JavaScript Let

Variables defined with let cannot be Redeclared.

Variables defined with let must be Declared before use.

Variables defined with let have Block Scope.

You cannot accidentally redeclare a variable.

With let you can not do this:	With var you can:

Example
`let x = "John Doe";`
`let x = 0;`
`// SyntaxError: 'x' has already been declared`
	Example
	`var x = "John Doe";`
	`var x = 0;`

Block Scope

JavaScript had only **Global Scope** and **Function Scope**.

ES6 introduced two important new JavaScript keywords: let and const.

These two keywords provide **Block Scope** in JavaScript.

Prof. Ankita Chauhan
CE Department
Assistant Professor
MBIT

Variables declared inside a { } block cannot be accessed from outside the block:
 	
Example
 	
`{`
`  let x = 2;`
`}`
`// x can NOT be used here`
	Variables declared with the var keyword can NOT have block scope.
	Variables declared inside a { } block can be accessed from outside the block.
	
	Example
	
	`{`
	`  var x = 2;`
	`}`
	`// x CAN be used here`

Example:
`<!DOCTYPE html>`
`<html>`
`<body>`
 	
`<h2>Redeclaring a Variable Using var</h2>`
 	
`<p id="demo"></p>`
 	
`<script>`
`let  x = 10;`
`// Here x is 10`
`{`
`  let x = 2;`
`  // Here x is 2`
`}`
`// Here x is 10`
`// Here x is 2`
`document.getElementById("demo").innerHTM`
`L = x;`
`</script>`
 	
`</body>`
`</html>`
 	
Output:
Redeclaring a Variable Using let
 	
2
	Example:
	`<!DOCTYPE html>`
	`<html>`
	`<body>`
	`<h2>Redeclaring a Variable Using let</h2>`
	`<p id="demo"></p>`
	`<script>`
	`var  x = 10;`
	`// Here x is 10`
	`{`
	`var x = 2;`
	`// Here x is 2`
	`}`
	`document.getElementById("demo").innerHTM`
	`L = x;`
	`</script>`
	`</body>`
	`</html>`
	
	Output:
	Redeclaring a Variable Using let
	
	10

JavaScript Const

The const keyword was introduced in ES6 (2015).

Variables defined with const cannot be Redeclared.

Variables defined with const cannot be Reassigned.

Variables defined with const have Block Scope.

Cannot be Reassigned

A const variable cannot be reassigned:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript const</h2>

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

<script>
try {
  const PI = 3.141592653589793;
  PI = 3.14;
}
catch (err) {
  document.getElementById("demo").innerHTML = err;
}
</script>

</body>
</html>
```

Output:

```
JavaScript const

TypeError: Assignment to constant variable.
```

Must be Assigned

JavaScript const variables must be assigned a value when they are declared:

Correct

```
const PI = 3.14159265359;
```

Incorrect

```
const PI;
PI = 3.14159265359;
```

When to use JavaScript const?

As a general rule, always declare a variable with const unless you know that the value will change.

Use const when you declare:

- A new Array
- A new Object
- A new Function
- A new RegExp

Constant Objects and Arrays

The keyword const is a little misleading.

It does not define a constant value. It defines a constant reference to a value.

Because of this you can NOT:

- Reassign a constant value
- Reassign a constant array
- Reassign a constant object

But you CAN:

- Change the elements of constant array
- Change the properties of constant object

Constant Arrays

You can change the elements of a constant array:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript const</h2>

<p>Declaring a constant array does NOT make the elements unchangeable:</p>

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

<script>
// Create an Array:
const cars = ["Saab", "Volvo", "BMW"];

// Change an element:
cars[0] = "Toyota";

// Add an element:
cars.push("Audi");

// Display the Array:
document.getElementById("demo").innerHTML = cars;
</script>

</body>
</html>
```

Output:

```
JavaScript const

Declaring a constant array does NOT make the elements unchangeable:

Toyota,Volvo,BMW,Audi
```

JavaScript Use Strict

"use strict"; Defines that JavaScript code should be executed in "strict mode".

It is not a statement, but a literal expression, ignored by earlier versions of JavaScript.

The purpose of "use strict" is to indicate that the code should be executed in "strict mode".

With strict mode, you can not, for example, use undeclared variables.

All modern browsers support "use strict" except Internet Explorer 9 and lower:

```
<!DOCTYPE html>
<html>
<body>
        <p>"use strict" in a function will only cause errors in that function.</p>
        <p>Activate debugging in your browser (F12) to see the error report.</p>
<script>
        x = 3.14;    // This will not cause an error.
        myFunction();
        function myFunction()
         {
                 "use strict";
                y = 3.14;  // This will cause an error (y is not defined).
        }
</script>
</body>
</html>
```

Output:

```
"use strict" in a function will only cause errors in that function.

Activate debugging in your browser (F12) to see the error report.
```

JavaScript Arrow Function

Arrow functions were introduced in ES6.

Arrow functions allow us to write shorter function syntax:

```
let myFunction = (a, b) => a * b;
```

<!DOCTYPE html>
<html>
<body>
 	
<h2>JavaScript Function</h2>
 	
<p>This example shows the syntax of a function, without the use of arrow function syntax.</p>
 	
<p id="demo"></p>
 	
<script>
var hello;
 	
hello = function() {
  return "Hello World!";
}
document.getElementById("demo").innerHTML = hello();
</script>
</body>
</html>
 	
Output:
JavaScript Function
 	
This example shows the syntax of a function, without the use of arrow function syntax.
 	
Hello World!
	<!DOCTYPE html>
	<html>
	<body>
	
	<h2>JavaScript Arrow Function</h2>
	
	<p>This example shows an Arrow Function without the brackets or the return keyword.</p>
	
	<p id="demo"></p>
	
	<script>
	var hello;
	
	hello = () => "Hello World!";
	
	document.getElementById("demo").innerHTML = hello();
	</script>
	
	</body>
	</html>
	Output:
	JavaScript Arrow Function
	
	This example shows the syntax of an Arrow Function, and how to use it.
	
	Hello World!

	It gets shorter! If the function has only one statement, and the statement returns a value, you can remove the brackets and the return keyword: **Arrow Functions Return Value by Default.** hello = () => "Hello World!"; **If you have parameters, you pass them inside the parentheses:** **Arrow Function With Parameters:** hello = (val) => "Hello " + val;

HTML DOM Document querySelector()

The querySelector() method returns the **first** element that matches a CSS selector.

To return **all** matches (not only the first), use the querySelectorAll() instead.

Both querySelector() and querySelectorAll() throw a SYNTAX_ERR exception if the selector(s) is invalid.

Syntax

document.querySelector(*CSS selectors*)

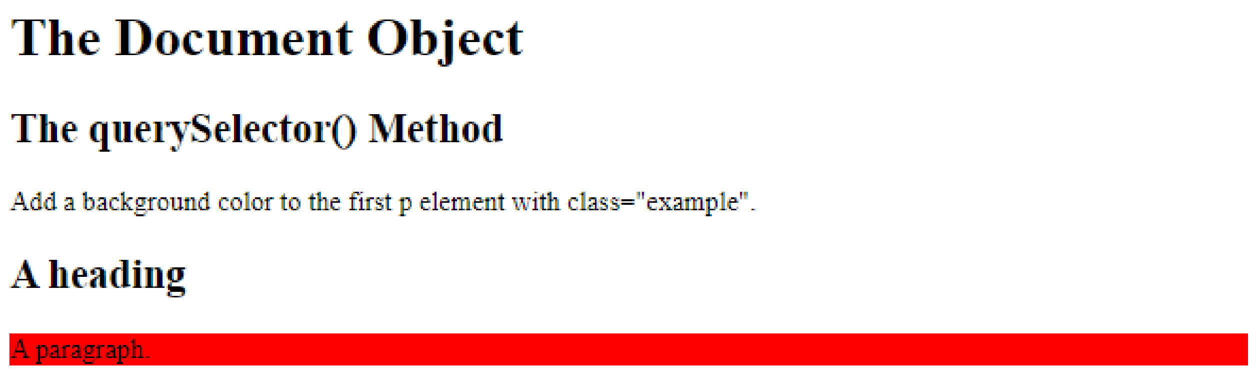
Parameters

Parameter	Description
CSS selectors	Required. One or more CSS selectors.

```
<!DOCTYPE html>
<html>
<body>

<h1>The Document Object</h1>
<h2>The querySelector() Method</h2>

<p >Add a background color to the first p element with class="example".</p>
<h2 class="example">A heading</h2>
<p class="example">A paragraph.</p>
<script>
        document.querySelector("p.example").style.backgroundColor = "red";
```

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

Output:


The Document Object

The querySelector() Method

Add a background color to the first p element with class="example".

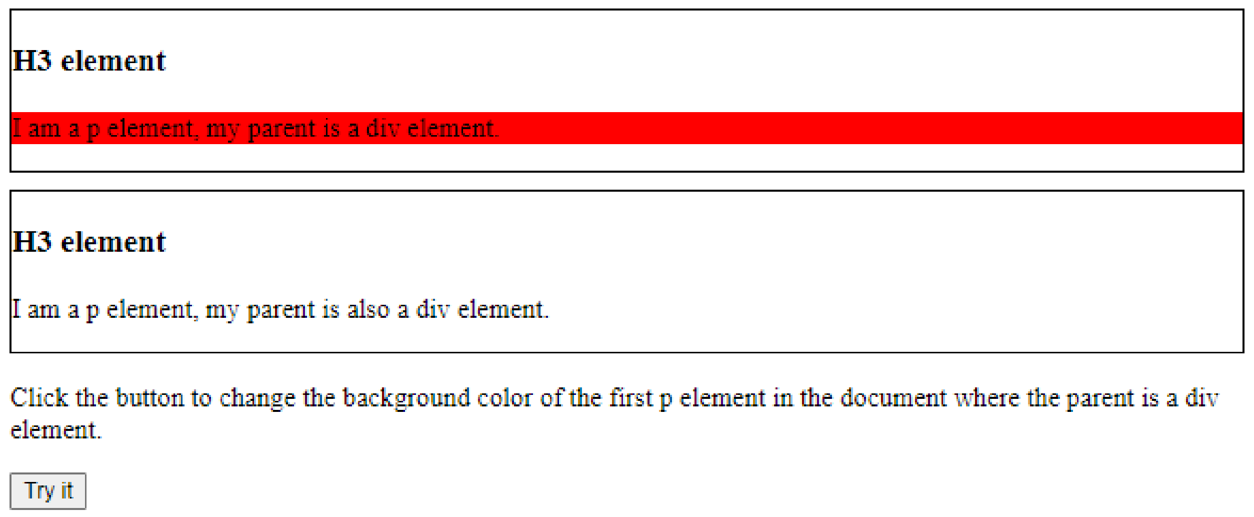
A heading

A paragraph.

```
<!DOCTYPE html>
<html>
<head>
<style>
div {
  border: 1px solid black;
  margin-bottom: 10px;
}
</style>
</head>
<body>

<div>
  <h3>H3 element</h3>
  <p>I am a p element, my parent is a div element.</p>
</div>

<div>
  <h3>H3 element</h3>
  <p>I am a p element, my parent is also a div element.</p>
</div>

<p>Click the button to change the background color of the first p element in the document where the parent is a div element.</p>

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

```
<script>
function myFunction() {
  var x = document.querySelector("div > p");
  x.style.backgroundColor = "red";
}
</script>

</body>
</html>
```

Output:



```
H3 element

I am a p element, my parent is a div element.

H3 element

I am a p element, my parent is also a div element.

Click the button to change the background color of the first p element in the document where the parent is a div element.

Try it
```

JavaScript RegExp

RegExp Object

A regular expression is a **pattern** of characters.

The pattern is used to do pattern-matching **"search-and-replace"** functions on text.

In JavaScript, a **RegExp Object** is a pattern with **Properties** and **Methods**.

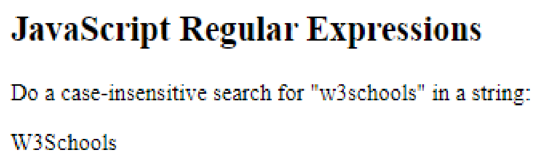
Syntax

/pattern/modifier(s);

Example

```
<!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 W3Schools";
let pattern = /w3schools/i;
let result = text.match(pattern);

document.getElementById("demo").innerHTML = result;
</script>

</body>
</html>
```

Output:



JavaScript Regular Expressions

Do a case-insensitive search for "w3schools" in a string:

W3Schools

Example explained:

w3schools	The pattern to search for
/w3schools/	A regular expression
/w3schools/i	A case-insensitive regular expression

Brackets

Brackets are used to find a range of characters:

Expression	Description
[abc]	Find any character between the brackets
[^abc]	Find any character NOT between the brackets
[0-9]	Find any character between the brackets (any digit)
[^0-9]	Find any character NOT between the brackets (any non-digit)
(x|y)	Find any of the alternatives specified

Metacharacters

Metacharacters are characters with a special meaning:

Metacharacter	Description
.	Find a single character, except newline or line terminator
\w	Find a word character
\W	Find a non-word character
\d	Find a digit
\D	Find a non-digit character

Quantifiers

Quantifier	Description
n+	Matches any string that contains at least one *n*
n*	Matches any string that contains zero or more occurrences of *n*
n?	Matches any string that contains zero or one occurrences of *n*
n$	Matches any string with *n* at the end of it
^n	Matches any string with *n* at the beginning of it

Email Validation:
An email is a string (a subset of ASCII characters) separated into two parts by @ symbol.
a "personal_info" and a domain, that is personal_info@domain.
The length of the personal_info part may be up to 64 characters long and domain name may be up to 253 characters.

The personal_info part contains the following ASCII characters.

Uppercase (A-Z) and lowercase (a-z) English letters.
Digits (0-9).
Characters ! # $ % & ' * + - / = ? ^ _ ` { | } ~

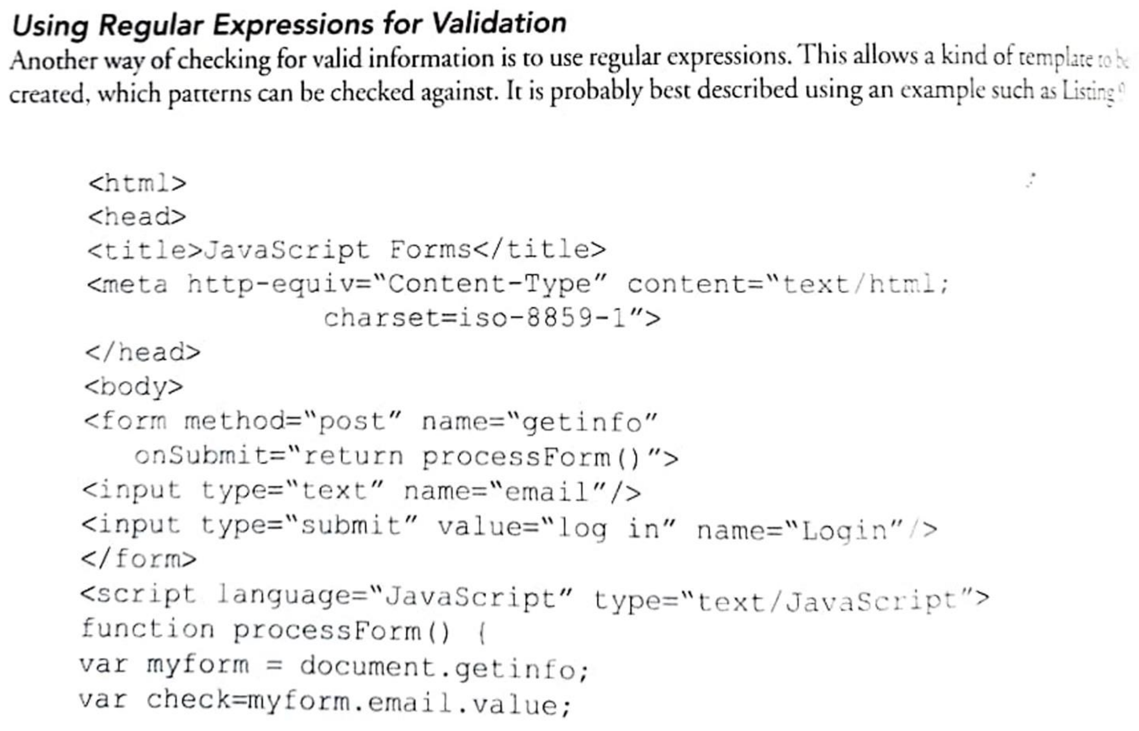
Character . (period, dot or fullstop) provided that it is not the first or last character and it will not come one after the other.

The domain name [for example com, org, net, in, us, info] part contains letters, digits, hyphens, and dots.

Example of valid email id

- mysite@ourearth.com
- my.ownsite@ourearth.org
- mysite@you.me.net

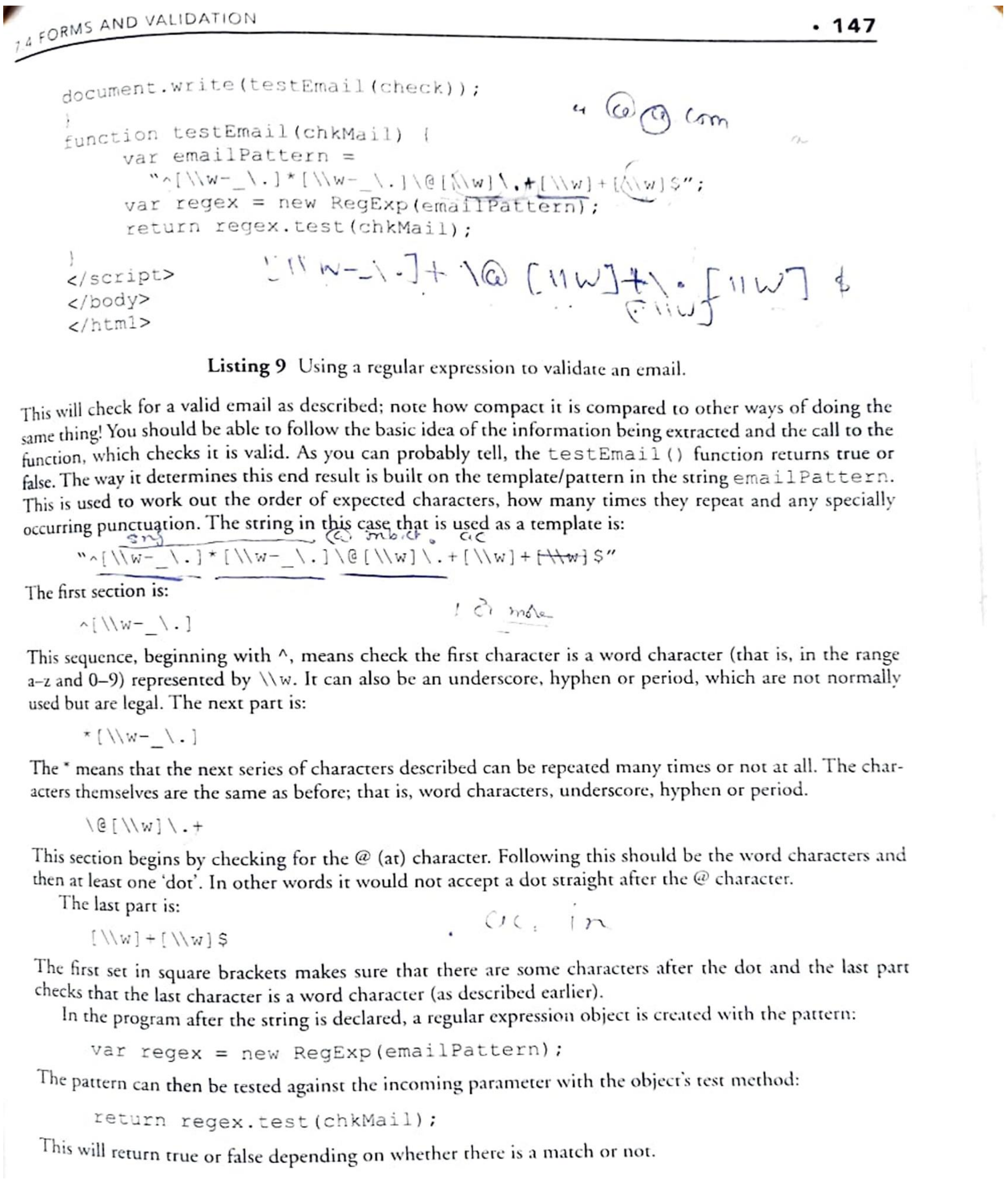
Example:



Using Regular Expressions for Validation

Another way of checking for valid information is to use regular expressions. This allows a kind of template to be created, which patterns can be checked against. It is probably best described using an example such as Listing

```
<html>
<head>
<title>JavaScript Forms</title>
<meta http-equiv="Content-Type" content="text/html;
              charset=iso-8859-1">
</head>
<body>
<form method="post" name="getinfo"
   onSubmit="return processForm()">
<input type="text" name="email"/>
<input type="submit" value="log in" name="Login"/>
</form>
<script language="JavaScript" type="text/JavaScript">
function processForm() {
var myform = document.getinfo;
var check=myform.email.value;
```



```
document.write(testEmail(check));
}
function testEmail(chkMail) {
     var emailPattern =
       "^[\\w-_\.]*[\\w-_\.]\@[\\w]\.+[\\w]+[\\w]$";
     var regex = new RegExp(emailPattern);
     return regex.test(chkMail);
}
</script>
</body>
</html>
```

" @ @ com

^[\\w-_\.]+ \@ [\\w]+\. [\\w] $
[.\\w]

Listing 9 Using a regular expression to validate an email.

This will check for a valid email as described; note how compact it is compared to other ways of doing the same thing! You should be able to follow the basic idea of the information being extracted and the call to the function, which checks it is valid. As you can probably tell, the `testEmail()` function returns true or false. The way it determines this end result is built on the template/pattern in the string `emailPattern`. This is used to work out the order of expected characters, how many times they repeat and any specially occurring punctuation. The string in this case that is used as a template is:

sm @ mbit . ac

```
"^[\\w-_\.]*[\\w-_\.]\@[\\w]\.+[\\w]+~~[\\w]~~$"
```

The first section is:

```
^[\\w-_\.]
```

1 or more

This sequence, beginning with ^, means check the first character is a word character (that is, in the range a–z and 0–9) represented by `\\w`. It can also be an underscore, hyphen or period, which are not normally used but are legal. The next part is:

```
*[\\w-_\.]
```

The * means that the next series of characters described can be repeated many times or not at all. The characters themselves are the same as before; that is, word characters, underscore, hyphen or period.

```
\@[\\w]\.+
```

This section begins by checking for the @ (at) character. Following this should be the word characters and then at least one 'dot'. In other words it would not accept a dot straight after the @ character.

The last part is:

```
[\\w]+[\\w]$
```

. ac. in

The first set in square brackets makes sure that there are some characters after the dot and the last part checks that the last character is a word character (as described earlier).

In the program after the string is declared, a regular expression object is created with the pattern:

```
var regex = new RegExp(emailPattern);
```

The pattern can then be tested against the incoming parameter with the object's test method:

```
return regex.test(chkMail);
```

This will return true or false depending on whether there is a match or not.

Prof. Ankita Chauhan
CE Department
Assistant Professor
MBIT

JSON Data Types

Valid Data Types

In JSON, values must be one of the following data types:

- a string
- a number
- an object (JSON object)
- an array
- a boolean
- *null*

JSON values **cannot** be one of the following data types:

- a function
- a date
- *undefined*

JSON Strings

Strings in JSON must be written in double quotes.

Example

```
{"name":"John"}
```

JSON Numbers

Numbers in JSON must be an integer or a floating point.

Example

```
{"age":30}
```

JSON Objects

Values in JSON can be objects.

Example

```
{
"employee":{"name":"John", "age":30, "city":"New York"}
}
```

JSON Arrays

Values in JSON can be arrays.

Example

```
{
"employees":["John", "Anna", "Peter"]
}
```

JSON Booleans

Values in JSON can be true/false.

Example

```
{"sale":true}
```

JSON null

Values in JSON can be null.

Example

{"middlename":null}

JavaScript HTML DOM

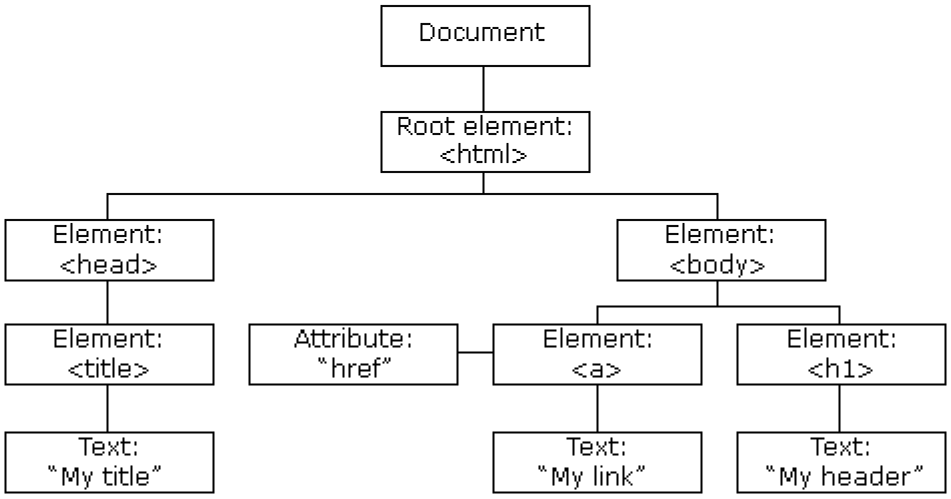
With the HTML DOM, JavaScript can access and change all the elements of an HTML document.

The HTML DOM (Document Object Model)

When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.

The **HTML DOM** model is constructed as a tree of **Objects**:

The HTML DOM Tree of Objects



With the object model, JavaScript gets all the power it needs to create dynamic HTML:

- JavaScript can change all the HTML elements in the page
- JavaScript can change all the HTML attributes in the page
- JavaScript can change all the CSS styles in the page

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

Refer Form Validation Program with attached email.