|  |  |  |
| --- | --- | --- |
| **EX.NO-5(i)** | **JAVASCRIPT** | **KAMALI.A**  **22Z436**  **B.E-CSE(G1)** |

**JAVASCRIPT INTRODUCTION:**

* JavaScript is the world's most popular programming language.
* JavaScript is the programming language of the Web.JavaScript is easy to learn.
* JavaScript (js) is a light-weight object-oriented programming language which is used by several websites for scripting the webpages.
* It is an interpreted, full-fledged programming language that enables dynamic interactivity on websites when applied to an HTML document.
* JavaScript is an object-based scripting language which is lightweight and cross-platform.
* JavaScript is not a compiled language, but it is a translated language. The JavaScript Translator (embedded in the browser) is responsible for translating the JavaScript code for the web browser. Since then, it has been adopted by all other graphical web browsers.
* Features of JavaScript
* All popular web browsers support JavaScript as they provide built-in execution environments.
* JavaScript follows the syntax and structure of the C programming language. Thus, it is a structured programming language.
* JavaScript is an object-oriented programming language that uses prototypes rather than using classes for inheritance.
* It is a light-weighted and interpreted language.It is a case-sensitive language.
* JavaScript is supportable in several operating systems including, Windows, macOS, etc.
* Application of JavaScript
* JavaScript is used to create interactive websites.
* Client-side validation,
* Dynamic drop-down menus,
* Displaying date and time,
* Displaying clocks etc.

**JAVASCRIPT STATEMENTS:**

* JavaScript statements are composed of:
* Values, Operators, Expressions, Keywords, and Comments.
* This statement tells the browser to write "Hello Dolly." inside an HTML element with id="demo".

|  |  |
| --- | --- |
| Example | Output |
| <html><body>  <h2>JavaScript Statements</h2>  <p>JavaScript</p>  <p id="demo"></p><script>  document.getElementById("demo").innerHTML = "Hello Dolly.";  </script></body></html> |  |

* Most JavaScript programs contain many JavaScript statements.
* The statements are executed, one by one, in the same order as they are written.
* Semicolons ;
* Semicolons separate JavaScript statements.
* Add a semicolon at the end of each executable statement:

|  |  |
| --- | --- |
| Example | Output |
| <html><body>  <h2>JavaScript Stmt</h2>  <p id="demo1"></p><script>  let a, b, c;  a = 3;b = 2;c = a + b;  document.getElementById("demo1").  innerHTML = c;  </script></body></html> |  |

**JAVASCRIPT LITERALS:**

* The two most important syntax rules for fixed values are:
* Numbers
* Strings
* Numbers
* Numbers are written with or without decimals:

|  |  |
| --- | --- |
| Example | Output |
| <html><body>  <h4>JavaScript Numbers</h4>  <p id="demo"></p><script>  document.getElementById("demo").inner  HTML = 30.50;  </script></body></html> |  |

* Strings
* Strings are text, written within double or single quotes:

|  |  |
| --- | --- |
| Example | Output |
| <html><body>  <h4>JavaScript Strings</h4>  <p id="demo"></p><script>  document.getElementById("demo").inner  HTML = 'Kamali';  </script></body></html> |  |

**JAVASCRIPT VARIABLES:**

* In a programming language, variables are used to store data values.
* JavaScript uses the keywords var, let and const to declare variables.
* An equal sign is used to assign values to variables.
* In this example, x is defined as a variable. Then, x is assigned (given) the value 3:

|  |  |
| --- | --- |
| Example | Output |
| <html><body><h4>JavaScript Variable</h4>  <p id="demo"></p><script>  let x; x = 3;  document.getElementById("demo").innerHTML = x;  </script></body></html> |  |

**JAVASCRIPT OPERATORS:**

* JavaScript uses arithmetic operators ( + - \* / ) to compute values:

|  |  |
| --- | --- |
| Example | Output |
| <html><body>  <h4>JavaScript Operators</h4>  <p id="demo">  </p><script>  document.getElementById("demo").  innerHTML = (2\*3) \* 5;  </script></body></html> |  |

* JavaScript uses an assignment operator ( = ) to assign values to variables:

|  |  |
| --- | --- |
| Example | Output |
| <html>  <h4>Assigning JavaScript Values</h4>  <body><p id="demo"></p>  <script>  let x, y; x = 1; y = 2;  document.getElementById("demo").  innerHTML = x + y;  </script>  </body></html> |  |

**JAVASCRIPT DATA TYPES:**

* JavaScript has 8 Datatypes

1. String
2. Number
3. Bigint
4. Boolean
5. Undefined
6. Null
7. Symbol
8. Object

* The Object Datatype

1. An object
2. An array
3. A date

|  |  |
| --- | --- |
| Example | Output |
| <html><body>  <h2>JavaScript</h2>  <p id="demo"></p>  <script>  let x = 20 + 10 + "JS";  document.getElementById("demo").  innerHTML = x;  </script>  </body></html> |  |

**JAVASCRIPT FUNCTIONS:**

* A JavaScript function is a block of code designed to perform a particular task.
* A JavaScript function is executed when "something" invokes it (calls it).
* JavaScript Function Syntax
* A JavaScript function is defined with the function keyword, followed by a name, followed by parentheses ().
* Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).
* The parentheses may include parameter names separated by commas:
* (parameter1, parameter2, ...)
* The code to be executed, by the function, is placed inside curly brackets: {}

|  |
| --- |
|  |

* 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.

|  |  |
| --- | --- |
| Example | Output |
| <html>  <body>  <h3>JS Functions</h3>  <p id="demo">  </p><script>  function myFunction(p1, p2) {  return p1 \* p2;}  let result = myFunction(2, 3);  document.getElementById("demo").  innerHTML = result;  </script>  </body>  </html> |  |

**JAVASCRIPT OBJECTS:**

* Real Life Objects, Properties, and Methods
* In real life, a car is an object.
* A car has properties like weight and color, and methods like start and stop:
* Example

|  |
| --- |
|  |

* All cars have the same properties, but the property values differ from car to car.
* All cars have the same methods, but the methods are performed at different times.
* Example

|  |  |
| --- | --- |
| <html><body>  <p id="demo"></p>  <script>  // Create and display a variable:  let car = "Fiat";  document.getElementById("demo")  .innerHTML = car;  </script>  </body></html> | Output: |

**JAVASCRIPT EVENTS:**

* HTML events are "things" that happen to HTML elements.
* When JavaScript is used in HTML pages, JavaScript can "react" on these events.
* HTML Events
* An HTML event can be something the browser does, or something a user does.
* Here are some examples of HTML events:
* An HTML web page has finished loading
* An HTML input field was changed
* An HTML button was clicked
* Often, when events happen, you may want to do something.
* JavaScript lets you execute code when events are detected.
* HTML allows event handler attributes, with JavaScript code, to be added to HTML elements.
* With single quotes:
* <element event='some JavaScript'>
* With double quotes:
* <element event="some JavaScript">

|  |  |
| --- | --- |
| Example | Output |
| <html>  <body>  <button onclick="document.getElementById('demo').innerHTML=Date()">The time is?</button>  <p id="demo"></p>  </body>  </html> |  |

* JavaScript Event Handlers
* Event handlers can be used to handle and verify user input, user actions, and browser actions:
* Things that should be done every time a page loads
* Things that should be done when the page is closed
* Action that should be performed when a user clicks a button
* Content that should be verified when a user inputs data
* And more ...
* Many different methods can be used to let JavaScript work with events:
* HTML event attributes can execute JavaScript code directly
* HTML event attributes can call JavaScript functions
* You can assign your own event handler functions to HTML elements
* You can prevent events from being sent or being handled
* And more ..

**JAVASCRIPT STRINGS:**

* Strings are for storing text
* Strings are written with quotes
* A JavaScript string is zero or more characters written inside quotes.

|  |  |
| --- | --- |
| Example | Output |
| <html><body>  <h3>JavaScript Strings</h3>  <p id="demo"></p><script>  let text = "Kamali";  document.getElementById("demo")  .innerHTML = text;  </script></body>  </html> |  |

**JAVASCRIPT STRING METHODS:**

* Basic String Methods
* Javascript strings are primitive and immutable:
* All string methods produces a new string without altering the original string.
* String length
* String charAt()
* String charCodeAt()
* String at()
* String [ ]
* String slice()
* String substring()
* String substr()…..

|  |  |
| --- | --- |
| Example | Output |
| <html><body>  <p id="demo"></p><script>  let text = "KAMALI.A";  document.getElementById("demo")  .innerHTML = text.length;  </script></body></html> |  |

**JAVASCRIPT TEMPLATE STRINGS:**

* Javascript template strings as..
* String Templates
* Template Strings
* Template Literals
* Back-Tics Syntax
* Template Strings use back-ticks (``) rather than the quotes ("") to define a string:

|  |  |
| --- | --- |
| Example | Output |
| <html><body>  <p id="demo"></p><script>  let text = `Hello world!`;  document.getElementById("demo")  .innerHTML = text;  </script></body></html> |  |

**JAVASCRIPT NUMBERS:**

* JavaScript has only one type of number.
* Numbers can be written with or without decimals.
* JavaScript Numbers are Always 64-bit Floating Point
* Unlike many other programming languages, JavaScript does not define different types of numbers, like integers, short, long, floating-point etc.
* JavaScript numbers are always stored as double precision floating point numbers, following the international IEEE 754 standard.
* This format stores numbers in 64 bits, where the number (the fraction) is stored in bits 0 to 51, the exponent in bits 52 to 62, and the sign in bit 63:

|  |
| --- |
|  |

* Js number example

|  |  |
| --- | --- |
| Example | Output |
| <html><body><p id="demo"></p>  <script>let x = 3.14;  let y = 3;  document.getElementById("demo")  .innerHTML = x + "<br>" + y;  </script></body></html> |  |

**JAVASCRIPT NUMBER METHODS:**

* These number methods can be used on all JavaScript numbers:

|  |
| --- |
|  |

* The toString() Method
* The toString() method returns a number as a string.
* All number methods can be used on any type of numbers (literals, variables, or expressions):

|  |  |
| --- | --- |
| Example | Output |
| <html><body>  <p id="demo"></p>  <script>  let x = 100;  document.getElementById("demo")  .innerHTML =  x.toString() + "<br>" +(100).toString() + "<br>" + (50 + 50).toString();  </script></body>  </html> |  |

* The toExponential() Method
* toExponential() returns a string, with a number rounded and written using exponential notation.
* A parameter defines the number of characters behind the decimal point:

|  |  |
| --- | --- |
| Example | Output |
| <html><body><p id="demo"></p>  <script>let x = 9.656;  document.getElementById("demo")  .innerHTML =  x.toExponential() + "<br>" +  x.toExponential(2) + "<br>" +  x.toExponential(4) + "<br>" +  x.toExponential(6);  </script></body></html> |  |

**JAVASCRIPT ARRAYS:**

* Creating an Array
* Using an array literal is the easiest way to create a JavaScript Array.
* Syntax:
* const array\_name = [item1, item2, ...];

|  |  |
| --- | --- |
| Example | Output |
| <html><body>  <h3>JavaScript Arrays</h3>  <p id="demo"></p><script>  const flower= ["LOTUS", "ROSE"];  document.getElementById("demo")  .innerHTML =flower;  </script></body></html> |  |

**JAVASCRIPT DATE OBJECTS:**

* Creating Date Objects
* Date objects are created with the new Date() constructor.
* There are 9 ways to create a new date object:
* new Date()
* new Date(date string)
* new Date(year,month)
* new Date(year,month,day)
* new Date(year,month,day,hours)
* new Date(year,month,day,hours,minutes)
* new Date(year,month,day,hours,minutes,seconds)
* new Date(year,month,day,hours,minutes,seconds,ms)
* new Date(milliseconds)
* JavaScript new Date()
* new Date() creates a date object with the current date and time:

|  |  |
| --- | --- |
| Example | Output |
| <html><body>  <p id="demo"></p>  <script>const d = new Date();  document.getElementById("demo")  .innerHTML = d;  </script>  </body></html> |  |

**JAVASCRIPT LOOPS:**

* Loops are handy, if you want to run the same code over and over again, each time with a different value.
* Often this is the case when working with arrays:

|  |  |
| --- | --- |
| Example | Output |
| <html><body>  <h2>JavaScript For Loop</h2>  <p id="demo"></p>  <script>  const cars = ["BMW", "Volvo", "Saab", "Ford", "Fiat", "Audi"];  let text = "";  for (let i = 0; i < cars.length; i++) {  text += cars[i] + "<br>";}  document.getElementById("demo")  .innerHTML = text;  </script></body>  </html> |  |

**JAVASCRIPT CLASSES:**

* JavaScript Classes are templates for JavaScript Objects.
* JavaScript Class Syntax
* Use the keyword class to create a class.
* Always add a method named constructor():
* Syntax

|  |
| --- |
|  |

* Using a Class

|  |  |
| --- | --- |
| Example | Output |
| <html><body><p id="demo"></p>  <script>class Car {  constructor(name, year) {  this.name = name;  this.year = year;}}  const myCar1 = new Car("Ford", 2014);  const myCar2 = new Car("Audi", 2019);  document.getElementById("demo")  .innerHTML =  myCar1.name + " " + myCar2.name;  </script></body>  </html> |  |

* Class Methods
* Class methods are created with the same syntax as object methods.
* Use the keyword class to create a class.
* Always add a constructor() method.
* Then add any number of methods.
* Syntax

|  |
| --- |
| class ClassName {  constructor() { ... }  method\_1() { ... }  method\_2() { ... }  method\_3() { ... }  } |

**CREATING THE FORM:**

* Forms are the basics of HTML. We use HTML form element in order to create the JavaScript form.
* For creating a form, we can use the following sample code:

|  |  |
| --- | --- |
| Example | Output |
| <html> <title>Login Form</title> <body>  <formform ="Login\_form"onsubmit="submit\_form()">  <h4> USERNAME</h4>  <input type="text" placeholder="Enter your email id"/> <h4> PASSWORD</h4>  <input type="password" placeholder="Enter your password"/></br></br>  <input type="submit" value="Login"/>  <input type="button" value="SignUp" onClick="create()"/> </form> </html> |  |

* Form name tag is used to define the name of the form. The name of the form here is "Login\_form". This name will be referenced in the JavaScript form.
* The action tag defines the action, and the browser will take to tackle the form when it is submitted. Here, we have taken no action.
* The method to take action can be either post or get, which is used when the form is to be submitted to the server. Both types of methods have their own properties and rules.
* The input type tag defines the type of inputs we want to create in our form. Here, we have used input type as 'text', which means we will input values as text in the textbox.Net, we have taken input type as 'password' and the input value will be password.
* Next, we have taken input type as 'button' where on clicking, we get the value of the form and get displayed.
* Other than action and methods, there are the following useful methods also which are provided by the HTML Form Element.
* submit (): The method is used to submit the form.
* reset (): The method is used to reset the form values.
* Referencing forms
* Now, we have created the form element using HTML, but we also need to make its connectivity to JavaScript. For this, we use the getElementById () method that references the html form element to the JavaScript code.
* The syntax of using the getElementById() method is as follows:

|  |
| --- |
| form = document.getElementById('subscribe'); |

* Submitting the form
* Next, we need to submit the form by submitting its value, for which we use the onSubmit() method. Generally, to submit, we use a submit button that submits the value entered in the form.
* The syntax of the submit() method is as follows:

|  |
| --- |
| <input type="submit" value="Subscribe"> |

* When we submit the form, the action is taken just before the request is sent to the server. It allows us to add an event listener that enables us to place various validations on the form. Finally, the form gets ready with a combination of HTML and JavaScript code.
* Login Form

|  |  |
| --- | --- |
| Example | Output |
| <html>  <title> Login Form</title>  <body>  <h3> LOGIN </h3>  <formform ="Login\_form" onsubmit="submit\_form()">  <h4> USERNAME</h4>  <input type="text" placeholder="Enter your email id"/> <h4> PASSWORD</h4>  <input type="password" placeholder="Enter your password"/></br></br>  <input type="submit" value="Login"/>  <input type="button" value="SignUp" onClick="create()"/> </form>  <script type="text/javascript">  function submit\_form(){  alert("Login successfully"); }  function create(){  window.location="signup.html"; }  </script>  </body>  </html> |  |

* SignUp Form

|  |  |
| --- | --- |
| Example | Output |
| <html>  <title>SignUp Page</title>  <body>  <h1>CREATE YOUR ACCOUNT</h1>  <table cellspacing="2" cellpadding="8" border="0">  <tr><td>Name</td>  <td><input type="text" placeholder="Enter your name" id="n1"></td></tr>  <tr><td>Email </td>  <td><input type="text" placeholder="Enter your email id" id="e1"></td></tr>  <tr><td>Set Password</td>  <td><input type="password" placeholder="Set a password" id="p1"></td></tr>  <tr><td>Confirm Password</td>  <td><input type="password" placeholder="Confirm your password" id="p2"></td></tr>  <tr><td>  <input type="submit" value="Create" onClick="create\_account()"/></table>  <script type="text/javascript">  function create\_account(){  var n=document.getElementById("n1").value;  var e=document.getElementById("e1").value;  var p=document.getElementById("p1").value;  var cp=document.getElementById("p2").value;  var letters = /^[A-Za-z]+$/;  var email\_val = /^([a-zA-Z0-9\_\.\-])+\@(([a-zA-Z0-9\-])+\.)+([a-zA-Z0-9]{2,4})+$/;  if(n==''||e==''||p==''||cp==''){  alert("Enter each detail correctly");}  else if(!letters.test(n)){  alert('Name is incorrect; it must contain alphabets only');}  else if (!email\_val.test(e)){ alert('Invalid email format; please enter a valid email id');}  else if(p!=cp){ alert("Passwords not matching");  }else if(document.getElementById("p1").  value.length > 12){  alert("Password maximum length is 12");}  else if(document.getElementById("p1").  value.length < 6)  { alert("Password minimum length is 6");}  else{alert("Your account has been created successfully... Redirecting to JavaTpoint.com");  window.location=  "https://www.javatpoint.com/";}}  </script>  </body>  </html> |  |

**STYLE THE FORM WITH CSS:**

* Styling a form with CSS involves applying visual design properties to form elements, such as input fields, labels, and buttons.
* This includes setting dimensions, colors, padding, borders, and other presentational attributes to enhance the form's appearance and user experience.

|  |  |
| --- | --- |
| Example | Output |
| <html lang="en">  <head>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <title>Styled Form</title>  <style>  /\* Your CSS code here \*/  form {  width: 300px;  margin: 20px auto;}  label {  display: block;  margin-bottom: 10px;}  input[type="text"],  input[type="email"],  input[type="password"] {  width: 100%;  padding: 8px;  margin-bottom: 10px;  box-sizing: border-box;}  input[type="submit"] {  background-color: #4caf50;  color: white;  padding: 10px 15px;  border: none;  border-radius: 5px;  cursor: pointer;}  input[type="submit"]:hover {  background-color: #45a049;}  </style></head><body>  <!-- Your form structure here -->  <form>  <label for="username">Username:</label>  <input type="text" id="username" name="username">  <label for="email">Email:</label>  <input type="email" id="email" name="email">  <label for="password">Password:</label>  <input type="password" id="password" name="password">  <input type="submit" value="Submit">  </form>  </body></html> |  |

**ADD JAVASCRIPT TO HTML:**

* JavaScript, also known as JS, is one of the scripting (client-side scripting) languages, that is usually used in web development to create modern and interactive web-pages. The term "script" is used to refer to the languages that are not standalone in nature and here it refers to JavaScript which run on the client machine.
* In other words, we can say that the term scripting is used for languages that require the support of another language to get executed. For example, JavaScript programs cannot get executed without the help of HTML or without integrated into HTML code.
* JavaScript is used in several ways in web pages such as generate warning messages, build image galleries, DOM manipulation, form validation, and more.
* There are following three ways in which users can add JavaScript to HTML pages.
* Embedding code
* Inline code
* External file
* Embedding code:
* To add the JavaScript code into the HTML pages, we can use the <script>.....</script> tag of the HTML that wrap around JavaScript code inside the HTML program.
* Users can also define JavaScript code in the <body> tag (or we can say body section) or <head> tag because it completely depends on the structure of the web page that the users use.

|  |  |
| --- | --- |
| Example | Output |
| <html> <head>  <title> Embedding code </title>  <script>  document.write("Welcome to JavaScript");  </script> </head>  <body> <p> Embedding code </p>  </body> </html> |  |

* Inline code:
* Generally, this method is used when we have to call a function in the HTML event attributes. There are many cases (or events) in which we have to add JavaScript code directly eg., OnMover event, OnClick, etc.
* Let's see with the help of an example, how we can add JavaScript directly in the html without using the <script>.... </script> tag.

|  |  |
| --- | --- |
| Example | Output |
| <html> <head>  <title> page title</title>  </head>  <body> <p>  <a href="#"  onClick=  "alert('Welcome !');">  Click Me</a> </p>  <p> inline JavaScript </p>  </body> </html> |  |

* External file:
* We can also create a separate file to hold the code of JavaScript with the (.js) extension and later incorporate/include it into our HTML document using the src attribute of the <script> tag.
* It becomes very helpful if we want to use the same code in multiple HTML documents.
* It also saves us from the task of writing the same code over and over again and makes it easier to maintain web pages.

|  |  |
| --- | --- |
| Example | Output |
| <html> <head>  <meta charset="utf-8">  <title>Including a External JavaScript File</title>  </head> <body> <form>  <input type="button" value="Result" onclick="display()"/>  </form>  <script src="hello.js">  </script> </body>  </html>  Hello.js  function display() {  alert("Hello World!");  } |  |

* Both of the above programs are saved in the same folder, but you can also store JavaScript code in a separate folder, all just you need to provide the address/path of the (.js) file in the src attribute of <script> tag.
* JavaScript files are common text files with (.js) extensions such as we created and used in the above program.
* External JavaScript file only contains JavaScript code and nothing else, even the <script>.... </script>tag are also not used in it.

**VALIDATING THE NAME FIELD:**

* Validating a name field typically involves checking if the input meets certain criteria, such as being non-empty, not containing invalid characters, and possibly having a minimum and maximum length**.**
* Name field validation in JavaScript refers to the process of ensuring that the input provided in a name field meets specific criteria or constraints.
* This validation aims to guarantee that the entered data is accurate, appropriate, and conforms to predefined rules.
* The rules for name validation can vary depending on the specific requirements of an application or system.
* Common criteria for name validation may include:
* Non-Empty: Ensuring that the name field is not left blank.
* Alphabetic Characters: Allowing only alphabetical characters, excluding numbers, special characters, or symbols.
* Length Limits: Setting minimum and maximum length constraints for the name.
* Format Restrictions: Defining a specific format for names, such as allowing only first names or first and last names with an optional middle name.
* Whitespace Handling: Managing leading or trailing whitespaces and preventing multiple consecutive spaces within the name.

|  |  |
| --- | --- |
| Example | Output |
| <html lang="en">  <head>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <script>  // JavaScript for name field validation  document.addEventListener  ("DOMContentLoaded", function()  {  const nameInput = document.getElementById("name");  nameInput.addEventListener("input", function()  {  const nameValue = nameInput.value.trim();  const nameError = document.getElementById("nameError");  if (nameValue === "")  {  nameError.textContent = "Name is required";  }  Else  { nameError.textContent = "";  }});  });  </script>  </head>  <body>  <form>  <label for="name">  Name:</label>  <input type="text" id="name" name="name">  <br>  <br>  <span id="nameError" style="color: red;">  </span>  <input type="submit" value="Submit">  </form>  </body>  </html> |  |

**VALIDATING THE EMAIL FIELD:**

* Validating the email field in JavaScript involves implementing checks and rules to ensure that user inputs into an email field adhere to a valid email address format.
* The goal is to enhance data accuracy, prevent input errors, and improve user experience by enforcing standards for the structure and content of email addresses.
* Email field validation typically includes checks for a non-empty value, proper syntax (username@domain), and may involve more advanced techniques to ensure a valid and existent domain.

|  |  |
| --- | --- |
| Example | Output |
| <html lang="en">  <head>  <script>  function validateEmail(emailId)  {  var mailformat =  /^([A-Za-z0-9\_\-\.])+\@([A-Za-z0-9\_\-\.])+\.([A-Za-z]{2,4})$/;  if(emailId.value.match(mailformat))  {  document.form1.text1.focus();  return true;  }  Else  {  alert("Invalid email address.");  document.form1.text1.focus();  return false;  }}  </script>  </head>  <body>  <div>  <h2>JavaScript email validation</h2>  <form name="form1" action="#">  Email: <input type='text' name='email'/>  </br></br>  <input type="submit" name="submit"  value="Submit"  onclick="validateEmail  (document.form1.email)"/>  </form>  </div>  </body>  </html> |  |

**VALIDATING THE PASSWORD FIELD:**

* Validating the password field in JavaScript involves implementing checks and rules to ensure that user-provided passwords meet specific criteria.
* Password validation is crucial for enhancing security by enforcing standards for password strength and complexity.
* Common validation criteria for passwords may include length requirements, a combination of uppercase and lowercase letters, numbers, and special characters.
* The goal is to encourage users to create strong and secure passwords that are less susceptible to unauthorized access.
* Format for Password Validation:
* Length: The password must be at least 8 characters long.
* Uppercase and Lowercase Letters: The password must include at least one uppercase letter and one lowercase letter.
* Digit: The password must include at least one digit.
* Example

<html><head>

<meta name="viewport" content="width=device-width, initial-scale=1">

<style>

/\* Style all input fields \*/

input {

width: 100%;

padding: 12px;

border: 1px solid #ccc;

border-radius: 4px;

box-sizing: border-box;

margin-top: 6px;

margin-bottom: 16px;

}

/\* Style the submit button \*/

input[type=submit] {

background-color: #04AA6D;

color: white;

}

/\* Style the container for inputs \*/

.container {

background-color: #f1f1f1;

padding: 20px;

}

/\* The message box is shown when the user clicks on the password field \*/

#message {

display:none;

background: #f1f1f1;

color: #000;

position: relative;

padding: 20px;

margin-top: 10px;

}

#message p {

padding: 10px 35px;

font-size: 18px;

}

.valid {

color: green;

}

.valid:before {

position: relative;

left: -35px;

content: "✔";

}

.invalid {

color: red;

}

.invalid:before {

position: relative;

left: -35px;

content: "✖";

}

</style></head><body>

<h3>Password Validation</h3>

<div class="container">

<form action="/action\_page.php">

<label for="usrname">Username</label>

<input type="text" id="usrname" name="usrname" required>

<label for="psw">Password</label>

<input type="password" id="psw" name="psw" pattern="(?=.\*\d)(?=.\*[a-z])(?=.\*[A-Z]).{8,}" title="Must contain at least one number and one uppercase and lowercase letter, and at least 8 or more characters" required>

<input type="submit" value="Submit">

</form></div>

<div id="message">

<h3>Password must contain the following:</h3>

<p id="letter" class="invalid">A <b>lowercase</b> letter</p>

<p id="capital" class="invalid">A <b>capital (uppercase)</b> letter</p>

<p id="number" class="invalid">A <b>number</b></p>

<p id="length" class="invalid">Minimum <b>8 characters</b></p>

</div>

<script>

var myInput = document.getElementById("psw");

var letter = document.getElementById("letter");

var capital = document.getElementById("capital");

var number = document.getElementById("number");

var length = document.getElementById("length");

myInput.onfocus = function() {

document.getElementById("message").style.display = "block";

}

myInput.onblur = function() {

document.getElementById("message").style.display = "none";

}

myInput.onkeyup = function() {

// Validate lowercase letters

var lowerCaseLetters = /[a-z]/g;

if(myInput.value.match(lowerCaseLetters)) {

letter.classList.remove("invalid");

letter.classList.add("valid");

} else {

letter.classList.remove("valid");

letter.classList.add("invalid");

}

// Validate capital letters

var upperCaseLetters = /[A-Z]/g;

if(myInput.value.match(upperCaseLetters)) {

capital.classList.remove("invalid");

capital.classList.add("valid");

} else {

capital.classList.remove("valid");

capital.classList.add("invalid");

}

// Validate numbers

var numbers = /[0-9]/g;

if(myInput.value.match(numbers)) {

number.classList.remove("invalid");

number.classList.add("valid");

} else {

number.classList.remove("valid");

number.classList.add("invalid");

}

// Validate length

if(myInput.value.length >= 8) {

length.classList.remove("invalid");

length.classList.add("valid");

} else {

length.classList.remove("valid");

length.classList.add("invalid");

}}

</script></body></html>

* Output

|  |
| --- |
|  |

**TESTING THE FORM:**

* Testing a form in JavaScript involves the process of verifying and validating the functionality and behavior of form elements within a web application.
* This can include checking user interactions, handling of input data, form submission, and ensuring the correct display of validation messages.
* Testing helps ensure that the form works as expected, minimizing the risk of errors and improving the overall user experience.
* Some testing forms in javascript
* User Interaction:
* Confirm that users can interact with form elements like input fields and buttons.
* Form Submission:
* Verify that submitting the form sends data to the server correctly.
* Input Validation:
* Test if the form correctly checks and responds to valid and invalid input.
* Error Handling:
* Check how the form displays error messages for users when they make mistakes.
* Mocking and Spying:
* Simulate external actions (like alerts) to see if the form responds correctly.
* Cross-browser Testing:
* Confirm that the form works well on different web browsers.

|  |  |
| --- | --- |
| Example | Output |
| **Index.html**  <html lang="en">  <meta charset="UTF-8"><meta name="viewport" content="width=device-width, initial-scale=1.0">  <title>Form Testing Example</title><body>  <form id="myForm"><label for="username">Username:</label>  <input type="text" id="username" name="username">  <input type="submit" value="Submit">  </form>  <script src="form.js"></script>  </body></html>  **form.js**  document.getElementById('myForm').addEventListener('submit', function(event) {  event.preventDefault();  const usernameInput = document.getElementById('username');  const usernameValue = usernameInput.value;  if (usernameValue.trim() === '') {  alert('Username cannot be empty!');  } else {  alert('Form submitted successfully!');  }});  **form.test.js**  test('Form submission with empty username should show an alert', () => {  jest.spyOn(window, 'alert').mockImplementation(() => {});  document.body.innerHTML = `  <form id="myForm">  <label for="username">Username:</label>  <input type="text" id="username" name="username" value="">  <input type="submit" value="Submit">  </form>  ;  document.getElementById('myForm').dispatchEvent(new Event('submit'));  expect(window.alert).toHaveBeenCalledWith('Username cannot be empty!');  }); |  |