

JavaScript on the Client Side

Outline

1. DOM Manipulation using JavaScript
2. Consume Web API using Fetch API
3. jQuery

DOM Manipulation using JavaScript

What Can JavaScript Do?

- **Server Side Web applications**
 - Write server-side application logic (using Node.js)
- **Client Side Dynamic Behavior**
 - **React to user input** i.e., handle client side events such as button clicked event. e.g., Changing an image on moving mouse over it
 - **Updating the page**
 - Add/update page content: **Manipulate the Document Object Model** (DOM) of the page: read, modify, add, delete HTML elements
 - Change how things look: CSS updates
 - **Validate form input** values before being submitted to the server
 - **Perform computations**, sorting and animation
 - **Perform asynchronous server calls** (AJAX) to load new page content or submit data to the server without reloading the page

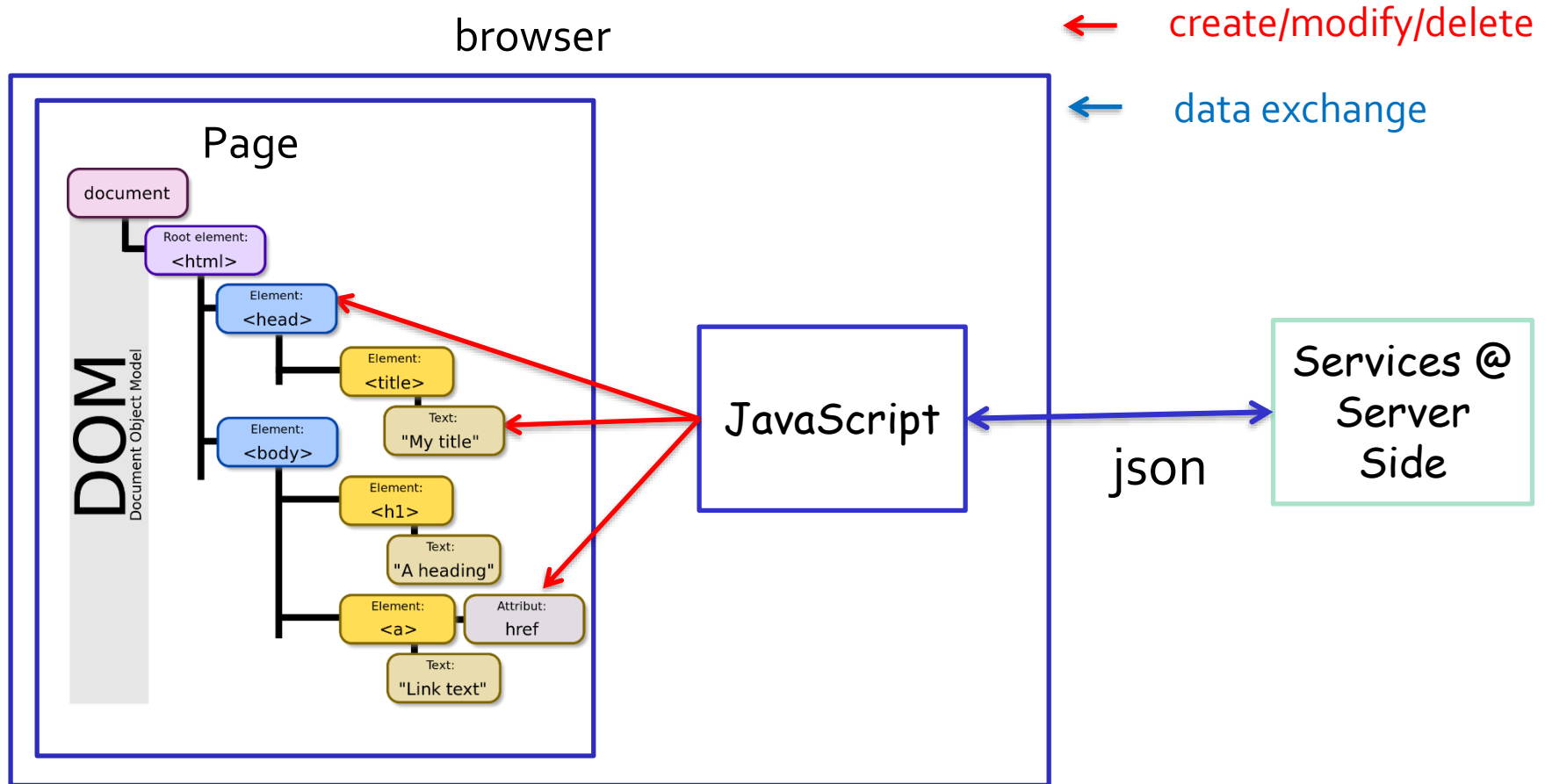
Where to place JavaScript Code?

- The JavaScript code can be placed in:
 - `<script>` tag in the head
 - `<script>` tag in the body - not recommended
 - In an external file and add [a reference to](#) it in the HTML file. This is the recommended way
 - Reference via `<script>` tag in the **head** or at the end of the **body**
 - Files usually have `.js` extension

```
<script src="script.js">  
    <!-- code placed here will not be executed! -->  
</script>
```

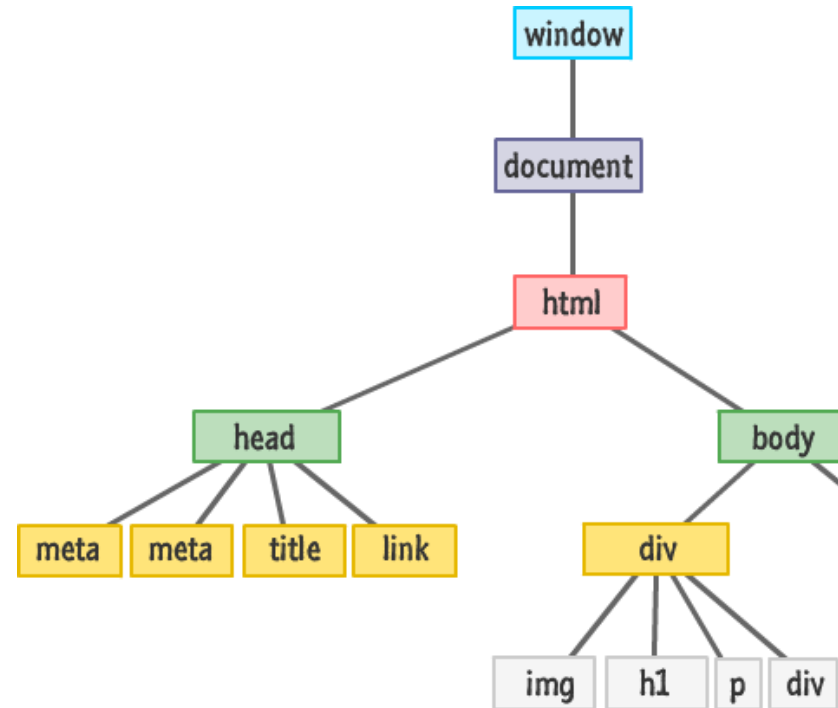
- The `.js` files get cached by the browser

Role of JavaScript on the Client Side



Document Object Model (DOM)

- DOM API consist of objects and methods to interact with the HTML page
 - **Select** page elements
 - **Add, update** or **remove** page elements
 - **Apply styles** dynamically
 - **Listen** to and **handle** events



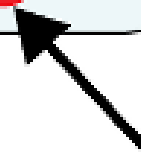
Example DOM Element

HTML

```
<p>  
  Look at this octopus:  
    
  Cute, huh?  
</p>
```



DOM Element Object	
Property	Value
tagName	"IMG"
<u>src</u>	"octopus.jpg"
alt	"an octopus"
id	"icon01"



JavaScript

```
var icon = document.getElementById("icon01");  
icon.src = "kitty.gif";
```


Selecting HTML Elements

- Elements must be **selected first** before changing them or listening to their events
 - **querySelector()** returns the first element that matches a specified *CSS selector* in the document
 - **querySelectorAll()** returns all elements in the document that matches a specified CSS selector

Example CSS selectors:

1. By tag name: `document.querySelector("p")`
 2. By id : `document.querySelector("#id")`
 3. By class: `document.querySelector(".classname")`
 4. By attribute: `document.querySelector("img[src='cat.png']")`
 - Return the first image whose src attribute is set to `cat.png`
- Examples
 - https://www.w3schools.com/jsref/met_document_queryselector.asp
 - https://www.w3schools.com/jsref/met_document_queryselectorall.asp

Selecting Elements – old way!

- Access elements via their ID attribute

```
let element = document.getElementById("some-id")
```

- Via the **name** attribute

```
let elArray = document.getElementsByName("some-name")
```

- Via tag name

```
let imgTags = document.getElementsByTagName("img")
```

- Returns array of `` elements

DOM Manipulation

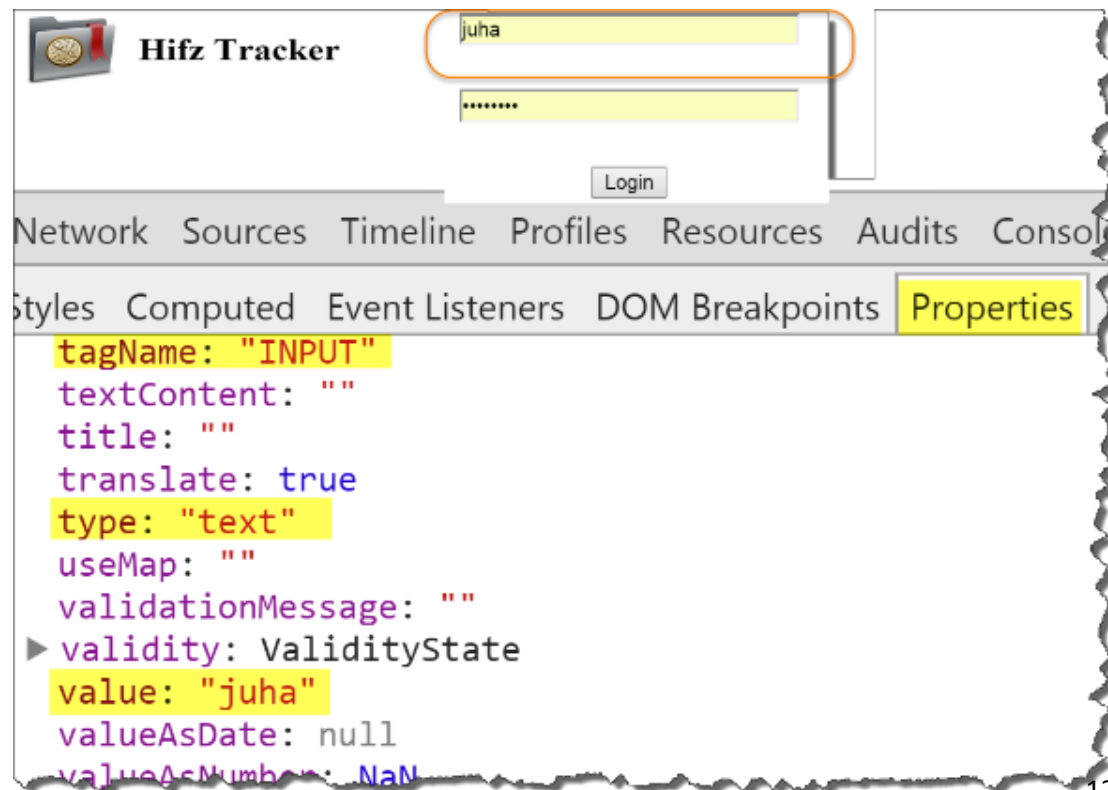
- Once we **select** an element, we can read and write its attributes

```
function change(state) {  
  let lampImg = document.querySelector("#lamp")  
  lampImg.src = `lamp_${state}.png`  
  let statusDiv =  
    document.querySelector("#statusDiv")  
  statusDiv.innerHTML = `The lamp is ${state}`  
}  
...  
 />
```

Common Element Properties

- **Value** - get/set value of input elements
- **innerHTML** - get/set the HTML content of an element
- **className** - the **class** attribute of an element

User Chrome
Dev Tool to see
the Properties of
Page element



Events Handling

- JavaScript can register event handlers
 - Events are fired by the Browser and are sent to the specified JavaScript **event handler** function
 - Can be set with HTML attributes:

```

```

- Can be set through the DOM:



```
let img = document.querySelector("#myImage")  
img.addEventListener('click', imageClicked)
```

Ask to be notified of clicks on element **#myImage**

Event Handler Example

```
<script>  
document.querySelector("#btnDate").  
  addEventListener("click", displayDate)  
  
function displayDate() {  
  document.querySelector("#date").innerHTML =  
    Date()  
}  
</script>
```

Try it @

http://www.w3schools.com/js/tryit.asp?filename=tryjs_addeventlistener_displaydate

Common DOM Events

- Mouse events:
 - `onclick`, `onmousedown`, `onmouseup`
 - `onmouseover`, `onmouseout`, `onmousemove`
- Key events:
 - `onkeypress`, `onkeydown`, `onkeyup`
 - Only for input fields
- Interface events:
 - `onblur`, `onfocus`, `onscroll`
- Form events
 - `onsubmit`: allows you to cancel a form submission if some input fields are invalid

DOMContentLoaded

- DOMContentLoaded is fired when the DOM tree is built, but external resources like images and stylesheets may be not yet loaded
 - Best event for adding event listeners to page elements

```
//When the document is loaded in the browser then listen to studentsDD on change event  
document.addEventListener("DOMContentLoaded", () => {  
    console.log("js-DOM fully loaded and parsed");  
    document.querySelector('#studentsDD').addEventListener("change", onStudentChange)  
})
```


Commonly used methods

- Add Element

e.g., add div element and assign it *alert-success* css class

```
let newDiv = document.createElement('div')
newDiv.innerText = 'Div added by script'
newDiv.classList.add('alert-success')
document.body.appendChild(newDiv)
```

- DOM Traversal

```
let parent = document.querySelector('#about').parentNode
let children = document.querySelector('#about').children
```

- Hide & Show

```
document.querySelector('myDiv').style.display = 'none';
document.querySelector('myDiv').style.display = '';
```

Consume Web API using Fetch API





- AJAX is acronym of Asynchronous JavaScript and XML
 - AJAX is used for **asynchronously** loading (in the background) of dynamic Web content and data from the Web server into a HTML page
 - Allows dynamically adding elements into the DOM
- Two styles of AJAX
 - Partial page rendering
 - Load an HTML fragment and display it in a **<div>**
 - Call REST service then use the received JSON object to update the page at the client-side using JavaScript / jQuery

Call REST Service using Fetch API

- Fetch content from the server

```
async function getStudent(studentId) {  
    let url = `/api/students/${studentId}`  
    let response = await fetch(url)  
    return await response.json()  
}
```

- **.json()** method is used to get the response body as a JSON object

Posting a request to the server using Fetch API

- Fetch could be used to post a request to the server

```
let email = document.querySelector( "#email" ).value,  
    password = document.querySelector("#password").value
```

```
fetch( "/login", {  
    method: "post",  
    headers: { "Accept": "application/json",  
               "Content-Type": "application/json" },  
    body: JSON.stringify({  
        email,  
        password  
    })  
})
```

//headers parameter is optional



<https://jquery.com/>

No longer popular. **Use plain JavaScript instead**

jQuery

- jQuery is a fast, small and feature-rich JavaScript library that works across a multitude of browsers
- Simplifies HTML document traversing, event handling and animation
- To include jQuery in your website, all you need is a script tag with its *src* pointed to the hosted location

```
<script  
src="https://code.jquery.com/jquery-3.1.1.slim.min.js">  
</script>
```

jQuery Syntax

- You can use the **\$()** function to **select** HTML elements and perform some **action** on the element(s)
- Basic syntax is: **\$(*selector*).*action*()**
 - A **\$** sign to access jQuery
 - A (***selector***) to find HTML elements
 - A ***action*()** to be performed on the element(s)

jQuery Selectors

jQuery supports CSS selectors:

1. By tag name: `$("p")`
2. By id : `$("#id")`
3. By class: `$(".classname")`
4. By attribute: `$("a[href]")`
5. ...

DOM method	jQuery equivalent
<code>querySelector("#id")</code>	<code>\$("#id")</code>
<code>querySelector("tag")</code>	<code>\$("tag")</code>
<code>querySelector(".classname")</code>	<code>\$(".classname")</code>

jQuery Syntax

- Examples:
 - `$("p").hide()` - hides all `<p>` elements
 - `$(".test").hide()` - hides all elements with `class="test"`
 - `$("#test").hide()` - hides the element with `id="test"`
 - `$('div').css('background', 'blue')` - Make all DIVs blue

.ready() event

- jQuery provides a *ready* event that is fired when the document is ready to be manipulated
- You'll put most of your code in this method

```
$(document).ready( () => {  
    // Your code here e.g.,  
    alert("Ok document is ready...")  
})
```

Creating Elements

- Creating new elements is also easy

```
let divElement = $('<div>')  
let paragraph = $('<p>Some text</p>')
```

- Creating complex node

```
$("<p>", {  
  "id": "myParagraph",  
  "class": "special",  
  "text": "My paragraph is awesome!"  
})
```

Adding Elements

- Adding elements can be done on the fly
 - `jQuery.appendTo()` / `jQuery.prependTo()`
 - `jQuery.append()` / `jQuery.prepend()`

```
<h2>QU Greetings</h2>
<div id="wrapper">
  <div>Hello, student</div>
  <div>Goodbye, student</div>
</div>
```

```
<h2>QU Greetings</h2>
<div id="wrapper">
  <div>Hello, student<p>Test</p></div>
  <div>Goodbye, student<p>Test</p></div>
</div>
```

```
$('#wrapper div').append('<p>Test</p>')
```

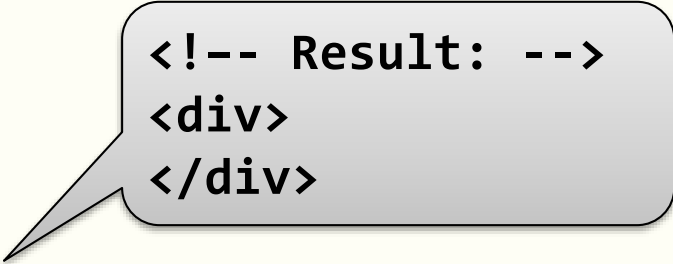
```
<div>First</div>
<h2>QU Greetings</h2>
<div id="wrapper">
  <div>Hello, student<div>
  <div>Goodbye, student<div>
</div>
```

```
$('#<div>First</div>').prependTo('body')
```

Removing Elements

- You can also easily remove elements from the DOM

```
<div>  
  <p>Red</p>  
  <p>Green</p>  
</div>
```



```
<!-- Result: -->  
<div>  
</div>
```

```
<script>  
  $('p').remove() // Remove all paragraphs  
</script>
```

jQuery Events

- jQuery has a convenient way for attaching and detaching events Using methods **on()** and **off()**

```
$('#button').on('click', onButtonClick)
```

```
function onButtonClick() {  
    $(this).hide()  
    // "this" is the event source (the button clicked)  
}
```

Looping over the DOM

- Using the DOM

```
let elems = document.querySelectorAll("li")
for ( let e of elems )
    // do stuff with e
}
```

- Using jQuery

```
let elems = $("li")
for ( let e of elems )
    // do stuff with e
}
```


jQuery css method parameters

- **Getter:**

```
$("#myid").css(propertyName)
```

- **Setter:**

```
$("#myid").css(propertyName, value)
```

- **Multi-setter:**

```
$("#myid").css({  
    'propertyName1': value1,  
    'propertyName2': value2,  
    ...  
})
```

More node manipulation with jQuery

jQuery method	functionality
<u>.hide()</u>	Hides an element
<u>.show()</u>	Shows an element
<u>.empty()</u>	remove the content of an element, equivalent to <code>innerHTML = ""</code>
<u>.html()</u>	get/set the content of an element in HTML format
<u>.text()</u>	get/set the content of an element as plain text
<u>.val()</u>	get/set the value of a form input, select, textarea, ...

Summary

- jQuery – the most popular client-side JS library
- Select DOM elements with jQuery
 - `$([selector])`
- DOM Traversal:
 - `$([selector]).next() / parent()`
- Altering the DOM:
 - `$([selector]).html(...) / append(...)`
- jQuery Events
 - `$([selector]).on([event], [callback])`

Resources

- W3C School:

<http://www.w3schools.com/jquery/>

- Code School:

<http://www.codeschool.com/courses/jquery-air-first-flight>