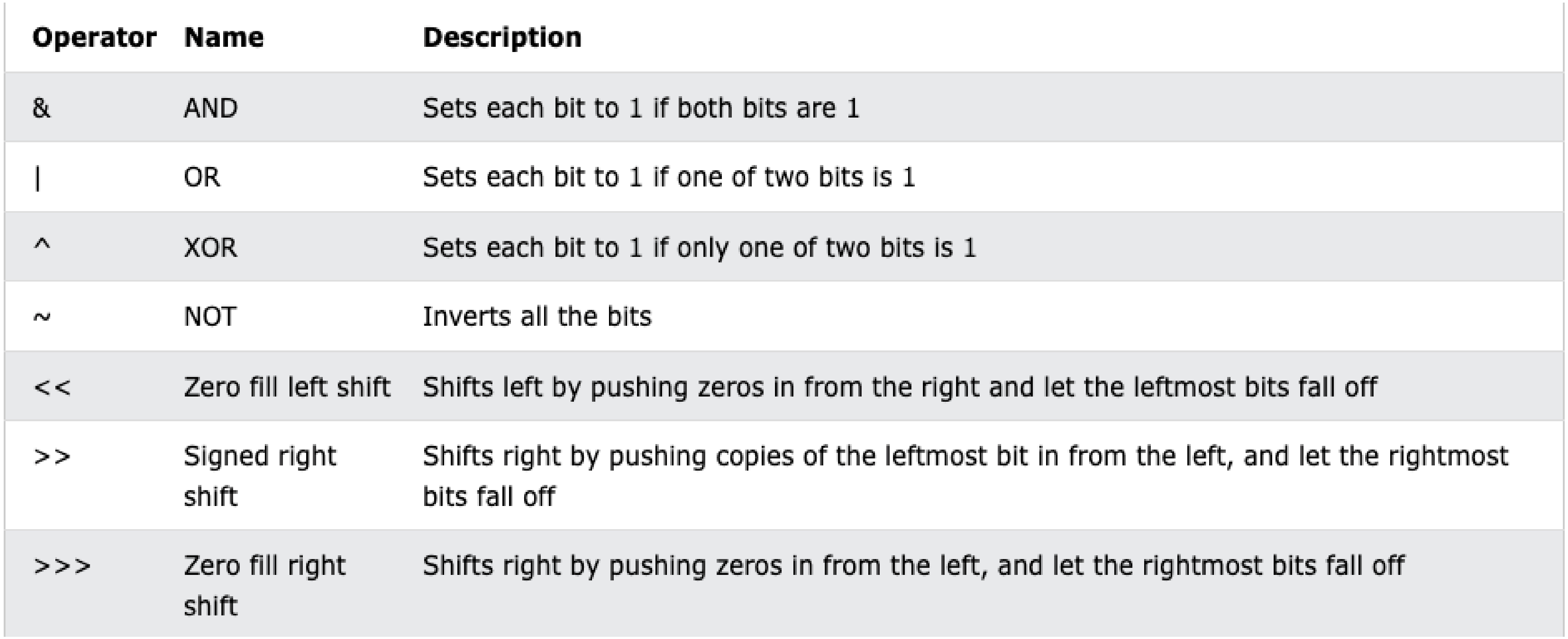
Example:

- <!DOCTYPE html> <html> <body>
- <p>Assign 5 to x, and display the value of the comparison (x == 5):</p>
- <p>Assign 5 to x, and display the value of the comparison (x === "5").</p>
- <p id="demo"></p>
- <script>
- let x = 5;
- document.getElementById("demo").innerHTML = (x == "5");
- document.getElementById("demo").innerHTML = (x === "5");
- </script></body></html>

JavaScript Bitwise Operators

The bitwise operators perform bitwise operations on operands. The bitwise operators are as follows:

Operator	Description	Example
&	Bitwise AND	(10==20 & 20==33) = false
|	Bitwise OR	(10==20 | 20==33) = false
^	Bitwise XOR	(10==20 ^ 20==33) = false
~	Bitwise NOT	(~10) = -10
<<	Bitwise Left Shift	(10<<2) = 40
>>	Bitwise Right Shift	(10>>2) = 2
>>>	Bitwise Right Shift with Zero	(10>>>2) = 2



Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shifts left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shifts right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off
>>>	Zero fill right shift	Shifts right by pushing zeros in from the left, and let the rightmost bits fall off

Example:

- <!DOCTYPE html>
- <html>
- <body>
- <h1>JavaScript Bitwise AND</h1>
- <h2>The & Operator</h2>
- <p id="demo"></p>
- <script>
- document.getElementById("demo").innerHTML = 5 & 1;
- document.getElementById("demo").innerHTML = 5 | 1;
- document.getElementById("demo").innerHTML = 5 ^ 1;
- </script> </body> </html>

JavaScript Logical Operators

The following operators are known as JavaScript logical operators.

Operator	Description	Example
&&	Logical AND	(10==20 && 20==33) = false
||	Logical OR	(10==20 || 20==33) = false
!	Logical Not	!(10==20) = true

Example:

- <!DOCTYPE html> <html> <body>
- <h2>The && Operator (Logical AND)</h2>
- <p>The && operator returns true if both expressions are true, otherwise it returns false.</p>
- <p id="demo"></p>
- <script>
- let x = 6;
- let y = 3;
- document.getElementById("demo").innerHTML =
- (x < 10 && y > 1) + "<br>" +
- (x < 10 && y < 1);
- </script> </body> </html>

JavaScript Assignment Operators

The following operators are known as JavaScript assignment operators.

Operator	Description	Example
=	Assign	10+10 = 20
+=	Add and assign	var a=10; a+=20; Now a = 30
-=	Subtract and assign	var a=20; a-=10; Now a = 10
*=	Multiply and assign	var a=10; a*=20; Now a = 200
/=	Divide and assign	var a=10; a/=2; Now a = 5
%=	Modulus and assign	var a=10; a%=2; Now a = 0

Recursive function

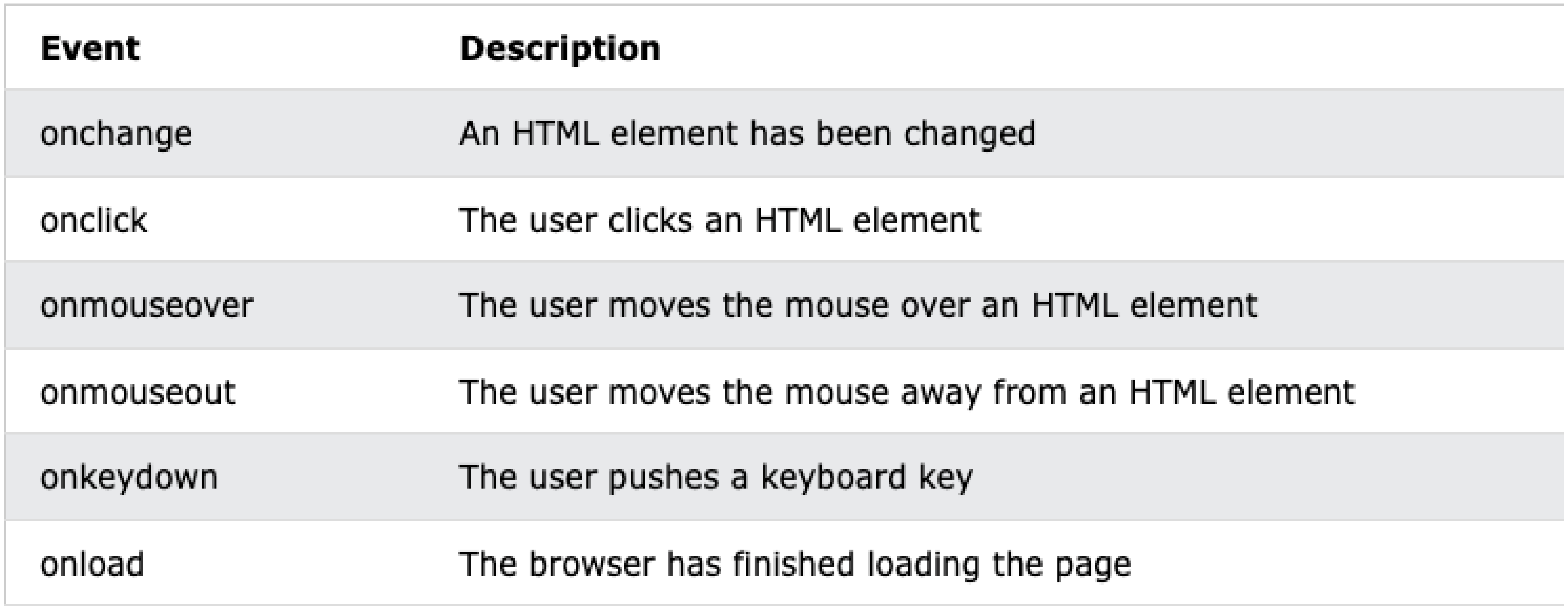
```
<script>
   // program to count down numbers to 1
                         function countDown(number) {
 // display the number
                          document.write(number, "<br>");
 // decrease the number value
                           const newNumber = number - 1;
// base case
                           if (newNumber > 0) {
                          countDown(newNumber);
                           }
                         }
                                    countDown(4);
        </script>
```

Events

- An HTML event can be something the browser does, or something a user does.
- Here are some examples of HTML events:
    - An HTML web page has finished loading
    - An HTML input field was changed
    - An HTML button was clicked
- Often, when events happen, you may want to do something.
- JavaScript lets you execute code when events are detected.

Common HTML Events

- Here is a list of some common HTML events:



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

Java Script Function Example :

```
<html>
<head>    <title>Add function</title>     </head>

<body onload="add()">   // Execute a JavaScript immediately after a page has been
loaded
    <script>
        var i=Number(prompt("Enter 1st variable"));
        var j=Number(prompt("enter 2nd varialbe"));
        function add()
        {
        document.write("the sum is ",(i+j));
        }
        </script>
</body>
</html>
```

Javascript Event Example:

```
<html>
<input type="button" value="Hover" id="btn"
onmouseover="this.style.background='red';
this.style.color='yellow'"
onmouseout="this.style.background='cadetblue';
this.style.color='white'"/>
</html>
```

Javascript Example 2 :

```
<html>     <input type="button" value="Click me" id="btn"
onmouseover="changeColorOnMouseOver(this)"
onmouseout="changeColorOnMouseOut(this)"/>
<script type="text/javascript">
function changeColorOnMouseOver(x)
{
x.style.background='red';
x.style.color='yellow';
}
function changeColorOnMouseOut(x)
{
x.style.background='cadetblue';
x.style.color='white';
}
</script>
</html>
```

JavaScript Arrow Function

- Arrow function is one of the features introduced in the ES6 version of JavaScript. It allows you to create functions in a cleaner way compared to regular functions. For example,
- This function

```
// function expression
let x = function(x, y) {
        return x * y;
                  }
```

can be written as

```
// using arrow functions
let x = (x, y) => x * y;
```

- Arrow Function Syntax

  The syntax of the arrow function is:

  let myFunction = (arg1, arg2, ...argN) => {

  statement(s)

  }

- Here,myFunction is the name of the function arg1, arg2, ...argN are the function argumentsstatement(s) is the function body
- If the body has single statement or expression, you can write arrow function as:

  let myFunction = (arg1, arg2, ...argN) => expression

Java Script Objects

- As we know from the chapter Data types, there are eight data types in JavaScript. Seven of them are called “primitive”, because their values contain only a single thing (be it a string or a number or whatever).
- In contrast, objects are used to store keyed collections of various data and more complex entities. In JavaScript, objects penetrate almost every aspect of the language. So we must understand them first before going in-depth anywhere else.
- An object can be created with figure brackets {…} with an optional list of properties. A property is a “key: value” pair, where key is a string (also called a “property name”), and value can be anything.
- We can imagine an object as a cabinet with signed files. Every piece of data is stored in its file by the key. It’s easy to find a file by its name or add/remove a file.

- An empty object (“empty cabinet”) can be created using one of two syntaxes:

```
let user = new Object(); // "object constructor" syntax
let user = {};  // "object literal" syntax
```

- Usually, the figure brackets {...} are used. That declaration is called an object literal.

- JavaScript object is a non-primitive data-type that allows you to store multiple collections of data.
- **Note: If you are familiar with other programming languages, JavaScript objects are a bit different. You do not need to create classes in order to create objects.**
- Example:

```
// object
const student = {
        firstName: 'ram',
        class: 10
    };
```

Here, student is an object that stores values such as strings and numbers.

- **JavaScript Object Declaration**
  - The syntax to declare an object is:

```
const object_name = {
key1: value1,
key2: value2            }
```

- **Accessing Object Properties:**
  - You can access the value of a property by using its key.

1. Using dot Notation
   - Here's the syntax of the dot notation.

objectName.key

2. Using bracket Notation
   - Here is the syntax of the bracket notation.

objectName["propertyName"]

JavaScript Methods and this Keyword

- In JavaScript, objects can also contain functions. For example,

```
// object containing method
const person = {
 name: 'John',
greet: function() { console.log('hello'); }
};
```

- In the above example, a person object has two keys (name and greet), which have a string value and a function value, respectively.
- Hence basically, the JavaScript method is an object property that has a function value.

Accessing Object Methods

- You can access an object method using a dot notation. The syntax is: **objectName.methodKey()**
- You can access property by calling an objectName and a key. You can access a method by calling an objectName and a key for that method along with (). For example,

```
// accessing method and property
const person = {
name: 'John',
greet: function() { console.log('hello'); }
};
// accessing property
person.name; // John
// accessing method
person.greet(); // hello
```

Here, the greet method is accessed as person.greet() instead of person.greet.

If you try to access the method with only person.greet, it will give you a function definition.

JavaScript this Keyword

- To access a property of an object from within a method of the same object, you need to use the this keyword. Let's consider an example.

```
const person = {
  name: 'John',
  age: 30,

  // accessing name property by using this.name
  greet: function() { console.log('The name is' + ' ' + this.name); }
};
person.greet();
```

- In the above example, a person object is created. It contains properties (name and age) and a method greet.
- In the method greet, while accessing a property of an object, this keyword is used.
- In order to access the properties of an object, this keyword is used following by . and key.
- **Note: In JavaScript, this keyword when used with the object's method refers to the object. this is bound to an object.**

Literals and properties

- We can immediately put some properties into {...} as “key: value” pairs:

```
let user = {     // an object
name: "John",  // by key "name" store value "John"
age: 30        // by key "age" store value 30
        };
```

- A property has a key (also known as “name” or “identifier”) before the colon ":" and a value to the right of it.
- In the user object, there are two properties:
- The first property has the name "name" and the value "John".
- The second one has the name "age" and the value 30.

- We can add, remove and read files from it at any time.
- Property values are accessible using the dot notation:

```
// get property values of the object:
        alert( user.name ); // John
        alert( user.age ); // 30
```

- The value can be of any type. Let’s add a boolean one:

```
user.isAdmin = true;
```

- To remove a property, we can use the delete operator:

```
delete user.age;
```

- We can also use multiword property names, but then they must be quoted:

```
let user = {
      name: "John",
      age: 30,
  "likes birds": true  // multiword property name must be quoted
};
```

- The last property in the list may end with a comma:

```
let user = {
    name: "John",
    age: 30,
}
```

- That is called a “trailing” or “hanging” comma. Makes it easier to add/remove/move around properties, because all lines become alike.

JavaScript Constructor Function

- In JavaScript, a constructor function is used to create objects. For example,

```
/ constructor function
function Person () {
  this.name = 'John',
    this.age = 23
}
// create an object
const person = new Person();
```

- In the above example, function Person() is an object constructor function.
- To create an object from a constructor function, we use the new keyword.

- JavaScript this Keyword
- In JavaScript, when this keyword is used in a constructor function, this refers to the object when the object is created. For example,

```
// constructor function
function Person () {
  this.name = 'John',
}
// create object
const person1 = new Person();
// access properties
console.log(person1.name);  // John
```

- Hence, when an object accesses the properties, it can directly access the property as person1.name.

Create Objects: Constructor Function Vs Object Literal

- Object Literal is generally used to create a single object. The constructor function is useful if you want to create multiple objects. For example,

```
// using object literal
                              let person = {
                              name: 'Sam'
                                        }
// using constructor function
                              function Person () {
                               this.name = 'Sam'
                                        }
                              let person1 = new Person();
                              let person2 = new Person();
```

Working With Arrays

- An array is an object that can store multiple values at once. For example,

```
const words = ['hello', 'world', 'welcome'];
```

- Here, words is an array. The array is storing 3 values.
- **Create an Array:**
- You can create an array using two ways:

1. Using an array literal

- The easiest way to create an array is by using an array literal []. For example,

```
const array1 = ["eat", "sleep"];
```

2. Using the new keyword

- You can also create an array using JavaScript's new keyword.

```
const array2 = new Array("eat", "sleep");
```

- In both of the above examples, we have created an array having two elements.

Note: It is recommended to use array literal to create an array.

- Here are more examples of arrays:

```
// empty array
                const myList = [ ];
// array of numbers
                const numberArray = [ 2, 4, 6, 8];
// array of strings
                const stringArray = [ 'eat', 'work', 'sleep'];
// array with mixed data types
                const newData = ['work', 'exercise', 1, true];
```

- You can also store arrays, functions and other objects inside an array. For example,

```
const newData = [
 {'task1': 'exercise'},
 [1, 2 ,3],
 function hello() { console.log('hello')}
];
```

- **Access Elements of an Array**
- You can access elements of an array using indices (0, 1, 2 …). For example,

```
const myArray = ['h', 'e', 'l', 'l', 'o'];
          // first element
console.log(myArray[0]);  // "h"
          // second element
console.log(myArray[1]); // "e"
```

- **Add an Element to an Array**
- You can use the built-in method push() and unshift() to add elements to an array.
- The push() method adds an element at the end of the array. For example,

```
let dailyActivities = ['eat', 'sleep'];
// add an element at the end
dailyActivities.push('exercise');
console.log(dailyActivities); //  ['eat', 'sleep', 'exercise']
```

- The unshift() method adds an element at the beginning of the array. For example,

```
let dailyActivities = ['eat', 'sleep'];
//add an element at the start
dailyActivities.unshift('work');
console.log(dailyActivities); // ['work', 'eat', 'sleep']
```

- **Change the Elements of an Array**
- You can also add elements or change the elements by accessing the index value.

```
        let dailyActivities = [ 'eat', 'sleep'];
    // this will add the new element 'exercise' at the 2 index
        dailyActivities[2] = 'exercise';
    console.log(dailyActivities); // ['eat', 'sleep', 'exercise']
```

- **Remove an Element from an Array**
- You can use the pop() method to remove the last element from an array. The pop() method also returns the returned value. For example,

```
    let dailyActivities = ['work', 'eat', 'sleep', 'exercise'];
    // remove the last element
    dailyActivities.pop();
    console.log(dailyActivities); // ['work', 'eat', 'sleep']
```

- If you need to remove the first element, you can use the shift() method.
- The shift() method removes the first element and also returns the removed element. For example,

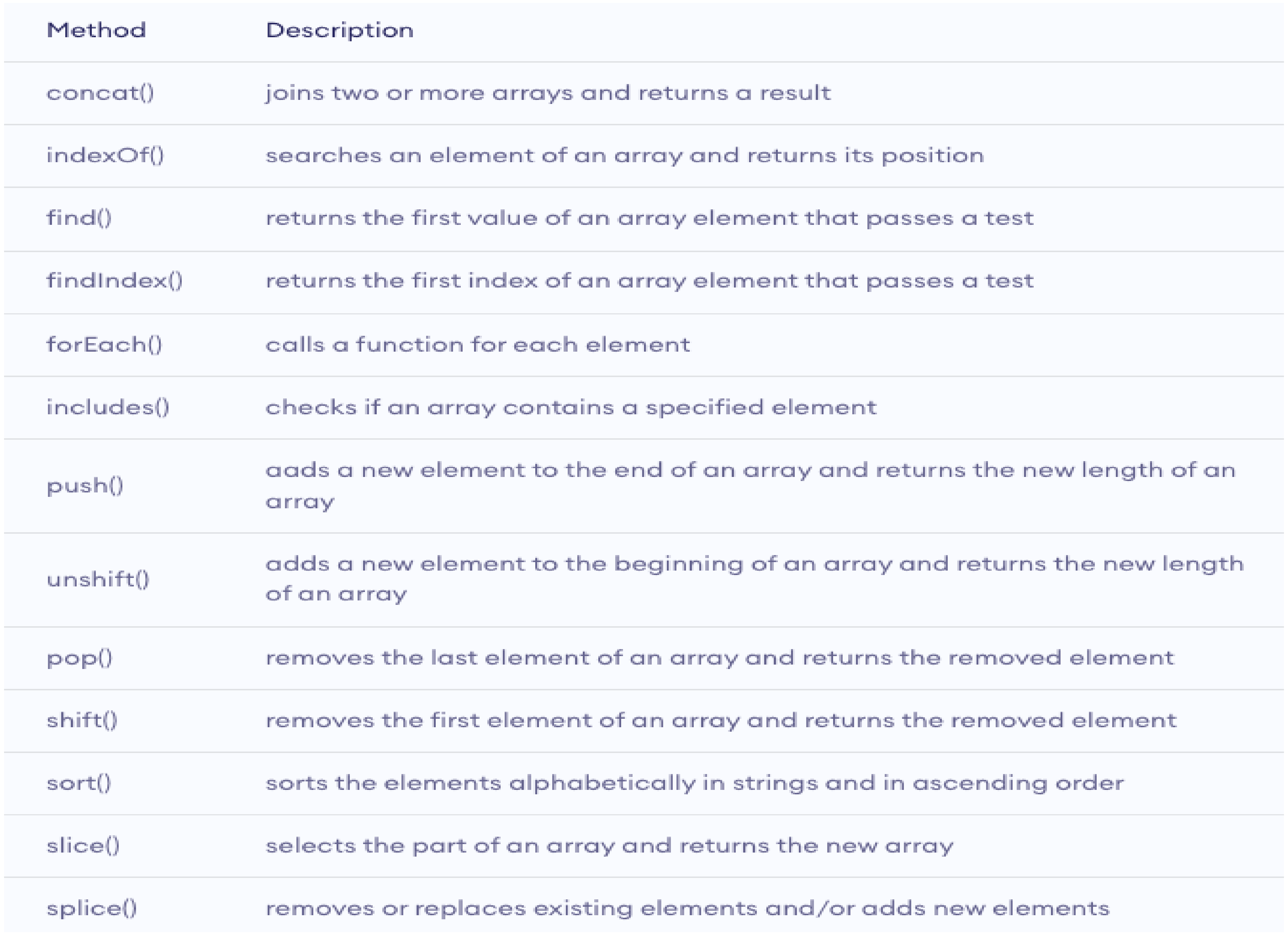
```
let dailyActivities = ['work', 'eat', 'sleep'];
        // remove the first element
dailyActivities.shift();
console.log(dailyActivities); // ['eat', 'sleep']
```

Array length

You can find the length of an element (the number of elements in an array) using the length property. For example,

```
const dailyActivities = [ 'eat', 'sleep'];
        // this gives the total number of elements in an array
console.log(dailyActivities.length); // 2
```

Array Methods



Method	Description
concat()	joins two or more arrays and returns a result
indexOf()	searches an element of an array and returns its position
find()	returns the first value of an array element that passes a test
findIndex()	returns the first index of an array element that passes a test
forEach()	calls a function for each element
includes()	checks if an array contains a specified element
push()	aads a new element to the end of an array and returns the new length of an array
unshift()	adds a new element to the beginning of an array and returns the new length of an array
pop()	removes the last element of an array and returns the removed element
shift()	removes the first element of an array and returns the removed element
sort()	sorts the elements alphabetically in strings and in ascending order
slice()	selects the part of an array and returns the new array
splice()	removes or replaces existing elements and/or adds new elements

JavaScript Multidimensional Array

- A multidimensional array is an array that contains another array. For example,

```
// multidimensional array
const data = [[1, 2, 3], [1, 3, 4], [4, 5, 6]];
```

Create a Multidimensional Array:

Example 1

```
let studentsData = [['Jack', 24], ['Sara', 23], ['Peter', 24]];
```

Example 2

```
let student1 = ['Jack', 24];
let student2 = ['Sara', 23];
let student3 = ['Peter', 24];
// multidimensional array
let studentsData = [student1, student2, student3];
```

Error Handling

- Types of Errors
  - In programming, there can be two types of errors in the code:
  - Syntax Error: Error in the syntax. For example, if you write consol.log('your result');, the above program throws a syntax error. The spelling of console is a mistake in the above code.
  - Runtime Error: This type of error occurs during the execution of the program. For example,

    calling an invalid function or a variable.
  - These errors that occur during runtime are called exceptions. Now, let's see how you can handle these exceptions.

JavaScript try...catch Statement

- The try...catch statement is used to handle the exceptions. Its syntax is:

```
try {      // body of try         }
catch(error) {     // body of catch    }
```

- The main code is inside the try block. While executing the try block, if any error occurs, it goes to the catch block. The catch block handles the errors as per the catch statements.
- If no error occurs, the code inside the try block is executed and the catch block is skipped.

Example 1: Display Undeclared Variable

- // program to show try...catch in a program

```
        const numerator= 100, denominator = 'a';
        try {
                console.log(numerator/denominator);
                        // forgot to define variable a
                         console.log(a);  }
        catch(error) {
                console.log('An error caught');
                console.log('Error message: ' + error);
                        }
```

JavaScript try...catch...finally Statement

- You can also use the try...catch...finally statement to handle exceptions. The finally block executes both when the code runs successfully or if an error occurs.
- The syntax of try...catch...finally block is:

```
try {    // try_statements}
catch(error) {       // catch_statements  }
finally() { // codes that gets executed anyway}
```

Note: You need to use catch or finally statement after try statement. Otherwise, the program will throw an error Uncaught SyntaxError: Missing catch or finally after try.

Example 2: try...catch...finally Example

```
const numerator= 100, denominator = 'a';
try {
     console.log(numerator/denominator);
     console.log(a);
}
catch(error) {
    console.log('An error caught');
    console.log('Error message: ' + error);
}
finally {
     console.log('Finally will execute every time');
}
```

JavaScript throw Statement

- In JavaScript, the throw statement handles user-defined exceptions. For example, if a certain number is divided by 0, and if you need to consider Infinity as an exception, you can use the throw statement to handle that exception.
- The syntax of throw statement is:

```
throw expression;
```

Here, expression specifies the value of the exception.

For example,

```
const number = 5;
throw number/0; // generate an exception when divided by 0
```

JavaScript throw with try...catch

- The syntax of try...catch...throw is:

```
try { // body of try
throw exception;        }
catch(error) {  // body of catch  }
```

Note: When the throw statement is executed, it exits out of the block and goes to the catch block. And the code below the throw statement is not executed.

Example 1: try...catch...throw Example

```
const number = 40;
try {
        if(number > 50) {
        console.log('Success');
                    }
         else {
      throw new Error('The number is low');                // user-defined throw
statement
         }
   // if throw executes, the below code does not execute
   console.log('hello');
}
catch(error) {
        console.log('An error caught');
        console.log('Error message: ' + error);
        }
```