



## RQF LEVEL 3



**SWDJF301**  
**SOFTWARE**  
**DEVELOPMENT**  

---

**JavaScript**  
**Fundamentals**

**TRAINEE'S MANUAL**

*October, 2024*



# JAVASCRIPT FUNDAMENTALS



## AUTHOR'S NOTE PAGE (COPYRIGHT)

The competent development body of this manual is Rwanda TVET Board ©, reproduce with permission.

All rights reserved.

- This work has been produced initially with the Rwanda TVET Board with the support from KOICA through TQUM Project
- This work has copyright, but permission is given to all the Administrative and Academic Staff of the RTB and TVET Schools to make copies by photocopying or other duplicating processes for use at their own workplaces.
- This permission does not extend to making of copies for use outside the immediate environment for which they are made, nor making copies for hire or resale to third parties.
- The views expressed in this version of the work do not necessarily represent the views of RTB. The competent body does not give warranty nor accept any liability
- RTB owns the copyright to the trainee and trainer's manuals. Training providers may reproduce these training manuals in part or in full for training purposes only. Acknowledgment of RTB copyright must be included on any reproductions. Any other use of the manuals must be referred to the RTB.

© Rwanda TVET Board

*Copies available from:*

- HQs: Rwanda TVET Board-RTB
- Web: [www.rtb.gov.rw](http://www.rtb.gov.rw)
- KIGALI-RWANDA

Original published version: October 2024

## ACKNOWLEDGEMENTS

The publisher would like to thank the following for their assistance in the elaboration of this training manual:

Rwanda TVET Board (RTB) extends its appreciation to all parties who contributed to the development of the trainer's and trainee's manuals for the TVET Certificate III in Software Development, specifically for the module "**SWDJF301: JavaScript Fundamentals.**"

We extend our gratitude to KOICA Rwanda for its contribution to the development of these training manuals and for its ongoing support of the TVET system in Rwanda.

We extend our gratitude to the TQUM Project for its financial and technical support in the development of these training manuals.

We would also like to acknowledge the valuable contributions of all TVET trainers and industry practitioners in the development of this training manual.

The management of Rwanda TVET Board extends its appreciation to both its staff and the staff of the TQUM Project for their efforts in coordinating these activities.

**This training manual was developed:**

Under Rwanda TVET Board (RTB) guiding policies and directives



Under Financial and Technical support of



## **COORDINATION TEAM**

RWAMASIRABO Aimable

MARIA Bernadette M. Ramos

MUTIJIMA Asher Emmanuel

## **PRODUCTION TEAM**

### **Authoring and Review**

HABIGENA Alexandre

### **Validation**

.....

### **Conception, Adaptation and Editorial works**

HATEGEKIMANA Olivier

GANZA Jean Francois Regis

HARELIMANA Wilson

NZABIRINDA Aimable

DUKUZIMANA Therese

NIYONKURU Sylvestre

KWIZERA INGABIRE Diane

### **Formatting, Graphics, Illustrations, and infographics**

YEONWOO Choe

SUA Lim

SAEM Lee

SOYEON Kim

WONYEONG Jeong

KAYUMBA Fiston

### **Financial and Technical support**

KOICA through TQUM Project

## TABLE OF CONTENT

AUTHOR'S NOTE PAGE (COPYRIGHT)-----	iii
ACKNOWLEDGEMENTS-----	iv
TABLE OF CONTENT -----	vii
ACRONYMS-----	ix
INTRODUCTION -----	1
MODULE CODE AND TITLE: SWDJF301 JAVASCRIPT FUNDAMENTALS -----	2
Learning Outcome 1Apply JavaScript Basic Concepts -----	3
Key Competencies for Learning Outcome 1 : Apply JavaScript Basic Concepts-----	4
Indicative content 1.1: Introduction to JavaScript -----	6
Indicative content 1.2: Integration of JavaScript to HTML-----	23
Indicative content 1.3: Use of variables in JavaScript -----	39
Indicative content 1.4: Use of data types in JavaScript -----	51
Indicative content 1.5: Use of operators in JavaScript-----	62
Learning outcome 1 end assessment -----	71
References -----	73
Learning Outcome 2: Manipulate data with JavaScript-----	74
Key Competencies for Learning Outcome 2: Manipulate data with JavaScript-----	75
Indicative content 2.1: Using string in JavaScript -----	77
Indicative content 2.2: Using conditional statements -----	96
Indicative content 2.3: Using Loop functions in JavaScript -----	110
Indicative content 2.4: Using Functions in JavaScript -----	122
Indicative content 2.5: Using objects in JavaScript -----	151
Indicative content 2.6: Using arrays in JavaScript-----	169
Indicative content 2.7: Using JavaScript in HTML -----	199
Indicative content 2.8: Applying regular expressions -----	251
Indicative content 2.9: Error handling -----	258
Learning outcome 2 end assessment -----	264
References -----	266
Learning Outcome 3: Apply JavaScript in project-----	267
Key Competencies for Learning Outcome 3: Apply JavaScript in project -----	268
Indicative content 3.1: Apply JavaScript in Project -----	270
Indicative content 3.2: Create pages with HTML-----	275

Indicative content 3.3: Apply CSS to HTML pages-----	307
Indicative content 3.4: Apply JavaScript concepts in project-----	314
Learning outcome 3 end assessment -----	318
References-----	320

## ACRONYMS

**API:** Application Programming Interface

**CND:** Content Delivery Network

**CSS:** Cascading Style Sheet

**DOM:** Document Object Model

**HTML:** Hypertext Markup Language

**IDE:** Integrated Development Environment

**JS:** JavaScript

**MVC:** Model-View-Controller

**OS:** Operating System

**RTB:** Rwanda TVET Board

**SPAs:** Single-Page Applications

**TQUM Project:** TVET Quality Management Project

**URI:** Uniform Resource Identifier

**VS Code:** Visual Studio Code

## INTRODUCTION

This trainee's manual includes all the knowledge and skills required in Software Development specifically for the module of "**SWDJF301: Javascript Fundamentals**". Trainees enrolled in this module will engage in practical activities designed to develop and enhance their competencies. The development of this training manual followed the Competency-Based Training and Assessment (CBT/A) approach, offering ample practical opportunities that mirror real-life situations.

The trainee's manual is organized into Learning Outcomes, which is broken down into indicative content that includes both theoretical and practical activities. It provides detailed information on the key competencies required for each learning outcome, along with the objectives to be achieved.

As a trainee, you will start by addressing questions related to the activities, which are designed to foster critical thinking and guide you towards practical applications in the labor market. The manual also provides essential information, including learning hours, required materials, and key tasks to complete throughout the learning process.

All activities included in this training manual are designed to facilitate both individual and group work. After completing the activities, you will conduct a formative assessment, referred to as the end learning outcome assessment. Ensure that you thoroughly review the key readings and the 'Points to Remember' section.

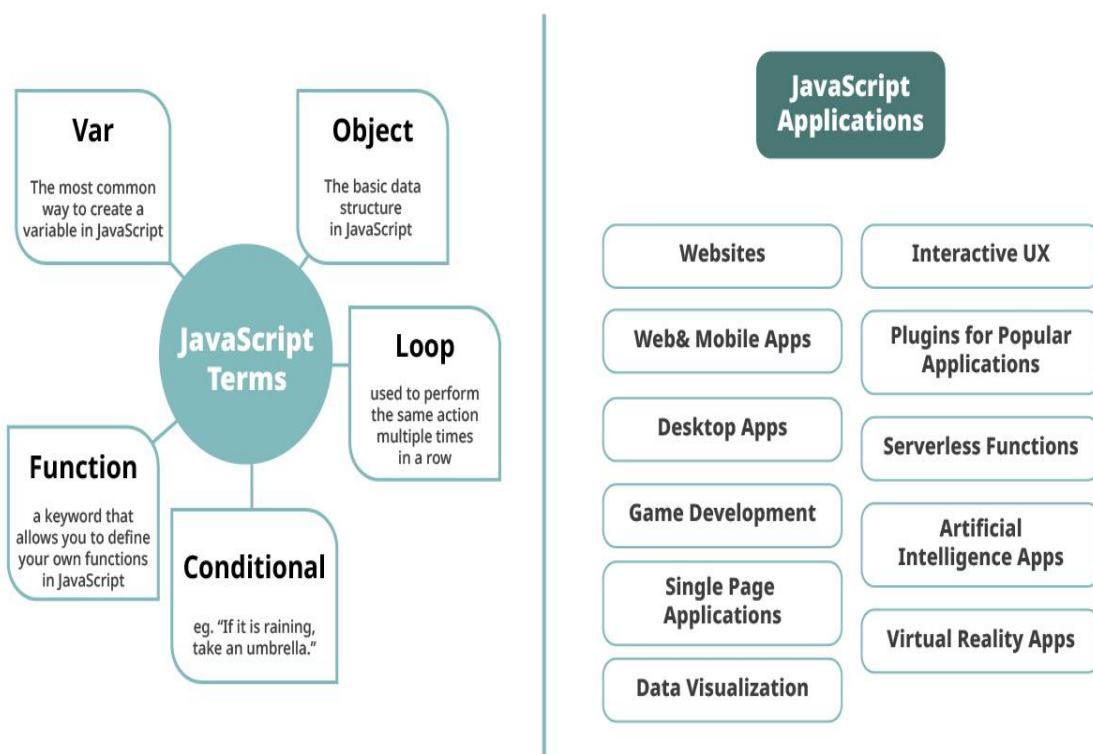
## **MODULE CODE AND TITLE: SWDJF301 JAVASCRIPT FUNDAMENTALS**

**Learning Outcome 1:** Apply JavaScript Basic Concepts

**Learning Outcome 2:** Manipulate Data with JavaScript

**Learning Outcome 3:** Apply JavaScript in Project

## Learning Outcome 1 Apply JavaScript Basic Concepts



## Indicative contents

- 1.1 Introduction to JavaScript**
- 1.2 Integration of JavaScript to HTML**
- 1.3 Use of variables in JavaScript**
- 1.4 Use of data types in JavaScript**
- 1.5 Use of operators in JavaScript**

### Key Competencies for Learning Outcome 1 : Apply JavaScript Basic Concepts

Knowledge	Skills	Attitudes
<ul style="list-style-type: none"><li>• Description of JavaScript</li><li>• Explanation of Variable and their scope</li><li>• Description of JavaScript Data Types</li><li>• Description of JavaScript Operators</li></ul>	<ul style="list-style-type: none"><li>• Installing vs code and node js.</li><li>• Integrating JavaScript in HTML</li><li>• Declaring and initializing variables in JavaScript</li><li>• Using data types in JavaScript program</li><li>• Using operators in JavaScript</li><li>• Applying operators in JavaScript</li></ul>	<ul style="list-style-type: none"><li>• Being Problem solver</li><li>• Being Team worker</li><li>• Being a critical thinker</li><li>• Being Innovative</li><li>• Having Creativity</li><li>• Being confident</li></ul>



**Duration: 30 hrs**

**Learning outcome 1 objectives:**



By the end of the learning outcome, the trainees will be able to:

1. Describe correctly JavaScript key concepts as used in programming.
2. Explain properly JavaScript variables and their scope based on task.
3. Use correctly JavaScript data types based on variables.
4. Integrate properly JavaScript in HTML based on project structure.
5. Install effectively vs code and node js in accordance with JavaScript standard.
6. Use properly operators in JavaScript program.



**Resources**

<b>Equipment</b>	<b>Tools</b>	<b>Materials</b>
<ul style="list-style-type: none"><li>• Computer</li><li>• Projector</li><li>• White board</li></ul>	<ul style="list-style-type: none"><li>• Text editor</li><li>• Node js</li><li>• Browser</li></ul>	<ul style="list-style-type: none"><li>• Internet</li></ul>



## Indicative content 1.1: Introduction to JavaScript



Duration: 8 hrs



### Theoretical Activity 1.1.1 Description of JavaScript overview



#### Tasks:

1. You are requested to answer the following questions related to the description of JavaScript.
  - i. What do you understand about the following JavaScript key concepts?
    - A. Variable
    - B. Data Types
    - C. Values
    - D. Operators
    - E. Expressions
    - F. Keywords
    - G. Comments
  - ii. What are some applications of JavaScript?
  - iii. Outline most known JavaScript libraries.
  - iv. What are some JavaScript frameworks used for web development?
2. Provide the answer for the asked questions and write them on papers.
3. Present the findings/answers to the whole class.
4. For more clarification, read the key readings 1.1.1 and ask questions where necessary.



#### Key readings 1.1.1: Description of JavaScript overview

##### Description of JavaScript overview

###### a. Definition of JAVASCRIPT

JavaScript is a dynamic programming language that's used for web development, in web applications, for game development, and lots more. It allows you to implement dynamic features on web pages that cannot be done with only HTML and CSS.

###### b. Application of Javascript

JavaScript is widely used for building websites and web applications. Let's discuss some practical applications of JavaScript in various segments.

###### 1. Web Development

JavaScript is a scripting language used to develop web pages.

Some famous websites built by the use of JavaScript are Google, YouTube, Facebook, Wikipedia, Yahoo, Amazon, eBay, Twitter, and LinkedIn, to name a few.

## **2. Presentations**

A very popular application of JavaScript is to create interactive presentations as websites. The **RevealJs** and **BespokeJs** libraries can be used to generate web-based slide decks using HTML.

## **3. Server Applications**

JavaScript is also used to write server-side software through Node.js open-source runtime environment. Developers can write, test and debug code for fast and scalable network applications.

## **4. Web Servers**

Node.js is a powerful JavaScript runtime that enables developers to build scalable and efficient web servers

## **5. Games**

Creating games on the web is another important one among applications of JavaScript. The combination of JavaScript and HTML5 plays a major role in games development using JS. The EaselJS library provides rich graphics for games.

## **6. Art**

A recent feature of HTML5 in JavaScript is the canvas element, which allows drawing 2D and 3D graphics easily on a web page.

## **7. Smartwatch Apps**

Pebble.js is a JavaScript framework by Pebble, allowing developers to create applications for Pebble watches using JavaScript. Create a smartwatch app with simple JavaScript code.

## **8. Mobile Apps**

One of the most powerful applications of JavaScript is to create apps for non-web contexts, meaning for things, not on the Internet. With the use of mobile devices at an all-time high, JavaScript frameworks have been designed to facilitate mobile app development across various platforms like IOS, Android, and Windows.

React Native framework allows cross-platform mobile app building, where developers can use a universal front end for Android and IOS platforms.

## **9. Flying Robots**

Last but not least, you can use JavaScript to program a flying robot. With the Node.js ecosystem, users can control numerous small robots, creative maker projects, and IoT devices.

c. **VS Code & node.js**

**Visual Studio Code** is a free, lightweight but powerful source code editor that runs on your desktop and on the web and is available for Windows, macOS, Linux, and Raspberry Pi OS.

Visual Studio Code(VS Code) has support for many languages, including Python, Java, C++, JavaScript, and more

**Node.js** (Node) is an open source, cross-platform runtime environment for executing JavaScript code.

Node is used extensively for server-side programming, making it possible for developers to use JavaScript for client-side and server-side code without needing to learn an additional language.

d. **JAVASCRIPT key concepts**

- **Variable**

A JavaScript variable is simply a name of a storage location.

**OTE:** Variables are classified into Global variables and Local variables based on their scope.

The main difference between Global and local variables is that global variables can be accessed globally in the entire program, whereas local variables can be accessed only within the function or block in which they are defined.

- **JavaScript Identifiers**

An identifier is a sequence of characters in the code that identifies a variable, function, or property.

An identifier is simply a name. In JavaScript, identifiers are used to name constants, variables, properties, functions, and classes and to provide labels for certain loops in JavaScript code.

- **Data Types**

Data types describe the different types or kinds of data that we're going to be working with and storing in variables.

JavaScript provides different data types to hold different types of values. There are two types of data types in JavaScript and these are **Primitive data type and Non-primitive (reference) data type**

- **Primitive data types**

The predefined data types provided by JavaScript language are known as primitive data types. Primitive data types are also known as in-built data types. There are five types of primitive data types in JavaScript. They are as follows:

Data Type	Description
String	represents sequence of characters e.g. "hello"
Number	represents numeric values e.g. 100
Boolean	represents boolean value either false or true
Undefined	represents undefined value
Null	represents null i.e. no value at all

### **The following examples illustrate the use primitive data types in JavaScript**

**1. Number:** Number data type in JavaScript can be used to hold decimal values as well as values without decimals.

**Examples:**

- a) let x = 250;
- let y = 40.5;

**String:** The string data type in JavaScript represents a sequence of characters that are surrounded by single or double quotes.

**Example:**

```
let str = 'Hello';
```

**3. Undefined:** The meaning of undefined is ‘value is not assigned’.

**4. Boolean:** The Boolean data type can accept only two values i.e. true and false.

**5. Null:** This data type can hold only one possible value that is null.

**Example:**

```
let x = null;
```

– **Non-primitive data types**

These data types that are derived from primitive data types of the JavaScript language. There are also known as **derived** data types or **reference** data types.

The non-primitive data types are as follows:

Data Type	Description
Object	represents instance through which we can access members
Array	represents group of similar values

Re gEx p	represents regular expression
----------------	-------------------------------

**Examples below explain the use of non-primitive data types in program**

**1. Object:** Object in JavaScript is an entity having properties and methods. Everything is an object in JavaScript.

How to create an object in JavaScript:

- **Using Constructor Function to define an object:**

// Create an empty generic object

var obj = new Object();

Create a user defined object

var mycar = new Car();

- **Using Literal notations to define an object:**

An empty object

var square = {};

Here a and b are keys and 20 and 30 are values

var circle = {a: 20, b: 30};

**2. Array:** With the help of an array, we can store more than one element under a single name.

**Ways to declare a single dimensional array:**

Call it with no arguments

var a = new Array();

Call it with single numeric argument

var b = new Array(10);

Explicitly specify two or more array elements

var d = new Array(1, 2, 3, "Hello");

- **Values**

A value is the representation of some entity that can be manipulated by a program. The members of a type are the values of that type. The "value of a variable" is given by the corresponding mapping in the environment.

JavaScript values are the values that comprise values like Booleans, Strings, arrays, Numbers, etc.

- **Operators**

In JavaScript, an operator is a special symbol used to perform operations on operands (values and variables). For example, 2 + 3.

- **Expressions**

JavaScript's expression is a valid set of literals, variables, operators, and expressions that evaluate to a single value.

- **Keywords**

In JavaScript, keywords are reserved words that have a specific purpose (meaning) and are already defined in the language.

A list of all the reserved words in JavaScript are given in the following table. They cannot be used as JavaScript variables, functions, methods, loop labels, or any object names.

abstract	Else	instanceof	switch
boolean	Enum	Int	synchronised
break	Export	interface	this
byte	Extends	Long	throw
case	False	Native	throws
catch	Final	New	transient
char	Finally	null	true
class	Float	package	try
const	For	private	typeof
continue	Function	protected	var
debugger	Goto	public	void
default	If	return	volatile
delete	Implements	short	while
do	Import	static	with
double	In	super	

- **Comments**

The JavaScript comments can be used to explain JavaScript code, and to make it more readable.

It is used to add information about the code, warnings or suggestions so that end users can easily interpret the code. The comments are ignored by the JavaScript engine.

#### **Advantages of JavaScript comments**

There are mainly two advantages of JavaScript comments.

- **To make code easy to understand**

It can be used to elaborate the code so that end users can easily understand the code.

- **To avoid the unnecessary code**

It can also be used to avoid the code being executed. Sometimes, we add the code to perform some action. But after sometime, there may be a need to disable the code. In such cases, it is better to use comments.

### **Types of JavaScript Comments**

There are two types of comments in JavaScript.

- Single-line Comment
- Multi-line Comment

#### **1. Single Line Comments**

Single line comments start with //.

Any text between // and the end of the line will be ignored by JavaScript (will not be executed).

##### **Example:**

```
//Declaration of a variable x  
var x;
```

#### **2. Multi-line Comments**

Multi-line comments start with /\* and end with \*/. Any text between /\* and \*/ will be ignored by JavaScript.

##### **Example:**

```
result=6*5;  
  
/* multiply two numbers and store the 30  
output in a variable called result*/
```

#### **e. JavaScript libraries**

JavaScript libraries is a file that contains a set of prewritten functions or codes that you can repeatedly use while executing JavaScript tasks.

We have 3 types of JavaScript libraries to discuss on:

- **React Javascript**

React (also known as React.js or ReactJS) is a free and open-source front-end JavaScript library for building user interfaces based on UI components.

- **JQuery**

It's a library of JavaScript functions that make it easy for web page developers to do common tasks like manipulating the webpage, responding to user events, getting data from their servers, building effects and animations, and much more.

- **Three JavaScript**

Three.js is a cross-browser JavaScript library and application programming interface (API) used to create and display animated 3D computer graphics in a web.

- f. **JavaScript frameworks**

A JavaScript framework is a set of JavaScript code libraries that provide pre-written code for everyday programming tasks to a web developer.

- **Vue JavaScript**

Vue.js is an open-source progressive JavaScript framework used to develop interactive web user interfaces and single-page applications (SPAs).

- **Angular JavaScript**

AngularJS is a client-side JavaScript MVC framework to develop a dynamic web application.

- **Express JavaScript**

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.

- g. **JavaScript runtime environment**

The JavaScript runtime environment provides access to built-in libraries and objects that are available to a program so that it can interact with the outside world and make the code work.

**a) Node JavaScript:** is an open-source, server-side JavaScript runtime environment that allows developers to execute JavaScript code outside of a web browser. It provides a platform for building scalable, networked applications and is widely used for server-side and command-line scripting.

**b) V8 Engine:** is an open-source JavaScript engine developed by Google. It is primarily used to execute JavaScript code in web browsers, but it is also the underlying runtime for the Node.js server-side JavaScript environment.

The V8 engine is written in C++ and is designed to optimize the execution of JavaScript code for performance and efficiency.

Simply, A JavaScript engine is a software component that executes JavaScript code.

- h. **JavaScript versions**

JavaScript was invented by Brendan Eich in 1995, and became an ECMA standard in 1997.

ECMAScript is the official name of the language.

ECMAScript versions have been abbreviated to ES1, ES2, ES3, ES5, and ES6.

Since 2016, versions are named by year (ECMAScript 2016, 2017, 2018, 2019, 2020).



## Practical Activity 1.1.2: Installation of node.js and VS Code



### Task:

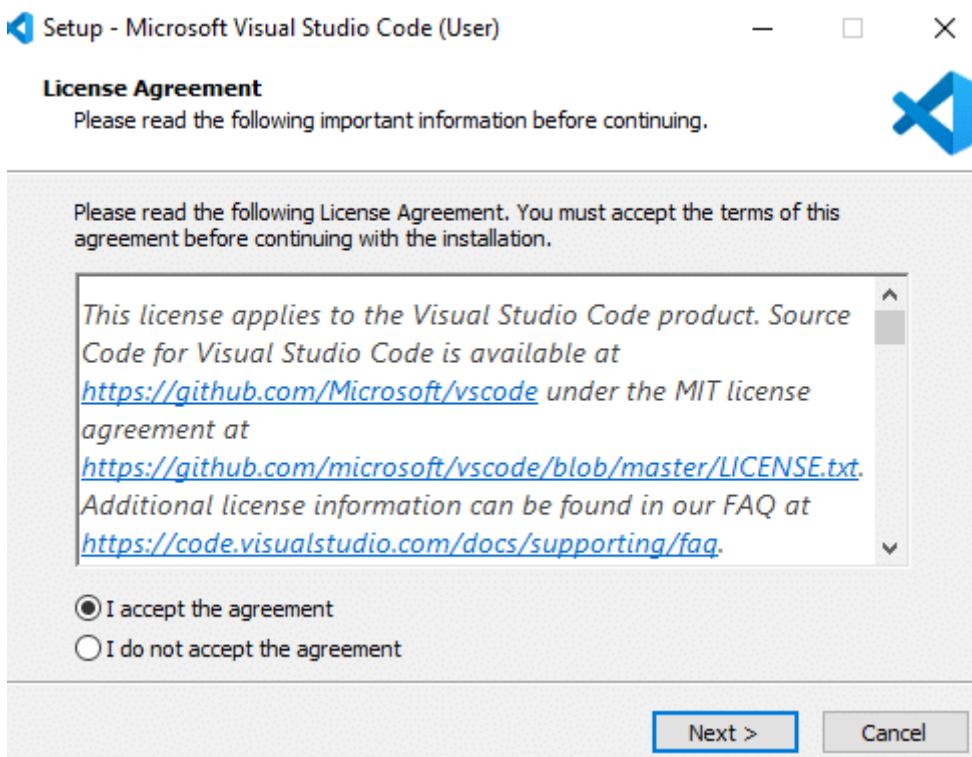
- 1: As a software developer, you are asked to install node.js and Vs Code in the computer.
- 2: Read the key reading 1.1.2. In trainee manual about installation of VS Code and Node.js
- 3: Referring to the steps provided, install node.js and Vs Code.
- 4: Open the installed software to verify whether has been installed correctly.



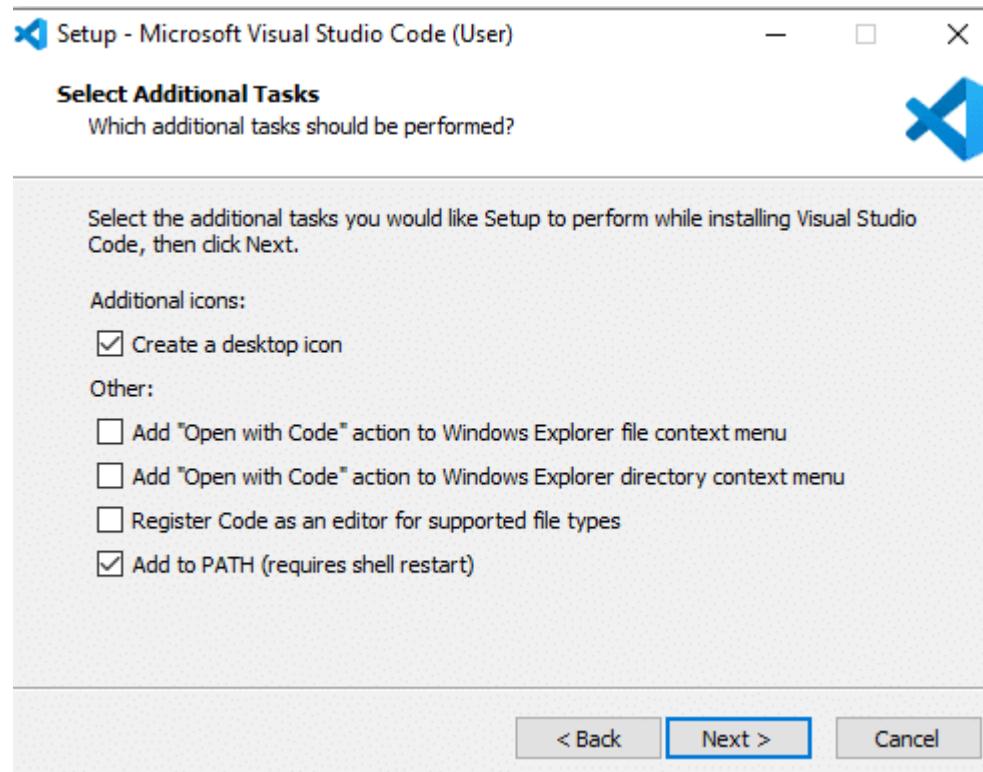
### Key readings 1.1.2

#### Steps to install VS code on windows

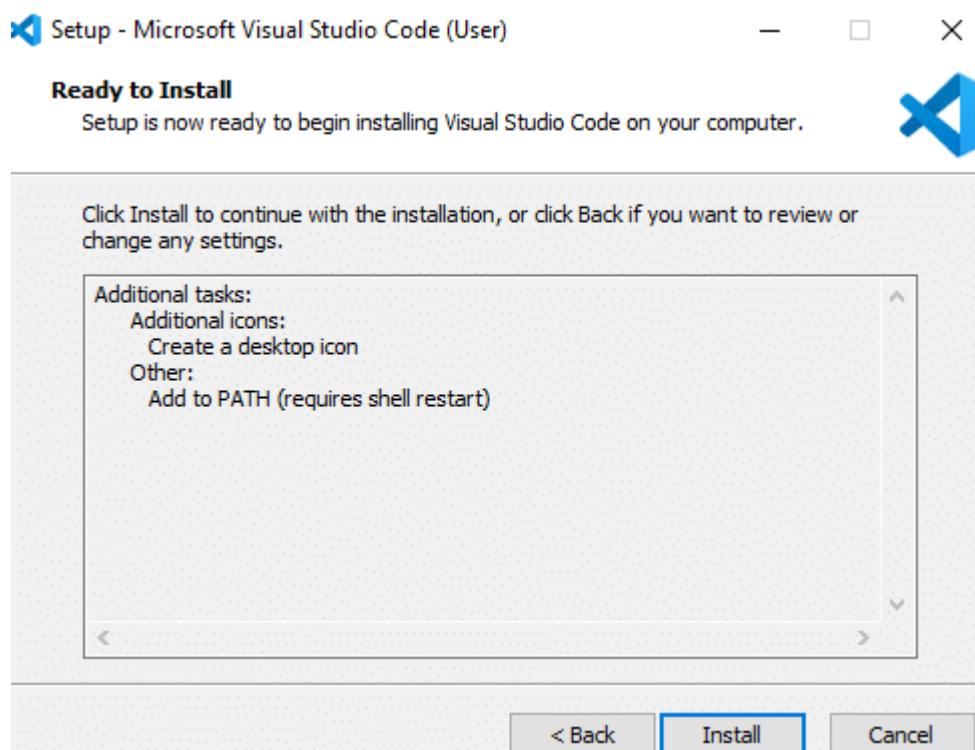
1. Locate VScode setup file (HDD, Flash disk, CD/DVD)
2. Run the setup file.
3. Then, accept the agreement and click on next.



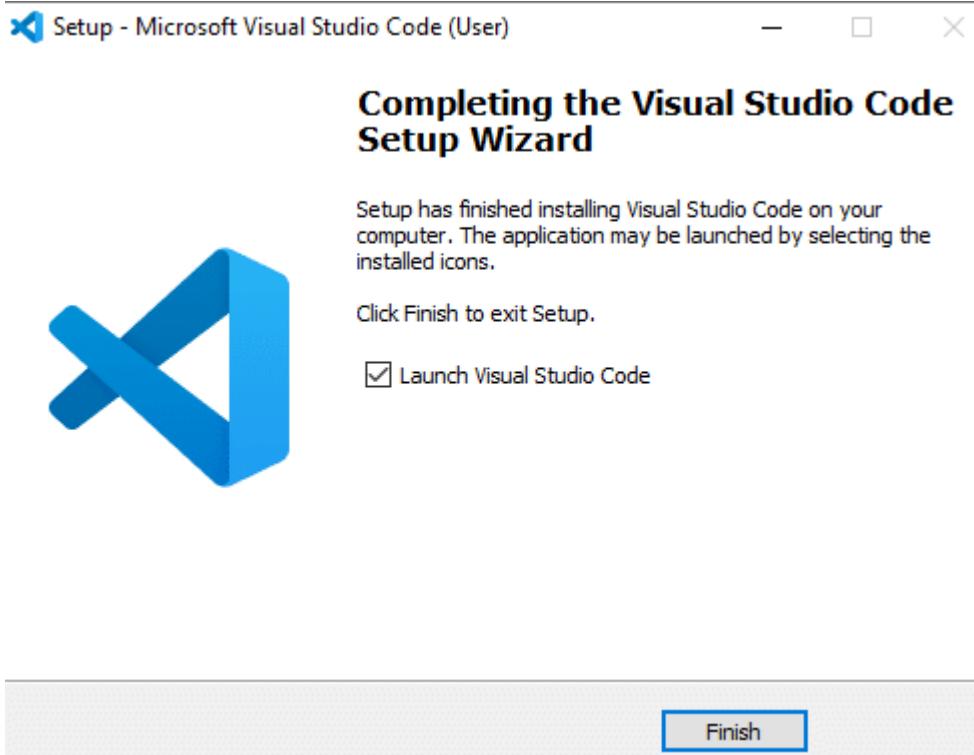
4. Click on "create a desktop icon" so that it can be accessed from desktop and click on Next.



After that, click on the install button



5. Finally, after installation completes, click on the finish button, and the visual studio code will open.



### Steps to install node.js on windows

1. Locate node.js setup file (HDD, Flash disk, CD/DVD)
2. Run setup file (Node.js installer.)

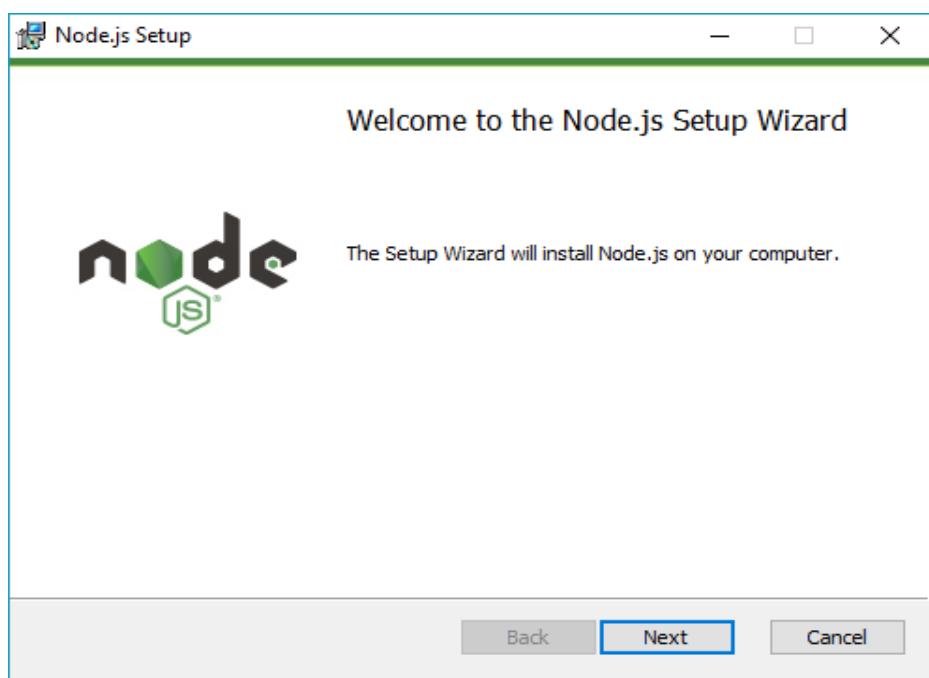
Now you need to install the node.js installer on your PC. You need to follow the following steps for the Node.js to be installed:

- Double click on the installer.

The Node.js Setup wizard will open.

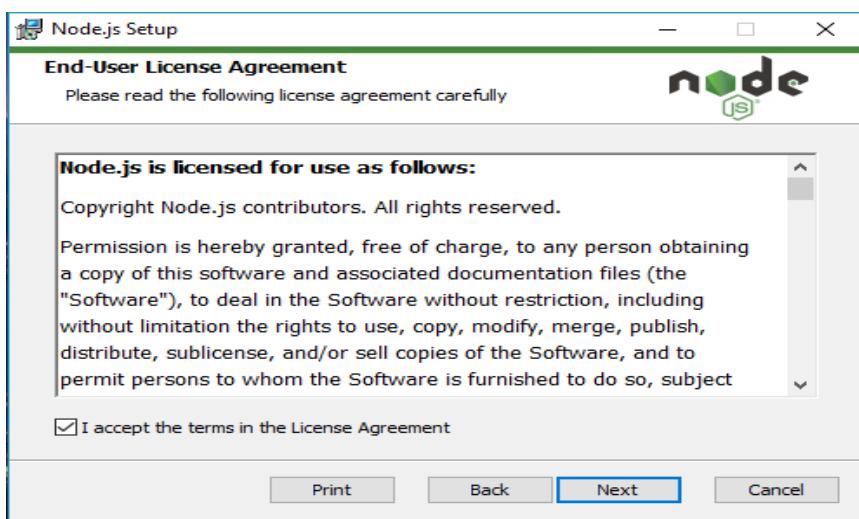
- Welcome To Node.js Setup Wizard.

Select “Next”

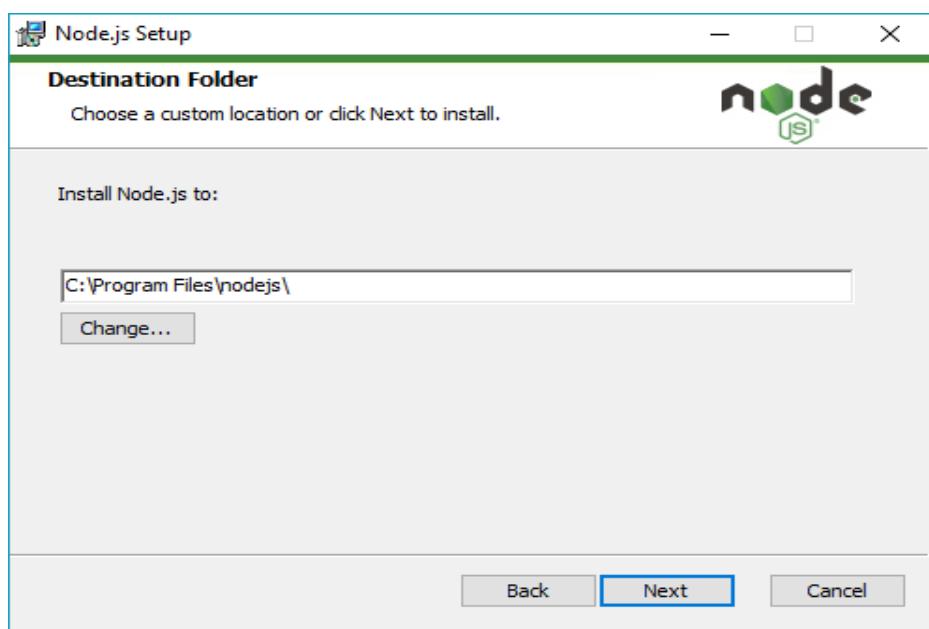


3. After clicking "Next", the End-User Licence Agreement (EULA) will open.

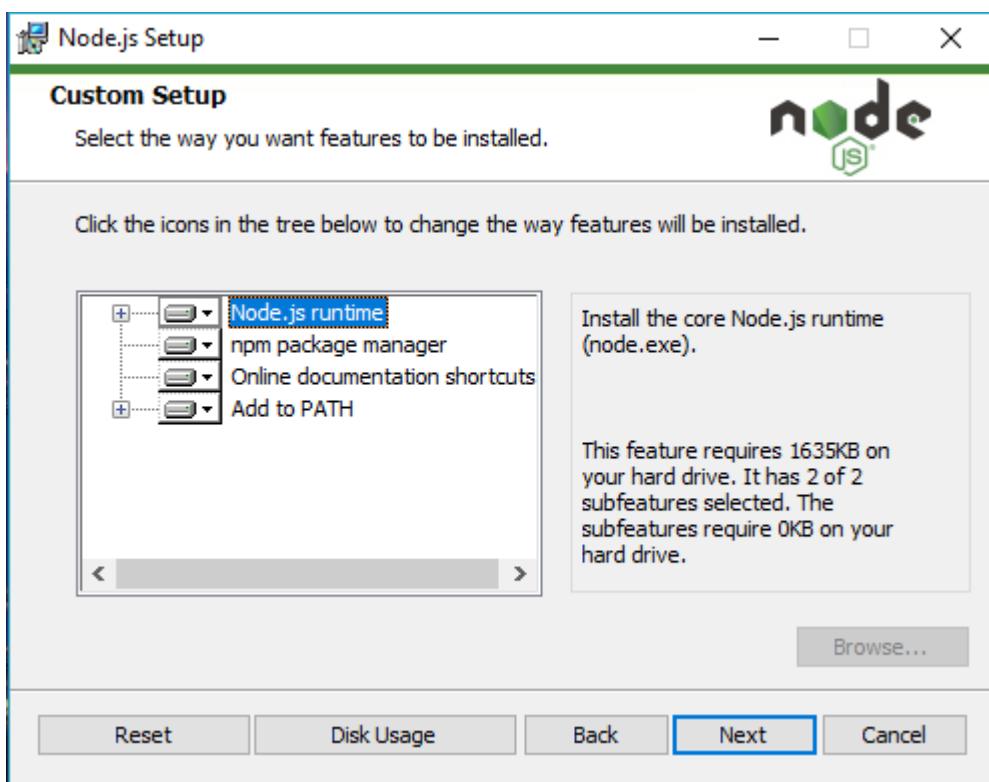
Check "I accept the terms in the Licence Agreement" and click "Next"



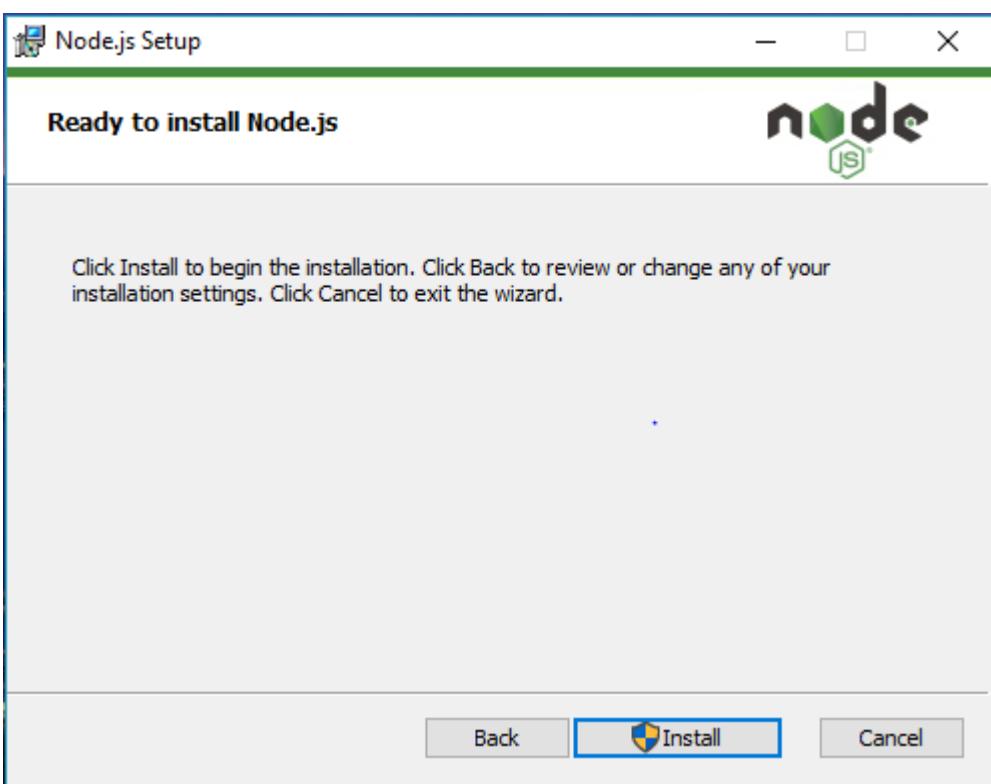
4. Set the Destination Folder where you want to install Node.js & Select "Next"



Select "Next"

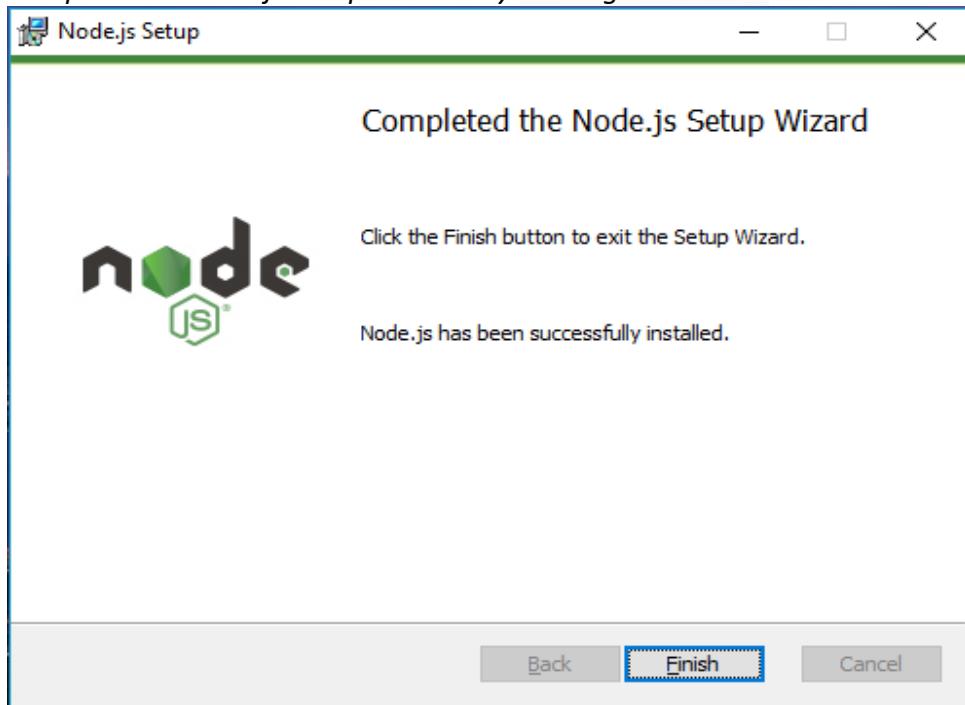


*The installer may prompt you to "install tools for native modules".*



*Do not close or cancel the installer until the install is complete*

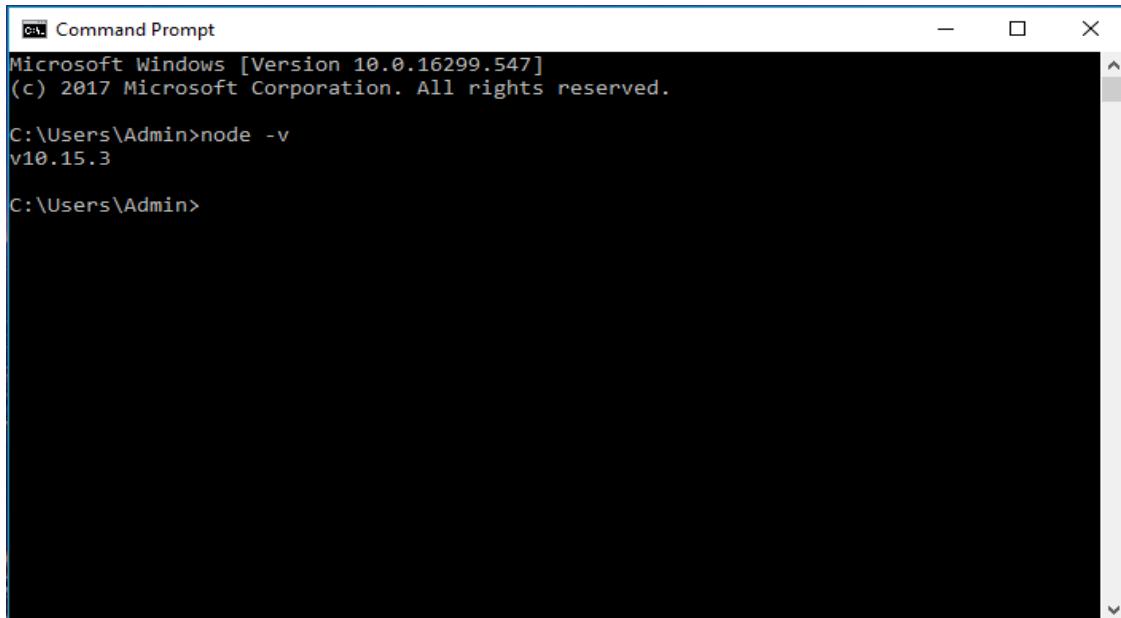
5. Complete the Node.js Setup Wizard. By Clicking on “Finish”



**Verify that Node.js was properly installed or not.**

To check that node.js was completely installed on your system or not, you can run the following command in your command prompt or Windows Powershell and test it:

```
-C:\Users\Admin> node -v
```



The screenshot shows a Windows Command Prompt window titled "Command Prompt". The window displays the following text:  
Microsoft Windows [Version 10.0.16299.547]  
(c) 2017 Microsoft Corporation. All rights reserved.  
C:\Users\Admin>node -v  
v10.15.3  
C:\Users\Admin>



### Points to Remember

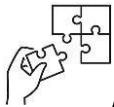
- **JavaScript** is a dynamic programming language that's used for web development, in web applications, for game development, and lots more.
- A **JavaScript variable** is simply a name of a storage location, each variable needs a data type to describe the kind of data to be stored in a variable. These data types may be categorised as **Primitive data type** and **Non-primitive data type**. While performing operations on operands (values and variables) we need a special symbol called an operator.
- Installing Node.js is a straightforward process that allows developers to leverage the power of JavaScript on the server-side.

Here are some key steps about installing Node.js:

1. Locate node.js setup file
  2. Run setup file
  3. Complete the Node.js Setup Wizard
- Installing Visual Studio Code (VS Code) is a simple process that allows developers to have a powerful and versatile code editor for their programming needs.

Here are some key steps about installing VS Code:

1. Locate VS code setup file
2. Run the setup file
3. Finally, after installation completes, click on the finish button.



### **Application of learning 1.1.**

BGS Ltd, is a young software development company that develops web applications. You are tasked by BGS Ltd to help them to install Microsoft Visual Studio Code and Node.JS so that their developers will be able to code easily and run JavaScript Applications.



## Indicative content 1.2: Integration of JavaScript to HTML



Duration: 5 hrs



### Theoretical Activity 1.2.1: Description of JavaScript integration to HTML



#### Tasks:

- 1: You are requested to answer the following questions related to the JavaScript integration to HTML:
  - i. What do you understand about the term “script tag as “used in JavaScript:
  - ii. Shortly, explain the following ways of integrating JavaScript to HTML.
    - a. Referencing HTML
    - b. External JavaScript reference (CDN)
    - c. External java script
- 2: Provide the answer for the asked questions and write them on papers
- 3: Present the findings to the trainer and the whole class
- 4: For more clarification, read the key readings 1.2.1 and ask questions where necessary.



#### Key readings 1.2.1.:

When working with files for the web, JavaScript needs to be loaded and run alongside HTML markup. This can be done either inline within an HTML document or in a separate file that the browser will download alongside the HTML document.

#### ✓ Referencing HTML to JavaScript

Referencing HTML elements in JavaScript refers to the process of accessing and manipulating HTML elements within JavaScript code. It allows developers to interact with the structure, content, and behavior of HTML elements dynamically.

To include an external JavaScript file, we can use the `script` tag with the attribute `src`. The `src` attribute specifies the URL of an external script file.

If you want to run the same JavaScript on several pages in a web site, you should create an external JavaScript file, instead of writing the same script over and over again. Save the script file with a `.js` extension, and then refer to it using the `src` attribute in the `<script>` tag. **Note:** The external script file cannot contain the `<script>` tag.

## ✓ <script> tag

The `<script>` is an HTML element used to embed or reference JavaScript code within an HTML document. It allows developers to include JavaScript code directly within the HTML file or link to an external JavaScript file.

The <script> tag is used to embed a client-side script (JavaScript).

### Example:

```
<script type="text/javascript">  
document.write("PHPTPOINT is the place to study javascript easily"); </script>
```

Using the script tag signifies that we are using JavaScript

## ✓ External JavaScript

External JavaScript refers to JavaScript code that is stored in a separate file with a `.js` extension and linked to an HTML document using the `<script>` tag's `src` attribute. Instead of embedding the JavaScript code directly within the HTML file, it is stored in an external file, which is then referenced by the HTML document.

With JavaScript, an external JavaScript file can be created and can easily be embedded into many HTML pages. Since a single JavaScript file can be used in various HTML pages hence, it provides code reusability.

In order to save a JavaScript file, .js extension must be used. To increase the speed of the webpages, it is recommended to embed the entire JavaScript file into a single file.

## ✓ Using external Javascript reference (CDN)

Using an external JavaScript reference through a Content Delivery Network (CDN) involves linking to a JavaScript file hosted on a remote server or network of servers. CDNs are designed to deliver content, including JavaScript files, to users efficiently and reliably. Instead of hosting the JavaScript file on your own server, you can reference a CDN-hosted file in your HTML document.

### What is a CDN?

A CDN is a Content Delivery Network. These are file hosting services for multiple versions of common libraries.

You can use external JavaScript references, often referred to as Content Delivery Network (CDN) links, to include JavaScript libraries or scripts in your HTML documents.

This is a common practice to save bandwidth and improve loading times. Here's how you can do it:

Find the CDN link for the JavaScript library you want to include. Many popular libraries like jQuery, Bootstrap, or React have CDN links available. These links are hosted on servers provided by various content delivery networks.

In your HTML document, add a <script> tag with the src attribute set to the CDN link.

By using CDN links, you can easily include external JavaScript libraries in your web pages without having to host the library files on your server, and your users will benefit from faster loading times since CDNs are designed for efficient content delivery.

### ✓ **JavaScript output**

JavaScript output refers to the process of displaying or presenting information, data, or results generated by JavaScript code. JavaScript provides various methods and techniques for outputting data, allowing developers to communicate with users or display information on web pages.

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 window.alert () .
- Writing into the browser console, using console.log () .



### **Practical Activity 1.2.2: Integration of JavaScript to HTML**



#### **Task:**

- 1: You are requested to go to the computer lab to Integrate JavaScript to HTML.
- 2: Read the key readings 1.2.2 in trainee manual about integration of JavaScript to HTML
- 3: Integrate JavaScript to HTML by using the following different ways: Referencing HTML to JavaScript, using <script> tag, using external JavaScript, using external JavaScript reference (CDN) and generate the output.
- 4: Present your work to the whole classroom or trainer



## Key readings 1.2.2

To integrate JavaScript into HTML by referencing HTML elements in JavaScript, you can follow these steps:

**1. Identify the HTML element(s):**

Determine which HTML element(s) you want to reference in your JavaScript code. This could be an element with a specific ID, class, tag name, or any other attribute that uniquely identifies the element(s).

**2. Choose a referencing method:** Select the appropriate method for referencing the HTML element(s) in JavaScript.

**3. Write the JavaScript code:** In your JavaScript code, use the chosen referencing method to access the desired HTML element(s).

**4. Perform desired operations:** Once you have referenced the HTML element(s) in JavaScript, you can perform various operations on them. This could include modifying their content, changing their attributes, adding event listeners, manipulating their styling, or interacting with their behavior.

**5. Ensure proper execution order:** Make sure that your JavaScript code is executed after the HTML elements have been loaded.

**6. Test and debug:** Test your JavaScript code to ensure that it correctly references and interacts with the desired HTML element(s).

By following these steps, you can effectively reference HTML elements in JavaScript and integrate them into your HTML document. This allows you to manipulate and interact with the elements dynamically, creating interactive and responsive web pages.

**Example:**

✓ **Referencing HTML to JavaScript**

1. Open your HTML file in a text editor or an integrated development environment (IDE).

2. Locate the part of the HTML file where you want to include your JavaScript code. This can be within the **<head>** section or the **<body>** section, depending on your requirements.

3. To reference an external JavaScript file, you can use the **<script>** tag. Place the following code within the **<head>** section or at the end of the **<body>** section:

```
<script src="path/to/your/javascript-file.js"></script>
```

Replace `"path/to/your/javascript-file.js"` with the actual file path of your JavaScript file. Make sure to provide the correct file path relative to your HTML file.

4. If you want to include JavaScript code directly within your HTML file, you can use the

<script> tag and place the code between the opening and closing tags.

**For example:**

```
<script>  
    // Your JavaScript code goes here  
</script>
```

5. You can also use the **type** attribute within the <script> tag to specify the scripting language. However, for JavaScript, this attribute is not necessary as JavaScript is the default scripting language.

**For example:**

```
<script type="text/javascript">  
    // Your JavaScript code goes here  
</script>
```

6. Save your HTML file with the changes.

By following these steps, you have successfully integrated JavaScript into your HTML file using referencing HTML tags.

The JavaScript code will be executed when the HTML file is loaded in a web browser, allowing you to add interactivity and dynamic functionality to your web page.

**Example:**

```
<!DOCTYPE html>  
<html>  
<body>  
<h1>The script src attribute</h1>  
<script src="demo_script_src.js">  
</script>  
</body>  
</html>
```

### Using <script> tags

- To integrate JavaScript into HTML using <script> tags, you can follow these steps:

**1. Create an HTML file:** Start by creating an HTML file using a text editor or an integrated development environment (IDE).

**2. Set up the HTML structure:** Define the structure of your HTML document by adding HTML tags such as <html>, <head>, and <body>. This is where you'll integrate your

JavaScript code.

**3. Add the <script> tag:** Inside the <body> or <head> section of your HTML file, add the <script> tag to include your JavaScript code. You can either write the JavaScript directly within the <script> tags or reference an external JavaScript file using the **src** attribute.

**Inline JavaScript:** To write JavaScript code directly within the `<script>` tags, place your code between the opening and closing `<script>` tags. For example:

```
<script>  
// Your JavaScript code goes here  
</script>
```

**External JavaScript file:** To reference an external JavaScript file, use the `src` attribute within the `<script>` tag and provide the path or URL to the JavaScript file. For example:

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

**4. Place the <script> tag appropriately:** Decide where to place the <script> tag based on your requirements.

Placing it in the <head> section allows the JavaScript code to load before the HTML content, while placing it at the end of the `<body>` section ensures that the HTML content is loaded before executing the JavaScript code.

**5. Test and debug:** Save your HTML file and open it in a web browser. Use the browser's developer tools to check for any errors in the JavaScript code and debug as needed. This will help ensure that your JavaScript code is properly integrated and functioning as expected.

By following these steps, you can integrate JavaScript into HTML using `<script>` tags and leverage its capabilities to create interactive and dynamic web pages.

Here is an example to understand this from an initial level:

#### **JavaScript Syntax: Code between the head tag**

##### **Example1.**

```
<!DOCTYPE html>  
<html>  
<head>  
<p id="demo"></p>  
<script>  
document.getElementById ("demo").innerHTML = "Paragraph changed.";  
</script>  
</head>  
<body>
```

```
<h2>Demo JavaScript in Head</h2>
</body>
</html>
```

**Example2.**

```
<!DOCTYPE html >
<html>
<head>
<title> page title</title>
<script>
document.write("Welcome to Javatpoint");
</script>
</head>
<body>
<p>In this example we saw how to add JavaScript in the head section </p>
</body>
</html>
```

**JavaScript Syntax: Code between the Body Tags**

**Example1.**

```
<html>
<body>
<script type="text/javascript">
document.write("JavaScript is a simple language for javatpoint learners");
</script>
</body>
</html>
```

**Example2.**

```
<!DOCTYPE html >
<html>
<head>
<title> page title</title>
</head>
<body>
<script>
document.write("Welcome to Javatpoint");
</script>
<p> In this example we saw how to add JavaScript in the body section </p>
</body>
</html>
```

## Using external JavaScript

To integrate external JavaScript into HTML, you can follow these steps:

**1. Choose a JavaScript file:** Select the external JavaScript file that you want to integrate into your HTML document. This could be a file you have written yourself or a library/framework file from a trusted source.

**2. Obtain the file's URL or path:** Ensure that you have the URL or local path to the external JavaScript file. If the file is hosted on a Content Delivery Network (CDN), you can usually [find the URL from the CDN provider's documentation](#).

**3. Link to the JavaScript file:** In your HTML file, add a `<script>` tag to link to the external JavaScript file. Use the **src** attribute and provide the URL or path to the JavaScript file. For

**Example:**

```
<script src="path/to/external.js"> </script>
```

**4. Place the `<script>` tag appropriately:** Decide where to place the `<script>` tag based on your requirements. It is common to place the `<script>` tag just before the closing `</body>` tag. This ensures that the JavaScript file is loaded after the HTML content, improving page loading performance. However, you can also place it in the `<head>` section if necessary.

**5. Test and debug:** Save your HTML file and open it in a web browser. Use the browser's developer tools to check for any errors in the JavaScript code and debug as needed. Ensure that the external JavaScript file is being loaded correctly and that it functions as expected.

**6. Utilize the JavaScript file:** Once the external JavaScript file is successfully integrated, you can use its functions, methods, or variables within your HTML document. Follow the documentation or guidelines provided by the JavaScript file's source to utilize its features effectively.

By following these steps, you can integrate external JavaScript into your HTML document, allowing you to leverage the functionality and capabilities provided by the JavaScript file.

**Here is a simple example:**

Let's include the JavaScript file into the html page. It calls the JavaScript function on button click.

**index.html file**

```
<html>
<head>
```

```
<script type="text/javascript" src="message.js"></script>
</head>
<body>
<p>Welcome to JavaScript</p>
<form>
<input type="button" value="click" onclick="msg()"/>
</form>
</body>
</html>
```

### Using external JavaScript reference (CDN)

To integrate JavaScript into HTML using an external JavaScript reference (CDN), you can follow these steps:

- 1. Choose a JavaScript library or framework:** Select the JavaScript library or framework that you want to integrate into your HTML document. Common examples include jQuery, React, Vue.js, or Bootstrap.
- 2. Find the CDN URL:** Locate the CDN (Content Delivery Network) URL for the chosen JavaScript library or framework. CDN providers like Google, Microsoft, or Cloudflare often host popular JavaScript libraries and offer CDN URLs for easy integration.
- 3. Link to the CDN URL:** In your HTML file, add a `<script>` tag to link to the CDN URL. Use the `src` attribute and provide the CDN URL for the JavaScript library or framework.

#### For Example:

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.6.0/jquery.min.js">
</script>
```

- 4. Place the `<script>` tag appropriately:** Decide where to place the '`<script>`' tag based on your requirements. It is common to place the `<script>` tag just before the closing `</body>` tag. This ensures that the JavaScript library is loaded after the HTML content, improving page loading performance. However, you can also place it in the `<head>` section if necessary.
- 5. Test and debug:** Save your HTML file and open it in a web browser. Use the browser's developer tools to check for any errors in the JavaScript code and debug as needed. Ensure that the external JavaScript library is being loaded correctly and that it functions as expected.
- 6. Utilize the JavaScript library:** Once the external JavaScript library is successfully integrated, you can use its functions, methods, or components within your HTML document. Follow the documentation or guidelines provided by the JavaScript library's source to utilize its features effectively.

**Here is a simple example:**

```
<!DOCTYPE html>
<html>
<head>
<title>Using CDN for JavaScript</title>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
</head>
<body>
<!-- Your HTML content here -->
</body>
</html>
```

**✓JavaScript output**

**To generate JavaScript output, you can follow these steps:**

1. Open a text editor or an integrated development environment (IDE) to write your JavaScript code.
2. Start by defining the desired output or the logic that generates the output. This could involve performing calculations, manipulating data, or retrieving information from user input.
3. Use JavaScript's built-in functions, operators, and syntax to write the code that generates the desired output. This may include variables, conditional statements (if/else), loops (for/while), arrays, functions, and more.
4. Within your JavaScript code, use the `console.log()` function to output text or values to the console.

**For example:**

```
console.log ("Hello, world!");
```

This will display the text "Hello, world!" in the console when the JavaScript code is executed.

5. Alternatively, you can output the result to the HTML document itself by manipulating the HTML elements using JavaScript. For example, you can use the `document.getElementById()` function to select an HTML element and modify its content.

**Here's an example:**

```
document.getElementById("output").innerHTML = "Hello, world!";
```

In this case, you need to have an HTML element with the `id` attribute set to "output" to display the output.

Open your HTML file in a web browser, or run the JavaScript file using Node.js, and check the console or the HTML output element to view the generated output.

### ★ Using innerHTML

**To generate JavaScript output using the innerHTML property, you can follow these steps:**

**1. Identify the HTML element:** Determine the HTML element where you want to generate the JavaScript output. This could be a `<div>`, `<span>`, `<p>`, or any other element that can contain content.

**2. Reference the HTML element in JavaScript:** Use JavaScript to reference the desired HTML element using methods like `getElementById`, `querySelector`, or any other appropriate method.

**For example:**

```
const outputElement = document.getElementById('output');
```

**3. Generate the JavaScript output:** Use the `innerHTML` property of the referenced HTML element to generate the desired JavaScript output. Assign the desired content, including HTML tags, to the `innerHTML` property.

**For example:**

```
outputElement.innerHTML = 'Hello, <strong>JavaScript!</strong>';
```

**4. Test and observe the output:** Save your HTML file and open it in a web browser. Check the specified HTML element to see the generated JavaScript output. In this case, the output would be "Hello, JavaScript!" with the word "JavaScript" displayed in bold.

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:

**Example**

```
<Html>
<Head>
<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()

To generate JavaScript output using the `document.write()` method, you can follow these steps:

**1. Identify the location for the JavaScript output:** Determine where you want the JavaScript output to appear in your HTML document. This could be within the ``<body>`` section or within a specific HTML element.

**2. Write the JavaScript code:** Write your JavaScript code that will generate the desired output. Use the `document.write ()` method to output content directly to the HTML document.

For example:

```
document.write ("Hello, JavaScript!");
```

**3. Place the JavaScript code appropriately:** Decide where to place the JavaScript code based on your requirements. You can place it directly within a `<script>` tag in the `<head>` section or at the end of the `<body>` section.

**4. Test and observe the output:** Save your HTML file and open it in a web browser. The JavaScript code will be executed, and the output will be displayed at the specified location. In this case, the output would be "Hello, JavaScript!"

By following these steps, you can generate JavaScript output using the `document.write ()` method.

For testing purposes, it is convenient to use `document.write()`:

#### **Example: 1**

```
<!DOCTYPE html>
<html>
<body>
<h1>My First Web Page</h1>
<p>My first paragraph. </p>
<script>
```

```
document.write(5 + 6);
</script>
</body>
</html>
```

**Example: 2**

```
<!DOCTYPE html>
<html>
<body>
<h1>My First Web Page</h1>
<p>My first paragraph.</p>
<button type="button" onclick="document.write(5 + 6)">Try it</button>
</body>
</html>
```

★ Using `window.alert()`

To generate JavaScript output using the `window.alert()` method, you can follow these steps:

**1. Determine the message to display:** Decide on the message or content that you want to display as the JavaScript output using the `window.alert ()` method. This could be a notification, a prompt, or any other information you want to present to the user.

**2. Write the JavaScript code:** Write your JavaScript code that includes the `window.alert ()` method. Provide the desired message as a string parameter within the parentheses. For example:

```
window.alert ("Hello, JavaScript!");
```

**3. Place the JavaScript code appropriately:** Decide where to place the JavaScript code based on your requirements. You can place it directly within a `<script>` tag in the `<head>` section or at the end of the `<body>` section.

**4. Test and observe the output:** Save your HTML file and open it in a web browser. When the JavaScript code is executed, a pop-up alert window will appear displaying the specified message. In this case, the output would be an alert window displaying "Hello, JavaScript!".

**5. Customize the alert message:** You can modify the message within the `window.alert()` method to display different content or variable values. You can concatenate strings or include variables within the message for dynamic output.

You can use an alert box to display data:

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

### ★ Using console.log ()

To generate JavaScript output using the `console.log()` method, you can follow these steps:

**1. Determine the message to display:** Decide on the message or content that you want to display as the JavaScript output using the console.log() method. This could be a string, variable values, or any other information you want to log for debugging or informational purposes.

**2. Write the JavaScript code:** Write your JavaScript code that includes the console.log() method. Provide the desired message or variable as a parameter within the parentheses.

**For example:**

```
console.log("Hello, JavaScript!");
```

**3. Place the JavaScript code appropriately:** Decide where to place the JavaScript code based on your requirements. You can place it directly within a `<script>` tag in the `<head>` section or at the end of the `<body>` section.

**4. Test and observe the output:** Save your HTML file and open it in a web browser. Open the browser's developer tools and navigate to the console tab. When the JavaScript code is executed, the specified message or variable value will be logged to the console. In this case, the output would be a log entry displaying "Hello, JavaScript!".

**5. Customize the log message:** You can modify the message within the console.log() method to display different content or variable values. You can concatenate strings or include variables within the message for dynamic output.

By following these steps, you can generate JavaScript output using the `console.log()` method. This method is commonly used for debugging and informational purposes, allowing you to log messages and variable values to the browser's console for analysis and troubleshooting.

For debugging purposes, you can call the console.log() method in the browser to display

data.

### Example

```
<!DOCTYPE html>
<html>
<body>
<script>
console.log(5 + 6);
</script>
</body>
</html>
```

### ★ JavaScript Print

To generate JavaScript output for printing purposes, you can follow these steps:

**1. Identify the content to be printed:** Determine the specific content that you want to generate as output for printing. This could be text, HTML elements, or a combination of both.

**2. Create a print function:** Write a JavaScript function that will handle the printing process. This function will be responsible for generating the output and initiating the print dialog. **For example:**

```
function printContent() {
    Var content = "Hello, JavaScript!";
    var printWindow = window.open("", '_blank');
    printWindow.document.write(content);
    printWindow.document.close();
    printWindow.print(); }
```

**3. Call the print function:** Invoke the print function when you want to generate the output for printing. You can trigger the function through a button click, a specific event, or any other appropriate mechanism.

**4. Test and observe the output:** Save your HTML file and open it in a web browser. Trigger the print function and observe the output. In this case, a new window will open with the content "Hello, JavaScript!" and the print dialog will appear, allowing the user to print the content.

**5. Customize the content:** Modify the content variable in the print function to include the specific content you want to print. You can dynamically generate the content based on user input, retrieve data from an API, or manipulate the DOM to select specific HTML elements for printing.

**6. Style the printed output:** You can apply CSS styles to the content that will be printed

to control the appearance of the output. This can be done by adding appropriate CSS rules to the content or by linking an external CSS file to the printed document.

By following these steps, you can generate JavaScript output for printing purposes. This allows you to generate specific content and initiate the print dialog to provide a printable version of your web page or application.

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>
```

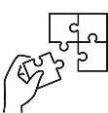


### Points to Remember

- Integrating JavaScript with HTML allows developers to enhance the interactivity and functionality of web pages. When working with files for the web, JavaScript needs to be loaded and run alongside HTML markup. Some ways of integrating JavaScript to HTML are: Using script tag, referencing HTML, external JavaScript reference (CDN) and external JavaScript. To integrate JavaScript into HTML you have to identify the HTML element(s) and write JavaScript codes to perform desired operations
- Generating JavaScript output allows developers to display information, results, or interact with users in various ways.

To generate JavaScript output, you can follow these steps:

1. After defining the desired output
2. Use one of JavaScript's built-in functions to display output.
3. Save your JavaScript file with a ` `.html` extension
4. Open your HTML file in a web browser



### Application of learning 1.2.

You are a web developer working on a project for a local event registration website. The client wants a simple registration form that includes basic validation using JavaScript. Your task is to implement the form and make it interactive.



## Indicative content 1.3: Use of variables in JavaScript



Duration: 5 hrs



### Theoretical Activity 1.3.1: Description of variables in JAVASCRIPT



#### Tasks:

- 1: In small groups, you are requested to answer the following questions related to the description of variables in JavaScript:
  - i. Differentiate declaration and re-declaration of variables.
  - ii. Explain the ways used while declaring a variable.
  - iii. Outline any four rules to be followed while naming variables in JavaScript
  - v. What do you understand about variable initialization?
  - vi. 2: Provide the answer for the asked questions and write them on papers.
- 3: Present the findings/answers to the whole class
- 4: For more clarification, read the key readings 1.3.1 and ask questions where necessary.



#### Key readings 1.3.1.:

##### JavaScript variables

A JavaScript variable is simply a name of a storage location. There are two types of variables in JavaScript: local variable and global variable

##### ✓ Declaration of variable

Creating a variable in JavaScript is called "declaring" a variable.

Before you use a variable in a JavaScript program, you must declare it.

##### Example:

**1. Syntax: In JavaScript, variables are declared using the `var`, `let`, or `const` keyword, followed by the variable name.**

##### For example:

```
var myVariable;
```

Four Ways to Declare a JavaScript Variable are:

- a. Using var
- b. Using let

- c. Using const
- d. Using nothing

**1. Var:** This keyword is used to declare variable globally.

**Syntax:** var variableName = "Variable-Value";

**Example:** var age =15;

**2. Let:** This keyword is used to declare variable locally.

**Syntax:** let variableName = "Variable-Value";

**Example:** let age =15;

**3. Const:** This keyword is used to declare variable locally.

**Syntax:** const variableName = "Variable-Value";

**Example:** const age =15;

**Note:** You can declare a variable with **let** and **var** without a value. In this case, the default value will be undefined.

```
var tomato;  
let potato;  
console.log(tomato); // undefined  
console.log(potato); // undefined
```

The above behavior is not valid for **const** because an initial value is required for it. If there is no initial value, the program throws a **SyntaxError**.

```
const avocado; // SyntaxError
```

**4. for nothing**, a value must be assigned to a variable declared without the **var**, **let** and **const** keyword.

**Syntax:** variableName = "Variable-Value";

**Example:** age =15;

The general rules for constructing names for variables (unique identifiers) are:

- Names can contain letters, digits, underscores, and dollar signs.
- Names must begin with a letter.
- Names can also begin with \$ and \_ (but we will not use it in this tutorial).
- Names are case sensitive (y and Y are different variables).
- Reserved words (like JavaScript keywords) cannot be used as names.

Declaring variables is a fundamental aspect of programming in JavaScript. Here are some key points to remember about variable declaration:

## **1. Syntax:**

Variables are declared using the **var**, **let**, or **const** keyword, followed by the variable name.

**2. Initialization:** Variables can be declared and initialized (assigned a value) at the same time using the assignment operator (=).

**3. Scope:** Variables have a scope, which determines their accessibility within a program. It's important to declare variables in the appropriate scope to ensure proper usage and avoid conflicts.

**4. Hoisting:** Variable declarations are hoisted to the top of their respective scopes during the compilation phase. However, the values for variables will be undefined until assigned.

**5. Type Inference:** JavaScript is a dynamically typed language, meaning variables can hold different types of data. The type of a variable is determined by the value assigned to it.

**6. Constants:** Variables declared with the **const** keyword are constants and cannot be reassigned after initialization. Use const for values that should remain constant throughout the program.

**7. Naming Conventions:** Choose meaningful and descriptive names for variables to improve code readability and maintainability. Follow naming conventions.

## **Variable initialization**

After the declaration, the variable has no value (technically it is undefined).

To assign a value to the variable, use the “=” sign:

The syntax of variable initialization is as follows:

`variable_name = variable_value;`

### **Example:**

`carName = "Volvo";`

You can also assign a value to the variable when you declare it:

`let carName = "Volvo";`

**Initialization:** Variables can be declared and initialized (assigned a value) at the same time.

### **For example:**

`var myVariable = 10;`

`let myVariable = "Hello";`

```
const myVariable = true;
```

### Javascript variable scope

**Scope** determines the accessibility (visibility) of variables.

Variable scope defines where in your code variables are accessible

The variable scope may be **local** or **global**.

#### ★ Javascript local variables

A JavaScript local variable is declared inside a block or function. It is accessible within the function or block only.

For example, variables declared inside a **{ } block** cannot be accessed from outside the block:

```
{
let x
}
// x can NOT be used here
```

#### ★ JavaScript global variables

A JavaScript global variable is accessible from any function. A variable declared outside a function or a block, becomes **GLOBAL**. This variable has a **Global Scope**.

**Global** variables can be accessed from anywhere in a JavaScript program.

#### Example

```
let carName = "Volvo";
// code here can use carName

function myFunction()

{
// code here can also use carName
}
```

#### Automatically Global

If you assign a value to a variable that has not been declared, it will automatically become a **GLOBAL** variable.

This code example will declare a global variable **carName**, even if the value is assigned inside a function.

### Example

```
myFunction();
// code here can use carName
function myFunction()
{
    carName = "Volvo";
}
```

### ✓ Redeclaration of variable

To redeclare a variable simply means to declare an already declared variable or identifier regardless of whether in the same block or outer scope.

**You could declare and initialise a variable using the `const` keyword.**

**Example:**

```
//declared and initialized the variable(firstName) and assigned a value of 'Patrick'.
const firstname='patrick';
console.log(firstname); //patrick
```

You may have tried to redeclare the variable or even try to change its value still using the `const` keyword.

```
const firstname='patrick';
const firstname='john';
//Uncaught SyntaxError: Identifier 'firstName' has already been declared.
```

It signals a syntax error because `const` never allows us to tamper with its identifier. With these experiments, we could agree that a variable declared using the '`const`' keyword cannot be redeclared and its value cannot be reassigned.

**Redeclaring Variables using the `let` keyword**

You can declare and initialize a variable using the `let` keyword.

```
//declared and initialized the variable and assigned a value of 'Okafor'.
let lastname='okafor';
console.log(lastname); //okafor
```

Let's try to redeclare the identifier and change its value.

```
let lastname='okafor';
let lastname='okafor';
//Uncaught SyntaxError: Identifier 'lastName' has already been declared
```

It logs out an error signalling that the identifier has already been declared therefore we must use another identifier.

Now with these experiments, you can see also that; a variable that is declared using the 'let' keyword cannot be redeclared again and its value cannot be changed or reassigned.

**Note:** We could declare a variable once and refer to it again without using the let keyword.

### For example

```
let lastname='okafor';  
lastname='doe';  
console.log(lastname); //doe
```

This can run successfully because we declared it with the let keyword. The const keyword would output an error.

```
const lastname='okafor';  
lastname='doe';  
console.log(lastname); //null
```

### Redeclaring Variables using the var keyword

Just like using the const and let keywords, You can declare and initialise a variable using the var keyword.

```
//declared and initialized the variable and assigned a value of 'Cat'.
```

```
var mypet='cat';  
console.log(mypet); //cat
```

Let's try to redeclare the variable and also change its value. We start with redeclaring the variable

```
var mypet='cat';  
var mypet='cat'; //cat
```

This time around, it doesn't display any errors. It logs out the value. Now, let's change the value;

```
var mypet='cat';  
var mypet='dog'; //dog
```

### Variable initialisation

Now, these variables once declared, are assigned some value. This assignment of value to these variables is called initialization of variables.

You can assign a value to a variable using the `=` operator when you declare it or after the declaration and before accessing it.

#### Example: Variable Initialization

```
var msg;  
msg = "Hello JavaScript!"; // assigned a string value  
alert(msg); // access a variable  
  
//the following declares and assign a numeric value  
var num = 100;  
var hundred = num; // assigned a variable to variable
```



#### Practical Activity 1.3.2: Use variables in JAVASCRIPT



##### Task:

- 1: You are requested to go to the computer lab and develop a simple JavaScript program by declaring and initializing variables. Remember to generate the output. This task should be done individually.
- 2: Read the key reading 1.3.2 in trainee manual about use of variable in JavaScript program.
- 3: Declare variables and initialize them with values.
- 4: Use one of JavaScript output functions to generate the result.



#### Key readings 1.3.2

- Variable declaration.

Certainly! Here are the key steps to remember about variable declaration in JavaScript:

**1. Choose the appropriate keyword:** JavaScript provides three keywords for variable declaration: `var`, `let`, and `const`.

Use `var` for traditional variable declaration, `let` for block-scoped variables, and `const` for constants.

**2. Use meaningful variable names:** Choose descriptive names for your variables that accurately represent their purpose. Follow naming conventions, such as starting with a letter and using camel case.

**3. Initialize variables when needed:** Variables can be declared and initialized at the same time using the assignment operator (=). This assigns an initial value to the variable.

**4. Understand variable scope:** Variables have scope, which determines their accessibility within a program. `var` has function scope, while `let` and `const` have block scope. Understand how variables are affected by scope and use them accordingly.

**5. Consider hoisting:** Variable declarations are hoisted to the top of their respective scopes during the compilation phase. This allows you to use variables before they are declared, but their values will be `undefined` until assigned.

**6. Be aware of variable reassignment:** Variables declared with var or let can be reassigned with a new value using the assignment operator (`=`). However, variables declared with const cannot be reassigned after initialization.

**7. Understand type inference:** JavaScript is a dynamically typed language, meaning variables can hold different types of data. The type of a variable is determined by the value assigned to it.

**8. Follow best practices:** Write clean and readable code by following best practices for variable declaration, such as declaring variables at the beginning of a scope and using the most appropriate keyword for the situation.

By remembering these key steps, you can effectively declare variables in JavaScript and ensure their proper usage within your programs.

**Here's an example of variable declaration in JavaScript:**

```
// Using var keyword  
  
var myVariable;  
  
myVariable = 10;  
  
// Using let keyword  
  
let anotherVariable = "Hello";  
  
// Using const keyword  
  
const PI = 3.14;  
  
// Variable reassignment  
  
myVariable = 20;  
  
anotherVariable = "World";  
  
// Outputting variable values
```

```
console.log(myVariable); // Output: 20  
console.log(anotherVariable); // Output: "World"  
console.log(PI); // Output: 3.14
```

In this example, we declare three variables:

myVariable, anotherVariable, and PI. We initialize myVariable with the value 10, anotherVariable with the string "Hello", and PI with the value 3.14. Later in the code, we reassign myVariable to 20 and anotherVariable to "World". Finally, we output the values of the variables using console.log().

- **Variable initialization**

Certainly! Here are the key steps to remember about variable initialization in JavaScript:

1. Declare the variable: Use the `var`, `let`, or `const` keyword to declare the variable.

**For Example:**

```
var myVariable;  
let myVariable;  
const myVariable;
```

2. Assign an initial value: Initialize the variable by assigning an initial value using the assignment operator (`=`). For example:

```
myVariable = 10;  
myVariable = "Hello";  
myVariable = true;
```

3. Default initialization: If you don't assign an initial value during declaration, the variable will be automatically initialized with the value **undefined**.

4. Dynamic typing: JavaScript is a dynamically typed language, meaning variables can hold different types of data. The type of a variable is determined by the value assigned to it during initialization.

5. Reassignment: After initialization, variables can be reassigned with a new value using the assignment operator (`=`).

**For example:** myVariable = 20;

6. Constants: If you want a variable to be a constant, use the `const` keyword during declaration and assign an initial value that cannot be changed later.

7. Initialization order: JavaScript hoists variable declarations to the top of their respective scopes during the compilation phase. This allows you to use variables before they are declared, but their values will be `undefined` until assigned.

By remembering these key steps, you can effectively initialize variables in JavaScript, assign appropriate values, and ensure they are ready for use in your code.

In the example below, we create a variable called carName and assign the value "Volvo" to it. Then we "output" the value inside an HTML paragraph with id="demo":

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Variables</h1>
<p>Create a variable, assign a value to it, and display it:</p>
<p id="demo"></p>
<script>
let carName = "Volvo";
document.getElementById("demo").innerHTML = carName;
</script>
</body>
</html>
```

**Output**

### **JavaScript Variables**

Create a variable, assign a value to it, and display it:

Volvo

### **Example2:**

```
<script>
var x = 10;
var y = 20;
var z=x+y;
document.write(z);
</script>
```

### **Output of the above example**

30

### **JavaScript local variable**

```
<script>
function abc()
{
    var x=10;//local variable
}
```

```

</script>
OR,
<script>
If(10<13)
{
var y=20;//JavaScript local variable
}
</script>

```

### **JavaScript global variable**

```

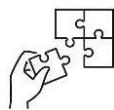
<script>
var data=200;//global variable
function a(){
document.writeln(data);
}
function b(){
document.writeln(data);
}
a();//calling JavaScript function
b();
</script>

```



### **Points to Remember**

- Creating a variable in JavaScript is called "declaring" a variable. Once a variable is created and get assigned with a value this action is known as variable initialization. Variables declared with the **const** keyword are constants and cannot be reassigned after initialization. There are four ways to Declare a JavaScript Variable are: Using **var**, Using **let**, using **const** and using nothing. Variable re-declaration means to declare an already declared variable.
- To initialize a variable, we use the assignment operator (=) followed by value.  
For example: **var salary =12000;**



### **Application of learning 1.3.**

You are building a temperature converter application. Users can input a temperature in either Celsius or Fahrenheit, and the application will convert it to the other unit. JavaScript variables will be used to store and manipulate the temperature values. Write a JavaScript program that

simulates a simple temperature converter. The program should prompt the user for a temperature in Celsius, convert it to Fahrenheit, and display the converted temperature.



## Indicative content 1.4: Use of data types in JavaScript



Duration: 5 hrs



### Theoretical Activity 1.4.1: Description of data types in JavaScript



#### Tasks:

- 1: In small groups, you are requested to answer the following questions related to the description of variables in JavaScript:
  - i. What do you understand about Data Type?
  - ii. Describe the different types of Data Type available in JavaScript.
  - iii. Explain the types of casting.
- 2: Provide the answer for the asked questions and write them on papers.
- 3: Present the findings/answers to the whole class.
- 4: For more clarification, read the key readings 1.4.1 and ask questions where necessary.



#### Key readings 1.4.1.:

##### Definition of data type

A **data type** is an attribute associated with a piece of data that tells a computer system how to interpret its value.

Data types describe the different types or kinds of data that we're going to be working with and storing in variables.

To hold different types of values, JavaScript provides different data types. In JavaScript, there are only two types of data types:

- **Primitive data type**
- **Non-primitive (reference) data type**

JavaScript is generally known as a dynamic type language that means there is no need to specify the type of variable as it is dynamically used by the JavaScript engine.

In order to specify the data, we have to use a var keyword. This can hold any type of values e.g. strings, numbers and more.

#### 1. Primitive data types in JavaScript

The predefined data types provided by JavaScript language are known as primitive data types. Primitive data types are also known as in-built data types.

In JavaScript, there are five types of primitive data types as depicted below:

Data Type	Description
String	represents sequence of characters e.g. "hello"
Number	represents numeric values e.g. 100
Boolean	represents boolean value either false or true
Undefined	represents undefined value
Null	represents null i.e. no value at all

## 2. Non-Primitive Data Types in JavaScript

These data types that are derived from primitive data types of the JavaScript language. There are also known as *derived* data types or *reference* data types.

There are generally three types of non-primitive data types which are as follows:

Data Type	Description
Object	represents instance through which we can access members
Array	represents group of similar values
RegExp	represents regular expression

### Type-casting

Type casting in JavaScript means converting one data type to another data type i.e., the conversion of a string data type to Boolean or the conversion of an integer data type to string data type. The typecasting in JavaScript is also known as **type conversion** or **type coercion**.

There are two types of type conversion in JavaScript.

- **Implicit Conversion:** automatic type conversion
- **Explicit Conversion:** manual type conversion



### Practical Activity 1.4.2: Use data types in JAVASCRIPT



#### Task:

- 1: Referring to the previous theoretical activities (1.4.1) you are requested to go to the computer lab and use data types in JavaScript program. This task should be done individually.
- 2: Read the key reading 1.4.2 in trainee manual about using data types in JavaScript program.
- 3: Use data types in JavaScript program.
- 4: Ask questions where necessary for more clarification.
- 5: Perform the task provided in application of learning 1.4.



#### Key readings 1.4.2

To be able to operate on variables, it is important to know something about the type.

#### The following examples illustrate the use primitive data types in JavaScript

- 1. Number:** Number data type in JavaScript can be used to hold decimal values as well as values without decimals.

##### *Example:*

```
<script>
    let x = 250;
    let y = 40.5;
    console.log("Value of x=" + x);
    console.log("Value of y=" + y);
</script>
```

##### **Output:**

```
Value of x=250
Value of y=40.5
```

- 2. String:** The string data type in JavaScript represents a sequence of characters that are surrounded by single or double quotes.

##### *Example:*

```
<script>
let str = 'Hello All';
let str1 = "Welcome to my new house";
console.log("Value of str=" + str);
console.log("Value of str1=" + str1);
</script>
```

**Output:**

```
Value of str=Hello All
Value of str1=Welcome to my new house
```

**3. Undefined:** The meaning of undefined is ‘value is not assigned’.

**Example:**

```
<script>
    console.log("Value of x=" + x);
</script>
```

**Output:**

```
Value of x=undefined
```

**5. Boolean:** The Boolean data type can accept only two values i.e. **true** and **false**.

**Example:**

```
<script>
console.log("value of bool=" + bool);
</script>
```

**Output:**

```
value of bool=true
```

**5. Null:** This data type can hold only one possible value that is null.

**Example:**

```
<script>
let x = null;
console.log("Value of x=" + x);
</script>
```

**Output:**

```
value of x=null
```

**Examples below explain the use of non-primitive data types in program**

**1. Object:** Object in JavaScript is an entity having properties and methods. Everything is an object in JavaScript.

For example, in a social media application, you might use objects to represent user profiles with attributes like name, age, and posts.

How to create an object in JavaScript:

- **Using Constructor Function to define an object:**

```
// Create an empty generic object
```

```
var obj = new Object();
```

```
// Create a user defined object
```

```
var mycar = new Car();
```

- **Using Literal notations to define an object:**

```
// An empty object
```

```
var square = {};
```

```
// Here a and b are keys, 20 and 30 are values
```

```
var circle = {a: 20, b: 30};
```

**Example:**

```
<script>
    // Creating object with the name person
    let person = {
        firstName: "Luiza",
        lastName: "Shaikh",
    };
    // Print the value of object on console
    console.log(person.firstName + " " + person.lastName);
</script>
```

**Output:**

```
Luiza Shaikh
```

**2. Array:** With the help of an array, we can store more than one element under a single name.

**Ways to declare a single dimensional array:**

```
var d = new Array(1, 2, 3, "Hello");
```

**Example:**

```
<script>
    var d = new Array(1, 2, 3, "Hello");
    console.log("value of d=" + d);
</script>
```

**Output:**

```
Value of d=1, 2, 3, Hello
```

**Type casting**

**a) JavaScript Implicit Conversion**

### **Example 1: Implicit Conversion to String**

```
// numeric string used with + gives string type  
  
let result;  
result = '3' + 2;  
console.log(result) // "32"  
  
result = '3' + true;  
console.log(result); // "3true"  
  
result = '3' + undefined;  
console.log(result); // "3undefined"  
  
result = '3' + null;  
console.log(result); // "3null"
```

**Note:** When a number is added to a string, JavaScript converts the number to a string before concatenation.

### **Example 2: Implicit Conversion to Number**

```
// numeric string used with - , / , * results number type  
  
let result;  
result = '4' - '2';  
console.log(result); // 2  
result = '4' - 2;  
console.log(result); // 2  
result = '4' * 2;  
console.log(result); // 8  
result = '4' / 2;  
console.log(result); // 2
```

### **Example 3: Non-numeric String Results to NaN**

```
// non-numeric string used with - , / , * results to NaN  
  
let result;  
result = 'hello' - 'world';  
console.log(result); // NaN  
result = '4' - 'hello';  
console.log(result); // NaN
```

**Note:** In JavaScript, NaN is short for "Not-a-Number". In JavaScript, NaN is a number that is not a legal number.

### **Example 4: Implicit Boolean Conversion to Number**

```
// if boolean is used, true is 1, false is 0
```

```
let result;  
result = '4' - true;  
console.log(result); // 3  
result = 4 + true;  
console.log(result); // 5  
result = 4 + false;  
console.log(result); // 4
```

In JavaScript considers 0 as false and all non-zero number as true. And, if true is converted to a number, the result is always 1.

#### **Example 5: null Conversion to Number**

```
// null is 0 when used with number  
let result;  
result = 4 + null;  
console.log(result); // 4  
result = 4 - null;  
console.log(result); // 4
```

#### **Example 6: undefined used with number, Boolean or null**

```
// Arithmetic operation of undefined with number, boolean or null gives NaN
```

```
let result;  
result = 4 + undefined;  
console.log(result); // NaN
```

```
result = 4 - undefined;  
console.log(result); // NaN
```

```
result = true + undefined;  
console.log(result); // NaN
```

```
result = null + undefined;  
console.log(result); // NaN
```

#### **b) JavaScript Explicit Conversion**

You can also convert one data type to another as per your needs.

The type conversion that you do manually is known as explicit type conversion.

In JavaScript, explicit type conversions are done using built-in methods.

Here are some common methods of explicit conversions.

##### **1. Convert to Number Explicitly**

To convert numeric strings and boolean values to numbers, you can use `Number()`.

```
For example,  
let result;  
// string to number  
result = Number('324');  
console.log(result); // 324  
result = Number('324e-1')  
console.log(result); // 32.4  
// boolean to number  
result = Number(true);  
console.log(result); // 1  
result = Number(false);  
console.log(result); // 0
```

In JavaScript, empty strings and null values return 0.

For example,

```
let result;  
result = Number(null);  
console.log(result); // 0  
let result = Number(' ')  
console.log(result); // 0
```

If a string is an invalid number, the result will be NaN.

For example,

```
let result;  
result = Number('hello');  
console.log(result); // NaN  
result = Number(undefined);  
console.log(result); // NaN  
result = Number(NaN);  
console.log(result); // NaN
```

**Note:** You can also generate numbers from strings using parseInt(), parseFloat(), unary operator + and Math.floor().

**For example,**

```
let result;  
result = parseInt('20.01');  
console.log(result); // 20  
  
result = parseFloat('20.01');  
console.log(result); // 20.01  
  
result = +'20.01';
```

```
console.log(result); // 20.01
```

```
result = Math.floor('20.01');  
console.log(result); // 20
```

## 2. Convert to String Explicitly

To convert other data types to strings, you can use either `String()` or `toString()`.

For example,

```
//number to string  
let result;  
result = String(324);  
console.log(result); // "324"
```

```
result = String(2 + 4);  
console.log(result); // "6"
```

```
//other data types to string  
result = String(null);  
console.log(result); // "null"
```

```
result = String(undefined);  
console.log(result); // "undefined"
```

```
result = String(NaN);  
console.log(result); // "NaN"
```

```
result = String(true);  
console.log(result); // "true"  
result = String(false);  
console.log(result); // "false"
```

```
// using toString()  
result = (324).toString();  
console.log(result); // "324"
```

```
result = true.toString();  
console.log(result); // "true"
```

**Note:** `String()` takes null and undefined and converts them to string. However, `toString()` gives error when null are passed.

### 3. Convert to Boolean Explicitly

To convert other data types to a boolean, you can use Boolean().

In JavaScript, undefined, null, 0, NaN, " converts to false.

For example,

```
let result;  
result = Boolean("");  
console.log(result); // false  
result = Boolean(0);  
console.log(result); // false  
result = Boolean(undefined);  
console.log(result); // false  
result = Boolean(null);  
console.log(result); // false  
result = Boolean(NaN);  
console.log(result); // false
```

All other values give true.

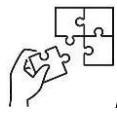
For example,

```
result = Boolean(324);  
console.log(result); // true  
result = Boolean('hello');  
console.log(result); // true  
result = Boolean(' ');  
console.log(result); // true
```



#### Points to Remember

- Data types describe the different types or kinds of data that we're going to be working with and storing in variables, in JavaScript, there are only two types of data types: Primitive data type and Non-primitive (reference) data type. Converting one data type to another data type is known as casting, there are two types of type casting in JavaScript and these are: Implicit Conversion and Explicit Conversion.
- To convert numeric strings and Boolean values to numbers, you can use Number () and to convert other data types to strings, you can use either String () or toString (). When converting other data types to a Boolean, you can use Boolean (). In JavaScript, undefined, null, 0, NaN, " converts to false.



### **Application of learning 1.4.**

You are developing a web application to track and manage student grades for a school. You'll encounter various data types in JavaScript to handle student information, course grades, and overall statistics.



## Indicative content 1.5: Use of operators in JavaScript



Duration: 7 hrs



### Theoretical Activity 1.5.1: Description of the operators in JavaScript



#### Tasks:

- 1: In small groups, you are requested to answer the following questions related to the operators in JavaScript:
  - i. What do you understand about operator?
  - ii. Explain different types of operators
- 2: Provide the answer for the asked questions and write them on papers.
- 3: Present the findings/answers to the trainer and the whole class
- 4: Ask questions for more clarification where necessary.
- 5: For more clarification, read the key readings 1.5.1 in trainee manual.



#### Key readings 1.5.1.:

##### Definition of Operator:

**Operators** in JavaScript are symbols or keywords that perform operations on operands (values or variables). They allow you to perform arithmetic calculations, compare values, assign values, and more.

##### The different types of JavaScript operators are explained below:

###### ➤ Assignment Operators

Assignment operators in JavaScript are used to assign a value to a variable. They combine the assignment operation with another operation, such as arithmetic or bitwise operations, providing a shorthand way to update the value of a variable based on its current value.

The basic assignment operator in JavaScript is the equal sign (=). It assigns the value on the right-hand side to the variable on the left-hand side.

For example:

```
let x = 5;
```

All the various types of Assignment operators are mentioned in the table below:

<b>Sym bol</b>	<b>Description</b>	<b>Example</b>
=	Assign	$10+10 = 20$
+=	Add and Assign	<code>var a=10; a+=20; Now a = 30</code>
-=	Subtract and assign	<code>var a=20; a-=10; Now a = 10</code>
*=	Multiply and assign	<code>var a=10; a*=20; Now a = 200</code>
/=	Divide and assign	<code>var a=10; a/=2; Now a = 5</code>
%=	Modulus and assign	<code>var a=10; a%=2; Now a = 0</code>

#### ➤ Arithmetic Operators:

These operators are used to operate mathematical operations between numeric operands.

These operators are used to operate mathematical operations between numeric operands.

<b>Opera tor</b>	<b>Description</b>
+	Used to add two numeric operands.
-	Used to subtract right operand from left operand
*	Used to multiply two numeric operands.
/	Used to divide left operand by right operand.
%	Modulus operator. It returns the remainder of two operands.
++	Increment operator. It Increases operand value by one.
--	Decrement operator. It decreases value by one.

The following examples will help you in understanding the Arithmetic Operators in running different tasks on Javascript operators:

```

var a = 2, b = 4;
a + b; //returns 6
b - a; //returns 2
a * b; //returns 8
b / a; //returns 2
  
```

```

a % 2; //returns 0
a++; //returns 3
a--; //returns 1

```

### ➤ String Operators

A string operator in JavaScript, simply put, is an operator that operates on a string.

There are only two operators that JavaScript supports for modifying a string. These operators allow you to join one string to another easily. These operators are called “concatenate” and “concatenate assignment” .

These two operators allow you to join one string to another.

- **Concatenate:** This refers to appending one string to another.
- **Concatenate Assignment:** Similar to the previous but refers more specifically to appending a string to the end of a variable. Assigning the final result to that variable.

To showcase JavaScript’s string operators, we have created a small table. This table shows both the supported string operators, their names, an example of them being used, and the result of that operation.

Operator	Name	Example	Result
+	Concatenate	“str1” + “str2”	String is joined. Results in “str1str2”
+=	Concatenate Assignment	str += “str2”	String is appended. str has “str2” appended to it.

### ➤ Comparison Operators

Any time you compare two values in JavaScript, the result is a Boolean true or false value. You have a wide selection of comparison operators to choose from, depending on the kind of test you want to apply to the two operands.

All the various types of comparison operators are mentioned in the table below:

Operator	Description	Example
==	Is equal to	11==22 //false
====	Identical (equal and of same type)	11====22 //false

<b>!=</b>	Not equal to	10!=20 // true
<b>!==</b>	Not Identical	22!==22 // false
<b>&gt;</b>	Greater than	22>12 // true
<b>&gt;=</b>	Greater than or equal to	22>=12 // true
<b>&lt;</b>	Less than	22<12 //false
<b>&lt;=</b>	Less than or equal to	22<=12 // false

#### **'==' operator:**

In JavaScript, the use of this operator is mainly in comparing two values on both the sides and then returning true or false after evaluating both sides.

For numeric values, the results are the same as those you would expect from your high school algebra class.

```
10 == 10    // true
10 == 10.0   // true
9 ==10      // false
```

#### **'===' operator:**

This JavaScript operator is also called a strict equality operator, it simultaneously compares the value and the type.

### **➤ JavaScript Logical Operators**

All the various types of Logical operators are mentioned in the table below:

Symbol	Description	Example
<b>&amp;&amp;</b>	Logical AND	(12==22 && 22==33) //false
<b>  </b>	Logical OR	(12==22    22==33) //false
<b>!</b>	Logical Not	!(12==22) // true

### **➤ JavaScript Bitwise Operators**

JavaScript bitwise operators convert their operands to 32-bit signed integers in two's complement format

The bitwise operator's carryout bitwise operations on the operands. Here are the bitwise operators:

Operator	Description	Example
<b>&amp;</b>	Bitwise AND	(10==20 & 20==33) // false
<b> </b>	Bitwise OR	(10==20   20==33) //false

<code>^</code>	Bitwise XOR	<code>(10==20 ^ 20==33) //false</code>
<code>~</code>	Bitwise NOT	<code>(~10) = -10</code>
<code>&lt;&lt;</code>	Bitwise Left Shift	<code>(10&lt;&lt;2) = 40</code>
<code>&gt;&gt;</code>	Bitwise Right Shift	<code>(10&gt;&gt;2) = 2</code>
<code>&gt;&gt;&gt;</code>	Bitwise Right Shift with Zero	<code>(10&gt;&gt;&gt;2) = 2</code>

### ➤ Ternary (Special) Operators

The conditional (ternary) operator is the only JavaScript operator that takes three operands:

a condition followed by a question mark ( ? ), then an expression to execute if the condition is truthy followed by a colon ( : ), and finally the expression to execute if the condition is falsy.

Sym bol	Description
<code>(?:)</code>	Conditional Operator returns value based on the condition. It is like if-else.



### Practical Activity 1.5.2: Use the operators in JavaScript



#### Task:

- 1: Referring to the previous theoretical activities (1.5.1) you are requested to go to the computer lab to use operators in JavaScript. This task should be done individually.
- 2: Read the key reading 1.5.2 in trainee manual about using operators in JavaScript program.
- 3: Use some operators in simple JavaScript program.
- 4: Read key reading 1.5.2 and ask clarification where necessary



### Key readings 1.5.2

By following these steps, you can effectively use operators in JavaScript to perform various operations, calculations, comparisons, and logical evaluations in your code.

#### ✓ Assignment operators

`<!DOCTYPE html>`

```
<html>
<body>
<h1>JavaScript Assignments</h1>
<h2>Simple Assignment</h2>
<h3>The = Operator</h3>
<p id="demo"></p>
<script>
let x = 10;

document.getElementById("demo").innerHTML = "Value of x is: " + x;
</script>
</body>
</html>
```

**Output:**

JavaScript Assignments  
Simple Assignment  
The = Operator  
Value of x is: 10

**✓ Arithmetic operators**

**Examples**

```

let x = 5;
let y = 3;

// addition
console.log('x + y = ', x + y); // 8

// subtraction
console.log('x - y = ', x - y); // 2

// multiplication
console.log('x * y = ', x * y); // 15

// division
console.log('x / y = ', x / y); // 1.6666666666666667

// remainder
console.log('x % y = ', x % y); // 2

// increment
console.log('++x = ', ++x); // x is now 6
console.log('x++ = ', x++); // prints 6 and then increased to 7
console.log('x = ', x); // 7

// decrement
console.log('--x = ', --x); // x is now 6
console.log('x-- = ', x--); // prints 6 and then decreased to 5
console.log('x = ', x); // 5

//exponentiation
console.log('x ** y = ', x ** y);

```

## String operator

In JavaScript, you can also use the + operator to concatenate (join) two or more strings

### Example:

```

// concatenation operator
console.log('hello' + 'world');

let a = 'JavaScript';

a += ' tutorial'; // a = a + ' tutorial';
console.log(a);

```

```

helloworld
JavaScript tutorial

```

## ✓ Comparison operators

### Examples:

```
// equal operator
console.log(2 == 2); // true
console.log(2 == '2'); // true

// not equal operator
console.log(3 != 2); // true
console.log('hello' != 'Hello'); // true

// strict equal operator
console.log(2 === 2); // true
console.log(2 === '2'); // false

// strict not equal operator
console.log(2 !== '2'); // true
console.log(2 !== 2); // false
```

## ✓ Logical operators

Examples:

```
// logical AND
console.log(true && true); // true
console.log(true && false); // false

// logical OR
console.log(true || false); // true

// logical NOT
console.log(!true); // false
```

## Output

```
true
false
true
false
```

## ✓ Bitwise operators

Examples:

Operation	Result	Same as	Result
5 & 1	1	0101 & 0001	0001
5   1	5	0101   0001	0101
~ 5	10	~0101	1010
5 << 1	10	0101 << 1	1010
5 ^ 1	4	0101 ^ 0001	0100
5 >> 1	2	0101 >> 1	0010
5 >>> 1	2	0101 >>> 1	0010

## ✓ Ternary operator

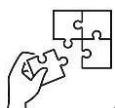
**Example:**

```
const age = 26;
const beverage = age >= 21 ? "Beer" : "Juice";
console.log(beverage); // "Beer"
```



### Points to Remember

- Operators in JavaScript are symbols or keywords that perform operations on operands (values or variables). There are different types of operators and these are assignment operators, arithmetic operators, string operators, comparison operators, logical operators, bitwise operators and conditional operator.
- Operators allow you to perform arithmetic calculations, compare values, assign values, and more.



### Application of learning 1.5.

You are tasked with developing an inventory management system for a retail store using JavaScript. You will utilize different operators to handle stock levels, track sales, and manage product availability.



## Learning outcome 1 end assessment

### Written assessment

1. Choose the correct answer

i) JavaScript is a \_\_\_\_\_ language.

- a) Object-Oriented
- b) High-level
- c) Assembly-languaged)

Object-Based

ii) Initialization of variables can be done by writing----- operator in between variable name and operand value.

- a)EQUALS
- b)=
- c)VALUE
- d)==

iii) Which of the following keywords is used to define a variable in JavaScript?

- a)var
- b)let
- c)Both A and B
- d)None of the above

2. Enumerate five JavaScript Data Types

3. Based on knowledge acquired in JavaScript variables, you are asked to read the following statement and state whether they are true or false.

- i. Variable Scope determines the accessibility (visibility) of variables.
- ii. A variable stores the data value that can be changed later.
- iii. Any variable declared inside a block such as a function can be accessed anywhere in a program.

4. Match the following items of A column which corresponding to B column.

A	B
1. Variable scope	A. is a JavaScript framework for building user interfaces.
2. vue	B. can be accessed from anywhere in a JavaScript program.
3. Global variables	C. is literally just where your application is running in.

**4. Runtime environment**

D. determines the accessibility (visibility) of variable.

5. Complete the following statement with the correct word(s)

"In JavaScript, an..... Is a sign or a special symbol used to perform operations on operands?

6. Give the output of the following JavaScript statements:

- a) result = '4' - 2;  
    console.log (result);
- b) result = 'hello' - 'world';  
    console.log(result);
- c) result = 4 + true;  
    console.log(result);
- d) result = 4 - undefined;  
    console.log(result);
- e) result = Number('324');  
    console.log(result);
  
- f) let text = "Hello";  
text += " World";  
document.write("MY TEXT is " +text);
- g) let x = 10;  
x += 5;  
document.getElementById("demo").innerHTML = "Value of x is: " + x;
- h) let x = 5;  
    console.log('--x = ', --x);
- i) let x = 5;  
    document.getElementById("demo").innerHTML = (x !== 5);
- j) let a = 6;  
let b = 14;  
result = a & b;  
window.alert(result);

### Practical assessment

The calculator should allow users to perform basic arithmetic operations like addition, subtraction, multiplication, and division. Your goal is to integrate JavaScript within an HTML document, use JavaScript variables, data types, and operator concepts to implement the calculator functionality. You are tasked with creating a simple calculator using HTML and JavaScript.



## References

*Escape characters in JavaScript.* (n.d.). <https://www.tutorialspoint.com/escape-characters-in-javascript>

Jain, M. (2022, August 29). String Concatenation in JavaScript - Scaler Topics. *Scaler Topics*. <https://www.scaler.com/topics/string-concatenation-javascript/>

*JavaScript comment - javatpoint.* (n.d.). www.javatpoint.com.

<https://www.javatpoint.com/javascript-comment>

*JavaScript Strings.* (n.d.). [https://www.w3schools.com/js/js\\_strings.asp](https://www.w3schools.com/js/js_strings.asp)

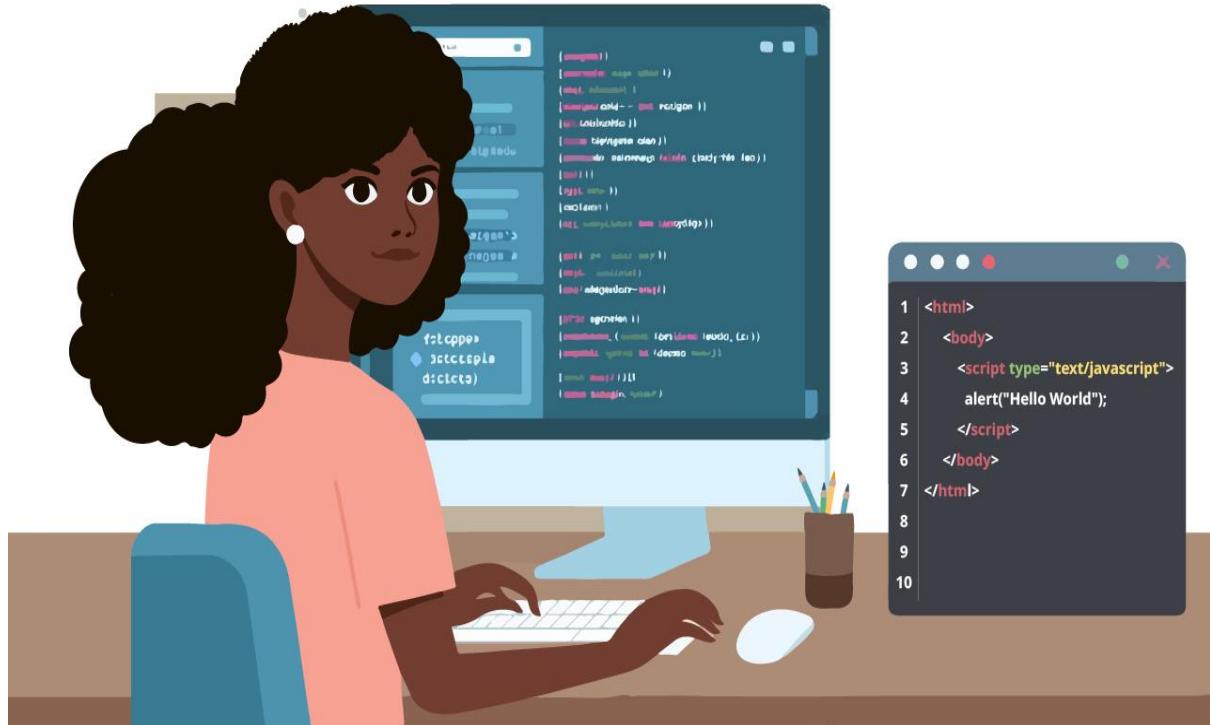
Sheldon, R., & Denman, J. (2022). Node.js (Node). *WhatIs.com*.

<https://www.techtarget.com/whatis/definition/Nodejs>

Wikipedia contributors. (2023). Value (computer science). *Wikipedia*.

[https://en.wikipedia.org/wiki/Value\\_\(computer\\_science\)](https://en.wikipedia.org/wiki/Value_(computer_science))

## Learning Outcome 2: Manipulate data with JavaScript



### **Indicative contents**

- 2.1 Using string in JavaScript**
- 2.2 Using conditional statement**
- 2.3 Using Loop functions in JavaScript**
- 2.4. Using Functions in JavaScript**
- 2.5 Using objects in JavaScript**
- 2.6. Using arrays in JavaScript**
- 2.7 Using JavaScript in HTML**
- 2.8 Applying regular expression**
- 2.9 Error handling**

### **Key Competencies for Learning Outcome 2: Manipulate data with JavaScript**

<b>Knowledge</b>	<b>Skills</b>	<b>Attitudes</b>
<ul style="list-style-type: none"><li>• Description of JavaScript strings</li><li>• Description of conditional statements</li><li>• Description of Loops in JavaScript</li><li>• Description of Functions in JavaScript</li><li>• Description of JavaScript objects.</li><li>• Description of arrays in JavaScript</li><li>• Application of JavaScript in HTML</li><li>• Description of regular expressions</li><li>• Description of errors in JavaScript</li></ul>	<ul style="list-style-type: none"><li>• Applying string in JavaScript.</li><li>• Using conditional statements in JavaScript</li><li>• Using loop statements in JavaScript</li><li>• Applying functions in JavaScript</li><li>• Using JavaScript objects</li><li>• Using arrays in JavaScript</li><li>• Applying JavaScript in HTML.</li><li>• Using Regular expressions in JavaScript.</li><li>• Handling errors in JavaScript program</li></ul>	<ul style="list-style-type: none"><li>• Being Problem solver</li><li>• Being Attentive</li><li>• Being confident</li><li>• Being a critical thinker</li><li>• Being analytical and details oriented</li><li>• Being Team worker</li></ul>



**Duration: 70 hrs**

**Learning outcome 2 objectives:**



By the end of the learning outcome, the trainees will be able to:

1. Describe properly JavaScript strings based on the task to be done
2. Apply effectively conditional statements in JavaScript based on control flow
3. Use effectively loops in JavaScript program based on the given task
4. Apply correctly functions in JavaScript program.
5. Use correctly objects in JavaScript.
6. Use correctly arrays in JavaScript program based on given task
7. Apply effectively regular expressions in JavaScript program
8. Handle correctly JavaScript errors based on JavaScript error handling mechanism
9. Apply effectively JavaScript in HTML based on events occurring.



**Resources**

<b>Equipment</b>	<b>Tools</b>	<b>Materials</b>
<ul style="list-style-type: none"><li>• Computer</li><li>• Projector</li><li>• White board</li></ul>	<ul style="list-style-type: none"><li>• Text editor</li><li>• Node js</li><li>• Browser</li></ul>	<ul style="list-style-type: none"><li>• Internet</li></ul>



## Indicative content 2.1: Using string in JavaScript



Duration: 6 hrs



### Theoretical Activity 2.1.1: Description of string in JavaScript



#### Tasks:

- 1: In formed groups, you are requested to answer the following questions related to JavaScript strings
  - i. What do you understand about the term “string”
  - ii. What is meant by string concatenation?
  - iii. Outline methods or ways by which we can concatenate strings in JavaScript.
- 2: Provide the answer for the asked questions and write them on papers.
- 3: Present the findings/answers to the whole class.
- 4: For more clarification, read the key readings 2.1.1 and ask questions where necessary.



#### Key readings 2.1.1.:

##### Definition of String in java script

A string is a data type used to represent textual data. It is a sequence of characters enclosed within single quotes ("'), double quotes ("") or backticks (``).

Strings can contain letters, numbers, symbols, and special characters.

##### ✓ String declaration

To declare a string we use the **var**, **let**, or **const** keyword as any other type.

In JavaScript, string declaration involves assigning a value to a variable using single quotes ("'), double quotes ("") or backticks (``).

##### Here are examples of string declarations:

1. Using single quotes:

```
Let message = 'Hello, World!';
```

2. Using double quotes:

```
let name = "John Doe";
```

3. Using backticks (template literals):

```
Let sentence = `I'm learning JavaScript. `;
```

#### ✓ **Escape characters**

In JavaScript, escape characters are special characters that are used to represent certain characters or sequences within a string. They are denoted by a backslash (\) followed by a specific character or code.

**Here are some commonly used escape characters in JavaScript:**

1. **\n:** Represents a newline character.

```
let message = "Hello\nWorld!";
```

2. **\t:** Represents a tab character.

```
let message = "Hello\tWorld!";
```

Following are the escape characters in JavaScript:

Code	Result
\b	Backspace
\f	Form Feed
\n	New Line
\r	Carriage Return
\t	Horizontal Tabulator
\v	Vertical Tabulator
\'	Single quote
\"	Double quote
\	Backslash

#### ✓ **String concatenation**

String concatenation in any programming language means to append one or more strings to the end of another string.

For example, Concatenating strings "World, " and "Good Afternoon" to the end of string "Hello " makes the final string as "Hello World, Good Afternoon".

**Here are four methods or ways by which we can concatenate strings in JavaScript:**

- using the concat() method
- using the '+' operator
- using the array join() method
- using template literals

### **1. Using the concatenation operator (+):**

You can use the + operator to concatenate strings together.

### **2. Using the concat() method:**

The concat() method can be used to concatenate strings. It takes one or more string arguments and returns a new string.

### **3. Using template literals:**

Template literals, enclosed in backticks (`), allow for string interpolation and concatenation of expressions within a string.

## **✓ String methods**

JavaScript provides a variety of built-in string methods that allow you to manipulate and work with strings.

Let's see the list of JavaScript string methods

Methods	Description
charAt()	It provides the char value present at the specified index.
charCodeAt()	It provides the Unicode value of a character present at the specified index.
concat()	It provides a combination of two or more strings.
indexOf()	It provides the position of a char value present in the given string.
lastIndexOf()	It provides the position of a char value present in the given string by searching for a character from the last position.
search()	It searches a specified regular expression in a given string and returns its position if a match occurs.
match()	It searches a specified regular expression in a given string and returns that regular expression if a match occurs.

replace()	It replaces a given string with the specified replacement.
substr()	It is used to fetch the part of the given string on the basis of the specified starting position and length.
substring()	It is used to fetch the part of the given string on the basis of the specified index.
slice()	It is used to fetch the part of the given string. It allows us to assign positive as well negative index.
toLowerCase	It converts the given string into lowercase letters.
toLocaleLow e()	It converts the given string into lowercase letter on the basis of current locale.
toUpperCase	It converts the given string into uppercase letters.
toLocaleUpp e()	It converts the given string into uppercase letter on the basis of current locale.
toString()	It provides a string representing the particular object.
valueOf()	It provides the primitive value of string object.
split()	It splits a string into substring array, then returns that newly created array.
trim()	It trims the white space from the left and right side of the string.

Here are some commonly used string methods:

#### **1. toUpperCase():**

The toUpperCase() method converts a string to uppercase.

#### **2. toLowerCase():**

The toLowerCase() method converts a string to lowercase.

#### **3. indexOf():**

The indexOf() method returns the index of the first occurrence of a specified substring within a string.

#### **4. substring():**

The substring() method extracts a portion of a string based on specified indices.

## 5. `split()`:

The `split()` method splits a string into an array of substrings based on a specified separator.

### ✓ **String search method**

The `search()` method matches a string against a regular expression. It returns the index (position) of the first match.

### ✓ **String Template literals**

Template literals (template strings) allow you to use strings or embedded expressions in the form of a string.

They are enclosed in backticks ``.

In the earlier versions of JavaScript, you would use a single quote '' or a double quote "" for strings.

In JavaScript, template literals, also known as template strings, are a way to create strings that allow for embedded expressions and multiline strings. Template literals are enclosed within backticks (``) instead of single or double quotes.

Template literals also allow for multiline strings without the need for manual concatenation or escape characters. The indentation is preserved within the string.

Template literals can include expressions that are evaluated and the resulting values are inserted into the string.



### Practical Activity 2.1.2: Use string in JavaScript



#### Task:

- 1: Referring to the previous theoretical activities (2.1.2) you are requested to go to the computer lab to use String in different ways as used in JavaScript data manipulation.
- 2: Read the key reading 2.1.2 in trainee manual about using string in JavaScript.
- 3: Referring to the outlined procedures provided by trainer, develop your own JavaScript program by using string concepts.
- 4: Ask questions where necessary for clarification.



### Key readings 2.1.2

1. **Declare a string variable:** Use the let, const, or var keyword to declare a variable that will hold the string value.
2. **Assign a value to the string variable:** Use the assignment operator (=) to assign a string value to the variable. Enclose the string value in single quotes ("'), double quotes ("") or backticks (``).
3. **Manipulate strings:** Use various string methods and operations to manipulate and work with the string. This includes concatenation, slicing, replacing, converting case, and more.
4. **Access characters and substrings:** Use indexing and string methods like charAt(), substring(), or slice() to access specific characters or substrings within the string.
5. **Perform string comparisons:** Use comparison operators (==, ===, etc.) to compare strings based on their values or lengths.
6. **Use conditional statements:** Utilize if statements, switch statements, or ternary operators to perform different actions based on conditions involving strings.
7. **Iterate over strings:** Use loops like for loops or while loops to iterate over characters or substrings within a string.
8. **Utilize regular expressions:** Apply regular expressions to perform advanced pattern matching and manipulation on strings.
9. **Use string interpolation:** Use template literals (``) to embed expressions and variables within a string using \${} for dynamic string construction.

By following these procedures, you can effectively work with strings in JavaScript, manipulate their content, perform comparisons, iterate over characters, and perform various operations to handle and process textual data in your JavaScript code.

#### ✓String declaration

**Here are the steps to use while declaring strings in JavaScript:**

1. Choose a variable name: Select a meaningful and descriptive name for your string variable. This will help you identify and refer to the string value later in your code.

2. Declare the variable: Use the `let`, `const`, or `var` keyword to declare the variable and allocate memory for it.
3. Assign the string value: Use the assignment operator (=) to assign the desired string value to the variable. Enclose the string content within single quotes ("'), double quotes ("") or backticks (``).
4. Use the string variable: You can now use the declared string variable in your code. You can perform operations, manipulate the string, or use it in various ways as needed.

**Example of the steps in action:**

```
// Step 1: Choose a variable name
message
// Step 2: Declare the variable
let message

// Step 3: Assign the string value
message = "Hello, World!"
```

The above three steps can be combined into a single step:

```
// let message = "Hello, World!"
```

Now, the variable "message" holds the string value "Hello, World!" and you can use the "message" variable in your code as needed.

```
console.log(message);
```

**Output will be:** Hello, World!

**Here's how you can declare strings in JavaScript -**

```
var name = "David";
var subject = "Programming";
let carName2 = 'Volvo XC60';
```

**Note:** you can use quotes inside a string, as long as they don't match the quotes surrounding the string:

```
let answer1 = "It's alright";
let answer2 = "He is called 'Johnny'";
let answer3 = 'He is called "Johnny"';
```

## ✓ Escape characters

- The sequence \" inserts a double quote in a string:

**Here are the steps to use escape characters in JavaScript:**

1. Identify the character or sequence you want to represent: Determine the special character or sequence that you want to include within a string.
2. Decide on the appropriate escape character: Choose the appropriate escape character that corresponds to the special character or sequence you want to represent. Common escape characters include backslash (\).
2. Insert the escape character before the special character or sequence: Place the chosen escape character (\) before the special character or sequence within the string.

**Examples of using escape characters in JavaScript:**

**Example 1:**

```
let message = "She said, \"Hello!\"";  
console.log(message);
```

**Output:** She said, "Hello!"

**Example 2:**

```
<script>  
let text = "We are the so-called \"Vikings\" from the north.";  
document.getElementById("demo").innerHTML = text;  
</ script>
```

**Output**

We are the so-called "Vikings" from the north.

**Example 3:** The sequence \' inserts a single quote in a string:

```
<script>  
let text = 'It\'s alright.';  
document.getElementById("demo").innerHTML = text;  
</ script>
```

**Output**

It's alright.

**Example 4:** The sequence \\ inserts a backslash in a string

```
<script>  
let text = "The character \\ is called backslash.";  
document.getElementById("demo").innerHTML = text;  
</ script>
```

**Output**

The character \ is called backslash.

### ✓ String concatenation

#### Here are the steps to use String Concatenation in JavaScript:

1. Declare the string variables
2. Assign values to the string variables
3. Use the concatenation operator (+) or use string methods to concatenate the string variables together.

### ★ String Concatenation Using '+' Operator

Let's understand this concept using some example codes:

#### ➤ Non-Mutative Concatenation ('+' Operator)

In this JavaScript example, we are using '+' operator to simply concatenate the strings. We have initialised three strings at the start and then using the '+' operator, we have concatenated them at the end of the string; "I am available on " and stored the newly generated string in the result variable.

#### Example

```
let instagram = 'Instagram';
let twitter = 'Twitter';
let facebook = 'Facebook';
let result = 'I am available on ' + instagram + ', ' + twitter + ' and ' + facebook;
console.log(result);
```

#### Output:

The value of the result variable is printed as output indicating the use-case of the '+' operator.

I am available on Instagram, Twitter and Facebook

#### ➤ Mutative Concatenation ('+=' Operator)

#### Example 1

In this example, we have initialised four strings at the start, and then using the '+=' operator, we are appending strings: instagram, twitter and facebook to the end of the string result and the final resulted string will again be stored in the result variable.

This is very useful in case if we need to append something to the result string at a later stage, then we don't have to write it again and we can simply use the '+=' operator to append in the already declared result string.

```
let instagram = 'Instagram';
let twitter = 'Twitter';
let facebook = 'Facebook';
let result = 'I am available on ';
result += instagram + ', ' + twitter + ' and ' + facebook;
console.log(result);
```

**Output:**

```
I am available on Instagram, Twitter and Facebook
```

## Example 2

While using concat() functional method, the starting variable (on whose end, we are concatenating) must be of string type for further concatenation but if we use '+' or '+=' operator, then we can concatenate on the end of any object, string or constant variable.

```
let str = 20;
//concatenating at the end of integer variable
str += " Testing '+=' operator: ";
str += 42 + '';
str += {} + '';
str += null;
console.log(str);
```

**Output:**

```
20 Testing '+=' operator: 42 [object Object] null
```

### ★ String Concatenation using Concat() Method

**Syntax :** string1.concat(value1, value2, ... value\_n);

## Example 1:

In this JavaScript example, we have a string temp as "Good Morning". We have passed two more new strings - ", " and "Hava a nice day!" in temp.concat() functional method which will concatenate all the strings and return a new string - message as "Good Morning, Have a nice day!".

```
let temp = "Good Morning";
let message = temp.concat(", ", "Have a nice day!");
console.log(message);
```

## **Output**

The resultant string message after concatenation is displayed as output.

**Good Morning, Have a nice day!**

### **Example 2. (Concatenating an Array of Strings)**

Using JavaScript, we can also concatenate items of an array to the end of a string. In this example, we are trying to concatenate items of an array - **colors** to the end of an empty string using spread operator in JavaScript (...colors) that unpacks all the items of the array as parameters inside concat() method.

```
let colors = ['Red',' ','Green',' ','Blue'];
let result = ".concat(...colors); //spread operator used
console.log(result);
```

## **Output**

After concatenation of array string items to the end of an empty string, we are printing the result message as output.

**Red Green Blue**

### **Example 3. (Concatenating Non-String Arguments)**

We can also pass non-string arguments inside the concat() function that are converted and treated as string parameters.

In this JavaScript example, we have passed integer constants;1,2,3 and a string parameter inside the concat() method that will be concatenated and the returned string will be stored as result in variable str.

```
let str = ".concat(1,2,3, "Hello, testing non-string arguments!");  
console.log(str);
```

## **Output**

After the conversion of integer constants to string parameters and finally the concatenation, we have printed the result of the str variable as the output.

**123 Hello, testing non-string arguments!**

### **Example 4. (TypeError)**

In this example, we have taken the str variable as a non-string (integer) and we are trying to concatenate the "Hello" string to the end of that non-string variable str. This clearly violates the rules and hence, will generate TypeError in the output console window.

**Note:** str variable must be of string type for using concat().

```
et str = 10;  
let ans = str.concat("Hello");  
console.log(ans);
```

### Output

TypeError message generated in output console window stating that Integer.concat() function doesn't exist.

**TypeError: str.concat is not a function**

### ★ String Concatenation using Array Join() Method

The join() method of JavaScript is used to concatenate all the elements present in an array by converting them into a single string. These elements are separated by a default separator i.e. comma(,) but we have the flexibility of providing our own customised separator as an argument inside the join() function.

One thing to be noted is that this join() functional method has no effect on the original array. Suppose if we need to concatenate any new string or object, we can push that in the already declared array and then use the join() method again.

#### Syntax:

For Using Join() Method, the following syntax will be used

```
array.join(separator);
```

Here, the separator can be user-customised and by default, if not specified, it uses the (,) comma as a separator. Hence, we get to know that the separator parameter is optional to include.

Let's understand this concept using some example codes:

#### Example 1

In this JavaScript example, we have initialised an array consisting of some string values. After that, we have used array.join() functional method along with customised separator - (" , ") comma along with space. This will combine all the string values of the array separated by comma and space & the newly generated string will be stored in final\_str variable.

```
let array = ["concat()", "join()", "+", "template literals"];  
let final_str = array.join(' , ');\nconsole.log(final_str);
```

**Output:**

The newly returned string after using the join() method will be displayed in the output console where array items are separated by comma and space.

**concat(), join(), '+', template literals****Example 2:**

Modifying the above example, in case if we wish to concatenate another string value at a later stage, then we can simply push it into the array and use join() method again.

```
let array = ["concat()", "join()", "+"];  
array.push("template literals");  
let final_str = array.join(', ');\nconsole.log(final_str);
```

**Output:**

Pushing string - "template literals" in the original array and then using the join() method will display the same output as in above Example 1.

**concat(), join(), '+', template literals**

### ★ String Concatenation using Template Literals

Before actually understanding the concept of Template Literals, let me draw your attention towards the common mistake that we all encounter while concatenating the strings i.e. problem of missing space in string concatenation.

**For Example:**

```
let name = "Mayank";\nlet str = "Hi, I am" + name + "and I am from" + "India";\nconsole.log(str);
```

**Output:**

In the above example, we haven't put the spaces required between some words to separate them. As a result, our output will look messy and not the desired one!

**Hi, I amMayankand I am fromIndia**

The concept of string literals helps us to resolve the above issue. Template literals are string literals that allow the use of embedded expressions within the string quoted with the help of back-ticks. Using these template literals, we can also use multi-line strings.

Syntax for Using Template Literals is as follow:

```
let expression_variable = 'string';

`string text ${expression_variable} string text`
```

Here, with the help of \$ and curly braces, we can embed a variable in our string quoted under back-ticks, indicating string concatenation :)

Let's discuss this concept using an example code:

### Example 1

In this JavaScript example, we have initialized three string variables at the start and then with the concept of template literals, we have embedded str1, str2, and str3 in the string; final\_str quoted under back-ticks. In this way, we can write exactly how we want our string to appear as output and also, it becomes easy to spot the missing spaces between the words.

```
let str1 = "Learning Javascript";
let str2 = "String";
let str3 = "Concatenation";
let final_str = `${str1} ${str2} ${str3} using Template Literals`;
console.log(final_str);
```

### Output:

Learning Javascript String Concatenation using Template Literals

### ✓ String methods

#### 1) JavaScript String charAt(index) Method

```
<script>
var str="javascript";
document.write(str.charAt(2));
</script>
```

#### Output:

v

#### 2) JavaScript String concat(str) Method

```
<script>
var s1="javascript ";
var s2="concat example";
var s3=s1.concat(s2);
document.write(s3);
```

```
</script>
```

**Output:**

javascript concat example

3) JavaScript String indexOf(str) Method

```
<script>
var s1="javascript from javatpoint indexof";
var n=s1.indexOf("from");
document.write(n);
</script>
```

**Output:**

11

4) JavaScript String lastIndexOf(str) Method

```
<script>
var s1="javascript from javatpoint indexof";
var n=s1.lastIndexOf("java");
document.write(n);
</script>
```

**Output:**

16

5) JavaScript String toLowerCase() Method

```
<script>
var s1="JavaScript toLowerCase Example";
var s2=s1.toLowerCase();
document.write(s2);
</script>
```

**Output:**

javascript tolowercase example

6) JavaScript String toUpperCase() Method

```
<script>
var s1="JavaScript toUpperCase Example";
var s2=s1.toUpperCase();
document.write(s2);
</script>
```

**Output:**

JAVASCRIPT TOUPPERCASE EXAMPLE

7) JavaScript String slice(beginIndex, endIndex) Method

```
<script>
var s1="abcdefghijklm";
```

```
var s2=s1.slice(2,5);
document.write(s2);
</script>
```

**Output:**

Cde

8) JavaScript String trim() Method

```
<script>
var s1=" javascript trim  ";
var s2=s1.trim();
document.write(s2);
</script>
```

**Output:**

javascript trim

9) JavaScript String split() Method

```
<script>
var str="This is JavaTpoint website";
document.write(str.split(" ")); //splits the given string.
</script>
```

**Output**

'This', 'is', 'JavaTpoint', 'website'

### ✓ String search method

**Example 1:** Search for "Blue":

```
let text = "Mr. Blue has a blue house"
let position = text.search("Blue");
document.getElementById("demo").innerHTML = position;
```

**Output**

4

**Example2:** Search for "blue":

```
let text = "Mr. Blue has a blue house"
let position = text.search("blue");
document.getElementById("demo").innerHTML = position;
```

**output**

15

### ✓ String Template literals

Here are the steps to use String Template literals in JavaScript:

1. Declare a string variable: Declare a string variable using the let, const, or var Keyword.
2. Use backticks (`): Instead of single quotes ('') or double quotes ("") used in traditional strings, enclose the string within backticks (`).
3. Embed expressions and variables: Inside the template literal, use `\${}` to embed expressions or variables within the string. Place the expressions or variables inside the curly braces.
4. Evaluate the expressions: The expressions or variables within the template literal are evaluated and their values are inserted into the resulting string.

#### **Example of using String Template literals in JavaScript:**

Step 1: Declare a string variable

```
let name = "John";
```

```
let age = 25;
```

// Step 2 and 3: Use backticks and embed expressions

```
let message = `My name is ${name} and I am ${age} years old.`;
```

```
console.log(message);
```

// Output: My name is John and I am 25 years old.

In the example above, the template literal is used to create a string that includes the values of the `name` and `age` variables. The expressions `\${name}` and `\${age}` are evaluated, and their values are inserted into the resulting string.

#### **For example,**

```
const name = 'Jack';
console.log(`Hello ${name}!`); // Hello Jack!
```

#### **Template Literals for Strings**

#### **For example,**

```
const str1 = 'This is a string';
// cannot use the same quotes
const str2 = 'A "quote" inside a string'; // valid code
const str3 = 'A 'quote' inside a string'; // Error
const str4 = "Another 'quote' inside a string"; // valid code
const str5 = "Another "quote" inside a string"; // Error
```

To use the same quotations inside the string, you can use the escape character \.

```
// escape characters using \
const str3 = 'A \'quote\' inside a string'; // valid code
const str5 = "Another \"quote\" inside a string"; // valid code
```

Instead of using escape characters, you can use template literals. For example,

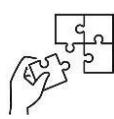
```
const str1 = `This is a string`;
const str2 = `This is a string with a 'quote' in it`;
const str3 = `This is a string with a "double quote" in it`;
```

As you can see, the template literals not only make it easy to include quotations but also make our code look cleaner...



## Points to Remember

- A string is a data type used to represent textual data. It is a sequence of characters enclosed within single quotes ("), double quotes (""" or backticks (`) and the Strings can contain letters, numbers, symbols, and special characters. String concatenation means to append one or more strings to the end of another string. To concatenate strings we can use the concat() method, the '+' operator, the array join() method or template literals.
- To manipulate and work with the string use various string methods and operations. This includes concatenation, slicing, replacing, converting case, and more.



## Application of learning 2.1.

You are tasked to work on a web application that allows users to perform basic string manipulation operations. Users can input a string, and the application provides them with options to perform operations like reversing the string, converting it to uppercase or lowercase, and counting the number of characters.



## Indicative content 2.2: Using conditional statements



Duration: 6 hrs



### Theoretical Activity 2.2.1 Description of conditional statement



#### Tasks:

- 1: In small groups, you are requested to answer the following questions related to the conditional statement
  - i. What do you understand about the term “conditional statements?”
  - ii. What are different types the conditional statements?
  - iii. Give the syntax of the switch statement.
- 2: Participate in group activity and write answers provided on flipchart/paper
- 3: Discuss on the provided answer and choose the correct answer.
- 4: For more clarification, read the key readings 2.2.1.
- 5: Ask questions for more clarification where necessary.



#### Key readings 2.2.1.:

##### JavaScript conditional statements

Conditional statements in JavaScript are programming constructs that allow you to control the flow of your code based on specific conditions. They enable you to make decisions and execute different blocks of code depending on whether certain conditions are true or false.

While writing a program, there may be a situation when you need to adopt one out of a given set of paths. In such cases, you need to use conditional statements that allow your program to make correct decisions and perform right actions.

Conditional statements help you create logic that responds to user input, handles different scenarios, and performs actions based on specific conditions. They allow your code to adapt and make decisions dynamically at runtime.

**In JavaScript, there are several types of conditional statements:**

##### **1. JavaScript If statement**

It evaluates the content only if the expression is true. The signature of JavaScript if statement is given below.

The if statement is used to execute a block of code if a specified condition is true. If the condition evaluates to true, the code block associated with the if statement is executed.

#### Syntax:

```
if(expression){  
    //content to be evaluated  
}
```

### 2. If-else statement in JavaScript

If-else statement computes the content whether the condition is true or false. It means if the condition is true then print some statements otherwise print some different statements.

It is used to *execute the code whether condition is true or false*. If the condition is true, the code block associated with the if statement is executed. Otherwise, if the condition is false, the code block associated with the else statement is executed.

#### Here is the syntax of if-else statement:

```
if(expression)  
{  
    //content that is to be evaluated if condition is true  
}  
else  
{  
    //content that is to be evaluated if condition is false  
}
```

### 3. If-else-if statement in JavaScript

If-else-if statement computes the content only if expression is true from several expressions.

When we want to use condition more than one time, then we can use Nested if-else.

The if.else ifelse statement allows you to test multiple conditions and execute different blocks of code based on the first condition that evaluates to true. It provides a way to handle multiple possible outcomes.

#### syntax:

```
if(expression1)  
{  
    //content that is to be evaluated if expression1 is true  
}
```

```
else if(expression2)
{
//content that is to be evaluated if expression2 is true
}
else if(expression3)
{
//content that is to be evaluated if expression3 is true
}
else
{
//content that is to be evaluated if no expression is true
}
```

#### **4. The conditional (ternary) operator**

The **conditional (ternary) operator** is the only JavaScript operator that takes three operands:

a condition followed by a question mark (?), then an expression to execute if the condition is true followed by a colon (:), and finally the expression to execute if the condition is false.

This operator is frequently used as an alternative to an if...else statement.

##### **Syntax:**

condition? exprIfTrue: exprIfFalse

Parameters:

##### **Condition:**

An expression whose value is used as a condition.

##### **exprIfTrue:**

An expression which is executed if the condition evaluates to a true value.

##### **exprIfFalse:**

An expression which is executed if the condition is false.

##### **Description**

Besides false, possible falsy expressions are: null, NaN, 0, the empty string (""), and undefined.

If condition is any of these, the result of the conditional expression will be the result of executing the expression exprIfFalse.

#### **5. JavaScript Switch**

In order to execute one code from multiple expressions, JavaScript Switch is used. It is known to be very similar to the if-else statement and due to the permitted use of numbers and characters it is more convenient than if-else-if.

The switch statement is used to perform different actions based on different cases. It evaluates an expression and executes the code block associated with the matching case. If no case matches, an optional default case can be executed.

Here is the signature of JavaScript Switch Statement:

```
switch(expression)
{
    case value_1:
        code that is to be executed;
        break;
    case value_2:
        code that is to be executed;
        break;
    .
    .
    .
    .
    case value_n:
        code that is to be executed;
        break;
    default:
        code that is to be executed if above values are not matched;
}
```



### Practical Activity 2.2.2: Use of conditional statement



#### Task:

- 1: Referring to the previous theoretical activities (2.2.1) you are requested to go to the computer lab to use different types of conditional statements in the JavaScript program. This task should be done individually.
- 2: Read the key reading 2.2.2 in trainee manual about use of conditional statements in JavaScript.
- 3: From explanation and instructions provided by the trainer use different types of conditional statements in JavaScript program.
- 4: Present your work to the trainer and whole class.
- 5: Ask questions for clarification where necessary



## Key readings 2.2.2

Here are the steps to use conditional statements in JavaScript:

- 1. Identify the condition:** Determine the condition that you want to evaluate. This condition could be based on user input, variable values, or any other logical expression.
- 2. Choose the appropriate conditional statement:** Select the appropriate conditional statement based on your needs. You can use the if statement, if...else statement, if...else if...else statement, or switch statement.
- 3. Write the code block:** Inside the conditional statement, write the code block that should be executed if the condition is true. This code block can contain one or more statements.
- 4. Handle the alternative case (if applicable):** If you are using an if...else statement or if...else if...else statement, write the code block that should be executed if the condition is false or if none of the conditions are true.
- 5. Evaluate the condition:** The condition is evaluated based on the logical expression or comparison. If the condition evaluates to true, the code block associated with the condition is executed. Otherwise, the code block associated with the alternative case is executed (if applicable).

Example that demonstrates the steps:

// Step 1: Identify the condition

```
let num = 10;
```

// Step 2: Choose the appropriate conditional statement

```
if (num > 0) {
```

// Step 3: Write the code block

```
    console.log("The number is positive.");
```

```
}
```

```
else
```

```
{
```

// Step 4: Handle the alternative case

```
    console.log("The number is zero or negative.");
```

```
}
```

In the example above, the if statement is used to check if the variable `num` is greater than 0. If the condition is true, the code block inside the if statement is executed and "The number is positive." is printed to the console. Otherwise, if the condition is false, the code block inside the else statement is executed and "The number is zero or negative." is printed.

## **1. JavaScript If statement**

**Here are the steps to use the if statement in JavaScript:**

1. Identify the condition: Determine the condition that you want to evaluate. This condition can be a logical expression, a comparison, or any other condition that results in a boolean value (true or false).
2. Write the if statement: Start by writing the if keyword, followed by a pair of parentheses (). Inside the parentheses, write the condition that you identified in step 1.
3. Define the code block: After the closing parenthesis ), open a pair of curly braces {}. Inside the curly braces, write the code block that should be executed if the condition evaluates to true.
4. Execute the code block: If the condition evaluates to true, the code block inside the curly braces will be executed. This code block can contain one or more statements.

**Here's an example that demonstrates the steps:**

```
// Step 1: Identify the condition  
let num = 10;  
// Step 2: Write the if statement  
if (num > 0) {  
    // Step 3: Define the code block  
    console.log("The number is positive.");  
}  
// Output: The number is positive.
```

In the example above, the if statement is used to check if the variable `num` is greater than 0. If the condition is true, the code block inside the if statement (console.log statement) is executed, and "The number is positive." is printed to the console.

**Example:2**

```
<script>  
var x=22;  
if(x>11)  
{  
document.write("value of x is bigger than 11");  
}  
</script>
```

**Output:**

value of x is bigger than 11.

## 2. If-else statement in JavaScript

**Here are the steps to use the if...else statement in JavaScript:**

1. Identify the condition: Determine the condition that you want to evaluate. This condition can be a logical expression, a comparison, or any other condition that results in a boolean value (true or false).
2. Write the if...else statement: Start by writing the if keyword, followed by a pair of parentheses (). Inside the parentheses, write the condition that you identified in step 1.
3. Define the code block for the if condition: After the closing parenthesis), open a pair of curly braces {}. Inside the curly braces, write the code block that should be executed if the condition evaluates to true.
4. Write the else keyword: After the closing curly brace } of the if code block, write the else keyword.
5. Define the code block for the else condition: After the else keyword, open another pair of curly braces {}. Inside the curly braces, write the code block that should be executed if the condition evaluates to false.
6. Execute the appropriate code block: If the condition evaluates to true, the code block inside the if statement will be executed. Otherwise, if the condition evaluates to false, the code block inside the else statement will be executed.

**Here's an example that demonstrates the steps:**

```
// Step 1: Identify the condition  
let num = 10;  
  
// Step 2: Write the if...else statement  
if (num > 0) {  
    // Step 3: Define the code block for the if condition  
    console.log("The number is positive.");  
} else {  
    // Step 5: Define the code block for the else condition  
    console.log("The number is zero or negative.");  
}  
  
// Output: The number is positive.
```

In the example above, the if...else statement is used to check if the variable `num` is greater than 0. If the condition is true, the code block inside the if statement (console.log

statement) is executed, and "The number is positive." is printed to the console. Otherwise, if the condition is false, the code block inside the else statement is executed, and "The number is zero or negative." is printed.

### **Example:2**

```
<script>
var x=10;
if(x%2==0)
{
document.write("x is even number");
}
else
{
document.write("x is odd number");
}
</script>
```

### **Output:**

x is even number

### **3. If-else-if statement in JavaScript**

**Here are the steps to use the if...else if...else statement in JavaScript:**

1. Identify the conditions: Determine the conditions that you want to evaluate. These conditions can be logical expressions, comparisons, or any other conditions that result in boolean values (true or false).
2. Write the if...else if...else statement: Start by writing the if keyword, followed by a pair of parentheses (). Inside the parentheses, write the first condition that you identified in step 1.
3. Define the code block for the if condition: After the closing parenthesis), open a pair of curly braces {}. Inside the curly braces {}, write the code block that should be executed if the first condition evaluates to true.
4. Write the else if keyword: After the closing curly brace} of the if code block, write the else if keyword, followed by another pair of parentheses (). Inside the parentheses, write the second condition that you identified in step 1.
5. Define the code block for the else if condition: After the closing parenthesis), open another pair of curly braces {}. Inside the curly braces, write the code block that should be executed if the second condition evaluates to true.
6. Repeat steps 4 and 5 for additional else if conditions: If you have more conditions to evaluate, repeat steps 4 and 5 for each additional condition.

7. Write the else keyword: After all the else if conditions, write the else keyword.
8. Define the code block for the else condition: After the else keyword, open another pair of curly braces {}. Inside the curly braces, write the code block that should be executed if none of the previous conditions evaluate to true.
9. Execute the appropriate code block: The code block associated with the first condition that evaluates to true will be executed. If none of the conditions evaluate to true, the code block inside the else statement will be executed.

**Here's an example that demonstrates the steps:**

```
// Step 1: Identify the conditions
let num = 10;
// Step 2: Write the if...else if...else statement
if (num > 0) {
    // Step 3: Define the code block for the if condition
    console.log("The number is positive.");
} else if (num === 0) {
    // Step 5: Define the code block for the else if condition
    console.log("The number is zero.");
} else {
    // Step 8: Define the code block for the else condition
    console.log("The number is negative.");
}
``javascript
// Step 1: Identify the conditions
let num = 10;
// Step 2: Write the if...else if...else statement
if (num > 0) {
    // Step 3: Define the code block for the if condition
    console.log("The number is positive.");
} else if (num === 0) {
    // Step 5: Define the code block for the else if condition
    console.log("The number is zero.");
} else {
    // Step 8: Define the code block for the else condition
    console.log("The number is negative.");
}
```

**Output: The number is positive.**

In the example above, the if...else if...else statement is used to check the value of the variable `num`. The conditions are evaluated in order. If the first condition (num > 0) is true, the code block inside the if statement is executed. If the first condition is false, the

second condition (`num === 0`) is evaluated.

If the second condition is true, the code block inside the `else if` statement is executed. If both the first and second conditions are false, the code block inside the `else` statement is executed.

By following these steps, you can use the `if...else if...else` statement in JavaScript to conditionally execute different code blocks based on multiple conditions.

### **Example:2**

```
<script>
var x=10;
if(x==5){
document.write("x is equal to 5");
}
else if(x==10){
document.write("x is equal to 10");
}
else if(x==15){
document.write("x is equal to 15");
}
else{
document.write("x is not equal to 5, 10 or 15");
}
</script>
```

### **Output:**

x is equal to 10

### **4. Conditional (ternary) operator**

**Here are the steps to use the conditional (ternary) operator in JavaScript:**

1. Identify the condition: Determine the condition that you want to evaluate. This condition can be a logical expression, a comparison, or any other condition that results in a boolean value (true or false).
2. Write the conditional (ternary) operator: Start by writing the condition, followed by a question mark (?).
3. Define the expression or value for the true case: After the question mark (?), write the expression or value that should be returned if the condition evaluates to true.

4. Write a colon (:): After the expression or value for the true case, write a colon (:).
5. Define the expression or value for the false case: After the colon (:), write the expression or value that should be returned if the condition evaluates to false.
6. Evaluate the condition: The condition is evaluated based on the logical expression or comparison. If the condition is true, the expression or value for the true case is returned. Otherwise, if the condition is false, the expression or value for the false case is returned.

**Here's an example that demonstrates the steps:**

```
// Step 1: Identify the condition  
let num = 10;  
// Step 2: Write the conditional (ternary) operator  
let result = (num > 0) ? "The number is positive." : "The number is zero or negative."  
console.log(result);  
// Output: The number is positive.
```

In the example above, the conditional (ternary) operator is used to check if the variable `num` is greater than 0. If the condition is true, the expression "The number is positive." is assigned to the variable `result`. Otherwise, if the condition is false, the expression "The number is zero or negative." is assigned to the variable `result`.

By following these steps, you can use the conditional (ternary) operator in JavaScript to conditionally assign values or expressions based on the evaluation of a condition.

### **Example:2**

```
const age = 26;  
const beverage = age >= 21 ? "Beer" : "Juice";  
console.log(beverage); // "Beer"
```

## **5.Javascript Switch**

**Here are the steps to use the switch statement in JavaScript:**

1. Identify the expression: Determine the expression whose value you want to compare. This expression can be a variable, a function call, or any other valid JavaScript expression.
2. Write the switch statement: Start by writing the switch keyword, followed by a pair of parentheses (). Inside the parentheses, write the expression that you identified in step 1.
3. Define the cases: After the closing parenthesis), open a pair of curly braces {}. Inside the curly braces, write multiple case statements.

4. Write the case keyword followed by a value: For each case, write the case keyword followed by a value that you want to compare the expression against. This value can be a number, a string, or any other valid JavaScript value.
5. Define the code block for each case: After the value, write a colon (:). After the colon, write the code block that should be executed if the expression matches the case value. This code block can contain one or more statements.
6. Write the break statement: After each code block, write the break keyword. This is important to ensure that only the code block associated with the matched case is executed, and the switch statement exits.
7. Optionally, write the default case: After all the case statements, you can write the default keyword followed by a colon (:). After the colon, write the code block that should be executed if none of the case values match the expression. This code block is optional.
8. Execute the appropriate code block: The code block associated with the case whose value matches the expression will be executed. If none of the case values match and a default case is provided, the code block inside the default case will be executed.

**Here's an example that demonstrates the steps:**

```
// Step 1: Identify the expression
let day = "Monday";
// Step 2: Write the switch statement
switch (day) {
    // Step 3 & 4: Define the case
    case "Monday":
        // Step 5: Define the code block for the case
        console.log("It's Monday.");
        // Step 6: Write the break statement
        break;
    case "Tuesday":
        console.log("It's Tuesday.");
        break;
    case "Wednesday":
        console.log("It's Wednesday.");
        break;
    // Step 7: Optionally, write the default case
    default:
        console.log("It's another day of the week.");
}
```

```
// Output: It's Monday.
```

In the example above, the switch statement is used to check the value of the variable `day`. The expression is compared against each case value. If a case value matches the expression, the code block associated with that case is executed. In this case, since `day` is "Monday", the code block inside the first case statement is executed, and "It's Monday." is printed to the console.

#### **Example:2**

```
<script>
    var Class='B';
    var result;

    switch(Class)
    {
        case 'A':
            result="A Class";
            break;
        case 'B':
            result="B Class";
            break;
        case 'C':
            result="C Class";
            break;
        default:
            result="No class";
    }
    document.write(result);
</script>
```

#### **Output:**

The output for the above-mentioned example is B Class.

#### **Example 3: (without using Break)**

```
<script>
    var Class='B';
    var result;
    switch(Class)
    {
        case 'A':
            result=" A Class";
```

```
case 'B':  
    result=" B Class";  
    case 'C':  
        result=" C Class";  
    default:  
        result=" No Class";  
    }  
    document.write(result);  
</script>
```

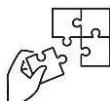
**Output:**

No Class



**Points to Remember**

- **Conditional statements** in JavaScript are programming constructs that allow you to control the flow of your code based on specific conditions. They help you to create logic that responds to user input, handles different scenarios, and performs actions based on specific conditions. There are different types of conditional statements and these are: If statement, If-else statement, If-else-if statement and Switch case statement.
- For using any conditional statement, you need to determine the condition that you want to evaluate and select the appropriate conditional statement based on your task.



**Application of learning 2.2.**

As a software developer, you are hired by your school to develop a web application that allows trainer to input marks of trainees, and the application will determine and display the largest mark from a list of entered marks JavaScript.



## Indicative content 2.3: Using Loop functions in JavaScript



Duration: 8 hrs



### Theoretical Activity 2.3.1: Description of Loop functions in JavaScript



#### Tasks:

- 1: In small groups, you are requested to answer the following questions related to the loop functions in JavaScript:
  - i. What do you understand about the term “Loop”.
  - ii. List types of loops available in JavaScript.
  - iii. Differentiate while loop from do... while loop in JavaScript.
- iv. JavaScript loops offer several advantages. Give some of them.
- 2: Provide the answer for the asked questions and write them on papers.
- 3: Present the findings/answers to the whole class
- 4: Discuss on the provided answers and choose the correct answer
- 5: For more clarification, read the key readings 2.3.1 and ask questions where necessary.



#### Key readings 2.3.1.:

##### JavaScript Loop

A **JavaScript loop** is a programming construct that allows you to repeat a block of code multiple times.

A loop can also be defined as a sequence of instructions that is continually repeated until a certain condition is reached.

It provides a way to automate repetitive tasks, iterate over data structures, and control the flow of execution. JavaScript offers several types of loops

It is generally used in arrays and helps in making the code compact.

**In JavaScript, there are generally five types of loops:**

1. for loop
2. while loop
3. do-while loop
4. for-in loop
5. for-of loop

Below is description of different types of loops:

##### 1. For Loop in JavaScript

The For Loop in JavaScript iterates the elements for a finite number of times and should be used when the number of iterations is known. It is commonly used when you know the number of iterations in advance.

For loop consists of an initialization statement, a condition, and an increment or decrement statement. The loop continues executing as long as the condition is true.

**Here is the syntax mentioned below:**

```
for (initialization; condition; increment/decrement)
{
    code that you want to be executed
}
```

## 2. While Loop in JavaScript

While loop in JavaScript iterates the elements for infinite times and it is generally used if the number of iterations is unknown.

The while loop is useful when you want to repeat a block of code while a certain condition is true. The condition is checked before each iteration, and if it evaluates to true, the loop continues. It's important to ensure that the condition eventually becomes false to avoid an infinite loop.

**Here is the syntax of while loop:**

```
while (condition)
{
    code that you wants to be executed
}
```

## 3. Do-While Loop in JavaScript

Just like a while loop, the do-while loop iterates the elements for an infinite number of times. But, the code is executed at least one time despite the fact that the condition is true or false.

The do...while loop is similar to the while loop, but it ensures that the block of code executes at least once before checking the condition. It continues to execute as long as the condition remains true.

**Here is the syntax of do-while loop:**

```
do
{
    code that you want to be executed
}
while (condition);
```

## 4. For-in Loop in JavaScript

The for-in loop in JavaScript is generally used to iterate the properties of an object/Associative array.

The for...in loop is used to iterate over the properties of an object. It loops through each enumerable property of an object, assigning the property key to a variable in each iteration.

**Here is the syntax for for-in loop:**

```
for (Tempvariablename in object/array)
{
    statement or code to be executed
}
```

## 5. The For Of Loop

The JavaScript for of statement loops through the values of an iterable object. It lets you loop over iterable data structures such as Arrays, Strings, Maps, NodeLists, and more.

The for...of loop is used to iterate over iterable objects like arrays, strings, or other collections. It provides a convenient way to access each element or item of the iterable without explicitly dealing with indexes.

### Syntax

```
for (variable of iterable)
{ // code block to be executed
}
```

### Difference between while loop and do while loop in JavaScript

The main difference between a while loop and a do-while loop in JavaScript lies in when the condition is evaluated.

1. In a while loop, the condition is checked before each iteration. If the condition evaluates to true, the loop body is executed. If the condition is initially false, the loop body is skipped entirely, and the program moves on to the next statement after the loop.
2. In a do-while loop, the condition is checked after each iteration. This means that the loop body is executed at least once, regardless of the condition's initial value. If the condition evaluates to true, the loop continues executing. If the condition is false, the loop terminates, and the program moves on to the next statement after the loop.

### The advantage of using loops in JavaScript programming are:

- 1. Efficiency:** Loops allow you to perform repetitive tasks efficiently by executing a block of code multiple times. This helps in automating processes and reduces the need for writing repetitive code manually.
- 2. Code reusability:** By using loops, you can write reusable code that can be applied to different scenarios. Instead of repeating the same code multiple times, you can encapsulate it within a loop and iterate over the desired data set.
- 3. Iterating over data structures:** Loops enable you to iterate over arrays, objects, and other data structures, allowing you to access and manipulate each element

individually. This makes it easier to perform operations such as filtering, sorting, or transforming data.

**4. Dynamic iteration:** Loops provide flexibility for dynamically determining the number of iterations based on runtime conditions or user input. This allows your code to adapt and handle varying data sizes or changing requirements.

**5. Nested loops:** JavaScript supports nesting loops, which allows you to iterate over nested data structures or perform complex iterations. This capability is useful when working with multidimensional arrays or performing operations that require multiple levels of iteration.

**6. Control flow:** Loops provide control flow statements like break and continue, which allow you to modify the flow of execution within a loop. The break statement can be used to exit a loop prematurely, while the continue statement allows you to skip the current iteration and move to the next one.

**7. Code readability and maintainability:** By using loops, you can write more concise and readable code. Loops make it clear that a certain block of code will be repeated, improving code organization and making it easier to understand and maintain.

Overall, loops in JavaScript provide a powerful mechanism for automating repetitive tasks, iterating over data structures, and optimizing code efficiency. They contribute to code reusability, readability, and maintainability, making them an essential tool for JavaScript programming.



### Practical Activity 2.3.2: Use Loop functions in JavaScript



#### Task:

- 1: Referring to the previous theoretical activities (2.3.1) you are requested to go to the computer lab and use loops in the JavaScript program. This task should be done individually.
- 2: Read the key reading 2.3.2 in trainee manual about using loops in JavaScript program.
- 3: Flow instructions and steps provided by the trainer to use loop functions JavaScript program.
- 4: Referring to the outlined steps provided on task 3, use loop functions to develop your own JavaScript program.
- 5: Ask questions for more clarification where necessary.



### Key readings 2.3.2

Here are the steps to use loop functions in JavaScript:

- 1. Determine the task:** Identify the specific task or set of instructions that need to be repeated.
- 2. Choose the appropriate loop type:** JavaScript offers different loop types, such as for, while, do...while, for...in, and for...of loops. Select the loop type that best suits your task based on its requirements and the data structure you are working with.
- 3. Initialize loop variables where necessary:** If using a loop that requires initialization, like a for loop, define and initialize the loop control variables. This includes setting the initial value of the loop counter or any other variables needed for the loop's execution.
- 4. Set the loop condition:** Define the condition that will be evaluated before each iteration. This condition determines whether the loop should continue executing or terminate. Ensure that the condition eventually becomes false to avoid infinite loops.
- 5. Write the loop body:** Inside the loop, write the block of code that will be executed with each iteration. This code should contain the instructions or operations to be performed on the data being looped over.
- 6. Update loop variables:** If necessary, update the loop variables within the loop body. This includes incrementing or decrementing the loop counter or modifying any other variables that control the loop's execution.
- 7. Control the loop flow:** Utilize control statements like break and continue to modify the flow of the loop when needed. The break statement allows you to exit the loop prematurely, while the continue statement skips the current iteration and moves to the next one.
- 8. Test and debug:** Test your loop function with different inputs and data sets to ensure it behaves as expected. Debug any issues or errors that may arise during testing.

By following these steps, you can effectively utilize loop functions in JavaScript to automate repetitive tasks, iterate over data structures, and control the flow of execution in your code.

#### For Loop in JavaScript

Here are the steps to use a for loop in JavaScript:

- 1. up the for-loop syntax:** Start by writing the for loop statement, which consists of three parts enclosed in parentheses and separated by semicolons:

- 2. Initialize loop variables:** In the initialization part, set the initial value of the loop counter or any other variables needed for the loop's execution. This code block typically executes only once before the loop begins.
- 3. Define the loop condition:** In the condition part, specify the condition that will be evaluated before each iteration. The loop will continue executing as long as the condition evaluates to true.
- 4. Write the loop body:** Inside the curly braces following the for statement, write the block of code that will be executed with each iteration. This code block contains the instructions or operations you want to perform on each iteration of the loop.
- 5. Update loop variables:** If needed, update the loop variables within the loop body. This includes incrementing or decrementing the loop counter or modifying any other variables that control the loop's execution.
- 6. Test and debug:** Test your for loop with different inputs and data sets to ensure it behaves as expected. Debug any issues or errors that may arise during testing.

By following these steps, you can effectively use a for loop in JavaScript to iterate over a range of values and execute a block of code multiple times.

**Its syntax is as follows:**

```
for (initialization; condition; increment/decrement) {  
    // Loop body  
}
```

**Example 1:**

```
<script>  
for (i=2; i<=6; i++)  
{  
    document.write(i + "<br/>")  
}  
</script>
```

Here is the output for the above mentioned

Example:

```
2  
3  
4  
5  
6
```

**Example 2 (Sum of 1 to 100)**

```
<script>  
    var sum=0;  
    for (var i=1; i<=100; i++)
```

```
{  
    sum=sum+i;  
}  
document.write("Sum of 1 to 100="+sum);  
</script>
```

Here is the output for the above mentioned example:

Sum of 1 to 100=5050

### While Loop in JavaScript

**Here are the steps to use a while loop in JavaScript:**

- 1. Set up the initial condition:** Before starting the while loop, define the initial condition that will determine whether the loop should execute or not.
- 2. Define the loop condition:** Inside the parentheses following the while keyword, specify the condition that will be evaluated before each iteration. The loop will continue executing as long as the condition evaluates to true.
- 3. Write the loop body:** Inside the curly braces following the while statement, write the block of code that will be executed with each iteration. This code block contains the instructions or operations you want to perform on each iteration of the loop.
- 4. Update loop variables:** If needed, update the loop variables within the loop body. This includes modifying any variables that control the loop's execution or the condition itself.
- 5. Test and debug:** Test your while loop with different inputs and data sets to ensure it behaves as expected. Debug any issues or errors that may arise during testing.

By following these steps, you can effectively use a while loop in JavaScript to repeatedly execute a block of code as long as a specified condition remains true.

**Its syntax is as follows:**

```
while (condition)  
{  
// Loop body  
// Perform operations or updates  
}
```

**Example1:**

```
<script>  
    var i=10;
```

```
while(i<=12)
{
document.write(i + "<br/>");
i++;
}
</script>
```

The output of the above-mentioned example will be as follows:

```
10
11
12
```

### **Example 2 (Print table of 5)**

```
script>
    var table=5;
    var i=1;
    var tab=0;
    while(i<=10)
    {
        var tab=table*i;
        document.write(tab + " ");
        i++;
    }
</script>
```

The output of the above-mentioned example will be as follows:

```
5 10 15 20 25 30 35 40 45 50
```

## **Do-While Loop in JavaScript**

**Here are the steps to use a do-while loop in JavaScript:**

- 1. Set up the initial condition:** Start by writing the do keyword followed by an opening curly brace. This will indicate the start of the do-while loop.
- 2. Write the loop body:** Inside the curly braces, write the block of code that will be executed with each iteration. This code block contains the instructions or operations you want to perform on each iteration of the loop.
- 3. Update loop variables:** If needed, update the loop variables within the loop body. This includes modifying any variables that control the loop's execution or the condition itself.
- 4. Define the loop condition:** After the closing curly brace of the loop body, write the while keyword followed by the condition that will be evaluated after each iteration. The loop will continue executing as long as the condition evaluates to true.

**5. Test and debug:** Test your do-while loop with different inputs and data sets to ensure it behaves as expected. Debug any issues or errors that may arise during testing.

By following these steps, you can effectively use a do-while loop in JavaScript to repeatedly execute a block of code at least once, and then continue executing as long as a specified condition remains true.

**Its syntax is given below:**

```
do {  
    // Loop body  
    // Perform operations or updates  
}  
while (condition);
```

**Example:1**

```
<script>  
    var i=20;  
    do  
    {  
        document.write(i + "<br/>");  
        i++;  
    }  
    while (i<=23);  
</script>
```

**Here is the output for the above-mentioned example:**

```
20  
21  
22  
23
```

### **For-in Loop in JavaScript**

**Here are the steps to use a for...in loop in JavaScript:**

- 1. Identify the object:** Determine the object over which you want to iterate using the for...in loop. This can be an array, an object, or any other iterable data structure.
- 2. Set up the for...in loop syntax:** Start by writing the for...in loop statement, which consists of the keyword "for" followed by parentheses and curly braces:
- 3. Define the loop variable:** Inside the parentheses, declare a variable that will represent each property or index of the object during each iteration of the loop. This variable will take on a different value with each iteration.

**4. Write the loop body:** Inside the curly braces, write the block of code that will be executed with each iteration. This code block contains the instructions or operations you want to perform on each property or index of the object.

**5. Access the object's properties or indices:** Within the loop body, you can access the properties or indices of the object using the loop variable. Perform any desired operations or access the values associated with each property or index.

**6. Test and debug:** Test your for...in loop with different objects and data sets to ensure it behaves as expected. Debug any issues or errors that may arise during testing.

By following these steps, you can effectively use a for...in loop in JavaScript to iterate over the properties or indices of an object and perform operations on each property or index.

**Its syntax is as follows:**

```
for (variable in object)
{
// Loop body
}
```

**Example:1**

```
<html>
<body>
    <script type = "text/javascript">
        var user = {fname:"sanjeev", lname:"kumar",country:"india"};
        var x;
        for (x in user)
        {
            document.write(x+": "+user[x]+"<br>");
        }
    </script>
</body>
</html>
```

**Here is the output for the above mentioned example:**

fname: sanjeev

lname: kumar

country: india

 **The For of Loop**

**Here are the steps to use a for...of loop in JavaScript:**

**1. Identify the iterable object:** Determine the object or data structure over which you want to iterate using the for...of loop. This can be an array, a string, a Set, a Map, or any other iterable object.

**2. Set up the for...of loop syntax:** Start by writing the for...of loop statement, which consists of the keyword "for" followed by parentheses and curly braces:

**3. Define the loop variable:** Inside the parentheses, declare a variable that will represent each element or value of the iterable object during each iteration of the loop. This variable will take on a different value with each iteration.

**4. Write the loop body:** Inside the curly braces, write the block of code that will be executed with each iteration. This code block contains the instructions or operations you want to perform on each element or value of the iterable object.

**5. Access the elements or values of the iterable object:** Within the loop body, you can access and work with the individual elements or values of the iterable object using the loop variable. Perform any desired operations or access the values associated with each element or value.

**6. Test and debug:** Test your for...of loop with different iterable objects and data sets to ensure it behaves as expected. Debug any issues or errors that may arise during testing.

By following these steps, you can effectively use a for...of loop in JavaScript to iterate over the elements or values of an iterable object and perform operations on each element or value.

**Its syntax is as follows:**

```
for (variable of iterable) {  
    // Loop body  
}
```

**Example:**

```
<html>  
<body>  
    <script type = "text/javascript">  
        let language = "JavaScript";  
        let text = "";  
        for (let x of language)  
        {  
            text += x + "<br>";  
        }  
        document.write(text);  
    </script>
```

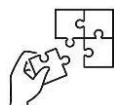
```
</body>  
</html>
```

**output**

```
J  
a  
v  
a  
S  
c  
r  
i  
p
```

**Points to Remember**

- A **JavaScript loop** is a sequence of instructions that is continually repeated until a certain condition is reached. It provides a way to automate repetitive tasks, iterate over data structures, and control the flow of execution. JavaScript offers five types of loops and these are: for loop, while loop, do-while loop, for-in loop and for-of loop. JavaScript loops offer several advantages such as Efficiency, Code reusability, Control flow, Code readability, code maintainability and more.
- When using loops, firstly you have to identify the specific task or set of instructions that need to be repeated and then choose the appropriate loop type from different types of loops. The best loop suits your task based on its requirements and the data structure you are working with is one to be selected.

**Application of learning 2.3.**

You are tasked to work on a web application that generates and displays the multiplication table for a given number. The application will iterate from 0 to 15 and show the multiplication results for each iteration. JavaScript will be used to perform the iterations and calculate the result



## Indicative content 2.4: Using Functions in JavaScript



Duration: 12 hrs



### Theoretical Activity 2.4.1: Description of functions in JavaScript



#### Tasks:

- 1: In small groups, you are requested to answer the following questions related to the functions in javascript:
  - i. Explain the term “function” as used in JavaScript programming.
  - ii. Differentiate built-in functions from user defined functions.
  - iii. Give the advantages of using functions in programming.
- 2: Provide the answer for the asked questions and write them on papers.
- 3: Present the findings/answers to the whole class
- 4: Discuss on provided answers and choose the correct answers.
- 5: For more clarification, read the key readings 2.4.1 and ask questions where necessary.



#### Key readings 2.4.1.:

##### ✓ Function's meaning

**Functions** in JavaScript are blocks of reusable code that can be defined and invoked to perform specific tasks.

A JavaScript function can also be defined as is a block of code designed to perform a particular task once called at any point in a program.

Function definition in JavaScript is the process of creating a reusable block of code that can be invoked or called to perform a specific task or operation. It involves specifying the function's name, parameters (optional), and the code to be executed when the function is invoked.

**Here are the key components of function definition in JavaScript:**

**1. Function Keyword:** The function keyword is used to declare a function in JavaScript. It is followed by the function name, parentheses, and curly braces.

**For example:**

```
function functionName()  
{
```

}

**2. Function Name:** The function name is a unique identifier for the function. It should follow the rules for naming identifiers like variables in JavaScript, such as starting with a letter or underscore, and can include letters, numbers, or underscores.

**3. Parameters:** Functions can accept parameters, which are placeholders for values that can be passed into the function when it is called. Parameters are listed inside the parentheses after the function name.

For example:

```
function addNumbers(a, b)
{
}
```

**4. Function Body:** The function body is enclosed within curly braces { }. It contains the code or operations that will be executed when the function is invoked. This is where you write the logic or instructions for the function to perform its task.

**5. Return Statement:** Functions can have a return statement to specify the value to be returned when the function is called. The return statement ends the function execution and returns the specified value. If no return statement is present, the function returns **undefined** by default.

**6. Function Invocation:** Once a function is defined, it can be invoked or called by using its name followed by parentheses. Arguments can be passed into the function inside the parentheses if the function expects parameters.

For example: `functionName();` or `functionName (2, 3);`

Function definition is a fundamental concept in JavaScript, allowing for code reuse, modularity, and the organization of logic into reusable blocks. Functions enable the encapsulation of logic, making it easier to manage and maintain complex codebases.

JavaScript functions are used to perform operations and we can call JavaScript functions many times to reuse the code.

**Below are the rules for creating a function in JavaScript:**

- Every function should begin with the keyword *function* followed by,
- A user defined function name which should be unique,
- A list of parameters enclosed within parenthesis and separated by commas,
- A list of statements composing the body of the function enclosed within curly braces {}.

**Example:**

```
function calcAddition(number1, number2)
{
```

```
    return number1 + number2;  
}
```

### ✓ Function parameters

In functions, parameters are the values or arguments that are passed to a function.

In JavaScript, the terms parameters and arguments of a function are often used interchangeably, although there exists a significant difference between them.

When we define a function, the function parameters are included. You can also specify a variable list while declaring a function, and these variables are known as **function parameters**. However, when we invoke or call the created function by passing some values, those values are called "**function arguments**".

In JavaScript, **function parameters** are placeholders for values that are passed to a function when it is invoked. They allow you to provide input to a function and make it more flexible and reusable.

**function arguments** refer to the values that are passed to a function when it is invoked. They provide a way to supply input or data to the function, allowing it to perform specific operations or calculations.

**Syntax:**

```
function name(parameter1, parameter2, parameter3) {  
    //body of the function  
}
```

Here, "x" and "y" are parameters in the following "addNumber()" function:

```
function addNumber(x, y){  
    return x+y;  
}
```

### ✓ Arrow functions

Arrow functions are anonymous functions (the functions without a name and not bound with an identifier).

They allow us to write smaller function syntax and make your code more readable and structured.

Arrow functions are anonymous functions (the functions without a name and not bound with an identifier).

They don't return any value and can be declared without the function keyword. Arrow functions cannot be used as the constructors. The context within the arrow functions is

lexically or statically defined. They are also called Lambda Functions in different languages.

Arrow functions do not include any prototype property, and they cannot be used with the new keyword.

### Syntax for defining the arrow function

```
const functionName = (arg1, arg2, ?..) => {  
    //body of the function  
}
```

#### ✓ Built-in functions

In JavaScript, built-in functions are the global functions that are called globally, rather than on an object.

Built-in functions in JavaScript are pre-defined functions that are provided as part of the JavaScript language or its standard library. They are built into the JavaScript runtime environment and can be directly used without any additional setup or import statements.

#### Examples of Built-in Functions:

- **console.log()**: Outputs messages to the console for debugging and logging purposes.
- **parseInt (), parseFloat()**: Converts strings to integers or floating-point numbers.
- **Math.random(), Math.floor(), Math.max(), Math.min()**: Perform mathematical operations and calculations.
- **Array.prototype.forEach(), Array.prototype.map(), Array.prototype.filter()**: Perform operations on arrays.
- **String.prototype.toUpperCase(), String.prototype.substring()**: Manipulate and transform strings.
- **Date.now(), Date.prototype.getTime()**: Work with dates and time.

#### ✓ Function apply

The JavaScript Function apply() method is used to call a function containing **this** value and an argument contains elements of an array.

The **apply()** method is a built-in function in JavaScript that allows you to invoke a function with a specified ‘**this**’ value and an array (or array-like object) of arguments. It is a versatile method that provides control over the execution context and the ability to pass arguments as an array.

The apply() method is called on a function object and takes two arguments: the value to be used as the “this” within the function, and an array (or array-like object) containing the arguments to be passed to the function.

**Syntax:**

```
function.apply(thisArg, [argsArray])
```

**✓ Function call**

The JavaScript Function **call()** method is used to call a function containing this value and an argument provided individually.

The “**call()**” method is a built-in function in JavaScript that allows you to invoke a function with a specified ‘this’ value and individual arguments. It is similar to the ‘apply()’ method, but instead of passing an array of arguments, you pass the arguments individually as function arguments.

The ‘call()’ method is called on a function object and takes two or more arguments: the value to be used as the ‘this’ within the function, followed by the individual arguments to be passed to the function.

**Syntax:**

```
function.call(thisArg, arg1, arg2, ...)
```

**✓ Function bind**

The JavaScript Function **bind()** method is used to create a new function.

The **bind()** method is a built-in function in JavaScript that allows you to create a new function with a specified ‘this’ value and, optionally, pre-set arguments. It returns a new function that, when called, has its ‘this’ value set to the provided value and can have arguments pre-filled.

The ‘bind()’ method is called on a function object and takes one or more arguments: the value to be used as the ‘this’ within the function, followed by any number of arguments to be pre-set.

**Syntax:**

```
function.bind(thisArg, arg1, arg2, ...)
```

**✓ Function closure**

A closure can be defined as a JavaScript feature in which the inner function has access to the outer function variable.

In JavaScript, a closure is a combination of a function and the lexical environment within which that function was declared. It allows a function to retain access to variables and parameters from its outer (enclosing) scope, even after the outer function has finished executing.

**Here are some key points about function closures in JavaScript:**

- 1. Definition:** A closure is formed when an inner function is defined within an outer function and the inner function references variables or parameters from the outer function's scope. The inner function retains access to those variables and can continue to use them even after the outer function has completed execution.
- 2. Lexical Scope:** Closures rely on lexical scoping, which means that they remember variables and parameters based on their location in the source code, rather than their value at the time the closure is created.

### ✓ Asynchronous functions

An asynchronous function is any function that delivers its result asynchronously.

Asynchronous functions in JavaScript allow for the execution of non-blocking operations, enabling efficient handling of tasks that may take time to complete, such as network requests or file operations. They utilize asynchronous programming patterns to ensure that the program doesn't get blocked while waiting for these operations to finish.

### ✓ Promise functions

In JavaScript, a Promise is an object which ensures to produce a single value in the future (when required).

Promise functions in JavaScript are a way to handle asynchronous operations and manage the flow of asynchronous code. Promises represent a future value that may be available immediately or at a later time. They provide a clean and structured way to handle success and failure scenarios of asynchronous tasks.

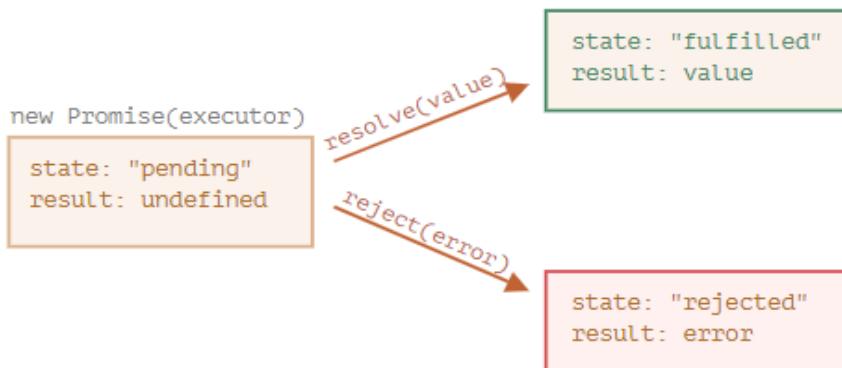
**Here are some key points about Promise functions in JavaScript:**

- 1. Promise Object:** A Promise is an object that represents the eventual completion (or failure) of an asynchronous operation and its resulting value. It is in one of three states: pending, fulfilled, or rejected.

### 2. Promise States:

The promise object in JavaScript has one of three states:

- **Pending:** The initial state before the promise settles (fulfilled or rejected).
- **Fulfilled:** The state when the promise is successfully resolved with a value.
- **Rejected:** The state when the promise encounters an error or is rejected with a reason.



### 3. Promise Methods:

- .then():** Attaches callbacks for handling the fulfillment of the promise. It takes two arguments: a callback function for the fulfillment case and an optional callback function for the rejection case.
- .catch():** Attaches a callback for handling promise rejections. It is used to handle errors that occur during the promise chain.
- .finally():** Attaches a callback that is executed regardless of whether the promise is fulfilled or rejected. It is commonly used for cleanup operations.

### ✓ Async/Await Function

**Async/Await** is an extension of **promises** that we get as language support

Async/await is a syntactic sugar in JavaScript that simplifies working with asynchronous functions and promises. It provides a more readable and synchronous-looking syntax for writing asynchronous code. Async/await builds upon promises and allows you to write asynchronous code in a more sequential and intuitive manner.

Here are some key points about **async/await** functions in JavaScript:

- 1. async Keyword:** The `async` keyword is used to define an asynchronous function. It can be placed before a function declaration or function expression. An async function always returns a promise.
- 2. await Keyword:** The `await` keyword can only be used inside an `async` function. It pauses the execution of the function until the promise is resolved or rejected. It allows you to write code that looks like synchronous code, even though it's actually asynchronous.
- 3. Sequential Execution:** With `async/await`, you can write asynchronous code in a more sequential and linear manner. The `await` keyword allows you to wait for promises to

resolve before moving to the next line of code, making the code easier to read and understand.

**4. Error Handling:** Error handling in async/await functions can be done using try-catch blocks. If an error occurs in the awaited promise, the catch block will handle the error. This simplifies error handling compared to using ` `.catch()` with promises.



### Practical Activity 2.4.2: Use of functions in JavaScript.



#### Task:

- 1: Referring to the previous theoretical activities (2.4.1) you are requested to go to the computer lab to use functions in the JavaScript. This task should be done individually.
- 2: Read the key reading 2.4.2 in trainee manual about using functions in JavaScript program.
- 3: Referring to the key reading 2.4.2, use functions in JavaScript programming.
- 4: Present your work to the trainer and whole class



### Key readings 2.4.2

#### ✓ Function Definition

To use function definition in JavaScript, follow these steps:

1. Start by deciding whether you want to use the `function` keyword or arrow function syntax `(`() => {}` )` to define your function.

2. If using the function keyword, start with the keyword followed by the name of the function.

For example: `function myFunction() { }.`

3. If using arrow function syntax, you can either use a concise arrow function `(`() => {}` )` or a block arrow function `(`() => { }` )`. For example: `const myFunction = () => {}` or `const myFunction = () => console.log('Hello')`.

4. Determine if your function needs to accept any parameters. Parameters are placeholders for values that can be passed into the function when it is called. Specify the parameters inside parentheses after the function name.

For example: `function myFunction(param1, param2) { }.`

5. Write the code or operations that you want the function to execute inside the function body. This is the block of code enclosed within curly braces `{}`.

For example:

```
function myFunction()
{
  console.log('Hello, world!');
}
```

6. If your function is expected to return a value, use the `return` statement followed by the value you want to return.

For example:

```
function addNumbers(a, b)
{
  return a + b;
}
```

7. Once your function is defined, you can invoke or call it by using its name followed by parentheses.

For example:

```
myFunction(); or addNumbers(2, 3);
```

By following these steps, you can define and use functions in JavaScript. Functions are a core concept in JavaScript programming, allowing for code reuse, modularity, and the organization of logic into reusable blocks.

Here are a few examples of how to use function in JavaScript:

### **1. Basic Function:**

```
function sayHello() {
  console.log('Hello, world!');
}

sayHello(); // Output: Hello, world!
```

### **2. Function with Parameters:**

```
function greet(name) {
  console.log('Hello, ' + name + '!');
}

greet('John'); // Output: Hello, John!
greet('Alice'); // Output: Hello, Alice!
```

### **3. Function with Return Value:**

```
function addNumbers(a, b) {  
    return a + b;  
}  
const result = addNumbers(2, 3);  
console.log(result); // Output: 5
```

### **4. Arrow Function:**

```
const multiply = (a, b) =>  
{  
    return a * b;  
}  
const product = multiply(4, 5);  
console.log(product); // Output: 20
```

### **5. Higher-Order Function:**

```
function operation(x, y, operationFunc)  
{  
    return operationFunc(x, y);  
}  
function add(a, b) {  
    return a + b;  
}  
const result = operation(3, 4, add);  
console.log(result); // Output: 7
```

### **6. Function Expression:**

```
const sayGoodbye = function()  
{  
    console.log('Goodbye!');  
}  
sayGoodbye(); // Output: Goodbye!
```

Let's see a simple example of a function in JavaScript that does not have arguments.

```
<script>  
function msg()  
{  
    alert("hello! this is message");  
}  
</script>
```

```
<input type="button" onclick="msg()" value="call function"/>
```

Output of the above example

**call function**

These examples demonstrate different ways of using function definition in JavaScript. You can define functions to perform specific tasks, accept parameters, return values, and even pass functions as arguments to other functions. Functions are versatile and powerful tools in JavaScript for organizing and executing code.

#### ✓ **JavaScript Function Arguments**

We can call functions by passing arguments. Let's see an example of a function that has one argument.

```
<script>
function getcube(number)
{
    alert(number*number*number);
}
</script>
<form>
<input type="button" value="click" onclick="getcube(4)"/>
</form>
```

Output of the above example

**call function**

#### **Function with Return Value**

We can call a function that returns a value and use it in our program. Let's see an example of a function that returns value.

```
<script>
function getInfo(){
    return "hello javatpoint! How are you?";
}
</script>
<script>
document.write(getInfo());
</script>
```

Output of the above example

hello javatpoint! How are you?

#### **JavaScript Function Object**

To use the JavaScript Function object, follow these steps:

- 1. Create a Function Object:** There are multiple ways to create a Function object in JavaScript:

- Using the `function` keyword: Define a function using the `function` keyword followed by the function name and function body. For example: `function myFunction() {}`.
- Using function expressions: Assign an anonymous function to a variable or constant.

For example:

```
const myFunction = function() {}
```

Using arrow functions: Create an arrow function and assign it to a variable or constant.

For example:

```
const myFunction = () => {}.
```

**2. Invoke the Function Object:** Once the Function object is created, you can invoke or call it by using its name followed by parentheses.

For example: `myFunction();`

**3. Pass Arguments:** If your Function object accepts parameters, you can pass arguments when invoking it. Arguments are specified inside the parentheses.

For example:

```
myFunction(arg1, arg2);
```

**4. Access Properties and Methods:** The Function object in JavaScript has several properties and methods that can be accessed. For example:

- `name`: The name of the function.
- `length`: The number of parameters the function expects.
- `toString()`: Returns the source code of the function as a string.

Here's examples that demonstrates the steps:

```
// Step 1: Create a Function Object
function greet(name) {
  console.log('Hello, ' + name + '!');
}

// Step 2: Invoke the Function Object
greet('John'); // Output: Hello, John!
// Step 3: Pass Arguments
greet('Alice'); // Output: Hello, Alice!
// Step 4: Access Properties and Methods
console.log(greet.name); // Output: greet
console.log(greet.length); // Output: 1
console.log(greet.toString()); // Output: function greet(name) { console.log('Hello, ' + name + '!'); }
```

By following these steps, you can create and use Function objects in JavaScript. Function objects are a fundamental concept in JavaScript, allowing you to define reusable blocks of code and perform specific tasks.

**Example 1:** Let's see an example to display the sum of given numbers.

```
<script>
```

```
var add=new Function("num1","num2","return num1+num2");
document.writeln(add(2,5));
</script>
```

Output of the above example: 7

**Example 2:** Let's see an example to display the power of provided value.

```
<script>
var pow=new Function("num1","num2","return Math.pow(num1,num2)");
document.writeln(pow(2,3));
</script>
```

Output of the above example: 8

### Default Parameters

```
<script>
function myFunction(x, y) {
if (y === undefined) {
y = 2;
}
return x * y;
}
document.getElementById("demo").innerHTML = myFunction(4);
</script>
```

#### output

8

#### Default Parameter Values

##### Example

If y is not passed or undefined, then y = 10.

```
<script>
function myFunction(x, y = 10) {
return x + y;
}
document.getElementById("demo").innerHTML = myFunction(5);
</script>
```

#### output

15

### Function Rest Parameter

The rest parameter (...) allows a function to treat an indefinite number of arguments as an array:

##### Example:

```
<script>
function sum(...args) {
  let sum = 0;
  for (let arg of args) sum += arg;
return sum;
}
let x = sum(4, 9, 16, 25, 29, 100, 66, 77);
document.getElementById("demo").innerHTML = x;
</script>
```

### ✓ Arrow functions

To use arrow functions in JavaScript, follow these steps:

1. Define the Arrow Function: Start by using the arrow function syntax, which consists of parentheses (optional if there's only one parameter), the arrow (`=>`) symbol, and the function body enclosed in curly braces (`{}`). For example:

- With a single parameter: `const functionName = parameter => {}`
- With multiple parameters: `const functionName = (param1, param2) => {}`

2. Write the Function Body: Inside the function body, write the code or operations that the arrow function should perform. This is the block of code enclosed within curly braces.

For example:

```
const greet = name => {
  console.log('Hello, ' + name + '!');
}
```

3. Use the Arrow Function: Once the arrow function is defined, you can invoke or call it by using its name followed by parentheses. Pass any necessary arguments inside the parentheses.

For example:

```
greet('John'); // Output: Hello, John!
greet('Alice'); // Output: Hello, Alice!
```

4. Return Values: Arrow functions can have an implicit return if the function body consists of a single expression. The result of that expression will be automatically returned. For Example:

```
const multiply = (a, b) => a * b;
console.log(multiply(2, 3)); // Output: 6
```

5. No Binding of `this`: Arrow functions do not bind their own `this` value. Instead, they inherit the `this` value from the surrounding context. This can be useful when using arrow functions as callbacks or in situations where you want to preserve the value of `this`.

By following these steps, you can define and use arrow functions in JavaScript. Arrow functions provide a concise syntax for writing functions and are especially useful for shorter, one-line functions or when you want to preserve the value of `this`.

**Example:**

```
<script>  
let myFunction = (a, b) => a * b;  
document.getElementById("demo").innerHTML = myFunction(4, 5);  
</script>
```

**Output**

20

In the following example, we are defining three functions that show Function Expression, Anonymous Function, and Arrow Function.

```
// function expression  
var myfun1 = function show() {  
    console.log("It is a Function Expression");  
}  
// Anonymous function  
var myfun2 = function () {  
    console.log("It is an Anonymous Function");  
}  
//Arrow function  
var myfun3 = () => {  
    console.log("It is an Arrow Function");  
};  
myfun1();  
myfun2();  
myfun3();
```

**Output**

It is a Function Expression

It is an Anonymous Function

It is an Arrow Function

**Syntactic Variations**

There are some syntactic variations for the arrow functions that are as follows:

- Optional parentheses for the single parameter

```
var num = x => {  
    console.log(x);  
}  
num(140);
```

**Output:** 140

- Optional braces for single statement and the empty braces if there is not any parameter required.

```
var show = () => console.log("Hello World");
```

```
show();
```

**Output:** Hello World

### Arrow Function with Parameters

If you require to pass more than one parameter by using an arrow function, then you have to pass them within the parentheses.

For example

```
var show = (a,b,c) => {  
    console.log(a + " " + b + " " + c);  
}  
show(100,200,300);
```

Output: 100 200 300

### Arrow function with default parameters

The function allows the initialization of parameters with default values, if there is no value passed to it, or it is undefined. You can see the illustration for the same in the following code:

For example

```
var show = (a, b=200) => {  
    console.log(a + " " + b);  
}  
show(100);
```

In the above function, the value of b is set to 200 by default. The function will always consider 200 as the value of b if no value of b is explicitly passed.

Output: 100 200

The default value of the parameter 'b' will get overwritten if the function passes its value explicitly.

You can see it in the following example:

```
var show = (a, b=200) => {  
    console.log(a + " " + b);  
}  
show(100,500);
```

**Output:** 100 500

### Arrow Function without Parentheses

If you have a single parameter to pass, then the parentheses are optional.

For example

```
var show = x => {  
    console.log(x);  
}  
show("Hello World");
```

Output: Hello World

### ✓ Function call

To use the `Function.call()` method in JavaScript, follow these steps:

1. Create a Function: Start by defining a function that you want to call using the Function.call() method.

For example:

```
function greet() {  
    console.log('Hello, ' + this.name + '!');  
}
```

2. Define an Object: Create an object that will be used as the context for the function call. This object will become the value of `this` within the function.

For example:

```
const person = {  
    name: 'John'  
};
```

3. Use the Function.call() Method: Call the function using the Function.call() method, passing the object as the first argument. This sets the value of `this` within the function to the provided object.

**For example:**

```
greet.call(person); // Output: Hello, John!
```

The Function.call() method allows you to invoke a function with a specified this value and optional arguments. It is useful when you want to explicitly set the context for a function call or when you need to borrow methods from other objects.

Note: The Function.call() method is just one of the ways to call a function with a specific context in JavaScript. There are also other methods like Function.apply() and the newer ES6 Function.bind() method that provide similar functionality.

Let's see a simple example of call() method.

```
<script>  
    function Emp(id,name) {  
        this.id = id;  
        this.name = name;  
    }  
    function PermanentEmp(id,name) {  
        Emp.call(this,id,name);  
    }  
    document.writeln(new PermanentEmp(101,"John Martin").name);  
</script>
```

**Output:**

John Martin

Let's see an example of call() method.

```
<script>
function Emp(id,name) {
    this.id = id;
    this.name = name;
}
function PermanentEmp(id,name) {
    Emp.call(this,id,name);
}
function TemporaryEmp(id,name) {
    Emp.call(this,id,name);
}
var p_emp=new PermanentEmp(101,"John Martin");
var t_emp=new TemporaryEmp(201,"Duke William")
document.writeln(p_emp.id+" "+p_emp.name+"<br>");
document.writeln(t_emp.id+" "+t_emp.name);
</script>
```

**Output:**

101 John Martin

201 Duke William

✓ **Function apply**

To use the Function.apply() method in JavaScript, follow these steps:

1. Create a Function: Start by defining a function that you want to call using the `Function.apply()` method.

For example:

```
function greet() {
    console.log('Hello, ' + this.name + '!');
}
```

2. Define an Object: Create an object that will be used as the context for the function call. This object will become the value of `this` within the function.

For example:

```
const person = {
    name: 'John'
};
```

3. Use the Function.apply() Method: Call the function using the Function.apply() method, passing the object as the first argument. This sets the value of `this` within the function to the provided object.

For example:

```
greet.apply(person); // Output: Hello, John!
```

The `Function.apply()` method allows you to invoke a function with a specified `this` value and an array (or array-like object) of arguments. The first argument to `'Function.apply()'` sets the value of `this` within the function, and the second argument (optional) is an array-like object containing the arguments to be passed to the function.

*Note:* The `Function.apply()` method is similar to the `Function.call()` method, but it accepts an array (or array-like object) of arguments instead of individual arguments. Both methods are useful when you want to explicitly set the context for a function call or when you need to borrow methods from other objects.

### Example 1

Let's see an example to determine the maximum element.

```
<script>
var arr = [7, 5, 9, 1];
var max = Math.max.apply(null, arr);
document.writeln(max);
</script>
```

Output: 9

### Example 2

Let's see an example to determine the minimum element.

```
<script>
var arr = [7, 5, 9, 1];
var min = Math.min.apply(null, arr);
document.writeln(min);
</script>
```

Output: 1

### ✓ Function bind

To use the `Function.bind()` method in JavaScript, follow these steps:

1. Create a Function: Start by defining a function that you want to bind using the `Function.bind()` method. For example:

```
function greet() {
  console.log('Hello, ' + this.name + '!');
}
```

2. Define an Object: Create an object that will be used as the context for the bound function. This object will become the value of `this` within the function. For example:

```
const person = {  
    name: 'John'  
};
```

3. Use the Function.bind () Method: Call the `bind()` method on the function, passing the object as the first argument. This creates a new bound function with the specified `this` value. For example:

```
const boundGreet = greet.bind(person);
```

4. Invoke the bound Function: Invoke or call the bound function as you would with any regular function. The bound function will always have the specified `this` value, regardless of how it is called. For example:

```
boundGreet(); // Output: Hello, John!
```

The Function.bind() method allows you to create a new function with a specified `this` value. It does not immediately invoke the function, but instead returns a new function that you can call later. The bound function will always have the specified `this` value, even if it is called in a different context.

*Note:* The Function.bind() method is useful when you want to create a new function with a fixed `this` value, often used in event handlers or when passing functions as callbacks.

### Example 1

Let's see a simple example of bind() method.

```
<script>  
var website = {  
    name: "Javatpoint",  
    getName: function() {  
        return this.name;  
    }  
}  
  
var unboundGetName = website.getName;  
var boundGetName = unboundGetName.bind(website);  
document.writeln(boundGetName());  
</script>
```

#### Output:

Javatpoint

#### ✓ Function closure

To use function closures in JavaScript, follow these steps:

1. Define an Outer Function: Start by defining an outer function that will serve as the closure. This function will contain the inner function and any variables that need to be accessed within the inner function. For example:

```
function outerFunction()
{
  // Variables within the outer function scope
  const outerVariable = 'Hello, ';

  function innerFunction(name) {
    // Access outer variables within the inner function
    console.log(outerVariable + name);
  }

  // Return the inner function or use it within the outer function
  return innerFunction;
}
```

2. Access Outer Variables: Within the outer function, define any variables that you want to be accessible within the inner function. These variables will be stored in the closure and retain their values even after the outer function has finished executing.

3. Define an Inner Function: Inside the outer function, define an inner function that will have access to the variables within the outer function's scope. This inner function forms a closure, which means it retains access to the variables and scope of its outer function even after the outer function has returned.

4. Return or Use the Inner Function: You can choose to either return the inner function from the outer function or use it within the outer function. By returning the inner function, you can assign it to a variable and invoke it later, maintaining access to the closure and the variables within it.

Here's an example that demonstrates the steps:

```
function outerFunction()
{
  const outerVariable = 'Hello, ';

  function innerFunction(name) {
    console.log(outerVariable + name);
  }

  return innerFunction;
}

const greet = outerFunction();
greet('John'); // Output: Hello, John!
greet('Alice'); // Output: Hello, Alice!
```

In this example, the outerFunction defines the outerVariable and the innerFunction.

The innerFunction has access to the outerVariable due to the closure formed by the function scope. The outerFunction returns the innerFunction, which is assigned to the greet variable. The greet variable can then be invoked with different names, and it will still have access to the outerVariable due to the closure.

Function closures are powerful in JavaScript as they allow inner functions to access and retain the scope of their outer functions, enabling encapsulation and data privacy.

Let's understand the closure by using an example.

#### **Example1**

```
<script>
    function fun()
    {
        var a = 4; // 'a' is the local variable, created by the fun()
        function innerfun() // the innerfun() is the inner function, or a closure
        {
            return a;
        }
        return innerfun;
    }
    var output = fun();
    document.write(output());
    document.write(" ");
    document.write(output());
</script>
// Output: 4 4
```

#### **✓ Asynchronous functions**

To use asynchronous functions in JavaScript, follow these steps:

1. Define an Asynchronous Function: Start by defining an asynchronous function using the `async` keyword before the function declaration. This indicates that the function will perform asynchronous operations and may use the `await` keyword.

For example:

```
async function fetchData()
{
    // Asynchronous operations
}
```

2. Use the `await` Keyword: Within the asynchronous function, use the ``await`` keyword to pause the execution of the function until a promise is resolved. The ``await`` keyword can only be used inside an asynchronous function.

For example:

```
async function fetchData() {  
  const response = await fetch('https://api.example.com/data');  
  const data = await response.json();  
  // Use the fetched data  
}
```

3. Handle Promises: Asynchronous functions typically involve working with promises. The `await` keyword is used to wait for a promise to resolve and return its resolved value. If the promise is rejected, an error will be thrown, which you can handle using `try/catch` blocks or by chaining .catch() to the awaited promise.

4. Invoke the Asynchronous Function: Call the asynchronous function like any regular function. It will return a promise that you can handle using `.then()` and `.catch()` or by using `async/await` syntax. For example:

```
fetchData()  
.then(data => {  
  // Handle the fetched data  
})  
.catch(error => {  
  // Handle errors  
});
```

Asynchronous functions and the `await` keyword provide a more readable and synchronous-like syntax for working with asynchronous operations in JavaScript. They allow you to write code that appears to be sequential and easier to understand, even though it is executing asynchronously.

Functions running in parallel with other functions are called asynchronous

**Example:**

```
<script>  
function myDisplayer(something) {  
  document.getElementById("demo").innerHTML = something;  
}  
function myCalculator(num1, num2, myCallback) {  
  let sum = num1 + num2;  
  myCallback(sum);  
}  
myCalculator(5, 5, myDisplayer);
```

```
</script>
```

**Output:** 10

In the example above, myDisplayer is the name of a function and it is passed to myCalculator() as an argument.

✓ **Promise functions**

To use Promise functions in JavaScript, follow these steps:

**1. Create a Promise:** Start by creating a new Promise using the Promise constructor.

The constructor takes a function as an argument, which has two parameters: **resolve** and **reject**. Inside this function, you perform your asynchronous operations and call **resolve** when the operation is successful, or `reject` when it encounters an error.

**For example:**

```
const fetchData = new Promise((resolve, reject) => {
  // Asynchronous operations
  if /* operation successful */
  {
    resolve(data); // Resolve with the fetched data
  } else {
    reject(error); // Reject with an error
  }
});
```

**2. Handle Promises:** Use `.then()` and `.catch()` methods to handle the resolved value or the rejected error of the Promise. The `.then()` method is used to handle the successful resolution of the Promise, while the `.catch()` method is used to handle any errors that occur during the Promise execution.

For example:

```
fetchData
  .then(data => {
    // Handle the resolved data
  })
  .catch(error => {
    // Handle the rejected error
  });
```

**3. Chaining Promises:** You can chain multiple Promise functions together using the `.then()` method. Each `.then()` method returns a new Promise, allowing you to perform further operations or make additional asynchronous calls.

For example:

```
fetchData
  .then(data => {
    // Perform operations with the resolved data
  })
```

```
return processedData;
})
then(processedData => {
// Perform further operations with the processed data
return finalResult;
})
then(finalResult => {
// Handle the final result
})
catch(error => {
// Handle any errors that occurred during the Promise chain
});
});
```

**4. Use Promise.all():** If you have multiple Promises and want to wait for all of them to resolve, you can use `Promise.all()`. This method takes an array of Promises as an argument and returns a new Promise that resolves with an array of the resolved values of the input Promises.

**For example:**

```
const promise1 = fetchData1();
const promise2 = fetchData2();
const promise3 = fetchData3();
Promise.all([promise1, promise2, promise3])
.then(([data1, data2, data3]) => {
// Handle the resolved data from all Promises
})
.catch(error => {
// Handle any errors that occurred during the Promise chain
});
```

By following these steps, you can use Promise functions in JavaScript to handle asynchronous operations in a more structured and manageable way. Promises provide a cleaner syntax and allow you to handle both successful resolutions and errors in a consistent manner.

Example using callback

```
<html>
<body>
<h1>JavaScript Functions</h1>
<h2>setInterval() with a Callback</h2>
<p>Wait 3 seconds (3000 milliseconds) for this page to change.</p>
<h1 id="demo"></h1>
<script>
setTimeout(function() { myFunction("I love You !!!"); }, 3000);
```

```
function myFunction(value) {  
document.getElementById("demo").innerHTML = value;  
}  
</script>  
</body>  
</html>
```

### Output

JavaScript Functions

setInterval() with a Callback

Wait 3 seconds (3000 milliseconds) for this page to change.

I love You !!!

#### ✓ **Async/await function**

To use async/await functions in JavaScript, follow these steps:

**1. Define an Async Function:** Start by defining an async function using the `async` keyword before the function declaration. This indicates that the function will perform asynchronous operations and may use the `await` keyword.

For example:

```
async function fetchData() {  
// Asynchronous operations  
}
```

**2. Use the await Keyword:** Within the `async` function, use the `await` keyword to pause the execution of the function until a promise is resolved. The `await` keyword can only be used inside an `async` function.

For example:

```
async function fetchData() {  
const response = await fetch('https://api.example.com/data');  
const data = await response.json();  
// Use the fetched data  
}
```

**3. Handle Errors:** To handle errors, you can use a `try/catch` block around the `await` statement or chain a `.catch()` method to the awaited promise. This allows you to catch and handle any errors that occur during the `async` operation.

For example:

```
async function fetchData() {  
try {  
const response = await fetch('https://api.example.com/data');  
const data = await response.json();  
// Use the fetched data  
} catch (error) {  
// Handle errors
```

```
}
```

```
}
```

**4. Invoke the Async Function:** Call the async function like any regular function. It will return a promise that you can handle using .then() and .catch() or by using async/await syntax.

For example:

```
fetchData()
  .then(data => {
    // Handle the fetched data
  })
  .catch(error => {
    // Handle errors
 });
```

Using async/await functions provides a more readable and synchronous-like syntax for working with asynchronous operations in JavaScript. It allows you to write code that appears to be sequential and easier to understand, even though it is executing asynchronously.

### Example

```
async function myFunction() {
  return "Hello";
```

Is the same as:

```
function myFunction() {
  return Promise.resolve("Hello");
```

Example:

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript async / await</h2>
<h1 id="demo"></h1>
<script>
async function myDisplay() {
let myPromise = new Promise(function(resolve, reject) {
resolve("I love You !!");
});
document.getElementById("demo").innerHTML = await myPromise;
}
myDisplay();
</script>
</body>
</html>
```

**Output**

JavaScript async / await

I love You!!

**Example2: Waiting for a Timeout**

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript async / await</h2>
<p>Wait 3 seconds (3000 milliseconds) for this page to change.</p>
<h1 id="demo"></h1>
<script>
async function myDisplay() {
let myPromise = new Promise(function(resolve) {
setTimeout(function() {resolve("I love You !!");}, 3000);
});
document.getElementById("demo").innerHTML = await myPromise;
}
myDisplay();
</script>
</body>
</html>
```

**Output**

JavaScript async / await

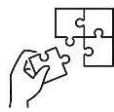
Wait 3 seconds (3000 milliseconds) for this page to change.

I love You !!



## Points to Remember

- Functions in JavaScript are blocks of reusable code that can be defined and invoked to perform specific tasks. A function may be a user defined function or a built-in function.
- You can define function to perform specific tasks. The defined function accepts parameters, returns values, and even passes functions as arguments to other functions.



## Application of learning 2.4.

You are tasked to develop an e-commerce website with a shopping cart feature using JavaScript. Functions are used to handle various aspects of the shopping cart, such as adding items, calculating the total, and processing the checkout.



## Indicative content 2.5: Using objects in JavaScript



Duration: 8 hrs



### Theoretical Activity 2.5.1: Description of objects in JavaScript



#### Tasks:

- 1: In small groups, you are requested to answer the following questions related to the object in JavaScript:
  - i. What do you understand about the term “object”.
  - ii. Give the ways of creating an object.
  - iii. Explain the ways of accessing object
- 2: Provide the answer for the asked questions and write them on flipchart/ papers.
- 3: Present the findings/answers to the trainer and the whole class.
- 4: Discuss on provided answer and choose the best answers.
- 5: For more clarification, read the key readings 2.5.1 and ask questions where necessary.



#### Key readings 2.5.1.:

##### Definition on an object:

An object is a fundamental data type that allows you to represent and organize complex data structures. It is a collection of key-value pairs, where each key (also known as a property) is a unique identifier, and each value can be of any data type (such as a string, number, boolean, function, or even another object).

Objects in JavaScript are dynamic, meaning you can add, modify, or delete properties and methods at any time. They are commonly used to represent entities, such as a person, car, or any other real-world or abstract concept that has properties and behaviors.

An object is a collection of properties, and a property is an association between a name (or key) and a value.

A JavaScript object is an entity having state and behaviour (properties and method).

For example: car, pen, bike, chair, glass, keyboard, monitor etc.

JavaScript is an object-based language. Everything is an object in JavaScript.

In JavaScript, objects are king. If you understand objects, you understand JavaScript.

##### Creating Objects in JavaScript

##### There are 3 ways to create objects.

1. By object literal
2. By creating instance of Object directly (using new keyword)
3. By using an object constructor (using new keyword)

### **1. By object literal:**

Using curly braces `{}` and defining key-value pairs within them, you can create objects directly with known properties and methods. This syntax is concise and commonly used for static objects with predefined properties.

#### **Object Literal Syntax:**

```
const objectName = {  
    property1: value1,  
    property2: value2,  
};
```

In this syntax:

- ObjectName is the name of the object variable.
- property1, property2, etc. are the names of the object properties.
- value1, value2, etc. are the values assigned to the corresponding properties.

### **2. By creating instance of Object directly (using new keyword)**

Creating an instance of an object directly using the `new` keyword in JavaScript involves using a constructor function or a built-in object constructor to create a new object with its own set of properties and behaviors.

#### **Here's an example of creating an instance of an object using the `new` keyword:**

```
// Constructor function  
function Person(name, age)  
{  
    this.name = name;  
    this.age = age;  
}  
  
// Creating an instance using new keyword  
const person1 = new Person('John', 25);
```

In this example, we define a constructor function called `Person` that takes `name` and `age` as parameters. Inside the constructor function, we assign the values of `name` and `age` to the newly created object using the `this` keyword.

Using an object constructor with the `new` keyword in JavaScript involves creating instances of objects based on a predefined constructor function. An object constructor is a function that serves as a blueprint for creating multiple objects with similar properties and behaviors.

### **3. By using an object constructor (using new keyword)**

A constructor is a function that initializes an object.

**Object constructor:** In JavaScript, there is a special constructor function known as `Object()` is used to create and initialize an object.

The return value of the `Object()` constructor is assigned to a variable. The variable contains a reference to the new object.

We need an object constructor to create an object “type” that can be used multiple times without redefining the object every time.

**There are two ways to instantiate object constructor,**

1. var object\_name = new Object();

OR

var object\_name = new Object("java", "JavaScript", "C#");

2. var object\_name = { };

In 1st method, the object is created by using new keyword like normal OOP languages, and “Java”, “JavaScript”, “C#” are the arguments, that are passed when the constructor is invoked.

In 2nd method, the object is created by using curly braces “{ }”.

To create an object, use the new keyword with Object() constructor, like this:

```
var mango = new Object ();
mango.color = "yellow";
mango.shape= "round";
mango.sweetness = 8;
mango.howSweetAmI = function ()
{
    console.log("Hmm Hmm Good");
}
```

## JavaScript Properties

Properties are the characteristics or attributes of an object. They define the state or data associated with an object. Properties can hold values of various data types, such as strings, numbers, booleans, arrays, functions, or even other objects. properties are the values associated with a JavaScript object.

Properties can usually be changed, added, and deleted, but some are read only.

### Accessing JavaScript properties

Properties in JavaScript can be defined or accessed two ways:

using dot notation ('object.property') or bracket notation ('object['property']').

**The syntax is:**

objectName.propertyName

OR

objectName["propertyName"]

**Note:** Both dot notation and bracket notation provide ways to access object methods and properties. Dot notation is commonly used when you know the property or method name in advance, while bracket notation is useful when the name is dynamic or contains special characters.

**Here's an example:**

```
const person = {
  name: 'John',
  age: 25,
  'favorite color': 'blue'
};

console.log(person.name); // Output: John
console.log(person.age); // Output: 25
console.log(person['favorite_color']); // Output: blue
```

In this example, `person` is an object with properties `name`, `age`, and `favorite\_color`.

The dot notation (`person.name`, `person.age`) and bracket notation (`person['favorite\_color']`) are used to access the values of these properties.

Properties in JavaScript can be added, modified, or removed dynamically.

**For example:**

```
person.job = 'Engineer'; // Adding a new property
person.age = 26; // Modifying an existing property
delete person['favorite color']; // Removing a property
```

In this example, a new property `job` is added to the `person` object, the value of the `age` property is modified, and the `favorite color` property is removed using the `delete` keyword.

### JavaScript Methods

JavaScript methods are actions that can be performed on objects.

A JavaScript **method** is a property containing a **function definition**.

Methods are functions that are associated with objects. They define the behaviors or actions that an object can perform. Methods are defined as properties of an object, where the value of the property is a function.

Methods can be called or invoked using dot notation (object.method()) or bracket notation (`object['method']()`). When a method is invoked, it can access and operate on the properties and other methods of the object.

### Here's an example of defining and using methods in JavaScript:

```
const person = {
  name: 'John',
  age: 25,
  greet: function() {
    console.log('Hello, my name is ' + this.name);
  },
  celebrateBirthday: function() {
    this.age++;
  }
};
```

```
        console.log('Happy birthday! Now I am ' + this.age + ' years old.');
    }
};

person.greet(); // Output: Hello, my name is John
person.celebrateBirthday(); // Output: Happy birthday! Now I am 26 years old.
```

In this example, the **person** object has two methods: **greet** and **celebrateBirthday**.

The **greet** method logs a greeting message using the **name** property of the object and the **celebrateBirthday** method increments the **age** property by one and logs a birthday message.

## Object sets

An object set is a built-in data structure that allows you to store unique values. It is similar to an array, but with the distinction that it only stores unique values, eliminating duplicates.

Each value can only occur once in a Set.

A Set can hold any value of any data type.

### How to Create a Set

You can create a JavaScript Set by:

- Passing an Array to new Set()
- Create a new Set and use add() to add values
- Create a new Set and use add() to add variables

Different Object Set methods are offered by JavaScript such as **add()**, **delete()**, **clear()**, and **has()**.

The “**add()**” object Set method is used for appending values to the set object, **delete()** and **clear()** object set methods for deleting a specific or all elements at once, and lastly, the “**has()**” method is utilized for searching any element in the created Set.

This write-up will discuss the object Set methods in JavaScript. Moreover, we will also demonstrate examples related to each object Set methods such as **add()**, **delete()**, **clear()**, and **has()**. So, let's start!

### Here's an example of using an object set in JavaScript:

```
const mySet = new Set();
mySet.add(1);
mySet.add(2);
mySet.add(3);
mySet.add(1); // Ignored, as it's already present in the set
console.log(mySet.size); // Output: 3
mySet.delete(2);
console.log(mySet.has(2)); // Output: false
```

In this example, we create a new Set object called mySet and We use the add method to add values 1, 2, and 3 to the set.

Since sets only store unique values, the duplicate value 1 is ignored. We can check the size of the set using the size property.

The delete method is used to remove a value from the set. In this case, we remove the value 2. The **has** method is then used to check if a value exists in the set. In this case, we check if 2 is present, which returns false since we removed it.

### Object maps

An object map (also known as a map or dictionary) is a built-in data structure that allows you to store key-value pairs. It is similar to an object, but with some key differences. Object maps provide a more flexible and powerful way to store and retrieve data based on keys.

A Map holds key-value pairs where the keys can be any datatype.

A Map remembers the original insertion order of the keys.

A Map has a property that represents the size of the map.

### How to Create a Map

You can create a JavaScript Map by:

- Passing an Array to new Map()
- Create a Map and use Map.set()

Here's an example of using an object map in JavaScript:

```
const myMap = new Map();
myMap.set('name', 'John');
myMap.set('age', 25);
myMap.set('city', 'New York');
console.log(myMap.get('name')); // Output: John
console.log(myMap.has('city')); // Output: true
myMap.delete('age');
console.log(myMap.size); // Output: 2
```

In this example, we create a new Map object called myMap and we use the set method to add key-value pairs to the map. Each key is associated with a value. We can retrieve values from the map using the get method, providing the corresponding key.

The **has** method is used to check if a key exists in the map. In this case, we check if the key '**city**' is present, which returns **true**. The **delete** method is used to remove a key-value pair from the map. In this case, we remove the key 'age'.

The **size** property returns the number of key-value pairs in the map



## Practical Activity 2.5.2: Using objects in JavaScript



### Task:

- 1: Referring to the previous theoretical activities (2.5.1) you are requested to go to the computer lab to use objects in javascript programming. This task should be done individually.
- 2: Read the key reading 2.5.2 in trainee manual about use of objects in JavaScript program.
- 3: Referring to the key reading 2.5.2, use objects in JavaScript program.
- 4: Present your work to the trainer and whole class
- 5: Ask questions for more clarification where necessary.



### Key readings 2.5.2

#### To use an object in JavaScript programming, you can follow these steps:

1. Object Definition: Define the object by either using object literal syntax or creating an object constructor function. Decide on the properties and methods that the object should have.
2. Object Creation: Create an instance of the object by either directly assigning values to the object properties using object literal syntax or using the `new` keyword with the object constructor function.
3. Accessing Properties and Methods: Use dot notation or bracket notation to access and manipulate the object's properties and invoke its methods. Dot notation is typically used when you know the property or method name in advance, while bracket notation is useful when the name is dynamic or contains special characters.
4. Modifying Object Properties: Update the values of object properties by assigning new values to them using assignment (`=`) operator.
5. Adding and Deleting Properties: Add new properties to the object by assigning values to them using dot notation or bracket notation. Delete properties from the object using the `delete` keyword followed by the property name.
6. Object Iteration: Iterate over the object's properties using loops or built-in methods like `for...in` loop or `Object.keys()`.
7. Object Comparison: Compare objects using strict equality (`==`) or by comparing their individual properties.

These steps provide a general guideline for working with objects in JavaScript. The specific implementation and usage may vary depending on your application's requirements and the specific object-oriented design patterns you are following.

**There are 3 ways to create objects.**

### **1) JavaScript Object by object literal**

To create an object using object literal syntax in JavaScript, you can follow these steps:

1. Start with an empty object literal: Begin by declaring an empty object using curly braces `{}`.
2. Define the properties: Inside the object literal, define the properties you want the object to have. Each property consists of a key-value pair, separated by a colon `:`. The key represents the property name, and the value represents the initial value of the property.
3. Assign values to properties: Assign values to the properties by specifying the value after the colon `:`. You can use literals, variables, or expressions as property values.
4. Add methods (optional): If you want the object to have methods, you can define them as functions within the object literal. Methods can be assigned to properties just like any other value.
5. End with a closing curly brace: Close the object literal with a closing curly brace `}` to indicate the end of the object definition.

Here's an example that demonstrates the steps:

```
const person = {  
    name: 'John',  
    age: 25,  
    city: 'New York',  
    greet: function() {  
        console.log('Hello, my name is ' + this.name);  
    }  
};
```

In this example, we create an object called `person` using object literal syntax. It has properties like `name`, `age`, and `city`, with corresponding values assigned to them. The `greet` property is a method defined as a function.

You can access and use the object properties and methods like this:

```
console.log(person.name); // Output: John  
person.greet(); // Output: Hello, my name is John
```

By following these steps, you can create objects using object literal syntax in JavaScript and define their properties and methods to store and manipulate data in your code.

**The syntax of creating object using object literal is given below:**

```
object={property1:value1,property2:value2.....propertyN:valueN}
```

As you can see, property and value is separated by : (colon).

Let's see a simple example of creating an object in JavaScript.

```
<script>
emp={id:102,name:"Shyam Kumar",salary:40000}
document.write(emp.id+" "+emp.name+" "+emp.salary);
</script>
```

Output of the above example:

```
102 Shyam Kumar 40000
```

## 2) By creating instance of Object

To create an object by creating an instance of the `Object` constructor in JavaScript, you can follow these steps:

1. Create a variable: Start by declaring a variable to store the object instance.
2. Create an instance: Use the `new` keyword followed by the `Object` constructor to create a new instance of the object.
3. Define properties: Assign values to properties of the object using dot notation or bracket notation.
4. Add methods: Define methods by assigning functions to properties of the object.

Here's an example that demonstrates the steps:

```
// Step 1: Create a variable
let person;
// Step 2: Create an instance
person = new Object();
// Step 3: Define properties
person.name = 'John';
person.age = 25;
person.city = 'New York';
// Step 4: Add methods
person.greet = function() {
  console.log('Hello, my name is ' + this.name);
};
```

In this example, we create an object called `person` by creating an instance of the `Object` constructor. Then we define properties such as `name`, `age`, and `city`, and assign values to them using dot notation. We also add a method called `greet` to the object.

You can access and use the object properties and methods like this:

```
console.log(person.name); // Output: John
person.greet(); // Output: Hello, my name is John
```

By following these steps, you can create objects by creating instances of the `Object` constructor in JavaScript and define their properties and methods to store and manipulate data in your code.

**The syntax of creating object directly is given below:**

```
var objectname=new Object();
```

Here, **new keyword** is used to create an object.

Let's see the example of creating object directly.

```
<script>
    var emp=new Object();
    emp.id=101;
    emp.name="Ravi Malik";
    emp.salary=50000;
    document.write(emp.id+" "+emp.name+" "+emp.salary);
</script>
```

Output of the above example

```
101 Ravi 50000
```

### **3) By using an Object constructor**

**To create an object by using an object constructor in JavaScript, you can follow these steps:**

1. Define the object constructor function: Create a function that will serve as the object constructor. This function will be used to create new instances of the object.
2. Define properties: Inside the constructor function, use the `this` keyword to define properties of the object. Assign initial values to these properties using the function's parameters.
3. Add methods: Define methods by adding them as functions to the prototype of the constructor function. This allows all instances of the object to share the same method implementation.
4. Create an instance: Use the `new` keyword followed by the constructor function to create a new instance of the object. Assign it to a variable.

Here's an example that demonstrates the steps:

```
// Step 1: Define the object constructor function
```

```
function Person(name, age, city) {
    this.name = name;
    this.age = age;
    this.city = city;
}
```

```
// Step 2: Define properties
```

```
/ Step 3: Add methods
```

```
Person.prototype.greet = function() {
```

```
        console.log('Hello, my name is ' + this.name);
    };
    // Step 4: Create an instance
    const person = new Person('John', 25, 'New York');
```

In this example, we define an object constructor function called `Person`. Inside the constructor function, we use the `this` keyword to define properties like `name`, `age`, and `city`. We also add a method called `greet` to the prototype of the constructor function.

To create an instance of the object, we use the `new` keyword followed by the constructor function `Person`. The instance is assigned to the variable `person`.

You can access and use the object properties and methods like this:

```
console.log(person.name); // Output: John
person.greet(); // Output: Hello, my name is John
```

By following these steps, you can create objects by using an object constructor in JavaScript and define their properties and methods to store and manipulate data in your code.

Here, you need to create a function with arguments. Each argument value can be assigned in the current object by using this keyword.

The **keyword** refers to the current object.

The example of creating an object by object constructor is given below.

```
<script>
    function emp(id,name,salary)
    {
        this.id=id;
        this.name=name;
        this.salary=salary;
    }
    e=new emp(103,"Vimal Jaiswal",30000);
    document.write(e.id+" "+e.name+" "+e.salary);
</script>
```

#### **Output of the above example**

103 Vimal Jaiswal 30000

#### **✓ Accessing object method and properties**

##### **Accessing JavaScript Properties**

To access JavaScript properties, you can follow these steps:

1. Identify the object: Determine the object that contains the property you want to access. It can be a predefined object, an object created using object literal syntax, or an object created using an object constructor.
2. Use dot notation: If the property name is known and does not contain special characters, you can access the property using dot notation. Write the object name, followed by a dot (`.`), and then the property name.
3. Use bracket notation: If the property name is dynamic, contains special characters, or is stored in a variable, you can use bracket notation. Write the object name, followed by square brackets (`[]`), and inside the brackets, provide the property name as a string or a variable.
4. Access nested properties: If the property you want to access is nested within another object or objects, you can chain multiple dot or bracket notation accessors to reach the desired property.

**Here's an example that demonstrates these steps:**

```
const person = {  
    name: 'John',  
    age: 25,  
    address: {  
        street: '123 Main St',  
        city: 'New York',  
        country: 'USA'  
    }  
};  
  
console.log(person.name); // Output: John  
console.log(person['age']); // Output: 25  
const propertyName = 'address';  
console.log(person[propertyName]['city']); // Output: New York
```

In this example, we have an object called `person` with properties like name, age, and address. We access these properties using dot notation (`person.name`, `person.age`) and bracket notation (`person['address']`).

We can also access nested properties using chained accessors (`person[propertyName]['city']`).

By following these steps, you can access JavaScript properties and retrieve their values, allowing you to work with the data stored within objects.

**The syntax for accessing the property of an object is:**

*objectName.property* // person.age  
or  
*objectName["property"]* // person["age"]  
or

```
objectName[expression] // x = "age"; person[x]
```

The expression must evaluate to a property name.

**Example1:**

```
<script>
const person = {
  firstname: "John",
  lastname: "Doe",
  age: 50,
  eyecolor: "blue"
};
document.getElementById("demo").innerHTML = person.firstname + " is " + person.age
+ " years old.";
</script>
```

**Output**

John is 50 years old.

**Example2:**

```
<script>
const person = {
firstname: "John",
lastname: "Doe",
age: 50,
eyecolor: "blue"
};
document.getElementById("demo").innerHTML = person["firstname"] + " is " +
person["age"] + " years old.";
</script>
```

**Output**

John is 50 years old.

**JavaScript Object Methods**

**Example:**

```
<script>
// Create an object:
const person = {
firstName: "John",
lastName: "Doe",
id: 5566,
fullName : function() {
return this.firstName + " " + this.lastName;
}};
// Display data from the object:
```

```
document.getElementById("demo").innerHTML = person.fullName();
</script>
```

## Output

John Doe

### Accessing Object Methods

To access object methods in JavaScript, you can follow these steps:

1. Identify the object: Determine the object that contains the method you want to access. It can be a predefined object, an object created using object literal syntax, or an object created using an object constructor.
2. Use dot notation: If the method name is known, you can access and invoke the method using dot notation. Write the object name, followed by a dot (`.`), and then the method name, followed by parentheses `()`.
3. Invoke the method: To execute the method and perform its associated actions or computations, add parentheses `()` after the method name.

**Here's an example that demonstrates these steps:**

```
const calculator = {
  operand1: 5,
  operand2: 3,
  add: function() {
    return this.operand1 + this.operand2;
  },
  subtract: function() {
    return this.operand1 - this.operand2;
  }
};
console.log(calculator.add()); // Output: 8
console.log(calculator.subtract()); // Output: 2
```

In this example, we have an object called calculator with properties like operand1 and operand2, as well as methods like add and subtract. We access and invoke these methods using dot notation (calculator.add(), calculator.subtract()).

By following these steps, you can access and invoke object methods in JavaScript, allowing you to perform actions or computations related to the object's properties and behavior.

**You access an object method with the following syntax:**

***objectName.methodName()***

You will typically describe fullName() as a method of the person object, and fullName as a property.

The fullName property will execute (as a function) when it is invoked with () .

This example accesses the fullName() method of a person object:

**Example:**

```
<script>
const person = {
  firstName: "John",
  lastName: "Doe",
  id: 5566,
  fullName: function() {
    return this.firstName + " " + this.lastName;
  }
};
document.getElementById("demo").innerHTML = person.fullName();
</script>
```

**Output**

John Doe

**Note:** If you access the `fullName` property, without `()`, it will return the function definition:

**Example:**

```
<script>
const person = {
  firstName: "John",
  lastName: "Doe",
  id: 5566,
  fullName: function() {
    return this.firstName + " " + this.lastName;
  }
};
document.getElementById("demo").innerHTML = person.fullName;
</script>
```

**Output**

`function() { return this.firstName + " " + this.lastName; }`

**✓ Object constructors****Example:**

```
<script>
// Constructor function for Person objects
function Person(first, last, age, eye) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eye;
}
// Create a Person object
const myFather = new Person("John", "Doe", 50, "blue");
```

```
// Display age
document.getElementById("demo").innerHTML =
"My father is " + myFather.age + ".";
</script>
```

**Output**

My father is 50.

✓ **Object sets**

**The new Set() Method**

Pass an Array to the new Set() constructor:

**Example:**

```
<script>
// Create a Set
const letters = new Set(["a","b","c"]);
// Display set.size
document.getElementById("demo").innerHTML = letters.size;
</script>
```

**Output:**

3

Create a Set and add literal values:

```
<script>
// Create a Set
const letters = new Set();
// Add Values to the Set
letters.add("a");
letters.add("b");
letters.add("c");
// Display set.size
document.getElementById("demo").innerHTML = letters.size;
</script>
```

**Output:**

3

**Create a Set and add variables:**

```
<script>
// Create a Set
const letters = new Set();
// Create Variables
const a = "a";
```

```
const b = "b";
const c = "c";
// Add the Variables to the Set
letters.add(a);
letters.add(b);
letters.add(c);
// Display set.size
document.getElementById("demo").innerHTML = letters.size;
</script>
```

**Output:** 3

**✓ Object maps**

- **new Map()**

You can create a Map by passing an Array to the new Map() constructor:

**Example:**

```
<script>
// Create a Map
const fruits = new Map([
["apples", 500],
["bananas", 300],
["oranges", 200]
]);
document.getElementById("demo").innerHTML = fruits.get("apples");
</script>
```

**Output:**

500

- **Map.set()**

You can add elements to a Map with the set() method:

**Example:**

```
<script>
// Create a Map
const fruits = new Map();
// Set Map Values
fruits.set("apples", 500);
fruits.set("bananas", 300);
fruits.set("oranges", 200);
document.getElementById("demo").innerHTML = fruits.get("apples");
</script>
```

```
</script>
```

**Output: 500**

The set() method can also be used to change existing Map values:

Example:

```
<script>
// Create a Map
const fruits = new Map([
  ["apples", 500],
  ["bananas", 300],
  ["oranges", 200]
]);
fruits.set("apples", 200);
document.getElementById("demo").innerHTML = fruits.get("apples");
</script>
```

**Output:**

200

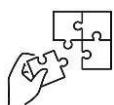


### Points to Remember

- An object is a fundamental data type that allows you to represent and organize complex data structures. It is a collection of key-value pairs, where each key (also known as a property) is a unique identifier, and each value can be of any data type.

There are three (3) ways of creating an object in JavaScript and these are: By object literal, by creating instance of Object directly and by using an object constructor.

- Properties and methods in JavaScript can be defined or accessed in two ways: using dot notation (`object.property`) or bracket notation (`object['property']`).



### Application of learning 2.5.

You are tasked to develop a Library Management System using JavaScript, where objects are employed to represent books, library patrons, and transactions. This system allows librarians to manage the library's inventory, track book borrowings, and maintain patron information.



## Indicative content 2.6: Using arrays in JavaScript



Duration: 6 hrs



### Theoretical Activity 2.6.1: Description of arrays in JavaScript



#### Tasks:

- 1: In small groups, you are requested to answer the following questions related to the arrays in JavaScript:
  - i.Explain the term “array”.
  - ii. Describe types of arrays.
  - iii. Explain how to access array elements.
- 2: Provide the answer for the asked questions and write them on papers.
- 3: Present the findings/answers to the whole class and choose the correct answers.
- 4: For more clarification, read the key readings 2.6.1 and ask questions where necessary.



#### Key readings 2.6.1.:

##### Definition of array

A **JavaScript array** is a special type of object that allows you to store multiple values in a single variable. It is an object that represents a collection of similar type of elements.

Each element in an array is identified by an index. Arrays provide a versatile and powerful way to organize and access multiple values within a single variable.

Arrays in JavaScript are created using square brackets ('[]') and can be assigned to a variable. Elements within an array are separated by commas.

##### Here's an example of creating a JavaScript array:

```
const fruits = ['apple', 'banana', 'orange'];
```

In this example, we have an array called **fruits** that contains three elements: 'apple', 'banana', and 'orange'. The elements are ordered, and their positions within the array can be accessed using zero-based indexing.

Arrays in JavaScript are dynamic, meaning their size can change dynamically by adding or removing elements. You can access individual elements within an array using their index, such as `fruits[0]` to access the first element ('apple').

##### There are 3 ways to construct array in JavaScript

1. By array literal
2. By creating instance of Array directly (using new keyword)

### 3. By using an Array constructor (using new keyword)

#### JavaScript array literal

The syntax of creating array using array literal is given below:

```
var arrayname=[value1,value2.....valueN];
```

As you can see, values are contained inside [ ] and separated by , (comma).

The most common way to create an array is by using array literal syntax, which involves enclosing the elements within square brackets `[]` and separating them with commas.

For example:

```
const array = [element1, element2, element3];
```

Here, element1, element2, and element3 represent the values or elements you want to store in the array. These elements can be of any data type, such as numbers, strings, booleans, objects, or even other arrays.

#### JavaScript Array directly (new keyword)

The syntax of creating array directly is given below:

```
var arrayname=new Array();
```

Here, **new keyword** is used to create instance of array.

The correct syntax for creating an array using the `new` keyword and `Array` constructor is as follows:

```
const array_name = new Array();
```

This syntax creates an empty array. You can then add elements to the array using various methods like `push()`, `unshift()`, or direct assignment.

**Example:**

```
const array = new Array();
array.push('apple', 'banana', 'orange');
```

In this example, we create an empty array using the new keyword and Array constructor. Then, we add elements to the array using the `push()` method.

#### JavaScript array constructor (new keyword)

Here, you need to create an instance of an array by passing arguments in the constructor so that we don't have to provide value explicitly.

**Array Constructor:** Another way to create an array is by using the `Array` constructor. You can pass the elements as arguments to the constructor. For example:

```
const array = new Array(element1, element2, element3);
```

This syntax is less commonly used compared to the array literal syntax.

It's important to note that arrays in JavaScript are dynamic, meaning their size can change dynamically by adding or removing elements. Additionally, arrays in JavaScript can store elements of different data types, making them versatile for storing and manipulating collections of data.

Remember to replace `element1`, `element2`, and `element3` with the actual values or variables you want to store in the array.

## 2. Types of Javascript arrays:

Javascript arrays are divided into two types:

1. One-dimensional arrays
2. Multi-dimensional arrays

### One -Dimensional Arrays

A one-dimensional array is a kind of linear array. It involves single sub-scripting. The [] (brackets) is used for the subscript of the array and to declare and access the elements from the array.

In JavaScript, a one-dimensional array is the most basic and commonly used type of array. It is a linear collection of elements stored in a single row or sequence. Each element in the array is assigned a numeric index, starting from 0, to access or manipulate its value.

**Here's an example of a one-dimensional array in JavaScript:**

```
const numbers = [1, 2, 3, 4, 5];"
```

In this example, the numbers array is a one-dimensional array that stores a sequence of numbers. Each element in the array is accessed using its index. For instance, numbers[0] refers to the first element, which is 1, numbers[1] refers to the second element, which is 2, and so on.

You can perform various operations on one-dimensional arrays, such as accessing elements, modifying elements, adding or removing elements, iterating over elements, and applying array methods like push(), pop(), slice(), forEach(), and more.

One-dimensional arrays are versatile and commonly used for organizing and manipulating collections of data in JavaScript. They provide a simple and efficient way to store and access elements in a linear manner.

Syntax for creating a one dimensional array is as follows:

DataType ArrayName [size];

For example: int a[10];

## Multi-Dimensional Arrays

In JavaScript, multi-dimensional arrays are arrays that contain other arrays as elements. This allows you to create arrays with multiple levels or dimensions, forming a grid-like structure. Multi-dimensional arrays are useful for representing matrices, tables, or nested data structures. Here's an example of a two-dimensional array in JavaScript:

```
const matrix = [ [1, 2, 3], [4, 5, 6], [7, 8, 9] ];
```

In this example, `matrix` is a two-dimensional array that represents a 3x3 matrix. Each element in the array is itself an array. You can access individual elements in the multi-dimensional array using multiple indices. For example, `matrix[0][0]` refers to the first element in the first row, which is 1, `matrix[1][2]` refers to the third element in the second row, which is 6, and so on.

Multi-dimensional arrays can have more than two dimensions as well. For example, a three-dimensional array can be used to represent a cube or a collection of matrices.

You can perform various operations on multi-dimensional arrays, such as accessing elements, modifying elements, iterating over elements using nested loops, and applying array methods like `push()`, `pop()`, `slice()`, and more.

Multi-dimensional arrays provide a way to organize and manipulate structured data in JavaScript, allowing you to work with complex data structures and solve problems that require multiple dimensions.

**In multi-dimensional arrays, we have to discuss on two categories:**

- Two-Dimensional Arrays
- Three-Dimensional Arrays

### Two-Dimensional Arrays

In JavaScript, a two-dimensional array is a type of multi-dimensional array that represents a grid-like structure with rows and columns. It is an array of arrays, where each inner array represents a row in the grid and contains elements as its columns.

An array involving two subscripts [] [] is known as a two-dimensional array. They are also known as the array of the array. Two-dimensional arrays are divided into rows and columns and are able to handle the data of the table.

**Syntax:** `DataType ArrayName[row_size] [column_size];`

Here's an example of a two-dimensional array in JavaScript:

```
const grid = [  
    [1, 2, 3],
```

```
[4, 5, 6],  
[7, 8, 9]  
];
```

In this example, grid is a two-dimensional array that represents a 3x3 grid. Each inner array represents a row, and the elements within the inner arrays represent the columns. You can access individual elements in the two-dimensional array using two indices.

For example, grid[0][0] refers to the element in the first row and first column, which is 1, grid[1][2] refers to the element in the second row and third column, which is 6, and so on.

Two-dimensional arrays are commonly used for representing matrices, game boards, tables, or any other data structure that can be visualized as a grid. You can perform various operations on two-dimensional arrays, such as accessing elements, modifying elements, iterating over elements using nested loops, and applying array methods like push(), pop(), slice(), and more.

Two-dimensional arrays provide a powerful tool for working with structured data in JavaScript, enabling you to solve problems that involve grid-like structures or tabular data.

**For Example:** int arr[5][5];

### Three-Dimensional Arrays

In JavaScript, a three-dimensional array is a type of multi-dimensional array that extends beyond the concept of rows and columns to include a third dimension.

It is an array of arrays of arrays, where each innermost array represents a single element, each middle array represents a row, and the outermost array represents a collection of rows.

**Syntax:** DataType ArrayName[size1][size2][size3];

Here's an example of a three-dimensional array in JavaScript:

```
const cube = [  
  [  
    [1, 2, 3],  
    [4, 5, 6]  
  ],  
  [  
    [7, 8, 9],  
    [10, 11, 12]  
  ]  
];
```

In this example, cube is a three-dimensional array that represents a cube-like structure. The outermost array represents the layers, the middle arrays represent the rows within each layer, and the innermost arrays represent the columns within each row. You can access individual elements in the three-dimensional array using three indices.

For example, `cube[0][1][2]` refers to the element in the first layer, second row, and third column, which is 6, `cube[1][0][1]` refers to the element in the second layer, first row, and second column, which is 8, and so on.

When we require to create two or more tables of the elements to declare the array elements, then in such a situation we use three-dimensional arrays.

**For Example:** int a[5][5][5];

#### Storing values in a one-dimensional array

To store values in a one-dimensional JavaScript array, you can use the array literal syntax or the `push()` method. Here are examples of how you can store values in a one-dimensional array:

#### Using Array Literal Syntax:

```
const numbers = [1, 2, 3, 4, 5];
```

In this example, the `numbers` array is created using the array literal syntax, and it stores five numeric values.

#### Using Push Method:

```
const fruits = [];
fruits.push('apple');
fruits.push('banana');
fruits.push('orange');
```

In this example, an empty array `fruits` is created. Then, the `push()` method is used to add values to the array. Each `'push()'` call appends the given value to the end of the array.

You can also directly assign values to specific indices of the array:

```
const colors = [];
colors[0] = 'red';
colors[1] = 'green';
colors[2] = 'blue';
```

In this example, an empty array `colors` is created, and values are assigned to specific indices using direct assignment.

Regardless of the method used, the values are stored in the array, and you can access them using their respective indices. For example, `'numbers[0]'` would return 1, `fruits[1]` would return `'banana'`, and `colors[2]` would return `'blue'`.

In JavaScript, once you have declared an array, it can be filled with values of any data type. The values are stored in a Js array using the assignment statement or input statement. In JavaScript, you can use the “**prompt**” dialog box to input data into the elements of the array during the execution of the script.

You can also assign value to the individual elements of the array using an assignment statement. For example, to assign values to an array “temp” having 4 elements, the following assignment statements are used:

```
Temp[0] = 15;
```

```
Temp[1] = 20;
```

```
Temp[2] = 28;
```

```
Temp[3] = 30;
```

### JavaScript Array Methods

JavaScript provides a variety of built-in array methods that allow you to manipulate and perform operations on arrays.

Here are some commonly used array methods in JavaScript:

Method	Description
<b>concat()</b>	joins two or more arrays and returns a result
<b>indexOf()</b>	searches an element of an array and returns its position
<b>find()</b>	returns the first value of an array element that passes a test
<b>findIndex()</b>	returns the first index of an array element that passes a test
<b>forEach()</b>	calls a function for each element
<b>includes()</b>	checks if an array contains a specified element
<b>push()</b>	adds a new element to the end of an array and returns the new length of an array
<b>unshift()</b>	adds a new element to the beginning of an array and returns the new length of an array
<b>pop()</b>	removes the last element of an array and returns the removed element
<b>shift()</b>	removes the first element of an array and returns the removed element
<b>sort()</b>	sorts the elements alphabetically in strings and in ascending order
<b>slice()</b>	selects the part of an array and returns the new array
<b>splice()</b>	removes or replaces existing elements and/or adds new elements
<b>join()</b>	Joins all elements of an array into a string, using a specified separator.
<b>lastIndexOf()</b>	Returns the last index at which a specified element is found in an array, or -1 if not found.

<b>map()</b>	Creates a new array with the results of calling a provided function on every element in the array.
<b>sort()</b>	Sorts the elements of an array in place and returns the sorted array
<b>reverse()</b>	Reverses the order of the elements in an array in place.
<b>filter()</b>	Creates a new array with all elements that pass a provided test implemented by a callback function

These are just a few examples of the many array methods available in JavaScript. Each method serves a specific purpose and can be used to manipulate, transform, or extract information from arrays in different ways.

### JavaScript Array Iteration

In JavaScript, there are several ways to iterate over the elements of an array. Here are some common methods for array iteration:

**1. For Loop:** You can use a traditional for loop to iterate over the array by accessing elements using their indices.

Here's an example:

```
const numbers = [1, 2, 3, 4, 5];
for (let i = 0; i < numbers.length; i++)
{
  console.log(numbers[i]);
}
```

**2. forEach():** The forEach() method allows you to iterate over each element of an array and perform a callback function on each element.

Here's an example:

```
const numbers = [1, 2, 3, 4, 5];
numbers.forEach(function(element)
{
  console.log(element);
});
```

**3. for of Loop:** The `for...of` loop is a more concise way to iterate over the elements of an array. It provides direct access to each element without using indices. **Here's an example:**

```
const numbers = [1, 2, 3, 4, 5];
for (const element of numbers) {
  console.log(element);
}
```

**4. map():** The map() method creates a new array by applying a callback function to each element of the original array. It returns a new array with the results of the callback function.

**Here's an example:**

```
const numbers = [1, 2, 3, 4, 5];
const squaredNumbers = numbers.map(function(element) {
  return element * element;
});
console.log(squaredNumbers);
```

These are some of the common methods for iterating over the elements of an array in JavaScript. Each method has its own benefits and use cases, so choose the one that best suits your needs.

Array iteration methods operate on every array item.



### Practical Activity 2.6.2: Apply arrays in JavaScript



#### Task:

- 1: Referring to the previous theoretical activities (2.6.1) you are requested to go to the computer lab to apply arrays as used in JavaScript programming. This task should be done individually.
- 2: Read the key reading 2.6.2 in trainee manual about application of arrays in JavaScript program.
- 3: Referring to the the key reading 2.6.2, apply arrays in JavaScript program.
- 4: Ask questions where necessary for more clarification.



### Key readings 2.6.2

**To apply arrays in JavaScript, you can follow these steps:**

**1. Declare an Array:** Start by declaring an array variable using the const, let, or var keyword.

For example:

```
const numbers = [];
```

**2. Add Elements:** Use the `push()` method or direct assignment to add elements to the array.

For example:

```
numbers.push(1);
numbers.push(2);
numbers.push(3);
```

**3. Access Elements:** Access individual elements in the array using their index. Arrays in JavaScript are zero-based, so the first element is at index 0. For example:

```
console.log(numbers[0]); // Output: 1
```

**4. Modify Elements:** You can modify elements in the array by assigning new values to specific indices.

```
numbers[1] = 4;
```

**5. Iterate Over Elements:** Use loops or array iteration methods like `forEach()`, `for...of`, or `map()` to iterate over the elements and perform operations on them. For example:

```
numbers.forEach(function(element) {
  console.log(element);
});
```

**6. Use Array Methods:** JavaScript provides various built-in array methods like `concat()`, `slice()`, `filter()`, `reduce()`, etc., to perform common operations on arrays. Utilize these methods based on your requirements.

**7. Remove Elements:** Use methods like `pop()`, `shift()`, or `splice()` to remove elements from the array.

For example:

```
numbers.pop(); // Removes the last element
```

**8. Get Array Length:** Use the `'length'` property to get the number of elements in the array.

For example:

```
console.log(numbers.length); // Output: 2
```

**9. Manipulate and Transform Data:** Arrays can be used to store and manipulate collections of data. Utilize array methods and operations to transform, filter, sort, or perform calculations on the array elements.

By following these steps, you can effectively apply arrays in JavaScript to store, access, modify, and manipulate collections of data.

- **JavaScript array literal**

**To apply a JavaScript array literal, you can follow these steps:**

1. Declare an Array Variable: Start by declaring a variable using the `'const'`, `'let'`, or `'var'` keyword to store the array. For example:

```
const fruits = [];"
```

2. Assign Values to the Array: Use the array literal syntax to assign values to the array. Separate each element with a comma and enclose them within square brackets `'[]'`.

**For example:**

```
const fruits = ['apple', 'banana', 'orange'];"
```

3. Access Array Elements: You can access individual elements in the array using their indices. Arrays in JavaScript are zero-based, so the first element is at index 0. For example:

```
console.log(fruits[0]); // Output: 'apple'
```

4. Modify Array Elements: You can modify elements in the array by assigning new values to specific indices.

For example:

```
fruits[1] = 'grape';
```

5. Iterate Over Array Elements: Use loops or array iteration methods like `forEach()`, `for...of`, or `map()` to iterate over the elements and perform operations on them.

For example:

```
fruits.forEach(function(fruit) {  
    console.log(fruit);  
});
```

6. Use Array Methods: JavaScript provides various built-in array methods like `push()`, `pop()`, `slice()`, `filter()`, `reduce()`, etc., to perform common operations on arrays. Utilize these methods based on your requirements.

7. Get Array Length: Use the `length` property to get the number of elements in the array.

For example:

```
console.log(fruits.length); // Output: 3
```

By following these steps, you can effectively apply a JavaScript array literal to create and work with arrays, assign values, access elements, modify data, iterate over elements, and utilize array methods.

Let's see the simple example of creating and using array in JavaScript.

```
<script>  
var emp=["Sonoo","Vimal","Ratan"];  
for (i=0;i<emp.length;i++){  
document.write(emp[i] + "<br/>");  
}  
</script>
```

The `.length` property returns the length of an array.

**Output of the above example**

Sonoo  
Vimal  
Ratan

### **JavaScript Array directly (new keyword)**

To apply a JavaScript array directly using the `new` keyword, you can follow these steps:

1. Declare an Array Variable: Start by declaring a variable using the `const`, `let`, or `var` keyword to store the array.

For example:

```
const fruits = new Array();
```

2. Assign Values to the Array: Use the `new` keyword followed by the `Array()` constructor to create a new array object. For example:

```
const fruits = new Array('apple', 'banana', 'orange');
```

3. Access Array Elements: You can access individual elements in the array using their indices. Arrays in JavaScript are zero-based, so the first element is at index 0. For example:

```
console.log(fruits[0]); // Output: 'apple'
```

4. Modify Array Elements: You can modify elements in the array by assigning new values to specific indices. For example:

```
fruits[1] = 'grape';
```

5. Iterate Over Array Elements: Use loops or array iteration methods like `forEach()`, `for...of`, or `map()` to iterate over the elements and perform operations on them.

For example:

```
fruits.forEach(function(fruit) {  
  console.log(fruit);  
});
```

6. Use Array Methods: JavaScript provides various built-in array methods like `push()`, `pop()`, `slice()`, `filter()`, `reduce()`, etc., to perform common operations on arrays. Utilize these methods based on your requirements.

7. Get Array Length: Use the `length` property to get the number of elements in the array.

**For example:**

```
console.log(fruits.length); // Output: 3'''
```

By following these steps, you can effectively apply a JavaScript array directly using the `new` keyword to create and work with arrays, assign values, access elements, modify data, iterate over elements, and utilize array methods.

Let's see an example of creating an array directly.

```
<script>
var i;
var emp = new Array();
emp[0] = "Arun";
emp[1] = "Varun";
emp[2] = "John";

for (i=0;i<emp.length;i++){
document.write(emp[i] + "<br>");
}
</script>
```

**Output of the above example**

Arun

Varun

John

### **JavaScript array constructor (new keyword)**

The example of creating an object by array constructor is given below.

```
<script>
var emp=new Array("Jai","Vijay","Smith");
for (i=0;i<emp.length;i++){
document.write(emp[i] + "<br>");
}
</script>
```

**Output of the above example**

Jai

Vijay

Smith

- **Types of JavaScript arrays:**

#### **One-Dimensional JavaScript array:**

To apply a one-dimensional JavaScript array, you can follow these steps:

1. Declare an Array Variable: Start by declaring a variable using the `const`, `let`, or `var` keyword to store the array.

For example:

```
const numbers = [];
```

2. Assign Values to the Array: Use the array literal syntax to assign values to the array. Separate each element with a comma and enclose them within square brackets `[]`.

For example:

```
const numbers = [1, 2, 3, 4, 5];
```

3. Access Array Elements: You can access individual elements in the array using their indices. Arrays in JavaScript are zero-based, so the first element is at index 0.

For example:

```
console.log(numbers[0]); // Output: 1
```

4. Modify Array Elements: You can modify elements in the array by assigning new values to specific indices.

For example:

```
numbers[1] = 10;
```

5. Iterate Over Array Elements: Use loops or array iteration methods like `forEach()`, `for...of`, or `map()` to iterate over the elements and perform operations on them. For example:

```
numbers.forEach(function(number) {  
  console.log(number);  
});
```

6. Use Array Methods: JavaScript provides various built-in array methods like `push()`, `pop()`, `slice()`, `filter()`, `reduce()`, etc., to perform common operations on arrays.

Utilize these methods based on your requirements.

7. Get Array Length: Use the `length` property to get the number of elements in the array. For example:

```
console.log(numbers.length); // Output: 5
```

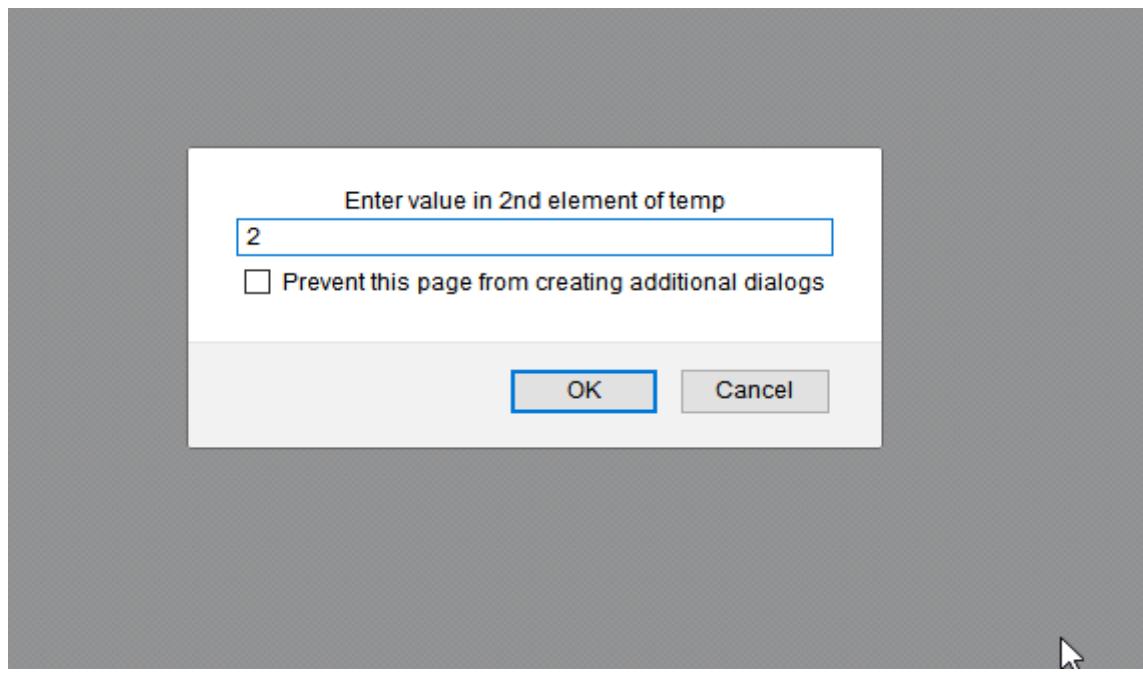
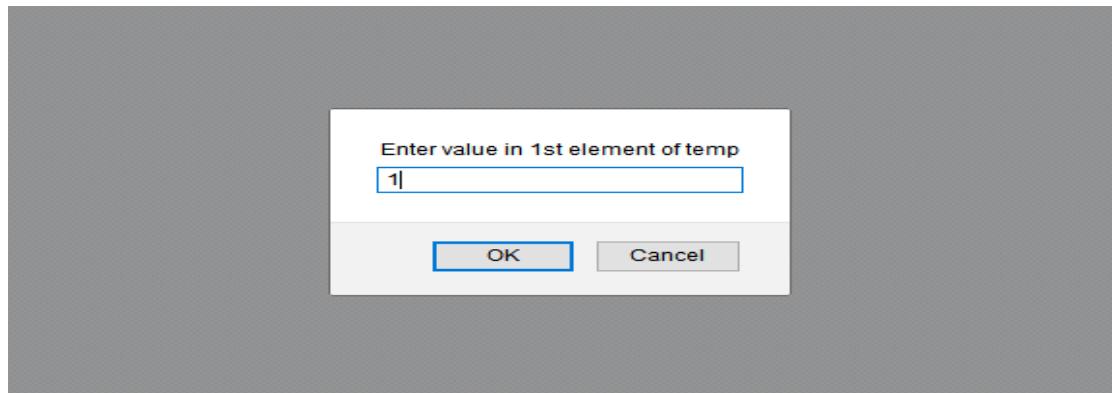
By following these steps, you can effectively apply a one-dimensional JavaScript array to create and work with arrays, assign values, access elements, modify data, iterate over elements, and utilize array methods.

### **Example:**

write JavaScript code to input values into individual elements of an array during the execution of the script and display the values of Js array:

```
<html>  
<body>  
<script type="text/JavaScript">  
temp = new Array(15);
```

```
temp[0]= parseInt(prompt("Enter value in 1st element of temp"));
temp[1]= parseInt(prompt("Enter value in 2nd element of temp"));
temp[2]= parseInt(prompt("Enter value in 3rd element of temp"));
temp[3]= parseInt(prompt("Enter value in 4th element of temp"));
temp[4]= parseInt(prompt("Enter value in 5th element of temp"));
document.write(temp[0]+ "<br>");
document.write(temp[1]+ "<br>");
document.write(temp[2]+ "<br>");
document.write(temp[3]+ "<br>");
document.write(temp[4]+ "<br>");
</script>
</body>
</html>
```



Enter value in 3rd element of temp

Prevent this page from creating additional dialogs

**OK** **Cancel**

Enter value in 4th element of temp

Prevent this page from creating additional dialogs

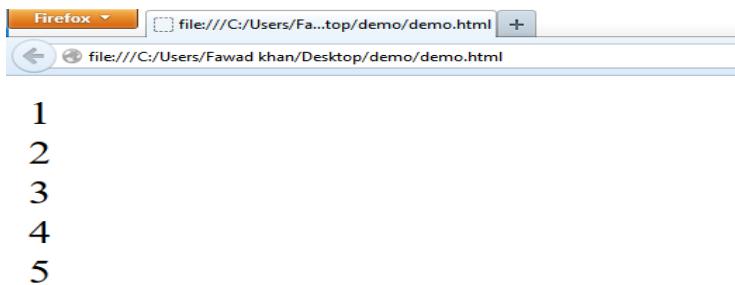
**OK** **Cancel**

Enter value in 5th element of temp

Prevent this page from creating additional dialogs

**OK** **Cancel**

The output will be displayed in the format below:



```
1  
2  
3  
4  
5
```

## JavaScript Multidimensional Array

To apply a multidimensional JavaScript array, you can follow these steps:

1. Declare an Array Variable: Start by declaring a variable using the `const`, `let`, or `var` keyword to store the multidimensional array. For example:

```
const matrix = [];
```

2. Assign Values to the Array: Use the array literal syntax to assign values to the multidimensional array. Each element in the outer array represents a row, and each element within the row arrays represents a column. For example:

```
const matrix = [  
    [1, 2, 3],  
    [4, 5, 6],  
    [7, 8, 9]  
];
```

3. Access Array Elements: You can access individual elements in the multidimensional array using their indices. Use two indices: the outer index for the row and the inner index for the column.

For example:

```
console.log(matrix[0][1]); // Output: 2
```

4. Modify Array Elements: You can modify elements in the multidimensional array by assigning new values to specific indices.

For example:

```
matrix[1][2] = 10;
```

5. Iterate Over Array Elements: Use nested loops or array iteration methods to iterate over the elements and perform operations on them. For example:

```
for (let i = 0; i < matrix.length; i++) {
```

```
for (let j = 0; j < matrix[i].length; j++) {  
    console.log(matrix[i][j]);  
}
```

6. Use Array Methods: JavaScript provides various built-in array methods like `push()`, `pop()`, `slice()`, `filter()`, `reduce()`, etc., to perform common operations on arrays. Utilize these methods based on your requirements.

7. Get Array Dimensions: You can use the `length` property of the outer and inner arrays to get the number of rows and columns, respectively. For example:

```
console.log(matrix.length); // Output: 3 (number of rows)  
console.log(matrix[0].length); // Output: 3 (number of columns in the first row)
```

By following these steps, you can effectively apply a multidimensional JavaScript array to create and work with arrays with multiple dimensions, assign values, access elements, modify data, iterate over elements, and utilize array methods.

#### For example

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

#### Create a Multidimensional Array

Here is how you can create multidimensional arrays in JavaScript.

#### 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];
```

Here, both example 1 and example 2 creates a multidimensional array with the same data.

#### Access Elements of an Array

You can access the elements of a multidimensional array using indices (**0, 1, 2 ...**).

For example,

```
let x = [  
    ['Jack', 24],  
    ['Sara', 23],  
    ['Peter', 24]  
];  
// access the first item  
console.log(x[0]); // ["Jack", 24]
```

```
// access the first item of the first inner array  
console.log(x[0][0]); // Jack  
  
// access the second item of the third inner array  
console.log(x[2][1]); // 24
```

You can think of a multidimensional array (in this case, x), as a table with 3 rows and 2 columns.

	Column 1	Column 2
Row 1	Jack x[0][0]	24 x[0][1]
Row 2	Sara x[1][0]	23 x[1][1]
Row 3	Peter x[2][0]	24 x[2][1]

## JavaScript Array Methods

To apply JavaScript array methods, you can follow these steps:

1. Create an Array: Start by creating an array and assigning it to a variable. You can use an array literal or the `new Array()` constructor. For example:

```
const numbers = [1, 2, 3, 4, 5];
```

2. Choose an Array Method: Determine which array method you want to apply based on the desired operation. Some commonly used array methods include `push()`, `pop()`, `shift()`, `unshift()`, `concat()`, `join()`, `slice()`, `splice()`, `indexOf()`, `lastIndexOf()`, `forEach()`, `map()`, `filter()`, `reduce()`, `sort()`, `reverse()`, and more.

3. Apply the Array Method: Use dot notation to access the array method and apply it to the array variable. Provide any necessary arguments to the method.

For example:

```
const doubledNumbers = numbers.map(function(number) {  
  return number * 2;  
});
```

4. Handle the Result: Store the result of the array method in a new variable or use it directly. Depending on the array method used, the result may be a new array, a modified array, or a single value.

5. Repeat as Needed: You can apply multiple array methods in sequence or combine them to achieve the desired results. Experiment with different array methods to perform various operations on your arrays.

By following these steps, you can effectively apply JavaScript array methods to manipulate, transform, filter, iterate, or perform calculations on arrays, based on your specific requirements.

### **Application of some array methods**

#### **JavaScript Array length**

The length property returns the length (size) of an array:

#### **Example**

```
<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let size = fruits.length;
document.getElementById("demo").innerHTML = size;
</script>
```

#### **Result**

4

#### **JavaScript Array toString()**

The JavaScript method `toString()` converts an array to a string of (comma separated) array values.

#### **Example**

```
<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.toString();
</script>
```

#### **Result:**

Banana, Orange, Apple, Mango

#### **The `join()` method** also joins all array elements into a string.

It behaves just like `toString()`, but in addition you can specify the separator:

#### **Example**

```
<script>
```

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.join(" * ");
</script>
```

### Result:

Banana \* Orange \* Apple \* Mango

## Popping and Pushing

When you work with arrays, it is easy to remove elements and add new elements.

This is what popping and pushing is:

Popping items **out** of an array, or pushing items **into** an array

### JavaScript Array pop()

The `pop()` method removes the last element from an array:

### Example

```
<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML = fruits;
fruits.pop();
document.getElementById("demo2").innerHTML = fruits;
</script>
```

### Result

Banana,Orange,Apple,Mango

Banana,Orange,Apple

The `pop()` method returns the value that was "popped out":

### Example

```
<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML = fruits.pop();
document.getElementById("demo2").innerHTML = fruits;
</script>
```

### Result

Mango

Banana,Orange,Apple

### JavaScript Array push()

The `push()` method adds a new element to an array (at the end):

### Example

```
<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
```

```
document.getElementById("demo1").innerHTML = fruits;  
fruits.push("Kiwi");  
document.getElementById("demo2").innerHTML = fruits;  
</script>
```

### Result

Banana,Orange,Apple,Mango

Banana,Orange,Apple,Mango,Kiwi

The push() method returns the new array length:

### Example

```
<script>  
const fruits = ["Banana", "Orange", "Apple", "Mango"];  
document.getElementById("demo1").innerHTML = fruits.push("Kiwi");  
document.getElementById("demo2").innerHTML = fruits;  
</script>
```

### Result

5

Banana,Orange,Apple,Mango,Kiwi

## Shifting Elements

Shifting is equivalent to popping, but working on the first element instead of the last.

### JavaScript Array shift()

The shift() method removes the first array element and "shifts" all other elements to a lower index.

### Example

```
<script>  
const fruits = ["Banana", "Orange", "Apple", "Mango"];  
document.getElementById("demo1").innerHTML = fruits;  
fruits.shift();  
document.getElementById("demo2").innerHTML = fruits;  
</script>
```

### Result

Banana,Orange,Apple,Mango

Orange,Apple,Mango

The shift() method returns the value that was "shifted out":

### Example

```
<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML = fruits.shift();
document.getElementById("demo2").innerHTML = fruits;
</script>
```

### Result

Banana

Orange, Apple, Mango

#### JavaScript Array unshift()

The unshift() method adds a new element to an array (at the beginning), and "unshifts" older elements:

### Example

```
<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML = fruits;
fruits.unshift("Lemon");
document.getElementById("demo2").innerHTML = fruits;
</script>
```

### Result

Banana,Orange,Apple,Mango

Lemon,Banana,Orange,Apple,Mango

The unshift() method returns the new array length:

### Example

```
<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML = fruits.unshift("Lemon");
document.getElementById("demo2").innerHTML = fruits;
</script>
```

### Result

5

Lemon,Banana,Orange,Apple,Mango

#### JavaScript Array length

The length property provides an easy way to append a new element to an array:

### Example

```
<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML = fruits;
```

```
fruits[fruits.length] = "Kiwi";
document.getElementById("demo2").innerHTML = fruits;
</script>
```

### Result

Banana,Orange,Apple,Mango  
Banana,Orange,Apple,Mango,Kiwi

#### JavaScript Array delete()

Warning !

Array elements can be deleted using the JavaScript operator delete.

Using delete leaves undefined holes in the array.

Use pop() or shift() instead.

### Example

```
<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML =
"The first fruit is: " + fruits[0];
delete fruits[0];
document.getElementById("demo2").innerHTML =
"The first fruit is: " + fruits[0];
</script>
```

### Result

The first fruit is: Banana

The first fruit is: undefined

#### Merging (Concatenating) Arrays

The concat() method creates a new array by merging (concatenating) existing arrays:

### Example (Merging Two Arrays)

```
<script>
const myGirls = ["Cecilie", "Lone"];
const myBoys = ["Emil", "Tobias", "Linus"];
const myChildren = myGirls.concat(myBoys);
document.getElementById("demo").innerHTML = myChildren;
</script>
```

### Result

Cecilie,Lone,Emil,Tobias,Linus

The concat() method does not change the existing arrays. It always returns a new array.

The concat() method can take any number of array arguments:

### Example (Merging Three Arrays)

```
<script>
const array1 = ["Cecilie", "Lone"];
const array2 = ["Emil", "Tobias", "Linus"];
const array3 = ["Robin", "Morgan"];
const myChildren = array1.concat(array2, array3);
document.getElementById("demo").innerHTML = myChildren;
</script>
```

### Result

Cecilie,Lone,Emil,Tobias,Linus,Robin,Morgan

The concat() method can also take strings as arguments:

### Example (Merging an Array with Values)

```
<script>
const myArray = ["Emil", "Tobias", "Linus"];
const myChildren = myArray.concat("Peter");
document.getElementById("demo").innerHTML = myChildren;
</script>
```

### Result

Emil,Tobias,Linus,Peter

## Flattening an Array

Flattening an array is the process of reducing the dimensionality of an array.

The flat() method creates a new array with sub-array elements concatenated to a specified depth.

### Example

```
<script>
const myArr = [[1,2],[3,4],[5,6]];
const newArr = myArr.flat();
document.getElementById("demo").innerHTML = newArr;
</script>
```

### Result

1,2,3,4,5,6

## Splicing and Slicing Arrays

The splice() method adds new items to an array.

The slice() method slices out a piece of an array.

### JavaScript Array splice()

The splice() method can be used to add new items to an array:

### Example

```
<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
```

```
document.getElementById("demo1").innerHTML = fruits;  
fruits.splice(2, 0, "Lemon", "Kiwi");  
document.getElementById("demo2").innerHTML = fruits;  
</script>
```

### Result

Banana,Orange,Apple,Mango

Banana,Orange,Lemon,Kiwi,Apple,Mango

The first parameter (2) defines the position **where** new elements should be **added** (spliced in).

The second parameter (0) defines **how many** elements should be **removed**.

The rest of the parameters ("Lemon", "Kiwi") define the new elements to be **added**.

The splice() method returns an array with the deleted items:

### Example

```
<script>  
const fruits = ["Banana", "Orange", "Apple", "Mango"];  
document.getElementById("demo1").innerHTML = "Original Array:<br> " + fruits;  
let removed = fruits.splice(2, 2, "Lemon", "Kiwi");  
document.getElementById("demo2").innerHTML = "New Array:<br>" + fruits;  
document.getElementById("demo3").innerHTML = "Removed Items:<br> " + removed;  
</script>
```

### Result

Original Array:

Banana,Orange,Apple,Mango

New Array:

Banana,Orange,Lemon,Kiwi

Removed Items:

Apple,Mango

### Using splice() to Remove Elements

With clever parameter setting, you can use splice() to remove elements without leaving "holes" in the array:

### Example

```
<script>  
const fruits = ["Banana", "Orange", "Apple", "Mango"];  
document.getElementById("demo1").innerHTML = fruits;  
fruits.splice(0, 1);  
document.getElementById("demo2").innerHTML = fruits;  
</script>
```

### Result

Banana,Orange,Apple,Mango

Orange,Apple,Mango

The first parameter (0) defines the position where new elements should be **added** (spliced in).

The second parameter (1) defines **how many** elements should be **removed**.

The rest of the parameters are omitted. No new elements will be added.

### JavaScript Array slice()

The slice() method slices out a piece of an array into a new array.

This example slices out a part of an array starting from array element 1 ("Orange"):

#### Example

```
<script>
const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
const citrus = fruits.slice(1);
document.getElementById("demo").innerHTML = fruits + "<br><br>" + citrus;
</script>
```

#### Result

Banana,Orange,Lemon,Apple,Mango

Orange,Lemon,Apple,Mango

#### Note:

The slice() method creates a new array.

The slice() method does not remove any elements from the source array.

This example slices out a part of an array starting from array element 3 ("Apple"):

#### Example

```
<script>
const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
const citrus = fruits.slice(3);
document.getElementById("demo").innerHTML = fruits + "<br><br>" + citrus;
</script>
```

#### Result

Banana,Orange,Lemon,Apple,Mango

Apple,Mango

The slice() method can take two arguments like slice(1, 3).

The method then selects elements from the start argument, and up to (but not including) the end argument.

#### Example

```
<script>
const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
const citrus = fruits.slice(1,3);
document.getElementById("demo").innerHTML = fruits + "<br><br>" + citrus;
</script>
```

#### Result

Banana,Orange,Lemon,Apple,Mango

Orange,Lemon

If the end argument is omitted, like in the first examples, the slice() method slices out the rest of the array.

#### Example

```
<script>
const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
const citrus = fruits.slice(2);
document.getElementById("demo").innerHTML = fruits + "<br><br>" + citrus;
</script>
```

#### Result

Banana,Orange,Lemon,Apple,Mango

Lemon,Apple,Mango

#### Automatic `toString()`

JavaScript automatically converts an array to a comma separated string when a primitive value is expected.

This is always the case when you try to output an array.

These two examples will produce the same result:

#### Example

```
<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.toString();
</script>
```

#### Result

Banana, Orange, Apple, Mango

### JavaScript Array Iterations

To apply JavaScript array iterations, you can follow these steps:

1. Create an Array: Start by creating an array and assigning it to a variable. You can use an array literal or the `new Array()` constructor.

For example:

```
const numbers = [1, 2, 3, 4, 5];
```

2. Choose an Iteration Method: Determine which array iteration method you want to apply based on the desired operation. Some commonly used iteration methods include `forEach()`, `for...of` loop, and `map()`.

3. Apply the Iteration Method: Use the chosen iteration method to loop through the array elements and perform operations on them. Provide a callback function as an argument to the iteration method. The callback function will be executed for each element in the array.

4. Handle the Result: Depending on the iteration method used, you can choose to store the result in a new variable or utilize it directly. For example, in the case of `map()`, the result is a new array.

5. Repeat as Needed: You can apply multiple iteration methods in sequence or combine them to achieve the desired results. Experiment with different iteration methods to perform various operations on your arrays.

By following these steps, you can effectively apply JavaScript array iterations to loop through array elements and perform operations on them, based on your specific requirements.

- **Using For Loop**

**Example:**

```
// Initializing the array with elements  
array = [11,12,13,14,15];  
for (i = 0; i < array.length; i++) {  
    // printing the elements of the array  
    console.log(array[i]);  
}
```

**Output:**

```
11  
12  
13  
14  
15
```

**Explanation:**

Here in the above code, we have first initialized the array with the integer type elements after that we used the for loop which accepts three expressions- first is the initialization of the variable, and we have specified the stopping condition after that, in the end, we have incremental expression. Inside the for loop in the curly braces, we have specified the statement for printing the elements of the array.

- **Using While Loop**

**Example:**

```
// Initializing the variable to iterate over the array  
idx = 0;  
//Initializing the array with elements  
array = [11,12,13,14,15];  
// while loop to iterate over the array  
while (idx < array.length){  
    //Printing the elements of the array  
    console.log(array[idx]);
```

```
// Incrementing the variable  
idx++;  
}  
}
```

**Output:**

```
11  
12  
13  
14  
15
```

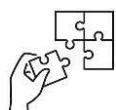
**Explanation:**

Here in the above code, we have first initialized a variable idx to iterate over the array and after that, we have initialized the array with the elements. Next, we used a while loop to iterate over the array after that inside the while loop, we used console.log () to print the current element of the array and at last, we incremented the variable to move to the next position of the array.



### Points to Remember

- A JavaScript array is a special type of object that allows you to store multiple values in a single variable. It is an object that represents a collection of similar type of elements. JavaScript arrays are divided into One-dimensional arrays and multi-dimensional arrays. Accessing array elements in JavaScript can be done using index-based notation.
- To apply arrays in JavaScript, you can Declare an Array, Add Elements, Access Element, Modify Elements, Iterate Over Elements and Use Array Methods for performing required tasks on arrays.



### Application of learning 2.6.

You need to use JavaScript arrays to store and manipulate student information. The system should allow teachers to input grades, calculate averages, and identify students who need additional support. You are tasked to develop a student grade tracking system for a school.



## Indicative content 2.7: Using JavaScript in HTML



Duration: 14 hrs



### Theoretical Activity 2.7.1: Description of JavaScript in HTML



#### Tasks:

1: In small groups, you are requested to answer the following questions related to the JavaScript in HTML :

- i. What is HTML Events
- ii. Describe window object
- iii. Define canvas in HTML
- iv. Describe JavaScript form validation
- v. Describe HTML DOM

2: Provide the answer for the asked questions and write them on papers.

3: Present the findings/answers to the whole class and choose the correct answer.

4: For more clarification, read the key readings 2.7.1 and ask questions where necessary.



#### Key readings 2.7.1.:

- **HTML events**

HTML events in JavaScript refer to actions or occurrences that can be detected and responded to by JavaScript code within an HTML document. These events are triggered by user interactions, changes in the document, or specific browser actions. JavaScript code can be used to listen for these events and execute custom logic or perform actions in response.

In JavaScript, there are numerous event types that can be used to detect and respond to specific actions or occurrences within an HTML document. These event types cover a wide range of user interactions, changes in the document, or specific browser actions.

HTML events are "things" that happen to HTML elements.

#### Window Event Attributes

Events triggered for the window object (applies to the <body> tag):

Attribute	Value	Description
onafterprint	Script	Script to be run after the document is printed
onbeforeprint	script	Script to be run before the document is printed
onbeforeunload	script	Script to be run when the document is about to be unloaded

onerror	script	Script to be run when an error occurs
onhashchange	script	Script to be run when there has been changes to the anchor part of a URL
onload	script	Fires after the page is finished loading
Onmessage	script	Script to be run when the message is triggered
onoffline	script	Script to be run when the browser starts to work offline
ononline	script	Script to be run when the browser starts to work online
Onpagehide	script	Script to be run when a user navigates away from a page
onpageshow	script	Script to be run when a user navigates to a page
Onpopstate	script	Script to be run when the window's history changes
onresize	script	Fires when the browser window is resized
Onstorage	script	Script to be run when a Web Storage area is updated
onunload	script	Fires once a page has unloaded (or the browser window has been closed)

### Form Events

These events are related to form elements and form submission.

Events triggered by actions inside a HTML form (applies to almost all HTML elements, but is most used in form elements):

Attribute	Value	Description
onblur	script	Fires the moment that the element loses focus
onchange	script	Fires the moment when the value of the element is changed
oncontextmenu	script	Script to be run when a context menu is triggered
onfocus	script	Fires the moment when the element gets focus
oninput	script	Script to be run when an element gets user input
oninvalid	script	Script to be run when an element is invalid
onreset	script	Fires when the Reset button in a form is clicked

onsearch	script	Fires when the user writes something in a search field (for <input="search">)
onselect	script	Fires after some text has been selected in an element
onsubmit	script	Fires when a form is submitted

### Keyboard Events:

These events are triggered when the user interacts with the keyboard.

Attribute	Value	Description
onkeydown	Script	Fires when a user is pressing a key
onkeypress	Script	Fires when a user presses a key
onkeyup	Script	Fires when a user releases a key

**Mouse Events:** These events are triggered by mouse-related actions.

Attribute	Value	Description
onclick	script	Fires on a mouse click on the element
ondblclick	script	Fires on a mouse double-click on the element
onmousedown	script	Fires when a mouse button is pressed down on an element
onmousemove	script	Fires when the mouse pointer is moving while it is over an element
onmouseout	script	Fires when the mouse pointer moves out of an element
onmouseover	script	Fires when the mouse pointer moves over an element
onmouseup	script	Fires when a mouse button is released over an element
Onmousewheel	script	Deprecated. Use the onwheel attribute instead
onwheel	script	Fires when the mouse wheel rolls up or down over an element

### Drag Events

Attribute	Value	Description
ondrag	script	Script to be run when an element is dragged
ondragend	script	Script to be run at the end of a drag operation
ondragenter	script	Script to be run when an element has been dragged to a valid drop target
ondragleave	script	Script to be run when an element leaves a valid drop target
ondragover	script	Script to be run when an element is being dragged over a valid drop target
ondragstart	script	Script to be run at the start of a drag operation
ondrop	script	Script to be run when dragged element is being dropped
onscroll	script	Script to be run when an element's scrollbar is being scrolled

### Clipboard Events

Attribute	Value	Description
oncopy	Script	Fires when the user copies the content of an element
oncut	Script	Fires when the user cuts the content of an element
onpaste	Script	Fires when the user pastes some content in an element

### Media Events

Events triggered by medias like videos, images and audio (applies to all HTML elements, but is most common in media elements, like `<audio>`, `<embed>`, `<img>`, `<object>`, and `<video>`).

Attribute	Value	Description
Onabort	Script	Script to be run on abort
Oncanplay	Script	Script to be run when a file is ready to start playing (when it has buffered enough to begin)

oncanplaythrough	<i>Script</i>	Script to be run when a file can be played all the way to the end without pausing for buffering
oncuechange	<i>Script</i>	Script to be run when the cue changes in a <track> element
ondurationchange	<i>Script</i>	Script to be run when the length of the media changes
Onemptied	<i>Script</i>	Script to be run when something bad happens and the file is suddenly unavailable (like unexpectedly disconnects)
Onended	<i>Script</i>	Script to be run when the media has reached the end (a useful event for messages like "thanks for listening")
Onerror	<i>Script</i>	Script to be run when an error occurs when the file is being loaded
onloadeddata	<i>Script</i>	Script to be run when media data is loaded
onloadedmetadata	<i>Script</i>	Script to be run when meta data (like dimensions and duration) are loaded
onloadstart	<i>Script</i>	Script to be run just as the file begins to load before anything is actually loaded
Onpause	<i>Script</i>	Script to be run when the media is paused either by the user or programmatically
Onplay	<i>Script</i>	Script to be run when the media is ready to start playing
Onplaying	<i>Script</i>	Script to be run when the media actually has started playing
Onprogress	<i>Script</i>	Script to be run when the browser is in the process of getting the media data
onratechange	<i>Script</i>	Script to be run each time the playback rate changes (like when a user switches to a slow motion or fast forward mode)
Onseeked	<i>Script</i>	Script to be run when the seeking attribute is set to false indicating that seeking has ended
Onseeking	<i>Script</i>	Script to be run when the seeking attribute is set to true indicating that seeking is active
Onstalled	<i>Script</i>	Script to be run when the browser is unable to fetch the media data for whatever reason

Onsuspend	<i>Script</i>	Script to be run when fetching the media data is stopped before it is completely loaded for whatever reason
Ontimeupdate	<i>Script</i>	Script to be run when the playing position has changed (like when the user fast forwards to a different point in the media)
onvolumechange	<i>Script</i>	Script to be run each time the volume is changed which (includes setting the volume to "mute")
Onwaiting	<i>Script</i>	Script to be run when the media has paused but is expected to resume (like when the media pauses to buffer more data)

### Misc Events

Attribute	Value	Description
<b>ontoggle</b>	<i>script</i>	Fires when the user opens or closes the <details> element

- **JavaScript HTML event listener**

In JavaScript, an event listener is a function that is attached to an HTML element to listen for a specific event and execute custom code in response.

Event listeners provide a flexible and powerful way to handle events in JavaScript, allowing developers to add interactivity and responsiveness to web pages.

#### The **addEventListener()** method

The `addEventListener()` method attaches an event handler to the specified element.

It attaches an event handler to an element without overwriting existing event handlers.

You can add many event handlers to one element.

You can add many event handlers of the same type to one element, i.e two "click" events.

You can add event listeners to any DOM object not only HTML elements. i.e the `window` object.

The `addEventListener()` method makes it easier to control how the event reacts to bubbling.

When using the `addEventListener()` method, the JavaScript is separated from the HTML markup, for better readability and allows you to add event listeners even when you do

not control the HTML markup. You can easily remove an event listener by using the `removeEventListener()` method.

The first parameter is the type of the event (like "click" or "mousedown" or any other HTML DOM Event.)

The second parameter is the function we want to call when the event occurs. The third parameter is a boolean value specifying whether to use event bubbling or event capturing. This parameter is optional.

**Note that** you don't use the "on" prefix for the event; use "click" instead of "onclick".

The `'addEventListener()'` method is a built-in JavaScript method that allows you to attach an event listener to an HTML element. It enables you to specify a function to be executed when a specific event occurs on the element.

**Here's how `'addEventListener()'` works:**

```
element.addEventListener(eventType, eventHandler, useCapture);
```

Parameters:

- `eventType`: A string that specifies the type of event to listen for, such as "click", "keydown", "submit", etc.
- `'eventHandler'`: The function to be executed when the specified event occurs.
- `useCapture` (optional): A boolean value that determines whether to use event capturing (true) or event bubbling (false, default behavior). This parameter is often omitted.

- **Add Many Event Handlers to the Same Element**

The `addEventListener()` method allows you to add many events to the same element, without overwriting existing events:

Add an Event Handler to the window Object

The `addEventListener()` method allows you to add event listeners on any HTML DOM object such as HTML elements, the HTML document, the window object, or other objects that support events, like the `xmlHttpRequest` object.

- **Event Bubbling or Event Capturing?**

There are two ways of event propagation in the HTML DOM, bubbling and capturing.

Event propagation is a way of defining the element order when an event occurs.

If you have a `<p>` element inside a `<div>` element, and the user clicks on the `<p>` element, which element's "click" event should be handled first? In *bubbling* the innermost element's event is handled first and then the outer: the `<p>` element's click event is handled first, then the `<div>` element's click event.

In *capturing* the outermost element's event is handled first and then the inner: the `<div>` element's click event will be handled first, then the `<p>` element's click event.

With the `addEventListener()` method you can specify the propagation type by using the "useCapture" parameter:

- **The `removeEventListener()` method**

The `removeEventListener()` method in JavaScript is used to remove an event listener that has been previously attached to an HTML element using the `addEventListener()` method. It allows you to detach a specific event handler function from an element, preventing it from being executed when the specified event occurs.

**Here's how the `removeEventListener()` method works:**

```
element.removeEventListener(eventType, eventHandler, useCapture);
```

**Parameters:**

- **eventType**: A string that specifies the type of event for which the listener was originally attached.
- **eventHandler**: The same function that was used as the event handler when attaching the listener.
- **useCapture (optional)**: A boolean value that determines whether the listener was attached for event capturing ('true') or event bubbling ('false', default behavior). This parameter should match the one used when attaching the listener.

The `removeEventListener()` method removes event handlers that have been attached with the `addEventListener()` method:

- **Window Object**

In JavaScript, the `window` object is a global object that represents the browser window or the global context in which JavaScript code is executed. It serves as the top-level object in the browser's JavaScript object hierarchy.

**■ Window object Properties**

**● console**

A `console` traditionally refers to a computer terminal where a user may input commands and view output such as the results of inputted commands or status messages from the computer.

A **Console method** is an object used to access the browser debugging console. The `console` object provides multiple methods to use.

The `console` object in JavaScript provides methods for logging information to the browser's console. It is commonly used for debugging and troubleshooting purposes.

Here are some commonly used methods of the '`console`' object:

**1. log**

`console.log()`: This method is used to log messages to the console. You can pass one or more values or variables as arguments, and they will be displayed in the console.

For example:

```
console.log('Hello, world!');  
console.log('The value of x is:', x);
```

**2. info**

`console.info()`: This method is used to log informational messages to the console. It is typically used to provide additional information or details about the code execution. It can accept one or more values or variables as arguments.

**For example:**

```
console.info('This is an informational message.');
console.info('Details:', detailsObject);
```

### **3. warn**

`console.warn()`: This method is used to log warning messages to the console. It is useful for highlighting potential issues or problematic areas in your code. It can accept one or more values or variables as arguments.

**For example:**

```
console.warn('This is a warning message!');
console.warn('Potential performance issue:', functionName);
```

### **4. error**

`console.error()`: This method is used to log error messages to the console. It is often used to indicate and track down errors in your code. It can accept one or more values or variables as arguments.

**For example:**

```
console.error('An error occurred!');
console.error('Invalid input:', userInput);'''
```

### **5. clear**

`console.clear()`: This method is used to clear the console, removing any previously logged messages. It does not accept any arguments.

**For example:**

```
console.clear();
```

## **• document**

A Document object represents the HTML document that is displayed in that window.

When an HTML document is loaded into a web browser, it becomes a **document object**.

The `document` object in JavaScript represents the HTML document currently loaded in the browser window. It provides methods, properties, and events that allow you to manipulate and interact with the content, structure, and styles of the document. Here is a description of the `document` object and its key features:

**1. DOM Manipulation:** The `document` object provides methods to access and manipulate elements in the document's DOM (Document Object Model).

Some commonly used methods include:

- getElementById()**: Retrieves an element from the document based on its unique `id` attribute.
- querySelector()**: Returns the first element that matches a specified CSS selector.
- **querySelectorAll()**: Returns a list of all elements that match a specified CSS selector.
- createElement()**: Creates a new HTML element.
- createTextNode()**: Creates a new text node.

**-appendChild():** Appends a node as the last child of a specified parent node.

**-removeChild():** Removes a child node from a specified parent node.

**2. Event Handling:** The `document` object allows you to handle events that occur in the document. Some commonly used methods and properties related to event handling include:

**-addEventListener():** Attaches an event listener to the document or a specific element.

**-removeEventListener():** Removes an event listener from the document or a specific element.

**-createEvent():** Creates a new event object that can be dispatched later.

**-dispatchEvent():** Dispatches a specified event to the document or a specific element.

**3. Document Information:** The `document` object provides properties to access information about the document itself.

Some commonly used properties include:

**-title:** Gets or sets the title of the document.

**-URL:** Gets the URL of the document.

**-domain:** Gets or sets the domain of the document's URL.

**-head:** Gets the `<head>` element of the document.

**-body:** Gets the `<body>` element of the document.

**4. Styling and CSS:** The `document` object allows you to access and modify styles and CSS-related information.

Some commonly used properties include:

**- styleSheets:** Provides access to the stylesheets associated with the document.

**- getElementById().style:** Allows you to access and modify the inline CSS styles of an element.

**- querySelector().classList:** Allows you to access and modify the classes of an element.

These are just a few examples of the methods, properties, and features available on the document object. The `document` object is a powerful tool in JavaScript that enables you to dynamically manipulate the HTML document, handle events, access and modify elements, and interact with the structure, content, and styles of the document.

#### ❖ innerHeight

The innerHeight property returns the height of a window's content area.

The innerHeight property in JavaScript is a property of the window object. It represents the height, in pixels, of the viewport's content area, excluding any scrollbars or other UI elements.

**Here are some key points about the `innerHeight` property:**

- The `innerHeight` property provides the height of the browser window's content area, which is the visible portion of a webpage.
- It does not include the height of any browser chrome, such as the address bar or toolbar, or any scrollbars that may be present.
- The value of `innerHeight` can change dynamically as the user resizes the browser window.
- You can access the `'innerHeight'` property using the `'window.innerHeight'` syntax.

**Here's an example of how you can use the `'innerHeight'` property:**

```
console.log(window.innerHeight); // Outputs the current height of the viewport's content area
```

By using the `'innerHeight'` property, you can retrieve the height of the visible area of the browser window. This information can be useful for responsive design, determining the available space for content, or adjusting the layout of elements based on the available vertical space.

❖ **innerWidth**

The `innerWidth` property returns the width of a window's content area.

The `'innerWidth'` property in JavaScript is a property of the `'window'` object. It represents the width, in pixels, of the viewport's content area, excluding any scrollbars or other UI elements.

**Here are some key points about the `'innerWidth'` property:**

- The `'innerWidth'` property provides the width of the browser window's content area, which is the visible portion of a webpage.
- It does not include the width of any browser chrome, such as the address bar or toolbar, or any scrollbars that may be present.
- The value of `'innerWidth'` can change dynamically as the user resizes the browser window.
- You can access the `'innerWidth'` property using the `'window.innerWidth'` syntax.

**Here's an example of how you can use the `'innerWidth'` property:**

```
console.log(window.innerWidth); // Outputs the current width of the viewport's content area
```

By using the `'innerWidth'` property, you can retrieve the width of the visible area of the browser window. This information can be useful for responsive design, determining the available space for content, or adjusting the layout of elements based on the available horizontal space.

❖ **length**

The `length` property returns the number of (framed) windows in the window.

In JavaScript, the `'length'` property is a built-in property that is available on certain objects, such as strings, arrays, and collections. It represents the number of elements or characters contained in the object.

**Here are some key points about the `length` property:**

- **For strings:** The `length` property returns the number of characters in a string.

```
const str = "Hello, world!";
console.log(str.length); // Outputs 13
```

- **For arrays:** The `length` property returns the number of elements in an array.

```
const arr = [1, 2, 3, 4, 5];
console.log(arr.length); // Outputs 5
```

- **For collections:** Some JavaScript objects, such as HTML collections returned by methods like `getElementsByTagName()` or `querySelectorAll()`, also have a `length` property that represents the number of elements in the collection.

```
const elements = document.getElementsByTagName("p");
console.log(elements.length); // Outputs the number of <p> elements in the document.
```

The `length` property is read-only, meaning you cannot directly modify its value. Instead, it reflects the current number of elements or characters in the object.

- It's important to note that the `length` property is zero-based, meaning it counts elements or characters starting from index 0. So, the highest index or position in an array or string will always be `length - 1`

The length property is a convenient way to determine the size or number of elements in strings, arrays, or collections in JavaScript. It allows you to access and work with the length of these objects dynamically, enabling you to perform various operations and checks based on their size.

❖ **localStorage**

The `localStorage` read-only property of the `window` interface allows you to access a `Storage` object for the Document's origin.

In JavaScript, the `localStorage` object is a web storage mechanism that allows you to store key-value pairs locally in the browser. It provides a simple way to persistently store data on the client-side, even when the browser is closed and reopened.

**Here are some key points about `localStorage`**

- **localStorage** is part of the Web Storage API, which includes `localStorage` and `sessionStorage`. Both provide similar functionality, but with different scopes and lifetimes.
- The `localStorage` object is accessible globally in the browser's JavaScript environment, allowing you to store and retrieve data from any page within the same domain.

- The data stored in `localStorage` is persistent, meaning it remains available even after the browser is closed and reopened. It is stored indefinitely until explicitly cleared by the user or through JavaScript code.
- You can use the `localStorage.setItem()` method to store a value with a specified key, and `localStorage.getItem()` to retrieve the value associated with a key.

**For example:**

```
localStorage.setItem('username', 'John');
const username = localStorage.getItem('username');
console.log(username); // Outputs 'John'
```

- You can also use dot notation to set and retrieve values from `localStorage`:

```
localStorage.username = 'John';
const username = localStorage.username;
console.log(username); // Outputs 'John'
```

- To remove an item from `localStorage`, you can use the `localStorage.removeItem()` method and pass in the key of the item you want to remove.
- To clear all items stored in `localStorage`, you can use the `localStorage.clear()` method
- It's important to note that `localStorage` is subject to browser storage limits, which vary across different browsers.

Typically, the limit is around 5MB per origin (domain). `localStorage` provides a convenient way to store and retrieve data locally in the browser, making it useful for tasks such as saving user preferences, caching data, or persisting state between page reloads.

However, it's important to handle data stored in `localStorage` carefully, as it is accessible to JavaScript code running on the same domain and can be manipulated by the user.

❖ **Location**

The `location` object contains information about the current URL.

In JavaScript, the **location** object represents the current URL of the browser window or the URL of the webpage that is currently loaded.

It provides properties and methods to access and manipulate different components of the URL.

**Here are some key points about the `location` object:**

- The `location` object is a property of the global `window` object, which is accessible in the browser's JavaScript environment.
- The `location` object has various properties that provide information about different parts of the URL:
- **location.href:** Returns the complete URL of the current webpage.

- **location.protocol:** Returns the protocol (e.g., "http:", "https:", "file:") of the current URL.
- **location.host:** Returns the hostname and port number of the current URL.
- **location.hostname:** Returns the hostname (domain) of the current URL.
- **location.port:** Returns the port number of the current URL.
- **location.pathname:** Returns the path and filename of the current URL.
- **location.search:** Returns the query string of the current URL.
- **location.hash:** Returns the fragment identifier (anchor) of the current URL.
- You can also modify certain properties of the `location` object to navigate to a different URL or modify specific components of the current URL.

**For example:**

```
location.href = 'https://www.example.com'; // Navigates to a different URL  
location.hash = 'section1'; // Changes the fragment identifier of the current URL
```

- The location object provides methods to perform common navigation actions:
- **location.reload():** Reloads the current page.
- **location.replace(url):** Replaces the current URL with a new URL without adding a new entry to the browser's history.
- **location.assign(url):** Loads a new URL, adding a new entry to the browser's history.
- The location object is read-only for most properties, meaning you can access their values but cannot directly modify them. However, you can assign a new URL to the **location.href** property or use the provided methods to navigate to different URLs.

The location object is a powerful tool for working with URLs in JavaScript. It allows you to access and manipulate different parts of the URL, navigate to different web pages, and control the browser's history.

#### ❖ Methods

In JavaScript, methods are functions that are associated with objects and can be called to perform specific actions or operations on those objects. Here are some key points about methods in JavaScript:

- Methods are defined within objects and are accessed using dot notation or bracket notation.
- Methods can be predefined methods provided by JavaScript itself, or they can be custom methods defined by developers.
- Predefined methods are built-in functions that are available on certain objects and can be called directly. Examples include `alert()`, `setInterval()`, and `clearInterval()` .

- Custom methods are functions that are defined within an object and can be called on instances of that object. They are used to encapsulate behavior or actions related to the object.

#### ❖ alert()

The alert() method displays an alert box with a message and an OK button. It is used when you want information to come through to the user.

The alert() function is a predefined method in JavaScript that displays a modal dialog box with a message and an OK button. It is commonly used for displaying simple notifications or alerts to the user.

Here are some key points about the alert() function.

- The `alert()` function takes a single argument, which is the message to be displayed in the dialog box. The message can be a string or a variable that holds a string value.
- When the `alert()` function is called, it interrupts the execution of the script and displays the dialog box with the specified message.
- The dialog box typically appears as a small window in the center of the browser window, and it prevents the user from interacting with the rest of the page until the OK button is clicked.
- Here's an example of how to use the `alert()` function:

```
alert('Hello, world!'); // Displays an alert dialog with the message "Hello, world!"
```

**Syntax:** alert(message)

Example: alert("Hello! I am an alert box!!");

#### ❖ setInterval()

The setInterval() method calls a function at specified intervals (in milliseconds).

The setInterval() method continues calling the function until clearInterval() is called, or the window is closed.

Syntax: setInterval(function, milliseconds, param1, param2, ...)

Parameter	Description
function	Required. The function to execute
milliseconds	Required. The execution interval. If the value is less than 10, 10 is used

param1, param2,    Optional.

Additional parameters to pass to the function

Not supported in IE9 and earlier.

Example: `setInterval(function () {element.innerHTML += "Hello"}, 1000);`

#### ❖ **clearInterval()**

The `clearInterval()` function is a predefined method in JavaScript that is used to stop the execution of a recurring action that was initiated by the `setInterval()` function. It is commonly used to cancel or clear a timer set by `setInterval()`.

Here are some key points about the `clearInterval()` function:

- The clearInterval() function takes a single argument, which is the identifier of the interval to be cleared. This identifier is typically returned by the `setInterval()` function when it is called to start a recurring action.
- When the clearInterval() function is called with the interval identifier, it stops the execution of the recurring action associated with that identifier.
- Here's an example of how to use the `setInterval()` and `clearInterval()` functions together:

```
const intervalId = setInterval(function() {  
  console.log('This message will be logged every 1 second'); }, 1000);  
  
// After some time or under certain conditions, you can clear the interval using  
// clearInterval():  
  
clearInterval(intervalId);
```

- In the example above, the setInterval() function is used to execute a function every 1 second (1000 milliseconds). The interval identifier returned by `setInterval()` is stored in the `intervalId` variable, which is later passed to `clearInterval()` to stop the recurring action.

- It's important to note that the interval identifier passed to `clearInterval()` must be the exact identifier returned by `setInterval()`. If you pass a different identifier or an invalid value, the interval will not be cleared.

- By using clearInterval(), you can control the execution of recurring actions and prevent them from running indefinitely. This can be useful for managing timers, animations, or any other actions that need to be stopped or canceled at a specific point in your code.

The clearInterval() function is an essential tool for managing recurring actions in JavaScript. It provides a way to stop the execution of a timer set by setInterval() and gives you control over when and how long the recurring action should run.

The clearInterval() method clears a timer set with the setInterval() method.

**Note:**

To clear an interval, use the id returned from setInterval():

```
myInterval = setInterval(function, milliseconds);
```

Then you can stop the execution by calling clearInterval():

```
clearInterval(myInterval);
```

Syntax: clearInterval(intervalId)

**❖ setTimeout()**

The `setTimeout()` function is a predefined method in JavaScript that allows you to execute a function or evaluate an expression after a specified delay (in milliseconds). It is commonly used to schedule a one-time action to occur in the future. Here are some key points about the `setTimeout()` function:

- The `setTimeout()` function takes two arguments: a function or an expression to be executed, and the delay in milliseconds before the execution should occur.
- The function or expression provided as the first argument will be executed once, after the specified delay has passed.
- Here's an example of how to use the `setTimeout()` function:

```
setTimeout(function() {  
  console.log('This message will be logged after 2 seconds');  
}, 2000);
```

- In the example above, the `setTimeout()` function is used to schedule the execution of an anonymous function after a delay of 2000 milliseconds (2 seconds). After the delay, the function will be executed, and the specified message will be logged to the console.
- The delay specified in `setTimeout()` can be any non-negative integer. A value of 0 or a negative number will still result in a minimum delay of 4 milliseconds due to browser limitations.
- The `setTimeout()` function returns a unique identifier that can be used with the `clearTimeout()` function to cancel the execution of the scheduled action before it occurs.
- Here's an example of how to use `clearTimeout()` to cancel a scheduled action:

```
const timeoutId = setTimeout(function() {  
  console.log('This message will not be logged');  
}, 2000);  
clearTimeout(timeoutId);
```

- In the example above, the `clearTimeout()` function is called with the `timeoutId` returned by `setTimeout()`, effectively canceling the execution of the scheduled action.

The `setTimeout()` method calls a function after a number of milliseconds.

Syntax: `setTimeout(function, milliseconds, param1, param2,)`

#### Parameters

Parameter	Description
<code>function</code>	Required.  The function to execute.
<code>milliseconds</code>	Optional.  Number of milliseconds to wait before executing. Default value is 0.
<code>param1,param2</code>	Optional  Parameters to pass to the function  Not supported in IE9 and earlier

#### ❖ `clearTimeout()`

The `clearTimeout()` method clears a timer set with the `setTimeout()` method.

Syntax: `clearTimeout(id_of_settimeout)`

#### ❖ `open()`

The `open()` method opens a new browser window, or a new tab, depending on your browser settings and the parameter values.

Syntax: `window.open(URL, name, specs, replace)`

#### Parameters

Parameter	Description
<code>URL</code>	Optional.  The URL of the page to open.  If no URL is specified, a new blank window/tab is opened

name	Optional. The target attribute or the name of the window. The following values are supported:
------	---

<b>Value</b>	<b>Description</b>
_blank default	URL is loaded into a new window, or tab. This is the
_parent	URL is loaded into the parent frame
_self	URL replaces the current page
_top	URL replaces any framesets that may be loaded

**Example:** Open an about:blank page in a new window/tab:

```
var myWindow = window.open("", "", "width=200,height=100");
```

❖ **confirm()**

The confirm() method displays a dialog box with a message, an OK button, and a Cancel button.

The confirm() method returns true if the user clicked "OK", otherwise false.

Syntax: confirm(message)

- **close()**

The close() method closes a window.

Syntax: window.close()

- **stop()**

The stop() method stops window loading.

The stop() method is the same as clicking stop in the browser.

Syntax: window.stop()

❖ **print()**

The print() method prints the contents of the current window.

The print() method opens the Print Dialog Box, which lets the user to select preferred printing options.

Syntax: window.print()

✓ **JavaScript form validation**

JavaScript form validation refers to the process of validating user input in HTML forms using JavaScript code. It helps ensure that the data entered by users meets certain criteria or constraints before it is submitted to a server or processed further.

**Data validation** is the process of ensuring that user input is clean, correct, and useful.

The form validation process typically consists of two parts—the *required fields validation* which is performed to make sure that all the mandatory fields are filled in, and the *data format validation* which is performed to ensure that the type and format of the data entered in the form is valid.

**JavaScript form validation can encompass various types of validation checks, including:**

1. Required fields: Ensuring that mandatory fields are filled in before the form can be submitted.
2. Data format validation: Verifying that the input matches the expected format, such as validating email addresses, phone numbers, dates, or URLs.
3. Length constraints: Checking if the input has a minimum or maximum length requirement.
4. Numeric input validation: Validating that numeric fields contain valid numbers and fall within specified ranges.
5. Pattern matching: Using regular expressions to validate input against specific patterns or rules.
6. Confirmation fields: Comparing the values of two fields, such as password and confirm password fields, to ensure they match.
7. Custom validation: Implementing custom validation logic based on specific business rules or requirements.

Client-side form validation using JavaScript improves the user experience by providing immediate feedback to users when they submit a form, reducing the need for server round-trips and enhancing data integrity. However, it is important to note that client-side validation should always be supplemented with server-side validation to ensure data integrity and security, as client-side validation can be bypassed or manipulated by malicious users.

By implementing JavaScript form validation, developers can create forms that validate user input in real-time, ensuring that the data entered by users meets the required criteria and minimizing potential errors or inconsistencies in the submitted data.

JavaScript provides a range of methods and properties to facilitate form validation, including accessing form elements and their values, performing conditional checks, using

regular expressions for pattern matching, and displaying error messages or applying visual cues to indicate validation errors.

- **Canvas**

- ❖ **Introduction**

<canvas> is an HTML element which can be used to draw graphics via scripting (usually JavaScript). This can, for instance, be used to draw graphs, combine photos, or create simple animations.

The <canvas> element is a rectangular area on a web page where you can draw and manipulate graphics. It is defined with a width and height attribute, which determine the size of the canvas on the screen. It is defined in HTML with a width and height attribute. It acts as a container for graphics and provides a blank canvas for drawing.

- To work with the canvas element in JavaScript, you need to obtain a rendering context using the getContext() method. The most commonly used context is the 2D rendering context, obtained by passing '2d' as an argument to getContext().
- Once you have the rendering context, you can use its methods and properties to draw various shapes, lines, text, images, and more on the canvas.
- The canvas API provides methods like fillRect(), strokeRect(), arc(), lineTo(), and fillText() for drawing different elements on the canvas. You can set attributes like stroke color, fill color, line width, and font properties to customize the appearance of the drawn elements.
- The canvas element also supports transformations, allowing you to scale, rotate, and translate the canvas or individual objects on it using methods like scale(), rotate(), and translate().
- Animations can be created on the canvas by repeatedly redrawing the canvas at specific intervals using techniques like requestAnimationFrame() or setInterval(). By updating the canvas content in each frame, you can achieve smooth and dynamic animations.
- Images can be loaded onto the canvas using the drawImage() method. This allows you to display images, create sprites, or manipulate pixels on the canvas.
- The canvas element can also handle user interactions by listening to events like mouse clicks, mouse movement, or touch events. This enables you to create interactive applications and games.
- The canvas element can be styled using CSS to control its position, size, and appearance within the web page layout.

### **Canvas Examples**

A **canvas** is a rectangular area on an HTML page. By default, a canvas has no border and no content.

The markup looks like this:

```
<canvas id="myCanvas" width="200" height="100"></canvas>
```

**Note:** Always specify an id attribute (to be referred to in a script), and a width and height attribute to define the size of the canvas. To add a border, use the style attribute.

### ✓ JavaScript HTML DOM

The Document Object Model (DOM) is the data representation of the objects that comprise the structure and content of a document on the web.

JavaScript HTML DOM (Document Object Model) is a programming interface that represents the structure of an HTML document as a tree-like structure. It allows JavaScript to interact with and manipulate the elements, attributes, and content of an HTML document.

**Accessing HTML elements:** The DOM provides methods to access HTML elements. You can use methods like `getElementById()`, `getElementsByClassName()`, `getElementsByTagName()`, or `querySelector()` to select elements based on their IDs, class names, tag names, or CSS selectors.

#### ● innerHTML

The `innerHTML` property sets or returns the HTML content (inner HTML) of an element.

In JavaScript, the `innerHTML` property is used to get or set the HTML content of an element. It allows you to access the markup inside an element, including its child elements, text nodes, and HTML tags. Here's how you can use the `innerHTML` property:

1. Get the HTML content: To retrieve the HTML content of an element, you can simply access the `innerHTML` property.

```
const element = document.getElementById('myElement');
const htmlContent = element.innerHTML;
console.log(htmlContent);
```

2. Set the HTML content: You can also use the `innerHTML` property to set new HTML content for an element. This can include plain text, HTML tags, or a combination of both.

```
const element = document.getElementById('myElement');
element.innerHTML = '<h1>New Heading</h1><p>New paragraph content</p>';
```

#### ● getElementById

In JavaScript, the `getElementById()` method is used to retrieve an HTML element from the document based on its unique ID attribute. It allows you to access and manipulate specific elements by referring to their unique identifier.

### Here's how you can use the `getElementById()` method:

1. Syntax: The `getElementById()` method is called on the `document` object and takes the ID of the element as a parameter. The ID is a string value that should match the value of the `id` attribute of the desired element.

```
const element = document.getElementById('elementId');
```

2. Retrieving an element: When you call `getElementById()` with the ID of an element, it returns the element object representing that specific element in the document. You can then store this object in a variable for further use.

```
const element = document.getElementById('myElement');
```

- The `getElementById()` method returns an element with a specified value.
- The `getElementById()` method returns null if the element does not exist.
- The `getElementById()` method is one of the most common methods in the HTML DOM.

#### • **getElementsByClassName**

The `getElementsByClassName()` method returns a collection of elements with a specified class name(s).

In JavaScript, the `getElementsByClassName()` method is used to retrieve a collection of HTML elements based on their class name. It allows you to access and manipulate multiple elements that share the same class. Here's how you can use the `getElementsByClassName()` method:

1. Syntax: The `getElementsByClassName()` method is called on the document object and takes the class name as a parameter. The class name is a string value that should match the value of the class attribute of the desired elements.

```
const elements = document.getElementsByClassName('className');
```

2. Retrieving elements: When you call `getElementsByClassName()` with the class name, it returns a collection (array-like object called a "NodeList") of elements that have the specified class. You can then iterate over this collection or access specific elements using index notation.

```
const elements = document.getElementsByClassName('myClass');
```

```
// Iterate over the collection
for (let i = 0; i < elements.length; i++) {
  const element = elements[i];
  // Perform actions on each element
}
// Access a specific element
const firstElement = elements[0];
```

#### • **getElementsByName**

The `getElementsByName()` method returns a collection of elements with a specified name.

In JavaScript, the `'getElementsByName()'` method is used to retrieve a collection of HTML elements based on their `'name'` attribute. It allows you to access and manipulate multiple elements that share the same name.

Here's how you can use the `'getElementsByName()'` method:

**1. Syntax:** The `getElementsByName()` method is called on the `'document'` object and takes the name attribute as a parameter. The name is a string value that should match the value of the name attribute of the desired elements.

```
const elements = document.getElementsByName('nameAttribute');
```

**2. Retrieving elements:** When you call `getElementsByName()` with the name attribute, it returns a collection (an array-like object called a "NodeList") of elements that have the specified name. You can then iterate over this collection or access specific elements using index notation.

```
const elements = document.getElementsByName('myName');
// Iterate over the collection
for (let i = 0; i < elements.length; i++) {
  const element = elements[i];
  // Perform actions on each element
}
// Access a specific element
const firstElement = elements[0];
```

- **getElementsByTagName**

The `getElementsByTagName()` method returns a collection of all elements with a specified tag name.

In JavaScript, the `getElementsByTagName()` method is used to retrieve a collection of HTML elements based on their tag name. It allows you to access and manipulate multiple elements that share the same tag name.

Here's how you can use the `'getElementsByTagName()'` method:

**1. Syntax:** The `getElementsByTagName()` method is called on the `document` object and takes the tag name as a parameter. The tag name is a string value that represents the desired HTML tag.

```
const elements = document.getElementsByTagName('tagName');
```

**2. Retrieving elements:** When you call `getElementsByTagName()` with the tag name, it returns a collection (an array-like object called a "HTMLCollection") of elements that have the specified tag name. You can then iterate over this collection or access specific elements using index notation.

```
const elements = document.getElementsByTagName('div');
```

```
// Iterate over the collection
for (let i = 0; i < elements.length; i++) {
  const element = elements[i];
  // Perform actions on each element
}
// Access a specific element
const firstElement = elements[0];
```

- **querySelector**

The `querySelector()` method returns the first element that matches a CSS selector.

In JavaScript, the `'querySelector()'` method is a powerful tool for selecting and retrieving elements from the document using CSS selector syntax. It allows you to select elements based on various criteria such as tag name, class name, ID, attribute, or even complex selectors. Here's how you can use the `'querySelector()'` method:

1. Syntax: The `querySelector()` method is called on the document object or on any element and takes a CSS selector as a parameter. The CSS selector is a string value that specifies the criteria for selecting the desired element(s).

```
const element = document.querySelector('selector');
```

2. Selecting elements: When you call `querySelector()` with a CSS selector, it returns the first element that matches the specified selector. If no matching element is found, it returns `'null'`.

You can select elements based on various criteria:

- **By tag name:** Use the tag name as the selector to select elements with that specific tag.

```
const element = document.querySelector('div');
```

- **By class name:** Use the class name preceded by a dot to select elements with that specific class.

```
const element = document.querySelector('.myClass');
```

- **By ID:** Use the ID preceded by a hash symbol to select the element with that specific ID.

```
const element = document.querySelector('#myId');
```

- **By attribute:** Use attribute selectors to select elements based on attribute values.

```
const element = document.querySelector('[data-attribute="value"]');
```

- **querySelectorAll**

The `querySelectorAll()` method returns all elements that matches a CSS selector(s).

In JavaScript, the `querySelectorAll()` method is similar to `querySelector()`, but it returns a collection of all elements in the document that match a specified CSS selector. It allows

you to select multiple elements based on various criteria using CSS selector syntax. Here's how you can use the querySelectorAll() method:

**1. Syntax:** The querySelectorAll() method is called on the document object or on any element and takes a CSS selector as a parameter. The CSS selector is a string value that specifies the criteria for selecting the desired elements.

```
const elements = document.querySelectorAll('selector');
```

**2. Selecting elements:** When you call querySelectorAll() with a CSS selector, it returns a collection (a "NodeList" or "StaticNodeList" object) of all elements that match the specified selector. If no matching elements are found, it returns an empty collection.

You can select elements based on various criteria:

- **By tag name:** Use the tag name as the selector to select elements with that specific tag.

```
const elements = document.querySelectorAll('div');
```

- **By class name:** Use the class name preceded by a dot to select elements with that specific class.

```
const elements = document.querySelectorAll('.myClass');
```

- **By ID:** Use the ID preceded by a hash symbol to select the element with that specific ID. Note that this will return a collection with a single element since IDs should be unique.

```
const elements = document.querySelectorAll('#myId');
```

- **By attribute:** Use attribute selectors to select elements based on attribute values.

```
const elements = document.querySelectorAll('[data-attribute="value"]');
```

#### ● JavaScript HTML styles

JavaScript HTML styles refer to the ability to manipulate and control the visual appearance of HTML elements using JavaScript. With JavaScript, you can dynamically modify the styles of HTML elements, including properties like color, size, position, visibility, and more. This allows you to create interactive and responsive web pages by changing the presentation of elements based on user actions, events, or other conditions.

#### ● Animation

Animation refers to the process of creating the illusion of motion or change over time by displaying a sequence of static images or frames in rapid succession. In the context of web development, animation is commonly used to enhance user interfaces, provide visual feedback, and create engaging user experiences.

In web development, there are different techniques and technologies available to create animations:

## **CSS Animations:**

CSS animations allow you to animate HTML elements using CSS properties and keyframes. With CSS animations, you can define the animation duration, timing function, and keyframe breakpoints to control the animation's behavior.

CSS animations can be triggered by adding or removing CSS classes, using JavaScript to modify CSS properties, or using CSS pseudo-classes like `:hover` and `:focus`.

You can use JavaScript to create a complex animation having, but not limited to, the following elements:

- ✓ Fireworks
- ✓ Fade Effect
- ✓ Roll-in or Roll-out
- ✓ Page-in or Page-out
- ✓ Object movements

JavaScript provides the following two functions to be frequently used in animation programs.

- ❖ `setTimeout(function, duration)` – This function calls function after duration milliseconds from now.
- ❖ `setInterval(function, duration)` – This function calls function after every duration milliseconds.
- ❖ `clearTimeout(setTimeout_variable)` – This function clears any timer set by the `setTimeout()` functions.

### **● Transition**

A **transition** is a change from one thing to the next, either in action or state of being.

In JavaScript, a transition in HTML styles refers to the process of smoothly animating changes to CSS properties of an HTML element over a specified duration. It involves gradually modifying the values of CSS properties to create a visually appealing and smooth visual effect. JavaScript allows you to control and trigger transitions dynamically based on user interactions or other programmatic events.

**Here's a breakdown of how transitions work in JavaScript HTML styles:**

**1. Select the element:**

- Use JavaScript methods like `'querySelector()'` or `'getElementById()'` to select the HTML element to which you want to apply the transition.

**2. Modify the CSS properties:**

- Use JavaScript to modify the CSS properties of the selected element over time.
- Update the CSS properties gradually by changing their values in small increments at regular intervals.

**3. Specify the transition properties:**

- Define the duration of the transition, which determines how long the transition should take to complete.

- Choose a timing function to control the pace of the transition. Timing functions like linear, ease-in, ease-out, and ease-in-out affect the speed of the transition.

#### **Difference between transitions and animations:**

Transitions	Animations
Transitions cannot loop (You can make them do that but they are not designed for that).	Animations have no problem in looping.
Transitions need a trigger to run like mouse hover.	The animation just starts. They don't need any kind of external trigger source.
Transitions are easy to work in JavaScript.	The animations are hard to work in JavaScript. The syntax for manipulating a keyframe and assigning a new value to it, is very complex.
Transitions animate a object from one point to another.	Animation allows you to define Keyframes which varies from one state to another with various properties and time frame.
Use transition for manipulating the value using JavaScript.	Flexibility is provided by having multiple keyframes and easy loop.

#### **• Slide show**

A web slideshow is a sequence of images or text that consists of showing one element of the sequence in a certain time interval.

In JavaScript HTML styles, a slideshow refers to a dynamic presentation of a series of images or content that is displayed sequentially, typically with a transition effect between each slide. Slideshows are commonly used to showcase images, highlight product features, or present content in a visually appealing and interactive manner.

**Here's an overview of how to create a slideshow using JavaScript HTML styles:**

#### **1. HTML Structure:**

- Create a container element in your HTML markup to hold the slideshow.
- Inside the container, create individual slide elements, such as <div> or <img>, for each slide in the slideshow.

#### **2. CSS Styling:**

- Apply CSS styles to position and style the slideshow container and slide elements.

- Use CSS properties like `position`, `display`, `width`, and `height` to control the layout and appearance of the slideshow.



### Practical Activity 2.7.2: Apply JavaScript in HTML



#### Task:

- 1: Referring to the previous activity (2.7.1) you are requested to go to the computer lab to open the computers, then the IDE and use Javascript in HTML in javascript program. This task should be done individually.
- 2: Read the key reading 2.7.2 in trainee manual about application of arrays in JavaScript program.
- 3: Referring to the description and steps provided in the key reading 2.7.2, apply javascript in HTML.
- 4: Ask questions for more clarification where necessary.



### Key readings 2.7.2

#### Here are the steps to apply JavaScript in HTML:

1. Create an HTML file: Start by creating a new HTML file or opening an existing one in a text editor.
2. Add the script tag: Inside the HTML file, within the `` or `` section, add the ``. The external JavaScript file should contain the JavaScript code you want to execute.
5. Place JavaScript code in the appropriate location: Depending on the purpose of your JavaScript code, place it in the appropriate location within the HTML file. For example, if

you want to manipulate the DOM, you might place the code inside a function that is called when the page loads or in an event handler that triggers the code execution.

6. Test and debug: Save the HTML file and open it in a web browser. Use the browser's developer tools to check for any JavaScript errors in the console and debug your code if necessary.

7. Update HTML elements with JavaScript: Use JavaScript to select HTML elements using methods like `querySelector()` or `getElementById()`. Then, manipulate the elements by changing their content, attributes, styles, or other properties using JavaScript code.

8. Handle events: Use JavaScript to add event listeners to HTML elements. This allows you to respond to user interactions such as clicks, mouse movements, or form submissions. Inside the event handler functions, write the JavaScript code that should be executed when the event occurs.

9. Interact with APIs (optional): Use JavaScript to interact with external APIs by making HTTP requests, fetching data, and processing responses. This can involve using JavaScript methods like `fetch()` or utilizing AJAX techniques.

10. Save and deploy: Once you are satisfied with your JavaScript code, save the HTML file and any linked JavaScript files. Deploy the HTML file and associated files to a web server or hosting platform to make it accessible on the internet.

By following these steps, you can effectively apply JavaScript code within an HTML file to add interactivity, manipulate

✓ **JavaScript HTML event listener**

To apply JavaScript event listeners in HTML, follow these steps:

1. Select the HTML element: Use JavaScript to select the HTML element to which you want to attach the event listener. You can use methods like `querySelector()`, `getElementById()`, or `getElementsByClassName()` to select the element(s) based on their CSS selector or ID.

2. Define the event listener function: Create a JavaScript function that will be executed when the event occurs. This function will contain the code that you want to run in response to the event.

3. Attach the event listener: Use the `addEventListener()` method to attach the event listener to the selected HTML element. This method takes two arguments: the event type and the event listener function.

4. Specify the event type: Choose the appropriate event type based on the user interaction you want to respond to. Common event types include "click", "mouseover", "submit", "keydown", and many more. You can find a comprehensive list of event types in the JavaScript documentation.

5. Handle the event: Inside the event listener function, write the JavaScript code that should be executed when the event occurs. This code can manipulate the DOM, modify CSS styles, interact with APIs, or perform any other desired actions.

**Here's an example that demonstrates applying an event listener to a button element:**

HTML:

```
<button id="myButton">Click Me</button>
```

JavaScript:

```
// Step 1: Select the button element
const button = document.getElementById("myButton");
// Step 2: Define the event listener function
function handleClick(event) {
    // Step 5: Handle the event
    console.log("Button clicked!");
}
// Step 3: Attach the event listener
button.addEventListener("click", handleClick);
```

In this example, we select the button element using `getElementById()`, define the event listener function `handleClick()`, and attach the event listener using `addEventListener()`. When the button is clicked, the `handleClick()` function is executed, and it logs a message to the console.

**Example:**

Add an event listener that fires when a user clicks a button:

```
document.getElementById("myBtn").addEventListener("click", displayDate);
```

### Add an Event Handler to an Element

Example

Alert "Hello World!" when the user clicks on an element:

```
element.addEventListener("click", function(){ alert("Hello World!"); });
```

**You can also refer to an external "named" function:**

Example

Alert "Hello World!" when the user clicks on an element:

```
element.addEventListener("click", myFunction);
function myFunction() {
    alert ("Hello World!");
}
```

### Add Many Event Handlers to the Same Element

#### Example

```
element.addEventListener("click", myFunction);
element.addEventListener("click", mySecondFunction);
```

You can add events of different types to the same element: Example

```
element.addEventListener("mouseover", myFunction);
element.addEventListener("click", mySecondFunction);
element.addEventListener("mouseout", myThirdFunction);
```

### Add an Event Handler to the window Object

#### Example

Add an event listener that fires when a user resizes the window:

```
window.addEventListener("resize", function(){
    document.getElementById("demo").innerHTML = sometext;
});
```

#### Passing Parameters

#### Example

```
element.addEventListener("click", function(){ myFunction(p1, p2); });
```

### Event Bubbling or Event Capturing?

```
addEventListener(event, function, useCapture);
```

#### Example

```
document.getElementById("myP").addEventListener("click",
    myFunction, true);
document.getElementById("myDiv").addEventListener("click", myFunction, true);
```

### The removeEventListener() method

#### Example

```
element.removeEventListener("mousemove", myFunction);
```

#### ✓ Window Object

To apply the Window object in JavaScript, follow these steps:

1. Access the Window object: The Window object is the global object in the browser environment, and it is automatically available in JavaScript without the need for any special setup. You can directly access the Window object and its properties and methods.

2. Use Window methods and properties: The Window object provides various methods and properties that allow you to interact with the browser window and its content.

3. Interact with the Window object: You can interact with the Window object by calling its methods or accessing its properties. For example:

```
// Display an alert dialog  
window.alert("Hello, World!");  
  
// Prompt the user for input  
const name = window.prompt("Please enter your name:");  
  
// Open a new window  
window.open("https://www.example.com");  
  
// Close the current window  
window.close();
```

You can also access the Window object implicitly without using the `window` prefix, as it is the default global object in the browser environment. For example:

```
// Access the location property  
console.log(location.href);
```

4. Handle window events: The Window object also provides event handling capabilities. You can attach event listeners to respond to various events that occur in the browser window, such as page load, resize, scroll, and more.

**For example:**

```
// Attach an event listener for the page load event  
window.addEventListener("load", function() {  
    console.log("Page loaded!");  
  
});
```

This code attaches an event listener to the `load` event of the Window object, which executes the provided function when the page finishes loading.

By following these steps, you can effectively utilize the Window object in JavaScript to interact with the browser window, display dialogs, navigate to URLs, access the HTML document, and handle window events.

- **Methods**

**To apply methods in JavaScript, follow these steps:**

1. Identify the object: Determine the object on which you want to apply the method. It can be a built-in JavaScript object or a custom object that you have defined.

2. Access the object: To access the object, create a new instance of it or reference an existing object.

3. Call the method: Use the dot notation to call the method on the object. The dot notation is used to access properties and methods of an object. For example, `object.method()`.
4. Pass arguments (if required): Some methods may require additional information to be passed as arguments. Check the method's documentation or specification to understand the required arguments and their order. Provide the necessary arguments when calling the method.
5. Capture the return value (if applicable): Methods can return values. If the method you are using returns a value, capture and store it in a variable for further use.

**Here's an example that demonstrates using the `alert()` method:**

```
// Step 1: Identify the object (window)  
// Step 2: Access the object (window)  
// Step 3: Call the method (alert)  
window.alert("Hello, World!");  
// Step 4: Pass arguments (message to be displayed)
```

When this code runs, it will display an alert dialog box in the browser window with the message "Hello, World!".

The `alert()` method is commonly used to provide important information or notifications to the user. It halts the execution of the JavaScript code until the user closes the alert dialog by clicking the OK button.

#### ❖ **alert()**

Here's an example of using the `alert()` method of the Window object to display an alert dialog box with a message:

```
// Display an alert dialog  
  
alert("Hello, World!");
```

When this code runs, it will display an alert dialog box with the message "Hello, World!" in the browser window.

The `alert()` method is commonly used to provide important information or notifications to the user. It halts the execution of the JavaScript code until the user closes the alert dialog by clicking the OK button.

#### ● **setInterval()**

**Here's an example of using the `setInterval()` method in JavaScript:**

```
// Define a function to be executed repeatedly  
function sayHello() {  
    console.log("Hello, World!");  
}  
// Call the function every 1 second using setInterval
```

```
setInterval(sayHello, 1000);
```

In this example, the `setInterval()` method is used to repeatedly execute the `sayHello()` function every 1 second (1000 milliseconds). The `sayHello()` function simply logs the message "Hello, World!" to the console.

The `setInterval()` method takes two arguments: the function to be executed and the time interval in milliseconds. In this case, we pass the `sayHello` function as the first argument and `1000` as the second argument to specify the interval of 1 second.

As a result, the message "Hello, World!" will be logged to the console every second until you stop the interval.

`setInterval()` is commonly used for tasks that need to be repeated at regular intervals, such as updating a clock, fetching data from a server, or animating elements on a web page. Remember to clear the interval using `clearInterval()` when you want to stop the execution.

**Example2:** `setInterval(function () {element.innerHTML += "Hello"}, 1000);`

- **ClearInterval()**

**Here's an example that demonstrates how to use `clearInterval()`:**

```
// Define a function to be executed repeatedly
function sayHello() {
  console.log("Hello, World!");
}

// Call the function every 1 second using setInterval
const intervalId = setInterval(sayHello, 1000);

// Stop the interval after 5 seconds using clearInterval
setTimeout(function() {
  clearInterval(intervalId);
  console.log("Interval stopped.");
}, 5000);
```

In this example, we first define the `sayHello()` function, which logs the message "Hello, World!" to the console.

Then, we use the `setInterval()` method to call the `sayHello()` function every 1 second. The `setInterval()` method returns an interval ID, which is stored in the `intervalId` variable.

Next, we use the `setTimeout()` method to schedule the execution of another function after 5 seconds. Inside that function, we call `clearInterval(intervalId)` to stop the interval execution. We also log a message to the console indicating that the interval has been stopped.

### **Example: 2**

```
<!DOCTYPE html>
<html>
<body>
<h1>The Window Object</h1>
<h2>The setInterval() and clearInterval() Methods</h2>
<p>In this example, the setInterval() method executes the setColor() function once every 500 milliseconds to toggle between two background colours.</p>
<button onclick="stopColor()">Stop Toggling</button>
<script> myInterval = setInterval(setColor, 500); function setColor() { let x =
document.body;
x.style.backgroundColor = x.style.backgroundColor == "yellow" ? "pink" : "yellow";
}
function stopColor() { clearInterval(myInterval);
}
</script>
</body>
</html>
```

- **setTimeout()**

Example: Display an alert box after 3 seconds (3000 milliseconds):

```
let timeout;
function myFunction() { timeout = setTimeout(alertFunc, 3000);
}
function alertFunc() { alert("Hello!");
}
```

- **clearTimeout()**

Example: How to prevent myGreeting() to execute:

```
const myTimeout = setTimeout(myGreeting, 3000);
function myGreeting()
{
document.getElementById("demo").innerHTML = "Happy Birthday to You !"
}
function myStopFunction()
{
clearTimeout(myTimeout);
}
```

- **open()**

Example: Open an about:blank page in a new window/tab:

```
var myWindow = window.open("", "", "width=200,height=100");
confirm()
```

**Example:** Confirmation box with line-breaks:

```
confirm("Press a button!\nEither OK or Cancel.");
```

- **close()**

**Here's an example that demonstrates how to use the `close()` method:**

```
// Close the current window  
window.close();
```

In this example, the `close()` method is called on the `window` object. When this code runs, it will close the current window or tab.

Please note that the `close()` method has certain restrictions due to security reasons. It can only be called on windows or tabs that were opened by a script using the `window.open()` method. If the window or tab was not opened by a script, attempting to close it using `window.close()` may not work or may prompt a confirmation dialog to the user.

Also, some modern browsers may prevent the use of `window.close()` if it is not triggered by a user action, such as a button click. This is to prevent malicious websites from automatically closing browser windows without user consent.

**Example:2**

```
function openWin() {  
    myWindow = window.open("https://www.w3schools.com", "_blank", "width=200,  
    height=100");  
}  
function closeWin() { myWindow.close();  
}
```

- **stop()**

**Example:**

```
<!DOCTYPE html>  
<html>  
<script>  
window.stop();  
</script>  
<body>  
<h1>The Window Object</h1>  
<h2>The stop() Method</h2>  
<p>The stop() method stops this document from loading.</p>  
</body>  
</html>
```

- **print()**

**Example:** Print the current page:

```
window.print();
```

### ✓ JavaScript form validation

To apply JavaScript form validation, follow these steps:

1. Access the form: Use the appropriate method to access the HTML form element in JavaScript. You can use methods like `getElementById()`, `querySelector()`, or `querySelectorAll()` to select the form element.
2. Add an event listener: Attach an event listener to the form element to listen for form submissions or specific events like `submit` or `click`. This will allow you to trigger the validation function when the form is submitted or a specific event occurs.
3. Create a validation function: Write a JavaScript function that will handle the form validation. This function should be called when the form is submitted or the desired event occurs. Inside the validation function, you can access the form elements and perform the necessary validation checks.
4. Perform validation checks: Within the validation function, access the form elements using their names, IDs, or classes. Use JavaScript methods and properties to perform the desired validation checks on the form inputs. Common validation checks include checking for empty fields, validating email addresses, verifying password strength, or validating numeric input.
5. Display error messages: If any validation checks fail, display error messages to the user. This can be done by adding error messages to the HTML document, modifying the CSS styles of the form elements, or using JavaScript methods like `alert()` or `console.log()` to display the error messages.
6. Prevent form submission (optional): If you want to prevent the form from being submitted when validation fails, use the `event.preventDefault()` method within the validation function. This will stop the form submission and allow you to display error messages or prompt the user to correct the input.

Here's a simplified example that demonstrates JavaScript form validation:

```
<form id="myForm">
  <input type="text" id="name" required>
  <input type="email" id="email" required>
  <button type="submit">Submit</button>
</form>
<script>
  const form = document.getElementById("myForm");
  form.addEventListener("submit", function(event) {
    event.preventDefault(); // Prevent form submission
    // Perform form validation
  })
</script>
```

```

const nameInput = document.getElementById("name");
const emailInput = document.getElementById("email");
if (nameInput.value === "") {
  alert("Please enter your name.");
  return;
}
if (emailInput.value === "") {
  alert("Please enter your email address.");
  return;
}
// If validation passes, submit the form
form.submit();
});
</script>"""

```

In this example, we access the form element using `getElementById()` and attach a `submit` event listener to it. When the form is submitted, the validation function is called.

#### **Here is code example 2 for validating form data:**

```

<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>JavaScript Form validation</title>
<link rel="stylesheet" href="/examples/css/form-style.css">
<script>
// Defining a function to display error message
function printError(elemId, hintMsg) {
  document.getElementById(elemId).innerHTML = hintMsg;
}
// Defining a function to validate form function validateForm() {
  // Retrieving the values of form elements var name =
document.contactForm.name.value;  var email = document.contactForm.email.value;
var mobile = document.contactForm.mobile.value;  var country =
document.contactForm.country.value;  var gender =
document.contactForm.gender.value;  var hobbies = [];
var checkboxes = document.getElementsByName("hobbies[]");
for(var i=0; i < checkboxes.length; i++) {

```

```
if(checkboxes[i].checked) {
    // Populate hobbies array with selected values
    hobbies.push(checkboxes[i].value);
}
}

// Defining error variables with a default value
var nameErr = emailErr = mobileErr = countryErr = genderErr = true;
// Validate name
if(name == "") {
    printError("nameErr", "Please enter your name");
} else {
    var regex = /^[a-zA-Z\s]+$/;
    if(regex.test(name) === false) {
        printError("nameErr", "Please enter a valid name");
    } else {
        printError("nameErr", "");
        nameErr = false;
    }
}

// Validate email address
if(email == "") {
    printError("emailErr", "Please enter your email address");
} else {
    // Regular expression for basic email validation
    var regex = /^[\w\.-]+\@[^\w\.-]+\.\w+$/;
    if(regex.test(email) === false) {
        printError("emailErr", "Please enter a valid email address");
    } else{
        printError("emailErr", "");      emailErr = false;
    }
}

// Validate mobile number
if(mobile == "") {
    printError("mobileErr", "Please enter your mobile number");
} else {
    var regex = /^[1-9]\d{9}$/;
    if(regex.test(mobile) === false) {
        printError("mobileErr", "Please enter a valid 10 digit mobile number");
    } else{
        printError("mobileErr", "");      mobileErr = false;
    }
}
```

```

}

}

// Validate country
if(country == "Select") {
printError("countryErr", "Please select your country");
} else {
printError("countryErr", "");
countryErr = false;
}
// Validate gender
if(gender == "") {
printError("genderErr", "Please select your gender");
} else {
printError("genderErr", "");
genderErr = false;
}
// Prevent the form from being submitted if there are any errors
if((nameErr || emailErr || mobileErr || countryErr || genderErr) == true) {
return false;
} else {
// Creating a string from input data for preview
var dataPreview = "You've entered the following details: \n" + "Full Name: " + name +
"\n" +
"Email Address: " + email + "\n" +
"Mobile Number: " + mobile + "\n" +
"Country: " + country + "\n" + "Gender: " + gender + "\n";
if(hobbies.length) {
dataPreview += "Hobbies: " + hobbies.join(", ");
}
// Display input data in a dialog box before submitting the form
alert(dataPreview);
}
</script>
</head>
<body>
<form name="contactForm" onsubmit="return validateForm()" action="/examples/actions/confirmation.php" method="post">
<h2>Application Form</h2>
<div class="row">
<label>Full Name</label>

```

```
<input type="text" name="name">
<div class="error" id="nameErr"></div>
</div>
<div class="row">
<label>Email Address</label>
<input type="text" name="email">
<div class="error" id="emailErr"></div>
</div>
<div class="row">
<label>Mobile Number</label>
<input type="text" name="mobile" maxlength="10">
<div class="error" id="mobileErr"></div>
</div>
<div class="row">
<label>Country</label>
<select name="country">
<option>Select</option>
<option>Australia</option>
<option>India</option>
<option>United States</option>
<option>United Kingdom</option>
</select>
<div class="error" id="countryErr"></div>
</div>
<div class="row">
<label>Gender</label>
<div class="form-inline">
<label><input type="radio" name="gender" value="male"> Male</label>
<label><input type="radio" name="gender" value="female"> Female</label></div>
<div class="error" id="genderErr"></div>
</div>
<div class="row">
<label>Hobbies <i>(Optional)</i></label>
<div class="form-inline">
<label><input type="checkbox" name="hobbies[]" value="sports"> Sports</label>
<label><input type="checkbox" name="hobbies[]" value="movies"> Movies</label>
<label><input type="checkbox" name="hobbies[]" value="music"> Music</label></div>
</div>
<div class="row">
<input type="submit" value="Submit">
```

```

</div>
</form>
</body>
</html>

Here is scripting code example for validating URL:
<h1 style="color:green;">
GeeksForGeeks
</h1>
<p id="GFG_UP" style="font-size: 15px; font-weight: bold;">
</p>
<button onclick="gfg_Run()">
click here
</button>
<p id="GFG_DOWN" style="color:green;
font-size: 20px;
font-weight: bold;">
</p>
<script>
var el_up = document.getElementById("GFG_UP");
var el_down = document.getElementById("GFG_DOWN");
var expression = /(https?:\/\/(?:www\.|(?!\www))[a-zA-Z0-9][a-zA-Z0-9-]+[a-zA-Z0-9]\.[^\s]{2,}|www\.[a-zA-Z0-9][azA-Z0-9-]+[a-zA-Z0-9]\.[^\s]{2,}|https?:\/\/(?:www\.|(?!\www))[a-zA-Z0-9]+\.[^\s]{2,}|www\.[a-zA-Z0-9]+\.[^\s]{2,})/gi;
var regex = new RegExp(expression);
var url =
'www.geekforgeeks.org';
el_up.innerHTML = "URL = " + url + "";
function gfg_Run() {
    var res = "";
    if (url.match(regex)) {
        res = "Valid URL";
    } else {
        res = "Invalid URL";
    }
    el_down.innerHTML = res;
}
</script>

```

✓ **Apply Canvas**

**To apply Canvas element in JavaScript, follow these steps:**

1. Create a Canvas element: In your HTML file, create a `<canvas>` element and give it an `id` attribute to uniquely identify it. For example: `<canvas id="myCanvas"></canvas>`
2. Access the Canvas element: In your JavaScript code, use the appropriate method (such as `getElementById()` or `querySelector()`) to access the Canvas element and store it in a variable.
3. Get the Canvas context: Use the `getContext()` method on the Canvas element to get the rendering context. The rendering context allows you to draw on the Canvas. The most commonly used context is the 2D context, which is obtained by passing the string `"2d"` as an argument to `getContext()`. Store the context in a variable.
4. Draw on the Canvas: Use the methods and properties of the `CanvasRenderingContext2D` object to draw shapes, lines, text, and images on the Canvas. Some commonly used methods include `fillRect()`, `strokeRect()`, `fillText()`, `lineTo()`, `arc()`, and `drawImage()`. Refer to the Canvas API documentation for a complete list of available methods.
5. Customize and style the drawings: Use the various properties and methods of the `CanvasRenderingContext2D` object to customize the appearance of the drawings, such as setting the stroke and fill colors, line widths, font styles, and more.

**Here's a simple example that demonstrates the basic steps of using the Canvas element in JavaScript:**

```
<canvas id="myCanvas"></canvas>
<script>
const canvas = document.getElementById("myCanvas");
const context = canvas.getContext("2d");
// Draw a rectangle
context.fillStyle = "red";
context.fillRect(50, 50, 100, 100);

// Draw a line
context.strokeStyle = "blue";
context.lineWidth = 5;
context.beginPath();
context.moveTo(200, 50);
context.lineTo(200, 150);
context.stroke();
// Draw text
context.font = "20px Arial";
context.fillStyle = "green";
context.fillText("Hello, Canvas!", 250, 100);
```

```
</script>
```

In this example, we first access the Canvas element with the id ``myCanvas`` using `getElementById()` and store it in the `canvas` variable.

Then, we get the 2D rendering context by calling `getContext("2d")` on the `canvas` variable and store it in the `context` variable.

Next, we use various methods and properties of the `context` object to draw a red rectangle, a blue line, and green text on the Canvas.

- **Draw a Line**

```
<script>  
var c = document.getElementById("myCanvas");  
var ctx = c.getContext("2d");  
ctx.moveTo(0, 0);  
ctx.lineTo(200,100);  
ctx.stroke();  
</script>
```

- **Draw a Circle**

```
<script>  
var c = document.getElementById("myCanvas");  
var ctx = c.getContext("2d");  
ctx.beginPath();  
ctx.arc(95, 50, 40, 0, 2 * Math.PI); ctx.stroke();  
</script>
```

- **Draw a Text**

```
<script>  
var c = document.getElementById("myCanvas");  
var ctx = c.getContext("2d");  
ctx.font = "30px Arial";  
ctx.fillText("Hello World", 10, 50);  
</script>
```

- **Stroke Text**

```
<script>  
var c = document.getElementById("myCanvas");  
var ctx = c.getContext("2d");  
ctx.font = "30px Arial";  
ctx.strokeText("Hello World", 10, 50);  
</script>
```

- **Draw Linear Gradient**

```
<script>  
var c = document.getElementById("myCanvas");  
var ctx = c.getContext("2d");  
// Create gradient  
var grd = ctx.createLinearGradient(0, 0, 200, 0);  
grd.addColorStop(0, "red");  
grd.addColorStop(1, "white");  
// Fill with gradient  
ctx.fillStyle = grd; ctx.fillRect(10, 10, 150, 80); </script>
```

- **Draw Circular Gradient**

```
<script>  
var c = document.getElementById("myCanvas");  
var ctx = c.getContext("2d");  
  
// Create gradient  
var grd = ctx.createRadialGradient(75, 50, 5, 90, 60, 100);  
grd.addColorStop(0, "red");  
grd.addColorStop(1, "white");  
  
// Fill with gradient  
ctx.fillStyle = grd;  
ctx.fillRect(10, 10, 150, 80);  
</script>
```

- **Draw Image**

```
<script>  
var c = document.getElementById("myCanvas");  
var ctx = c.getContext("2d");  
var img = document.getElementById("scream"); ctx.drawImage(img, 10, 10);  
</script>
```

## JavaScript HTML DOM

The Document Object Model (DOM) is the data representation of the objects that comprise the structure and content of a document on the web.

- **innerHTML**

The innerHTML property sets or returns the HTML content (inner HTML) of an element.

### Example

**Get the HTML content of an element with id="myP":**

```
let html = document.getElementById("myP").innerHTML;  
Change the HTML content of an element with id="demo":  
document.getElementById("demo").innerHTML = "I have changed!";
```

- **getElementById**

**Example**

Get the element with the specified id:

```
document.getElementById("demo");
```

Get the element and change its color:

```
const myElement = document.getElementById("demo");  
myElement.style.color = "red";
```

Or just change its color:

```
document.getElementById("demo").style.color = "red";
```

- **getElementsByClassName**

**Example**

Get all elements with class="example":

```
const collection = document.getElementsByClassName("example");
```

Get all elements with both the "example" and "color" classes:

```
const collection = document.getElementsByClassName("example color");
```

- **getElementsByName**

**Example**

Get all elements with the name "fname":

```
let elements = document.getElementsByName("fname");
```

Number of elements with name="animal":

```
let num = document.getElementsByName("animal").length;
```

- **getElementsByTagName**

**Example**

Get all elements with the tag name "li":

```
const collection = document.getElementsByTagName("li");
```

Get all elements in the document:

```
const collection = document.getElementsByTagName("*");
```

Change the inner HTML of the first element in the document:

```
document.getElementsByTagName("p")[0].innerHTML = "Hello World!";
```

- **querySelector**

**Examples**

Get the first <p> element:

```
document.querySelector("p");
```

Get the first element with class="example":

```
document.querySelector(".example");
```

- **querySelectorAll**

### Example

Select all elements with class="example":

```
document.querySelectorAll(".example");
```

- ✓ **JavaScript HTML styles**

- **Animation**

Here is code example:

```
<html>
<head>
<title>JavaScript Animation</title>
<script type = "text/javascript">
<!--
var imgObj = null;
var animate ;
function init() {
imgObj = document.getElementById('myImage');
imgObj.style.position= 'relative';
imgObj.style.left = '0px';
}
function moveRight() {
imgObj.style.left = parseInt(imgObj.style.left) + 10 + 'px';
animate = setTimeout(moveRight,20); // call moveRight in 20msec
}
function stop() {
clearTimeout(animate);
imgObj.style.left = '0px';
}
window.onload = init;
//-->
</script>
</head>
<body>
<form>
<img id = "myImage" src = "/images/html.gif" />
<p>Click the buttons below to handle animation</p>
<input type = "button" value = "Start" onclick = "moveRight();"/>
```

```
<input type = "button" value = "Stop" onclick = "stop();" />
</form>
</body>
</html>
```

- **Transition**

A **transition** is a change from one thing to the next, either in action or state of being.

```
<!DOCTYPE html>
<html>
<head>
<style>
#myDIV {
    border: 1px solid black;
    background-color: lightblue;
    width: 270px;
    height: 200px;
    overflow: auto;
}
#myDIV:hover {
    background-color: coral;
    width: 570px;
    height: 500px;
    padding: 100px;
    border-radius: 50px;
}
</style>
</head>
<body>
<p>Mouse over the DIV element and it will change, both in color and size!</p>
<p>Click the "Try it" button and mouse over the DIV element again. The change will now happen gradually, like an animation:</p>
<button onclick="myFunction()">Try it</button>
<div id="myDIV">
    <h1>myDIV</h1>
</div>
<script>
function myFunction() {
    document.getElementById("myDIV").style.transition = "all 2s";
}
</script>
</body>
```

```
</html>



- Slide show



Simple code example:



```
<!DOCTYPE html>
<html>
<head>
<title>JavaScript Slideshow Demo</title>
</head>
<body>
<div>
<img id="image1" />
</div>
<script>
var imgArray = [
'images/image1.jpg',
'images/image2.jpg',
'images/image3.jpg'
];
var curIndex = 0;
var imgDuration = 5000;
function slideShow() {
document.getElementById('image1').src = imgArray[curIndex];
curIndex++;
if (curIndex == imgArray.length) { curIndex = 0; }
setTimeout("slideShow()", imgDuration);
}
slideShow();
</script>
</body>
</html>
```

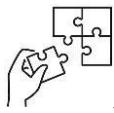

```



### Points to Remember

- HTML events in JavaScript refer to actions or occurrences that can be detected and responded to by JavaScript code within an HTML document. Events in JavaScript play a crucial role in creating interactive and dynamic web applications. The `window` object is a fundamental part of JavaScript in a browser environment which provides a gateway to interact with the browser window, access its **properties** and **method**.

- The `<canvas>` element in JavaScript provides a versatile and powerful platform for creating and manipulating graphical content dynamically. With its drawing context and various **drawing operations**, you can create **Gradients**, render images, and design interactive visual elements on the web.
- By implementing JavaScript form validation, you can enhance the user experience by ensuring that the data submitted through HTML forms meets the required criteria.



### **Application of learning 2.7.**

Build a javascript project to create an age calculator. From the final version of the project, create a container with the title age calculator with an input of a date. If the user clicks on the date input, he/she can choose the date of their birthday. For example, if the user chooses a date in 2002 and clicks on Calculate Age, he/she can see the age is calculated based on this date and saying your age is 21 years old.



## Indicative content 2.8: Applying regular expressions



Duration: 6 hrs



### Theoretical Activity 2.8.1: Description of regular expressions in JavaScript



#### Tasks:

- 1: In small groups, you are requested to answer the following questions related to the description of regular expression in javascript:
  - i . Explain the term “regular expression”
  - ii. Describe the following terms used in regular expression:
    - a. Modifiers
    - b. Groups
    - c. Metacharacters
    - d. Quantifiers
- 2: Present your findings to your trainer and classmates
- 3: Discuss on provided findings and choose the correct ones.
- 4: For more clarification, read the key readings 2.8.1 and ask questions where necessary.



#### Key readings 2.8.1.

Regular expressions are patterns used to match character combinations in strings.

##### • Modifiers

The JavaScript regular expression modifiers are optional parts of a regular expression and allow us to perform case insensitive and global searchers.

The modifiers can also be combined together.

Following are the modifiers

| Modifier | Description   |
|----------|---|
| G        | It enables global matching and returns all the matched results instead of stopping at first match |
| I        | It enables case insensitive matching  |
| M        | It enables multiline matching   |

#### Brackets

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                      |

- **Group**

What is Group in Regex? A group is a part of a regex pattern enclosed in parentheses () metacharacter. We create a group by placing the regex pattern inside the set of parentheses ( and ).

For example, the regular expression (cat) creates a single group containing the letters 'c', 'a', and 't'.

Parentheses () is used to group sub-patterns. For example, (a|b|c)xz match any string that matches either a or b or c followed by xz

| Expression | String       | Matched?                      |
|------------|--------------|-------------------------------|
|            | ab xz        | No match                      |
| (a b c)xz  | abxz         | 1 match (match at abxz)       |
|            | axz<br>cabxz | 2 matches (at axzbc<br>cabxz) |

## Metacharacters

Metacharacters are characters that are interpreted in a special way by a RegEx engine. Here's a list of metacharacters:

[] . ^ \$ \* + ? {} () \ |

### [] - Square brackets

Square brackets specify a set of characters you wish to match.

### . -Period

A period matches any single character (except newline '\n').

### ^ - Caret

The caret symbol ^ is used to check if a string **starts with** a certain character.

### \$ - Dollar

The dollar symbol \$ is used to check if a string **ends with** a certain character.

### \* - Star

The star symbol \* matches **zero or more occurrences** of the pattern left to it.

#### + - Plus

The plus symbol + matches one or more occurrences of the pattern left to it.

#### ? - Question Mark

The question mark symbol ? matches **zero or one occurrence** of the pattern left to it.

#### { } – Braces

Consider this code: {n,m}. This means at least n, and at most m repetitions of the pattern left to it.

#### | - Alternation

Vertical bar | is used for alternation (or operator).

#### ( ) – Group

Parentheses () is used to group sub-patterns. For example, (a|b|c)xz match any string that matches either a or b or c followed by xz

#### \ - Backslash

Backslash \ is used to escape various characters including all metacharacters. For example,

\\$a match if a string contains \$ followed by a. Here, \$ is not interpreted by a RegEx engine in a special way.

If you are unsure if a character has special meaning or not, you can put \ in front of it.

This makes sure the character is not treated in a special way.

#### Special Sequences

Special sequences make commonly used patterns easier to write. Here's a list of special sequences:

\A - Matches if the specified characters are at the start of a string.

\b - Matches if the specified characters are at the beginning or end of a word.

\B - Opposite of \b. Matches if the specified characters are not at the beginning or end of a word.

\d - Matches any decimal digit. Equivalent to [0-9]

\D - Matches any non-decimal digit. Equivalent to [^0-9]

\s - Matches where a string contains any whitespace character. Equivalent to [\t\n\r\f\v].

\S - Matches where a string contains any non-whitespace character. Equivalent to [^\t\n\r\f\v].

\w - Matches any alphanumeric character (digits and alphabets). Equivalent to [a-zA-Z0-9\_]. By the way, underscore \_ is also considered an alphanumeric character.

\W - Matches any non-alphanumeric character. Equivalent to [^a-zA-Z0-9\_]

\Z - Matches if the specified characters are at the end of a string.

- **Quantifiers**

Quantifiers specify how many instances of a character, group, or character class must be present in the input for a match to be found.

| Quantifier | Description  |
|------------|--|
| $n^+$      | Matches any string that contains at least one $n$                  |
| $n^*$      | Matches any string that contains zero or more occurrences of $n$   |
| $n^?$      | Matches any string that contains zero or one occurrences of $n$    |
| $n\{X\}$   | Matches any string that contains a sequence of $X$ $n$ 's          |
| $n\{X,Y\}$ | 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 $n$   |



### Practical Activity 2.8.2: Apply regular expressions in JavaScript



#### Task:

- 1: Referring to the previous theoretical activities (2.8.1) you are requested to go to the computer lab to apply regular expression in javascript. This task should be done individually.
- 2: Read the key reading 2.8.2 in trainee manual about application of regular expressions in JavaScript program.
- 3: Referring to the presented steps provided in the key reading 2.8.2, apply the regular expressions JavaScript program.
- 4: Present your work to the trainer and the whole class.
- 5: Ask questions for more clarification where necessary



### Key readings 2.8.2

Regular expressions are used with the RegExp methods test() and exec() and with the String methods match(), replace(), search(), and split().

| Method | Description |
|--------|-------------|
|        |             |

|                     |  |
|---------------------|--|
| <b>exec()</b>       | Executes a search for a match in a string. It returns an array of information or null on a mismatch.             |
| <b>test()</b>       | Tests for a match in a string. It returns true or false.   |
| <b>match()</b>      | Returns an array containing all of the matches, including capturing groups, or null if no match is found.        |
| <b>matchAll()</b>   | Returns an iterator containing all of the matches, including capturing groups.                                   |
| <b>search()</b>     | Tests for a match in a string. It returns the index of the match, or -1 if the search fails.                     |
| <b>replace()</b>    | Executes a search for a match in a string, and replaces the matched substring with a replacement substring.      |
| <b>replaceAll()</b> | Executes a search for all matches in a string, and replaces the matched substrings with a replacement substring. |
| <b>split()</b>      | Uses a regular expression or a fixed string to break a string into an array of substrings.                       |

When you want to know whether a pattern is found in a string, use the `test()` or `search()` methods; for more information (but slower execution) use the `exec()` or `match()` methods.

If you use `exec()` or `match()` and if the match succeeds, these methods return an array and update properties of the associated regular expression object and also of the predefined regular expression object, `RegExp`.

If the match fails, the `exec()` method returns null (which coerces to false)

#### Using String Methods:

In JavaScript, regular expressions are often used with the two string methods: `search()` and `replace()`.

- The **search() method** uses an expression to search for a match and returns the position of the match.
- The **replace() method** returns a modified string where the pattern is replaced.

**Using String search() With a Regular Expression:** Use a regular expression to do a case-insensitive search for “GeeksforGeeks” in a string:

#### Example:

```
function myFunction() {  
    // input string  
    let str = "Visit geeksforGeeks!";  
    // searching string with modifier i  
    let n = str.search(/GeeksforGeeks/i);  
    console.log(n);  
    // searching string without modifier i  
    let n = str.search(/GeeksforGeeks/);  
    console.log(n);  
}  
myFunction();
```

**Output:**

6

-1

**Use String replace() With a Regular Expression :**

Use a case insensitive regular expression to replace gfG with GeeksforGeeks in a string:

**Example:**

```
function myFunction()  
{  
    // input string  
    let str = "Please visit gfG!";  
  
    // replacing with modifier i  
    let txt = str.replace(/gfG/i, "geeksforgeeks");  
    console.log(txt);  
}  
myFunction();
```

**Output:**

Please visit geeksforgeeks!

**Another example:**

```
<html>  
<body>  
<p id="demo"></p>  
<script>  
let text = "Visit W3Schools";  
let pattern = /w3schools/i;
```

```
let result = text.match(pattern);
document.getElementById("demo").innerHTML = result;
</script>
</body>
</html>
```

**Output:**

W3Schools

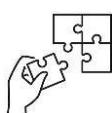
**Example explained:**

w3schools	The pattern to search for
/w3schools/	A regular expression
/w3schools/i	A case-insensitive regular expression



**Points to Remember**

- Regular expressions in JavaScript are a valuable tool for handling pattern matching and string manipulation tasks, providing efficient ways like Modifiers, Groups, Metacharacters, Quantifiers to work with text patterns. Regular expressions are used with the RegExp methods test() and exec() and with the String methods match(), replace(), search(), and split().
- When you want to know whether a pattern is found in a string, use the test() or search() methods; for more information (but slower execution) use the exec() or match() methods.



**Application of learning 2.8.**

You are developing a user registration form for a website, and you want to ensure that users enter valid information. You are tasked to use regular expressions in JavaScript to validate certain fields.



## Indicative content 2.9: Error handling



Duration: 4 hrs



### Theoretical Activity 2.9.1: identification of error handling in JavaScript



#### Tasks:

- 1: In small groups, you are requested to answer the following questions related to the error handling in javascript:
  - i. What do you understand about the term “error handling”
  - ii. What are different types of errors in JavaScript?
  - iii. Explain the difference between Try & catch in error handling.
  - iv. Explain the throw statement in error handling.
- 2: Provide the answer for the asked questions and write them on papers.
- 3: Present the findings/answers to the whole class.
- 4: Discuss on provided answers and choose the correct answer.
- 5: For more clarification, read the key readings 2.9.1 and ask questions where necessary.



#### Key readings 2.9.1.:

##### • Error handling

**Errors** are statements that don't let the program run properly.

JavaScript code can encounter different errors when it is executed. Errors can be caused by programming mistakes, incorrect input, or other unforeseeable events.

Javascript error handling is a strategy that handles the errors or exceptions which occur at runtime.

##### ✓ Types of error

The following are the 7 types of errors in JavaScript:

1. **Syntax error** - The error occurs when you use a predefined syntax incorrectly.
2. **Reference Error** - In a case where a variable reference can't be found or hasn't been declared, then a Reference error occurs.

3. **Type Error** - An error occurs when a value is used outside the scope of its data type.
4. **Evaluation Error** - Current JavaScript engines and EcmaScript specifications do not throw this error. However, it is still available for backward compatibility. The error is called when the eval() backward function is used, as shown in the following code block:
5. **RangeError** - There is an error when a range of expected values is required.
6. **URI Error** - When the wrong character(s) are used in a URI function, the error is called.
7. **Internal Error** - In the JS engine, this error occurs most often when there is too much data and the stack exceeds its critical size. When there are too many recursion patterns, switch cases, etc., the JS engine gets overwhelmed.

### ✓ Try & catch

The try statement allows you to define a block of code to be tested for errors while it is being executed.

The catch statement

allows you to define a block of code to be executed, if an error occurs in the try block.

The JavaScript statements try and catch come in pairs:

#### Syntax:

```
try {  
    Block of code to try  
}  
catch(err) {  
    Block of code to handle errors  
}
```

### ✓ Thrown

When an error occurs, JavaScript will normally stop and generate an error message.

The technical term for this is: JavaScript will throw an exception (throw an error).

JavaScript will actually create an Error object with two properties: name and message.

#### The throw Statement

The throw statement allows you to create a custom error.

Technically you can throw an exception (throw an error).

The exception can be a JavaScript String, a Number, a Boolean or an Object:

```
throw "Too big"; // throw a text  
throw 500;      // throw a number
```

If you use throw together with try and catch, you can control program flow and generate custom error messages.



### Practical Activity 2.9.2: handling error in JavaScript



#### Task:

- 1: Referring to the previous theoretical activity (2.9.1) you are requested to go to the computer lab. This task should be done individually.
- 2: Read the key reading 2.9.2 in trainee manual about error handling in JavaScript.
- 3: Referring to the key reading 2.9.2 in trainee manual, apply Try & catch and throw in javascript program.
- 4: Ask questions for more clarification where necessary.



### Key readings 2.9.2

#### • Error handling

Example of an error

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Error Handling</h2>
<p>This example demonstrates how to use <b>catch</b> </b> to display an error.</p>
<p id="demo"></p>
<script>
try {
  adddler("Welcome guest!");
}
catch(err) {
  document.getElementById("demo").innerHTML = err.message;
}
</script>
</body>
</html>
```

#### Output:

JavaScript Error Handling

This example demonstrates how to use catch to display an error.

addAlert is not defined

Note: JavaScript catches addAlert as an error, and executes the catch code to handle it.

#### ✓ Types of error

##### 1. Syntax error

```
const func = () =>
  console.log(hello)
}
```

**Output:**

```
}
```

```
^
```

SyntaxError: Unexpected token }

In the above example, an opening bracket is missing in the code, which invokes the Syntax error constructor.

##### 2. Reference Error

```
console.log(x);
```

**Output:**

```
console.log(x);
```

```
^
```

ReferenceError: x is not defined

##### 3. Type Error

```
let num = 15;
console.log(num.split(""))); //converts a number to an array
```

**Output:**

```
console.log(num.split(""))); //converts a number to an array
```

```
^
```

TypeError: num.split is not a function

##### 4. Evaluation Error

```
try{
  throw new EvalError("Throws an error")
} catch(error){
  console.log(error.name, error.message)
}
```

**Output:**

EvalError 'Throws an error'

##### 5. RangeError

```
const checkRange = (num)=>{
if (num < 30) throw new RangeError("Wrong number");
return true
}

checkRange(20);
```

**Output:**

```
if (num < 30) throw new RangeError("Wrong number");
^
RangeError: Wrong number
```

## 6. URI Error

```
console.log(decodeURI("https://www.educative.io/shoteditor"))
console.log(decodeURI("%sdfk"));
```

**Output:**

```
console.log(decodeURI("%sdfk"));

URIError: URI malformed
```

## 7. Internal Error

```
switch(condition) {
  case 1:
    break
  case 2:
    break
  case 3:
    break
  case 4:
    break
  case 5:
    break
  case 6:
```

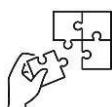
```
break  
case 7:  
  
break  
up to 500 cases  
}
```

**Output:** Its output will be like InternalError.



### Points to Remember

- **Errors** are statements that don't let the program run properly. JavaScript codes can encounter different errors when it is executed. These errors may be one of the followings: Syntax error, Reference Error, Type Error, Evaluation Error, Range Error, URI Error and/or Internal Error. Errors in programming can be caused by programming mistakes, incorrect input, or other unforeseeable events. The strategy that handles the errors or exceptions in JavaScript at runtime is known as error handling.
- The try statement allows you to define a block of code to be tested for errors while it is being executed. When an error occurs, JavaScript will normally stop and generate an error message.



### Application of learning 2.9.

you are developing a web application that allows users to submit a form with their personal information. The form includes fields like name, email, and age. You're using JavaScript for client-side validation, and you are tasked to implement error handling to provide a better user experience.



## Learning outcome 2 end assessment

### Theoretical assessment

1. The following statement are related to regular expressions as used in JavaScript program.

Read them carefully and state whether it is True or False.

- i. In JavaScript, there are three ways to write a string, they can be written inside single quotes (' '), double quotes (" "), or backticks (` `).
- ii. A named function definition executes automatically.
- iii. The asterisk quantifier (\*) specifies that zero or more of the preceding characters must match.

**The answer is:**

i. True

ii. False

iii. True

2. Analyse the JavaScript codes given below and choose the correct output from the results provided.

```
int a=1;  
if(a>10)  
{  
document.write(10);  
}  
else  
{  
document.write(a);  
}
```

- a) 10
- b) 0
- c) 1
- d) Undefined

The answer is c

3. Give the output of the following JavaScript codes.

```
<script type >var mango = new Object ();  
mango.color = "yellow";  
mango.price= 200;  
mango.sweetness = 8;  
mango.howSweetAmI = function (
```

```

{
document.write("Good Fruit");
}
document.write("color: " +mango.color)
document.write("price: " +mango. price)
document.write("<br>");
mango. howSweetAmI()
</script>

```

**Answer**



color: yellow price: 200  
Good Fruit

4 Match the following items of A column which corresponding to B column.

A	B
1. setTimeout()	A. This function can be used to call the function after each duration milliseconds.
2. setInterval ()	B. This function can be used to call the function after a millisecond delay.
3. clearTimeout()	C. This function can be used to clear the timer that has been set by the setTimeout()

**Answer**

1.....B

2.....A

3.....C

**Practical assessment**

1. By using any loop of your choice Write a JS code to print a pattern shown below.

```

*
* *
* * *
* * * *
* * * * *

```



## References

<http://support.kodable.com/en/articles/417331-what-are-loops>

<https://www.w3resource.com/javascript-exercises/javascript-conditional-statements-and-loops-exercises.php>

<https://gist.github.com/ourschar/f88cfcdc3367f79e5ec5e81f92e73ace>

<https://www.semrush.com/blog/javascript/>

<https://www.w3resource.com/javascript-exercises/error-handling/>

<https://www.w3resource.com/javascript-exercises/javascript-basic-exercises.php>

<https://launchschool.com/books/javascript/read/preparations>

## Learning Outcome 3: Apply JavaScript in Project



```
1 <div id="test">
2 </div>
3 <script>
4 var a = document.createElement('p');
5 a.innerHTML = 'Hello';
6 document.querySelector('#test').appendChild(a);
7 </script>
8
9
10
```

### **Indicative contents**

**3.1. Preparing project environment**

**3.2 Create pages with HTML**

**3.3 Apply CSS to HTML pages**

**3.4 Apply JavaScript**

### **Key Competencies for Learning Outcome 3: Apply JavaScript in project**

<b>Knowledge</b>	<b>Skills</b>	<b>Attitudes</b>
<ul style="list-style-type: none"><li>• Description of JavaScript project environment</li></ul>	<ul style="list-style-type: none"><li>• Preparation of project environment<ul style="list-style-type: none"><li>• Creating a project folder and file structures</li><li>• Creating of HTML pages</li><li>• Applying CSS files to HTML pages</li><li>• Applying JavaScript in a project</li></ul></li></ul>	<ul style="list-style-type: none"><li>• Being Problem solver</li><li>• Being Attentive</li><li>• Being confident</li><li>• Being a critical thinker</li><li>• Being analytical and details oriented</li><li>• Being Team worker</li></ul>



**Duration: 50 hrs**

**Learning outcome 3 objectives:**



By the end of the learning outcome, the trainees will be able to:

1. Prepare properly project environment according to the work to be done
2. Create correctly HTML pages based on project requirements
3. Manipulate properly CSS files within HTML pages
4. Apply effectively JavaScript functions in accordance with project requirements



**Resources**

<b>Equipment</b>	<b>Tools</b>	<b>Materials</b>
<ul style="list-style-type: none"><li>• Computer</li><li>• Projector</li></ul>	<ul style="list-style-type: none"><li>• Vs code</li><li>• Node</li><li>• Sublime</li><li>• Notepad++</li><li>• Browser</li></ul>	<ul style="list-style-type: none"><li>• Internet</li><li>• Electricity</li></ul>



## Indicative content 3.1: Apply JavaScript in Project



Duration: 10 hrs



### Theoretical Activity 3.1.1: Description of JavaScript project environment



#### Tasks:

1. In small groups, you are requested to answer the following questions related to the project environment in javascript:

What do you understand about the following terms used in javascript?

I.javascript project environment.

II.project folder.

2. Provide the answer for the asked questions and write them on papers.
3. Present the findings/answers to the trainer and the whole class
4. Read the key readings 3.1.1 for more clarification and ask questions where necessary.



#### Key readings 3.1.1.:

##### ✓ Create Project folder

In the context of JavaScript or programming in general, the term "project folder" refers to a directory or folder on your computer's file system that contains all the files and resources related to a particular software project or application. These files and resources typically include:

1. **Source Code:** JavaScript files, HTML files, CSS files, and any other programming or scripting files that make up your application.
2. **Assets:** This can include images, fonts, audio, and other media files that your project uses.
3. **Configuration Files:** Configuration files like **package.json**, **webpack.config.js**, or **.gitignore** that define project settings, dependencies, and build configurations.
4. **Documentation:** Any documentation related to the project, such as README files or user guides.

5. **Dependencies:** If you're using a package manager like npm (Node Package Manager) or yarn, your project folder may also contain a **node\_modules** folder that stores the dependencies required by your project.
6. **Build Output:** If your project requires a build step, the output of the build process may be stored in the project folder.

The structure and contents of a project folder can vary depending on the type of project and the tools you are using. Organizing your project files into a well-structured folder hierarchy is essential for code maintainability and collaboration with others.

### ✓ Folders and files structuring

Structuring folders and files for a JavaScript project is essential for maintaining a clean and organized codebase. While JavaScript itself does not provide tools for creating or managing directories and files on your computer's file system, you can use JavaScript in conjunction with Node.js or build tools to automate the process of setting up and organising your project structure.



### Practical Activity 3.1.2: Preparation of project environment

#### Task:

- 1: You are asked to go to computer lab, prepare a project environment in javascript. This task should be done individually.
- 2: Read the key reading 3.1.2 in trainee manual about error handling in JavaScript.
- 3: Referring to the presented steps provided in the key reading 3.1.2 in trainee manual, prepare the javascript environment.
- 4: Ask question for more clarification where necessary.



### Key readings 3.1.2



#### Create project folder

To create a project folder in JavaScript, you would typically use Node.js, which is a JavaScript runtime environment that allows you to run JavaScript outside of a web browser. Here's an example of how you can create a project folder using Node.js:

1. **Install Node.js:** First, you need to install Node.js on your machine. You can download the installer from the official Node.js website (<https://nodejs.org>) and follow the installation instructions for your operating system.

**2. Open your terminal or command prompt:** Once Node.js is installed, open your terminal or command prompt.

**3. Create a new directory:** Use the `mkdir` command to create a new directory for your project. For example, to create a folder named "my-project", you can run the following command:

```
mkdir my-project
```

**4. Navigate into the project folder:** Use the `cd` command to navigate into the newly created project folder. For example, to navigate into the "my-project" folder, you can run the following command:

```
cd my-project
```

Now you have successfully created a project folder named "my-project" using Node.js.

You can further configure your project folder by adding files like JavaScript files, HTML files, CSS files, etc., based on your project requirements.

#### ✓ Folders and files structuring

Here's a step-by-step guide on how to structure folders and files for a JavaScript project using Node.js:

##### **1. Create a New Project Directory:**

Start by creating a new directory (folder) for your JavaScript project. You can do this manually using your operating system's file explorer or by running the following command in your terminal:

```
mkdir my-javascript-project
```

Replace my-javascript-project with the desired name of your project directory.

##### **2. Navigate to the Project Directory:** Change your working directory to the project folder:

```
cd my-javascript-project
```

##### **3. Initialize a Node.js Project:** If you're building a JavaScript project with Node.js, initialize a Node.js project by running:

```
npm init
```

Follow the prompts to set up your project. This will create a **package.json** file in your project directory to manage project dependencies and configurations.

##### **2. Create Subfolders:** Organize your project files into subfolders based on their purpose or type.

Common subfolders include:

- **src**: This folder typically contains your JavaScript source code files.
  - **public or static**: This is where static assets like HTML, CSS, images, and client-side JavaScript files go.
  - **config**: For configuration files.
  - **test**: If you're writing tests, store them in a separate folder.
  - **node\_modules**: Automatically created by npm to store project dependencies.
  - **build or dist**: If your project requires a build step, the output can be stored here.
- Use the `mkdir` command to create these folders:

```
mkdir src public config test build
```

3. **Create Files**: Create files within the appropriate subfolders. For example, you can create an `index.html` file in the **public** folder and a `main.js` file in the **src** folder.  
`touch public/index.html src/main.js`
4. **Organize Configuration Files**: If your project requires configuration files (e.g., `.env`, `.babelrc`, `.eslintrc`), create and place them in the **config** folder.
5. **Add Code and Assets**: Write your JavaScript code, HTML, CSS, and add any other assets to their respective folders.
6. **Install Dependencies**: Use npm to install any dependencies your project needs.

**For example:**

```
npm install --save dependency-name
```

7. **Version Control**: If you plan to use version control (e.g., Git), initialize a repository in your project folder:

```
git init
```

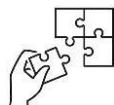
8. **Documentation**: Consider adding documentation files, such as a `README.md`, to describe your project's purpose, setup instructions, and usage.
9. **Build Tools (Optional)**: Depending on the complexity of your project, you may want to set up build tools like Webpack, Babel, or a task runner like Gulp to automate tasks like bundling, transpiling, and minification.
10. **Testing (Optional)**: If you're writing tests for your JavaScript code, set up a testing framework (e.g., Mocha, Jest) and organize your test files in the test folder.

By following these steps, you can create a well-structured folder and file layout for your JavaScript project. Keep in mind that the exact structure may vary depending on the project's size and requirements, but the key is to keep your code organized and easy to maintain.



### Points to Remember

- In the context of JavaScript or programming in general, the term "project folder" refers to a directory or folder on your computer's file system that contains all the files and resources related to a particular software project or application.
- Structuring folders and files for a JavaScript project is essential for maintaining a clean and organized codebase.



### Application of learning 3.1.

You are a developer tasked with setting up the development environment for a new JavaScript project. The project involves building a web application that allows users to create and share interactive quizzes. You need to ensure that your environment is properly configured to support efficient development, testing, and deployment of the application.



## Indicative content 3.2: Create pages with HTML



Duration: 10 hrs



### Theoretical Activity 3.2.1: Description of HTML pages



#### Tasks:

- 1: In small groups, you are requested to answer the following questions related to the drawing materials:
  - i. What do you understand about HTML pages?
  - ii. Explain HTML tables and HTML forms
  - iii. Discuss about the elements of HTML table
  - iv. Explain HTML Form and give its importance in creating HTML pages.
- 2: Provide the answer for the asked questions and write them on papers.
- 3: Present the findings/answers to the whole class
- 4: For more clarification, read the key readings 3.2.1 and ask questions where necessary.



#### Key readings 3.1.1.:

**HTML** (Hypertext Markup Language) pages are documents designed for the world wide web. They consist of structured code that defines the elements and content of a web page.

#### ✓ Tables

In HTML, a table is a structural element used to display data in a tabular format, which consists of rows and columns. Tables are often used to present data in a structured and organised manner, making it easier for users to read and understand information.

##### Table tags

Tags are a set of elements used in web development to structure and format content for websites. These instructions are used to format a web page content.

Table tags are described below:

##### 1. Table

The table's contents are defined by `<table>` and `</table>`

##### 2. Table Cells

Each table cell is defined by a `<td>` and a `</td>` tag. **td** stands for table data.

Everything between `<td>` and `</td>` are the content of the table cell.

**Note:** A table cell can contain all sorts of HTML elements: text, images, lists, links, other tables, etc.

##### 3. Table Rows

Each table row starts with a <tr> and ends with a </tr> tag. **tr** stands for table row. You can have as many rows as you like in a table; just make sure that the number of cells are the same in each row.

#### 4. Table Headers

Sometimes you want your cells to be table header cells. In those cases use the <th> tag instead of the <td> tag. **th** stands for table header.

#### ✓ Form

In HTML, a "form" is a structural element used to create interactive user interfaces that allow users to input and submit data to a web page or a web application. Forms are a fundamental part of web development because they enable users to interact with and provide information to websites. Forms can be used for various purposes, such as user registration, login, search, contact forms, surveys, and more.

A form will take input from the site visitor and then will post it to a back-end application such as CGI, ASP Script or PHP script etc. The back-end application will perform required processing on the passed data based on defined business logic inside the application.

There are various form elements available like text fields, textarea fields, drop-down menus, radio buttons, checkboxes, etc.

The HTML <form> tag is used to create an HTML form and it has following syntax:

```
<form action="Script URL" method="GET|POST">  
form elements like input, textarea etc.  
</form>
```

#### From Attributes

Apart from common attributes, following is a list of the most frequently used form attributes:

Attribute	Description
Action	Backend script ready to process your passed data.
Method	Method to be used to upload data. The most frequently used are GET and POST methods.
Target	Specify the target window or frame where the result of the script will be displayed. It takes values like _blank, _self, _parent etc.
Enctype	You can use the enctype attribute to specify how the browser encodes the data before it sends it to the server. Possible values are: <ul style="list-style-type: none"><li><b>application/x-www-form-urlencoded</b> - This is the standard method most forms use in simple scenarios.</li><li><b>multipart/form-data</b> - This is used when you want to upload binary data in the form of files like image, word file etc.</li></ul>

## Difference Between GET and POST Method in HTML

Both GET and POST method is used to transfer data from client to server in HTTP protocol.

The two methods are distinct where GET method adds the encoded data to the URI while in case of POST method the data is appended to the body rather than URI. Additionally, GET method is used for retrieving the data. Conversely, POST method is used for storing or updating the data.

## HTML Form Controls

There are different types of form controls that you can use to collect data using HTML form:

- Text Input Controls
- Checkboxes Controls
- Radio Box Controls
- Select Box Controls
- File Select boxes
- Hidden Controls
- Button controls

### Text Input Controls

There are three types of text input used on forms:

- **Single-line text input controls:** This control is used for items that require only one line of user input, such as search boxes or names. They are created using HTML `<input>` tag.
- **Password input controls:** This is also a single-line text input but it masks the character as soon as a user enters it. They are also created using HTML `<input>` tag.
- **Multi-line text input controls :** This is used when the user is required to give details that may be longer than a single sentence. Multi-line input controls are created using HTML `<textarea>` tag.

#### 1. Single-line text input controls

This control is used for items that require only one line of user input, such as search boxes or names. They are created using HTML `<input>` tag.

Following is the list of attributes for `<input>` tag for creating text field.

Attribute	Description

Type	Indicates the type of input control and for text input control it will be set to <b>text</b> .
Name	Used to give a name to the control which is sent to the server to be recognized and get the value.
Value	This can be used to provide an initial value inside the control.
Size	Allows to specify the width of the text-input control in terms of characters.
Maxlength	Allows to specify the maximum number of characters a user can enter into the text box.

## 2. Password input controls

This is also a single-line text input but it masks the character as soon as a user enters it. They are also created using HTML <input> tag but type attribute is set to **password**.

Following is the list of attributes for <input> tag for creating password field.

Attribute	Description
Type	Indicates the type of input control and for password input control it will be set to <b>password</b> .
Name	Used to give a name to the control which is sent to the server to be recognized and get the value.
Value	This can be used to provide an initial value inside the control.
Size	Allows to specify the width of the text-input control in terms of characters.
Maxlength	Allows to specify the maximum number of characters a user can enter into the text box.

## 3. Multiple-Line Text Input Controls

This is used when the user is required to give details that may be longer than a single sentence. Multi-line input controls are created using HTML <textarea> tag.

Following is the list of attributes for <textarea> tag.

Attribute	Description
Name	Used to give a name to the control which is sent to the server to be recognized and get the value.
Rows	Indicates the number of rows of text area box.
Cols	Indicates the number of columns of text area box

#### **4. Checkbox Control**

Checkboxes are used when more than one option is required to be selected. They are also created using HTML <input> tag but type attribute is set to checkbox

Following is the list of attributes for <checkbox> tag.

Attribute	Description
Type	Indicates the type of input control and for checkbox input control it will be set to <b>checkbox</b> .
Name	Used to give a name to the control which is sent to the server to be recognized and get the value.
Value	The value that will be used if the checkbox is selected.
Checked	Set to <i>checked</i> if you want to select it by default.

#### **5. Radio Button Control**

Radio buttons are used when out of many options, just one option is required to be selected. They are also created using HTML <input> tag but type attribute is set to **radio**.

Following is the list of attributes for radio button.

Attribute	Description
type	Indicates the type of input control and for checkbox input control it will be set to <b>radio</b> .
name	Used to give a name to the control which is sent to the server to be recognized and get the value.
value	The value that will be used if the radio box is selected.
checked	Set to <i>checked</i> if you want to select it by default.

#### **6. Select Box Control**

A select box, also called drop down box which provides option to list down various options in the form of drop down list, from where a user can select one or more options.

##### **Attributes**

Following is the list of important attributes of <select> tag:

Attribute	Description
name	Used to give a name to the control which is sent to the server to be recognized and get the value.
size	This can be used to present a scrolling list box.
multiple	If set to "multiple" then allows a user to select multiple items from the menu.

Following is the list of important attributes of <option> tag:

Attribute	Description
value	The value that will be used if an option in the select box is selected.
selected	Specifies that this option should be the initially selected value when the page loads.
label	An alternative way of labeling options

### 7. File Upload Box

If you want to allow a user to upload a file to your web site, you will need to use a file upload box, also known as a file select box. This is also created using the <input> element but type attribute is set to **file**.

Following is the list of important attributes of file upload box:

Attribute	Description
name	Used to give a name to the control which is sent to the server to be recognized and get the value.
accept	Specifies the types of files that the server accepts.

### 8. Button Controls

There are various ways in HTML to create clickable buttons. You can also create a clickable button using <input> tag by setting its type attribute to **button**. The type attribute can take the following values:

Type	Description
submit	This creates a button that automatically submits a form.

Reset	This creates a button that automatically resets form controls to their initial values.
Button	This creates a button that is used to trigger a client-side script when the user clicks that button.
Image	This creates a clickable button but we can use an image as background of the button.

## 9. Hidden Form Controls

Hidden form controls are used to hide data inside the page which later on can be pushed to the server. This control hides inside the code and does not appear on the actual page. For example, following hidden form is being used to keep current page number. When a user will click next page then the value of hidden control will be sent to the web server and there it will decide which page has been displayed next based on the passed current page.



### Practical Activity 3.2.1: Creation of pages with HTML



#### Task:

- 1: Referring to the previous theoretical activities 3.2.1 you are requested to go to the computer lab and create pages including form and table. This task should be done individually.
- 2: Read the key reading 3.2.2 in trainee manual about creation of pages with HTML.
- 3: Referring to the key reading 3.2.2 in trainee manual, create html pages with form and table.
- 4: Ask questions for more clarification where necessary.



### Key readings 3.2.1

#### Creating HTML Tables

HTML (Hypertext Markup Language) is a markup language used to structure the content of a webpage. To create a table in HTML, you can use `<table>` element along with its related tags.

Certainly! Here's an example of how you can create a basic HTML table:

```
<!DOCTYPE html>
<html>
<head>
<title>Table Example</title>
</head>
```

```
<body>
  <table>
    <thead>
      <tr>
        <th>Name</th>
        <th>Age</th>
        <th>City</th>
      </tr>
    </thead>
    <tbody>
      <tr>
        <td>John Doe</td>
        <td>25</td>
        <td>New York</td>
      </tr>
      <tr>
        <td>Jane Smith</td>
        <td>30</td>
        <td>London</td>
      </tr>
    </tbody>
  </table>
</body>
</html>
```

In this example, we have a simple table with three columns: Name, Age, and City. The table has a header row defined using the `<thead>` element, and the data is placed within the `<tbody>` element. Each row is created using the `<tr>` element, and the data within each row is placed within `<td>` elements.

An HTML table is created with an opening `<table>` tag and a closing `</table>` tag. Inside these tags, data is organized into rows and columns by using opening and closing table row `<tr>` tags and opening and closing table data `<td>` tags.

Table row `<tr>` tags are used to create a row of data. Inside opening and closing table `<tr>` tags, opening and closing table data `<td>` tags are used to organize data in columns. As an example, here is a table that has two rows and three columns:

```
<table>
  <tr>
    <td>Column 1</td>
    <td>Column 2</td>
    <td>Column 3</td>
  </tr>
  <tr>
```

```
<td>Column 1</td>
<td>Column 2</td>
<td>Column 3</td>
</tr>
</table>
```

To explore how HTML tables work in practice, paste the code snippet above into the index.html file or other html file you are using for this tutorial.

Save and reload the file in the browser to check your results.

Your webpage should now have a table with three columns and two rows

**Column 1 Column 2 Column 3**  
**Column 1 Column 2 Column 3**

#### **Adding a Border to a Table**

In general, tables should be styled with CSS. If you do not know CSS, you can add some light styling using HTML by adding the attributes to the `<table>` element.

For example, you can add a border to the table with the `border` attribute:

```
<table border="1">
<tr>
<td>column 1</td>
<td>column 2</td>
<td>column 3</td>
</tr>
<tr>
<td>column 1</td>
<td>column 2</td>
<td>column 3</td>
</tr>
```

Add the highlighted border attribute to your table and checking your results in the browser. (You can clear your index.html file and paste in the HTML code snippet above.) Save your file and load it in the browser. Your table should now have a border surrounding each of your rows and columns like this:

column 1	column 2	column 3
column 1	column 2	column3

#### **Adding Headings to Rows and Columns**

Headings can be added to rows and columns to make tables easier to read. Table headings are automatically styled with bold and centered text to visually distinguish

them from table data. Headings also make tables more accessible as they help individuals using screen readers navigate table data.

Headings are added by using opening and closing `<th>` tags.

- To add column headers, you must insert a new `<tr>` element at the top of your table where you can add the column names using `<th>` tags.
- To add row headers, you must add opening and closing `<th>` tags as the first item in every table row `<tr>` element. Add the row headers and data by adding the highlighted code snippet below between the closing `</tr>` tag and the closing `</table>` tag of the table in your index.html file:

```
<table border="1">
<tr>
<th></th>
<th>Column Header 1</th>
<th>Column Header 2</th>
<th>Column Header 3</th>
</tr>
<tr>
<th>Row Header 1</th>
<td>Data</td>
<td>Data</td>
<td>Data</td>
</tr>
<tr>
<th>Row Header 2</th>
<td>Data</td>
<td>Data</td>
<td>Data</td>
</tr>
<tr>
<th>Row Header 3</th>
<td>Data</td>
<td>Data</td>
<td>Data</td>
</tr>
</table>
```

Save the index.html file and reload it in your browser. You should receive something like this:

	<b>Column Header 1</b>	<b>Column Header 2</b>	<b>Column Header 3</b>
<b>Row Header 1</b>	Data	Data	Data
<b>Row Header 2</b>	Data	Data	Data
<b>Row Header 3</b>	Data	Data	Data

You should now have a table with three column headings and three row headings.

### **Example of creating tables in HTML.**

#### **Example1:** Table cell

Everything between `<td>` and `</td>` are the content of the table cell.

```
<table border=1>
<tr>
<td>Emil</td>
<td>Tobias</td>
<td>Linus</td>
</tr>
</table>
```

These codes will produce a table as shown below:

Emil	Tobias	Linus
------	--------	-------

### **Example 2 : Table Rows**

Each table row starts with a `<tr>` and ends with a `</tr>` tag.

```
<table border=1>
<tr>
<td>Emil</td>
<td>Tobias</td>
<td>Linus</td>
</tr>
<tr>
<td>16</td>
<td>14</td>
<td>10</td>
</tr>
</table>
```

**The output will be:**

Emil	Tobias	Linus
16	14	10

You can have as many rows as you like in a table; just make sure that the number of cells

is the same in each row.

### Examnple 3 : Table Headers

Sometimes you want your cells to be table header cells. In those cases use the `<th>` tag instead of the `<td>` tag:

#### Example

Let the first row be table header cells:

```
<table>
<tr>
<th>Person 1</th>
<th>Person 2</th>
<th>Person 3</th>
</tr>
<tr>
<td>Emil</td>
<td>Tobias</td>
<td>Linus</td>
</tr>
<tr>
<td>16</td>
<td>14</td>
<td>10</td>
</tr>
</table>
```

These codes will produce something like this:

<b>Person 1</b>	<b>Person 2</b>	<b>Person 3</b>
Emil	Tobias	Linus
16	14	10

#### Specifying table sizes

You can specify width for a table both in percents of page width and in pixels.

```
<HTML>
<HEAD>
<TITLE>Table Example </TITLE>
</HEAD>

<BODY>
<TABLE WIDTH=50% BORDER=1>
<TR>
<TD>Cell Row1 Col1</TD>
<TD>Cell Row1 Col2</TD>
```

```
</TR>
<TR>
<TD>Cell Row2 Col1</TD>
<TD>Cell Row2 Col2</TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

If you want you can determine table width in pixels.

```
<TABLE WIDTH=250 BORDER=1>
<TR>
<TD>Cell Row1 Col1</TD>
<TD>Cell Row1 Col2</TD>
</TR>
<TR>
<TD>Cell Row2 Col1</TD>
<TD>Cell Row2 Col2</TD>
</TR>
</TABLE>
```

You can specify table height too. In this way you can determine height and width of table.

Width and height of table will be divided between cells in rows and columns so if table width is 100 and there are 2 columns then width of each cell will be 50.

Just pay attention to this important point that if you put a lot of text in a cell of a table it will be expanded to fit the text in it.

### **Text alignments in table cells**

By default, text entered in a cell will appear at the left side of the cell. You can add either of these options to <TD> tags to specify horizontal alignment of text.

```
<TD ALIGN=CENTER> or
<TD ALIGN=RIGHT> or
<TD ALIGN=LEFT>
```

As we saw, left alignment is default for cells. You can also determine vertical alignment of text in a cell by adding VALIGN option to <TD> tag.

There are three values for VALIGN option: TOP, BOTTOM and MIDDLE.

MIDDLE is default value if you do not use this parameter.

```
<HTML>
<HEAD>
<TITLE>Table Attributes Example</TITLE>
</HEAD>
<BODY>
<TABLE WIDTH=50% HEIGHT=100 BORDER=3>
```

```

<TR>
<TD ALIGN=LEFT VALIGN=TOP>TOP LEFT</TD>
<TD ALIGN=RIGHT VALIGN=TOP>TOP RIGHT</TD>
</TR>
<TR>
<TD ALIGN=LEFT VALIGN=BOTTOM>BOTTOM LEFT</TD>
<TD ALIGN=RIGHT VALIGN=BOTTOM>BOTTOM RIGHT</TD>
</TR>
</TABLE>
</BODY>
</HTML>

```

**Output:**

TOP LEFT	TOP RIGHT
BOTTOM LEFT	BOTTOM RIGHT

**Images in table cells**

You will soon need to insert images in table cells. You can insert an image in a table cell by inserting **<IMG>** tag between **<TD></TD>** tags of a certain cell.

```

<HTML>
<HEAD>
<TITLE>Example</TITLE>
</HEAD>
<BODY>
<TABLE BORDER=4>
<TR>
<TD><IMG SRC="image53.gif"></TD>
</TR>
</TABLE>
</BODY>
</HTML>

```

**Cell Width (Column Width)**

In previous lesson we learned how we can determine width and height of a table.

```

<HTML>
<HEAD>
<TITLE>Table: Column widths not specified</TITLE>
</HEAD>
<BODY>

```

```
<TABLE WIDTH=400 HEIGHT=100 BORDER=3>
<TR>
<TD>TOP LEFT</TD>
<TD>TOP RIGHT</TD>
</TR>
<TR>
<TD>BOTTOM LEFT</TD>
<TD>BOTTOM RIGHT</TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

In above table we have not determined sizes for two cells in first row. In this way you will not be able to say how these cells will display in different browsers and different screen modes.

You can determine width of each column in your table by specifying width of cells in first row.

Just be careful about correctness of sizes you specify. For example if your table width is 200 pixels sum of cell widths must be exactly 200.

```
<HTML>
<HEAD>
<TITLE>Example</TITLE>
</HEAD>
<BODY>
<TABLE WIDTH=400 HEIGHT=100 BORDER=3>
<TR>
<TD WIDTH=140>TOP LEFT</TD>
<TD WIDTH=260>TOP RIGHT</TD>
</TR>
<TR>
<TD>BOTTOM LEFT</TD>
<TD>BOTTOM RIGHT</TD>
</TR>
</TABLE>
</BODY> </HTML>
```

You can also determine cell widths in percent. Sum of cell width percentages must be 100%.

```
<HTML>
<HEAD>
```

```
<TITLE>Example</TITLE>
</HEAD>
<BODY>
<TABLE WIDTH=400 HEIGHT=100 BORDER=3>
<TR>
<TD WIDTH=35%>TOP LEFT</TD>
<TD WIDTH=65%>TOP RIGHT</TD>
</TR>
<TR>
<TD>BOTTOM LEFT</TD>
<TD>BOTTOM RIGHT</TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

When you determine sizes of first row cells you will not need to determine widths for second row cells. If you want a cell to be empty, you cannot omit definition for that cell. Insert cell definition, and enter a &nbsp; between <TD> </TD> tags.

As we told in later lessons this means a space character. You must enter at least a space in this form if you need an empty cell. Otherwise area of that cell will not appear like an empty cell.

```
<HTML>
<HEAD>
<TITLE>Example</TITLE>
</HEAD>
<BODY>
<TABLE WIDTH=400 HEIGHT=100 BORDER=3>
<TR>
<TD WIDTH=140>TOP LEFT</TD>
<TD WIDTH=260>&nbsp;</TD>
</TR>
<TR>
<TD>&nbsp;</TD>
<TD>BOTTOM RIGHT</TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

In above example we have two empty cells but as we have specified both tables and

cell sizes, table will not lose its shape. If we remove sizes, we cannot guarantee how it will be displayed on different browsers and screen modes.

### Cell padding

You can specify two other important size parameters for a table. Cell padding is the space between cell borders and table contents such as text, image etc.

```
<HTML>
<HEAD>
<TITLE>Example</TITLE>
</HEAD>
<BODY>
Cell padding effect: <BR><BR>
<TABLE BORDER=3 CELLPADDING=20>
<TR>
<TD>TOP LEFT</TD>
<TD>TOP RIGHT</TD>
</TR>
<TR>
<TD>BOTTOM LEFT</TD>
<TD>BOTTOM RIGHT</TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

Default value for this option is 1. It means that contents of a cell will have a distance of one pixel with borders. If you don't want any space between object inside the cells and its borders you can determine the value of 0 for this option.

### Cell spacing

Cell spacing parameter determines the space between inner and outer parts of a table. In fact, a table is constructed from two borders: A border area and a cell area. There is a space between cell area and outer border. We call this "cell spacing".

If you increase this value, you will have a thick border. Default value for this property is 2. If you specify 0 for it, you will have a very thin border.

```
<HTML>
<HEAD>
<TITLE>Example</TITLE>
</HEAD>
<BODY>
Cell spacing effect : <BR><BR>
```

```
<TABLE BORDER=3 CELLSPACING=10>
<TR>
<TD>TOP LEFT</TD>
<TD>TOP RIGHT</TD>
</TR>
<TR>
<TD>BOTTOM LEFT</TD>
<TD>BOTTOM RIGHT</TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

You can also mix cell spacing and cell padding options to make specific tables that you need.

#### **Table background color**

We can use background colors for tables in new browsers. You can specify background color options inside <TABLE> tag.

```
<HTML>
<HEAD>
<TITLE>Table bgcolor Example</TITLE>
</HEAD>
<BODY>
<TABLE width="300" BGCOLOR="#66CCFF">
<TR>
<TD width="60">A</TD>
<TD width="60">B</TD>
</TR>
<TR>
<TD width="70">C</TD>
<TD width="50">D</TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

In above example entire table will change to new color even table borders. You can also determine background color for each row of your table. If you want to do this, you must use BGCOLOR option inside <TR> tag of the desired row. This second method will only change colors of cells in specified row.

```
<HTML>
```

```

<HEAD>
<TITLE>Another Example </TITLE>
</HEAD>
<BODY>
<TABLE width="300" BORDER=1>
<TR BGCOLOR="#66CCFF">
<TD>A</TD>
<TD>B</TD>
</TR>
<TR BGCOLOR="#CCFFFF">
<TD width="50%">C</TD>
<TD width="50%">D</TD>
</TR>
</TABLE>
</BODY>
</HTML>

```

You can even change color of individual cells by using BGCOLOR option in <TD> </TD> cell tags. You can mix all above options to create your desired table. In next example we will change color of first row to "#336699". Then we will change color of two cells in second row to "#66CCFF" and "#CCFFFF" respectively.

```

<HTML>
<HEAD>
<TITLE>Example</TITLE>
</HEAD>
<BODY>
<TABLE width="300" BORDER=1>
<TR BGCOLOR="#336699">
<TD width="50%">A</TD>
<TD width="50%">B</TD>
</TR>
<TR>
<TD width="50%" BGCOLOR="#66CCFF">C</TD>
<TD width="50%" BGCOLOR="#CCFFFF">D</TD>
</TR>
</TABLE>
</BODY></HTML>

```

### **Column Span**

Sometimes you need to join two cells in a row to each other. For example, in a 2\*3 table we may want to join two cells with each other. In this way we will have two cells in first

row and three cells in second row. Enter this html code in a file and browse it in your browser to see what column spans is.

```
<HTML>
<HEAD>
<TITLE>Example </TITLE>
</HEAD>
<BODY>
<TABLE BORDER=1>
<TR>
<TD COLSPAN=2>A</TD>
<TD>B</TD>
</TR>

<TR>
<TD>A</TD>
<TD>B</TD>
<TD>C</TD>
</TR>
</TABLE>
</BODY> </HTML>
```

Just be careful that when you have for example 2 cells in first row and first one uses column span parameter COLSPAN=2 it means that it is equal to two cells.

Therefore you must have three cells in next row (three <TR> tags) or you may use COLSPAN to create cells that when you add them, it will be equal to previous row or 3 in this example.

### Row Span

This time we want to join two cells in a column (from different rows). This is the same as previous section with the difference that we will join cells from different rows rather than cells in different columns.

This time we must use ROWSPAN instead of COLSPAN.

```
<HTML>
<HEAD>
<TITLE>Example </TITLE>
</HEAD>
<BODY>
<TABLE BORDER="1" WIDTH="200">
<TR>
<TD ROWSPAN="2">A</TD>
<TD>B</TD>
```

```
<TD>C</TD>
</TR>
<TR>
<TD>D</TD>
<TD>E</TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

Again you must be careful that when you have for example a cell in first column that you have joined two cells to create it using the option ROWSPAN=2 then your table must have two rows and you must take this in mind in next parts of your table. In above example we only entered two cells in second row (started from second <TR>) as first cell of first row has occupied first cell of this row too and we have only two cells left of 3 cells.

### Nested Tables

Yes, we can nest tables in each other. If you are going to design complicated web pages you will always do this.

```
<HTML>
<HEAD>
<TITLE>Example</TITLE>
</HEAD>
<BODY>
<TABLE border="0" width="750">
<TR>
<TD width="25%">&ampnbsp</TD>
<TD width="25%">&ampnbsp</TD>
<TD width="25%">
<TABLE border="2" width="100%">
<TR>
<TD width="50%">>1-</TD>
<TD width="50%">>HTML</TD>
</TR>
<TR>
<TD width="50%">>2-</TD>
<TD width="50%">>C Prog. </TD>
</TR>
<TR>
<TD width="50%">>3-</TD>
<TD width="50%">>JScript</TD>
</TR>
```

```

</TABLE>
</TD>
<TD width="25%">&nbsp;</TD>
</TR>
</TABLE>
</BODY>
</HTML>

```

### Creating HTML Form

HTML Forms are required when you want to collect some data from the site visitor. For example, during user registration you would like to collect information such as name, email address, credit card, etc.

A form will take input from the site visitor and then will post it to a back-end application such as CGI, ASP Script or PHP script etc. The back-end application will perform required processing on the passed data based on defined business logic inside the application.

There are various form elements available like text fields, textarea fields, drop-down menus, radio buttons, checkboxes, etc.

The HTML **<form>** tag is used to create an HTML form and it has following syntax:

```

<form action="Script URL" method="GET|POST">
    form elements like input, textarea etc.
</form>

```

### Form Attributes

Apart from common attributes, following is a list of the most frequently used form attributes:

Attribute	Description
Action	Backend script ready to process your passed data.
Method	Method to be used to upload data. The most frequently used are GET and POST methods.
Target	Specify the target window or frame where the result of the script will be displayed. It takes values like _blank, _self, _parent etc.
Enctype	You can use the enctype attribute to specify how the browser encodes the data before it sends it to the server. Possible values are:

- **application/x-www-form-urlencoded** - This is the standard method most forms use in simple scenarios.
- **multipart/form-data** - This is used when you want to upload binary data in the form of files like image, word file etc.

## Difference Between GET and POST Method in HTML

Both GET and POST method is used to transfer data from client to server in HTTP protocol.

The two methods are distinct where GET method adds the encoded data to the URI while in case of POST method the data is appended to the body rather than URI. Additionally, GET method is used for retrieving the data. Conversely, POST method is used for storing or updating the data.

## HTML Form Controls

There are different types of form controls that you can use to collect data using HTML form.

### ✓ Text Input Controls

There are three types of text input used on forms: Single-line text input controls, Password input controls and multi-line text input controls.

#### 1. Single-line text input controls

This control is used for items that require only one line of user input, such as search boxes or names. They are created using HTML <input> tag.

Here is a basic example of a single-line text input used to take first name and last name:

```
<!DOCTYPE html>
<html>
<head>
<title>Text Input Control</title>
</head>
<body>
<form >
  First name: <input type="text" name="first_name" />
<br>
  Last name: <input type="text" name="last_name" />
</form>
</body>
</html>
```

This will produce the following result:

First name:

Last name:

Following is the list of attributes for <input> tag for creating text field.

Attribute	Description
Type	Indicates the type of input control and for text input control it will be set to <b>text</b> .
Name	Used to give a name to the control which is sent to the server to be recognized and get the value.
Value	This can be used to provide an initial value inside the control.
Size	Allows to specify the width of the text-input control in terms of characters.
Maxlength	Allows to specify the maximum number of characters a user can enter into the text box.

## 2. Password input controls

This is also a single-line text input but it masks the character as soon as a user enters it. They are also created using HTML <input> tag but type attribute is set to **password**.

Here is a basic example of a single-line password input used to take user password:

```
<!DOCTYPE html>
<html>
<head>
<title>Password Input Control</title>
</head>
<body>
<form >
User ID : <input type="text" name="user_id" />
<br>
Password: <input type="password" name="password" />
</form>
</body>
</html>
```

This will produce following result:

User ID :

Password:

Following is the list of attributes for <input> tag for creating password field.

Attribute	Description
Type	Indicates the type of input control and for password input control it will be set to <b>password</b> .
Name	Used to give a name to the control which is sent to the server to be recognized and get the value.
Value	This can be used to provide an initial value inside the control.
Size	Allows to specify the width of the text-input control in terms of characters.
Maxlength	Allows to specify the maximum number of characters a user can enter into the text box.

### 3. Multiple-Line Text Input Controls

This is used when the user is required to give details that may be longer than a single sentence. Multi-line input controls are created using HTML <textarea> tag.

Here is a basic example of a multi-line text input used to take item description:

```
<!DOCTYPE html>
<html>
<head>
<title>Multiple-Line Input Control</title>
</head>
<body>
<form>
Description : <br />
<textarea rows="5" cols="50" name="description">
Enter description here...
</textarea>
</form>
</body>
</html>
```

This will produce following result:

Description :

Enter description here...

Following is the list of attributes for <textarea> tag.

Attribute	Description
Name	Used to give a name to the control which is sent to the server to be recognized and get the value.
Rows	Indicates the number of rows of text area box.
Cols	Indicates the number of columns of text area box

#### ✓ Checkbox Control

Checkboxes are used when more than one option is required to be selected. They are also created using HTML <input> tag but type attribute is set to **checkbox**.

Here is an example HTML code for a form with two checkboxes:

```
<!DOCTYPE html>
<html>
<head>
<title>Checkbox Control</title>
</head>
<body>
<form>
<input type="checkbox" name="maths" value="on"> Maths
<input type="checkbox" name="physics" value="on"> Physics
</form>
</body>
</html>
```

This will produce following result:

Maths  Physics

Following is the list of attributes for <checkbox> tag.

Attribute	Description
Type	Indicates the type of input control and for checkbox input control it will be set to <b>checkbox</b> .

Name	Used to give a name to the control which is sent to the server to be recognized and get the value.
Value	The value that will be used if the checkbox is selected.
Checked	Set to <i>checked</i> if you want to select it by default.

### ✓ Radio Button Control

Radio buttons are used when out of many options, just one option is required to be selected. They are also created using HTML `<input>` tag but type attribute is set to **radio**.

Here is example HTML code for a form with two radio buttons:

```
<!DOCTYPE html>
<html>
<head>
<title>Radio Box Control</title>
</head>
<body>
<form>
<input type="radio" name="subject" value="maths"> Maths
<input type="radio" name="subject" value="physics"> Physics
</form>
</body>
</html>
```

This will produce following result:



Maths Physics

Following is the list of attributes for radio button.

Attribute	Description
type	Indicates the type of input control and for checkbox input control it will be set to <b>radio</b> .
name	Used to give a name to the control which is sent to the server to be recognized and get the value.
value	The value that will be used if the radio box is selected.
checked	Set to <i>checked</i> if you want to select it by default.

### ✓ Select Box Control

A select box, also called drop down box which provides option to list down various options in the form of drop down list, from where a user can select one or more options.

Here is example HTML code for a form with one drop down box

```
<!DOCTYPE html>
<html>
<head>
<title>Select Box Control</title>
</head>
<body>
<form>
<select name="dropdown">
<option value="Maths" selected>Maths</option>
<option value="Physics">Physics</option>
</select>
</form>
</body>
</html>
```

This will produce following result:



Following is the list of important attributes of <select> tag:

Attribute	Description
name	Used to give a name to the control which is sent to the server to be recognized and get the value.
size	This can be used to present a scrolling list box.
multiple	If set to "multiple" then allows a user to select multiple items from the menu.

Following is the list of important attributes of <option> tag:

Attribute	Description
value	The value that will be used if an option in the select box is selected.
selected	Specifies that this option should be the initially selected value when the page loads.

label	An alternative way of labeling options
-------	--

### ✓ File Upload Box

If you want to allow a user to upload a file to your web site, you will need to use a file upload box, also known as a file select box. This is also created using the `<input>` element but type attribute is set to **file**.

#### Example

Here is example HTML code for a form with one file upload box:

```
<!DOCTYPE html>
<html>
<head>
<title>File Upload Box</title>
</head>
<body>
<form>
<input type="file" name="fileupload" accept="image/*" />
</form>
</body>
</html>
```

This will produce following result:

No file selected.

Following is the list of important attributes of file upload box:

Attribute	Description
name	Used to give a name to the control which is sent to the server to be recognized and get the value.
accept	Specifies the types of files that the server accepts.

### ✓ Button Controls

There are various ways in HTML to create clickable buttons. You can also create a clickable button using `<input>` tag by setting its type attribute to **button**. The type attribute can take the following values:

Type	Description
submit	This creates a button that automatically submits a form.

Reset	This creates a button that automatically resets form controls to their initial values.
Button	This creates a button that is used to trigger a client-side script when the user clicks that button.
Image	This creates a clickable button but we can use an image as background of the button.

Here is example HTML code for a form with three types of buttons:

```
<!DOCTYPE html>
<html>
<head>
<title>File Upload Box</title>
</head>
<body>
<form>
<input type="submit" name="submit" value="Submit" />
<input type="reset" name="reset" value="Reset" />
<input type="button" name="ok" value="OK" />
<input type="image" name="imagebutton" src="/html/images/logo.png" />
</form>
</body>
</html>
```

This will produce following result:

The screenshot shows a simple HTML form within a browser window. The form contains three buttons: "Submit", "Reset", and "OK". To the right of the buttons, there is a logo for "tutorials point" with the tagline "SIMPLY EASY LEARNING". The logo features a stylized green figure inside a diamond shape.

#### ✓ Hidden Form Controls

Hidden form controls are used to hide data inside the page which later on can be pushed to the server. This control hides inside the code and does not appear on the actual page. For example, following hidden form is being used to keep current page number.

When a user will click next page then the value of hidden control will be sent to the web server and there it will decide which page has been displayed next based on the passed current page.

Here is example HTML code to show the usage of hidden control:

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

```
<title>File Upload Box</title>
</head>
<body>
<form>
<p>This is page 10</p>
<input type="hidden" name="pagename" value="10" />
<input type="submit" name="submit" value="Submit" />
<input type="reset" name="reset" value="Reset" />
</form>
</body>
</html>
```

**The output will be:**

This is page 10

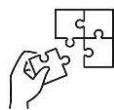
**Submit**

**Reset**



## Points to Remember

- A table is a structural element used to display data in a tabular format, which consists of rows and columns. Tables are often used to present data in a structured and organised manner, making it easier for users to read and understand information. To create table in HTML we use tags which are instructions to format a web page content. Examples of these tags are `<table>....</table>` , `<td>...</td>`, `<tr>...</tr>` and `<th>...</th>` .
- In HTML, a "form" is a structural element used to create interactive user interfaces that allow users to input and submit data to a web page or a web application. There are different types of form controls that you can use to collect data using HTML form: these are Text Input Controls, Checkboxes Controls, Radio Box Controls, Select Box Controls, etc.
- The HTML `<form>` tag is used to create an HTML form and it has following syntax: `<form action="Script URL" method="GET|POST">` form elements like input, textarea etc. The most frequently used methods are GET and POST and both methods are used to transfer data from client to server in HTTP protocol. Additionally, GET method is used for retrieving the data. Conversely, POST method is used for storing or updating the data.



## Application of learning 3.2.

You need to create a basic HTML page to track your monthly expenses. The goal is to design a form where you can input your expenses, and the page will display the expense records in a table.



## Indicative content 3.3: Apply CSS to HTML pages



Duration: 15



### Theoretical Activity 3.3.1: Description of CSS to HTML pages



#### Tasks:

1. In small groups, you are requested to answer the following questions related to the CSS to HTML pages:
  - i. What do you understand about the term CSS?
  - ii. Describe the ways through which CSS can be added to HTML.
2. Provide the answer for the asked questions and write them on papers.
3. Present the findings/answers to the trainer and whole class
4. Read the key readings 3.3.1 and ask questions where necessary.



#### Key readings 3.3.1.:

CSS is the acronym of “Cascading Style Sheets”. CSS is a computer language for laying out and structuring web pages (HTML or XML).

Applying CSS (Cascading Style Sheets) to HTML pages is a fundamental part of web development, as it allows you to control the presentation and styling of your web content.

To apply CSS (Cascading Style Sheets) to HTML pages, you need to include the CSS rules within your HTML document or link to an external CSS file. CSS is used to define the styling and layout of your HTML content, such as fonts, colors, spacing, and positioning.

Here's how you can apply CSS to your HTML pages:

##### ✓ **Inline css**

Inline CSS refers to the practice of applying CSS styles directly to individual HTML elements using the style attribute. This means that you specify the styling rules within the HTML element itself, rather than in a separate external CSS file or within a `<style>` element in the HTML document's `<head>` section.

##### ✓ **Internal css**

Internal CSS, also known as embedded CSS or in-line CSS, is a method of including CSS styles directly within an HTML document, typically within the `<style>` element located in the document's `<head>` section. Internal CSS is a middle-ground approach between

inline CSS and external CSS (styles in a separate CSS file). With internal CSS, you can define styles for a specific HTML document without the need for a separate external CSS file.

#### ✓ External css

External CSS, also known as an external style sheet, is a method of separating the presentation (styling) of a web page from its content (HTML) by placing the CSS rules in a separate external file with a ".css" extension. This file is then linked to one or more HTML documents, allowing you to maintain consistent and organised styles across multiple web pages.

#### ✓ Imported css

Importing **css** means to **import** the file of **css** from inside the directory or project.

By using these methods, you can apply CSS styles to your HTML pages to control the appearance and layout of your content. External CSS is generally the preferred way to manage styles for larger projects, as it promotes separation of concerns and easier maintenance.



### Practical Activity 3.3.2: Applying CSS to HTML pages



#### Task:

- 1: Do the task described below:

You are asked to open the computers, then the IDE and add CSS to HTML pages by using  
Inline css, Internal css, External css and imported css

- 2: Read the key reading 3.3.2 in trainee manual on how to apply CSS to HTML pages.
- 3: Referring to the key reading 3.3.2 in trainee manual, apply CSS to HTML pages.
- 4: Ask questions for more clarification where necessary.



### Key readings 3.3.2

To apply **CSS** (Cascading Style Sheets) to HTML pages, you need to include the CSS rules within your HTML document or link to an external CSS file. CSS is used to define the styling and layout of your HTML content, such as fonts, colors, spacing, and positioning. Here's how you can apply CSS to your HTML pages:

#### ✓ Inline css

Inline CSS is a way of defining the styling of an HTML element by adding CSS rules directly to the element's tag using the "style" attribute. It is used for quick and simple styling changes to specific elements, without creating a separate CSS file.

Inline CSS Syntax:

```
<p style="css_styles">  
    // Content  
</p>
```

**Inline CSS Example:**

In this example, we will change the color and font size of a paragraph element with the help of the “style” attribute.

```
<!DOCTYPE html>  
<html>  
<head>  
    <title>  
        Inline CSS  
    </title>  
</head>  
<body>  
    <h2 style="color: green;  
            font-size: 18px;">  
        Welcome To GFG  
    </h2>  
    <p style="color: red;  
            font-size: 14px;">  
        This is some text. style by inline CSS  
    </p>  
</body>  
</html>
```

**Output:**

**Welcome To GFG**

This is some text. style by inline CSS

**✓ Internal css**

**Internal** CSS, also known as **embedded** CSS, involves adding CSS rules directly within the `<style>` element in the `<head>` section of an HTML document.

It allows styling specific to that document.

**Internal CSS Syntax:**

```
<style>  
// CSS Properties  
</style>
```

**Here's how you can use internal CSS:**

1. Open your HTML page and locate **<head>** opening tag.
2. Put the following code right after the **<head>** tag
3. **<style type="text/css">**
4. Add CSS rules on a new line. Here's an example:

```
body {  
background-color: blue;  
}  
h1 {  
color: red;  
padding: 60px;  
}
```

Type the closing tag: **</style>**

**Internal CSS Example:**

Here is the basic implementation of internal CSS.

```
<!DOCTYPE html>  
<html>  
<head>  
<title>  
Internal CSS  
</title>  
<style>  
h1 {  
color: blue;  
font-size: 24px;  
font-weight: bold;  
}  
p {  
color: green;  
font-size: 16px;  
}
```

```
</style>
</head>
<body>
<h1>SECTOR: ICT AND MULTIMEDIA</h1>
<p>TRADE: SOFTWARE DEVELOPMENT</p>
</body>
</html>
```

**OUTPUT:**

**SECTOR: ICT AND MULTIMEDIA**

TRADE: SOFTWARE DEVELOPMENT

**✓ External css**

For larger projects or when you want to reuse styles across multiple HTML pages, it's best to create an external CSS file and link to it from your HTML pages. External CSS is used to place CSS code in a separate file and link to the HTML document.

To use external CSS, create a separate file with the .css file extension that contains your CSS rules.

Here's how you can do that:

Create an external CSS file (e.g., styles.css):

```
/* styles.css */
```

```
p {
color: green;
font-size: 20px;
}
```

```
.container {
background-color: #f0f0f0;
padding: 20px;
}
```

Link the external CSS file to your HTML document:

```
<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
<p>This is green text with a larger font size.</p>
<div class="container">
<p>This text is inside a container with a gray background.</p>
```

```
</div>
</body>
</html>
```

### ✓Imported css

To import CSS in JavaScript, you can use the import statement if your environment supports ES6 modules. Here's an example:

Assuming you have a JavaScript file (main.js) and a CSS file (styles.css) in the same directory:

#### 1. styles.css:

```
.body {
    background-color: #f0f0f0;
    font-family: Arial, sans-serif;
}
.header {
    color: #333;
    font-size: 24px;
}
```

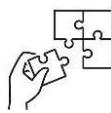
#### 2. main.js:

```
// main.js
import './styles.css';
// Your JavaScript code
```



### Points to Remember

- CSS is the acronym of “Cascading Style Sheets”. CSS is a computer language for laying out and structuring web pages (HTML or XML). Applying CSS (Cascading Style Sheets) to HTML pages is a fundamental part of web development, as it allows you to control the presentation and styling of your web content. CSS can be added to HTML by using **Inline CSS, Internal CSS, External CSS or imported CSS**. To apply CSS to HTML pages, you need to include the CSS rules within your HTML document or link to an external CSS file.



### Application of learning 3.3.

You have successfully created the Expense Tracker web page using HTML. Now, apply CSS to improve the visual appearance and layout of the page. You want to enhance the design by adding styles to the form, table, headings, and buttons.



## Indicative content 3.4: Apply JavaScript concepts in project



Duration: 15



### Theoretical Activity 2.1.1: Description JavaScript concepts used in a project



#### Tasks:

1. In small groups, you are asked to answer the following questions related to the javascript concepts:

What do you understand about the following terms as used in JavaScript:

- i. Variables
- ii. Operators
- iii. Conditional statements
- vi. Looping statements
- v. Functions
- vi. Objects

2. Provide the answer for the asked questions and write them on papers.
3. Present the findings/answers to the trainer and the whole class
4. For more clarification, read the key readings 3.4.1. In addition, ask questions where necessary.



#### Key readings 3.4.1.:

Here's a brief explanation of how to Apply JavaScript in Project using JavaScript variables, operators, conditional statements, Looping statements, functions, and objects:

##### ✓ Variables:

Before using any variable in javascript you need to declare it first. To declare variables, use var, let, or const keywords.

##### Example

```
let age = 25;
```

##### ✓ Operators:

Operators can be used to perform operations on operands to manipulate values stored in variables.

##### ✓ Conditional Statements:

Use if, if... else, if ...else...if and switch case for decision-making.

##### ✓ Looping statements

Some examples of loops to be used are: for loop, while loop and do...while loop:

**✓ Functions:**

Define functions using the function keyword.

**✓ Objects:**

Create objects to group related data and functions.



### Practical Activity 3.4.2: Applying JavaScript concepts in a project



**Task:**

- 1: Referring to the previous theoretical activities (3.4.2) you are requested to go to the computer lab to apply javascript concepts. This task should be done individually.
- 2: Read the key reading 3.4.2 in trainee manual on how to apply JavaScript concepts in a project.
- 3: Reffering to the key reading 3.4.2 in trainee manual apply JavaScript concepts in a project.
4. Ask questions for more clarification where necessary



### Key readings 3.4.2

Here's a brief explanation of how to Apply JavaScript in Project using JavaScript variables, operators, conditional statements, Looping statements, functions, and objects:

**✓ Variables:**

Declare variables using var, let, or const.

**Example**

```
let age = 25;
```

**✓ Operators:**

Arithmetic operators (+, -, \*, /, %)

Comparison operators (==, ===, !=, !==, <, >, <=, >=)

Logical operators (&&, ||, !)

**Example:**

```
let result = (5 + 3) * 2;
```

**✓ Conditional Statements:**

Use if, if... else, if ...else...if and switch case for decision-making.

**Example:**

```
let grade = 85;
```

```
if (grade >= 90) {
```

```
    console.log("A");
```

```
} else if (grade >= 80) {  
    console.log("B");  
} else {  
    console.log("C");  
}
```

### ✓Looping statements

#### for loop:

```
for (let i = 0; i < 5; i++) {  
    console.log(i);  
}
```

This prints numbers from 0 to 4. It initializes i to 0, executes the code block as long as i is less than 5, and increments i after each iteration.

#### while loop:

```
let count = 0;  
while (count < 3) {  
    console.log(count);  
    count++;  
}
```

This prints numbers from 0 to 2. It repeats the code block as long as the count is less than 3.

#### do...while loop:

```
let x = 0;  
do {  
    console.log(x);  
    x++;  
} while (x < 3);
```

This also prints numbers from 0 to 2. It ensures the code block is executed at least once before checking the condition.

### ✓ Functions:

Define functions using the function keyword.

#### Example:

```
function greet(name) {  
    return "Hello, " + name + "!";  
}
```

```
let greeting = greet("John");  
console.log(greeting);
```

### ✓ Objects:

Create objects to group related data and functions.

#### Example:

```
let person = {
```

```
name: "Alice",
age: 30,
greet: function() {
  console.log("Hello, " + this.name + "!");
}
};

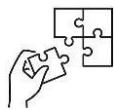
console.log(person.age);
person.greet();
```

These are fundamental concepts in JavaScript that allow you to manipulate data, make decisions, and organise code effectively.



### Points to Remember

- There are some concepts in JavaScript that are essential for building interactive and dynamic web applications. Understanding these concepts will help you write efficient and effective JavaScript code. These concepts are variables, operators, conditional statements, Looping statements, functions, and objects.
- To apply JavaScript concepts effectively in your web development projects, follow these steps:
  1. Set Up Your Development Environment
  2. Link JavaScript to HTML
  3. Understand Basic Syntax and Data Types
  4. Write Functions
  5. Implement Control Structures
  6. Work with Arrays and Objects



### Application of learning 3.4.

You have created an Expense Tracker web page with HTML and applied CSS to improve its visual appearance. Now add JavaScript functionality to calculate and display the total expenses for the entered items. The page should update the total whenever a new expense is added.



## Learning outcome 3 end assessment

### Written assessment

1. Choose the correct answer

i. A set of opening, and closing tags are \_\_\_\_\_ to create a table

- a. required
- b. unnecessary
- c. optional
- d. rows

**Answer:** d. required

ii. The <td> tag is used to create a \_\_\_\_\_

- a. row
- b. table
- c. heading
- d. column

**Answer:** d. column

iii. What is the correct syntax for referring an external CSS?

- A. <link rel="stylesheet" type="text/css" href="mystyle.css">
- B. <stylesheet rel="stylesheet" type="text/css" href="mystyle.css">
- C. <style rel="stylesheet" type="text/css" href="mystyle.css">
- D. All of the above

**Answer:** A) <link rel="stylesheet" type="text/css" href="mystyle.css">

2. Fill in the blank the correct word(s)

i. Inline styles are written within the \_\_\_\_\_ attribute.

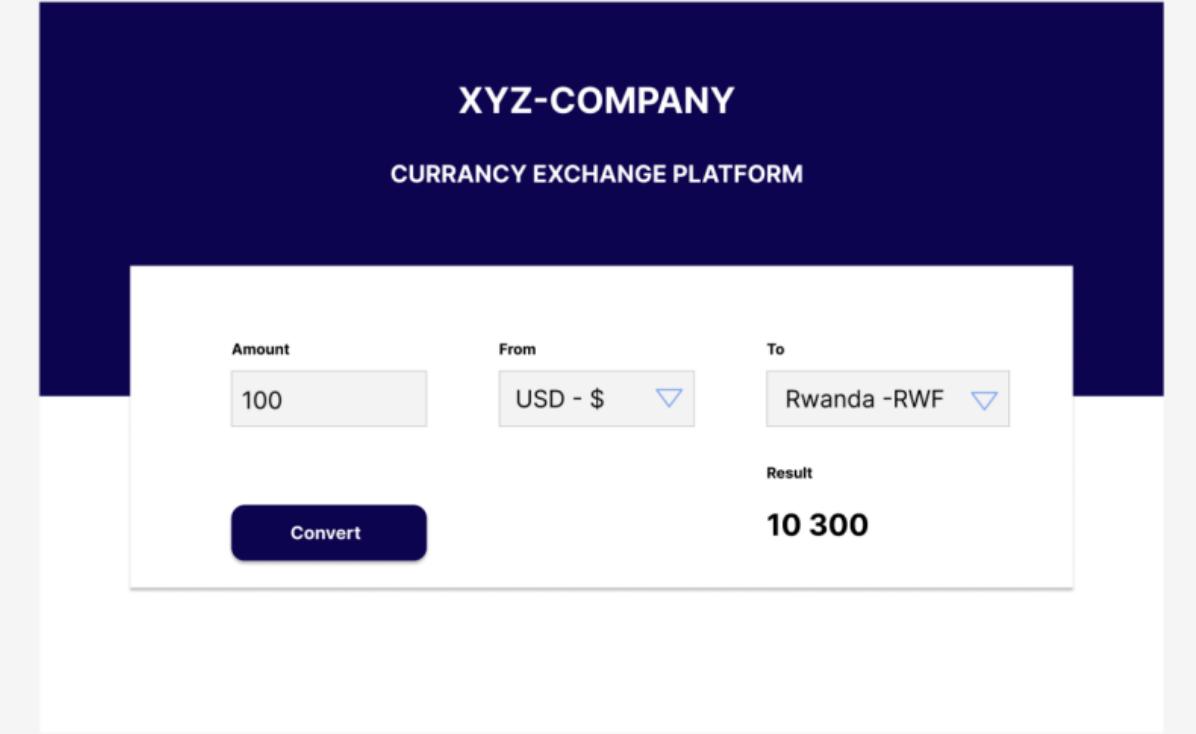
**Answer:** Inline styles are defined within the **style** attribute of the relevant element.

ii. The \_\_\_\_\_ property is used to define the background color in CSS.

**Answer:** The **background-color** property is used to define the background color in CSS.

### Practical assessment

XYZ Company is a forex bureau located in Rubavu District, they exchange money from one currency to another with cash. In that company, they use a manual calculator in exchanging currencies. They want to have an online web calculator project for currency exchange. This platform will be able to convert amounts from one currency to another. They hired a UI/UX Designer to design a mockup for the project, that mockup is provided below.



The image shows a wireframe mockup of a currency exchange application. At the top, there is a dark blue header with the text "XYZ-COMPANY" and "CURRENCY EXCHANGE PLATFORM" in white. Below the header is a light gray form area. On the left side of the form, there is a large empty rectangular placeholder. To the right of this placeholder are three input fields: "Amount" containing "100", "From" containing "USD - \$", and "To" containing "Rwanda - RWF". Below these fields is a "Result" section showing the value "10 300". At the bottom left of the form is a dark blue button labeled "Convert".

XYZ Company hired you as a frontend developer to develop the platform above by using HTML, CSS and JavaScript.



## References

CSS Multiple-Choice Questions (MCQs) and Answers

<https://www.mygreatlearning.com/blog/javascript-projects/#:~:text=With%20JavaScript%2C%20you%20can%20breathe,fascinating%20world%20of%20web%20development.>

<https://www.tutorialrepublic.com/css-tutorial/css-get-started.php>



October, 2024