Dr. Abdelkarim Erradi

CSE @ QU

Web Dev

Web Client

Request

Response

Web Server

Frontend development

- HTML for page Structure & Content
- CSS for styling
- JavaScript for interaction

Backend development

- Dynamic Content
- Web API
- Data Management

express + node

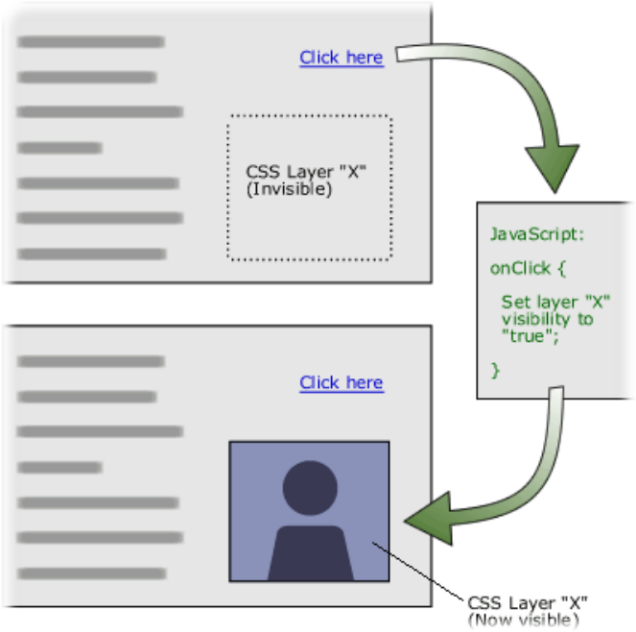
mongoDB

Table of Contents

Introduction to JavaScript

Dynamic Behavior at the **Client Side**

Or Server-Side Web applications



Back

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., valid the form data when the submit button is clicked
  - **Updating the page**
    - Add/update/delete 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

JavaScript Syntax

- JavaScript is syntactically a C family language
  - It differs from C mainly in its type system
- The JavaScript syntax is similar to Java and C#
  - Variables (by dynamically typed in JavaScript)
  - Operators (+, *, =, !=, &&, ++, …)
  - Conditional statements (`if`, `else`, `switch`)
  - Loops (`for`, `while`)
  - Arrays (`myArray[]`) and associative arrays (`myArray['abc']`)
  - Functions
  - Classes
- Although there are **strong outward similarities** between JavaScript and Java, the two are **distinct languages and differ greatly in their design**.

Data Types in JavaScript

Declaring Variables

- Names in JavaScript are case-sensitive
- The syntax is the following:

```
let <identifier> [= <initialization>];
```

- Example:

```
let height = 200;
```

- **let** – creates a block scope variable (accessible only in its scope)

```
for(let number of [1, 2, 3, 4]){
  console.log(number);
}
//accessing number here throws exception
```

Declaring Variables using var

- **var** – creates a variable accessible outside its scope (**avoid using var and use let**)

```
for(var number of [1, 2, 3, 4]){
  console.log(number);
}
console.log(number); //accessing number here is OK
```

Declaring a Constant

- **const** – creates a constant variable. Its value is read-only and cannot be changed

```
const MAX_VALUE = 16;
MAX_VALUE = 15; // throws exception
```

Primitive types

- JavaScript is a **Loosely Typed** and **Dynamic** language
  - The variable datatype is derived from the assigned value
- There are 6 data types in JavaScript:
  - number
  - string
  - boolean
  - undefined
  - function
  - object (Everything else is an object)
- A string is a sequence of characters enclosed in single (' ') or double quotes (" ")

```
let str1 = "Some text saved in a string variable";
let str2 = 'text enclosed in single quotes';
```

Template Literals

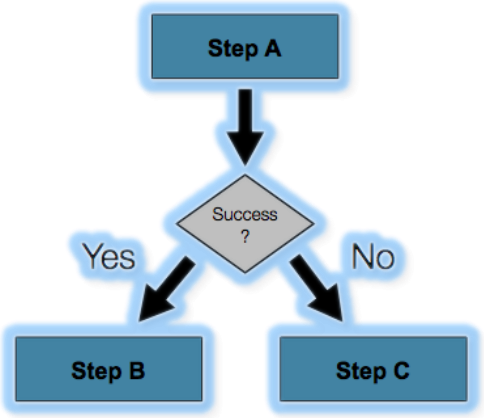
- Template Literals allow creating dynamic templated string with placeholders
  - Replaces long string concatenation!

```
let person = {fname: 'Samir',  lname:'Mujtahid'};

console.log(`Full name: ${person.fname} ${person.lname}`);
```

Comments

```
// slash slash line comment

/*
    slash star
    block
    comment
*/
```

Conditional Statements



Back

`if-else` Statement – Example

- Checking a number if it is odd or even

```
let number = 10;

if (number % 2 === 0)
{
    console.log('This number is even');
}
else
{
    console.log('This number is odd');
}
```

`switch-case` Statement

- Selects for execution a statement from a list depending on the value of the **`switch`** expression

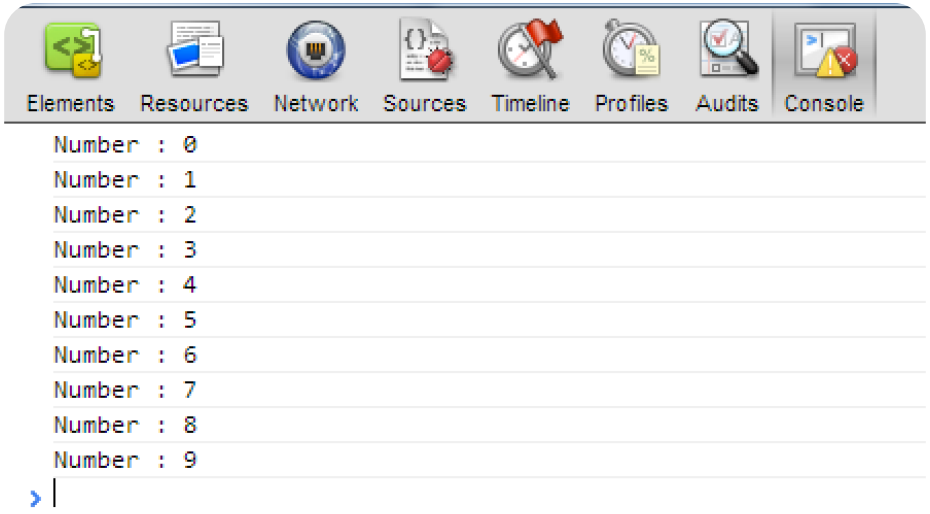
```
switch (day)
{
   case 1: console.log('Monday'); break;
   case 2: console.log('Tuesday'); break;
   case 3: console.log('Wednesday'); break;
   case 4: console.log('Thursday'); break;
   case 5: console.log('Friday'); break;
   case 6: console.log('Saturday'); break;
   case 7: console.log('Sunday'); break;
   default: console.log('Error!'); break;
}
```

while (…)
do { … }
for { … }
Loops

Execute Blocks of Code Multiple Times

While Loop – Example

```
let counter = 0;
while (counter < 10){
    console.log(`Number : ${counter}`);
    counter++;
}
```



Other loop structures

- Do-While Loop:

```
do {
    statements;
}
while (condition);
```

- **For** loop:

```
for (initialization; test; update) {
    statements;
}
```

For-of loop

- For-of loop iterates over a list of values

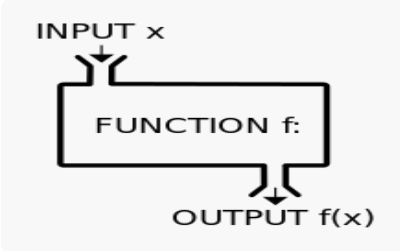
```
let sum = 0;
for(let number of [1, 2, 3])
   sum += number;
console.log(sum);
```

For-in loop

- For-in loop iterates over the properties of an object

```
let obj = { fName: "Ali", lName: "Mujtahid" };
for (let prop in obj) {
     console.log(prop , ':' , obj[prop]);
}
```

```
function (parameter) {
    return expression;
}
```




```
function double (number) { return number * 2; }
double(212); // call function


let average = function (a, b) {
    return (a + b) / 2;
}
average(10, 20); // call function
```

OR

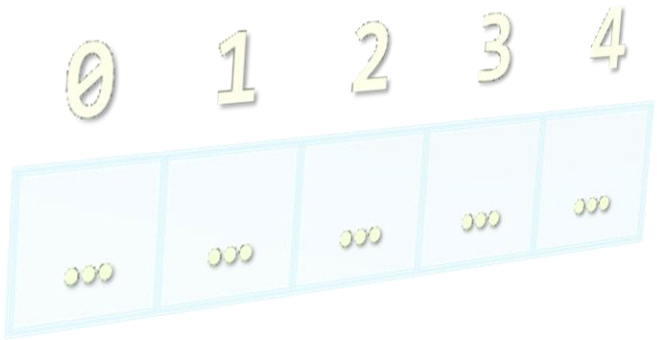
```
let average = (a, b) => (a + b) / 2;
average(10, 20);  // call function
```

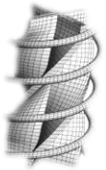
Arrow Function
Also called LAMBDA expressions

Arrays

Processing Sequences of Elements

https://sdras.github.io/array-explorer/





Processing Arrays Using for Loop

- The for-of loop iterates over a list of values

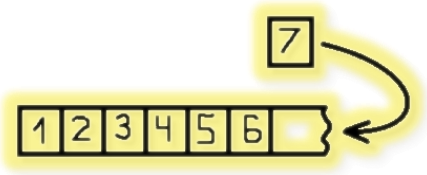
```
let sum = 0;
for(let number of [1, 2, 3])
  sum+= number;
```


- Printing array of integers in reversed order:

```
let array = [1, 2, 3, 4, 5];
for (let i = array.length-1; i >= 0; i--) {
    console.log(array[i]);
} // Result: 5 4 3 2 1
```

- Initialize an array:

```
for (let index = 0; index < array.length; index++) {
    array[index] = index;
}
```



Dynamic Arrays

- All arrays in JavaScript are dynamic
  - Their size can be changed at runtime
  - New elements can be inserted to the array
  - Elements can be removed from the array
- Methods for array manipulation:
  - `array.push(element)`
    - Inserts a new element at the tail of the array
  - `array.pop()`
    - Removes the element at the tail
    - Returns the removed element

Deleting Elements

- Splice removes item(s) from an array and returns the removed item(s)
- This method changes the original array
- Syntax:

array.splice(index,howmany)

```
let myArray = ['a', 'b', 'c', 'd'];
let removed = myArray.splice(1, 1);
// myArray after splice ['a', 'c', 'd']
```

Common operations on arrays

`.map` 

- Applies a function to each array element

`.filter(condition)` 

- Returns a new array with the elements that satisfy the condition

`.find(condition)` / `findIndex(condition)` 

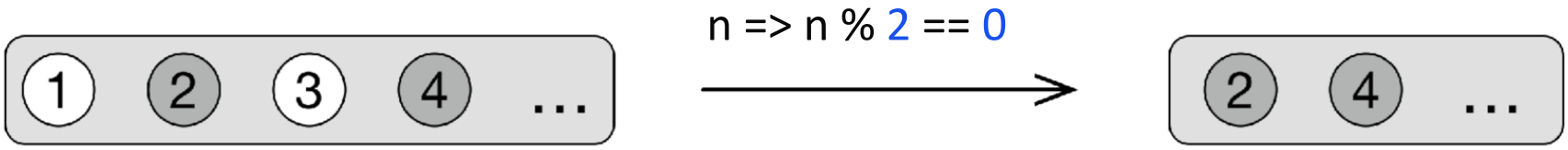
- Returns the first array element that satisfy the condition

`.reduce` 

- Applies an accumulator function to each array element to reduce them to a single value

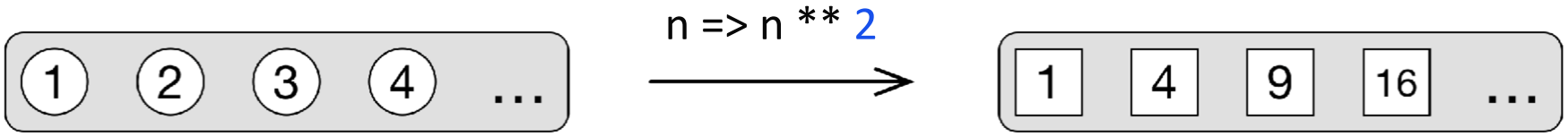
Filter

Return elements that satisfy a condition



Map

Transform elements by applying a Lambda to each element



Reduce

Apply an accumulator function to each element of the array to reduce them to a single value

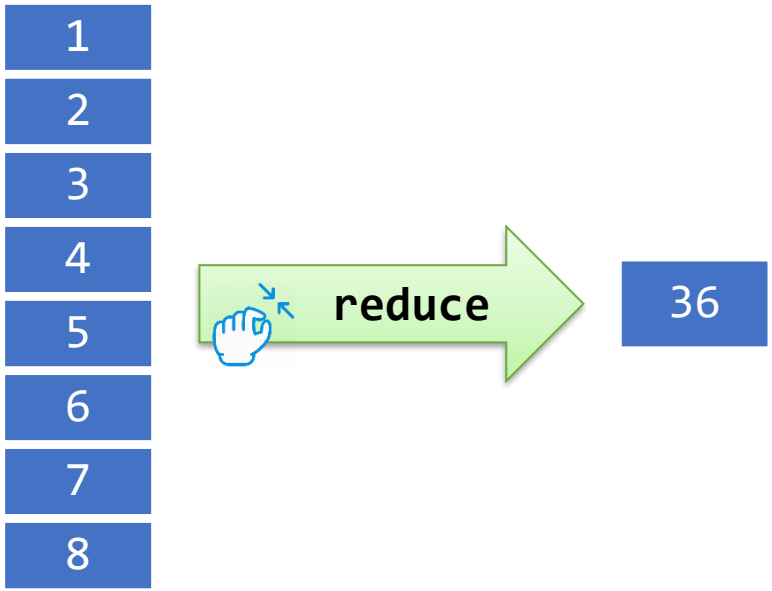
```
// Imperative
let sum = 0
for(let n of numbers)
    sum += n
```

```
//Declarative
let total = numbers.reduce ( (sum, n) => sum + n )
```

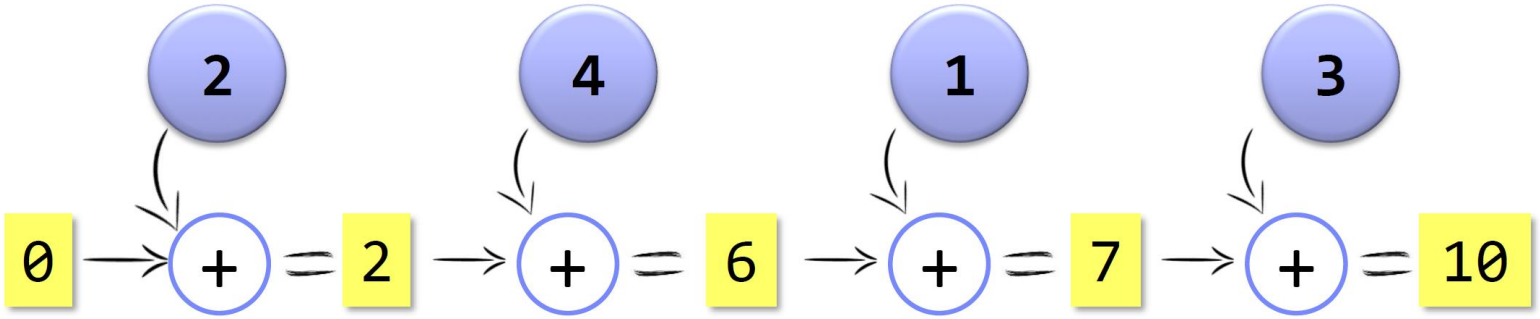
Accumulation Variable

Accumulation Lambda

Collapse the multiple elements of an array into a single element



Reduce




```
.reduce ( (sum, n) => sum + n )
```

Reduce is **terminal** operation that yields a single value

JSON

Create an Object Literal using JSON

(JavaScript Object Notation)

```
const person = {
    firstName: 'Samir',
    lastName: 'Saghir',
    height: 54,
    getName () {
        return `${this.firstName} ${this.lastName}`;
    }
};

//Two ways to access the object properties
console.log(person['height'] === person.height);

console.log(person.getName());
```

JSON.stringify and JSON.parse

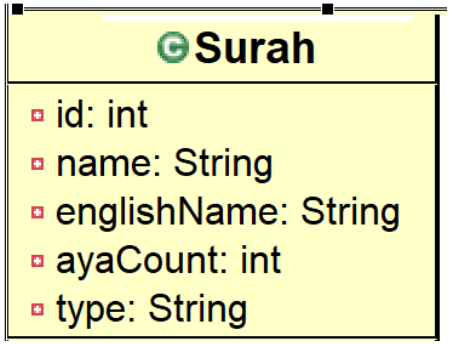
```
/* Serialise the object to a string in JSON format
-- only properties get serialised */
const jsonString = JSON.stringify(person);
console.log(jsonString);

//Deserialise a JSON string to an object
//Create an object from a string!
const personObject = JSON.parse(jsonString);
console.log(personObject);
```

- More info https://developer.mozilla.org/en-US/docs/JSON

JSON Data Format

- **JSON** is a very popular lightweight data format to transform an object to a **text** form to ease storing and transporting data
- **JSON** class could be used to transform an object to json or transform a json string to an object

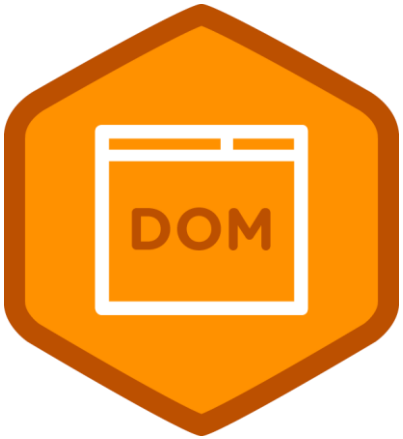
Transform an instance of Surah class to a JSON string:



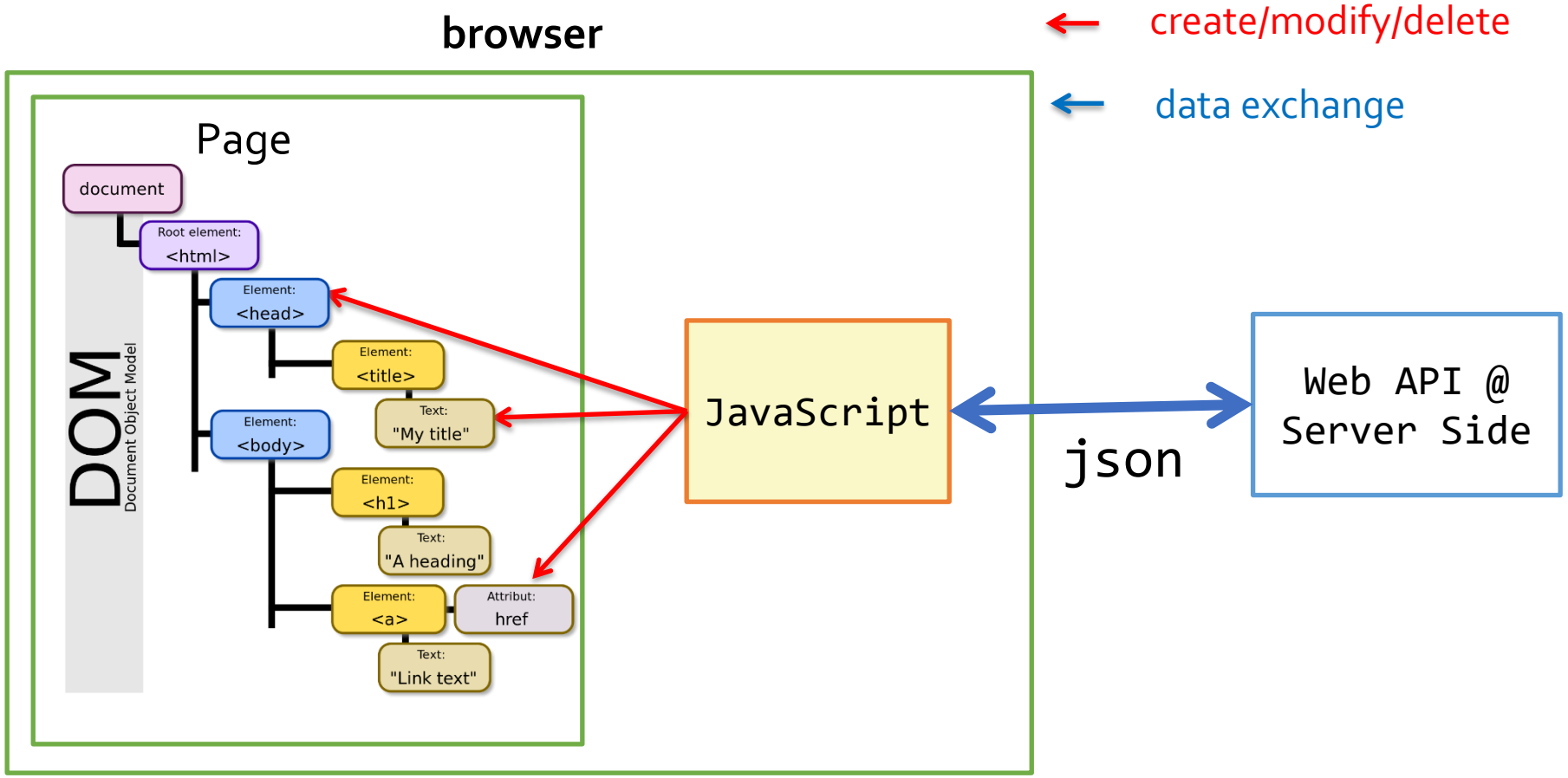

```
const fatiha = {id: 1, name: "الفاتحة",
englishName: "Al-Fatiha", ayaCount: 7, type:"Meccan")
const surahJson = JSON.stringify(fatiha)

// Converting a json string to an object
const surah = JSON.parse(surahJson)
```

```
{    "id": 1,
     "name": ","الفاتحة
     "englishName": "Al-Fatiha",
     "ayaCount": 7,
     "type": "Meccan"
 }
```

DOM Manipulation using JavaScript



Role of JavaScript on the Client Side



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

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

DOM Manipulation

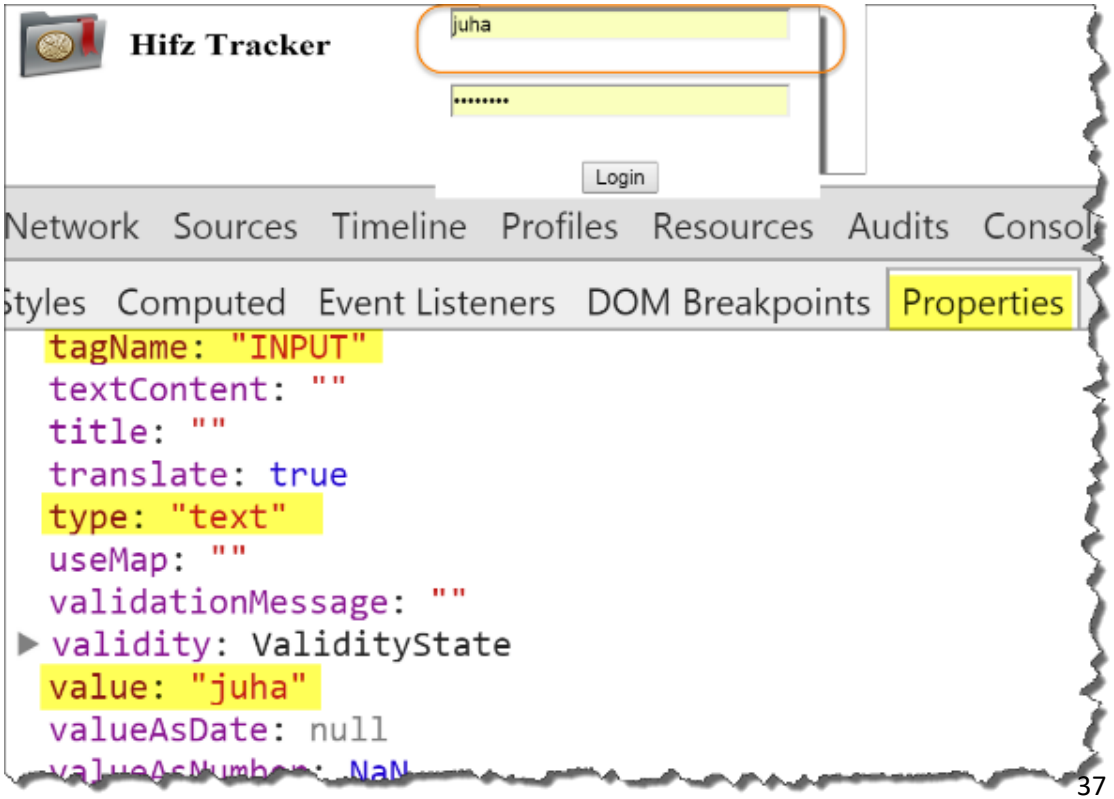
- Once we **select** an element, we can read / change its attributes

```
function change(state) {
  const lampImg = document.querySelector("#lamp")
  lampImg.src = `lamp_${state}.png`
  const statusDiv =
    document.querySelector("#statusDiv")
  statusDiv.innerHTML = `The lamp is ${state}`
}
…
<img src="lamp-on.jpg"  id="lamp"
  onmouseover="change('off')"
  onmouseout="change('on')" />
```

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



Commonly used DOM methods

- **Add Element**

```
const newDiv = document.createElement('div')
newDiv.innerText = 'Div added by script'
document.body.append(newDiv)
```

- **Remove Element**

```
document.querySelector('#myDiv').remove()
```

- **DOM Traversal**

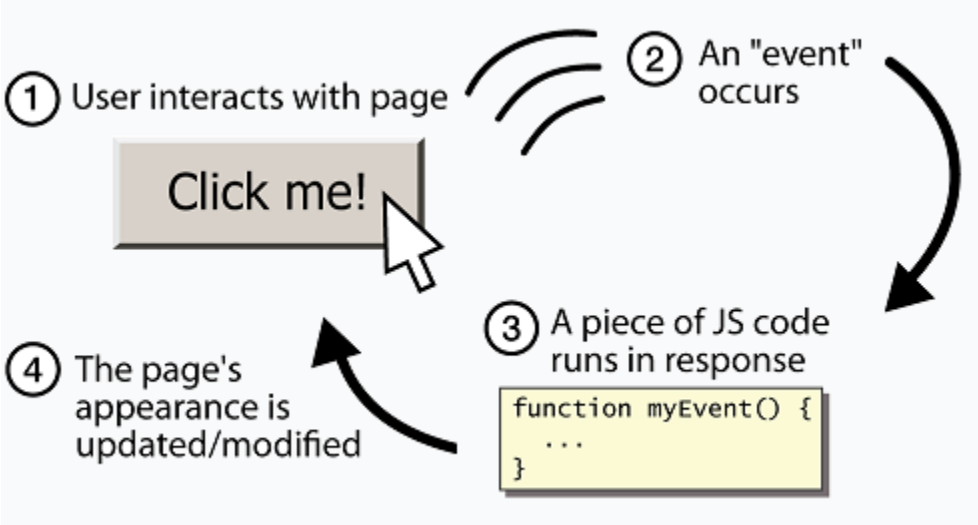
```
const parent = document.querySelector('#about').parentNode
const children = document.querySelector('#about').children
```

- **Hide & Show**

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

- **Add/Remove/Toogle CSS Classes**
  - `document.querySelector('#myDiv').classList.add('alert-success')`
  - `document.querySelector('#myDiv').classList.remove('alert-success')`
  - `document.querySelector('#myDiv').classList.toogle('alert-success')`

Event Handling



Events Handling

- UI elements raise Events when the user interacts with them (such as a Clicked event is raised when a button is pressed)
- JavaScript can register event handlers to respond to UI events
  - Events are fired by the Browser and are sent to the specified JavaScript **event handler** function
  - Can be set with HTML attributes:

```
<img src="test.gif" onclick="imageClicked()" />
```

  - 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

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)
})
```

4

`${...}`

HTML Template to generate the UI

HTML template

- **HTML template**: a piece of HTML text that has some parts to fill in (placeholders)
  - The placeholders are filled with data from objects, the rest remains always the same
  - HTML template has **static parts** and **dynamic parts** (the gaps to fill in)

```
Date: ___/___/___
Received from: ___________, the amount of QR_________
For: ____________________________________________
Received by: ___________________
```

  - This template can be printed and used many times filling in the blanks with the data of each payment.
- **Template literals** could be used to define an HTML template to generate the UI.

HTML template example

```
const payment = {
    date: '1/2/2021',
    name: 'Mr Bean',
    amount: 200,
    reason: 'Donation',
    receiver: 'Juha'
}

const receiptTemplate = (payment) =>
    `<div>
      <p>Date: ${payment.date}</p>
      <p>Received from: ${payment.name}, the amount of QR ${payment.amount}</p>
      <p>For: ${payment.reason}</p>
      <p>Received by: ${payment.receiver}</p>
    </div>`

console.log(receiptTemplate(payment));

// Render the template in the DOM
document.body.innerHTML = receiptTemplate(payment);
```

Template literals

Support:

- **Expression interpolation**: a template literal can contain placeholders `${expression}` that get evaluated to produce a string value

```
const a = 5, b = 10;
console.log(`${a} + ${b} = ${a + b}`);
```

- **Conditional expression**

```
const isHappy = true;
const state = `${ isHappy ? '😀' : '😢'}`;
console.log(state);
```

Display an Array using a Template literal

- Display an array elements using a template literal with the .map function

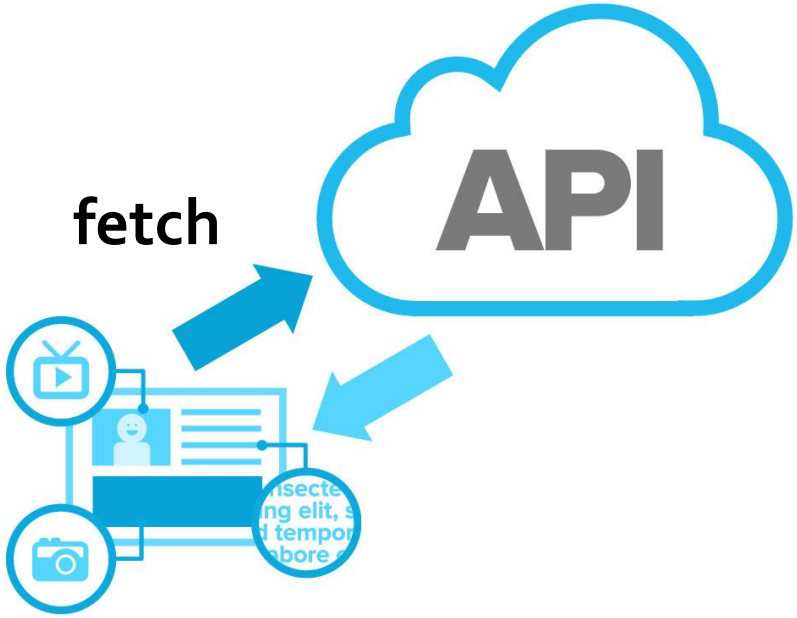
```
const  days = ["Mon", "Tue", "Wed", "Thurs", "Fri", "Sat", "Sun"];
const daysHtml = `<ul>
   ${days.map(day => `<li>${day}</li>`).join('\n')}
</ul>`;
console.log(daysHtml);
```

HTML template – Example 2

- Using HTML template to generate the UI

```
const person = {
    name: 'Mr Bean',
    job: 'Comedian',
    hobbies: ['Make people laugh', 'Do silly things', 'Visit interesting places']
}

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

// Render the template in the DOM
document.body.innerHTML = personTemplate(person);
```

Access Web API using Fetch



Web API Get Request using Fetch

- Fetch content from the server

```
async function getStudent(studentId) {
    const url = `/api/students/${studentId}`
    const 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

```
const 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
```

JavaScript Resources

- Mozilla JavaScript learning links
  - https://developer.mozilla.org/en-US/learn/javascript
- Fetch API

https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

- JavaScript features
  - https://github.com/mbeaudru/modern-js-cheatsheet
  - https://exploringjs.com/
- Modern JavaScript Tutorial
  - https://javascript.info/
- JavaScript code camp
  - https://www.freecodecamp.org/learn/javascript-algorithms-and-data-structures/
  - https://nodeschool.io/