

DOM

The browser creates a document object of a page when a web page is loaded and It represents the HTML document that is displayed in the window.

The Document object has various properties that refer to other objects which allow access and modification of document content. This is known as **Document Object Model (DOM)**.

The Objects are organized in a hierarchy. This hierarchical structure applies to the organization of objects in a Web document.

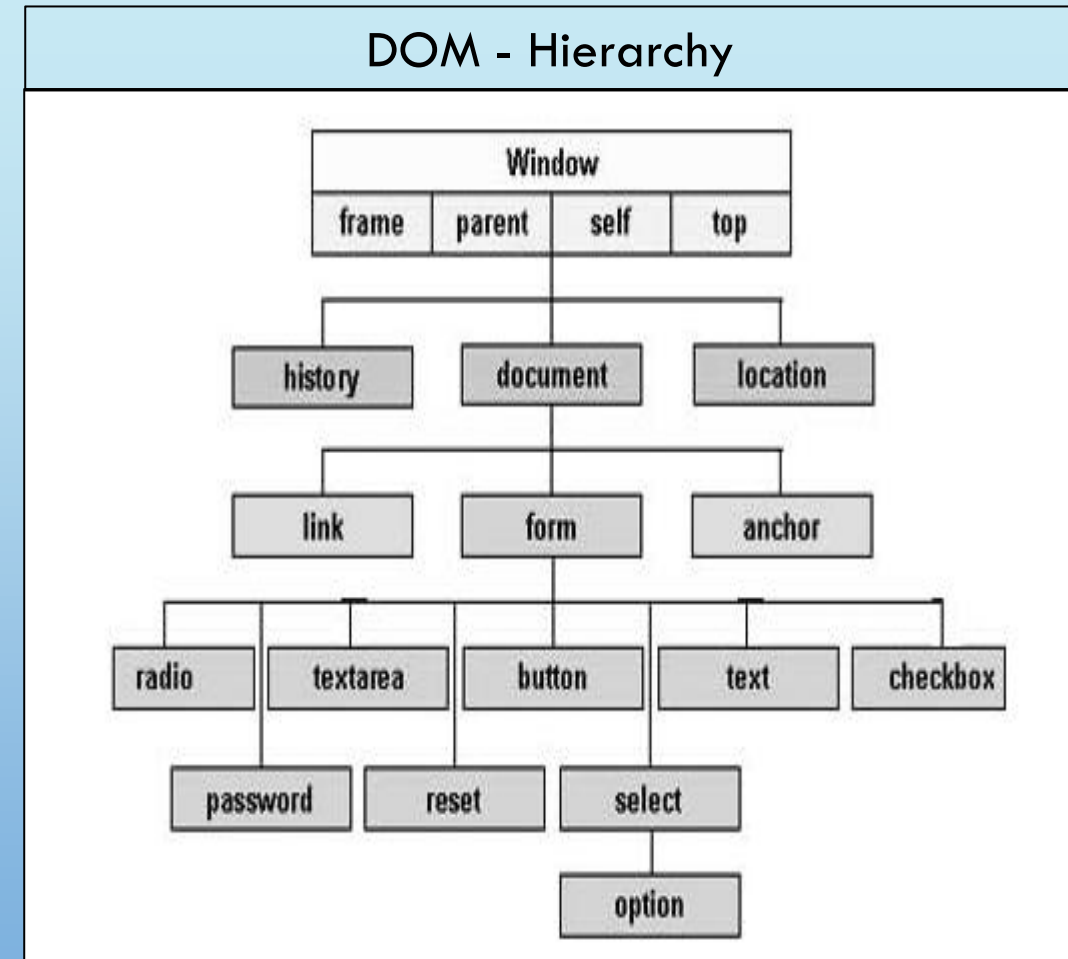
DOM

Window object – Top of the hierarchy. It is the outmost element of the object hierarchy.

Document object – Each HTML document that gets loaded into a window becomes a document object. The document contains the contents of the page.

Form object – Everything enclosed in the `<form>...</form>` tags sets the form object.

Form control elements – The form object contains all the elements defined for that object such as text fields, buttons, radio buttons, and checkboxes.



DOM – Applications

- JavaScript DOM can change all the HTML elements in the page
- JavaScript DOM can change all the HTML attributes in the page
- JavaScript DOM can change all the CSS styles in the page
- JavaScript DOM can remove existing HTML elements and attributes
- JavaScript DOM can add new HTML elements and attributes
- JavaScript DOM can react to all existing HTML events in the page
- JavaScript DOM can create new HTML events in the page

HTML DOM

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

- The HTML elements as **objects**
- The **properties** of all HTML elements
- The **methods** to access all HTML elements
- The **events** for all HTML elements

The HTML DOM can be accessed with JavaScript. In the DOM, all HTML elements are defined as **objects**. The programming interface is the properties and methods of each object.

- A **property** is a value that can be get or set (like changing the content of an HTML element).
- A **method** is an action like add or deleting an HTML element.

HTML DOM – property and method

The most common way to access an HTML element is to use the **id** of the element.

The **innerHTML** property is useful for getting or replacing the content of HTML elements.

The **innerHTML** property can be used to get or change any HTML element, including **<html>** and **<body>**.

HTML DOM – Elements Access

To access any element in an HTML page, use any of the following methods:

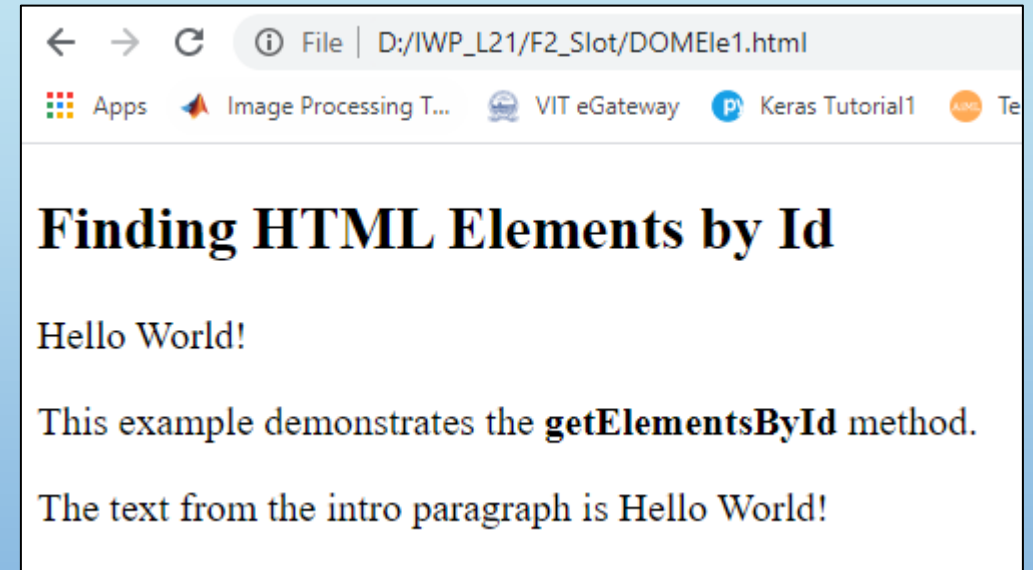
- Finding HTML elements by id
- Finding HTML elements by tag name
- Finding HTML elements by class name
- Finding HTML elements by CSS selectors
- Finding HTML elements by HTML object collections(forms, images, links, scripts, title, etc.)

| Method | Description |
|--|--|
| <code>document.getElementById(id)</code> | Find an element by element id |
| <code>document.getElementsByTagName(name)</code> | Find elements by tag name |
| <code>document.getElementsByClassName(name)</code> | Find elements by class name |
| <code>document.querySelector(element.stylename)</code> | Returns a list of all <element> with the class “stylename” |
| <code>document.objectname(“userObjectName”)</code> | Returns a list of elements belong to userObjectName |

HTMLDOM – getElementById

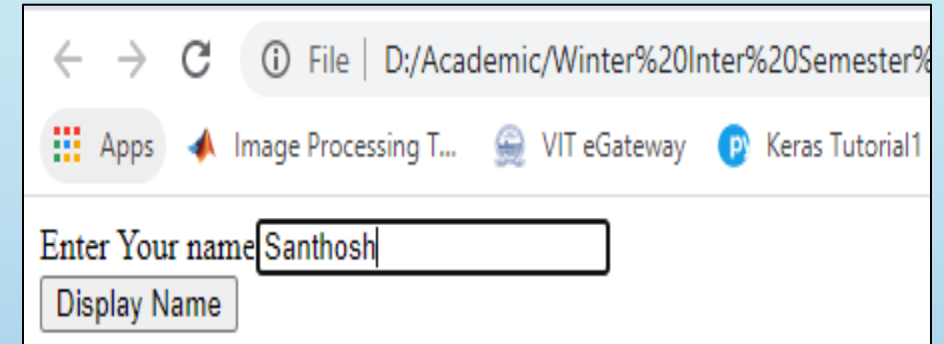
```
<html> <body>
<h2>Finding HTML Elements by Id</h2>
<p id="intro">Hello World!</p>
<p>This example demonstrates the <b>getElementsById</b>
method.</p>
<p id="demo"></p>
</body>

<script>
var myElement = document.getElementById("intro").innerHTML;
document.getElementById("demo").innerHTML =
"The text from the intro paragraph is " + myElement;
</script>
```



HTMLDOM – getElementById(Form)

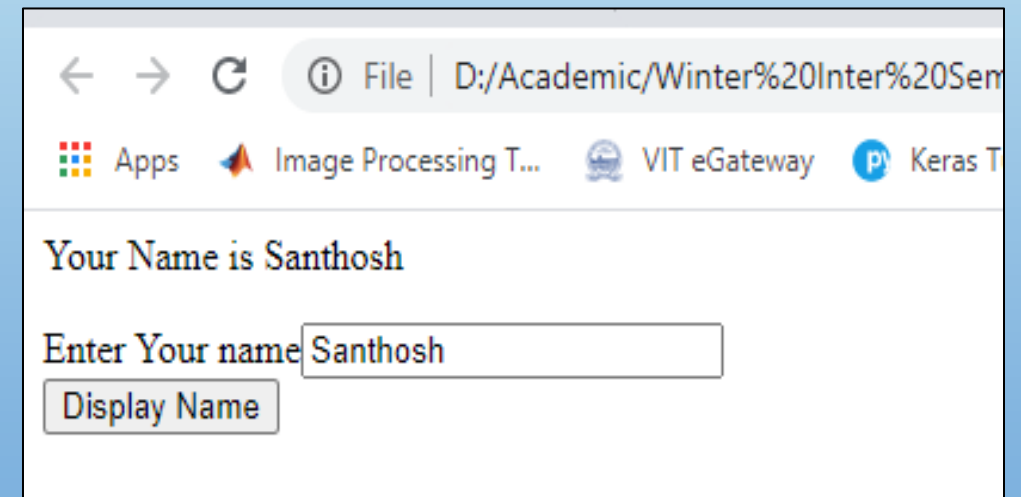
```
<script>
function CheckName(){
var str;
str = document.getElementById("usernameid").value;
document.getElementById('dispName').innerHTML = "Your Name
is "+str;
}
</script>
<body>
<p id="dispName"></p>
<form>
Enter Your name<input type="text" name="username"
id="usernameid"><br>
<button type="button" onclick="CheckName()">Display
Name</button>
</form>
```



← → ↻ ⓘ File | D:/Academic/Winter%20Inter%20Semester%

Apps Image Processing T... VIT eGateway Keras Tutorial1

Enter Your name



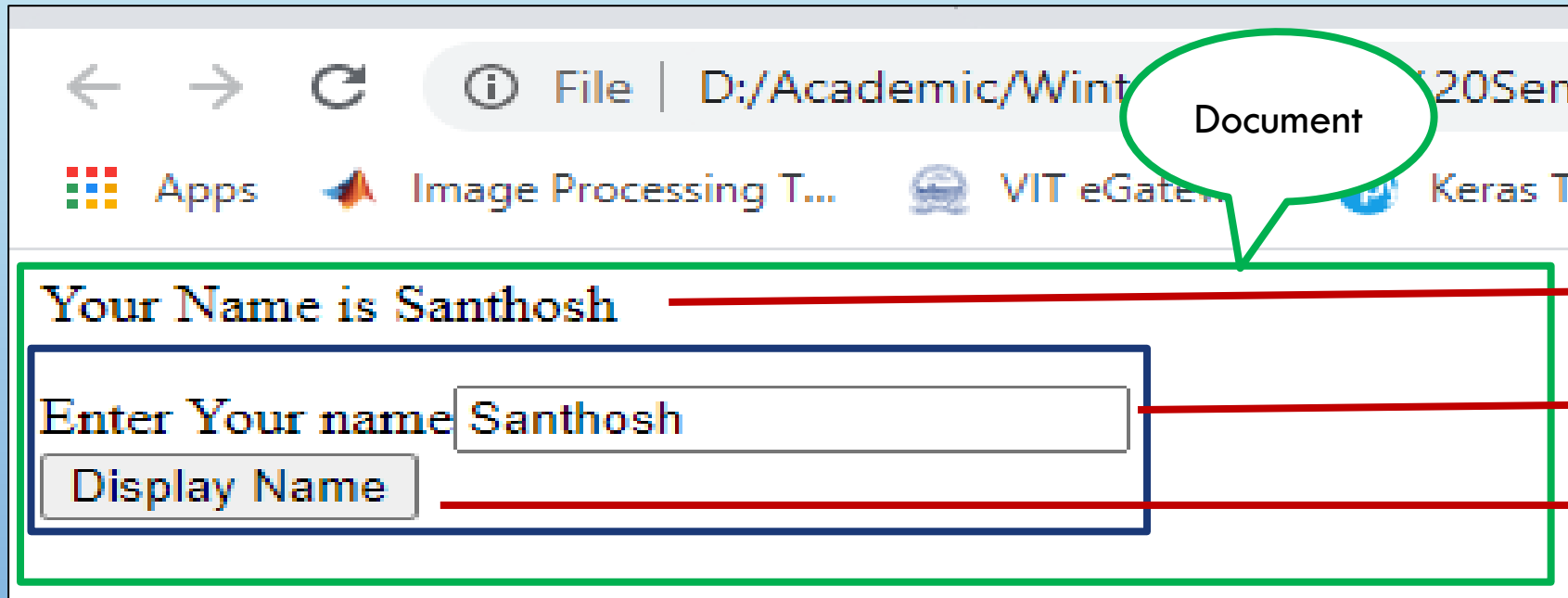
← → ↻ ⓘ File | D:/Academic/Winter%20Inter%20Sem

Apps Image Processing T... VIT eGateway Keras T

Your Name is Santhosh

Enter Your name

HTMLDOM – Diagram Representation



Document

Your Name is Santhosh

Enter Your name Santhosh

Display Name

Element<p> with id 'dispName'

Element textbox underform with id
'userNameid'

Element button under form

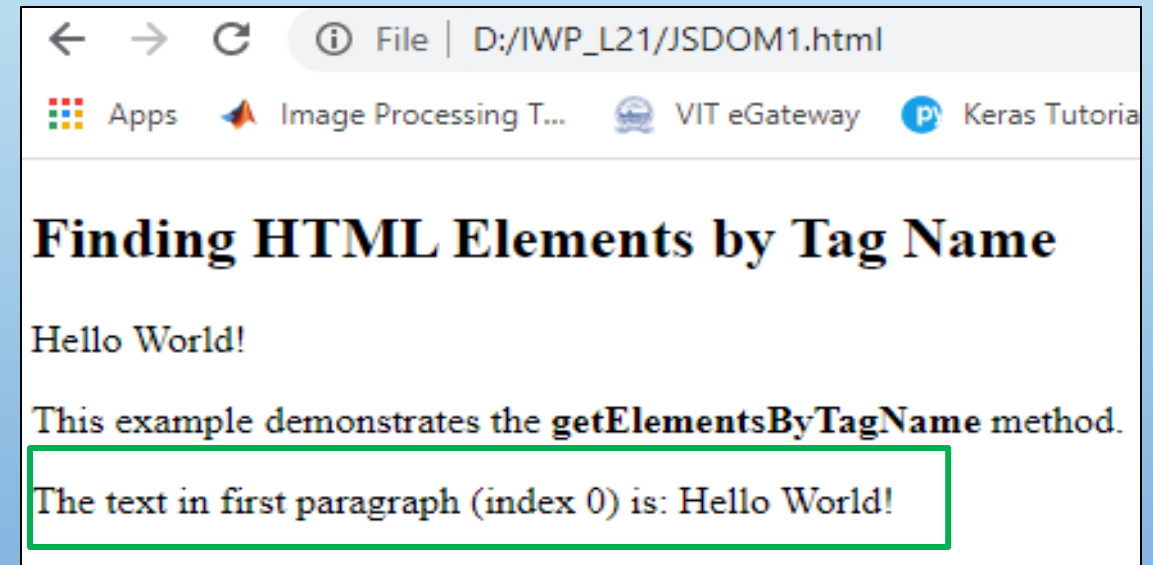
HTMLDOM – getElementByTagName

```
<!DOCTYPE html>
<html>
<script>
var x = document.getElementsByTagName("p");
document.getElementById("demo").innerHTML =
'The text in first paragraph (index 0) is: ' + x[0].innerHTML;
</script>
<body>

<h2>Finding HTML Elements by Tag Name</h2>

<p>Hello World!</p>
<p>This example demonstrates the
<b>getElementsByTagName</b> method.</p>

<p id="demo"></p>
</body>
</html>
```



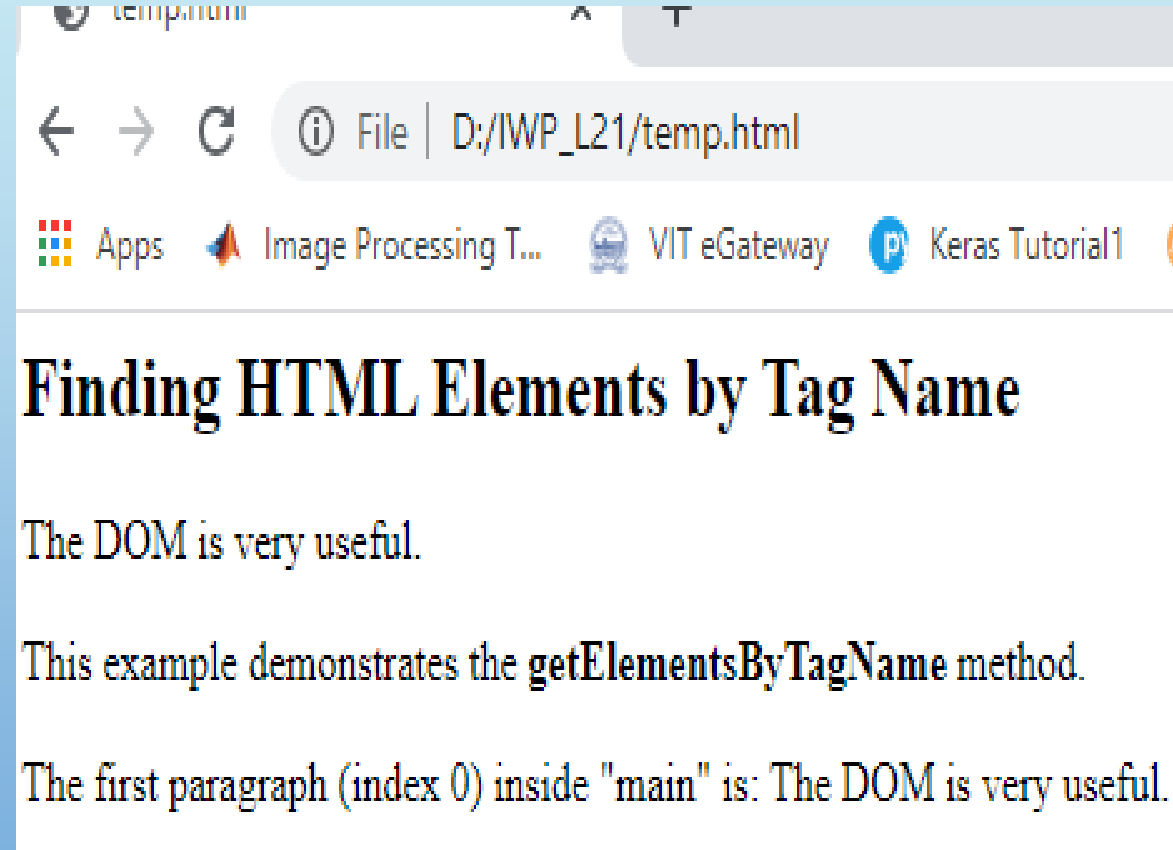
HTMLDOM – getElementByTagName (nested)

```
<script>
var x = document.getElementById("main");
var y = x.getElementsByTagName("p");
document.getElementById("demo").innerHTML =
'The first paragraph (index 0) inside "main" is: ' +
y[0].innerHTML;
</script> <body>
<h2>Finding HTML Elements by Tag Name</h2>
```

```
<div id="main">
<p>The DOM is very useful.</p>
<p>This example demonstrates the
<b>getElementsByTagName</b> method.</p>
</div>
```

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

```
</body>
```



HTMLDOM – getElementByClassName

```
<script>
var x = document.getElementsByClassName("intro");
document.getElementById("demo").innerHTML =
'The first paragraph (index 0) with class="intro": ' +
x[0].innerHTML;
</script> <body>

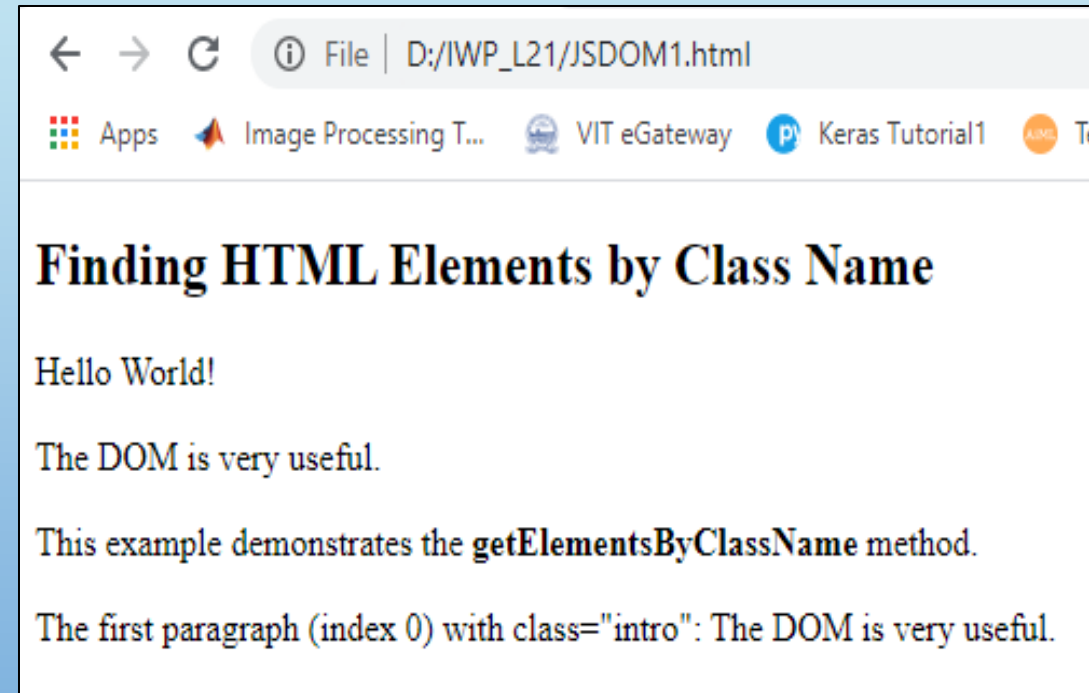
<h2>Finding HTML Elements by Class Name</h2>

<p>Hello World!</p>

<p class="intro">The DOM is very useful.</p>
<p class="intro">This example demonstrates the
<b>getElementsByClassName</b> method.</p>

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

</body>
```



HTMLDOM – CSS Selector

```
<script>
var x = document.querySelectorAll("p.intro");
document.getElementById("demo").innerHTML =
'The first paragraph (index 0) with class="intro": ' +
x[0].innerHTML;
</script> <body>

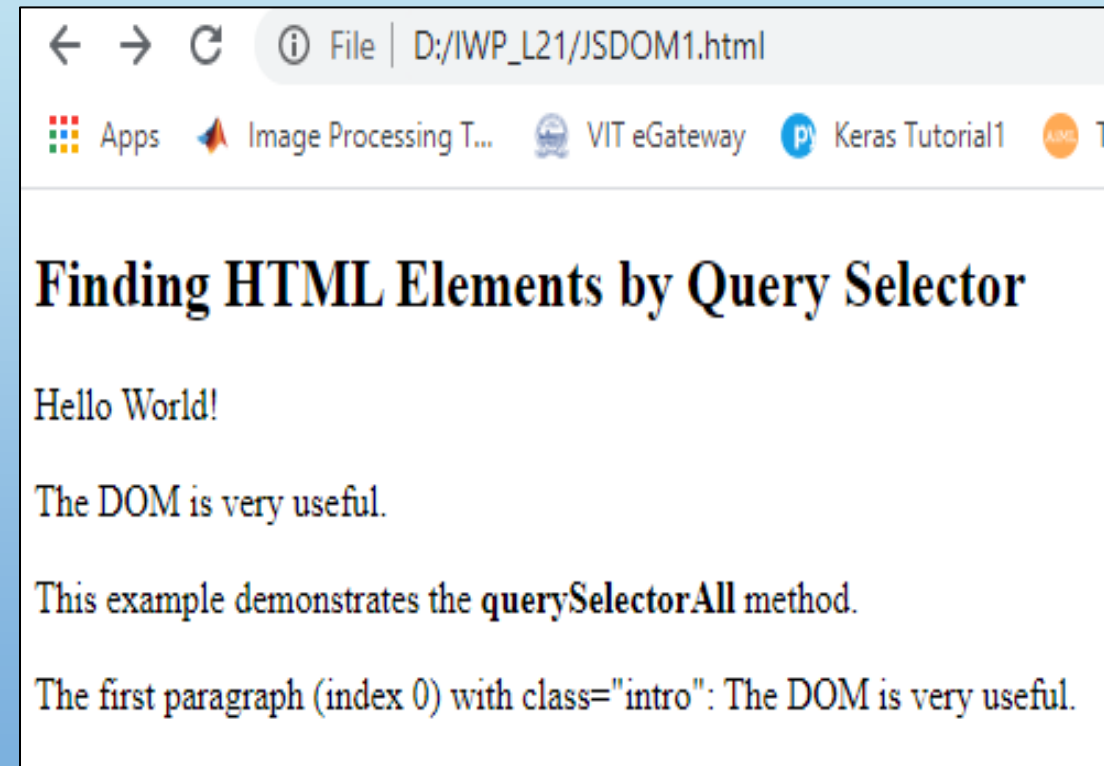
<h2>Finding HTML Elements by Query Selector</h2>

<p>Hello World!</p>

<p class="intro">The DOM is very useful.</p>
<p class="intro">This example demonstrates the
<b>querySelectorAll</b> method.</p>

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

</body>
</html>
```



HTMLDOM – CSS Selector

```
<br><b> Choose your favourite season: </b><br>
<input type="radio" name="season" id="summer" value="Summer"> Summer
<br>
<input type="radio" name="season" id="winter" value="Winter"> Winter <br>
<input type="radio" name="season" id="rainy" value="Rainy"> Rainy <br>
<input type="radio" name="season" id="autumn" value="Autumn"> Autumn
<br><br>
<button type="button" onclick=" checkButton()"> Submit </button>
<h3 id="disp" style=" color:green"> </h3>
<h4 id="error" style=" color:red"> </h4>
</body> <script>
    function checkButton() {
        var getSelectedValue =
document.querySelector('input[name="season"]:checked');
        if(getSelectedValue != null) {
            document.getElementById("disp").innerHTML = getSelectedValue.value
+ " season is selected"; }
        else {
            document.getElementById("error").innerHTML = "*You have not
selected any season";}
    }
```

Choose your favroite season:

- ☐ Summer
- ☐ Winter
- ☒ Rainy
- ☐ Autumn

Submit

Rainy season is selected

Choose your favroite season:

- ☐ Summer
- ☐ Winter
- ☐ Rainy
- ☐ Autumn

Submit

***You have not selected any season**

HTMLDOM – Object Collector

<h2>Finding HTML Elements Using document.forms</h2>

```
<form id="frm1" action="/action_page.php">
```

```
  First name: <input type="text" name="fname"
value="Donald"><br>
```

```
  Last name: <input type="text" name="lname"
value="Duck"><br><br>
```

```
  <input type="submit" value="Submit"> </form>
```

<p>Click "Try it" to display the value of each element in the form.</p>

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

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

```
</body>
```

```
</html>
```

```
<script> function myFunction() {
  var x = document.forms["frm1"];
  var text = "";
  var i;
  for (i = 0; i < x.length ;i++) {
    text += x.elements[i].value + "<br>";  }
  document.getElementById("demo").innerHTML = text; }
</script>
```

← → ↻ ⓘ File | D:/IWP_L21/JSDOM1.html

Apps Image Processing T... VIT eGateway Keras Tutor

Finding HTML Elements Using document

First name:

Last name:

Click "Try it" to display the value of each element in the form.

Donald
Duck
Submit

DOM

Object - Form

Document

Handling HTML Elements Using document.forms

First name:

Last name:

Click "Try it" to display the value of each element in the form.

Donald

Duck

Submit

Element inside Form

Element <p>

Element <button>

Element <p with id demo>

HTML – Input Attributes

name: The name attribute is used to specify the name of the input element.

value: The value attribute is used to specify the value of the input element.

type: The type attribute is used to specify the type of the input element. Its default value is text.

placeholder: Placeholder attribute is used to specify hint that describes the expected value of an input field.

required: The required attribute specifies that an input field must be filled out before submitting the form.

disabled: The disabled attribute specifies that the element should be disabled. The disabled attribute can be set to keep a user from using the < input > element until some other condition has been met.

readonly: The readonly attribute specifies that an input field is read-only. A read-only input field cannot be modified. A form will still submit an input field that is readonly, but will not submit an input field that is disabled.

HTML – Input Attributes

min: This property is used to specifies a minimum value for an <input> element

max : The max attribute is used to specify the maximum value for an < input > element.

step: This property is used to specifies the legal number intervals for an input field

maxlength: This property is used to specifies the maximum number of characters allowed in an <input> element

size: This property is used to specifies the width, in characters, of an <input> element

checked: The checked attribute specifies that an element should be pre-selected (checked) when the page loads. The checked attribute can be used with < input type="checkbox" > and < input type="radio" >.

multiple: This property is used to specifies that a user can enter more than one value in an <input> element

pattern: This property is used to specifies a regular expression that an <input> element's value is checked against.

Example – Input Attributes

<form >

First name

<input type="text" name="fname" id = "fname" value="John" readonly>

Last name: <input type="text" name="lname" id = "lname">

Middle Name: <input type="text" name="mname" id = "mname" value="Abraham" disabled>

Enter Pin with the length of 6:

<input type="text" name="pin" id = "pin" maxlength="6">

Enter a date before 1980-01-01:

<input type="date" name="datemax" id = "datemax" max="1979-12-31">

Enter a date after 2000-01-01:

<input type="date" name="datemax" id = "datemin" min="2000-01-02">

Quantity (between 1 and 5):

<input type="number" name="quantity" id = "quantity" min="1" max="5">

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

</form>

HTML – Input Attribute Output

HTML Input Attributes | Tryit Editor v3.6

File | D:/IWP_L21/temp.html

Apps Image Processing T... VIT eGateway Keras Tutorial1

First name:

Last name:

Middle Name:

Enter Pin with the length of 6:

Enter a date before 1980-01-01:

Enter a date after 2000-01-01:

Quantity (between 1 and 5):

Read-only Attribute

Disabled Attribute

Max length =6 Attribute

Date with max Attribute

December, 1979

| Mo | Tu | We | Th | Fr | Sa | Su |
|----|----|----|----|----|----|----|
| 26 | 27 | 28 | 29 | 30 | 1 | 2 |
| 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 1 | 2 | 3 | 4 | 5 | 6 |

Today

HTML Input Attributes | Tryit Editor v3.6

File | D:/IWP_L21/temp.html

Apps Image Processing T... VIT eGateway Keras Tutorial1

First name:

Last name:

Middle Name:

Enter Pin with the length of 6:

Enter a date before 1980-01-01:

Enter a date after 2000-01-01:

Quantity (between 1 and 5):

Date with min Attribute

March, 2021

| Mo | Tu | We | Th | Fr | Sa | Su |
|----|----|----|----|----|----|----|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| 15 | 16 | 17 | 18 | 19 | 20 | 21 |
| 22 | 23 | 24 | 25 | 26 | 27 | 28 |
| 29 | 30 | 31 | 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 | 9 | 10 | 11 |

Today

HTML Input Attributes | Tryit Editor v3.6

File | D:/IWP_L21/temp.html

Apps Image Processing T... VIT eGateway Keras Tutorial1

First name:

Last name:

Middle Name:

Enter Pin with the length of 6:

Enter a date before 1980-01-01:

Enter a date after 2000-01-01:

Quantity (between 1 and 5):

Range between 1 and 5

Java Script – Form Validation

Form validation normally used to occur at the server, after the client had entered all the necessary data and then pressed the Submit button. If the data entered by a client was incorrect or was simply missing, the server would have to send all the data back to the client and request that the form be resubmitted with correct information.

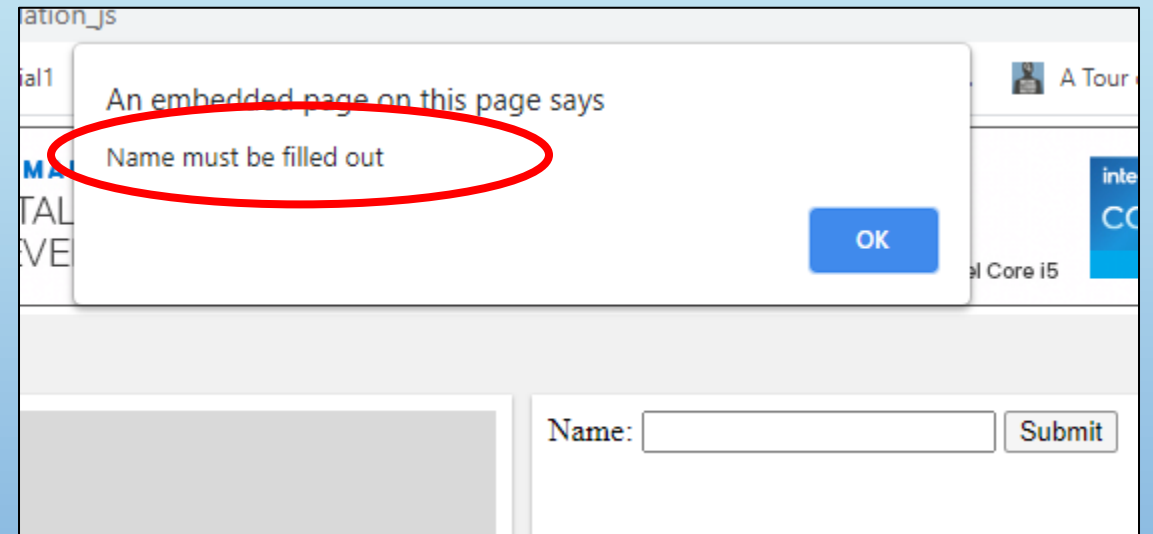
This was really a lengthy process which used to put a lot of burden on the server. JavaScript provides a way to validate form's data on the client's computer before sending it to the web server. Form validation generally performs two functions.

1. **Basic Validation** – First of all, the form must be checked to make sure all the mandatory fields are filled in. It would require just a loop through each field in the form and check for data.
2. **Data Format Validation** – Secondly, the data that is entered must be checked for correct form and value. The code must include appropriate logic to test correctness of data.

JavaScript – Basic Form Validation

HTML form validation can be done by JavaScript. If a form field (fname) is empty, this function alerts a message, and returns false, to prevent the form from being submitted:

```
<!DOCTYPE html> <html> <body>
<form name="myForm" action="/action_page.php"
onsubmit="return validateForm()" method="post">
  Name: <input type="text" name="fname">
  <input type="submit" value="Submit"> </form>
</body>
<script>
function validateForm() {
  var x = document.forms["myForm"]["fname"].value;
  if (x == "") {
    alert("Name must be filled out");
    return false;
  }
}
</script> </html>
```



JavaScript – Data Validation

Data validation is the process of ensuring that user input is clean, correct, and useful.

Typical validation tasks are:

- has the user filled in all required fields?
- has the user entered a valid date?
- has the user entered text in a numeric field?

HTML5 introduced a new HTML validation concept called **constraint validation**.

Input attributes are introduced:

- Disabled
- Max
- Min
- Pattern
- Required
- type

DOM – Validate Input Elements

`checkValidity()` is a constraint DOM validation method and it is used to check whether the input element has valid data. It returns true if element has valid data.

Constraints validation properties:

| Property | Description |
|-------------------|--|
| validity | Contains boolean properties related to the validity of an input element. |
| validationMessage | Contains the message a browser will display when the validity is false. |
| willValidate | Indicates if an input element will be validated. |

DOM – Values of Validity Property

| Property | Description |
|-----------------|--|
| customError | Set to true, if a custom validity message is set. |
| patternMismatch | Set to true, if an element's value does not match its pattern attribute. |
| rangeOverflow | Set to true, if an element's value is greater than its max attribute. |
| rangeUnderflow | Set to true, if an element's value is less than its min attribute. |
| stepMismatch | Set to true, if an element's value is invalid per its step attribute. |
| tooLong | Set to true, if an element's value exceeds its maxLength attribute. |
| typeMismatch | Set to true, if an element's value is invalid per its type attribute. |
| valueMissing | Set to true, if an element (with a required attribute) has no value. |
| valid | Set to true, if an element's value is valid. |

HTML – checkValidity()

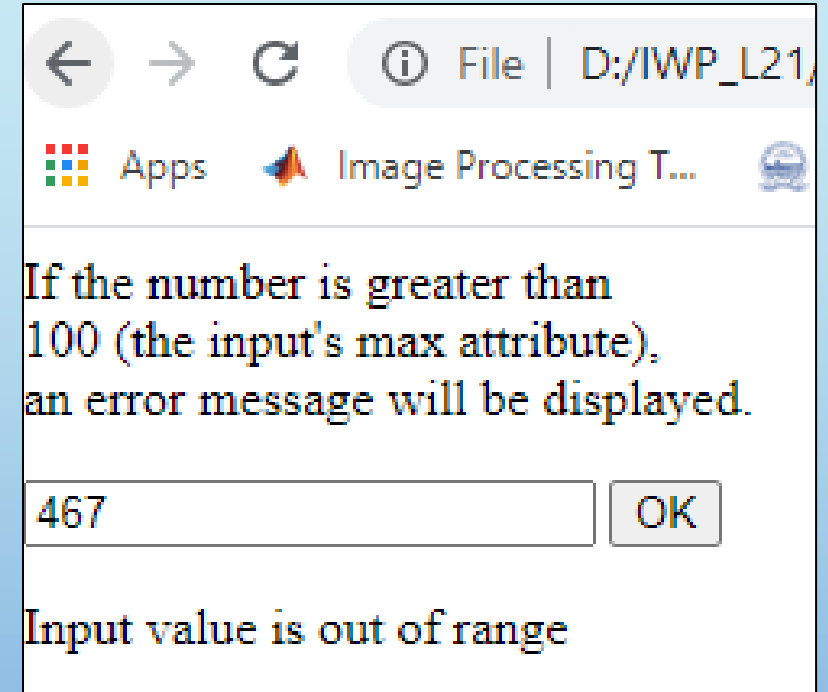
```
<input id="id1" type="number" max="100">  
<button onclick="myFunction()">OK</button>
```

```
<p>If the number is greater than 100 (the input's max attribute),  
an error message will be displayed.</p>
```

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

```
<script>
```

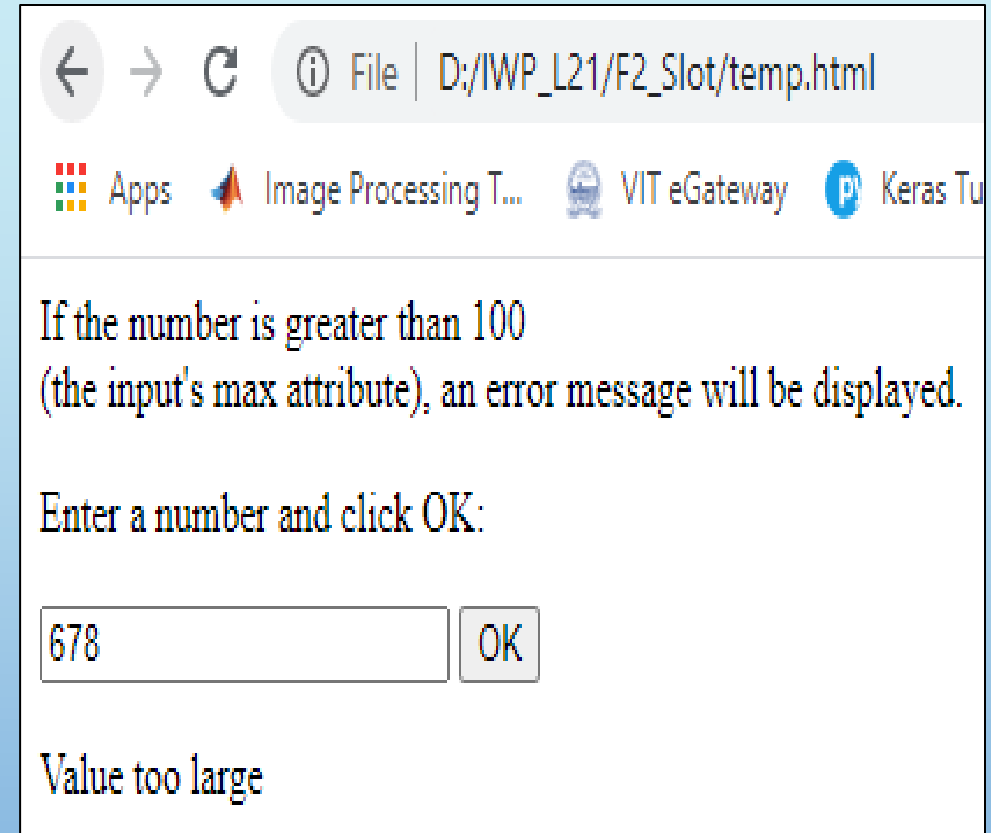
```
function myFunction() {  
  var txt = "";  
  if (document.getElementById("id1").checkValidity()) {  
    txt = "Valid INput";  
  } else {  
    txt = "Input value is out of range";  
  }  
  document.getElementById("demo").innerHTML = txt;  
}
```



HTML – validity property

```
<p>Enter a number and click OK:</p>
<input id="id1" type="number" max="100">
<button onclick="myFunction()">OK</button>
<p>If the number is greater than 100<br> (the input's max
attribute), an error message will be displayed.</p>
<p id="demo"></p>

<script>
function myFunction() {
  var txt = "";
  if (document.getElementById("id1").validity.rangeOverflow) {
    txt = "Value too large";
  } else {
    txt = "Input OK";
  }
  document.getElementById("demo").innerHTML = txt;
}
</script>
```



<body>

```
<form action="exercise.html" method="post" onsubmit="return validate();">
```

```
<label for="username">Enter your username</label>
```

```
<input type="text" required name="username" id="username">
```

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

```
</form>
```

```
<script>
```

```
function validate()
```

```
{
```

```
    const uname = document.getElementById("username").value;
```

```
    if(uname.trim().length==0)
```

```
{
```

```
        document.write("Enter a valid username");
```

```
        return false;
```

```
}
```

```
    return true;
```

```
}
```

JavaScript – Regular Expressions

A regular expression is an object that describes a pattern of characters. The JavaScript **RegExp** class represents regular expressions, and both **String** and **RegExp** define methods that use regular expressions to perform powerful pattern-matching and search-and-replace functions on text.

A regular expression could be defined with the **RegExp ()** constructor, as given below:

```
var pattern = new RegExp(pattern[, attributes]);  
e.g., var mobNum= new RegExp (/^[0-9]{10}+$/);
```

pattern – A string that specifies the pattern of the regular expression or another regular expression.

attributes – An optional string containing any of the "g", "i", and "m" attributes that specify global, case-insensitive, and multi-line matches, respectively.

- A **REGULAR EXPRESSION** IS A SEQUENCE OF CHARACTERS THAT FORMS A SEARCH PATTERN. **REGULAR EXPRESSIONS** CAN BE USED TO PERFORM ALL TYPES OF TEXT SEARCH AND TEXT REPLACEMENT OPERATIONS
- A REGULAR EXPRESSION CAN BE A SINGLE CHARACTER OR A MORE COMPLICATED PATTERN.
- **SYNTAX:**
- /PATTERN/MODIFIERS;
- CLIENT SIDE FORM VALIDATION USUALLY DONE WITH JAVASCRIPT.
- FORM DATA THAT TYPICALLY ARE CHECKED BY A JAVASCRIPT COULD BE:
 - REQUIRED FIELDS
 - VALID USER NAME
 - VALID PASSWORD
 - VALID EMAIL ADDRESS
 - VALID PHONE NUMBER

JAVASCRIPT REGULAR EXPRESSIONS

- `<!DOCTYPE HTML>`
- `<HTML>`
- `<BODY>`
- `<H2>JAVASCRIPT REGULAR EXPRESSIONS</H2>`
- `<P>DO A CASE-INSENSITIVE SEARCH FOR "W3SCHOOLS" IN A STRING:</P>`
- `<P ID="DEMO"></P>`
- `<SCRIPT>`
- `LET TEXT = "VISIT EDIFY3SCHOOLS";`
- `LET RESULT = TEXT.MATCH(/EDIFY3SCHOOLS/I);`
- `DOCUMENT.GETELEMENTBYID("DEMO").INNERHTML = RESULT;`
- `</SCRIPT> </BODY> </HTML>`

OUTPUT: EDIFY3SCHOOLS

REGEXP I MODIFIER

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Regular Expression</h1>
<h2>The i Modifier</h2>

<p>Perform a case-insensitive match:</p>

<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>
```

JavaScript Regular Expression

The i Modifier

Perform a case-insensitive match:

W3Schools

JavaScript- Regular Expression – Meta Characters

| Meta Character | Description | Meta Character | Description |
|----------------|----------------------------------|----------------|---|
| + | One or more occurrences | \s | White space character like space, tab, new |
| * | Zero or more occurrences | \S | Non white space character |
| ? | Exactly the presence of one time | \w | Word character(a-z,A-Z,0-9,_) |
| {n} | Length of exactly n | \W | Non-word character |
| {m,n} | Length from 'm' to 'n' | \d | digit[0-9] |
| {m,} | Minimum length of m | \D | Non-digit |
| {,n} | Maximum length of n | ^ | Indicates starting of a pattern when it is outside of a group |

JavaScript- Regular Expressions

| Expression | Description |
|--------------|--|
| [...] | Any one character between the brackets. |
| [^...] | Any one character not between the brackets. |
| [0-9] | It matches any decimal digit from 0 through 9. |
| [a-z] | It matches any character from lowercase a through lowercase z. |
| [A-Z] | It matches any character from uppercase A through uppercase Z. |
| [a-Z] | It matches any character from lowercase a through uppercase Z. |
| ([a-z][0-9]) | Sub Group |

Regular Expression – Methods

test() – is a method of RegExp class to validate the predefined regular expression for the input. It returns true if the given value has the predefined format, otherwise false

Syntax:

```
Var regExpPat = new RegExp(Pattern);  
Var result= regExpPat.test("value");
```

E.g.

```
var mobPatt = new RegExp(/^[6-9]{1}[0-9]{9}$/); //mobile number pattern  
var mobNum1 = 9867541231;  
var res = mobPatt.test(mobNum1);  
If (res == true)  
    alert("mobile number is correct");  
Else  
    alert("Mobile number is in wrong format");
```

JavaScript –Name and Mobile Number Validation

```
<html>
Enter Your Name: <input type="text" id="userName" pattern="([A-Z][a-z]+ )+[A-Z]\."><br>
Enter Your Mobile Number: <input type="text" id="mobNum" pattern="[0-9]{10}"><br>
<button onclick="checkName()">CHECK</button>
<script>
function checkName(){
    //var regExpName = new RegExp(/^[A-Z]{1}[a-z]+ )+[A-Z]\.$/);

    if (document.getElementById("userName").validity.patternMismatch)
        alert("Name is in wrong pattern");

    if (document.getElementById("mobNum").validity.patternMismatch)
        alert("Mobile number is in wrong format");
    else
        alert("valid mobile Number");
}
</script></html>
```

JAVASCRIPT REGULAR EXPRESSIONS

- `<!DOCTYPE HTML>`
- `<HTML>`
- `<BODY>`
- `<H2>JAVASCRIPT REGULAR EXPRESSIONS</H2>`
- `<P>DO A GLOBAL SEARCH FOR "IS" IN A STRING:</P>`
- `<P ID="DEMO"></P>`
- `<SCRIPT>`
- `LET TEXT = "IS THIS ALL THERE IS?";`
- `LET RESULT = TEXT.MATCH(/IS/G);`
- `DOCUMENT.GETELEMENTBYID("DEMO").INNERHTML = RESULT;`
- `</SCRIPT>`
- `</BODY>`
- `</HTML>` OUTPUT: DO A GLOBAL SEARCH FOR "IS" IN A STRING:
- IS,IS

JAVASCRIPT REGULAR EXPRESSIONS

- `<!DOCTYPE HTML>`
- `<HTML>`
- `<BODY>`
- `<H2>JAVASCRIPT REGULAR EXPRESSIONS</H2>`
- `<P>DO A MULTILINE SEARCH FOR "IS" AT THE BEGINNING OF EACH LINE IN A STRING:</P>`
- `<P ID="DEMO"></P>`
- `<SCRIPT>`
- `LET TEXT = "\NIS TH\NIS IT ?";`
- `LET RESULT = TEXT.MATCH(/^IS/M);`
- `DOCUMENT.GETELEMENTBYID("DEMO").INNERHTML = "RESULT IS: " + RESULT;`
- `</SCRIPT>`
- `</BODY>`
- `</HTML>`
- OUTPUT:DO A MULTILINE SEARCH FOR "IS" AT THE BEGINNING OF EACH LINE IN A STRING:
RESULT IS: IS

JAVASCRIPT REGULAR EXPRESSIONS

- `<!DOCTYPE HTML>`
 - `<HTML>`
 - `<BODY>`
 - `<H2>JAVASCRIPT REGULAR EXPRESSIONS</H2>`
 - `<P>DO A GLOBAL SEARCH FOR DIGITS IN A STRING:</P>`
 - `<P ID="DEMO"></P>`
 - `<SCRIPT>`
 - `LET TEXT = "GIVE 100%!";`
 - `LET RESULT = TEXT.MATCH(/\d/G);`
 - `DOCUMENT.GETELEMENTBYID("DEMO").INNERHTML = RESULT;`
 - `</SCRIPT>`
 - `</BODY>`
 - `</HTML>`
- OUTPUT:DO A GLOBAL SEARCH FOR DIGITS IN A STRING: 1,0,0

JAVASCRIPT REGULAR EXPRESSIONS

- `<!DOCTYPE HTML>`
- `<HTML>`
- `<BODY>`
- `<H2>JAVASCRIPT REGULAR EXPRESSIONS</H2>`
- `<P>DO A GLOBAL SEARCH FOR AT LEAST ONE "O" IN A STRING:</P>`
- `<P ID="DEMO"></P>`
- `<SCRIPT>`
- `LET TEXT = "HELLOOO WORLD! HELLO W3SCHOOLS!";`
- `LET RESULT = TEXT.MATCH(/O+/G);`
- `DOCUMENT.GETELEMENTBYID("DEMO").INNERHTML = RESULT;`
- `</SCRIPT>`
- `</BODY>`
- `</HTML>`

OUTPUT:DO A GLOBAL SEARCH FOR AT LEAST ONE "O" IN A STRING:

O O O O O O O

EMAIL VALIDATION

- FUNCTION VALIDATE(FORM_ID,EMAIL)
- {
- VAR REG = /^([A-Z a-z0-9_-\.\.])+\@[A-Za-z0-9_-\.\.]+\.[A-Za-z]{2,4}\$/;
- VAR ADDRESS = DOCUMENT.FORMS[FORM_ID].ELEMENTS[EMAIL].VALUE;
- IF(REG.TEST(ADDRESS) == FALSE)
- {
- ALERT('INVALID EMAIL ADDRESS');
- RETURN FALSE;
- }
- }

JavaScript – Objects

A JavaScript object is just a collection of named values. These named values are usually referred to as properties of the object. An object is similar to an array, but the difference is that keys are defined by the user, such as name, age, gender, and so on.

An object can be created with curly brackets `{}` with an optional list of properties. A property is a "key: value" pair, where the key (or *property name*) is always a string, and value (or *property value*) can be any data type, like strings, numbers, Booleans or complex data type like arrays, functions, and other objects.

Additionally, properties with functions as their values are often called methods to distinguish them from other properties.

To access or get the value of a property, you can use the dot (`.`), or square bracket (`[]`) notation, as demonstrated in the following example:

JavaScript – Objects

Object Syntax:

```
var objName={key1:value1,key2:value2,...,keyn: value}
```

E.g.

```
var person = { name: "Peter",  
               age: 28,  
               gender: "Male",  
               displayName: function() {  
                   alert(this.name); }  
};
```

Accessing:

```
alert(person.name) or alert(person["name"])
```

JavaScript – Objects

Objects can be iterated through the key-value pairs using the for...in loop. This loop is specially optimized for iterating over object's properties

```
var person = { name: "Peter",  
               age: 28,  
               gender: "Male" }; // Iterating  
over object properties  
for(var i in person) {  
  document.write(person[i] + "<br>");  
}
```

JavaScript – Objects

can set the new properties or update the existing one using the dot (.) or bracket ([]) notation, as demonstrated in the following example

```
var person = { name: "Peter", age: 28, gender: "Male" };
```

```
// Setting a new property
```

```
person.country = "United States";
```

```
document.write(person.country);
```

```
person["email"] = "peterparker@mail.com";
```

```
document.write(person.email);
```

```
// Updating existing property
```

```
person.age = 30;
```

```
document.write(person.age); // Prints: 30
```

```
person["name"] = "Peter Parker";
```

```
document.write(person.name); // prints: Peter Parker
```

JavaScript – Objects

The delete operator deletes both the value of the property and the property itself. After deletion, the property cannot be used

```
<script>
var person = { name: "Peter", age: 28, gender: "Male", eyecolor:
"Brown" };
document.write("properties of object person is.. <br>");
for(i in person){
  document.write("person["+i+"]: "+person[i]); }
delete person.gender;
document.write("<br> after removing gender from person.. <br>");
for(i in person){
  document.write("person["+i+"]: "+person[i]);}
</script>
```

properties of object person is..

person[name]: Peter person[age]: 28 person[gender]: Male person[eyecolor]: Brown

after removing gender from person..

person[name]: Peter person[age]: 28 person[eyecolor]: Brown

JavaScript – Objects

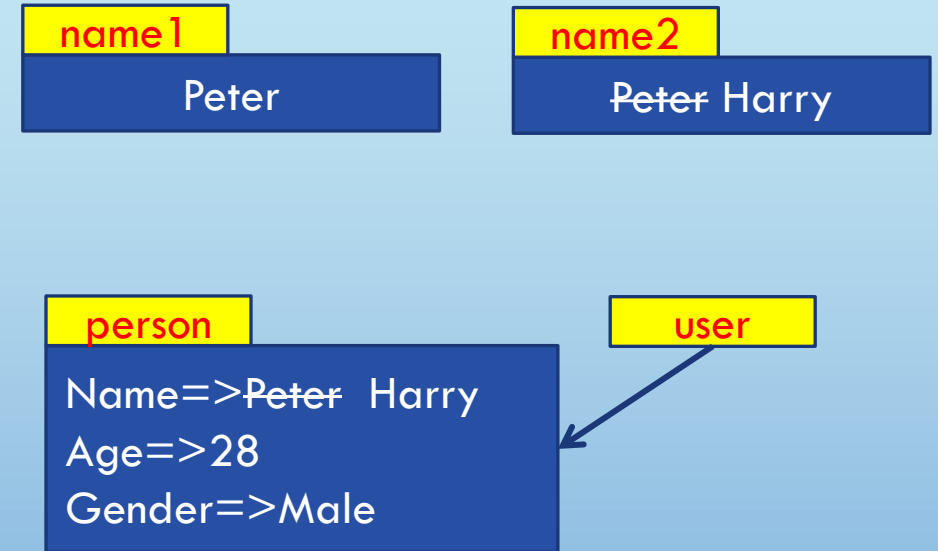
can access an object's method the same way as you would access properties—using the dot notation or using the square bracket notation.

```
var person = { name: "Peter",  
               age: 28,  
               gender: "Male",  
               displayPerson: function() {  
                   document.write(this.name);  
                   document.write(this.age);  
                   document.write(this.gender);  
               }  
};  
person.displayPerson(); // Outputs: Peter 28 Male  
person["displayName"](); // Outputs: Peter 28 Male
```

JavaScript – Objects

JavaScript objects are reference types that mean when you make copies of them, you're really just copying the references to that object.

```
var name1 = "Peter";
var name2 = name1;
name2 = "Harry";
document.write(name1); // Peter
document.write(name2); // Harry
var person = { name: "Peter",
                age: 28,
                gender: "Male" };
var user = person; // Assign person variable to
a new variable user.name = "Harry";
document.write(person.name); // Prints: Harry
document.write(user.name); // Prints: Harry
```



JavaScript – Scope of Variable

- Variables declared within a JavaScript function, become **LOCAL** to the function. They can only be accessed from within the function.
- A variable declared outside a function, becomes **GLOBAL**. All scripts and functions on a web page can access it

```
<script>
var g =100; // global variable

function myFunction() {
  var l =10; // local variable
  document.write("global -"+ g); // global - 100
  document.write("local - "+ l);// local - 10
  g=200;
}

function myFunction2() {
  document.write("global "+ g); // global - 200
  document.write("local "+ l); // error can't access local variable of other functions
}
```

JavaScript – Scope of Variable

```
<script>
var arrName=[10,20,30,40,50];
myF2();

function myFunction() {
  document.write("Array values in myFunction.... <br>");
  for(i in arrName)
    document.write(arrName[i]+" , ");
  document.write("<br>");
  arrName[0]=100;
}
function myF2()
{
  myFunction();
  document.write("Array values after updating in myFunction...
<br>");
  for(i in arrName)
    document.write(arrName[i]+ " , ");
} </script>
```

Array values in myFunction....

10 , 20 , 30 , 40 , 50 ,

Array values after updating in myFunction...

100 , 20 , 30 , 40 , 50 ,

JavaScript – Number Objects

The **Number** object represents numerical data, either integers or floating-point numbers.

The syntax for creating a number object is as follows –

```
var val = new Number(number);
```

In the place of number, if you provide any non-number argument, then the argument cannot be converted into a number, it returns NaN (Not-a-Number).

| <i>Method</i> | <i>Method & Description</i> |
|----------------------------------|---|
| <i>toExponential(num)</i> | Forces a number to display in exponential notation, even if the number is in the range in which JavaScript normally uses standard notation. |
| <i>toFixed(num)</i> | Formats a number with a specific number of digits to the right of the decimal. |
| <i>toPrecision()</i> | Defines how many total digits (including digits to the left and right of the decimal) to display of a number. |
| <i>toString()</i> | Returns the string representation of the number's value. |
| <i>valueOf()</i> | Returns the number's value. |

JavaScript – String Objects

The **String** object lets you work with a series of characters; it wraps Javascript's string primitive data type with a number of helper methods.

Syntax:

```
var val = new String(string);
```

JavaScript – String Objects

```
var str1 = new String("Hello World");
```

| <i>Method</i> | <i>Description</i> | <i>Example</i> |
|---------------|--|--|
| charAt() | Returns the character at the specified index. | str1.charAt(6) → W |
| concat() | Combines the text of two strings and returns a new string. | str1.concat("sa","cd") → Hello Worldsacd |
| indexOf() | Returns the index within the calling String object of the first occurrence of the specified value, or -1 if not found. | str1.indexOf('l') → 2 |
| lastIndexOf() | Returns the index within the calling String object of the last occurrence of the specified value, or -1 if not found. | str1.lastIndexOf('l') → 9 |
| split() | Splits a String object into an array of strings by separating the string into substrings. | str1.split(" ") → Hello World |
| substring() | Returns the characters in a string between two indexes into the string. | str1.substring(2,5) → llo str1.substring(5) → World |
| toLowerCase() | Returns the calling string value converted to lower case. | str1.toLowerCase() → hello world |
| toUpperCase() | Returns the calling string value converted to uppercase. | str1.toUpperCase() → HELLO WORLD |

JavaScript – Date Objects

The **JavaScript date** object can be used to get year, month and day. You can display a timer on the webpage by the help of JavaScript date object. You can use different Date constructors to create date object. It provides methods to get and set day, month, year, hour, minute and seconds.

Can use 4 variant of Date constructor to create date object.

- ***Date()***
- ***Date(milliseconds)***
- ***Date(dateString)***
- ***Date(year, month, day, hours, minutes, seconds, milliseconds)***

`var d1 = new Date()` → returns current date and time

`var d2 = new Date("2021-05-21")`

JavaScript – Date Methods

| Methods | Description |
|--------------------------------|--|
| <code>getDate()</code> | It returns the integer value between 1 and 31 that represents the day for the specified date on the basis of local time. |
| <code>getDay()</code> | It returns the integer value between 0 and 6 that represents the day of the week on the basis of local time. |
| <code>getFullYear()</code> | It returns the integer value that represents the year on the basis of local time. |
| <code>getHours()</code> | It returns the integer value between 0 and 23 that represents the hours on the basis of local time. |
| <code>getMilliseconds()</code> | It returns the integer value between 0 and 999 that represents the milliseconds on the basis of local time. |
| <code>getMinutes()</code> | It returns the integer value between 0 and 59 that represents the minutes on the basis of local time. |
| <code>getMonth()</code> | It returns the integer value between 0 and 11 that represents the month on the basis of local time. |
| <code>getSeconds()</code> | It returns the integer value between 0 and 60 that represents the seconds on the basis of local time. |
| <code>toDateStr()</code> | It returns the date portion of a Date object. |
| <code>toString()</code> | It returns the date in the form of string. |

References

1. www.javatpoint.com
2. www.tutorialspoint.com
3. www.w3schools.com