

FULL STACK DEVELOPMENT

UNIT -3

jQuery Programming: Selectors & Mouse Events, Form Events, DOM Manipulation, Effects & Animation, Traversing & Filtering.

Backend Programming with Node.js: Installation and Simple Server, Express Setup and Routing, Template Engines, Node MongoDB Driver, Setup, Middleware & Routes, Creating the UI, Form Validation and User Register, Password Encryption, Login Functionality, Access Control & Logout

1) Selectors & Events

1.1. jQuery Selectors

jQuery Selectors are used to select and manipulate HTML elements. They are very important part of jQuery library.

With jQuery selectors, you can find or select HTML elements based on their id, classes, attributes, types and much more from a DOM.

In simple words, you can say that selectors are used to select one or more HTML elements using jQuery and once the element is selected then you can perform various operation on that.

All jQuery selectors start with a dollar sign and parenthesis e.g. **\$()**. It is known as the factory function.

The **\$()** factory function

Every jQuery selector start with this sign **\$()**. This sign is known as the factory function. It uses the three basic building blocks while selecting an element in a given document.

S.No.	Selector	Description
-------	----------	-------------

1)	Tag Name:	It represents a tag name available in the DOM. For example: \$('p') selects all paragraphs 'p' in the document.
2)	Tag ID:	It represents a tag available with a specific ID in the DOM. For example: \$('#real-id') selects a specific element in the document that has an ID of real-id.
3)	Tag Class:	It represents a tag available with a specific class in the DOM. For example: \$('.real-class') selects all elements in the document that have a class of real-class.

Let's take a simple example to see the use of Tag selector. This would select all the elements with a tag name

and the background color is set to "pink".

File: firstjquery.html

```

1. <!DOCTYPE html>
2. <html>
3. <head>
4. <title>First jQuery Example</title>
5. <script type="text/javascript" src="jquery.min.js">
6. </script>
7. <script type="text/javascript" language="javascript">
8. $(document).ready(function() {
9. $("p").css("background-color", "pink");
10. });
11. </script>
12. </head>
13. <body>
14. <p>This is first paragraph.</p>
15. <p>This is second paragraph.</p>
16. <p>This is third paragraph.</p>
17. </body>
18. </html>

```

Output:

This is first paragraph.

This is second paragraph.

This is third paragraph.

Note: 1. All of the above discussed selectors can be used alone or with the combination of other selectors.

Note: 2. If you have any conflict with the use of dollar sign \$ in any JavaScript library then you can use jQuery() function instead of factory function \$(). The factory function \$() and the jQuery function is the same.

How to use Selectors

The jQuery selectors can be used single or with the combination of other selectors. They are required at every steps while using jQuery. They are used to select the exact element that you want from your HTML document.

S.No.	Selector	Description
1)	Name:	It selects all elements that match with the given element name.
2)	#ID:	It selects a single element that matches with the given id.
3)	.Class:	It selects all elements that matches with the given class.
4)	Universal(*)	It selects all elements available in a DOM.
5)	Multiple Elements A,B,C	It selects the combined results of all the specified selectors A,B and C.

Different jQuery Selectors

Selector	Example	Description
*	\$("#*")	It is used to select all elements.
#id	\$("#firstname")	It will select the element with id="firstname"
.class	\$(".primary")	It will select all elements with class="primary"

class,.class	\$(".primary,.secondary")	It will select all elements with the class "primary" or "secondary"
element	\$("p")	It will select all p elements.
el1,el2,el3	\$("h1,div,p")	It will select all h1, div, and p elements.
:first	\$("p:first")	This will select the first p element
:last	\$("p:last")	This will select the last p element
:even	\$("tr:even")	This will select all even tr elements
:odd	\$("tr:odd")	This will select all odd tr elements
:first-child	\$("p:first-child")	It will select all p elements that are the first child of their parent
:first-of-type	\$("p:first-of-type")	It will select all p elements that are the first p element of their parent
:last-child	\$("p:last-child")	It will select all p elements that are the last child of their parent
:last-of-type	\$("p:last-of-type")	It will select all p elements that are the last p element of their parent
:nth-child(n)	\$("p:nth-child(2)")	This will select all p elements that are the 2nd child of their parent
:nth-last-child(n)	\$("p:nth-last-child(2)")	This will select all p elements that are the 2nd child of their parent, counting from the last child
:nth-of-type(n)	\$("p:nth-of-type(2)")	It will select all p elements that are the 2nd p element of their parent
:nth-last-of-type(n)	\$("p:nth-last-of-type(2)")	This will select all p elements that are the 2nd p element of their parent, counting from the last child
:only-child	\$("p:only-child")	It will select all p elements that are the only child of their parent
:only-of-type	\$("p:only-of-type")	It will select all p elements that are the only child, of its type, of their parent
parent > child	\$("div > p")	It will select all p elements that are a direct child of a div element

parent descendant	<code>\$("div p")</code>	It will select all p elements that are descendants of a div element
element + next	<code>\$("div + p")</code>	It selects the p element that are next to each div elements
element ~ siblings	<code>\$("div ~ p")</code>	It selects all p elements that are siblings of a div element
<code>:eq(index)</code>	<code>\$("ul li:eq(3)")</code>	It will select the fourth element in a list (index starts at 0)
<code>:gt(no)</code>	<code>\$("ul li:gt(3)")</code>	Select the list elements with an index greater than 3
<code>:lt(no)</code>	<code>\$("ul li:lt(3)")</code>	Select the list elements with an index less than 3
<code>:not(selector)</code>	<code>\$("input:not(:empty)")</code>	Select all input elements that are not empty
<code>:header</code>	<code>\$(":header")</code>	Select all header elements h1, h2 ...
<code>:animated</code>	<code>\$(":animated")</code>	Select all animated elements
<code>:focus</code>	<code>\$(":focus")</code>	Select the element that currently has focus
<code>:contains(text)</code>	<code>\$(":contains('Hello'))"</code>	Select all elements which contains the text "Hello"
<code>:has(selector)</code>	<code>\$("div:has(p)")</code>	Select all div elements that have a p element
<code>:empty</code>	<code>\$(":empty")</code>	Select all elements that are empty
<code>:parent</code>	<code>\$(":parent")</code>	Select all elements that are a parent of another element
<code>:hidden</code>	<code>\$("p:hidden")</code>	Select all hidden p elements
<code>:visible</code>	<code>\$("table:visible")</code>	Select all visible tables
<code>:root</code>	<code>\$(":root")</code>	It will select the document's root element
<code>:lang(language)</code>	<code>\$("p:lang(de)")</code>	Select all p elements with a lang attribute value starting with "de"
<code>[attribute]</code>	<code>\$("[href]")</code>	Select all elements with a href attribute
<code>[attribute=value]</code>	<code>\$("[href='default.htm']")</code>	Select all elements with a href attribute value equal to "default.htm"
<code>[attribute!=value]</code>	<code>\$("[href!='default.htm']")</code>	It will select all elements with a href attribute value not equal to "default.htm"

[attribute\$=value]	\$("#[href\$='.jpg']")	It will select all elements with a href attribute value ending with ".jpg"
[attribute =value]	\$("#[title ='Tomorrow']")	Select all elements with a title attribute value equal to 'Tomorrow', or starting with 'Tomorrow' followed by a hyphen
[attribute^=value]	\$("#[title^='Tom']")	Select all elements with a title attribute value starting with "Tom"
[attribute~=value]	\$("#[title~='hello']")	Select all elements with a title attribute value containing the specific word "hello"
[attribute*=value]	\$("#[title*='hello']")	Select all elements with a title attribute value containing the word "hello"
:input	\$("#:input")	It will select all input elements
:text	\$("#:text")	It will select all input elements with type="text"
:password	\$("#:password")	It will select all input elements with type="password"
:radio	\$("#:radio")	It will select all input elements with type="radio"
:checkbox	\$("#:checkbox")	It will select all input elements with type="checkbox"
:submit	\$("#:submit")	It will select all input elements with type="submit"
:reset	\$("#:reset")	It will select all input elements with type="reset"
:button	\$("#:button")	It will select all input elements with type="button"
:image	\$("#:image")	It will select all input elements with type="image"
:file	\$("#:file")	It will select all input elements with type="file"
:enabled	\$("#:enabled")	Select all enabled input elements
:disabled	\$("#:disabled")	It will select all disabled input elements
:selected	\$("#:selected")	It will select all selected input elements
:checked	\$("#:checked")	It will select all checked input elements

1.2: jQuery Events:

jQuery events are the actions that can be detected by your web application. They are used to create dynamic web pages. An event shows the exact moment when something happens.

These are some examples of events.

- A mouse click
- An HTML form submission
- A web page loading
- A keystroke on the keyboard
- Scrolling of the web page etc.

These events can be categorized on the basis their types:

1.2.1 Mouse Events

- click
- dblclick
- mouseenter
- mouseleave

1.2.2 Keyboard Events

- keyup
- keydown
- keypress

1.2.3 Form Events

- submit
- change
- blur
- focus

1.2.4 Document/Window Events

- load
- unload
- scroll
- resize

Syntax for event methods

Most of the DOM events have an equivalent jQuery method. To assign a click events to all paragraph on a page, do this:

```
1. $("p").click ();
```

The next step defines what should happen when the event fires. You must pass a function to the event.

```
1. $("p").click(  
2. function()  
3. {  
4.    // action goes here!!  
5. }  
6. );
```

1.2.1 Mouse Events

a. jQuery click()

When you click on an element, the click event occurs and once the click event occurs it execute the click () method or attaches a function to run.

It is generally used together with other events of jQuery.

Syntax:

1. `$(selector).click()`

It is used to trigger the click event for the selected elements.

`$(selector).click(function)`

It is used to attach the function to the click event.

Let's take an example to demonstrate jQuery click() event.

1. `<!DOCTYPE html>`
2. `<html>`
3. `<head>`
4. `<script src="jquery.min.js"></script>`
5. `<script>`
6. `$(document).ready(function(){`
7. `$("p").click(function(){`
8. `alert("This paragraph was clicked.");`
9. `});`
10. `});`
11. `</script>`
12. `</head>`
13. `<body>`
14. `<p>Click on the statement.</p>`
15. `</body>`
16. `</html>`

Output:

Click on the statement.

Let's take an example to demonstrate the jquery click() event. In this example, when you click on the heading element, it will hide the current heading.

1. `<!DOCTYPE html>`

```
2. <html>
3. <head>
4. <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"></script>

5. <script>
6. $(document).ready(function(){
7.     $("h1,h2,h3").click(function(){
8.         $(this).hide();
9.     });
10. });
11. </script>
12. </head>
13. <body>
14. <h1>This heading will disappear if you click on this.</h1>
15. <h2>I will also disappear.</h2>
16. <h3>Me too.</h3>
17. </body>
18. </html>
```

Output:

This heading will disappear if you click on this.

I will also disappear.

Me too.

b.jQuery dblclick() method

The **dblclick()** method is used to trigger a dblclick event or attach a function to execute on double-clicking the element. The event occurs when an element is clicked twice in a very short span of time. It is an inbuilt method in [jQuery](#).

Syntax

We can either simply use the **dblclick()** method or can add a function to the **dblclick** event. The syntax of using the **dblclick()** method is given as follows -

Trigger the event for the selected elements

1. `$(selector).dblclick()`

Attach a function

1. `$(selector).dblclick(function)`

The **selector** in the above syntax is the selected element. The parameter **function** mentioned in the above syntax is optional. It is a function to execute when the **dblclick** event occurs. The attached function performs a specific task on double-clicking the element.

Let's see some illustrations of using the **dblclick()** method.

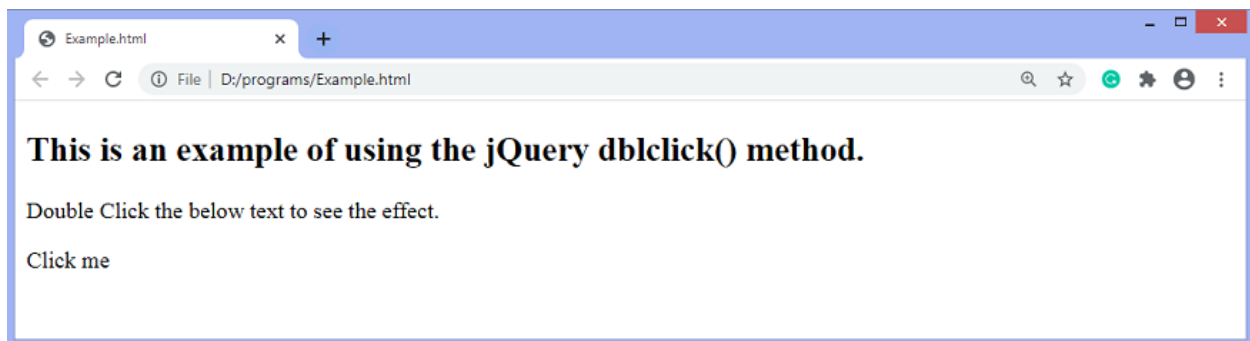
Example1

It is a simple example of using the **dblclick()** method. Here, on double-clicking the given paragraph with the text **Click me**, an alert box will be displayed showing some message.

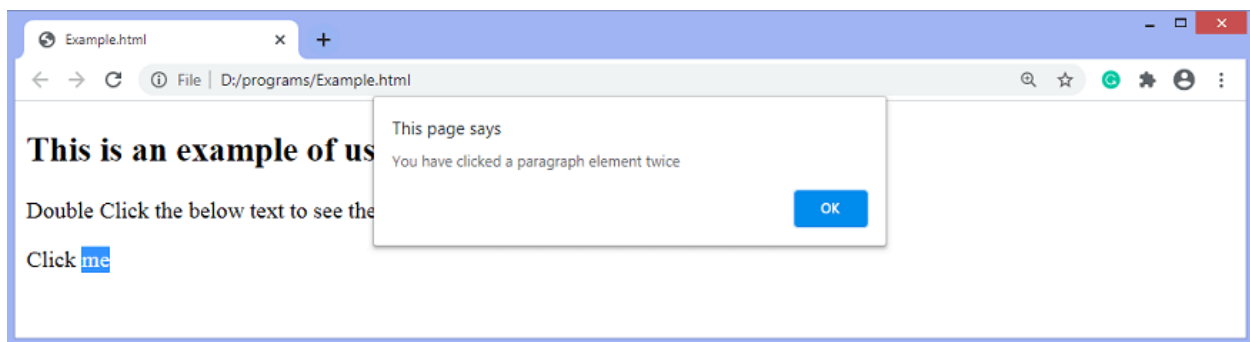
1. `<!DOCTYPE html>`
2. `<html>`
3. `<head>`
4. `<script src = "https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"> </script>`
`>`
5. `</head>`
6. `<body>`
7. `<h2>` This is an example of using the jQuery dblclick() method. `</h2>`
8. `<p>` Double Click the below text to see the effect. `</p>`
9. `<p id = "p1">` Click me `</p>`
10. `<script>`
11. `$(document).ready(function){`
12. `$("#p1").dblclick(function){`

```
13. alert("You have clicked a paragraph element twice");
14. });
15. });
16. </script>
17. </body>
18. </html>
```

After the execution of the above code, the output will be -



After clicking the text **Click me**, the output will be -



Example2

It is another example of the **dblclick()** method. Here, on double-clicking the paragraph with the text **Click me twice**, the style and text of the paragraph will get changed.

```
1. <!DOCTYPE html>
2. <html>
3. <head>
```

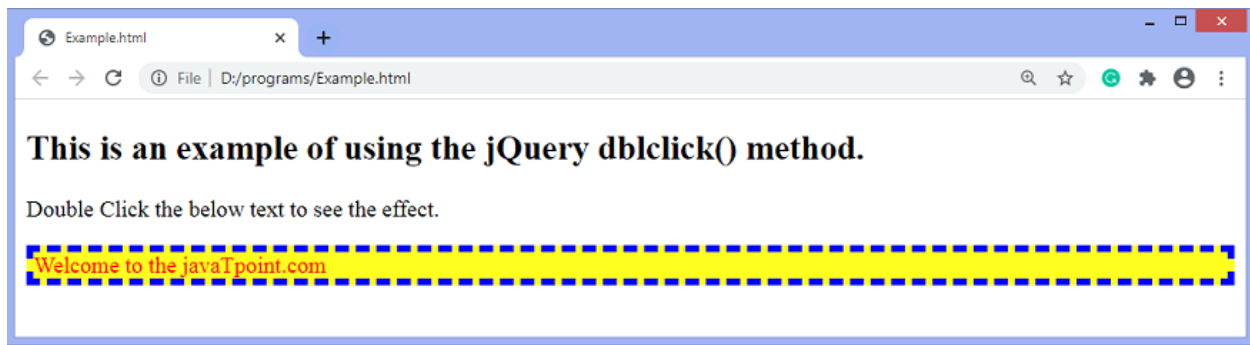
4. `<script src = "https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"> </script>`
5. `</head>`
6. `<body>`
7. `<h2>` This is an example of using the jQuery dblclick() method. `</h2>`
8. `<p>` Double Click the below text to see the effect. `</p>`
9. `<p id = "p1">` Click me twice `</p>`
10. `<script>`
11. `$(document).ready(function(){`
12. `$("#p1").dblclick(function(){`
13. `$("#p1").text("Welcome to the javaTpoint.com").css({ "border": "5px dashed blue",`
`"color": "red", "background-color": "yellow"});`
14. `});`
15. `});`
16. `</script>`
17. `</body>`
18. `</html>`

Output

After the execution of the above code, the output will be -



After clicking the text **Click me twice**, the output will be -



c. jQuery mouseenter()

The mouseenter() method adds an event handler function to an HTML element. This function is executed, when the mouse pointer enters the HTML element.

When you enter your mouse cursor over the selected element, it triggers the mouseenter event and once the mouseenter event is occurred, it executes the mouseenter() method to attach the event handler function to run.

This event is generally used together with mouseleave() event.

Syntax:

1. \$(selector).mouseenter()

It triggers the mouseenter event for selected elements.

1. \$(selector).mouseenter(function)

It adds a function to the mouseenter event.

Parameters of jQuery mouseenter() event

Parameter	Description
Function	It is an optional parameter. It executes itself when the mouseenter event is triggered.

Example of jQuery mouseenter() event

Let's take an example to demonstrate jQuery mouseenter() event.

1. `<!DOCTYPE html>`
2. `<html>`
3. `<head>`
4. `<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"></script>`
5. `<script>`
6. `$(document).ready(function(){`
7. `$("#h1").mouseenter(function(){`
8. `$("div").text("Mouse entered on heading").show().fadeOut(2000);`
9. `});`
10. `});`
11. `</script>`
12. `</head>`
13. `<body>`
14. `<h1 id="h1">Enter this heading.</h1>`
15. `<div></div>`
16. `</body>`
17. `</html>`

Output:

Enter this heading.

jQuery mouseenter() event example 2

Let's see another example of jQuery mouseenter() event.

1. `<!DOCTYPE html>`
2. `<html>`
3. `<head>`
4. `<script src="jquery.min.js"></script>`

```
5. <script>
6. $(document).ready(function(){
7.     $("p").mouseenter(function(){
8.         $("p").css("background-color", "lightgreen");
9.     });
10.  $("p").mouseleave(function(){
11.      $("p").css("background-color", "yellow");
12.  });
13. });
14. </script>
15. </head>
16. <body>
17. <p>Move your mouse cursor over this statement.</p>
18. </body>
19. </html>
20.
```

Output:

Move your mouse cursor over this statement.

d. jQuery mouseleave()

The `mouseleave()` method adds an event handler function to an HTML element. This function is executed, when the mouse pointer leaves the HTML element.

When your mouse cursor leaves the selected element, it triggers the `mouseleave` event and once the `mouseleave` event is occurred, it executes the `mouseleave()` method attached with the event handler function to run.

This event is generally used together with `mouseenter()` event.

Syntax:

1. `$(selector).mouseleave()`

It triggers the mouseleave event for selected elements.

1. `$(selector).mouseleave(function)`

It adds a function to the mouseleave event.

Parameters of jQuery mouseleave() event

Parameter	Description
Function	It is an optional parameter. It executes itself when the mouseleave event is triggered.

Example of jQuery mouseleave() event

Let's take an example to demonstrate jQuery mouseleave() event.

1. `<!DOCTYPE html>`
2. `<html>`
3. `<head>`
4. `<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"></script>`
5. `<script>`
6. `$(document).ready(function(){`
7. `$("#h1").mouseleave(function(){`
8. `$("div").text("Bye Bye... leaving heading").show().fadeOut(2000);`
9. `});`
10. `});`
11. `</script>`
12. `</head>`
13. `<body>`
14. `<h1 id="h1">Enter this heading.</h1>`
15. `<div></div>`
16. `</body>`

17. `</html>`

Output:

Enter this heading.

jQuery mouseleave() event example 2

Let's see another example of jQuery mouseleave() event.

1. `<!DOCTYPE html>`
2. `<html>`
3. `<head>`
4. `<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js"></script>`
5. `<script>`
6. `$(document).ready(function(){`
7. `$("p").mouseenter(function(){`
8. `$("p").css("background-color", "red");`
9. `});`
10. `$("p").mouseleave(function(){`
11. `$("p").css("background-color", "blue");`
12. `});`
13. `});`
14. `</script>`
15. `</head>`
16. `<body>`
17. `<p>Move your mouse cursor over this statement.</p>`
18. `</body>`
19. `</html>`
- 20.

Output:

Move your mouse cursor over this statement.

1.2.2 Keyboard Events

- keyup
- keydown
- keypress

a. jQuery keyup()

The jQuery keyup() event occurs when a keyboard button is released after pressing. This method is executed or attach a function to run when a keyup() event occurs.

Syntax:

1. \$(selector).keyup()

It triggers the keyup event for selected elements.

1. \$(selector).keyup(function)

It adds a function to the keyup event.

Parameters of jQuery keyup() event

Parameter	Description
Function	It is an optional parameter. It is executed itself when the keypress event is triggered.

Example of jQuery keyup() event

Let's take an example to demonstrate jQuery keyup() event.

1. <!DOCTYPE html>
2. <html>

```
3. <head>
4. <script src="https://code.jquery.com/jquery-1.10.2.js"></script>
5. <script>
6. $(document).ready(function(){
7.     $("input").keydown(function(){
8.         $("input").css("background-color", "red");
9.     });
10.    $("input").keyup(function(){
11.        $("input").css("background-color", "yellow");
12.    });
13. });
14. </script>
15. </head>
16. <body>
17. Write something: <input type="text">
18. </body>
19. </html>
```

Output:

Write something:

Note: If you write something in the above text box then the background color will be changed on keydown and keyup.

b. jQuery keydown()

When you press a key on the keyboard, the `keydown()` event is occurred and once the `keydown()` event is occurred, it executes the function associated with `keydown()` method to run.

The `keydown()` event is generally used with two other events.

- **Keypress() event:** It specifies that the key is pressed down.
- **KeyUp() event:** It specifies that the key is released.

Syntax:

1. `$(selector).keydown()`

It triggers the keydown event for selected elements.

`$(selector).keydown(function)`

It adds a function to the keydown event.

Parameters of jQuery keydown() event

Parameter	Description
Function	It is an optional parameter. It is executed itself when the keydown event is triggered.

Example of jQuery keydown() event

Let's take an example to demonstrate jQuery keydown() event.

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4. <script src="https://code.jquery.com/jquery-1.10.2.js"></script>
5. <script>
6. $(document).ready(function(){
7.     $("input").keydown(function(){
8.         $("input").css("background-color", "green");
9.     });
10.    $("input").keyup(function(){
11.        $("input").css("background-color", "violet");
12.    });
13. });
14. </script>
15. </head>
16. <body>
17. Write something: <input type="text">
```

18. `</body>`

19. `</html>`

Output:

Write something:

Note: If you write something in the above text box then the background color will be changed on keydown and keyup.

c. jQuery keypress()

The jQuery keypress () event is occurred when a keyboard button is pressed down. This event is similar to keydown() event. The keypress() method is executed or attach a function to run when a keypress() event occurs.

Syntax:

1. `$(selector).keypress()`

It triggers the keypress event for selected elements.

1. `$(selector).keypress(function)`

It adds a function to the keypress event.

[Play Video](#)

Parameters of jQuery keypress() event

Parameter	Description
Function	It is an optional parameter. It is executed itself when the keypress event is triggered.

Example of jQuery keypress() event

Let's take an example to demonstrate jQuery keypress() event.

1. `<!DOCTYPE html>`
2. `<html>`
3. `<head>`
4. `<script src="https://code.jquery.com/jquery-1.10.2.js"></script>`
5. `<script>`
6. `i = 0;`
7. `$(document).ready(function(){`
8. `$("input").keypress(function(){`
9. `$("span").text (i += 1);`
10. `});`
11. `});`
12. `</script>`
13. `</head>`
14. `<body>`
15. Write something: `<input type="text">`
16. `<p>Keypresses: 0</p>`
17. `</body>`
18. `</html>`

Output:

Write something:

Keypresses: 0

1.2.3 Form Events

- submit
- change
- blur
- focus

a. jQuery submit()

jQuery submit event is sent to the element when the user attempts to submit a form.

This event is only attached to the <form> element. Forms can be submitted either by clicking on the submit button or by pressing the enter button on the keyboard when that certain form elements have focus. When the submit event occurs, the submit() method attaches a function with it to run.

Syntax:

1. \$(selector).submit()

It triggers the submit event for selected elements.

\$(selector).submit(function)

It adds a function to the submit event.

Parameters of jQuery submit() event

Parameter	Description
Function	It is an optional parameter. It is used to specify the function to run when the submit event is executed.

Example of jQuery submit() event

Let's take an example to demonstrate jQuery submit() event.

1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4. <meta charset="utf-8">
5. <title>submit demo</title>
6. <style>
7. p {
8. margin: 0;
9. color: blue;
10. }
11. div,p {
12. margin-left: 10px;


```

13. }
14. span {
15.   color: red;
16. }
17. </style>
18. <script src="https://code.jquery.com/jquery-1.10.2.js"></script>
19. </head>
20. <body>
21. <p>Type 'javatpoint' to submit this form finally.</p>
22. <form action="javascript:alert( 'success!' );">
23.   <div>
24.     <input type="text">
25.     <input type="submit">
26.   </div>
27. </form>
28. <span></span>
29. <script>
30. $( "form" ).submit(function( event ) {
31.   if ( $( "input:first" ).val() === "javatpoint" ) {
32.     $( "span" ).text( "Submitted Successfully." ).show();
33.     return;
34.   }
35.   $( "span" ).text( "Not valid!" ).show().fadeOut( 2000 );
36.   event.preventDefault();
37. });
38. </script>
39. </body>
40. </html>

```

Output:

Type 'javatpoint' to submit this form finally.

b. jQuery change()

jQuery change event occurs when the value of an element is changed. It works only on form fields. When the change event occurs, the change () method attaches a function with it to run.

Note: This event is limited to <input> elements, <textarea> boxes and <select> elements.

- **For select boxes, checkboxes, and radio buttons:** The event is fired immediately when the user makes a selection with the mouse.
- **For the other element types:** The event is occurred when the field loses focus.

Syntax:

1. \$(selector).change()

It triggers the change event for selected elements.

\$(selector).change(function)

It adds a function to the change event.

Parameters of jQuery change() event

Parameter	Description
Function	It is an optional parameter. It is used to specify the function to run when the change event occurs for the selected elements.

Example of jQuery change() event

Let's take an example to demonstrate jQuery change() event.

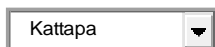
1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4. <meta charset="utf-8">
5. <title>change demo</title>

```

6. <style>
7.   div {
8.     color: red;
9.   }
10. </style>
11. <script src="https://code.jquery.com/jquery-1.10.2.js"></script>
12. </head>
13. <body>
14.   <select id="se" name="actors" >
15.     <option>Uthappa</option>
16.     <option selected="selected">Kattapa</option>
17.     <option>Veerappa</option>
18.     <option>Bahubali</option>
19.     <option>Bhallal Dev</option>
20.     <option>Awantika</option>
21.   </select>
22.   <div id="loc"></div>
23.   <script>
24.     $( "select" ).change(function () {
25.       document.getElementById("loc").innerHTML="You selected: "+document.getElem
26.         entById("se").value;
27.     });
28.   </script>
29. </body>
30. </html>

```

Output:



Let's see another example of jQuery change event where we are providing option to select multiple data using ctrl key.

```

1. <!DOCTYPE html>
2. <html lang="en">

```

```
3. <head>
4.   <meta charset="utf-8">
5.   <title>change demo</title>
6.   <style>
7.     div {
8.       color: red;
9.     }
10.  </style>
11.  <script src="https://code.jquery.com/jquery-1.10.2.js"></script>
12. </head>
13. <body>
14.  <select name="Employees" multiple="multiple">
15.    <option>Uthappa</option>
16.    <option selected="selected">Kattapa</option>
17.    <option>Veerappa</option>
18.    <option selected="selected">Bahubali</option>
19.    <option>Bhallal Dev</option>
20.    <option>Awantika</option>
21.  </select>
22. <div></div>
23. <script>
24. $( "select" )
25. .change(function () {
26.   var str = "";
27.   $( "select option:selected" ).each(function() {
28.     str += $( this ).text() + " ";
29.   });
30.   $( "div" ).text( str );
31. })
32. .change();
33. </script>
34. </body>
35. </html>
```

Output:

Uthappa

c. jQuery blur()

The jQuery blur event occurs when element loses focus. It can be generated by via keyboard commands like tab key or mouse click anywhere on the page.

It makes you enable to attach a function to the event that will be executed when the element loses focus. Originally, this event was used only with form elements like <input>. In latest browsers, it has been extended to include all element types.

The blur () method is often used together with focus () method.

Syntax:

1. \$(selector).blur()

It triggers the blur event for selected elements.

1. \$(selector).blur(function)

It adds a function to the blur event.

Parameters of jQuery blur() event

Parameter	Description
Function	It is an optional parameter. It is used to specify the function to run when the element loses the focus (blur).

Example of jQuery blur() event

Let's take an example to demonstrate jQuery blur() event.

1. <!DOCTYPE html>
2. <html>

```
3. <head>
4. <script src="https://code.jquery.com/jquery-1.10.2.js"></script>
5. <script>
6. $(document).ready(function(){
7.     $("input").blur(function(){
8.         alert("This text box has lost its focus.");
9.     });
10. });
11. </script>
12. </head>
13. <body>
14. Enter your name: <input type="text">
15. </body>
16. </html>
```

Output:

Enter your name:

Note: Write your name in the input field, and then click outside the field to lose focus (blur).

d. jQuery focus()

The jQuery focus event occurs when an element gains focus. It is generated by a mouse click or by navigating to it.

This event is implicitly used to limited sets of elements such as form elements like `<input>`, `<select>` etc. and links `<a href>`. The focused elements are usually highlighted in some way by the browsers.

The focus method is often used together with blur () method.

Syntax:

1. `$(selector).focus()`

It triggers the focus event for selected elements.

1. `$(selector).focus(function)`

It adds a function to the focus event.

Parameters of jQuery focus() event

Parameter	Description
Function	It is an optional parameter. It is used to specify the function to run when the element gets the focus.

Example of jQuery focus() event

Let's take an example to demonstrate jQuery focus() event.

1. `<!doctype html>`
2. `<html lang="en">`
3. `<head>`
4. `<meta charset="utf-8">`
5. `<title>focus demo</title>`
6. `<style>`
7. `span {`
8. `display: none;`
9. `}`
10. `</style>`
11. `<script src="https://code.jquery.com/jquery-1.10.2.js"></script>`
12. `</head>`
13. `<body>`
14. `<p><input type="text"> Focus starts.. Write your name.</p>`
15. `<p><input type="password"> Focus starts.. Write your password.</p>`
16. `<script>`

```
17. $( "input" ).focus(function() {  
18.   $( this ).next( "span" ).css( "display", "inline" ).fadeOut( 2000 );  
19. });  
20. </script>  
21. </body>  
22. </html>
```

Output:

If you want to stop people from writing in text input box in the above example then try the following code.

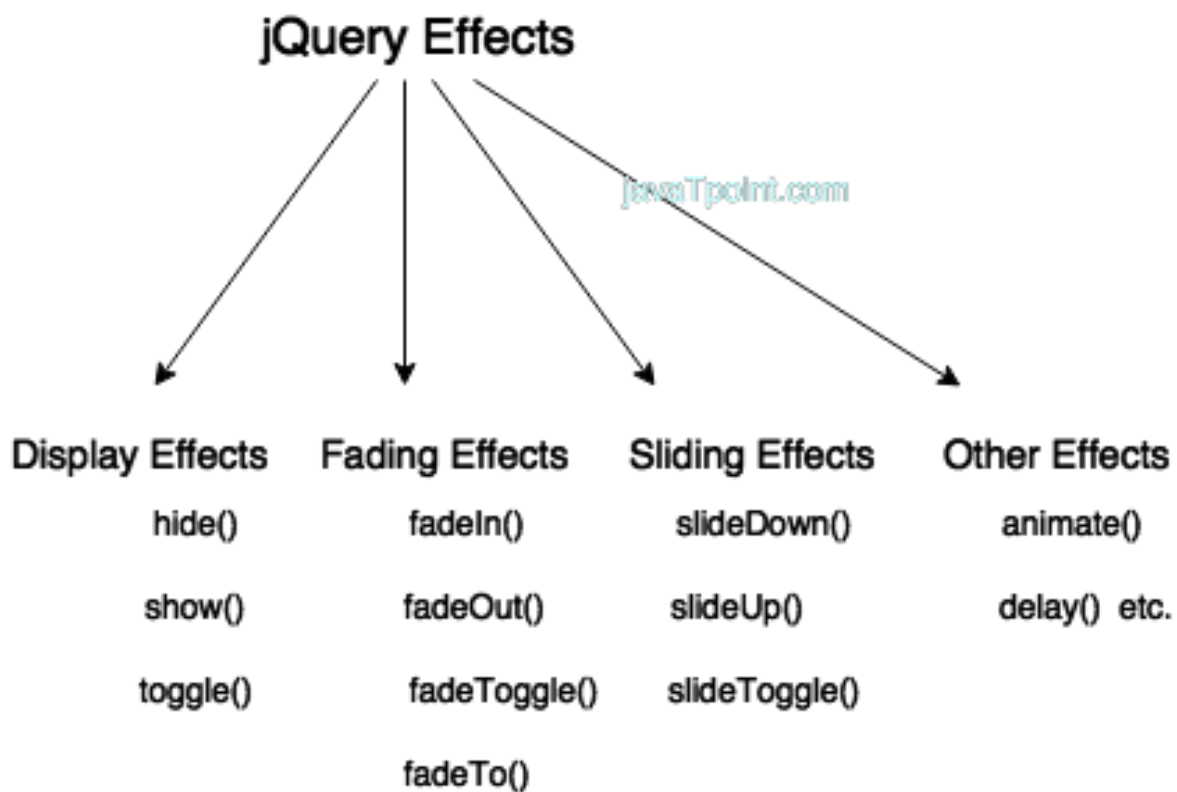
It will disable to write in the text box.

```
1. <!DOCTYPE html>  
2. <html lang="en">  
3. <head>  
4.   <meta charset="utf-8">  
5.   <title>focus demo</title>  
6.   <script src="https://code.jquery.com/jquery-1.10.2.js"></script>  
7. </head>  
8. <body>  
9.   <p><input type="text" value="you can't write"></p>  
10.  <p><input type="password"> </p>  
11. <script>  
12.   $( "input[type=text]" ).focus(function() {  
13.     $( this ).blur();  
14.   });  
15. </script>  
16. </body>  
17. </html>
```


2 jQuery Effects & Animation

jQuery Effects

jQuery enables us to add effects on a web page. jQuery effects can be categorized into fading, sliding, hiding/showing and animation effects.



jQuery provides many methods for effects on a web page. A complete list of jQuery effect methods are given below:

No.	Method	Description
1)	animate()	performs animation.
2	clearQueue()	It is used to remove all remaining queued functions from the selected elements.
3)	delay()	sets delay execution for all the queued functions on the selected elements.
4	dequeue()	It is used to remove the next function from the queue, and then execute the function.
5)	fadeIn()	shows the matched elements by fading it to opaque. In other words, it fades in the selected elements.
6)	fadeOut()	shows the matched elements by fading it to transparent. In other words, it fades out the selected elements.
7)	fadeTo()	adjusts opacity for the matched element. In other words, it fades in/out the selected elements.
8)	fadeToggle()	shows or hides the matched element. In other words, toggles between the fadeIn() and fadeOut() methods.
9)	finish()	It stops, removes and complete all queued animation for the selected elements.
10)	hide()	hides the matched or selected elements.
11)	queue()	shows or manipulates the queue of methods i.e. to be executed on the selected elements.
12)	show()	displays or shows the selected elements.

13)	slideDown()	shows the matched elements with slide.
14)	slideToggle()	shows or hides the matched elements with slide. In other words, it is used to toggle between the slideUp() and slideDown() methods.
15)	slideUp()	hides the matched elements with slide.
16)	stop()	stops the animation which is running on the matched elements.
17)	toggle()	shows or hides the matched elements. In other words, it toggles between the hide() and show() methods.

jQuery hide()

The jQuery hide() method is used to hide the selected elements.

Syntax:

1. \$(selector).hide();
2. \$(selector).hide(speed, callback);
3. \$(selector).hide(speed, easing, callback);

speed: It is an optional parameter. It specifies the speed of the delay. Its possible values are slow, fast and milliseconds.

easing: It specifies the easing function to be used for transition.

callback: It is also an optional parameter. It specifies the function to be called after completion of hide() effect.

Let's take an example to see the jQuery hide effect.

```
<!DOCTYPE html>
<html>
<head>
<script
    src="jquery.min.js"></script>
```

```

<script>
$(document).ready(function(){
    $("#hide").click(function(){
        $("p").hide();
    });
});
</script>
</head>
<body>
    <p>
        <b>This is a little poem:
        </b><br /> Twinkle, twinkle, little star<br />
        How I wonder what you are<br />
        Up above the world so high<br />
        Like a diamond in the sky<br />
        Twinkle, twinkle little star<br />
        How I wonder what you are
    </p>
    <button id="hide">Hide</button>
</body>
</html>

```

Output:

This is a little poem:

Twinkle, twinkle, little star
 How I wonder what you are
 Up above the world so high
 Like a diamond in the sky
 Twinkle, twinkle little star
 How I wonder what you are

Hide

jQuery show()

The jQuery show() method is used to show the selected elements.

Syntax:

1. `$(selector).show();`
2. `$(selector).show(speed, callback);`
3. `$(selector).show(speed, easing, callback);`

speed: It is an optional parameter. It specifies the speed of the delay. Its possible values are slow, fast and milliseconds.

easing: It specifies the easing function to be used for transition.

callback: It is also an optional parameter. It specifies the function to be called after completion of show() effect.

Let's take an example to see the jQuery show effect.

1. `<!DOCTYPE html>`
2. `<html>`
3. `<head>`
4. `<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"></script>`
5. `<script>`
6. `$(document).ready(function(){`
7. `$("#hide").click(function(){`
8. `$("p").hide();`
9. `});`
10. `$("#show").click(function(){`
11. `$("p").show();`
12. `});`
13. `});`
14. `</script>`
15. `</head>`
16. `<body>`
17. `<p>`
18. `This is a little poem:
`
19. `Twinkle, twinkle, little star
`
20. `How I wonder what you are
`

```
21. Up above the world so high<br/>
22. Like a diamond in the sky<br/>
23. Twinkle, twinkle little star<br/>
24. How I wonder what you are
25. </p>
26. <button id="hide">Hide</button>
27. <button id="show">Show</button>
28. </body>
29. </html>
```

Output:

This is a little poem:

Twinkle, twinkle, little star
How I wonder what you are
Up above the world so high
Like a diamond in the sky
Twinkle, twinkle little star
How I wonder what you are

Hide Show

jQuery show() effect with speed parameter

Let's see the example of jQuery show effect with 1500 milliseconds speed.

```
1. $(document).ready(function(){
2.     $("#hide").click(function(){
3.         $("p").hide(1000);
4.     });
5.     $("#show").click(function(){
6.         $("p").show(1500);
7.     });
8. });
```

jQuery toggle()

The jQuery toggle() is a special type of method which is used to toggle between the hide() and show() method. It shows the hidden elements and hides the shown element.

Syntax:

1. \$(selector).toggle();
2. \$(selector).toggle(speed, callback);
3. \$(selector).toggle(speed, easing, callback);
4. \$(selector).toggle(display);

speed: It is an optional parameter. It specifies the speed of the delay. Its possible values are slow, fast and milliseconds.

easing: It specifies the easing function to be used for transition.

callback: It is also an optional parameter. It specifies the function to be called after completion of toggle() effect.

display: If true, it displays element. If false, it hides the element.

Let's take an example to see the jQuery toggle effect.

1. <!DOCTYPE html>
2. <html>
3. <head>
4. <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"> </script>
5. <script>
6. \$(document).ready(function(){
7. \$("button").click(function(){
8. \$("div.d1").toggle();
9. });
10. });
11. </script>
12. </head>

13. `<body>`
14. `<button>Toggle</button>`
15. `<div class="d1" style="border:1px solid black;padding:10px;width:250px">`
16. `<p>`This is a little poem: `
`
17. Twinkle, twinkle, little star`
`
18. How I wonder what you are`
`
19. Up above the world so high`
`
20. Like a diamond in the sky`
`
21. Twinkle, twinkle little star`
`
22. How I wonder what you are`</p>`
23. `</div>`
24. `</body>`
25. `</html>`

Output:

Toggle

This is a little poem:

Twinkle, twinkle, little star
How I wonder what you are
Up above the world so high
Like a diamond in the sky
Twinkle, twinkle little star
How I wonder what you are

jQuery toggle() effect with speed parameter

Let's see the example of jQuery toggle effect with 1500 milliseconds speed.

1. `$(document).ready(function(){`
2. `$("#button").click(function(){`
3. `$("#div.d1").toggle(1500);`
4. `});`
5. `});`

jQuery fadeIn()

jQuery fadeIn() method is used to fade in the element.

Syntax:

1. \$(selector).fadeIn();
2. \$(selector).fadeIn(speed,callback);
3. \$(selector).fadeIn(speed, easing, callback);

speed: It is an optional parameter. It specifies the speed of the delay. Its possible values are slow, fast and milliseconds.

easing: It specifies the easing function to be used for transition.

callback: It is also an optional parameter. It specifies the function to be called after completion of fadeIn() effect.

Let's take an example to demonstrate jQuery fadeIn() effect.

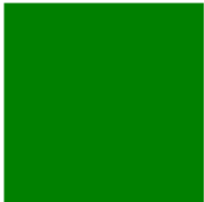
1. <!DOCTYPE html>
2. <html>
3. <head>
4. <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"> </script>
5. <script>
6. \$(document).ready(function(){
7. \$("button").click(function(){
8. \$("#div1").fadeIn();
9. \$("#div2").fadeIn("slow");
10. \$("#div3").fadeIn(3000);

11. `});`
12. `});`
13. `</script>`
14. `</head>`
15. `<body>`
16. `<p>`See the fadeIn() method example with different parameters.`</p>`
17. `<button>`Click to fade in boxes`</button>` `
` `
`
18. `<div id="div1" style="width:80px;height:80px;display:none;background-color:red;">``</div>` `
`
19. `<div id="div2" style="width:80px;height:80px;display:none;background-color:green;">``</div>` `
`
20. `<div id="div3" style="width:80px;height:80px;display:none;background-color:blue;">``</div>`
21. `</body>`
22. `</html>`

Output:

See the `fadeIn()` method example with different parameters.

Click to fade in boxes



jQuery fadeOut()

The jQuery `fadeOut()` method is used to fade out the element.

Syntax:

1. `$(selector).fadeOut();`
2. `$(selector).fadeOut(speed,callback);`
3. `$(selector).fadeOut(speed, easing, callback);`

speed: It is an optional parameter. It specifies the speed of the delay. Its possible values are slow, fast and milliseconds.

easing: It specifies the easing function to be used for transition.

callback: It is also an optional parameter. It specifies the function to be called after completion of fadeOut() effect.

Let's take an example to demonstrate jQuery fadeOut() effect.

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4. <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"></script>

5. <script>
6. $(document).ready(function(){
7.     $("button").click(function(){
8.         $("#div1").fadeOut();
9.         $("#div2").fadeOut("slow");
10.        $("#div3").fadeOut(3000);
11.    });
12. });
13. </script>
14. </head>
15. <body>
16. <p>See the fadeOut() method example with different parameters.</p>
17. <button>Click to fade out boxes</button><br><br>
18. <div id="div1" style="width:80px;height:80px;background-color:red;"></div><br>
19. <div id="div2" style="width:80px;height:80px;background-
    color:green;"></div><br>
20. <div id="div3" style="width:80px;height:80px;background-color:blue;"></div>
21. </body>
22. </html>
```

Output:

See the fadeOut() method example with different parameters.

Click to fade out boxes

jQuery animations

The jQuery `animate()` method provides you a way to create custom animations.

Syntax:

1. `$(selector).animate({params}, speed, callback);`

Here, **params** parameter defines the CSS properties to be animated.

The **speed** parameter is optional and specifies the duration of the effect. It can be set as "slow" , "fast" or milliseconds.

The **callback** parameter is also optional and it is a function which is executed after the animation completes.

Let's take a simple example to see the animation effect.

1. `<!DOCTYPE html>`
2. `<html>`
3. `<head>`
4. `<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"></script>`
5. `<script>`
6. `$(document).ready(function(){`
7. `$("button").click(function(){`
8. `$("div").animate({left: '450px'});`
9. `});`
10. `});`
11. `</script>`
12. `</head>`
13. `<body>`
14. `<button>Start Animation</button>`
15. `<p>A simple animation example:</p>`
16. `<div style="background:#98bf21;height:100px;width:100px;position:absolute;"></div>`
17. `</body>`
18. `</html>`

Output:

Start Animation

A simple animation example:

Note: The default position of all HTML elements is static. If you want to manipulate their position, set the CSS position property to the element to relative, fixed or absolute.

jQuery animate() method using multiple properties

You can use multiple properties to animate at the same time.

1. `<!DOCTYPE html>`
2. `<html>`
3. `<head>`
4. `<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"> </script>`
5. `<script>`
6. `$(document).ready(function(){`
7. `$("button").click(function(){`
8. `$("div").animate({`
9. `left: '250px',`
10. `opacity: '0.5',`
11. `height: '150px',`
12. `width: '150px'`
13. `});`
14. `});`
15. `});`
16. `</script>`
17. `</head>`
18. `<body>`
19. `<button>Start Animation</button>`
20. `<div style="background:#125f21;height:100px;width:100px;position:absolute;"> </div>`
21. `</body>`
22. `</html>`

Output:

Start Animation

jQuery animate() method using relative values

You can also define relative values (it is relative to the element's current value) by putting += or -= in front of the value.

1. `<!DOCTYPE html>`
2. `<html>`
3. `<head>`
4. `<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"></script>`
5. `<script>`
6. `$(document).ready(function(){`
7. `$("button").click(function(){`
8. `$("div").animate({`
9. `left: '250px',`
10. `height: '+=150px',`
11. `width: '+=150px'`
12. `});`
13. `});`
14. `});`
15. `</script>`
16. `</head>`
17. `<body>`
18. `<button>Start Animation</button>`
19. `<div style="background:#98bf21;height:100px;width:100px;position:absolute;"></div>`
20. `</body>`
21. `</html>`

Output:

Start Animation

jQuery animate() method using predefined value

You can also specify a property's animation value as "show" , "hide" , or "toggle".

In this example, we are using "toggle" value for height, it means it will show/hide the selected element.

1. `<!DOCTYPE html>`
2. `<html>`
3. `<head>`
4. `<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"></script>`
5. `<script>`
6. `$(document).ready(function(){`
7. `$("button").click(function(){`
8. `$("div").animate({`
9. `height: 'toggle'`
10. `});`
11. `});`
12. `});`
13. `</script>`
14. `</head>`
15. `<body>`
16. `<button>Start Animation</button>`
17. `<div style="background:#98bf21;height:100px;width:100px;position:absolute;"></div>`
18. `</body>`
19. `</html>`

Output:

Start Animation

jQuery Color animation

You can also animate the properties of elements between colors.

1. `<!doctype html>`
2. `<html lang="en">`
3. `<head>`
4. `<meta charset="utf-8">`
5. `<title>jQuery UI Effects - Animate demo</title>`
6. `<link rel="stylesheet" href="http://code.jquery.com/ui/1.11.4/themes/smoothness/jquery-ui.css">`
7. `<script src="http://code.jquery.com/jquery-1.10.2.js"></script>`
8. `<script src="http://code.jquery.com/ui/1.11.4/jquery-ui.js"></script>`
9. `<style>`
10. `.toggler { width: 500px; height: 200px; position: relative; }`
11. `#button { padding: .5em 1em; text-decoration: none; }`
12. `#effect { width: 240px; height: 135px; padding: 0.4em; position: relative; background: #fff; }`
13. `#effect h3 { margin: 0; padding: 0.4em; text-align: center; }`
14. `</style>`
15. `<script>`
16. `$(function() {`
17. `var state = true;`
18. `$("#button").click(function() {`
19. `if (state) {`
20. `$("#effect").animate({`
21. `backgroundColor: "#aa0000",`
22. `color: "#fff",`
23. `width: 500`
24. `}, 1000);`
25. `} else {`
26. `$("#effect").animate({`
27. `backgroundColor: "#fff",`

```

28.     color: "#000",
29.     width: 240
30. }, 1000 );
31. }
32.     state = !state;
33. });
34. });
35. </script>
36. </head>
37. <body>
38. <div class="toggler">
39. <div id="effect" class="ui-widget-content ui-corner-all">
40. <h3 class="ui-widget-header ui-corner-all">Animate</h3>
41. <p>Javatpoint.com is the best tutorial website to learn Java and other programming languages.</p>
42. </div>
43. </div>
44. <button id="button" class="ui-state-default ui-corner-all">Toggle Effect</button>
45. </body>
46. </html>

```

jQuery Traversing and Filtering[10 marks]

In jQuery, traversing means moving through or over the HTML elements to find, filter or select a particular or entire element.

Based on the traversing purposes following methods are Categorized as follows:

Tree Traversing:

Ancestors:

- **parent()**
it gives parent element of specified selector

Syntax:

```
$(selector).parent();
```

- **parents()**
it gives all ancestor elements of the specified selector.
Syntax:
`$(selector).parents();`
- **parentsUntil()**
it gives all ancestor elements between specified selector and arguments.
Syntax:
 - `$(selector).parentsUntil(selector, filter element)`
 - `$(selector).parentsUntil(element, filter element)`
- **offsetParent()**
it gives the first positioned parent element of specified selector.
Syntax:
`$(selector).offsetParent();`
- **closest()**
it gives the first ancestor of the specified selector.
Syntax:
 - `$(selector).closest(selector);`
 - `$(selector).closest(selector, context);`
 - `$(selector).closest(selection);`
 - `$(selector).closest(element);`

Descendants:

- **children()**
it gives the children of each selected elements, optionally filtered by a selector.
Syntax:
 - `$(selector).children();`
- **find()**
it gives descendant elements of specified elements, filtered by a selector, jQuery object, or element.
Syntax:
`$(selector).find('selector to find');`

Siblings:

- **siblings()**
it gives all siblings of the specified selector.
Syntax:
`$(selector).siblings();`

- **next()**
it gives the next sibling element of the specified selector.
Syntax:
 - `$(selector).next();`
- **nextAll()**
it gives all next sibling elements of the specified selector.
Syntax:
 - `$(selector).nextAll();`
- **nextUntil()**
it gives all next sibling elements between specified selector and arguments.
Syntax:
 - `$(selector).nextUntil();`
- **prev()**
it gives the previous sibling element of the specified selector.
Syntax:
 - `$(selector).prev(selector);`
 - `$(selector).prev();`
- **prevAll()**
it gives all previous sibling elements of the specified selector.
Syntax:
 - `$(selector).prevAll(selector, filter element)`
 - `$(selector).prevAll(element, filter element)`
- **prevUntil()**
it gives all previous sibling elements between specified selector and arguments.
Syntax:
 - `$(selector).prevUntil(selector, filter element)`
 - `$(selector).prevUntil(element, filter element)`

Filtering

- **first()**
it gives the first element of the specified selector.
Syntax:
 - `$(selector).first();`
- **last()**
it gives the last element of the specified selector.
Syntax:
 - `$(selector).last();`

- **eq()**
it gives an element with a specific index number of the specified selector.
Syntax:
 - `$(selector).eq(index);`
 - `$(selector).eq(indexFromEnd);`
- **filter()**
it remove/detect an elements that are matched with specified selector.
Syntax:
 - `$(selector).filter(selector)`
 - `$(selector).filter(function)`
 - `$(selector).filter(selection)`
 - `$(selector).filter(elements)`
- **has()**
it gives all elements that have one or more elements within, that are matched with specified selector.
Syntax:
 - `$(selector).has(selector);`
- **is()**
it checks if one of the specified selector is matched with arguments.
Syntax:
 - `.is(selector)`
 - `.is(function)`
 - `.is(selection)`
 - `.is(elements)`
- **map()**
Pass each element in the current matched set through a function, producing a new jQuery object containing the return values
Syntax:
 - `.map(callback)`
- **slice()**
it selects a subset of specified selector based on its argument index or by start and stop value.
Syntax:
 - `$(selector).slice(start, end);`
 - `$(selector).slice(start);`

Miscellaneous Traversing

- **add()**
it add all elements to set of matched elements to manipulate them at the same time.
Syntax:
`$(selector).add(selector to add);`
- **addBack()**
it add the previous set of elements on the stack to the current set, optionally filtered by a selector.
Syntax:
`$(selector).addBack();`
- **andSelf()**
Deprecated 1.8 which is alias of addBack().
Syntax:
`$(selector).addSelf();`
- **contents()**
it gives all direct children, including text and comment nodes, of the specified selector.
Syntax:
`$(selector).contents();`
- **not()**
it gives all elements that do not match with specified selector.
Syntax:
`$(selector).not(selector);`
- **end()**
it is most recent filtering operation in the current chain and return the set of matched elements to its previous state and it doesn't accept any arguments.
Syntax:
 - `$(selector).each(callback function);`

Collection Manipulation

- **each()**
it iterates over the DOM elements and execute call back function

Example 1:

```
<!DOCTYPE html>
<html>

<head>
<style>
.siblings * {
  display: block;
  border: 2px solid lightgrey;
```

```

        color: lightgrey;
        padding: 5px;
        margin: 15px;
    }
</style>
<script src="jquery.min.js">
</script>
<script>
    $(document).ready(function() {
        $("h2").siblings().css({
            "color": "red",
            "border": "2px solid red"
        });
        $("h2").parent().css({
            "color": "green",
            "border": "2px solid blue"
        });
        $("p").first().css(
            "background-color", "yellow");
        $("p").has("span").css(
            "background-color", "indigo");
    });
</script>
</head>

<body class="siblings">

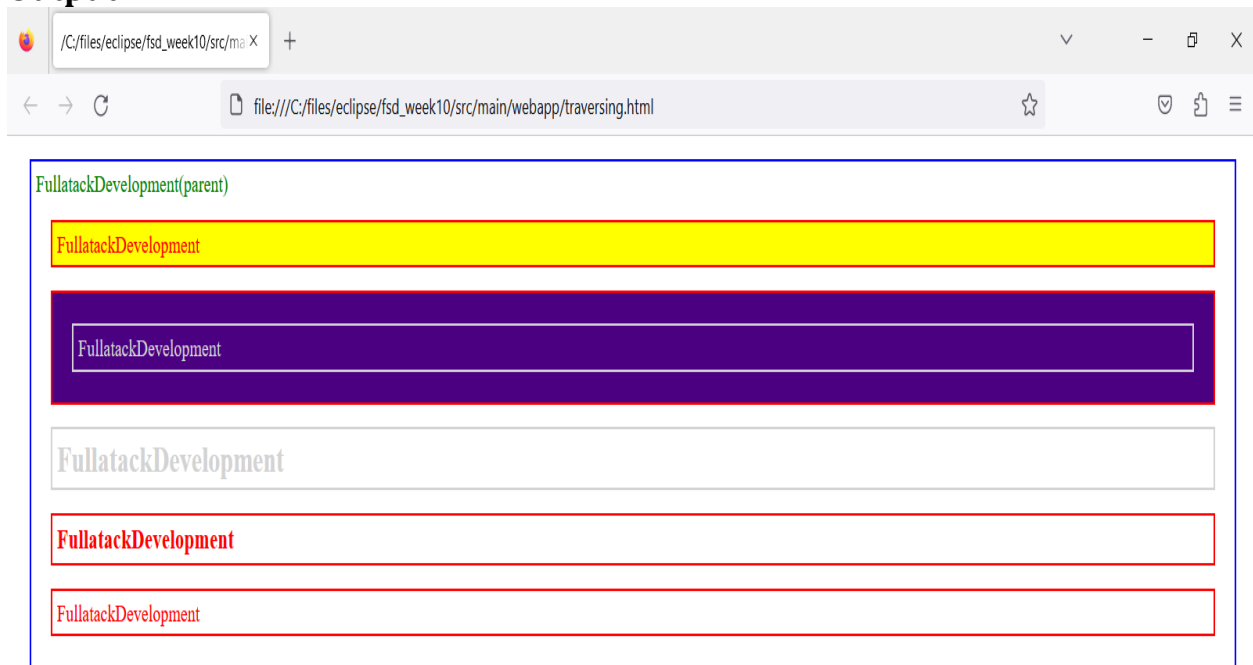
    <div>
        FullatackDevelopment(parent)
        <p>FullatackDevelopment</p>
        <p>
            <span>FullatackDevelopment</span>
        </p>
        <h2>FullatackDevelopment</h2>
        <h3>FullatackDevelopment</h3>
        <p>FullatackDevelopment</p>
    </div>

</body>

</html>

```

Output:



Example 2:

```
<!DOCTYPE html>

<html lang="en">

<head>
  <meta charset="utf-8">
  <style>
    p {
      color: grey;
      margin: 10px;
      padding: 10px;
    }

    form {
```



```
        margin: 10px;
        padding: 10px;
    }

    #article b {
        color: blue;
        font-weight: bold;
    }

    label {
        color: green;
        font-weight: bold;
    }
</style>
<script src=
"https://code.jquery.com/jquery-1.10.2.js">
</script>
</head>

<body>

    <p><em>Hello</em>GeeksforGeeks</p>
    <p id="article"><b>Article Info: </b></p>
    <form>
        <label>Article Title:</label>
        <input type="text"
            name="article"
```

```
        value="jQuery |  
        Traversing Example 2" />  
  
<br>  
  
<br>  
  
<label>Author:</label>  
  
<input type="text"  
        name="author"  
        value="Vignesh" />  
  
<br>  
  
<br>  
  
<label>Author's Email id:</label>  
  
<input type="text"  
        name="author"  
        value="vignesh@gmail.com" />  
  
<br>  
  
<br>  
  
<label>Website:</label>  
  
<input type="text"  
        name="url"  
        value="https://www.geeksforgeeks.org/" />  
  
<br>  
  
<br>  
  
</form>  
  
<script>  
    $("#article")  
        .append($("#input").map(function() {
```

```

        return $(this).val();
    })
    .get()
    .join(", ");
</script>
<script>
    $("p")
        .find("em")
        .end()
        .css("border", "2px red solid");
</script>

</body>

</html>

```

Output

Hello, GeeksforGeeks

Article Info: jQuery | Traversing Example 2, Vignesh, vignesh@gmail.com, <https://www.geeksforgeeks.org/>

Article Title: jQuery | Traversing Example

Author: Vignesh

Author's Email id: vignesh@gmail.com

Website: <https://www.geeksforgeeks.org/>

jQuery filter()

The **filter()** method returns the elements matching the specified criteria. If elements do not match the criteria, they are removed from the selection.

It can take a **selector** or a **function** as its arguments for filtering the set of matched elements. When using **selector**, the method filters the elements that don't match the given selector. If we use **function**, the method filters the elements that don't match the given function. Generally, this method is used to reduce the search for an element in the set of selected elements.

Syntax

Using *selector*

1. `$(selector).filter(selector)`

Using *function*

`$(selector).filter(function(index))`

The parameter values of this function are defined as follows.

selector: It is an optional attribute. It could be a **jQuery** object or a selector expression. We can also use the comma-separated list of expressions to apply multiple filters at once. It can be written as follows:

1. `filter("id1, #id2")`

function: It is also an optional parameter. This parameter specifies a function that runs for every element in the group. The element is kept if the function returns true. Otherwise, on returning false, the element is removed.

The **index** argument represents the position of the element in the set. It begins with the **0** position.

Let's see some examples to understand how to use the **filter()** method.

Example1

In this example, we are using the **selector** attribute of the **filter()** function. Here, the **filter()** function returns all paragraph elements with class name **para**. There are some div elements, paragraph elements and others. There are three paragraph elements out of four related to the class **para**.

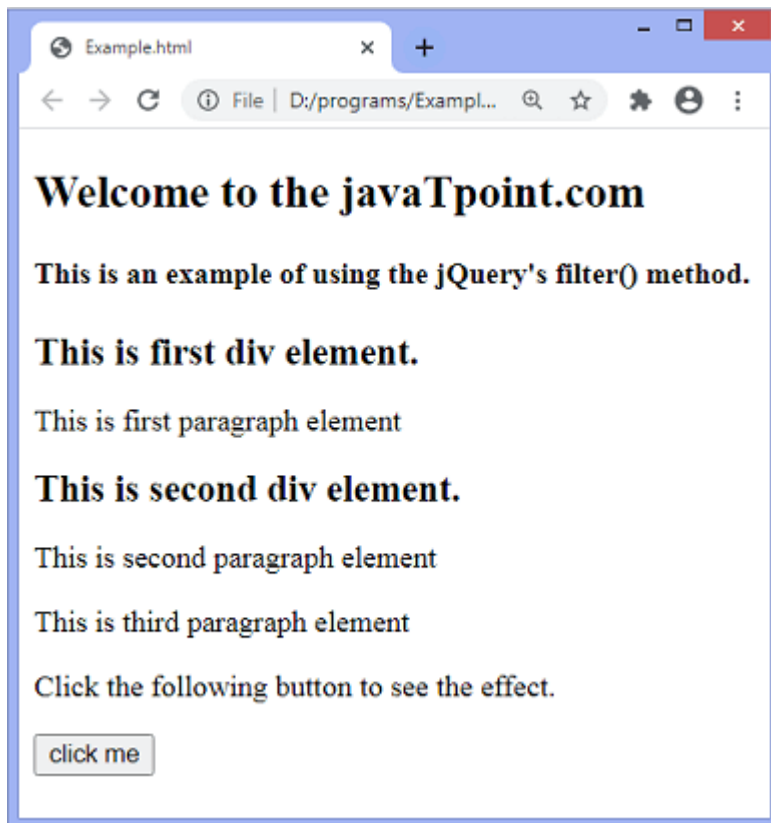
We have to click the given button to see the result.

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4. <style>
5. div{
6. font-size: 20px;
7. font-weight: bold;
8. }
9. </style>
10. <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"> </script>

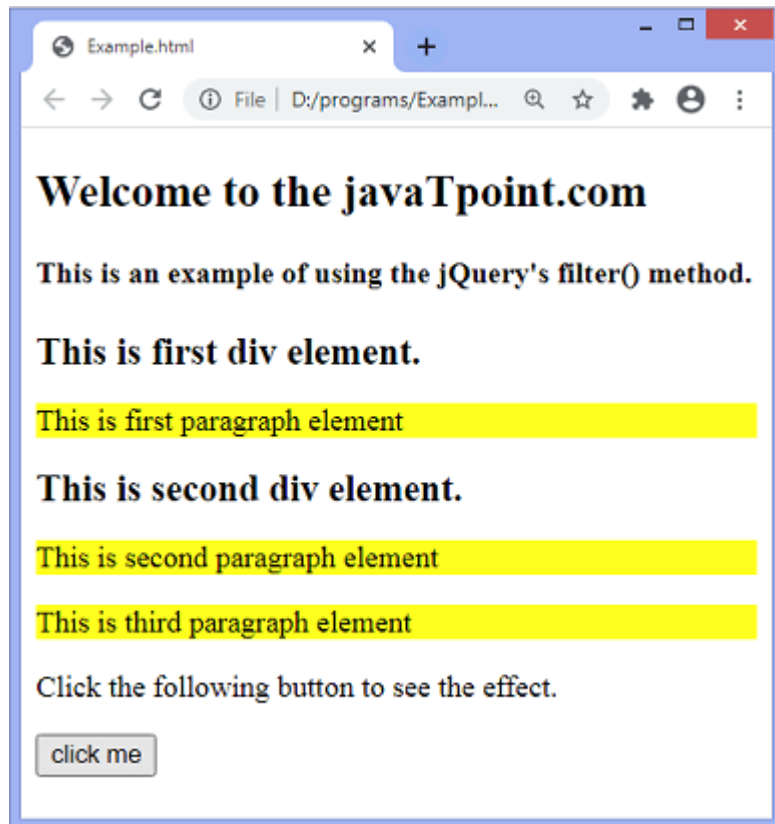
11. <script>
12. function fun(){
13. $(document).ready(function(){
14.  $("p").filter(".para").css({"background": "yellow"});
15. });
16. }
17. </script>
18. </head>
19.
20. <body>
21. <h2> Welcome to the javaTpoint.com </h2>
22. <h4> This is an example of using the jQuery's filter() method. </h4>
23. <div id = "div1"> This is first div element. </div>
24.
25.     <p class = "para"> This is first paragraph element </p>
26.     <div id = "div2"> This is second div element. </div>
```

27. `<p class = "para">` This is second paragraph element `</p>`
28. `<p class = "para">` This is third paragraph element `</p>`
29. `<p>` Click the following button to see the effect. `</p>`
30. `<button onclick = "fun()">` click me `</button>`
31. `</body>`
32. `</html>`

Output



After clicking the button, we can see that the function returns the paragraph elements related to the class name **para**.



Example2

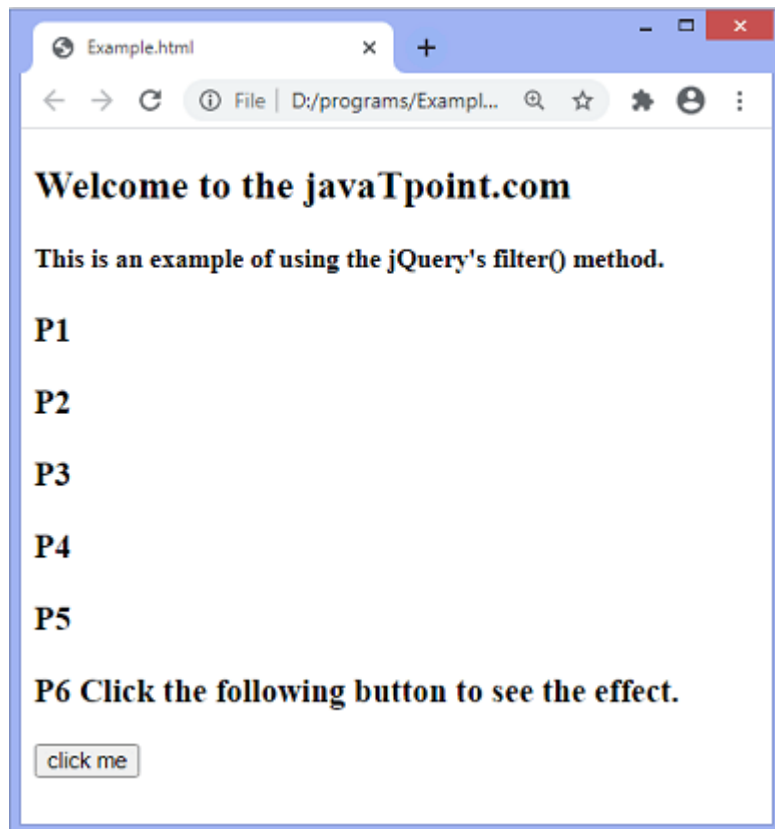
In this example we are using the ***function(index)*** parameter of the filter() function.. Here, the function highlights the paragraph elements with index position **1, 3, and 5**. Because the index begins with **0** so, it highlights the paragraph on even position.

1. `<!DOCTYPE html>`
2. `<html>`
3. `<head>`
4. `<style>`
5. `p{`
6. `font-size: 20px;`
7. `font-weight: bold;`
8. `}`
9. `</style>`
10. `<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"> </script>`
11. `<script>`

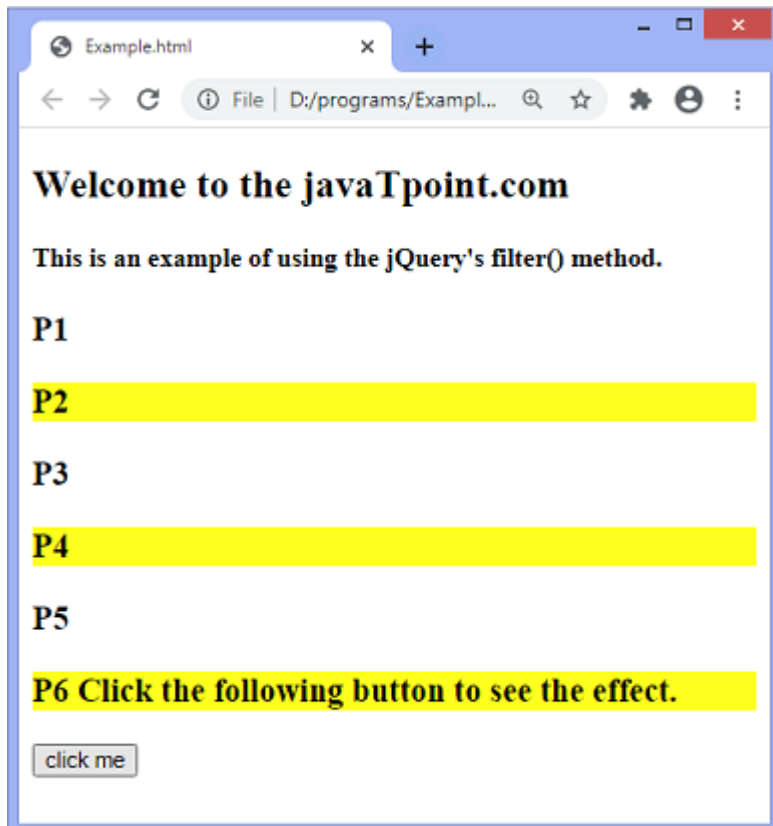
```
12. function fun(){
13. $(document).ready(function(){
14.   $("p").filter(function(index) {
15.     if(index == 1 || index == 3 || index == 5){
16.       return true;
17.     }
18.   }).css({"background": "yellow"});
19. });
20. }
21. </script>
22. </head>
23.
24. <body>
25. <h2> Welcome to the javaTpoint.com </h2>
26. <h4> This is an example of using the jQuery's filter() method. </h4>
27. <p class = "para1"> P1 </p>
28.     <p class = "para2"> P2 </p>
29.     <p class = "para3"> P3 </p>
30.     <p class = "para4"> P4 </p>
31.     <p class = "para5"> P5</p>
32. <p class = "para6"> P6 Click the following button to see the effect. </p>
33. <button onclick = "fun()"> click me </button>
34. </body>
35. </html>
```

Output

After the execution of the above code, the output is -



After clicking the given button, we can see that the function return the paragraph elements with index positions 1, 3, and 5. The position of the index starts at **0**.



jQuery DOM Manipulation[5 marks]

What is DOM?

The **Document Object Model** (DOM) is a notion of how to represent the structure of a "document". It is a platform and language-neutral interface that allows programs and scripts to **dynamically access** and update the content, structure, and style of a document.

DOM Structure

DOM provides a **structured representation** of the document and it defines a way that the structure can be accessed from programs so that they can change the document structure, style and content. The main object in the tree structure is the Document Object, which in turn

contains several other **child objects** . Also several properties of the document object include information about the current document in general. For e.g. document.title represent the title of the current page, defined by the HTML < title > tag.

```
<script type="text/javascript"> alert(document.title); </script>
```

HTML Document

The **Document Object Model** provides a uniform representation of the HTML document, and it achieves this by representing the entire HTML document as a **tree structure** . When a web page is loaded in the browser, it creates a Document Object Model of the web page. Each and every single element in the document will have a corresponding presence in the DOM.

Sample HTML Page:

```
<!DOCTYPE html>

<html>

  <head>

    <title>The DOM</title>

  </head>

  <body>

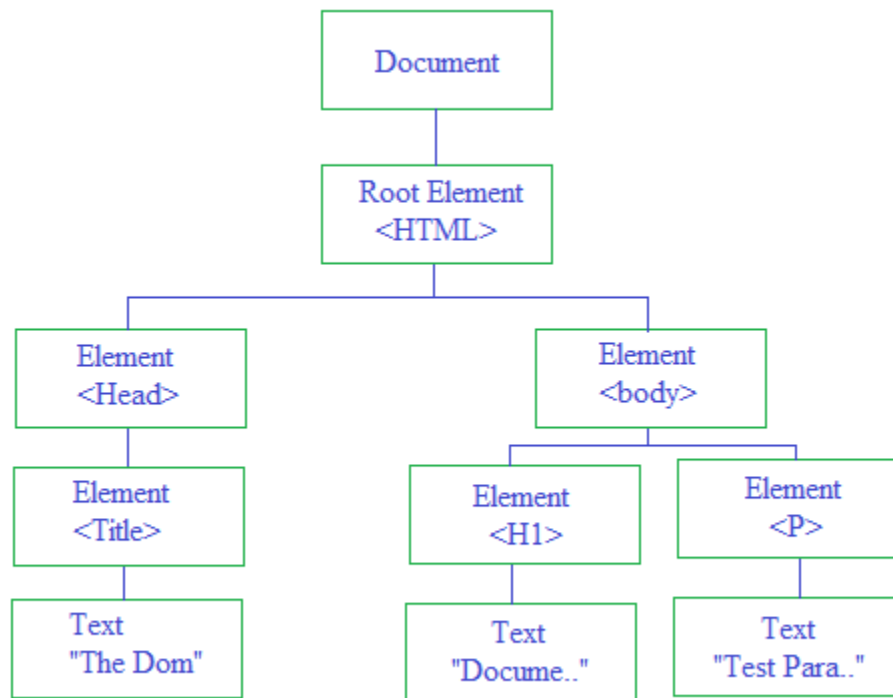
    <h1>Document Object Model</h1>

    <p id="pr">Test Paragraph</p>

  </body>

</html>
```

HTML Dom Structure



The **Docume**

nt Object Model currently consists of two parts, DOM Core and DOM HTML. The DOM Core represents the functionality used for XML documents, and also serves as the basis for DOM HTML. All DOM implementations must support the interfaces listed as **fundamental** in the Core specification; in addition, XML implementations must support the interfaces listed as "extended" in the Core specification. The Level 1 DOM HTML specification defines additional functionality needed for **HTML documents** .

DOM and JavaScript

JavaScript is an **interactive language** , and it is easier to understand new concepts by doing. The DOM is not a programming language, but without it, the **JavaScript language** wouldn't have any model or notion of web pages, HTML documents, XML documents, and their component

parts (e.g. elements). JavaScript is the client-side scripting language that connects to the DOM in an **internet browser** .

DOM and jQuery

jQuery is a library written in **Javascript** that is specialized for changing web documents on the fly. jQuery includes a whole range of methods that make it easy to **traverse the DOM** . The DOM (document object model) is the interface that enables JavaScript to interact with a web page's content. Whenever we use JavaScript (or jQuery) to inspect or manipulate elements on the web page, we're accessing the DOM. In the following chapter, we'll look at how to move up and down through the **DOM structure** and work with element hierarchy and parent/ child relationships to change the page structure on the fly using jQuery.

CHAPTER -II

Backend Programming with Node.js:

1. Installation and Simple Server
2. Express Setup and Routing
3. Template Engines
4. Node MongoDB Driver
5. Setup
6. Middleware & Routes
7. Creating the UI
8. Form Validation and User Register
9. Password Encryption
10. Login Functionality
11. Access Control & Logout

1. Installation and Simple Server

Introduction

In this , I am going to introduce NodeJS with Node Package Module (NPM), step-by-step basic implementation and explanation.

This article covers the following areas of NodeJS.

- Introduction of NodeJS
- Installation of NodeJS and NPM
- Node Package Module (NPM)
- Package.json
- Basic Example

NodeJS

[NodeJS](#) is an open-source, cross-platform runtime environment for developing server-side web applications. NodeJS also has an event-driven architecture capable of [asynchronous I/O](#).

NodeJS uses an event-driven, non-blocking I/O model that makes it lightweight and efficient.

Installation of NodeJS and NPM

Installation of NodeJS and NPM is straightforward using the installer package available at NodeJS official web site.

- Download the installer from [NodeJS WebSite](#).
- Run the installer.
- Follow the installer steps, agree the license agreement and click the next button.
- Restart your system/machine.

Now, test NodeJS by printing its version using the following command in Command Prompt:

```
1> node -v
```

bash

and test npm by printing its version using command

```
1> npm -v
```

bash

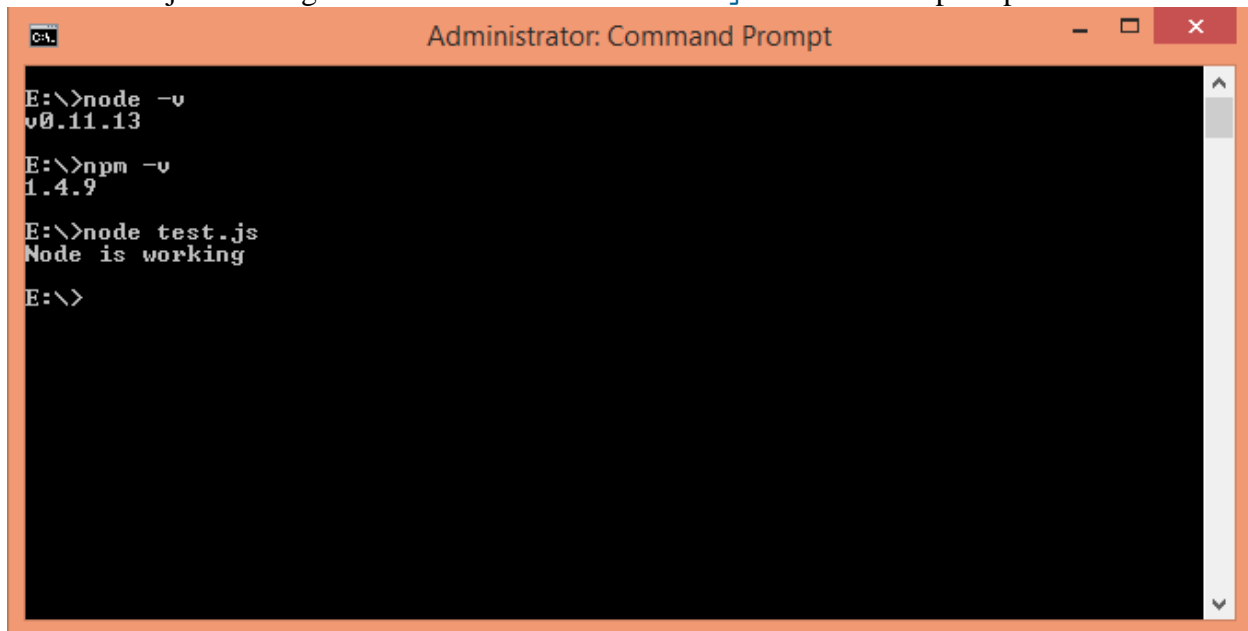
Simple way to test nodeJS work in your system is to create a javascript file which print a message.

Lets create test.js file

```
1/*test.js file*/  
2console.log("Node is working");
```

js

Run the test.js file using Node command > `node test.js` in command prompt.

A screenshot of a Windows Command Prompt window titled "Administrator: Command Prompt". The window has a black background with white text. The text shows the following commands and their outputs:
E:\>node -v
v0.11.13
E:\>npm -v
1.4.9
E:\>node test.js
Node is working
E:\>
The window has standard Windows window controls (minimize, maximize, close) in the top right corner.

You are done with installation.

Node Package Module

[NPM](#) is the package module which helps javascript developers load dependencies effectively. To load dependencies we just have to run a command in command prompt:

```
1> npm install
```

bash

This command is finding a json file named as `package.json` in root directory to install all dependencies defined in the file.

Package.json

[Package.json](#) looks like:

```
1{
2  "name": "ApplicationName",
3  "version": "0.0.1",
4  "description": "Application Description",
5  "main": "server.js",
6  "scripts": {
7    "start": "node server.js"
8  },
9  "repository": {
10    "type": "git",
11    "url": "https://github.com/npm/npm.git"
12  },
13  "dependencies": {
14    "express": "~3.0.1",
15    "sequelize": "latest",
16    "q": "latest",
17    "tedious": "latest",
18    "angular": "latest",
19    "angular-ui-router": "~0.2.11",
20    "path": "latest",
21    "dat-gui": "latest"
22  }
23}
```

json

The most important things in your package.json are name and version. Those are actually required, and your package won't install without them. The name and version together form an identifier that is assumed to be completely unique. Changes to the package should come along with changes to the version.

Repository

```
1{
2  "repository": {
3    "type": "git",
4    "url": "https://github.com/npm/npm.git"
```



```
5  }  
6 }
```

json

Specify the place where your code lives. Through this repository, developers can reach out and contribute to your application. If the git repository is not GitHub, then the [npm docs](#) command will be able to find you.

Scripts

```
1 {  
2   "scripts": {  
3     "start": "node server.js"  
4   }  
5 }
```

json

NPM provide many useful [Scripts](#) like [npm install](#), [npm start](#), [npm stop](#) etc. Some default script values are based on package contents.

```
1 "start": "node server.js"
```

json

If there is a server.js file in the root of your package, then npm will default the start command to node server.js.

Dependencies

```
1 {  
2   "dependencies": {  
3     "express": "~3.0.1",  
4     "sequelize": "latest",  
5     "q": "latest",  
6     "tedious": "latest",  
7     "angular": "latest",  
8     "angular-ui-router": "~0.2.11",  
9     "path": "latest",  
10    "dat-gui": "latest"  
11  }  
12 }
```

json

[Dependencies](#) are specified in a simple object that maps a package name to a version range. Version Name must be Version exactly.

If you want to install the latest version of a file, you just have to put `latest` in place of the version name.

Tilde(~) is used to tell "Approximately equivalent to version".

Basic Example

Create a server.js javascript file with following code

```
1/*server.js*/
2
3const http = require('http');
4
5const hostname = '127.0.0.1';
6const port = 3000;
7
8const server = http.createServer(function(req, res) {
9  res.statusCode = 200;
10  res.setHeader('Content-Type', 'text/plain');
11  res.end('Hello World\n');
12});
13
14server.listen(port, hostname, function() {
15  console.log('Server running at http://' + hostname + ':' +
port + '/');
16});
```

js

As we need `http` to create an http server we use `require('http')` and pass it to a variable named `http`

```
1var http = require('http');
```

js

We also need to defined hostname and port number, here we use `localhost` i.e `127.0.0.1` and `port` number `3000` and assign this to the variables `hostname` and `port`, respectively.

```
1var hostname = '127.0.0.1';
```

```
2var port = 3000;
```

js

Next we create the http server using the `createServer` method.

```
1var server = http.createServer(function(req, res){  
2  res.statusCode = 200;  
3  res.setHeader('Content-Type', 'text/plain');  
4  res.end('Hello World\n');  
5});
```

js

This created the server as well as a response having `statusCode: 200`, `Content-Type` header of plain text and ends with the string `Hello World`. This is the response that the server can send to browser.

the function has two parameters `req` and `res` which is the `request` from and `response` to the server, respectively.

In our example we are creating responses.

We created the server, now we have to assign it a hostname and port number.

```
1server.listen(port, hostname, function() {  
2  console.log('Server running at http://' + hostname + ':' +  
port + '/');  
3});
```

js

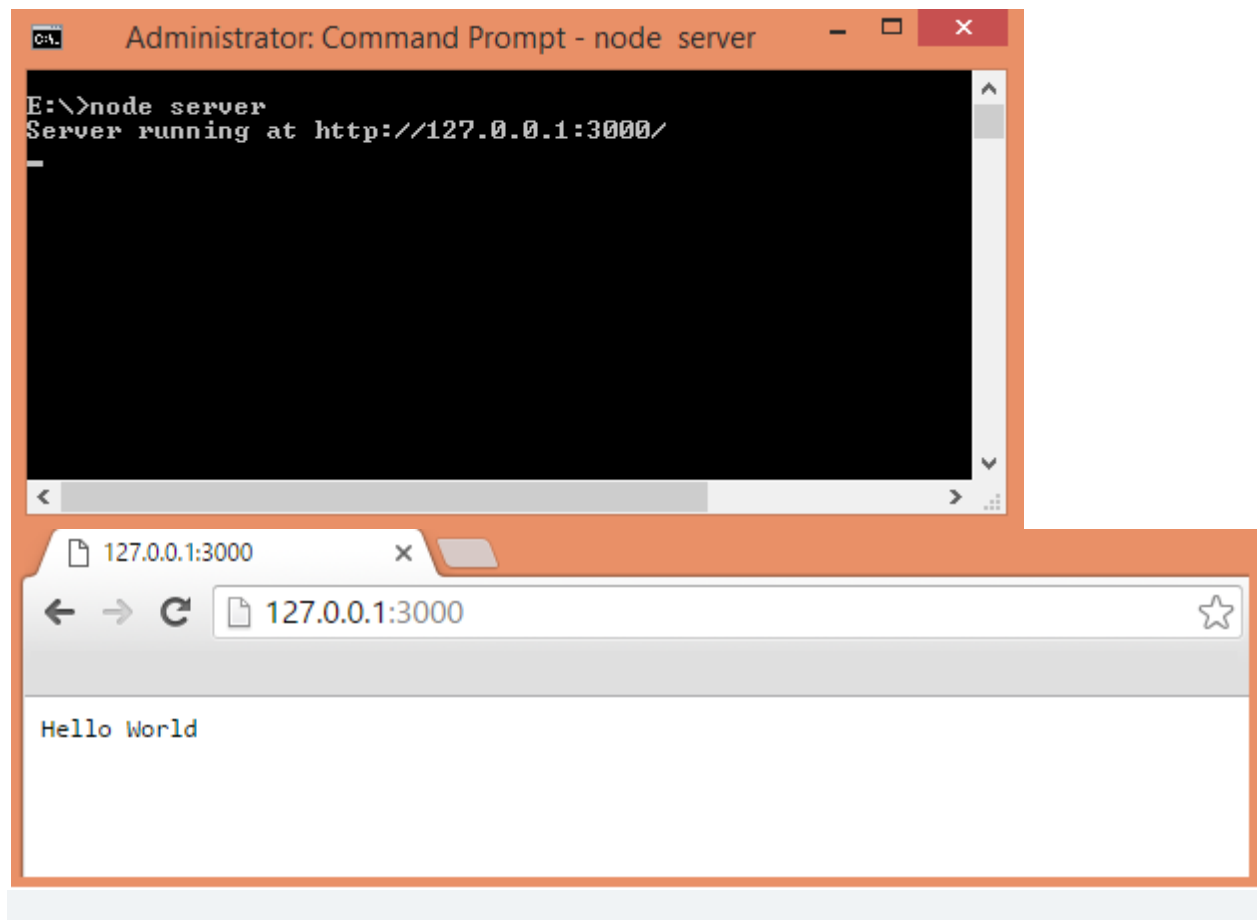
Here, the server listens to localhost on port 3000 and prints "Server running at <http://127.0.0.1:3000/>" in command prompt.

Now Run `server.js` file un node using command

```
1> node server
```

bash

Open a browser and enter url <http://127.0.0.1:3000/>. The browser will display Hello World message on the screen.



NodeJs Routing Expression:[10 marks]

Node.js & Express application: The implementation

This application will be implemented using Visual Studio Code (VSCode). VSCode is a new IDE from Microsoft used for developing and debugging modern application. This is a free IDE and can be installed on Windows OS, Linux and MAC OSX. You can download VSCode from [this link](#).

Step 1: Create a folder on your machine with the name **RoutingUsingExpress**. Open VSCode. Using File > Open Folder option, open **RoutingUsingExpress**. Using VSCode create a new JavaScript file of the name app.js.

Step 2: We need to install Node Intellisense and Express.js for the application. To install these, right click on app.js and select **Open in Command Prompt**. This will open the command prompt from where we will install the required packages.

Run the following command from the command prompt:

```
npm install tsd -g
```

This is a TSD package manager for searching and installing TypeScript definition files. We will use this package for installing Express. We also need TSD to use intellisense in VSCode.

```
tsd query node --action install
```

This will install TSD for Node so that we can use its intellisense.

```
Npm install express
```

```
tsd query express --action install
```

This will install Express using Node Package manager and install the TypeScript definition for Express so that we can use its intellisense in VSCode.

Step 3: In the project, add the Views folder with following three files

- Home.html
- About.html
- Contact.html

Step 4: In app.js, add the following code:

```
//1.
var express = require('express');
//2.
var path = require('path');
//3.
var app = express();
//4.
var objExpress = express.Router();

//5.
objExpress.get("/", function (request, response) {
    response.sendFile('home.html', { root: path.join(__dirname, './Views') });
});

//6.
objExpress.get('/about', function (request, response) {
    response.sendFile('about.html', { root: path.join(__dirname, './Views') });
});
//7.
objExpress.get('/contact', function (request, response) {
    response.sendFile('contact.html', { root: path.join(__dirname, './Views') });
});
//8.
objExpress.get('/data', function (req, resp) {

    var Emp = [
        { 'EmpNo': 101, 'EmpName': "A" },
        { 'EmpNo': 102, 'EmpName': "B" },
    ];

    resp.send(Emp);
});
```

```

});
//9.
objExpress.get('/data/:dname', function (req, resp) {
  var id = req.params.dname;
  if (id == 'IT') {
    var Emp = [
      { 'EmpNo': 101, 'EmpName': "A", 'DeptNae': 'IT' },
      { 'EmpNo': 102, 'EmpName': "B", 'DeptNae': 'IT' },
    ];

  } else {
    var Emp = [
      { 'EmpNo': 101, 'EmpName': "C", 'DeptNae': 'Systems' },
      { 'EmpNo': 102, 'EmpName': "D", 'DeptNae': 'Systems' },
    ];
  }
  resp.send(Emp);
});

//10.
app.use(objExpress);
//11.
app.listen(8090, function () {
  console.log('Listening on port 8090');
});
console.log('Application started....');

```

The above code has the following specifications:

Note: Line numbers below matches with the comments applied on the code.

1. Loads the **Express** module
2. Loads the **path** module so that the directory path for html files can be read.
3. Creates an instance of Express object. This will be used to define the port on which incoming http requests will be read.
4. This line calls the **Router()** function so that functions for route can be defined for http request types (get, post, put, delete, etc.). The instance of the Routing is defined as **objExpress**.
5. Use the **get()** function on the Express router object. This accepts two parameters - the first parameter is Route Expression URI and second parameter is the callback which responds with the home.html file in the **Views** directory.
6. Route expression for about.html
7. Route expression for contact.html.
8. The **get()** call which has the URL expression '/data'. This function is written to respond with an Employee array.
9. The **get()** call uses **/data:dname**, URI expression. Here **dname** is the parameter which will be passed with Http request in the URL. The **get()** function will execute based on the value of **dname**. In the current example, if **dname** is **IT** then the **if** statement will be executed, otherwise **else** statement will be executed.

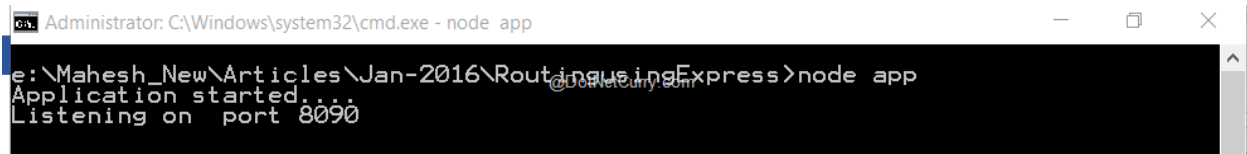
10. The **use()** function loads the router object in Express instance so that it will be used in http requests.

11. Express will start listening on 8090 port using the **listen()** function.

Step 5: To run the application, enter the following command in the command prompt which can be opened using Step 2.

Node app

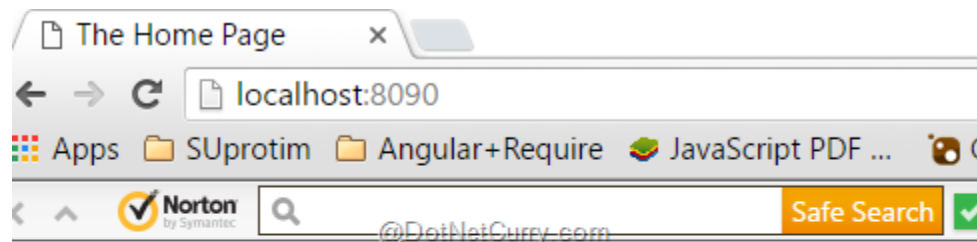
The following result will be displayed



```
Administrator: C:\Windows\system32\cmd.exe - node app
e:\Mahesh_New\Articles\Jan-2016\Routing using Express>node app
Application started...
Listening on port 8090
```

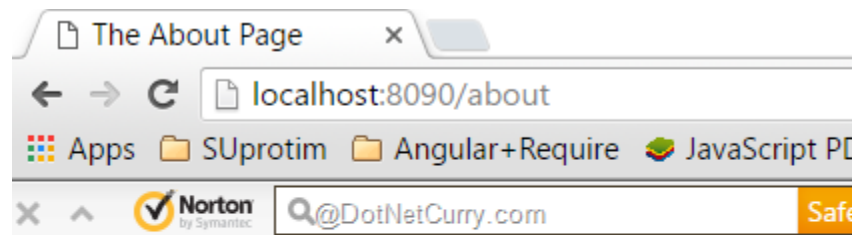
Open the Browser e.g. Chrome and enter the following URL

<http://localhost:8090>



The Home Page

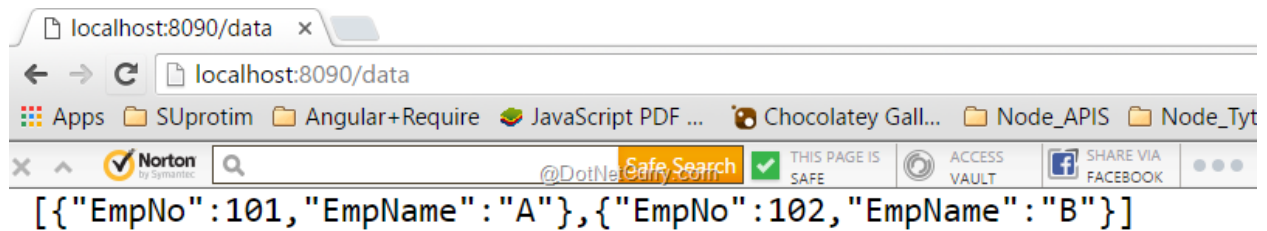
<http://localhost:8090/about>



The About Page

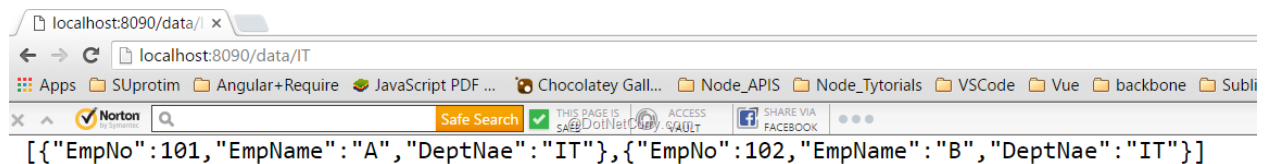
<http://localhost:8090/data>

This will show the Employees data as shown in the following image



<http://localhost:8090/data/IT>

This will show Employee data with DeptName value as IT:



Express web application in Node.js provides functions using which we can **implement routing** for responding with html views and data.

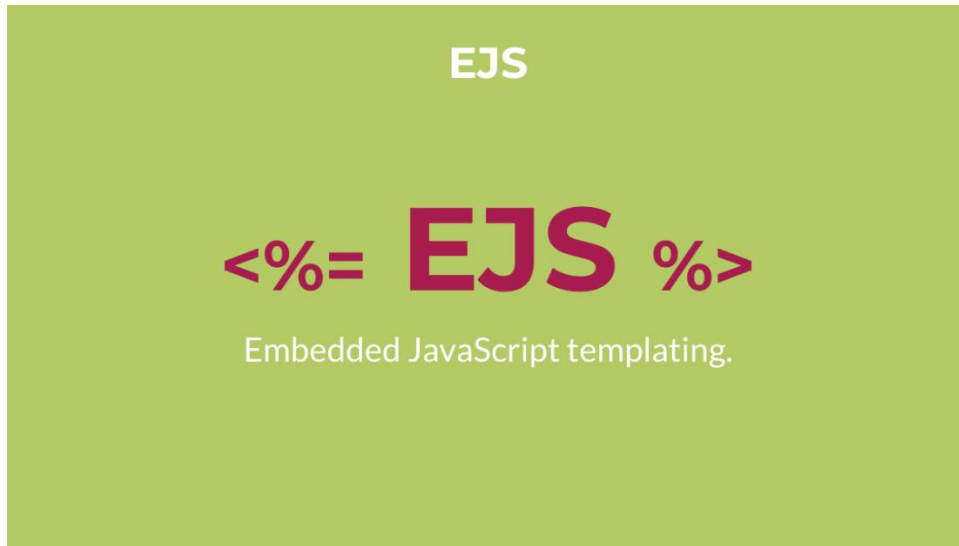
Template engines: [5 marks]

Template engines are used when you want to rapidly build web applications that are split into different components. Templates also enable fast rendering of the server-side data that needs to be passed to the application.

For example, you might want to have components such as body, navigation, footer, dashboard, etc.

Popular template engines

Template engines are mostly used for server-side applications that are run on only one server and are not built as APIs. The popular ones include Ejs, Jade, Pug, Mustache, HandlebarsJS, Jinja2, and Blade.



How template engines work

When you build a server-side application with a template engine, the template engine replaces the variables in a template file with actual values, and displays this value to the client. This makes it easier to quickly build our application.

Example with `expressJS` and `ejs` template engine

For a server-side application written with NodeJS runtime, you can use a template engine.

The following steps demonstrate how template engines work using `expressJS` and the `ejs` template engine. The example given below renders a user's data on the web page.

Step 1: Installing `express` and the `ejs` template engine

The following command installs the `ejs` template engine and the `express` framework:

```
npm install express ej
```

Step 2: Setting up the view engine

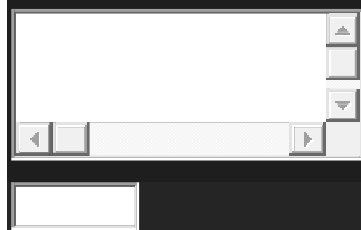
```
1
2
3
4
5
```

```

6
7
8

const express = require("express")
const app = express();
// Set the View Engine or Template Engine
app.set('view engine', 'ejs');
app.listen(3000)

```



In the code above, we created our express application. The application listens on port `3000`.

This line of code: `app.set('view engine', 'ejs')`, tells our express application that we want to use EJS as our template engine.

Step 3: Setting up the view folder

Create a folder called “view”. The view folder should contain our templates. One of these templates is `index.ejs`, which will generate our front page. The second template is `user.ejs`, which will be used to pass user data from the server-side to be immediately rendered on the webpage.

```

index.js
>view
  index.ejs
  user.ejs

```

Step 4: Setting up the routes

Let's create the `routes` for our homepage and user page.

Please note the `res.render()` method below. This is how you render a template in `expressJS`.

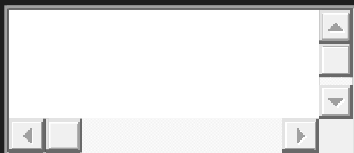
1

2

3
4
5
6
7
8
9
10
11
12

13

```
app.get('/', function (req, res) {  
  res.render("index");  
})  
  
app.get("/user", function(req,res){  
  const user = {  
    name: "Theodore Kelechukwu O.",  
    stack: "MERN",  
    email: "theodoreonyejiaku@gmail.com",  
    hobby: ["singing", "playing guitar", "reading", "philosoph"]  
  }  
  res.render("user", {user});  
})
```



As we have seen, the default route “/”, when accessed, displays or renders the `index.ejs` page. Meanwhile, the “/user” renders the `user.ejs` page.

We passed the `user` object to the render object so as to pass the `user` properties to the web page and render it.

Step 5: Templating our view files

Now that we have passed user data from server-side, we need to display it right away on our frontend or webpage.

`index.ejs`

`user.ejs`

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
```

16

```
<html>
  <head>
    <title>This is the title</title>
  </head>
```

```

<body>

  <h1>Welcome to User Details</h1>

  <p><b>Name:</b> <%= user.name %></p>

  <p><b>Email:</b> <%= user.email %></p>

  <p><b>Stack:</b> <%= user.stack %></p>

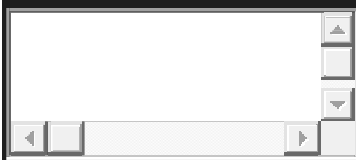
  <u><b>Hobbies</b></u>

  <% user.hubby.forEach(hubby =>{ %>
    <li><%= hobby %></li>
  <% })%>

</body>

</html>

```



Note the `<%= variable %>` pattern of displaying values. That is the way it is used in `ejs`. Also notice the `user.forEach()`; this is to show how powerful template engines can be.

I hope you now have a better understanding of template engines. Personally, I have used `Express Handlebars` and `EJS`, but I prefer `EJS`.

Node.js MongoDB Driver [5 Marks]

Node.js can be used in database applications.

One of the most popular NoSQL database is MongoDB.

MongoDB

To be able to experiment with the code examples, you will need access to a MongoDB database.

You can download a free MongoDB database at <https://www.mongodb.com>.

Or get started right away with a MongoDB cloud service at <https://www.mongodb.com/cloud/atlas>.

Install MongoDB Driver

Let us try to access a MongoDB database with Node.js.

To download and install the official MongoDB driver, open the Command Terminal and execute the following:

Download and install mongodb package:

```
C:\Users\Your Name>npm install mongodb
```

Now you have downloaded and installed a mongodb database driver.

Node.js can use this module to manipulate MongoDB databases:

```
var mongo = require('mongodb');
```

Password encryption in Node.js [5 Marks]

While submitting a form, there are some sensitive data (like passwords) that must not be visible to anyone, not even to the database admin. To avoid the sensitive data being visible to anyone, Node.js uses “bcryptjs”.

This module enables storing passwords as hashed passwords instead of plaintext.

Installation of bcryptjs module:

You can visit the link to Install bcryptjs module. You can install this package by using this command.

```
npm install bcryptjs
```

After installing bcryptjs module you can check your request version in the command prompt using the command.

```
npm version bcryptjs
```

After that, you can create a folder and add a file for example index.js, To run this file you need to run the following command.

```
node index.js
```

Example:

- Javascript

```
// Requiring module
const bcrypt = require('bcryptjs');

const password = 'pass123';
const hashedPassword;

// Encryption of the string password
bcrypt.genSalt(10, function (err, Salt) {

    // The bcrypt is used for encrypting password.
    bcrypt.hash(password, Salt, function (err, hash) {

        if (err) {
            return console.log('Cannot encrypt');
        }

        hashedPassword = hash;
        console.log(hash);

        bcrypt.compare(password, hashedPassword,
            async function (err, isMatch) {

                // Comparing the original password to
                // encrypted password
                if (isMatch) {
                    console.log('Encrypted password is: ', password);
                    console.log('Decrypted password is: ', hashedPassword);
                }

                if (!isMatch) {

                    // If password doesn't match the following
                    // message will be sent
                    console.log(hashedPassword + ' is not encryption of '
                        + password);
                }
            })
        })
    })
}
```

```
} )  
})
```

Step to run the application: Run the application using the following command:
`node index.js`

Output: We will see the following output on the console screen.

```
$2a$10$4DRBPlbjKO7WuL2ndpbisOheLfgVwDlngY7t18/ZZBFNcW3HdWFGm  
Encrypted password is: pass123 Decrypted password is:  
$2a$10$4DRBPlbjKO7WuL2ndpbisOheLfgVwDlngY7t18/ZZBFNcW3HdWFGm
```

Login form using Node.js and MongoDB[10 marks]

Follow these simple steps to learn how to create a login form using Node.js and MongoDB. Login form allows users to login to the website after they have created their account using the signup form.

We will be using the following technologies:

Express is a small framework that sits on top of Node.js's web server functionality to simplify its APIs and add helpful new features. It makes it easier to organize your application's functionality with middle ware and routing; it adds helpful utilities to Node.js's HTTP objects; it facilitates the rendering of dynamic HTTP objects.

mongoDB, the most popular NoSQL database, is an open-source document-oriented database. The term 'NoSQL' means 'non-relational'. It means that MongoDB isn't based on the table-like relational database structure but provides an altogether different mechanism for storage and retrieval of data. This format of storage is called BSON (similar to JSON format).

Passport is authentication middleware for Node. js. Extremely flexible and modular, Passport can be unobtrusively dropped in to any Express-based web application. A comprehensive set of strategies support authentication using a username and password, Facebook, Twitter, and more.

Steps to create project and install required module:

Step 1: Start the project using the following command in your project folder
`npm init`

Step 2: Install the required modules using following command
`npm install express`

`npm install ejs`


```
npm install mongoose
```

```
npm install body-parser
```

```
npm install express-session
```

```
npm install passport passport-local
```

```
npm install passport-local-mongoose
```

Step 3: Create two folders inside the project directory using the following command
`mkdir model`

```
mkdir views
```

Step 4: Create another file named app.js inside project directory
`touch app.js`

Step 5: Navigate inside model folder and create a file User.js which will contain our Schema
`cd model`

```
touch User.js
```

Step 6: Navigate inside views folder and create the following ejs files
`cd views`

```
touch home.ejs
```

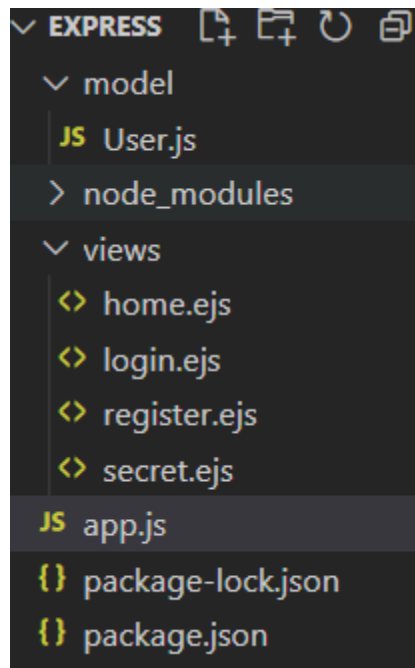
```
touch login.ejs
```

```
touch secret.ejs
```

```
touch register.ejs
```

Step 7: Run the following command to ensure all modules are loaded
`npm install`

After following the above mentioned steps your **project structure** should look like:



The **package.json** file should look like:

```
{  
  "name": "express",  
  "version": "1.0.0",  
  "main": "index.js",  
  "scripts": {  
    "test": "echo \"Error: no test specified\" && exit 1"  
  },  
  "keywords": [],  
  "author": "",  
  "license": "ISC",  
  "dependencies": {  
    "body": "^5.1.0",  
    "ejs": "^3.1.8",  
    "express": "^4.18.2",  
    "express-session": "^1.17.3",  
    "mongoose": "^6.9.1",  
  },  
}
```

```

    "parser": "^0.1.4",
    "passport": "^0.6.0",
    "passport-local": "^1.0.0",
    "passport-local-mongoose": "^7.1.2"
  },
  "description": ""
}

```

Add the following code in **App.js** and **User.js** file

```

// App.js

var express = require("express"),
    mongoose = require("mongoose"),
    passport = require("passport"),
    bodyParser = require("body-parser"),
    LocalStrategy = require("passport-local"),
    passportLocalMongoose =
      require("passport-local-mongoose")
const User = require("../model/User");
var app = express();

mongoose.connect("mongodb://localhost/27017");

app.set("view engine", "ejs");
app.use(bodyParser.urlencoded({ extended: true }));
app.use(require("express-session")({
  secret: "Rusty is a dog",
  resave: false,

```

```
        saveUninitialized: false
    }));

app.use(passport.initialize());
app.use(passport.session());

passport.use(new LocalStrategy(User.authenticate()));
passport.serializeUser(User.serializeUser());
passport.deserializeUser(User.deserializeUser());

//=====
// ROUTES
//=====

// Showing home page
app.get("/", function (req, res) {
    res.render("home");
});

// Showing secret page
app.get("/secret", isLoggedIn, function (req, res) {
    res.render("secret");
});

// Showing register form
app.get("/register", function (req, res) {
    res.render("register");
});
```

```
});
```

```
// Handling user signup
```

```
app.post("/register", async (req, res) => {
```

```
    const user = await User.create({
```

```
        username: req.body.username,
```

```
        password: req.body.password
```

```
    });
```

```
    return res.status(200).json(user);
```

```
});
```

```
//Showing login form
```

```
app.get("/login", function (req, res) {
```

```
    res.render("login");
```

```
});
```

```
//Handling user login
```

```
app.post("/login", async function(req, res){
```

```
    try {
```

```
        // check if the user exists
```

```
        const user = await User.findOne({ username: req.body.username });
```

```
        if (user) {
```

```
            //check if password matches
```

```
            const result = req.body.password === user.password;
```

```
            if (result) {
```

```
                res.render("secret");
```

```

        } else {
            res.status(400).json({ error: "password doesn't match" });
        }
    } else {
        res.status(400).json({ error: "User doesn't exist" });
    }
} catch (error) {
    res.status(400).json({ error });
}
});

```

//Handling user logout

```

app.get("/logout", function (req, res) {
    req.logout(function(err) {
        if (err) { return next(err); }
        res.redirect('/');
    });
});

```

```

function isLoggedIn(req, res, next) {
    if (req.isAuthenticated()) return next();
    res.redirect("/login");
}

```

```

var port = process.env.PORT || 3000;
app.listen(port, function () {
    console.log("Server Has Started!");
});

```

```
});
```

Add the following codes in the folder of **views**

- html
- HTML
- HTML
- HTML

```
// home.ejs  
  
<h1>This is home page</h1>  
  
<li><a href="/register">Sign up!!</a></li>  
<li><a href="/login">Login</a></li>  
<li><a href="/logout">Logout</a></li>
```

Steps to run the application.

Step 1: To run the above code you should first have the **mongoose** server running
If you do not have the mongoose folder setup follow [this](#) article
After setting up mongoDB start the server using following command

```
mongod
```

Step 2: Type the following command in terminal of your project directory
`node app.js`

Step 3: Open your web browser and type the following address in the URL bar
`http://localhost:3000/`

Access control with NodeJS [10 marks]

System security is one of the key considerations when building software, and there are various mechanisms used in ensuring a software system is secure. The common ones are role-based access control (RBAC) and attribute-based access control (ABAC).

AccessControl, a Node.js module, can be used to implement these two access control mechanisms. Before diving into how AccessControl works, let's briefly explain these two mechanisms and how they work.

Role-based access control (RBAC)

Role-based access control, also known as role-based security, is a mechanism that restricts system access to users using their roles and privileges and permissions.

Within an application, roles are created for various user types (e.g., writer or reader). The permission to perform certain actions or access application resources are assigned to specific roles. For instance, in a writing application, a writer can be granted the permission to create, update, read, and delete a post, whereas a reader can be restricted to only being able to read a post.

When using RBAC, there are three guiding rules:

1. Role assignment: A subject (i.e., a user) can exercise a permission only if the subject has selected or been assigned a role.
2. Role authorization: A subject's active role must be authorized for the subject. With rule 1 above, this rule ensures that users can take on only roles for which they are authorized.
3. Permission authorization: A subject can exercise a permission only if the permission is authorized for the subject's active role. With rules 1 and 2, this rule ensures that users can exercise only permissions for which they are authorized.

A user can have multiple roles and a role can have multiple permissions. RBAC also supports role hierarchy where a high-level role inherits the permissions of its sub roles. Additional constraints may be applied to place restrictive rule on the potential inheritance of permissions from another role. Some examples of constraints are:

- A role cannot inherit itself
- Cross-inheritance is not allowed (if the writer inherits from the reader, the reader cannot inherit from the writer)
- A role cannot pre-extend a non-existing role

Attribute-based access control (ABAC)

[Attribute-based access control](#), also known as policy-based access control for IAM, defines an access control paradigm whereby access rights are granted to users through the use of policies which combine attributes together. The policies can use any type of attributes, such as user attributes, resource attributes, object, and environment attributes.

ABAC can be used to complement RBAC in that in addition to roles and permissions, a policy can be used to define what attribute is allowed or not allowed.

Node.js RBAC-ABAC using AccessControl

[AccessControl](#), a Node.js module, merges the best features of RBAC and ABAC. It implements RBAC basics and also focuses on resource and action attributes. For a full list of the module's features, view the documentation.

Installation

With [npm](#): `npm i accesscontrol --save`.

With [Yarn](#): `yarn add accesscontrol`

Roles

Roles serve as containers for permissions. They are assigned to users depending on their responsibility. You can create and define roles simply by calling `.grant(<role>)` or `.deny(<role>)` methods on an [AccessControl](#) instance.

```
import { AccessControl } from 'accesscontrol';
```

```
const ac = new AccessControl();  
ac.grant('reader');
```

Roles can extend other roles. You can extend a role by calling `.extend` on an existing role.

```
ac.grant('reader').extend('writer');
```

Actions and action-attributes

Actions and action-attributes represent what can be performed on resources by role(s). They are a finite fixed list based on classic CRUD. There are two action-attributes which define the possession of the resource by a role: own and any.

For example, an [editor](#) role can [create](#), [read](#), [update](#) or [delete](#) (CRUD) any [post](#) resource. But a [writer](#) role might only read or update its own [post](#) resource.

You can define an action and possession on a resource using: [createOwn](#), [readOwn](#), [updateOwn](#), [deleteOwn](#), [createAny](#), [readAny](#), [updateAny](#), and [deleteAny](#) methods.

```
const ac = new AccessControl();
ac.grant('reader')
  .readAny('post')
  .grant('writer')
    .createOwn('post')
    .deleteOwn('post')
    .readAny('post')
  .grant('editor')
    .extend('writer')
    .updateAny('post')
    .deleteAny('post');
```

Resources and resource-attributes

These represent system elements that we want to protect, such as `post`. Multiple roles can have access to a specific resource but may not have equal access to all attributes of the resource. You can use Glob notation to define allowed or denied attributes.

For example, we have a `post` resource that has the following attributes: `id`, `title`, and `description`. All attributes of any `post` resource can be read by an `editor` role:

```
ac.grant('editor').readAny('post', ['*']);
```

But the `id` attribute should not be read by a `reader` role.

```
ac.grant('reader').readAny('post', ['!id']);
```

Checking permissions and filtering attributes

The permission granted is determined using a combination of role, action, and resource. You can add `.can(<role>).<action>(<resource>)` on an `AccessControl` instance to check for granted permissions for a specific resource and action.

```
const permission = ac.can('reader').readAny('post');
permission.granted;
permission.attributes;
permission.filter(data);
```

Defining all grants at once

You can pass the grants directly to the `AccessControl` constructor. It accepts either an `Object`:

```
let grantObjects = {
  reader: {
    post: {
      'read:any': ['*', '!id']
    }
  },
  writer: {
    post: {
      'create:own': ['*'],

```

```

        'read:any': ['*'],
        'update:own': ['*'],
        'delete:own': ['*']
      },
    },
    editor: {
      post: {
        'create:any': ['*'],
        'read:any': ['*'],
        'update:any': ['*'],
        'delete:any': ['*']
      }
    }
  }
}

```

```
const ac = new AccessControl(grantsObject);
```

Or an array:

```

let grantArray = [
  { role: 'reader', resource: 'post', action: 'read:any', attributes:
    '*', !id' },
  { role: 'writer', resource: 'post', action: 'read:any', attributes:
    '*' },
  { role: 'writer', resource: 'post', action: 'create:own',
    attributes: '*' },
  { role: 'writer', resource: 'post', action: 'update:own',
    attributes: '*' },
  { role: 'writer', resource: 'post', action: 'delete:own',
    attributes: '*' },
  { role: 'editor', resource: 'post', action: 'read:any', attributes:
    '*' },
  { role: 'editor', resource: 'post', action: 'create:any',
    attributes: '*' },
  { role: 'editor', resource: 'post', action: 'update:any',
    attributes: '*' },
  { role: 'editor', resource: 'post', action: 'delete:any',
    attributes: '*' },
]

```

```
const ac = new AccessControl(grantArray);
```

Example with Express.js

```
const ac = new AccessControl(grants);
```

```

router.get('/posts/:title', function (req, res, next) {
  const permission = ac.can(req.user.role).readAny('post');

```

```
if (permission.granted) {  
  Video.find(req.params.title, function (err, data) {  
    if (err || !data) return res.status(404).end();  
    res.json(permission.filter(data));  
  });  
} else {  
  res.status(403).end();  
}  
});
```

We have shown how we can use `AccessControl` for authorization in a server-side application. We can also use it to authorize routes and UI elements in a client-side application, which can be done by using the same grant object for both the server and the client. `AccessControl` is one only library for implementing access control in Node.js. Node-casbin and CASL are also Node.js libraries for implementing access control.