UNIT – I

Overview & Introduction to JavaScript

1.1 What is JavaScript?

- JavaScript is *an object-based scripting language* which is lightweight and cross-platform.
- JavaScript is interpreted programming language. The JavaScript translator (embedded in the browser) is responsible for translating the JavaScript code for the web browser.
- JavaScript is used by several websites for scripting the webpages.
- JavaScript enables dynamic interactivity on websites when applied to an HTML document.
- With JavaScript, users can build modern web applications to interact directly without reloading the page every time. The traditional website uses js to provide several forms of interactivity and simplicity.
- Although, JavaScript has no connectivity with Java programming language. The name was suggested and provided in the times when Java was gaining popularity in the market.
- It was created in 1995 by Brendan Eich while he was an engineer at Netscape.
- JavaScript was first known as **LiveScript,** but Netscape changed its name to JavaScript, possibly because of the excitement being generated by Java.
- JavaScript made its first appearance in Netscape 2.0 in 1995 with the name **LiveScript**.

Features of JavaScript

1. All popular web browsers support JavaScript as they provide built-in execution environments.
2. JavaScript follows the syntax and structure of the C programming language. Thus, it is a structured programming language.
3. JavaScript is a weakly typed language, where certain types are implicitly cast (depending on the operation).

4. JavaScript is an object-oriented programming language that uses prototypes rather than using classes for inheritance.
5. It is a light-weighted and interpreted language.
6. It is a case-sensitive language.
7. It is supportable in several operating systems including, Windows, macOS, etc.
8. It provides good control to the users over the web browsers.
9. It is open source and cross-platform.

Application of JavaScript

- **Web Development:** Adding interactivity and behaviour to static sites JavaScript was invented to do this in 1995. By using AngularJS that can be achieved so easily.
- **Web Applications:** With technology, browsers have improved to the extent that a language was required to create robust web applications. It uses Application Programming Interfaces(APIs) that provide extra power to the code. The Electron and React are helpful in this department.
- **Server Applications:** With the help of Node.js, JavaScript made its way from client to server and Node.js is the most powerful on the server side.
- **Games:** Not only in websites, but JavaScript also helps in creating games. The combination of JavaScript and HTML 5 makes JavaScript popular in game development as well. It provides the EaseJS library which provides solutions for working with rich graphics.
- **Smartwatches:** JavaScript is being used in all possible devices and applications. It provides a library PebbleJS which is used in smartwatch applications. This framework works for applications that require the Internet for their functioning.
- **Art:** Artists and designers can create whatever they want using JavaScript to draw on HTML 5 canvas, and make the sound more effective also can be used **p5.js** library.
- **Machine Learning:** This JavaScript ml5.js library can be used in web development by using machine learning.
- **Mobile Applications:** JavaScript can also be used to build an application for non-web contexts. The features and uses of JavaScript make it a powerful tool for creating mobile applications. Using React Native, we can build mobile applications for different operating systems.

JavaScript Example

```
<script>
    document.write("Welcome to NIIT Nanded");
</script>
```

1.2 The Development Workflow in JavaScript

The development workflow in JavaScript can vary based on the project, team preferences, and tools in use. However, there are common practices and tools that many JavaScript developers use to ensure efficient development, testing, and deployment. Here's a general overview of a typical JavaScript development workflow:

1. **Project Setup**
   - **Initialization**: Use package managers like **nmp** (Node Package Manager) or **yran** initialize a new project.
   - **Version Control**: Initialize a git repository.
   - **Folder Structure**: Organize your project files into a structured directory, often following the MVC (Model-View-Controller) or similar patterns.
2. **Development**:
   - **Code Editor**: Use a text editor or an Integrated Development Environment (IDE) like Visual Studio Code, WebStorm, or Atom.
   - **Version Control**: Commit your changes regularly to Git and push to a remote repository (e.g., GitHub, GitLab, Bitbucket).
3. **Testing**:
   - **Unit Testing**: Use testing frameworks like Jest, Mocha, or Jasmine to write and run unit tests for your JavaScript code.
   - **Integration & End-to-End Testing**: Use tools like Cypress, Selenium, or Puppeteer for integration and end-to-end testing.
4. **Linting & Formatting**:
   - Use tools like ESLint, Prettier, or TSLint (for TypeScript projects) to enforce coding standards, catch errors, and ensure consistent code formatting.
   - Integrate linting and formatting tools into your code editor or build process.
5. **Build & Bundling**:

- Use bundlers like Webpack, Rollup, or Parcel to bundle your JavaScript modules, optimize assets, and generate production-ready builds.
- Transpile modern JavaScript (ES6+) to compatible ES5 code using Babel.

6. **Development Server**:
   - Use development servers like Webpack Dev Server, Parcel, or Create React App's development server for hot module replacement and fast feedback during development.
7. **Documentation**:
   - Maintain documentation using tools like JSDoc, Markdown, or tools integrated with your code editor/IDE.
8. **Performance Optimization**:
   - Profile and optimize your JavaScript code using browser developer tools, tools like Lighthouse, or third-party monitoring solutions.
   - Use code-splitting, lazy loading, and other optimization techniques to improve loading and runtime performance.
9. **Deployment:**
   - Deploy your JavaScript applications to hosting platforms like Vercel, Netlify, AWS, Heroku, or traditional web servers.
   - Ensure proper environment configuration, security measures, and monitoring in the production environment.

Remember, the specific tools and practices in a JavaScript development workflow can vary based on the project's requirements, team preferences, and the ecosystem's evolution. Regularly updating your skills and staying informed about the latest tools and best practices is essential for efficient and effective development.

1.3 Selecting The Right Tools For The Job

Selecting the right tools for JavaScript development is crucial for ensuring productivity, maintainability, and the successful delivery of projects. The JavaScript ecosystem offers a wide range of tools, libraries, and frameworks tailored for various purposes and requirements. Here's a systematic approach to help you select the right tools for JavaScript development:

1. **Define Project Requirements**:
   - **Type of Application**: Determine whether you're building a web application, mobile application, desktop application, or server-side application.

- **Functionalities & Features**: List down the essential functionalities, features, and technical requirements of the project.
- **Performance & Scalability**: Assess the expected performance metrics and scalability requirements of the application.

2. **Choose the Right Framework**:
   - **Web Development**:
     - **React**: For building interactive UIs and single-page applications.
     - **Angular**: For building robust and scalable applications with a comprehensive framework.
     - **Vue.js**: For building progressive and lightweight applications with a flexible ecosystem.
   - **Node.js**: For building scalable and efficient server-side applications using JavaScript.
   - **Mobile Development**:
     - **React Native**: For building cross-platform mobile applications with native performance.
     - **Flutter**: For building mobile applications with a rich set of customizable widgets and native performance.
   - **Electron**: For building desktop applications using web technologies.
3. **Package Management**:
   - **npm (Node Package Manager)**: For managing project dependencies, scripts, and publishing packages.
   - **yarn**: An alternative to npm with performance optimizations and improved caching.
4. **Build & Bundling Tools**:
   - **Webpack**: For bundling JavaScript modules, assets optimization, and generating production builds.
   - **Rollup**: For creating smaller, faster, and more efficient bundles.
   - **Parcel**: A zero-configuration bundler for fast and simple builds.
5. **Transpilers & Compilers**:
   - **Babel**: For transpiling modern JavaScript (ES6+) to compatible ES5 code and supporting experimental features.
6. **Linting & Code Quality**:

- **ESLint**: For linting JavaScript code, enforcing coding standards, and identifying potential errors or issues.
- **Prettier**: For code formatting and ensuring consistent code style across the project.

7. **Testing Frameworks & Libraries**:
   - **Jest**: For testing JavaScript code with a focus on simplicity and support for various testing methodologies.
   - **Mocha**: A flexible testing framework with support for various assertion libraries and testing styles.
   - **Cypress**: For end-to-end testing of web applications with a focus on simplicity and real-time feedback.
8. **State Management & Data Handling**:
   - **Redux**: For managing application state and data flow in React applications.
   - **Vuex**: For managing state in Vue.js applications with a centralized store.
   - **GraphQL**: For querying and manipulating data with a flexible and efficient API.
9. **Development Environment & Tools**:
   - **Visual Studio Code**: A popular code editor with extensive support for JavaScript development and a rich ecosystem of extensions.
   - **Node.js**: A JavaScript runtime for executing JavaScript server-side and building scalable applications.
10. **Version Control & Collaboration**:
    - **Git**: For version control, collaboration, and managing codebase changes.
    - **GitHub, GitLab, Bitbucket**: Platforms for hosting Git repositories, managing projects, and facilitating collaboration.
11. **Deployment & Hosting**:
    - **Vercel, Netlify**: For deploying and hosting static websites, single-page applications, and serverless functions.
    - **Heroku, AWS, Azure**: Cloud platforms for deploying and scaling applications with support for various server configurations and services.

By aligning the selection of tools with the project's requirements, objectives, and technical constraints, you can build robust, scalable, and maintainable JavaScript applications. Regularly updating and optimizing your toolset in response to evolving project needs, technological advancements, and feedback from the development team will help ensure efficient development processes and successful project outcomes.

1.4 Just Enough HTML and CSS

Using HTML and CSS within JavaScript can be beneficial for creating dynamic user interfaces, generating content, or manipulating styles based on various conditions or user interactions. Here's a guide to incorporating just enough HTML and CSS within JavaScript.

HTML:

HTML is mark-up language used for structuring web pages, uses tags to perform different operation in a structure. Some of the important HTML tags for JavaScript

1. Lists: Lists are a fundamental part of HTML and CSS. They are used to create a structure for your content and can be styled to match your website's design. List items can contain other HTML elements, including links, images, and even other lists.

2. Tables: Tables are a great way to present information in a structured way. They can be used to display tabular data, or to align elements on a page. Tables can be created using the table element, which contains the table row (tr) and table header (th) elements.

3. Forms: Forms are one of the most important elements in web design. They allow users to input data, which can then be processed by a server-side script. Forms can be used to log in to users to their accounts, search for data, and much more.

4. Links: Creating links is a very important part of web development, as they allow users to **navigate between pages on a website.** The <a> tag defines a hyperlink, which is used to link from one page to another. The href attribute specifies the URL of the page that the link should point to.

5. Images: The very common method everyone knows is by adding **img tag with src attribute.** To add an image to your HTML, you'll need to use the <img> tag. This tag tells the browser that you want to add an image to the page. The <img> tag has two important attributes: src and alt.

6. Meta tag: The most underrated and ignored tags are meta tags. People skip adding meta tags because there are too many. They help search engines understand what your website is about, and they can also be used to improve your website's SEO.Meta tags are placed in the <head> section of your HTML code.

7. HTML5 Semantics: In recent years, there has been a major update to HTML with the release of HTML5. This new version of HTML includes new features that make **coding web pages more semantic**. Semantic code is code that is easy for both humans and machines to read and understand. some schematic tags are Header, Footer, Section, Aside, Article, etc.

CSS:

On other side CSS is used to style web page element, including HTML tags. It enable you to give a different style to each element , including height, width, colors, fonts, CSS unit, FlexBox, Grids, position, Border etc. these properties will help to create stylish and beautiful web pages.

1. **Height/Width:** The height and width properties specify the height and width of a HTML element respectively.
2. **Color:** Perhaps the most common property is color. the color property allows us to specify the text color of an HTML element.
3. **Font-Size:** The font-size property allows us to change the size of the font for any text based content.
4. **Background Color:** The background-color property allows you to set the background color of an HTML element.
5. **Border:** The border property allows you set a border for your HTML element.
6. **Background Image:** The background-image property allows us to add a background image to an HTML element. This is similar to the background color, however an image is shown instead of a color.

1.5 Understanding Objects

In JavaScript, objects are one of the fundamental data structures, allowing you to store and organize data using key-value pairs. Each key value pair is called property. Understanding objects is essential for effective JavaScript programming.

1. **Object Basics:**

Definition: An object is a collection of properties, where each property consists of a key (or name) and a value.

Literal Notation: Objects can be created using object literal notation.

```
const person = {
    firstName: 'NIIT',
    lastName: 'Nanded',
    age: 30,
```

```
    greet: function() {
     console.log('Hello, ' + this.firstName);
    }
   };
```

2. **Accessing Properties**: To access the properties of an object we can use two notation

   Dot Notation: Use dot notation to access properties.

```
console.log(person.firstName);  // Output: NIIT
```

Bracket Notation: Use bracket notation for dynamic property access or when the property name is not a valid identifier.

```
console.log(person['lastName']);  // Output: Nanded
```

3. Adding & Modifying Properties:

Adding Properties:

```
person.email = 'nandigramiit@gmail.com';
```

Modifying Properties:

```
person.age = 35;
```

4. Methods:

- Objects can contain methods (functions) as properties.
- Methods can access the object's properties using the **this** keyword.

```
person.greet();  // Output: Hello, NIIT
```

1.6 Understanding Variable in JavaScript

In JavaScript, variables are used to store and manipulate data. Understanding variables is fundamental to programming in JavaScript, as they enable you to work with values, perform calculations, and store information. JavaScript Variables can be declared in 4 ways: Automatically, using var, let, const.

```
var name = 'NIIT'; //The var was used in all JavaScript code from 1995 to
2015.
let age = 25;       // Only use let if you can't use const
const PI = 3.14;    //  Always use const if the value should not be changed
```

JavaScript Variable Scope: The scope of a variable is the region of your program in which it is defined. JavaScript variable will have only two scopes.

- **Global Variables:** A global variable has global scope which means it is defined everywhere in your JavaScript code.
- **Local Variables:** A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.

Identifier Rules:

- You should not use any of the JavaScript reserved keyword as variable name. For example, break or boolean variable names are not valid.
- JavaScript variable names should not start with a numeral (0-9).
- They must begin with a letter or the underscore (_) character. For example, 123test is an invalid variable name but _123test is a valid one.
- JavaScript variable names are case sensitive. For example, Name and name are two different variables.

1.7 Making Comparison

- JavaScript supports comparison operators to compare two values.
- The JavaScript comparison operators take two values, compare them, and return a Boolean result, either true or false.
- These operators are very useful in decision-making and loop programs in JavaScript. The following is a list of the comparison operators supported by JavaScript:

Operator	Name	Example	Return Value
==	Equal to	a==b	True or false
		a==3	True or false
		a=="3"	True or false
===	Equal value and types	a===b	True or false
		a====="3"	True or false
!=	Not equal to	a!=b	True or false
!==	Not equal to and types	a!==b	True or false
		a!=="b"	True or false
>	Greater than	a>b	True or false
<	Less than	a<b	True or false
>=	Greater than or equal to	a>=b	True or false
<=	Less than or equal to	a<=b	True or false

Logical Operators in JavaScript:

- The logical operators can be bound with the comparison operators to create various use cases. Let's see the available logical operators in JavaScript:
- The following are the logical operators supported by JavaScript:

Operator	Name	Example
&&	Logical And	(a< 5 && b>2)
||	Logical Or	(a<5 || b>2)
!	Not	(a!=5)

Ternary (Conditional) Operator:

JavaScript also supports a conditional operator, known as the ternary operator, specifying whether the output is true or not based on a condition. It is represented by the **"?"** symbol.

Syntax:

```
Variable_name = (condition) ? value1:value2
```

Example:

```
<script>
    var a=10;
    var b=20;
    c=a>b? a:b;
    document.write("Largest Number=" + c);
</script>
```

1.8 Understanding Event

- Events are “action” or "things" that is performed by user on an HTML elements or tags such as button, text, headings etc.
- When JavaScript is used in HTML pages, JavaScript can "react" or “responds” on these events that is called event Handler.
- This process of reacting over the events is called Event Handling. Thus, js handles the HTML events via Event Handlers.
- An event can be something or actions the browser does, or something a user does.
- Here are some examples of HTML events:
- An HTML web page has finished loading
- An HTML input field was changed
- An HTML button was clicked
- Often, when events happen, you may want to do something or expects some things.
- HTML allows event handler attributes, with JavaScript code, to be added to HTML elements

Syntax:

```
<element event='some JavaScript'>
```

Example:

```
<button onclick="document.getElementById('demo').innerHTML=
Date()">The time is?
</button>
```

- Here is a list of some common HTML events:

Mouse Event:

Event Performed	Event Handler	Description
click	onclick	When mouse click on an element
mouseover	onmouseover	When the cursor of the mouse comes over the element
mouseout	onmouseout	When the cursor of the mouse leaves an element
mousedown	onmousedown	When the mouse button is pressed over the element
mouseup	onmouseup	When the mouse button is released over the element
mousemove	onmousemove	When the mouse movement takes place.

Keyboard Event:

Event Performed	Event Handler	Description
Keydown & Keyup	onkeydown & onkeyup	When the user press and then release the key

Form Event

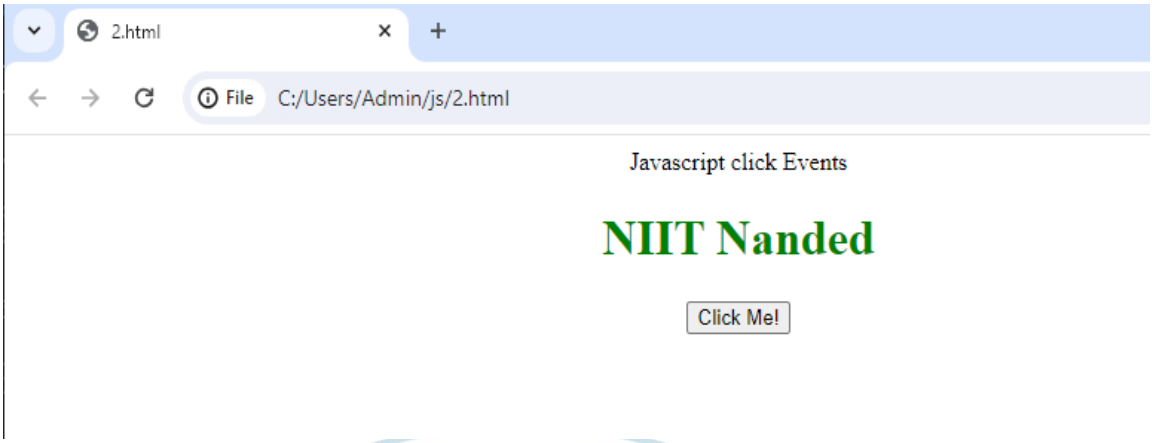
Event Performed	Event Handler	Description
focus	onfocus	When the user focuses on an element
submit	onsubmit	When the user submits the form
blur	onblur	When the focus is away from a form element
change	onchange	When the user modifies or changes the value of a form element

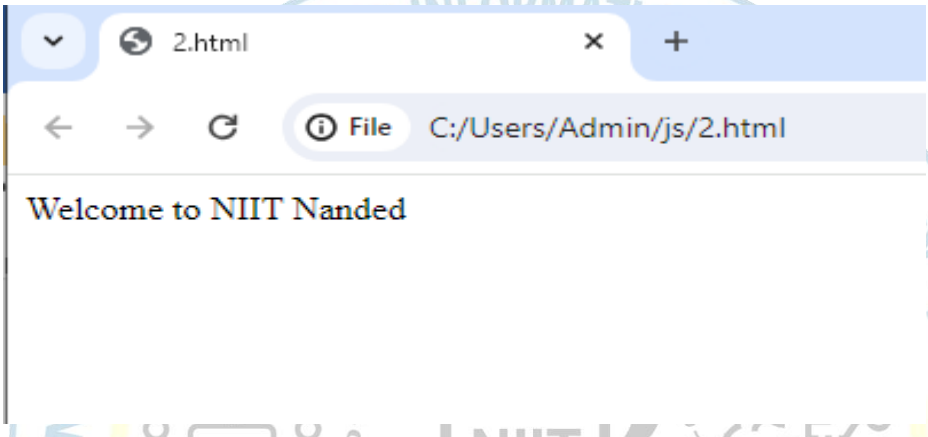
Window/Document events

Event Performed	Event Handler	Description
load	onload	When the browser finishes the loading of the page
unload	onunload	When the visitor leaves the current webpage, the browser unloads it
resize	onresize	When the visitor resizes the window of the browser

Click Event Example:

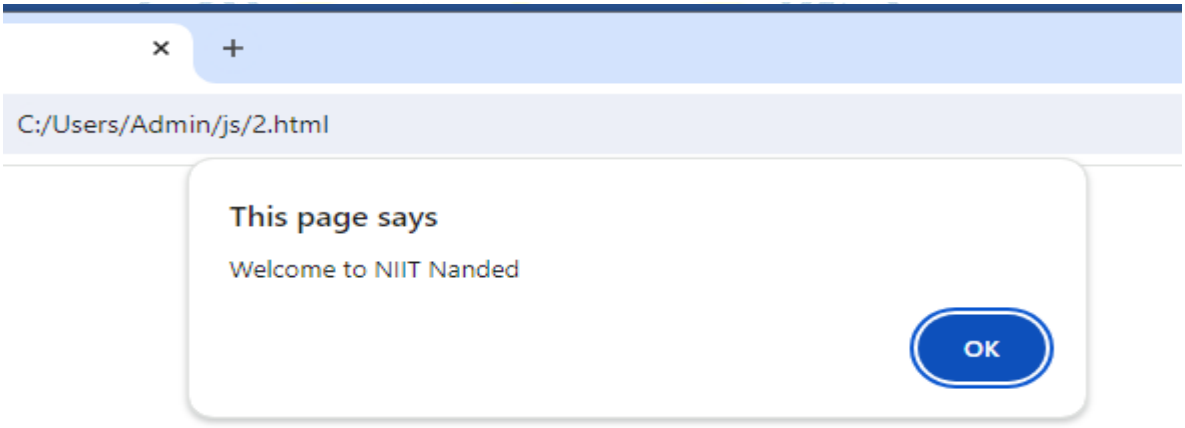
```
<html>
 <head> Javascript click Events </head>
 <body style="text-align: center;">
    <h1 style="color:green;">
        NIIT Nanded
    </h1>
<script language="Javascript" type="text/Javascript">
    function clickevent()
        {
        document.write("Welcome to NIIT Nanded");      }
</script>
<form>
<input type="button" onclick="clickevent()" value="Click Me!"/>
</form>
</body>
</html>
```





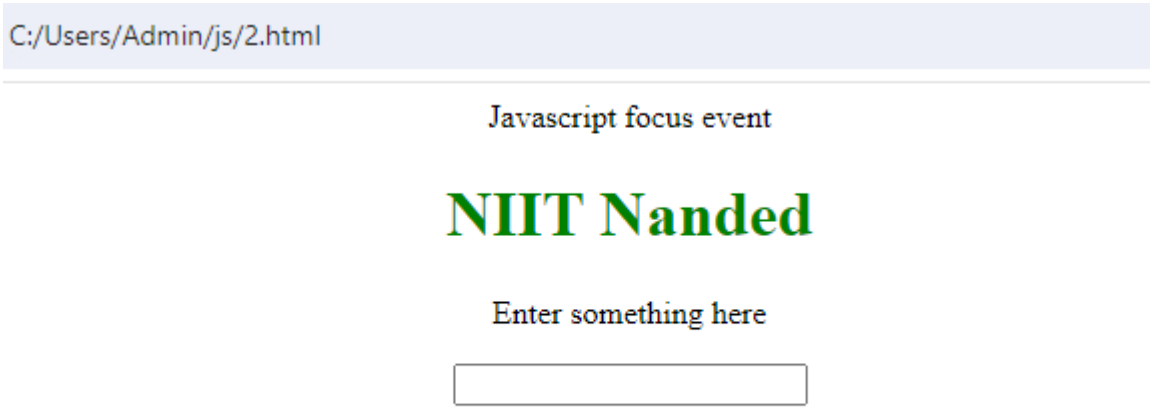
MouseOver Event Example:

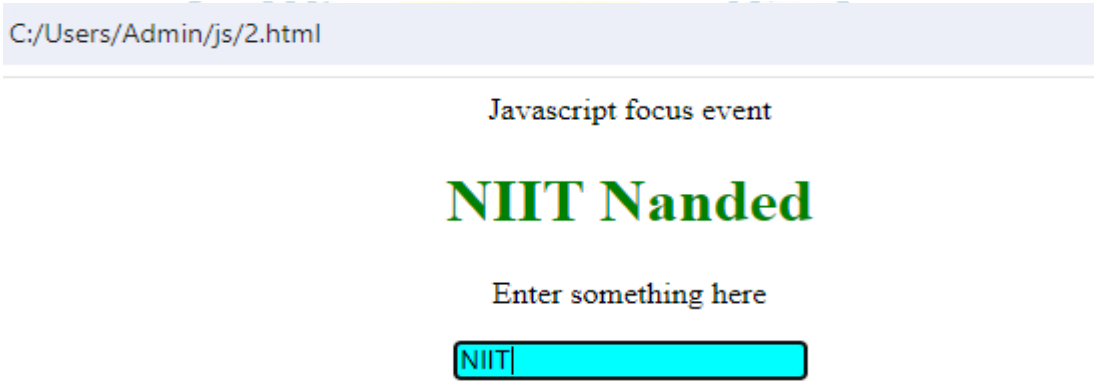
```
<html>
<head>
</head>
<body style="text-align: center;">
    <h1 style="color:green;">
        NIIT Nanded
    </h1>
    <h2> Javascript mouseover event </h2>
    <script language="Javascript" type="text/Javascript">

        function mouseoverevent()
        {
        alert("Welcome to NIIT Nanded");
    }
</script>
<p onmouseover="mouseoverevent()"> Keep cursor over me</p>
</body>
</html>
```



Focus Event Example:

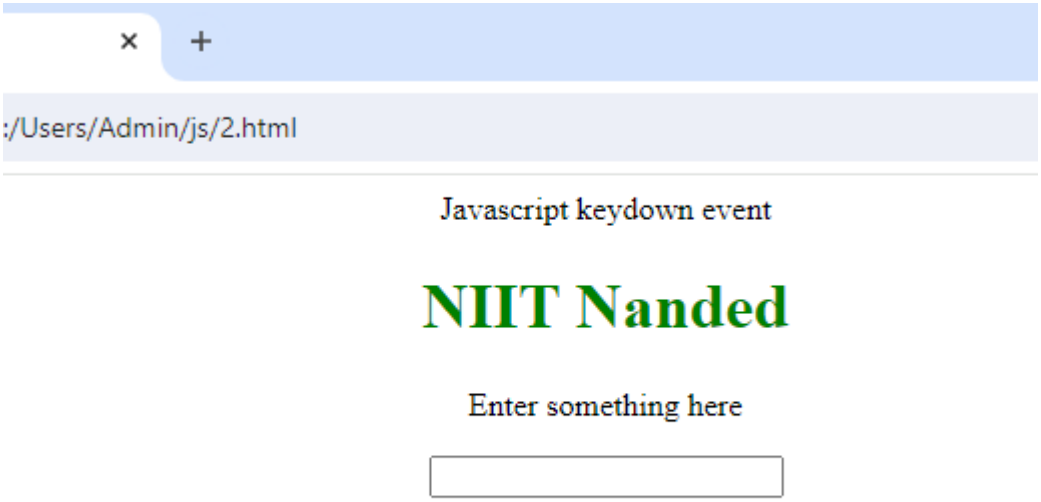
```
<html>
<head> Javascript focus event </head>
<body style="text-align: center;">
    <h1 style="color:green;">
        NIIT Nanded
    </h1>
    <p>Enter something here</p>
    <input type="text" id="input1" onfocus="focusevent()"/>
    <script>
        function focusevent()
        {
         document.getElementById("input1").style.background="aqua";
        }
     </script>
</body>
</html>
```

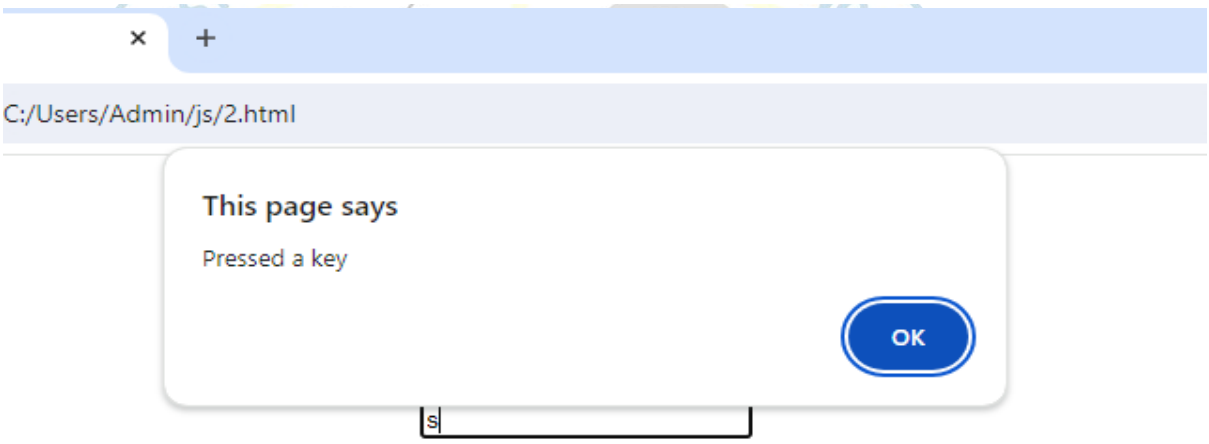




Keydown Event Example:

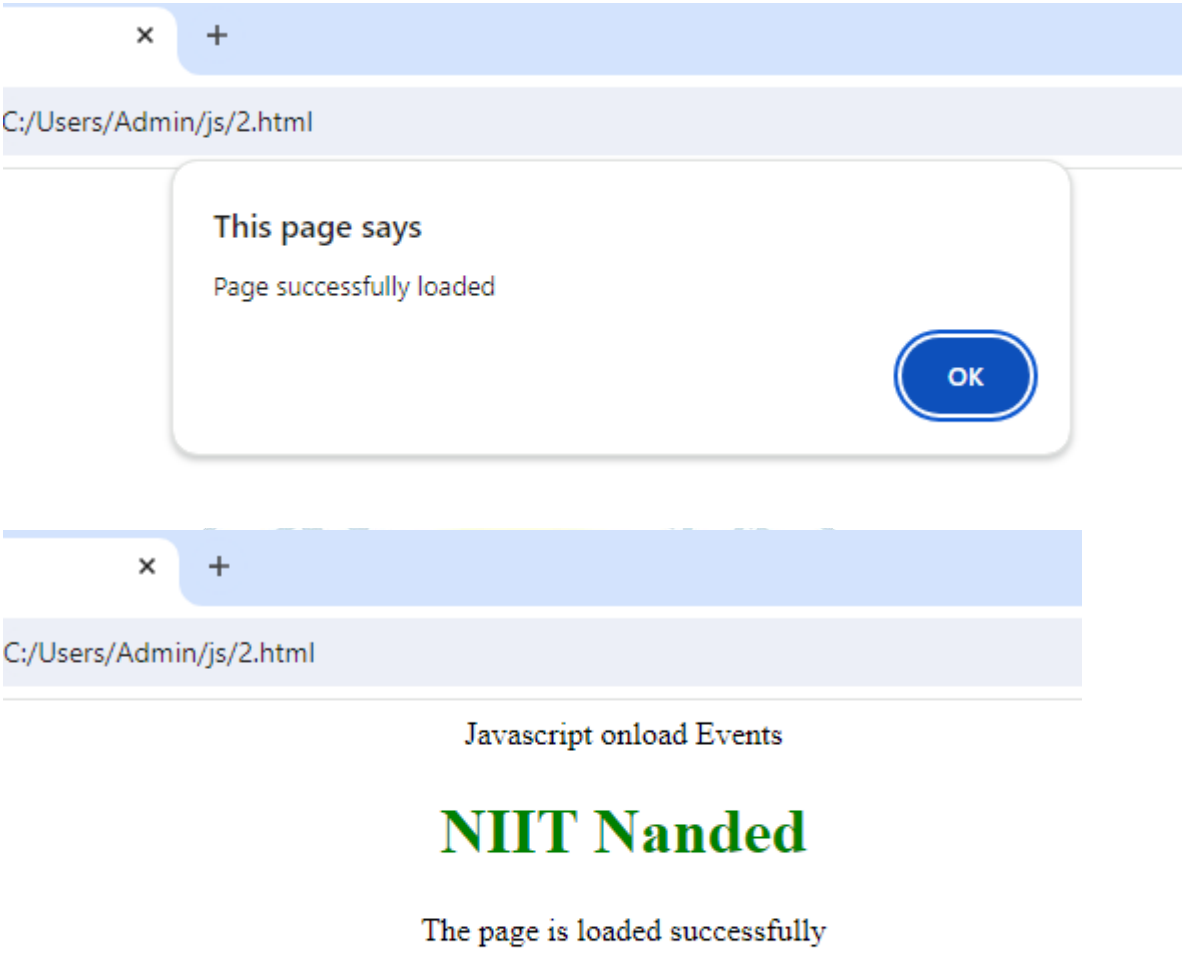
```
<html>
<head> Javascript keydown event</head>
    <body style="text-align: center;">
    <h1 style="color:green;">
        NIIT Nanded
    </h1>
    <p> Enter something here</p>
    <input type="text" id="input1" onkeydown="keydownevent()"/>
    <script>
        function keydownevent()
        {
       document.getElementById("input1");
        alert("Pressed a key");
       }
 </script>
</body>
</html>
```





Load event Example:

```
<html>
<head>Javascript onload Events</head>
</br>
   <body onload="window.alert('Page successfully loaded');"
style="text-align: center;">
     <h1 style="color:green;">
        NIIT Nanded
    </h1>
     <script>
       document.write("The page is loaded successfully");
    </script>
   </body>
</html>
```



1.9 Writing Your First Script

- JavaScript provides three places to put the JavaScript code:

  1. within body tag,

  2. within head tag and

  3. external JavaScript file.

- The basic syntax and example of writing the first script is as below.

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

- The script tag specifies that we are using JavaScript.
- The text/javaScript is the content type that provides information to the browser about the data.
- The document. write () function is used to display dynamic content through JavaScript. We will learn about document object in detail later.
- Java Script Code can also be written directly in to HTML TAG attribute.

Code Between the Body Tag

```
<html>
<body>
<script type="text/javascript">
document. write ("Hello JavaScript");
</script>
</body>
</html>
```

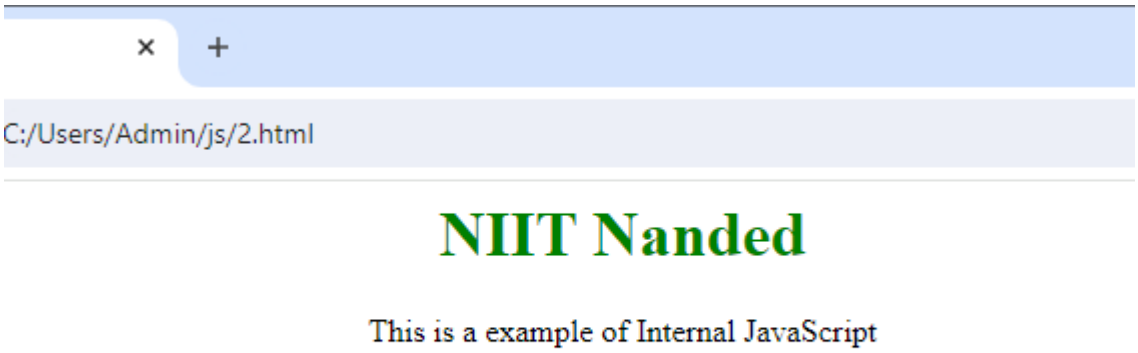
Code Between The Head Tag:

```
<html>
<head>
   <script type="text/javascript">
   function msg()
   {
    alert ("Welcome to NIIT Nanded");
   }
  </script>
</head>
<body style="text-align: center;">
  <h1 style="color:green;">
      NIIT Nanded
  </h1>
    <p>Welcome to JavaScript</p>
   <form>
   <input type="button" value="click" onclick="msg()"/>
   </form>
</body>
</html>
```

1.10 Internal Vs. External Script

- There are 2 ways to use the javascript in the HTML file:
- **Internal JavaScript:** JavaScript can be added directly to the HTML file by writing the code inside the <script> tag. We can place the <script> tag either inside <head> or the <body> tag according to the need.

```
<html>
<head>
    <title>Internal JS</title>
</head>

<body style="text-align: center;">
    <h1 style="color:green;">
        NIIT Nanded
    </h1>
    <script>
    /*Internal Javascript*/
   document.write("This is a example of Internal JavaScript");
    </script>
</body>

</html>
```



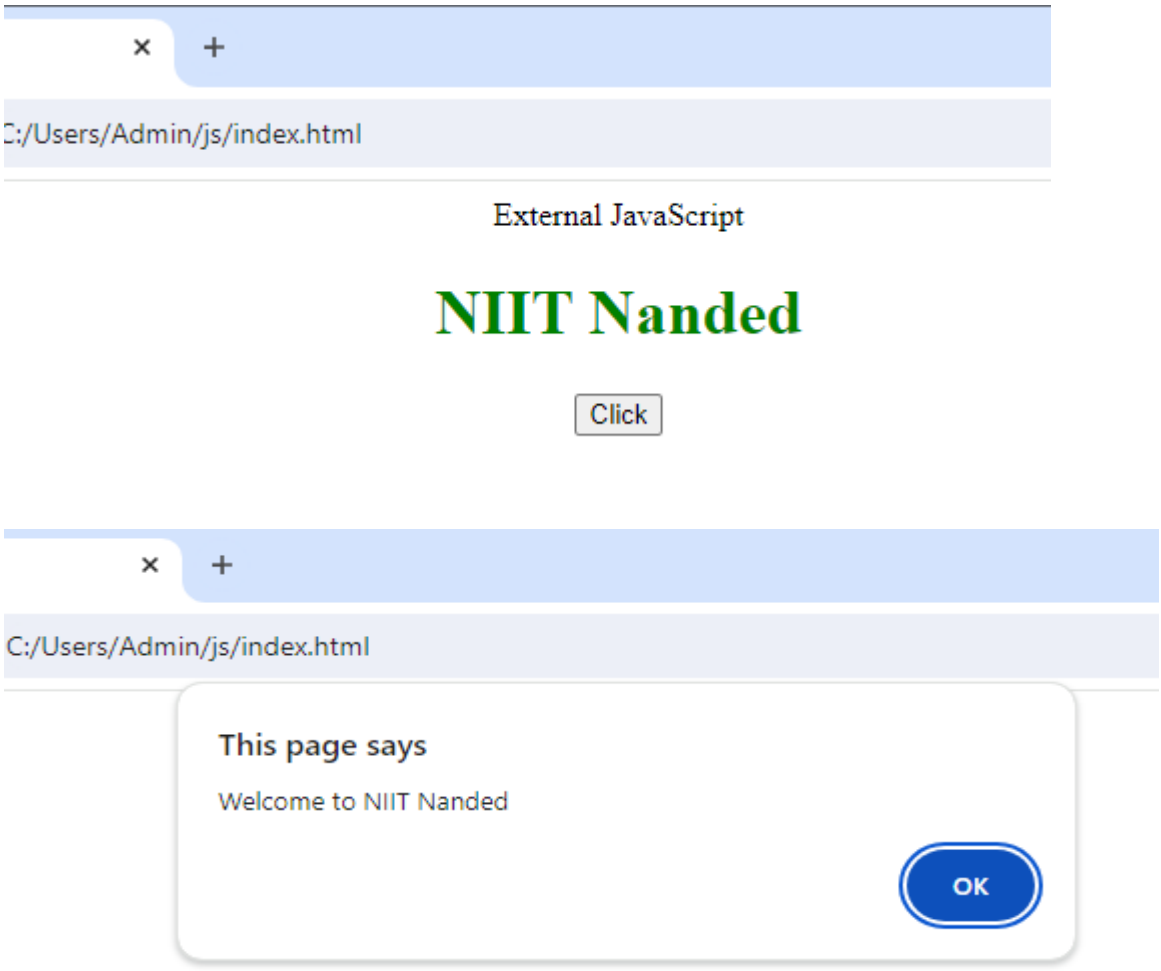
- **External JavaScript:** The other way is to write JavaScript code in another file having a .js extension and then link the file inside the <head> or <body> tag of the HTML file in which we want to add this code.
- Following example create external JavaScript file that prints Welcome to NIIT in a alert - dialog box.
- In the following is the external.js file which is javaSript file included in html code which is index.html

external.js

```
function msg(){
    alert("Welcome to NIIT Nanded");
   }
```

index.html

```
<html>
<head>
  <p> External JavaScript</p>
</head>
<body style="text-align: center;">
    <h1 style="color:green;">
        NIIT Nanded
    </h1>
    <script type="text/javascript" src="external.js"></script>
<form>
<input type="button" value="Click" onclick="msg()"/>
</form>
</body>
</html>
```



Advantages of External JavaScript

- It helps in the reusability of code in more than one HTML file.
- It allows easy code readability.
- It is time-efficient as web browsers cache the external js files, which further reduces the page loading time.
- It enables both web designers and coders to work with html and js files parallelly and separately, i.e., without facing any code conflictions.
- The length of the code reduces as only we need to specify the location of the js file.

Disadvantages of External JavaScript

- The stealer may download the coder's code using the url of the js file.
- If two js files are dependent on one another, then a failure in one file may affect the execution of the other dependent file.
- The web browser needs to make an additional http request to get the js code.

- A tiny to a large change in the js code may cause unexpected results in all its dependent files.
- We need to check each file that depends on the commonly created external javascript file.
- If it is a few lines of code, then better to implement the internal javascript code

1.11 Using Comments in Script

- JavaScript comments are used to explain the code to make it more readable.
- It can be used to prevent the execution of a section of code if necessary.
- The JavaScript comment is ignored by the JavaScript engine i.e. embedded in the browser.
- To make code easy to understand. It can be used to elaborate the code so that end user 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 need to disable the code. In such case, it is better to use comments.
- There are two types of comments in JavaScript.
  1.Single-line Comment
  2. Multi-line Comment

JavaScript Single line Comment

- It is represented by double forward slashes (//).
- It can be used before and after the statement.
- Example of single-line comment

```
<script>
var a=10;
var b=20;
var c=a+b; //It adds values of a and b variable
document.write(c); //prints sum of 10 and 20
</script>
```

JavaScript Multi line Comment

- It can be used to add single as well as multi line comments. So, it is more convenient.

- It is represented by forward slash with asterisk (/*) then asterisk with forward slash(*/).

  Syntax : /* your code here */

  Example:

```
<script>
/* It is multi line comment.
It will not be displayed */
document.write("example of javascript multiline comment");
</script>
```

1.12 Using the noScript

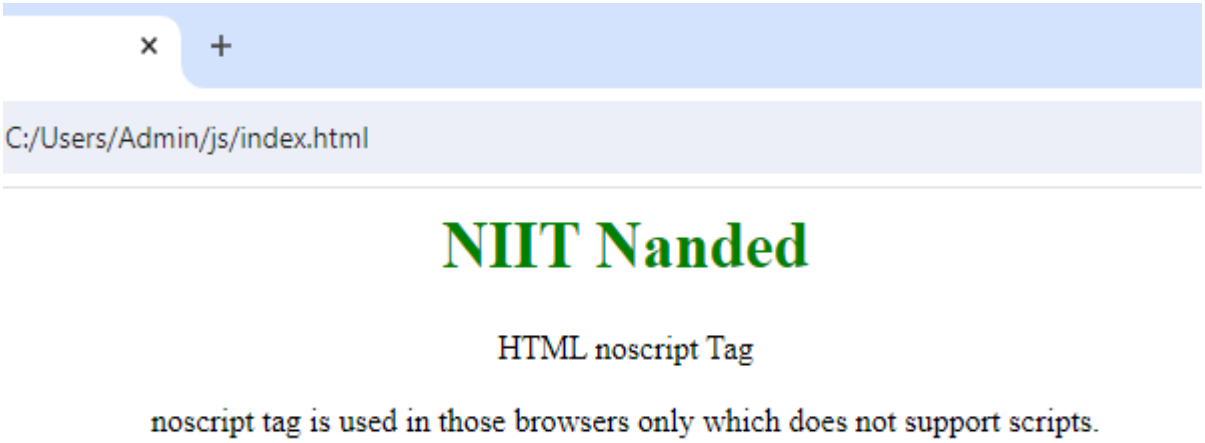
- The <noscript> tag defines an alternate content to be displayed to users that have disabled scripts in their browser or have a browser that doesn't support script.
- The <noscript> element can be used in both <head> and <body>.
- When used inside <head>, the <noscript> element could only contain <link>, <style>, and <meta> elements.
- This tag is used in those browsers only which does not support scripts.

  Syntax:

```
<noscript> Contents... </noscript>
```

  Example:

```
<html>
    <body style="text-align: center;">
        <h1 style="color:green;">
            NIIT Nanded
        </h1>
    <p>HTML noscript Tag</p>
    <script>
        document.write("nosript tag is used in those browsers only
which does not support scripts.")
    </script>
    <!-- noscript tag starts -->
    <noscript>A computer science portal</noscript>
    <!-- noscript tag ends -->
</body>
</html>
```



1.13 Creating Alert Dialogs

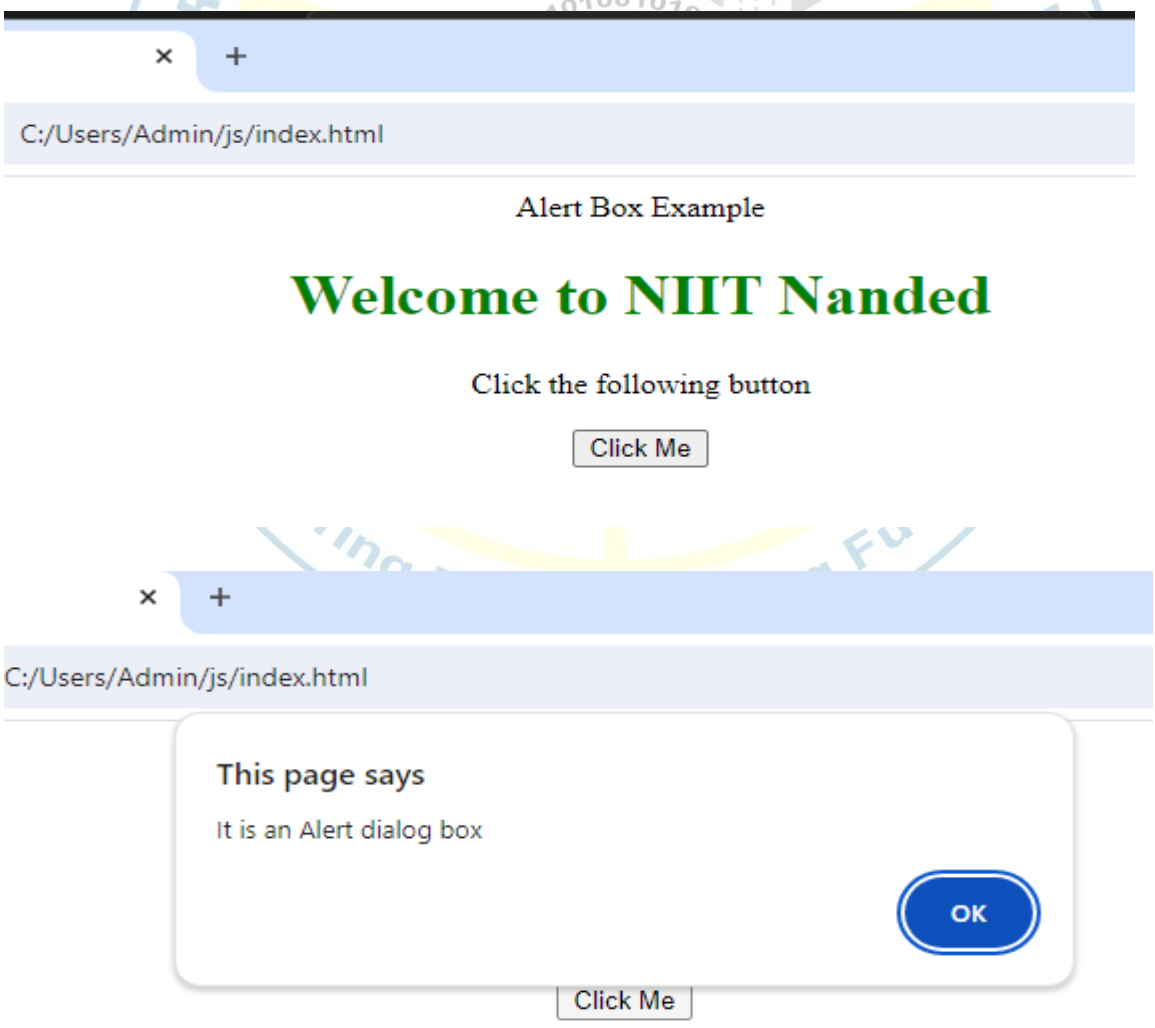
- An alert dialog box is mostly used to give a warning message to the users.
- Alert box gives only one button "OK" to select and proceed.
- For example, if one input field requires to enter some text but the user does not provide any input, then as a part of validation, you can use an alert box to give a warning message.
- Rather than showing the warnings or errors, the alert dialog box can be used for normal messages such as 'welcome NIIT', 'Hello NIIT', etc.

Syntax:

```
window.alert("sometext");
        OR
alert("sometext"); // method can be written without the window prefix.
```

Example:

```
<html>
<head>
    <script type="text/javascript">
        function show() {
            alert("It is an Alert dialog box");
            }
    </script>
</head>
<body style="text-align: center;">
    <p>Alert Box Example</p>
    <h1 style="color:green;">
        Welcome to NIIT Nanded
    </h1>
        <p>Click the following button </p>
        <input type="button" value="Click Me" onclick="show();"/>
</body>
</html>
```



1.14 Understanding Conditional Statement

- Conditional statements in JavaScript allow you to execute specific blocks of code based on conditions. If the condition meets then a particular block of action will be executed otherwise it will execute another block of action that satisfies that particular condition.
- There are several methods that can be used to perform Conditional Statements in JavaScript.
    1. if Statement
    2. if-else Statement
    3. if….else if….statement
    4. nested if statements

1. The if statement

It is one of the simplest decision-making statement which is used to decide whether a block of JavaScript code will execute if a certain condition is true.

Syntax:

```
if (condition)
{
  //  block of code to be executed if the condition is true
}
```

If the condition evaluates to true, the code within if statement will execute, but if the condition evaluates to false, then the code after the end of if statement (after the closing of curly braces) will execute.

Example:

```
let num = 20;

if (num % 2 === 0) {
    console.log("Given number is even number.");
}

if (num % 2 !== 0) {
    console.log("Given number is odd number.");
};
```

2. The if….else statement

The if….else statement includes two blocks that are if block and else block. It is the next form of the control statement, which allows the execution of JavaScript in a more controlled way. It is used when you require to check two different conditions and execute a different set of codes. The else statement is used for specifying the execution of a block of code if the condition is false.

Syntax:

```
if (condition)
{
  //  block of code to be executed if the condition is true
}
else
{
  //  block of code to be executed if the condition is false
}
```

If the condition is true, then the statements inside if block will be executed, but if the condition is false, then the statements of the else block will be executed.

Example:

```
let age = 25;

if (age >= 18) {
    console.log("You are eligible of driving licence")
} else {
    console.log("You are not eligible for driving licence")
};
```

3. The if….else if…..else statement

It is used to test multiple conditions. The if statement can have multiple or zero else if statements and they must be used before using the else statement. You should always be kept in mind that the else statement must come after the else if statements.

Syntax:

```
if (condition1)
{
  //  block of code to be executed if condition1 is true
}
else if (condition2)
{
  //  block of code to be executed if the condition1 is
false and condition2 is true
}
else
{
  //  block of code to be executed if the condition1 is
false and condition2 is false
}
```

Example:

```
const num = 0;

if (num > 0) {
    console.log("Given number is positive.");
} else if (num < 0) {
    console.log("Given number is negative.");
} else {
    console.log("Given number is zero.");
};
```

4. **The nested if statements:**

JavaScript allows us to nest if statements within if statements. i.e, we can place an if statement inside another if statement. A nested if is an if statement that is the target of another if or else.

Syntax:

```
if condition1
{
   // code to be executed if condition1 is true
  if condition2
    {
    // code to be executed if both condition1 and condition2
are true
    }
}
```

Example:

```
let i = 10;

if (i == 10) {  // First if statement
    if (i < 15) {
        console.log("i is smaller than 15");
        // Nested - if statement
        // Will only be executed if statement above
        // it is true
        if (i < 12)
            console.log("i is smaller than 12 too");
        else
            console.log("i is greater than 15");
    }
}
```

1.15 Getting Conformation From Users

- JavaScript confirm method invokes a function that asks the user for a confirmation dialogue on a particular action.
- The confirm () method uses a window object to invoke a dialogue with a question and two option buttons, OK and Cancel. If the user selects the OK option, it will continue to the function execution; selecting the Cancel option will abort the block code's execution.
- It returns true if the user selects the OK option; otherwise, it returns false.

- The JavaScript confirm () method is used to display a specific message on a dialogue window with the OK and Cancel options to confirm the user action
- It is used to accept or verify something.
- It stops all the actions until the confirmation window is closed.

Syntax:

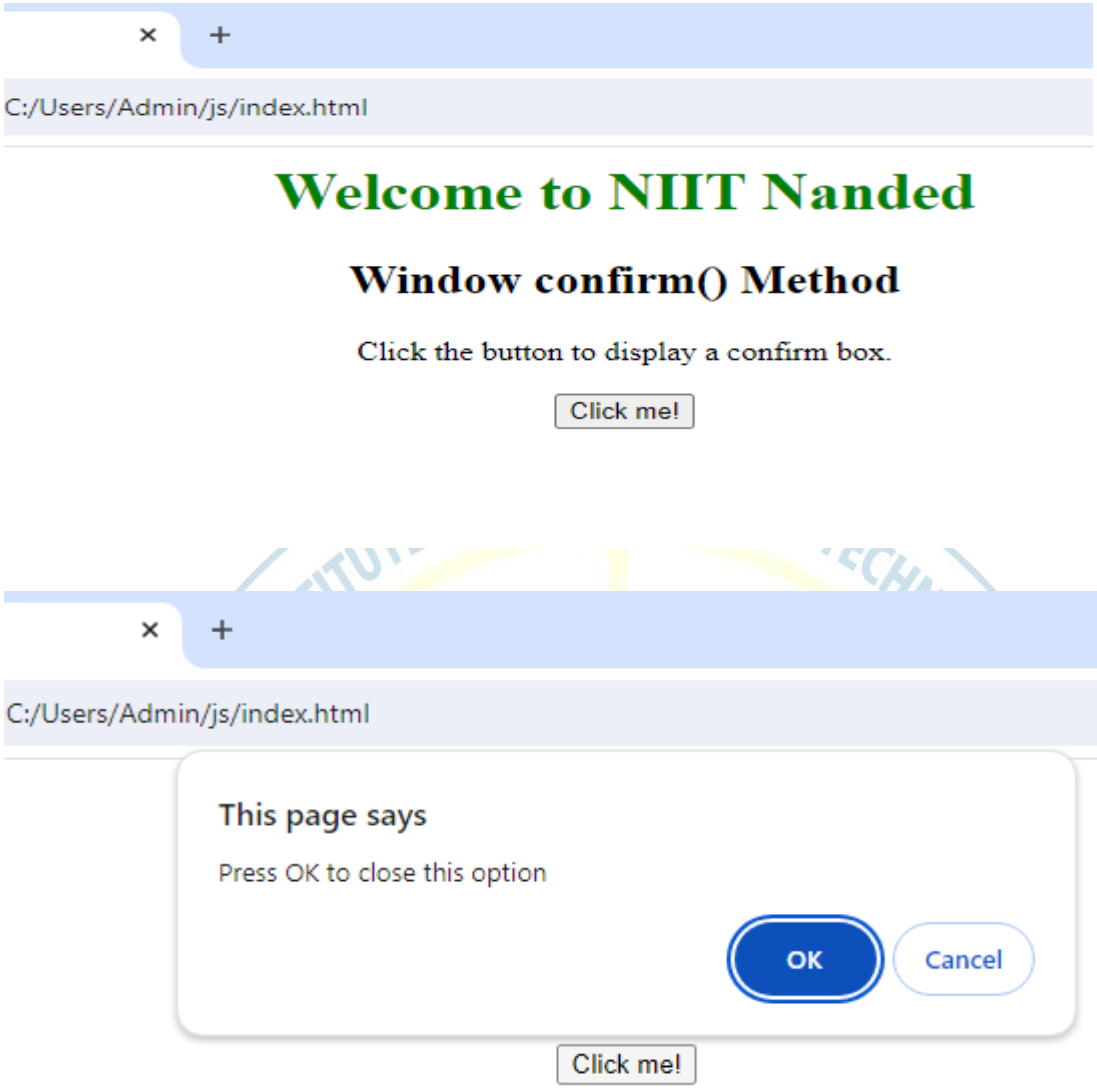
```
confirm(message);
```

Parameters: It takes a "message" value in string format to display in the confirmation dialogue you want to show the user.

Return value: The confirm method returns a Boolean output, either true or false, if the OK is selected. A boolean indicating whether OK (true) or Cancel (false) was selected. If a browser ignores in-page dialogues, then the returned value is always false.

Example:

```
<html>
<head>
    <script type="text/javascript">
        function show() {
            alert("It is an Alert dialog box");
        }
    </script>
</head>
<body style="text-align: center;">
    <h1 style="color:green;">
       Welcome to NIIT Nanded
    </h1>
    <h2>
        Window confirm() Method
    </h2>
    <p>Click the button to display a confirm box.</p>
    <button onclick="callNiit()">Click me!</button>
    <script>
        function callNiit() {
            confirm("Press OK to close this option");
        }
    </script>
</body>
</html>
```



1.16 Creating Prompts For Users

- The prompt() method in JavaScript is used to display a prompt box that prompts the user for the input.
- It is generally used to take the input from the user before entering the page.
- It can be written without using the window prefix.
- When the prompt box pops up, we have to click "OK" or "Cancel" to proceed.
- The box is displayed using the prompt() method, which takes two arguments.
- The first argument is the label which displays in the text box, and the second argument is the default string, which displays in the textbox.
- The prompt box consists of two buttons, OK and Cancel. It returns null or the string entered by the user.

- When the user clicks "OK," the box returns the input value. Otherwise, it returns null on clicking "Cancel".
- The prompt box takes the focus and forces the user to read the specified message. So, it should avoid overusing this method because it stops the user from accessing the other parts of the webpage until the box is closed.

Syntax:

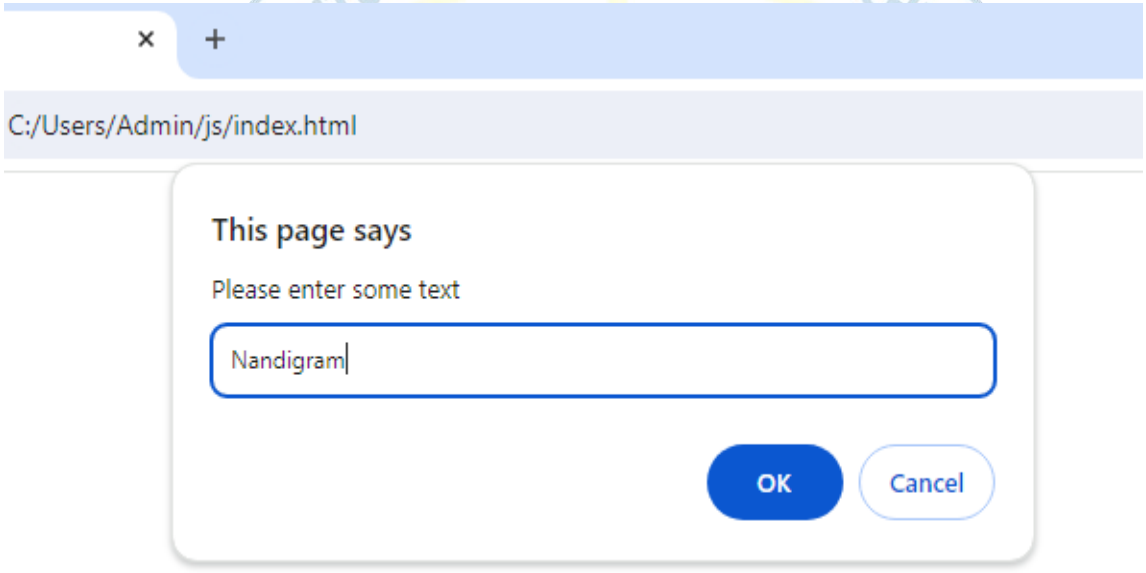
```
prompt(message, default);
          OR
windows.prompt(message, default);
```

- **Message:** is a string of text to display to the user. It can be omitted if there is nothing to show in the prompt window i.e. it is optional.
- **Default:** is a string containing the default value displayed in the text input field. It is also optional.
- **Return value:** The input value is returned if the user clicks the 'OK' button. If not so then null is returned.

Example:

```
<html lang="en">
<head>
    <title>Document</title>
</head>
<body style="text-align: center;">
    <h1 style="color:green;"> NIIT Nanded </h1>
    <h2> Window prompt() Method  </h2>
    <button onclick="callNiit()"> Click me </button>
    <p id="g"></p>
    <script>
        function callNiit() {
            let doc = prompt("Please enter some text");
            if (doc != null) {
                document.getElementById("g").innerHTML ="Welcome to " + doc;
            }
        }
    </script>
</body>
</html>
```


C:/Users/Admin/js/index.html
NIIT Nanded
Window prompt() Method
Click me!


C:/Users/Admin/js/index.html
This page says
Please enter some text
Nandigram
OK
Cancel


C:/Users/Admin/js/index.html
NIIT Nanded
Window prompt() Method
Click me!
Welcome to Nandigram

1.17 Understanding Functions

- A function is the set of input statements, which performs specific computations and produces output.
- It is a block of code that is designed for performing a particular task. It is executed when it gets invoked (or called).
- Functions allow us to reuse and write the code in an organized way.
- Functions in JavaScript are used to perform operations.
- In JavaScript, functions are defined by using function keyword followed by a name and parentheses ().
- The function name may include digits, letters, dollar sign, and underscore.
- The brackets in the function name may consist of the name of parameters separated by commas.
- The body of the function should be placed within curly braces {}.

```
function functionName(parameter1, parameter2, parameter3)

{
  //code to be executed
}
```

- Function parameters are listed inside the parentheses () in the function definition.
- Function arguments are the values received by the function when it is invoked.
- Inside the function, the arguments (the parameters) behave as local variables.

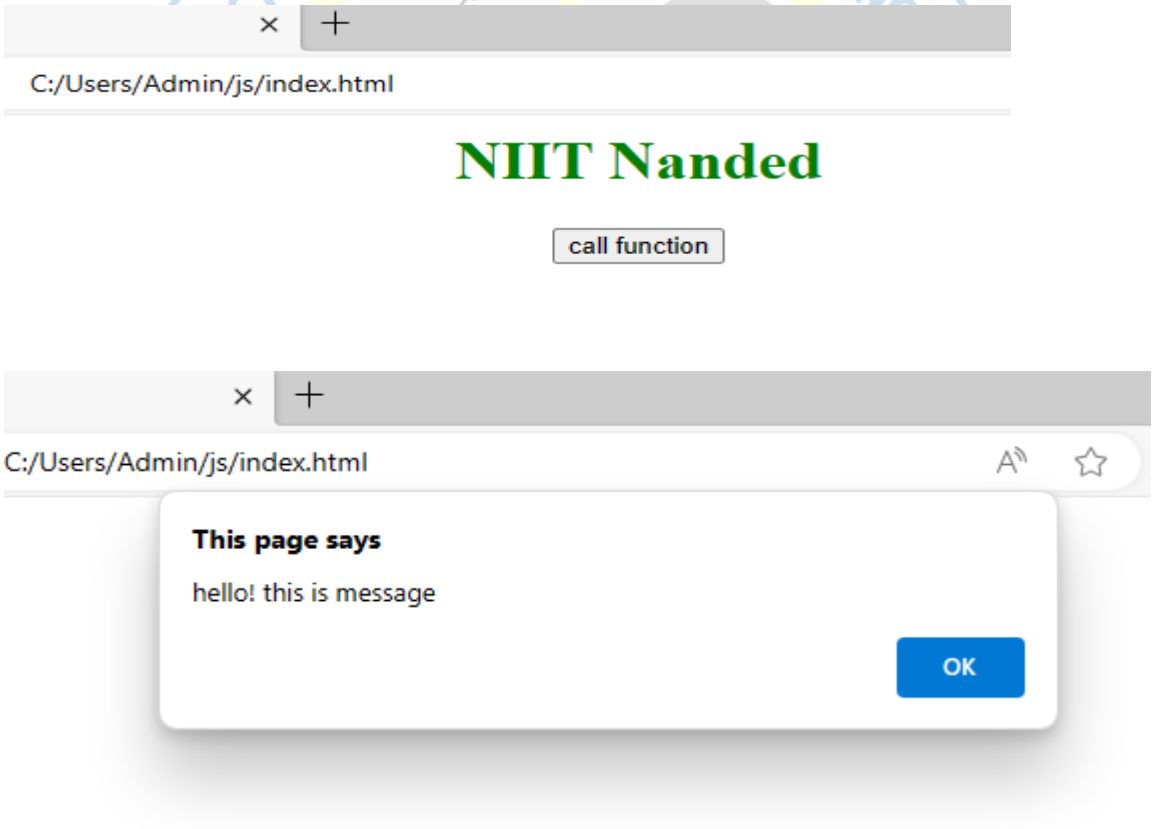
Function Invocation:

The code inside the function will execute when "something" invokes (calls) the function:

- ✓ When an event occurs (when a user clicks a button)
- ✓ When it is invoked (called) from JavaScript code
- ✓ Automatically (self invoked)

Example:

```
<html>
<body style="text-align: center;">
  <h1 style="color:green;">
    NIIT Nanded
  </h1>
<script>
function msg(){
alert("hello! this is message");
}
</script>
<input type="button" onclick="msg()" value="call function"/>
</body>
</html>
```

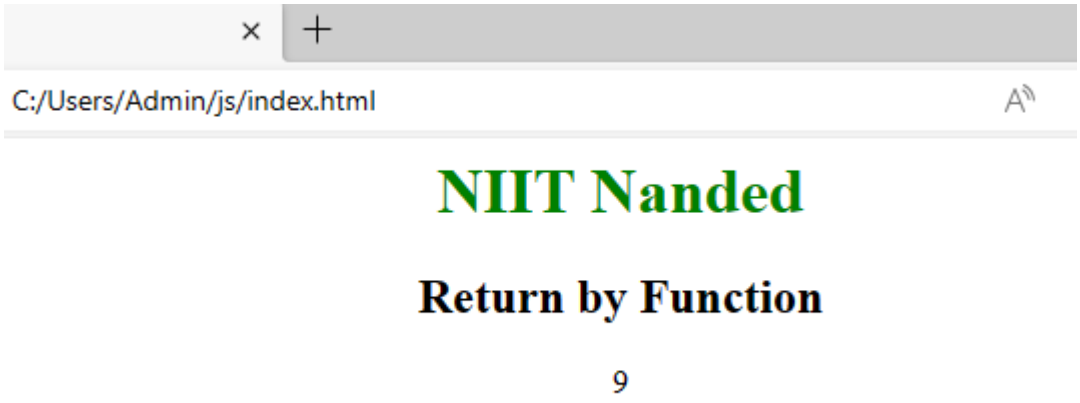


Function Return

- When JavaScript reaches a return statement, the function will stop executing.
- If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.
- Functions often compute a **return value**. The return value is "returned" back to the "caller":

Example:

```
<html>
<body style="text-align: center;">
  <h1 style="color:green;">
    NIIT Nanded
</h1>
<h2>
  Return by Function
</h2>
<script>
function getInfo(number){
return number*number;
}
</script>
<script>
document.write(getInfo(3));
</script>
</body>
</html>
```



Arrow Function:

It is one of the most used and efficient methods to create a function in JavaScript because of its comparatively easy implementation. It is a simplified as well as a more compact version of a regular or normal function expression or syntax.

Syntax:

```
let function_name = (argument1, argument2 ,..) => expression
```

Example:

```
const a = ["Hydrogen", "Helium", "Lithium", "Beryllium"];

const a2 = a.map(function (s) {
    return s.length;
});

console.log("Normal way ", a2); // [8, 6, 7, 9]

const a3 = a.map((s) => s.length);

console.log("Using Arrow Function ", a3); // [8, 6, 7, 9]
```

1.18 Making Link Smarter

- HTML Links are connections from one web resource to another. A link has two ends, An anchor and a direction.
- The link starts at the “source” anchor and points to the “destination” anchor, which may be any Web resource such as an image, a video clip, a sound bite, a program, an HTML document or an element within an HTML document.
- You will find many websites or social media platforms (Like YouTube, and Instagram) which link an image to a URL or a text to a URL etc.
- This means that by using the ‘a’ tag, you can link 1 element of the code to another element that may/may not be in your code.

Syntax:

```
<a href="url">link text</a>
```

Example:

```
<a href="url">www.nandigramiit.org</a>
```

The Target Attribute:

By default, the linked page will be displayed in the current browser window. To change this, you must specify another target for the link.

The target attribute specifies where to open the linked document. The target attribute can have one of the following values:

- **_self** - Default. Opens the document in the same window/tab as it was clicked
- **_blank** - Opens the document in a new window or tab
- **_parent** - Opens the document in the parent frame
- **_top** - Opens the document in the full body of the window

Example:

```
<html>
<body style="text-align: center;">
    <h1 style="color:green;">
        NIIT Nanded
    </h1>
<h2>The target Attribute</h2>

<p>If target="_blank", the link will open in a new browser window or
tab.</p>
<a href="https://www.nandigramiit.org" target="_blank">Visit NIIT
Nanded!</a>
<p>If target="_self", the link will open in same window or tab.</p>
<a href="https://www.nandigramiit.org" target="_self">Visit NIIT Nanded!</a>

</body>
</html>
```

NIIT Nanded

The target Attribute

If target="_blank", the link will open in a new browser window or tab.

Visit NIIT Nanded!

If target="_self", the link will open in same window or tab.

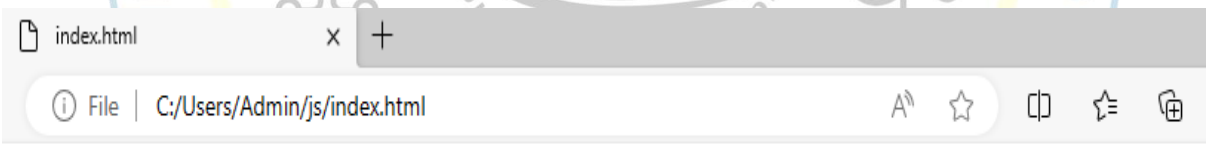
Visit NIIT Nanded!

Use a link to send an Email:

When placing an email address within a webpage or a contact form it is helpful to let users quickly get in touch. This is possible within the standard link format by using the “mailto” keyword prior to writing the recipient.

Example:

```
<html>
<body style="text-align: center;">
    <h1 style="color:green;">
        NIIT Nanded
    </h1>

<h2>Link to an Email Address</h2>

<p>To create a link that opens in the user's email program (to let them send
a new email), use mailto: inside the href attribute:</p>

<p><a href="mailto:someone@example.com">Send email</a></p>

</body>
</html>
```



NIIT Nanded

Link to an Email Address

To create a link that opens in the user's email program (to let them send a new email), use mailto: inside the href attribute:

Send email

1.19 Using Switch Case Statement

JavaScript **switch statement** is used to execute a block of code from multiple expressions. JavaScript switch statement evaluates an expression. The expression’s value is compared with the values of each case in the structure. If a match is found, the related block of code is executed.

- The expression can be of type numbers or strings.
- Duplicate case values are not allowed.
- The default statement is optional. If the expression passed to the switch does not match the value in any case then the statement under default will be executed.
- The break statement is used inside the switch to terminate a statement sequence.
- The break statement is optional. If omitted, execution will continue on into the next case.
- Cases are compared strictly.

JavaScript switch statement is used with a **break** or **default** keyword (optional and both can be used together also).

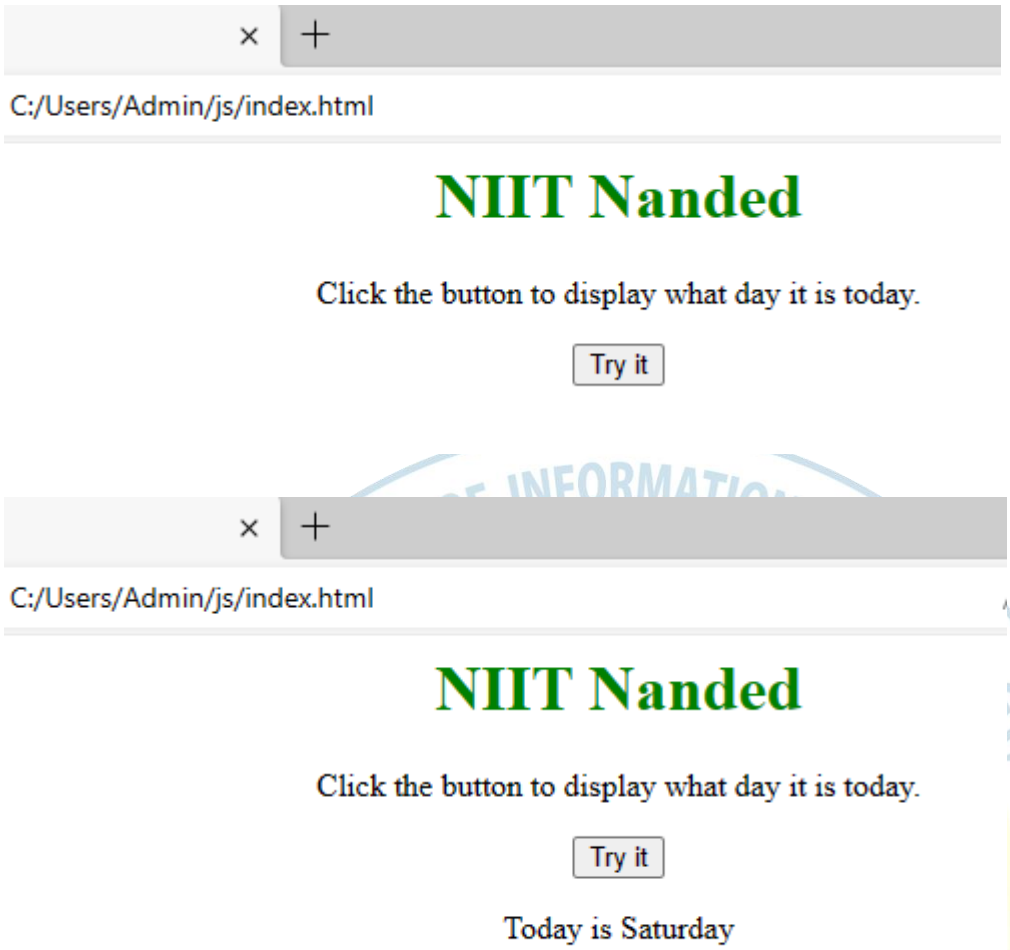
- **break:** This keyword is used to break out of the switch block. This stops the execution inside the code block.
- **default:** This keyword is used to specify a piece of code if no case matches the given condition. There can be only one default keyword in a switch statement.

Syntax:

```
switch(expression)
{
  case value1:
    code block
    break;
  case value2:
    code block
    break;
     .
     .
case valueN:
    code block
  default:
    default code block
}
```

Example:

```
<html>
<body style="text-align: center;">
    <h1 style="color:green;"> NIIT Nanded </h1>

<p>Click the button to display what day it is today.</p>

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

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

<script>
function myFunction() {
  var day;
  switch (new Date().getDay()) {
    case 0:
      day = "Sunday";
      break;
    case 1:
      day = "Monday";
      break;
    case 2:
      day = "Tuesday";
      break;
    case 3:
      day = "Wednesday";
      break;
    case 4:
      day = "Thursday";
      break;
    case 5:
      day = "Friday";
      break;
    case  6:
      day = "Saturday";
      break;
    default:
      day = "Unknown Day";
  }
  document.getElementById("demo").innerHTML = "Today is " + day;
}
</script>
</body>
</html>
```



1.20 Handling Error

In programming, handling errors or exception is a process or method used for handling the abnormal statements/ syntactically incorrect statement in the code and executing them.

For example, the Division of a non-zero value with zero will result into infinity always, and it is a run time error or exception. Thus, with the help of exception handling, it can be executed and handled.

An error is an action that is inaccurate or incorrect. There are three types of errors in programming which are discussed below:

- Syntax error
- Logical error
- Runtime error

1. **Syntax Error:** When a user makes a mistake in the pre-defined syntax of a programming language, a syntax error may appear.

```
<script type="text/javascript">

    // An runtime error here
    window.printme();

</script>
```

2. **Logical Error:** An error which occurs when there is any logical mistake in the program that may not produce the desired output, and may terminate abnormally. Such an error is known as Logical error
3. **Runtime Error:** When an error occurs during the execution of the program, such an error is known as Runtime error. The codes which create runtime errors are known as Exceptions. Thus, exception handlers are used for handling runtime errors

As in runtime errors, there are exceptions and these exceptions can be handled with the help of the try-and-catch method

Exception Handling Statements

There are following statements that handle if any exception occurs:

1.try…catch…finally statements.

2. throw statements.

try…catch…finally statements.

A try…catch is a commonly used statement in various programming languages. Basically, it is used to handle the error-prone part of the code. It initially tests the code for all possible errors it may contain, then it implements actions to tackle those errors (if occur). A good programming approach is to keep the complex code within the try…catch statements.

```
<script type = "text/javascript">
   <!--
      try {
         // Code to run
         [break;]
      }

      catch ( e ) {
         // Code to run if an exception occurs
         [break;]
      }

      [ finally {
         // Code that is always executed regardless of
         // an exception occurring
      }]
   //-->
</script>
```

The **try** block must be followed by either exactly one **catch** block or one **finally** block (or one of both). When an exception occurs in the **try** block, the exception is placed in **e** and the **catch** block is executed. The optional **finally** block executes unconditionally after try/catch.

catch {} statement executes only after the execution of the try {} statement. Also, one try block can contain one or more catch blocks.

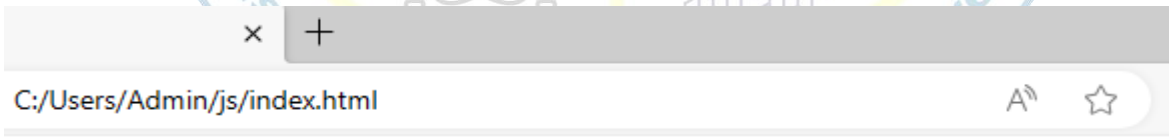
```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Document</title>
</head>
<body style="text-align: center;">
    <h1 style="color:green;"> NIIT Nanded </h1>
    <script>
        function First() {
            let a = 123;
            let b = 0;
            try {
                if (b == 0) {
                    throw "Do not divide by zero";
                }
            }
            catch (e) {
```

```
                document.getElementById('NIIT').innerHTML +=
                      "<br>" + e + "<br>";
            }
            finally {
                document.getElementById('NIIT').innerHTML +=
                      "  Finally block will always execute!";
            }
        }
    </script>
    <p id="NIIT">Click the NIIT button to see the result:</p>
    <form>
        <input type="button" value="NIIT" onclick="First()" />
    </form>
</body>
</html>
```

NIIT Nanded

Click the NIIT button to see the result:

NIIT



NIIT Nanded

Click the NIIT button to see the result:
Do not divide by zero
Finally block will always execute!

NIIT

The throw Statement

You can use **throw** statement to raise your built-in exceptions or your customized exceptions. Later these exceptions can be captured and you can take an appropriate action.

It will return the simple string message when any error occurs. When we throw an error, the program execution control goes to the nearest catch block to handle the exception. If we haven't defined any catch block, program execution terminates due to the exception fault.

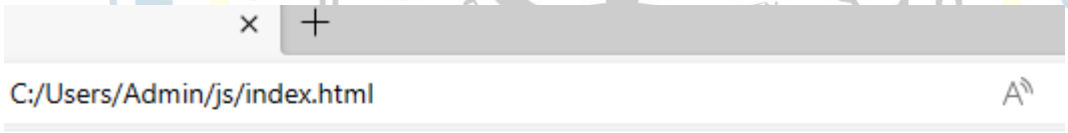
Example:

```
const number = 5;
try {
     // user-defined throw statement
     throw new Error('This is the throw');

}
catch(error) {
    console.log('An error caught');
    if( number + 8 > 10) {

        // statements to handle exceptions
         console.log('Error message: ' + error);
        console.log('Error resolved');
    }
    else {
        // cannot handle the exception
        // rethrow the exception
        throw new Error('The value is low');
    }
}
```

Example:

```
<!DOCTYPE html>
<html>
    <body style="text-align: center;">
        <h1 style="color:green;">
            NIIT Nanded
        </h1>

<h2>JavaScript try catch and throw</h2>

<p>Please input a number between 5 and 10:</p>
```

```
<input id="demo" type="text">
<button type="button" onclick="myFunction()">Test Input</button>
<p id="p01"></p>

<script>
function myFunction() {
  const message = document.getElementById("p01");
  message.innerHTML = "";
  let x = document.getElementById("demo").value;
  try {
    if(x.trim() == "")  throw "empty";
    if(isNaN(x)) throw "not a number";
    x = Number(x);
    if(x < 5)  throw "too low";
    if(x > 10)   throw "too high";
  }
  catch(err) {
    message.innerHTML = "Input is " + err;
  }
}
</script>

</body>
</html>
```



NIIT Nanded

JavaScript try catch and throw

Please input a number between 5 and 10:

Nanded Test Input

Input is not a number

NIIT Nanded

JavaScript try catch and throw

Please input a number between 5 and 10:

Test Input

Input is empty