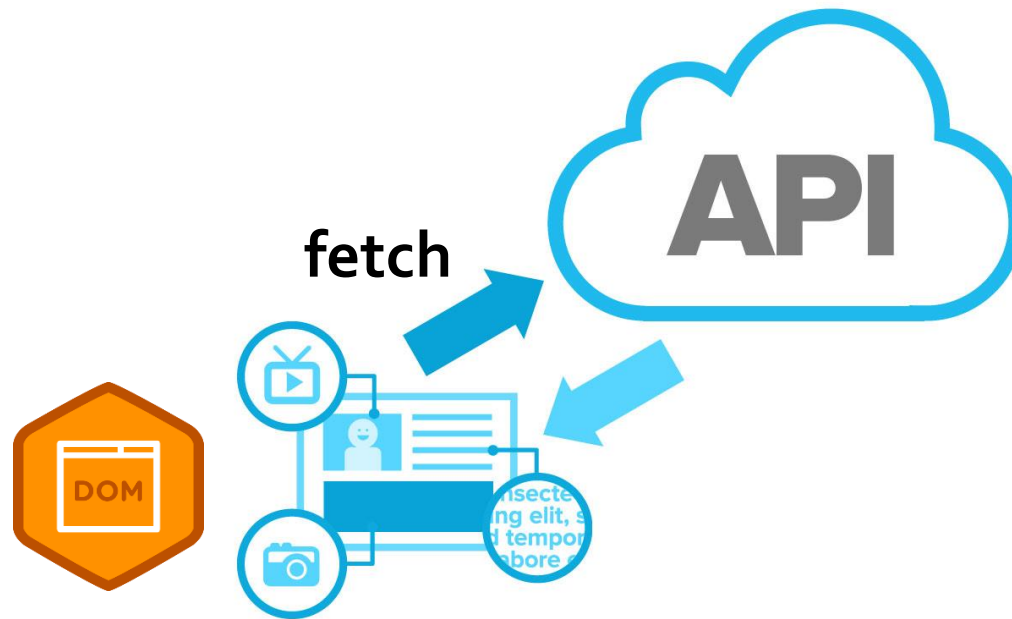


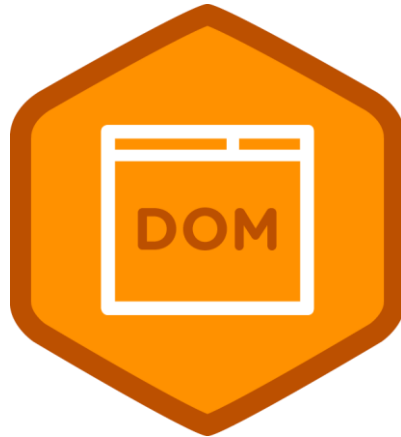
# JavaScript on the Client Side



# Outline

1. DOM Manipulation using JavaScript
2. Consume Web API using Fetch
3. Template String to generate UI

# DOM Manipulation using JavaScript



# What Can JavaScript Do?

- **Server Side Web applications**
  - Write server-side application logic and Web API (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 Web API calls** (AJAX) to get or submit JSON data to the server without reloading the page

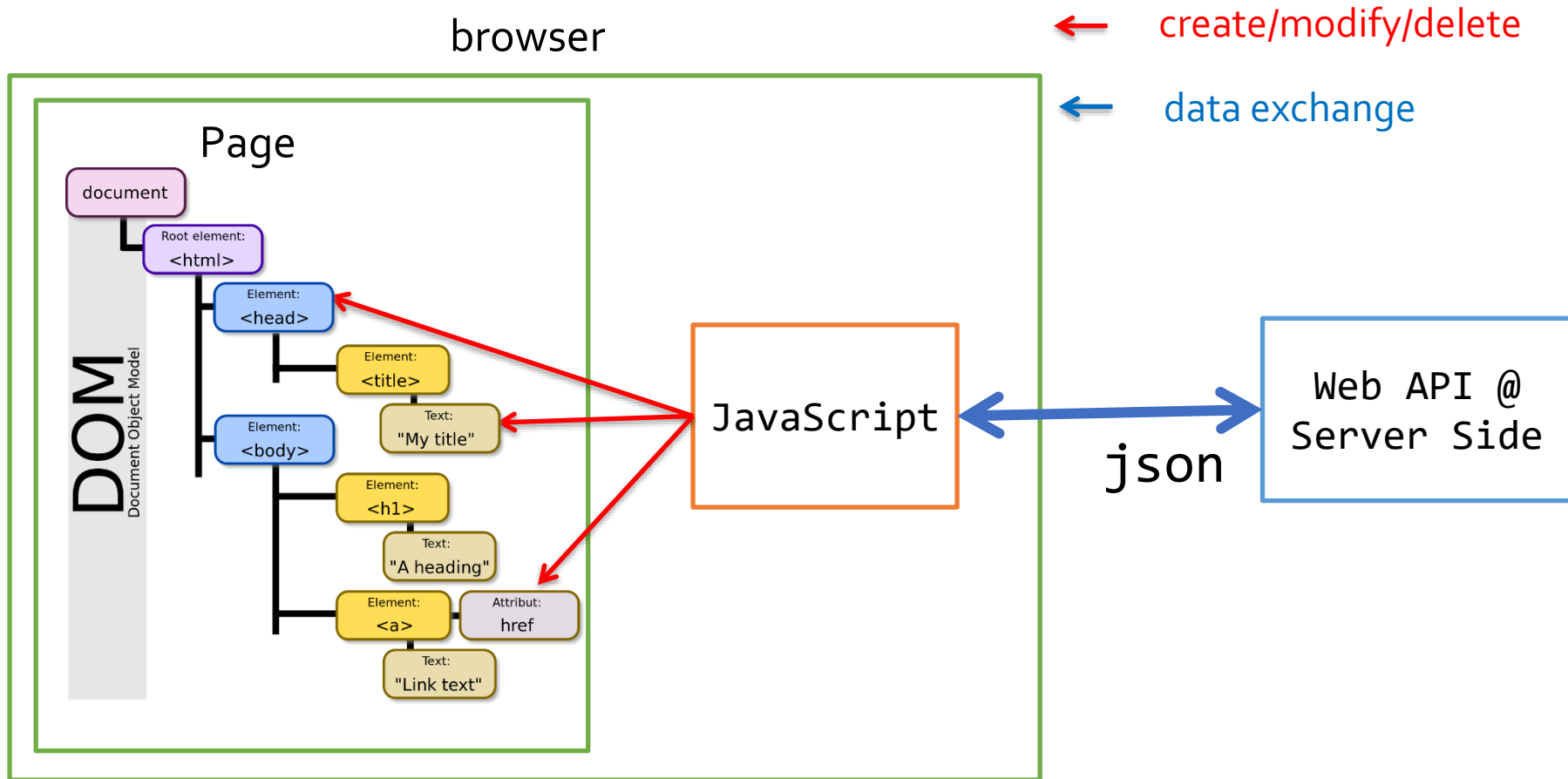
# Where to place JavaScript Code?

- The JavaScript code can be placed in:
  - `<script>` tag in the head
  - 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**

```
<script src="script.js"></script>
```

- JavaScript files usually have **.js** extension
- The **.js** files get cached by the browser

# Role of JavaScript on the Client Side

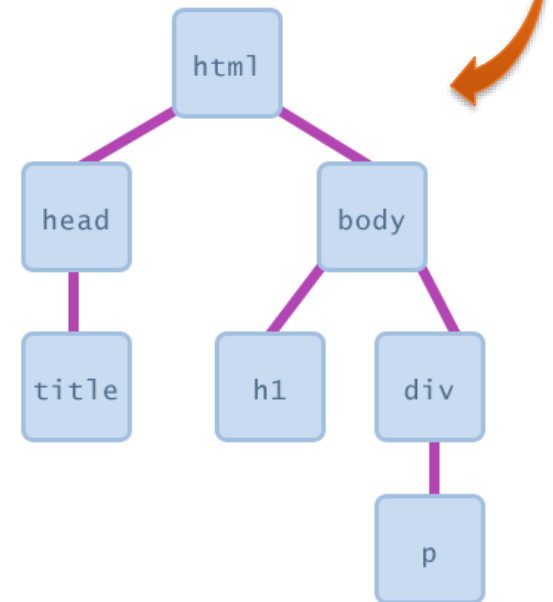


- DOM = A tree structure built out of the page HTML elements
- Use JavaScript to manipulate the DOM by changing the properties of DOM elements

# 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

```
<html>
<head>
  <title> ... </title>
</head>
<body>
  <h1> ... </h1>
  <div>
    <p> ... </p>
  </div>
</body>
</html>
```



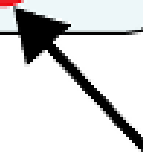
# Example DOM Element

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_queryselector.asp)
    - [https://www.w3schools.com/jsref/met\\_document\\_queryselectorall.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 `<img>` elements

# DOM Manipulation

- Once we **select** an element, we can read / change its attributes

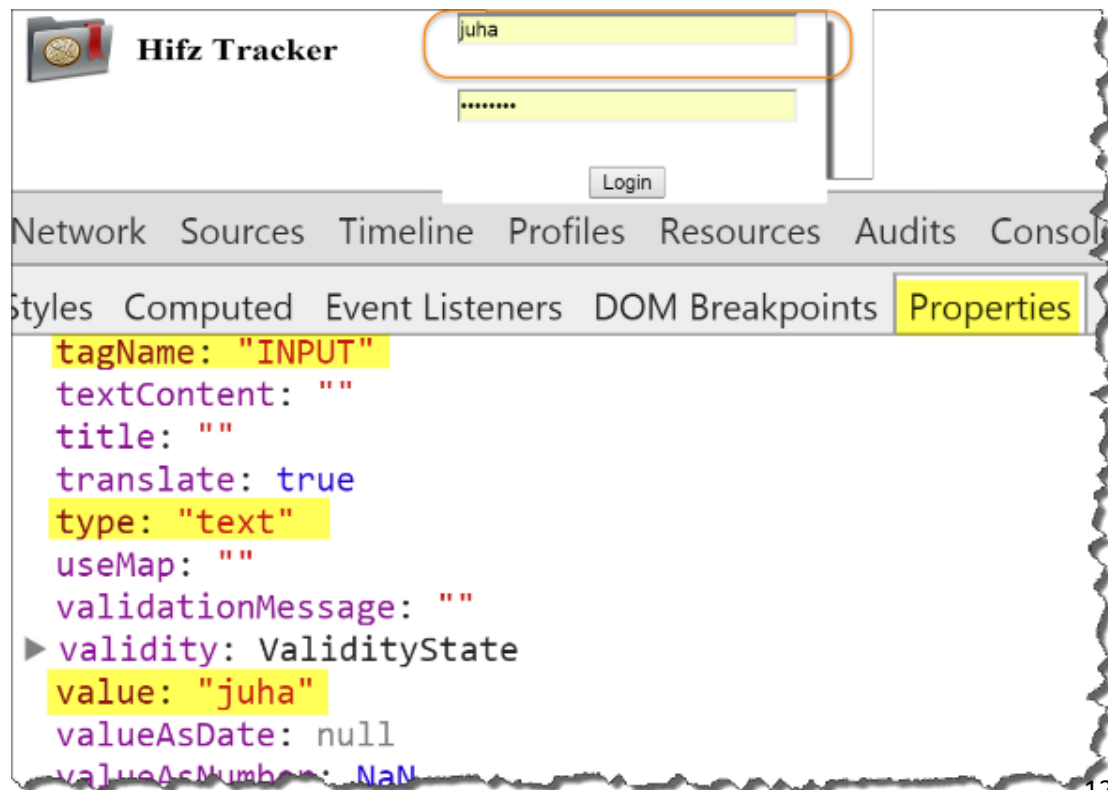
```
function change(state) {  
  let lampImg = document.querySelector("#lamp")  
  lampImg.src = `lamp_${state}.png`  
  let statusDiv =  
    document.querySelector("#statusDiv")  
  statusDiv.innerHTML = `The lamp is ${state}`  
}  
...  

```

# 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:

```
const 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](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)  
})
```



# The Event Object

```
function name (event) {  
    // an event handler function...  
}
```

- Event handlers can accept an optional parameter to represent the event that is occurring
- Event objects have the following properties/methods:

Property	Description
type	what kind of event, such as "click" or "mousedown"
target	the element on which the event occurred
timestamp	when the event occurred

# Stopping an Event

- `preventDefault()` stops the browser from doing its default action on an event.
  - for example, stops the browser from following a link when `<a>` tag is clicked
  - Or return false in an event handler to stop an event

```
<a href="#" onclick="onAddHero(event)">Add Hero</a>
```

```
async function onAddHero(event) {  
  event.preventDefault();  
  
  const heroesDiv = document.querySelector("#heroes");  
  const heroEditor = await getHeroEditor();  
  heroesDiv.innerHTML = heroEditor;  
}
```

# 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)
```

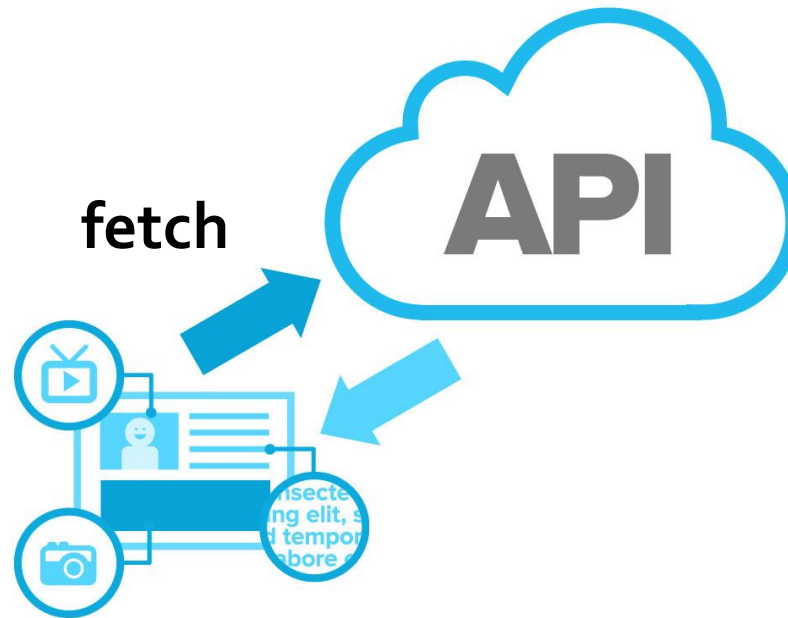
- DOM Traversal

```
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





- AJAX is acronym of Asynchronous JavaScript and ~~XML~~ JSON
  - AJAX is used for **asynchronously** fetching (in the background) of dynamic Web content and data from Web API
  - Allows dynamically adding elements into the DOM
- Two styles of using AJAX for **partial page update**
  - Load an HTML fragment and inject it in the page
  - Call Web API then use the received JSON object to update the page at the client-side using JavaScript

# Web API Get Request using Fetch

- 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

# Web API Post Request using Fetch

- 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*

# Template String to generate UI

``$ {...}``



# Template String to generate UI

- Using Template String to generate UI

```
function personTemplate({name, hobbies, job}){
  return `<article class="person">
    <h3>${name}</h3>
    <div>
      <div>Hobbies:</div>
      <ul>
        ${hobbies.map(hobby => `<li>${hobby}</li>`).join(" ")}
      </ul>
    </div>
    <p>Current job: ${job}</p>
  </article>`;
}
```

# Resources

- DOM

[https://developer.mozilla.org/en-US/docs/Web/API/Document Object Model/Introduction](https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction)

- Fetch API

[https://developer.mozilla.org/en-US/docs/Web/API/Fetch API](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API)