AJAX & JSON

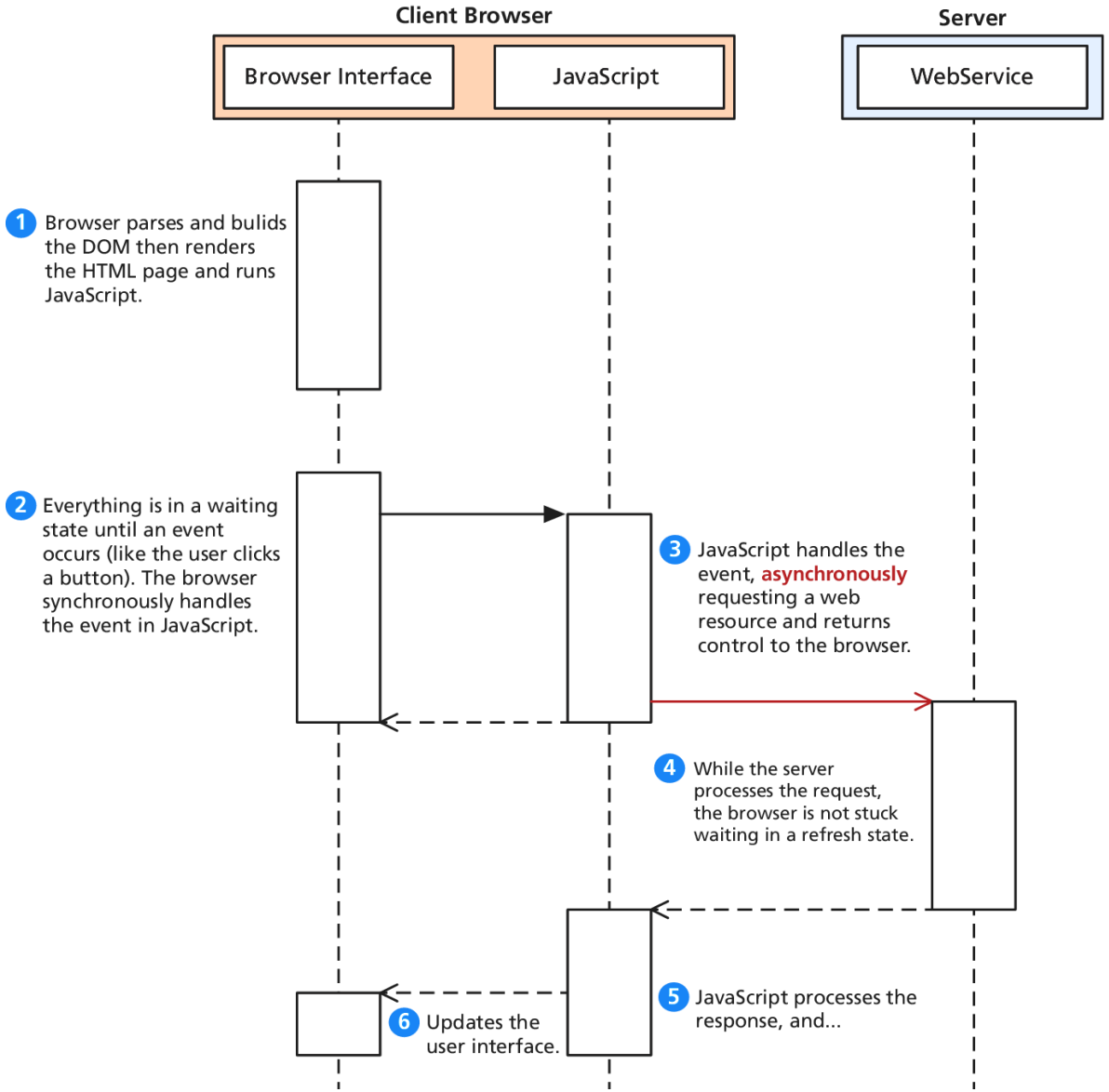
2020/21 COMP3322 Modern Technologies on WWW

Contents

- What is AJAX?
  - Demo 1 – Registration Form
- New form of AJAX - fetch()
  - Promise
  - async/await
  - Demo 2 – Registration Form
- Overview of JSON

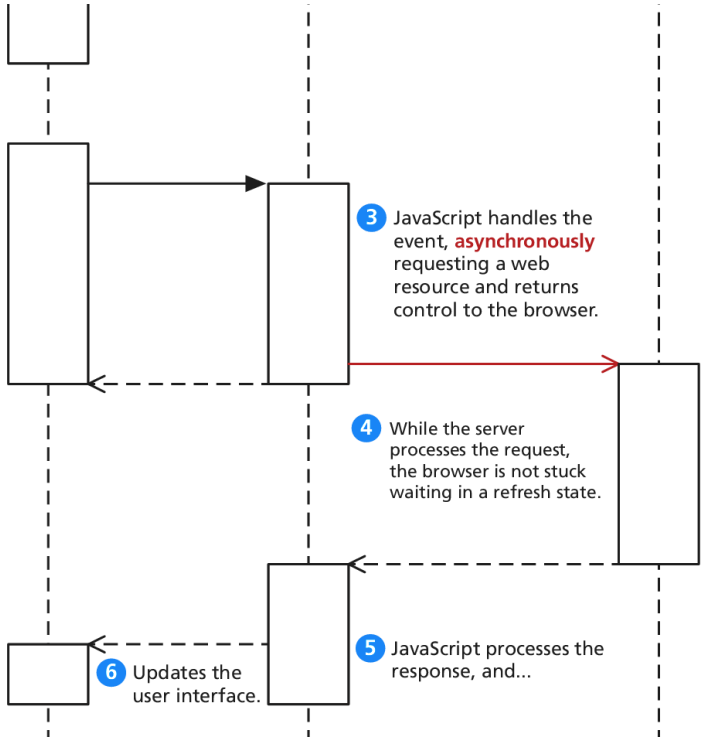
AJAX

- Asynchronous JavaScript with XML (AJAX) is a term used to describe a paradigm that allows a web browser to make quick, incremental updates to the web page without reloading the entire page.
  - Classic web pages must reload the entire page if the content changes.
- This makes the application faster and more responsive to user actions.
- Although X in AJAX stands for XML, JSON is used more than XML nowadays.
  - Actually, can be used to retrieve any type of data.



The XMLHttpRequest Object

- The **XMLHttpRequest object** is the core of AJAX.
- How AJAX works:
  - At client-side (JavaScript)
    - To send an HTTP request, **create** an XMLHttpRequest object.
    - ❸ Register a callback function to the ***onreadystatechange*** property of the XMLHttpRequest object.
    - ❸ Use the **open()** and **send()** methods of XMLHttpRequest object to sent the request to server.
      - Use either GET or POST method.
      - Usually asynchronously.
    - ❺ When the response arrives, the callback function is triggered to act on the data.
    - ❻ Usually rendering the web page using the DOM, that eliminates page refresh.



The XMLHttpRequest Object

- All modern browsers support the XMLHttpRequest object.
- However, old versions of Internet Explorer (5/6) use an ActiveX object instead.

```
// Old compatibility code, no longer needed.
if (window.XMLHttpRequest) { // Mozilla, Safari, IE7+ ...
    httpRequest = new XMLHttpRequest();
} else if (window.ActiveXObject) { // IE 6 and older
    httpRequest = new ActiveXObject("Microsoft.XMLHTTP");
}
```

- If you do not want to support old versions, just simple declare the object.

```
var httpRequest = new XMLHttpRequest();
if (!httpRequest) {
  // do something to alert user
}
```

Open() & send()

- To define the request, use open()

  `open(method, url, async, user, password)`

  - where:
    - method: the request type GET, POST, PUT, DELETE, etc
    - url: the server resource
    - async: [default] true if asynchronously; false if synchronously.

- To send the request, use send()

  `send() or send(string)`

  - string: only used for POST

```
httpRequest.open('GET', 'process.php?name=value', true);
httpRequest.send();
```

- To use POST to **send form data**, you may have to set the "Content-Type" of the request too.

```
httpRequest.open('POST', 'process.php', true);
httpRequest.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
httpRequest.send('name=value');
```

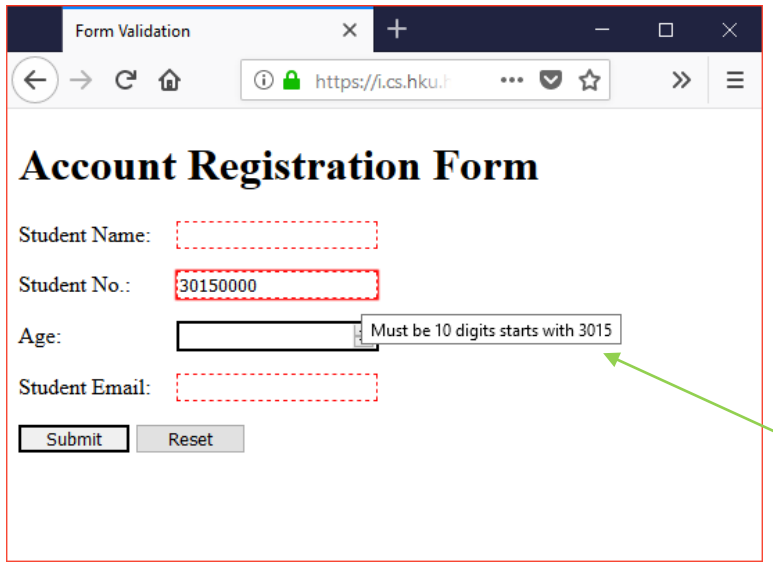
XMLHttpRequest Object Properties

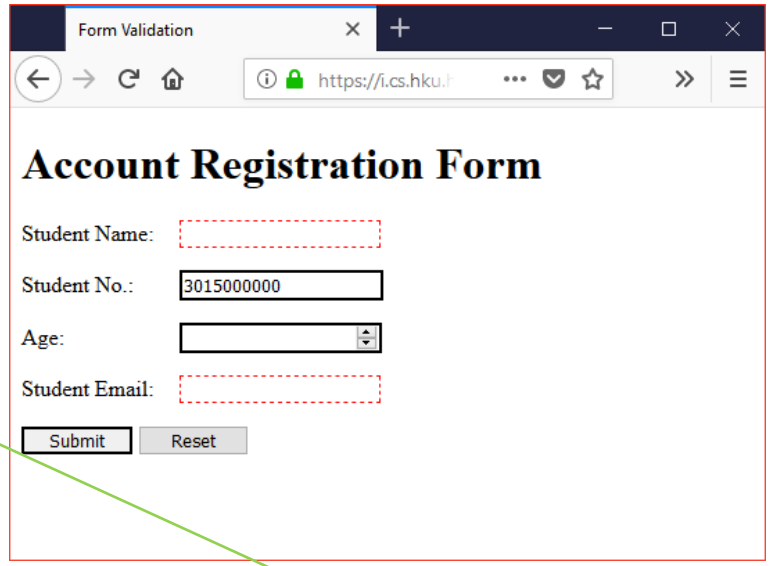
Property	Description
onreadystatechange	Registers a function to be called **when the readyState property changes**
readyState	Holds the status of the XMLHttpRequest object. 0: request not initialized 1: server connection established 2: request received 3: processing request 4: **request finished and response is ready**
responseType	Returns an enumerated value that defines the response type.
response	Returns the response's body content
responseText	Returns the response data as a string
responseXML	Returns the response data as XML data
status	Returns the status-number of a request, e.g, 200, 302, 404, ..
statusText	Returns the status-text

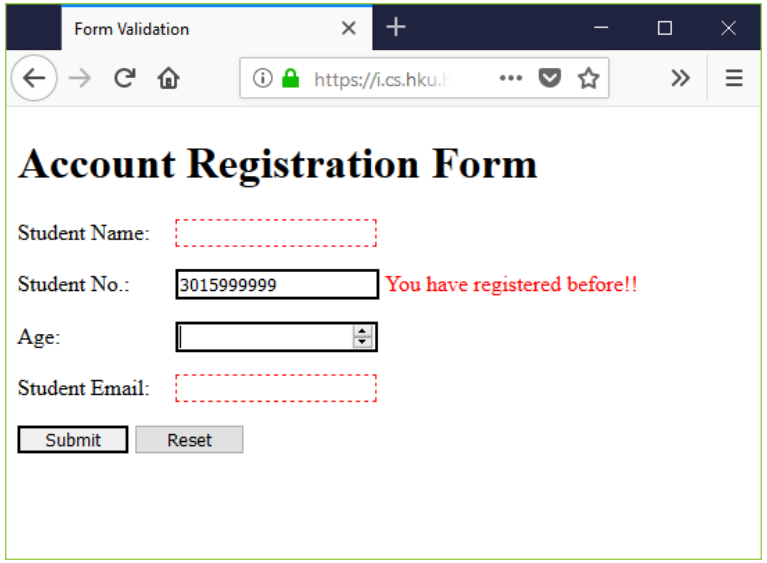
Register a callback function to the *onreadystatechange* property.

```
function callback() {
  if (receive good response)
      :
  else
      :
}
httpRequest.onreadystatechange = callback;
```

Demo 1 – Our Registration Form Again



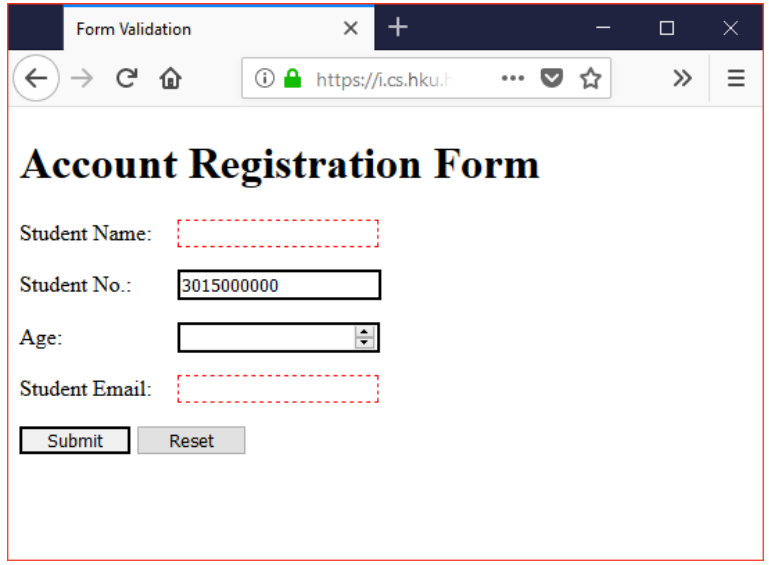


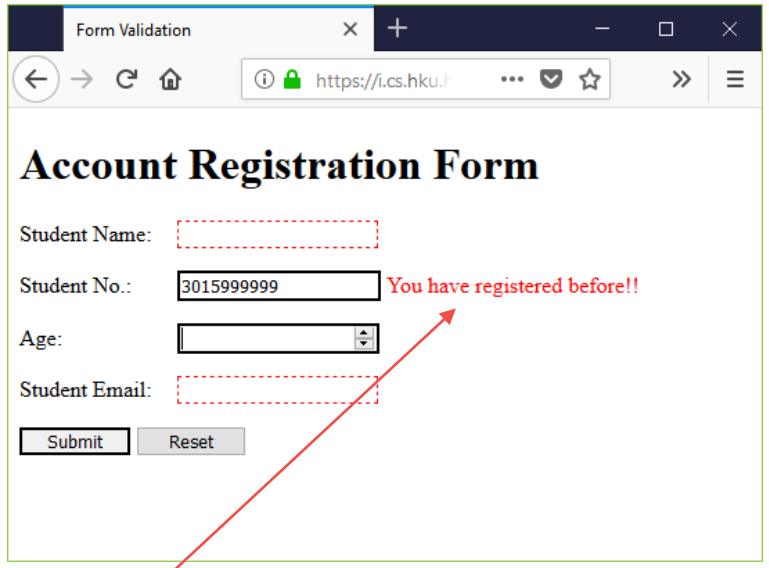



```html
<p>
  <label for="number">Student No.:</label>
  <input type="text" id="number" name="number"
maxlength="10" pattern="3015[0-9]{6}" required>
</p>
```

```javascript
var snum = document.getElementById("number");
snum.addEventListener("input", function (event) {
  if (snum.validity.patternMismatch) {
    console.log(snum.validationMessage);
    snum.setCustomValidity("Must be 10 digits starts with 3015");
  } else {
    snum.setCustomValidity("");
  }
});
```

Demo 1 – Add an empty inline element






```
<p>
  <label for="number">Student No.:</label>
  <input type="text" id="number" name="number" maxlength="10"
pattern="3015[0-9]{6}" required>
  <span id="chkReg"></span>
</p>
```

https://i7.cs.hku.hk/~atctam/c3322/AJAX/form-ajax.html

Demo 1 – Add AJAX

checking.php

```php
<?php

define("USERNAME", '3015999999');

if (isset($_GET['number']) && ($_GET['number'] == USERNAME))
{
  echo "You have registered before!!";
} else {
  echo "";
}
?>
```

```javascript
// create XMLHttpRequest object
var ajaxObj = new XMLHttpRequest();
if (!ajaxObj) {
  alert('Cannot create XMLHttpRequest object!!');
}

// onblur – when moving out of this field
function ajaxRequest() {
  ajaxObj.onreadystatechange = ajaxResponse;
  ajaxObj.open('GET', "checking.php?number="+snum.value, true);
  ajaxObj.send();
}
snum.addEventListener('blur', ajaxRequest); // add the onblur eventlistener

// the response callback function
function ajaxResponse() {
  if (ajaxObj.readyState == 4 && ajaxObj.status == 200) { // receive response with 200
    // add the received response text content to the placeholder
    document.getElementById('chkReg').innerHTML = ajaxObj.responseText;
  }
}
```

This object has two eventlisteners

Using Fetch()

- Fetch is a new **native JavaScript API**, which allows you to make network requests similar to XMLHttpRequest.
- Fetch API provides an interface for fetching (network) resources.
- By default the Fetch API uses the GET method, a simple call would be like this:

```
fetch(url) // passing the url as a parameter
.then(function() {
    // Code for handling the data you get from the API
})
.catch(function() {
    // Code to be run if the server returns any errors
});
```

Promise

- fetch() returns a **Promise** object that (eventually) resolves to the **Response** object to the fetch() request, whether it is successful or not.
- What is Promise?
  - Promise is a way to handle asynchronous operations.
  - Essentially, a promise is a **returned object** to the asynchronous operation which we can attach callbacks.
  - Callbacks will never be called before the completion of the (asynchronous) operation.
  - Callbacks added with then() even after the success or failure of the asynchronous operation, will be called.
  - Chaining - multiple callbacks may be added by calling then() several times.
    - Each callback is executed one after another, in the order in which they were inserted.

Promise Example

```
function tryluck(x) {
  function randomInt() {
    return Math.floor(Math.random() * 100);
  }

  var luck = new Promise(function(resolve, reject) {
    setTimeout(
      () => {
        if (randomInt()%2 == 0) {
          resolve("Finally Done["+x+"]!!!");
        } else {
          reject("Such a bad luck ["+x+"]##");
        }
      }, 3000);
  });

  luck.then(data => {
    console.log(data);
    return data+" Good!!!";
  })
  .then(data => {
    console.log(data);
  })
  .catch(err => {
    console.log(err);
  });

  console.log("Running after the promise ["+x+"]");
}

tryluck(1);
tryluck(2);
tryluck(3);
```

A function for generating random numbers between 0 to 99

The Promise constructor creates a Promise object.
We call resolve(...) when the async op was successful, and reject(...) when it failed.
In this example, we use setTimeout(...) to simulate async code.
In reality, you will probably be using something like XHR or an HTML5 API.

These are the callbacks

A synchronous operation

Response Object

- Represents the response to a request.
- We can get these information: and the response body:

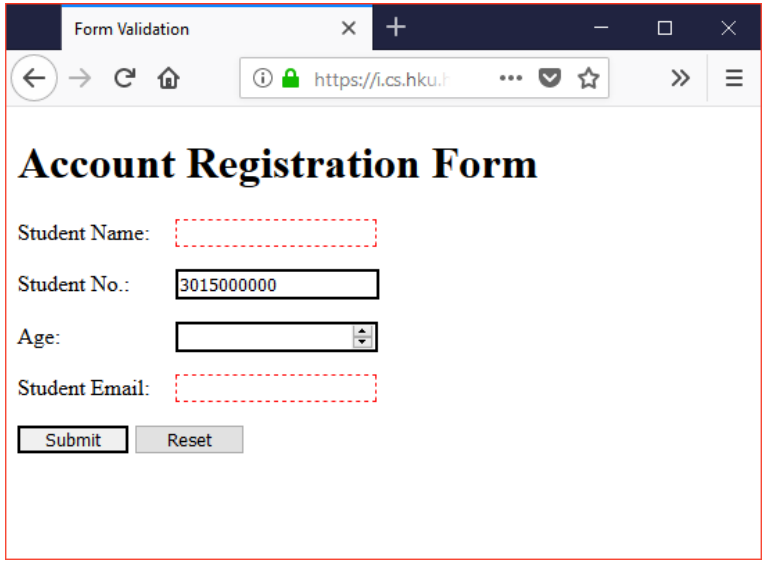
Properties	
status	HTTP response code in the 100–599 range
statusText	Status text as reported by the server
ok	True if status is HTTP 2xx
headers	The Headers object associated with the response
url	The URL of the response
type	The type of the response (e.g., basic, cors)
redirected	Indicates whether or not the response is the result of a redirection

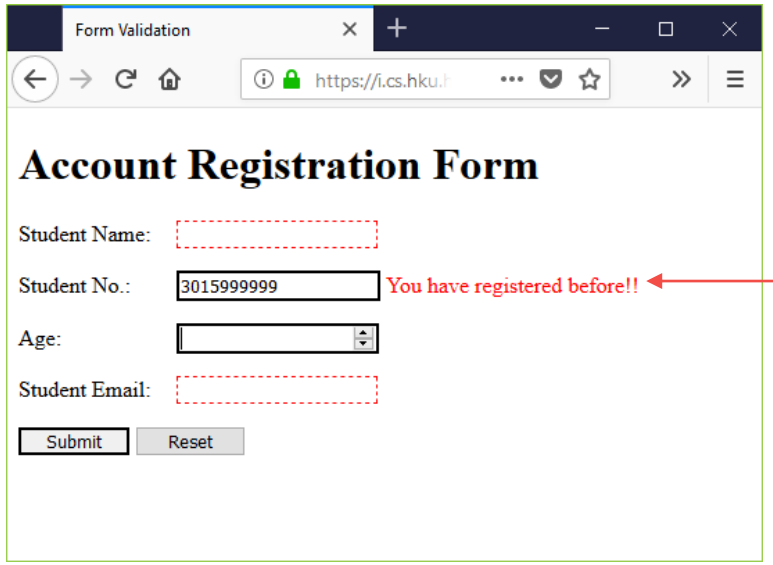
Methods	
text()	returns the body as a string
json()	parses the body text as JSON and returns a JSON object
blob()	returns the body as a Blob object
arrayBuffer()	returns the body as an ArrayBuffer object
formDate()	returns the body as a FormData object

Headers Object and Request Object

- Headers:
  - Represents response/request headers.
  - Allow you to query them and take different actions depending on the results.
  - Create your own headers before sending the fetch() request.
- Request:
  - Represents a resource request.
  - We can manually compose a Request object that uses the POST method and customize body and pass the Request object to fetch()

Demo 2 – Our AJAX Registration Form Again using fetch()






```
<p>
  <label for="number">Student No.:</label>
  <input type="text" id="number"
name="number" maxlength="10"
pattern="3015[0-9]{6}" required>
  <span id="chkReg"></span>
</p>
```

https://i7.cs.hku.hk/~atctam/c3322/AJAX/form-fetch.html

Demo 2 – Our AJAX Registration Form Again using fetch()

```
var snum = document.getElementById("number");
       :

//Using fetch()
snum.addEventListener('blur', fetchRequest);

function fetchRequest() {
  fetch('checking.php?number='+snum.value)
    .then( response => {
      if (response.status == 200) {
        response.text().then( data => {
          // add the received response text content to the placeholder
          document.getElementById('chkReg').innerHTML = data;
        });
      } else {
        console.log("HTTP return status: "+response.status);
      }
    })
    .catch( err => {
      console.log("Fetch Error!");
      });
}
```

checking.php

```
<?php

define("USERNAME", '3015999999');

if (isset($_GET['number']) && ($_GET['number'] == USERNAME)) {
  echo "You have registered before!!";
} else {
  echo "";
}
?>
```

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

Supplying Options to fetch()

- To customize your fetch() request, you need to set the second parameter **init object** with the following options:
  - First option: **method**
    - default is GET
    - POST, PUT, DELETE
  - Second option: **headers**
    - a set of name-value pairs (i.e, HTTP headers) or a headers object
  - Third option: **body**
    - Any body that you want to add to your request.
  - Fourth option: cache
    - The cache mode you want to use for the request.
  - Other options: referrer, keepalive, credentials, mode, . . .

fetch() - Using POST method

```
//Using fetch()
snum.addEventListener('blur', fetchRequest);

function fetchRequest() {
  let init = {
    method: 'POST',
    body: "number="+snum.value,
    headers: {'Content-Type': 'application/x-www-form-urlencoded'}
  };
  fetch('checkPost.php', init)
    .then( response => {
        if (response.status == 200) {
          response.text().then( data => {
            // add the received response text content to the placeholder
            document.getElementById('chkReg').innerHTML = data;
          });
        } else {
          console.log("HTTP return status: "+response.status);
        }
    })
    .catch( err => {
      console.log("Fetch Error!");
      });
}
```

`async/await`

- There's a special syntax to work with promises in a **more comfortable fashion**, called "async/await".
  - They make async code look more like old-school synchronous code, so they're well worth learning.
- **async function** declaration defines an asynchronous function, which **returns** an implicit **Promise**.
  - But the syntax and structure of code using async functions looks like standard synchronous functions.
- An async function X() can contain an **await expression** that **'pauses' the execution of X**.
  - Because this await expression is an async promise-based function Y() which needs certain duration **for resolving** Y's Promise,
  - When Y has resolved, the system resumes/continues the async function X's execution to generate the resolved value of X.
- The await keyword is only valid inside async functions.

async/await with fetch()

```
//Using fetch()
snum.addEventListener('blur', fetchRequest);

async function fetchRequest() {
  try {
    let response = await fetch('checking.php?number='+snum.value);
    if (response.status == 200) {
      let data = await response.text();
      // add the received response text content to the placeholder
      document.getElementById('chkReg').innerHTML = data;
    } else {
          console.log("HTTP return status: "+response.status);
    }
  } catch(err) {
      console.log("Fetch Error!");
  }
}
```

JSON

JavaScript Object Notation

- JSON is a standard **text-based** format for **representing structured data** based on **JavaScript object syntax**.
- Now is the most widely used **data format for data interchange** on the web.
  - JSON exists as a string — useful when you want to transmit data across a network.
- JSON is a human and machine readable format.
- It can be used independently from JavaScript, and **many programming languages** feature the ability to read (parse) and generate JSON.

Syntax and Structure

- JSON data is written as name/value pairs.
  - Key-value pairs have a colon between them.

```
"name":"John"
```

- A JSON **object** is a key-value data format that is typically rendered in curly braces.
  - Each key-value pair is separated by a comma.

```
{
  "first_name" : "Sammy",
  "last_name" : "Shark",
  "location" : "Ocean",
  "online" : true,
  "followers" : 987
}
```

- JSON values can be of one of 6 **simple data types**:
  - strings – must be written with double quotes
  - numbers
  - objects – circumscribed by curly braces { }
  - arrays – circumscribed by square brackets [ ]
  - booleans
  - null or empty

Syntax and Structure

- JSON can store a collection of related items as a JSON **array**.

```
[
 {
  "first_name" : "Sammy",
  "last_name" : "Shark",
  "location" : "Ocean",
  "online" : true,
  "followers" : 987
 },
 {
  "first_name" : "Jimmy",
  "last_name" : "Lee",
  "location" : "Peak",
  "online" : true,
  "followers" : 542
 }
]
```

- JSON can store nested objects in JSON format in addition to nested arrays.

```
{
    "name" : "Admin",
    "age" : 36,
    "rights" : [ "admin", "editor", "contributor" ]
}
```

Points to Note

- JSON is purely a data format - It contains only properties, no methods.
- JSON requires double quotes to be used around strings and **property names**. Single quotes are not valid.
  - Unlike in JavaScript code in which object properties may be unquoted, in JSON only quoted strings may be used as properties.
- Even a single misplaced comma or colon can cause a JSON file to go wrong, and not work.

Using JSON in JavaScript

- JavaScript provides a mechanism to **translate** a **JavaScript object** into a JSON string:

```
JSON.stringify(object);
```

- This method converts a JavaScript value (JSON object) to a JSON string representation.
  - It can optionally use a ***replacer* function** to replace values using custom logic.
- JavaScript provides the function:

```
JSON.parse(JSON string);
```

- This method converts a JSON string representation to a JavaScript value (JSON object).
  - It can optionally use a ***reviver* function** to perform a transformation on the resulting object before it is returned.

stringify() & parse()

```
var json_data =
{ library: {DVD: [
      {
        id: 1,
        title: "Breakfast at Tiffany's",
        format: "Movie",
        genre: "Classic"
      },
      {
        id: 2,
        title: "Contact",
        format: "Movie",
        genre: "Science fiction"
      }  ]
  }
};
```

```
var json_str = JSON.stringify(json_data);
console.log(json_str);
//->
{"library":{"DVD":[{"id":1,"title":"Breakfast at
Tiffany's","format":"Movie","genre":"Classic"},{"i
d":2,"title":"Contact","format":"Movie","genre":"S
cience fiction"}]}}
```

```
var clone = JSON.parse(json_str);
console.log(clone.library.DVD[1].title);
//-> Contact
```

```
console.log(json_data.library.DVD[0].title);
//-> Breakfast at Tiffany's
```

Reading

- AJAX
  - MDN web docs
    - AJAX – Getting Started
      - https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX/Getting_Started
- Fetch API
  - MDN web docs
    - Using Fetch
      - https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch
- JSON
  - MDN web docs
    - Working with JSON
      - https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON

References

- Some slides are borrowed from the book:
  - Fundamentals of Web Development by Randy Connolly and Ricardo Hoar, published by Pearson.
- MDN Web Doc
  - Ajax - https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX
  - Fetch API - https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API
- Learn AJAX – W3Schools.com
  - https://www.w3schools.com/js/js_ajax_intro.asp