INFO2180 - LECTURE 6

JQUERY, AJAX AND FORM VALIDATION

JQUERY

JQUERY IS A FAST, SMALL, AND FEATURE-RICH JAVASCRIPT LIBRARY WHICH MAKES THINGS LIKE HTML DOCUMENT TRAVERSAL AND MANIPULATION, EVENT HANDLING, ANIMATION, AND AJAX MUCH SIMPLER.

http://jquery.com/

ADDING JQUERY TO YOUR HTML PAGE

```
<html>
  <head>
    <script src="https://code.jquery.com/
jquery-3.6.1.min.js"></script>
    <script src="myscripts.js"></script>
  </head>
  <body>
    <h1>My Web page</h1>
  </body>
</html>
```

JQUERY OBJECT

```
$("p")
// or
jQuery("p")
```

JQUERY DOCUMENT READY

```
$( document ).ready(function() {
    // Your code here.

});
```

SELECTING ELEMENTS WITH JQUERY

```
let someId = $("#myId");
let someClass = $(".my-class");
let someElem = $("p");
let myUlInADiv = $("div ul");
```

Any CSS selector can be used.

JQUERY EVENT BASICS

```
$("a").click(function() {
    console.log("You clicked a link!");
});
```

```
$("a").on("click", function() {
    console.log("You clicked a link");
});
```

You can reference any event handlers (e.g. mouseover, keypress, etc.)

JQUERY ADD AND REMOVE CLASSES

```
$("a").addClass("test");
```

```
$("a").removeClass("test");
```

JQUERY SET AND GET ATTRIBUTES

```
$("div").attr("id", "my-id");
```

```
$("a").attr("href");
```

JQUERY EFFECTS

- jQuery has some handy methods to produce simple effects.
- `show()` and `hide()`
- `slideUp()` and `slideDown()`
- `fadeOut` and `fadeIn()`
- `toggle()`, `slideToggle()`, `fadeToggle()` - This will toggle the display of the matched element.
- And there are many others.

EXAMPLES OF JQUERY EFFECTS

```
// Instantaneously hide all
paragraphs
$( "p" ).hide();

// Instantaneously show all divs that
have the hidden style class
$( "div.hidden" ).show();
```

EXAMPLES OF JQUERY EFFECTS

```
// slowly hide all paragraphs
$( "p" ).hide("slow");

// quickly show all divs that have
the hidden class
$( "div.hidden" ).show("fast");
```

EXAMPLES OF JQUERY EFFECTS

```
// hide all paragraphs over half a
second
$( "p" ).hide(500);

// show all divs that have the hidden
class over 1.25 secs
$( "div.hidden" ).show(1250);
```

These times are in milliseconds (ms)

AJAX

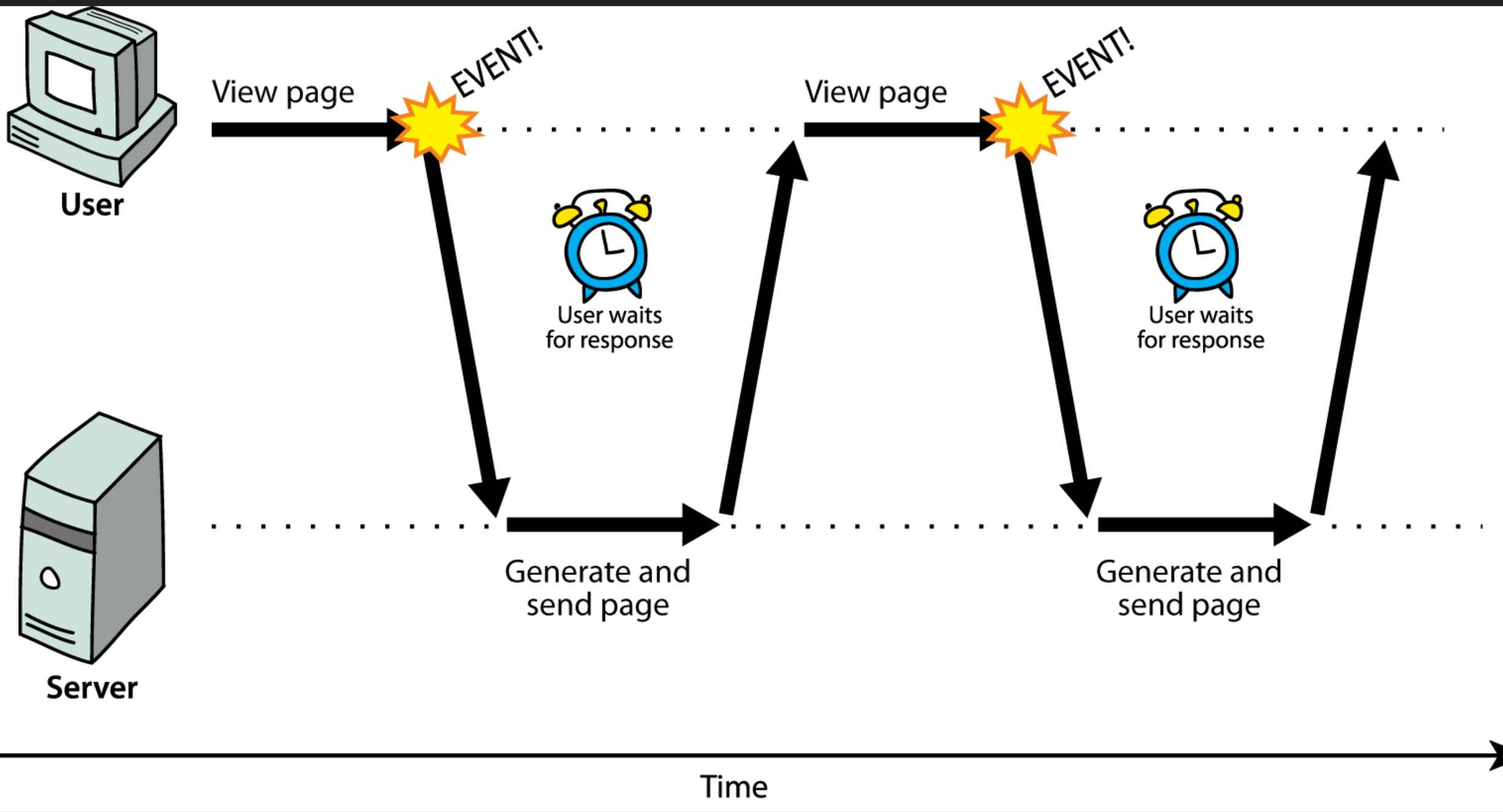
AJAX STANDS FOR ASYNCHRONOUS JAVASCRIPT AND XML.

https://developer.mozilla.org/en-US/docs/AJAX/Getting_Started

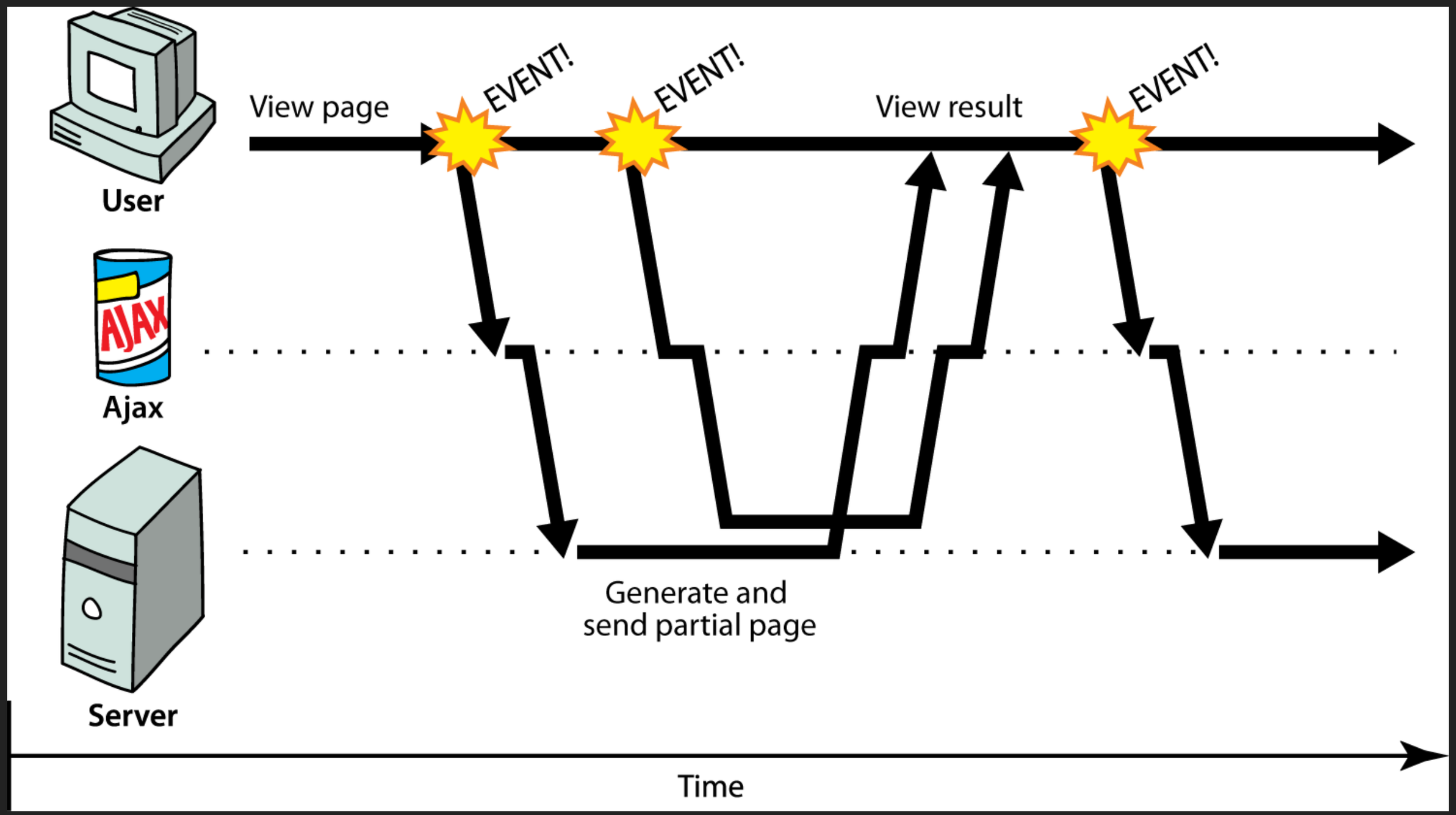
AJAX OVERVIEW

- It is not a programming language, but rather a particular way of using JavaScript.
- It uses the **`XMLHttpRequest`** object to communicate with server-side scripts.
- It can send as well as receive information in a variety of formats, including JSON, XML, HTML, and even text files.
- its "asynchronous" nature means it can update portions of a page without having to refresh the entire page.

SYNCHRONOUS COMMUNICATION



ASYNCHRONOUS COMMUNICATION



MAKING AN AJAX REQUEST

```
const httpRequest = new XMLHttpRequest();

let url = "http://example.com/somepage.php";

httpRequest.onreadystatechange = doSomething;

httpRequest.open('GET', url);

httpRequest.send();
```

HANDLING THE RESPONSE TO AN AJAX REQUEST

```
function doSomething() {
 if (httpRequest.readyState === XMLHttpRequest.DONE) {
   if (httpRequest.status === 200) {
     let response = httpRequest.responseText;
     alert(response);
   } else {
     alert('There was a problem with the request.');
   }
 }
}
```

JQUERY AJAX EXAMPLE

```
$.ajax("somepage.php")
   .done(function(result) {
     $("#weather-temp").html("<strong>" + result +
"</strong> degrees");
   }).fail(function(result) {
     $("#message").html("There seems to be an error.");
   });
```

FETCH API EXAMPLE

```
fetch('https://example.com/somepage.php')
  .then(response => response.text())
  .then(data => {
    // Here's some data!

    console.log(data)

  })
  .catch(error => {
    console.log(error);
  });
```

HANDLING ASYNCHRONOUS OPERATIONS IN JAVASCRIPT

CALLBACKS

CALLBACKS

- A **Callback** is a simple function that is passed as a value to another function, and will only be executed after the other function has finished executing.
- In other words, callbacks are a way to make sure certain code doesn’t execute until other code has already finished execution.

A SIMPLE EXAMPLE OF A CALLBACK

```
setTimeout(function(){
  // runs after 2 seconds
}, 2000)
```

```
window.addEventListener('load', function(){
  // window loaded
  // do what you want
})
```

THE PROBLEM WITH CALLBACKS

- Callbacks can get troublesome when you have lots of callbacks nested. This causes your code to get complicated very quickly.
- When this happens, you may hear people refer to it as "*Callback Hell*" or the "Pyramid of Doom".

AN EXAMPLE OF CALLBACK HELL

```
doSomething(function(result) {
  doSomethingElse(result, function(newResult) {
    doThirdThing(newResult, function(finalResult) {
      console.log('Got the final result: ' + finalResult);
    }, failureCallback);
  }, failureCallback);
}, failureCallback);
```

AN EXAMPLE OF CALLBACK HELL

```
window.addEventListener('load', () => {
  document.getElementById('button')
    .addEventListener('click', () => {
      setTimeout(() => {
        items.forEach(item => {
          // your code here
        })
      }, 2000)
    })
})
```

ALTERNATIVES TO CALLBACKS

- Starting with ES6, JavaScript introduced several features that help us with asynchronous code that do not involve using callbacks: Promises (ES6) and Async/Await (ES2017).

PROMISES

PROMISES

- Promises are JavaScript objects that represent the eventual result of an asynchronous operation.
- Promises can be in one of three states: **`pending`**, **`fulfilled`**, or **`rejected`**.
- A promise is settled if it is either fulfilled or rejected.
- We construct a promise by using the **`new`** keyword and passing an **`executor`** function to the Promise constructor method.
- The executor function typically has two params, **`resolve()`** and **`reject()`**. Resolve changes the promises status from pending to fulfilled, reject on the other hand changes it from pending to rejected.

AN EXAMPLE OF DEFINING A PROMISE

```
let greatDay = true;
const isItAGreatDay = new Promise((resolve, reject) => {
  if (greatDay) {
    resolve("Today is a Great Day");
  } else {
    reject("It's not the best of days");
  }
});
```

PROMISES

- Once we have defined a promise we use `.then()` with a success handler callback containing the logic for what should happen if a promise resolves.
- We use `.catch()` with a failure handler callback containing the logic for what should happen if a promise rejects.
- We can also chain multiple `.then()`'s and `.catch()`'s if we need to.

AN EXAMPLE OF CONSUMING A PROMISE

```
/* ...continuing from the previous coding example */

isItAGreatDay(greatDay)
  .then((resolvedValue) => {
    console.log(resolvedValue);
    // Do something with the successful value
})
.catch((error) => {
  console.log(error);
  // Do something with the rejected/error value
});
```

ASYNC...AWAIT

ASYNC/AWAIT

- **`async`**...**`await`** syntax allows us to write asynchronous code that reads similarly to traditional synchronous, imperative programs.
- It is considered syntactic sugar, as it introduces a new syntax for using Promises and Generators.
- This helps to improve the readability and scalability of our code.

ASYNC...AWAIT

- You declare an async function with the keyword **`async`**.
- Then within the function we would use the **`await`** operator to pause the execution of the function until our asynchronous action completes.
- If we want to we could have multiple await statements to make our code read like synchronous code.
- To handle errors with our async functions we use **`try`**...**`catch`** statements.
- It is also important to note that prepending the **`async`** keyword to any function means that the function will return a promise.

A SIMPLE EXAMPLE OF USING ASYNC...AWAIT

```
const doSomethingAsync = () => {
  return new Promise(resolve => {
    setTimeout(() => resolve('I did something'), 3000)
  })
}

const doSomething = async () => {
  console.log(await doSomethingAsync())
}

console.log('Before')
doSomething()
console.log('After')
```

EXAMPLE USING FETCH API USING PROMISES

```
const getFirstStudentData = () => {
  return fetch('/students.json') // get student list
    .then(response => response.json()) // parse JSON
    .then(students => students[0]) // pick first student
    .then(student => fetch(`/students/${student.name}
`)) // get user data
    .then(studentResponse => studentResponse.json()) //
parse JSON
}
```

EXAMPLE THIS TIME USING FETCH API WITH ASYNC...AWAIT

```
const getFirstStudentData = async () => {
  const response = await fetch('/students.json') // get users
list
  const students = await response.json() // parse JSON
  const student = students[0] // pick first user
  const studentResponse = await fetch(`/students/${student.name}
`) // get user data
  const studentData = await userResponse.json() // parse JSON
  return studentData
}
```

FORM VALIDATION

Validation is the process of ensuring that user input is clean, correct, and useful.

VALIDATION CAN HELP TO…

- prevent blank (empty) values
- ensure values are of a particular type or format (e.g. integer, currency, phone number, TRN number, email address, credit card, date)
- ensure values are between a specific range.
- ensure two values match (e.g. email or password confirmation)

VALIDATION TYPES

- Client side validation is performed by a web browser, before input is sent to a web server.
- Server side validation is performed by a web server, after input has been sent to the server.

Never trust user input.

Always validate user input on both the client-side and server-side.

CLIENT-SIDE FORM VALIDATION

CLIENT-SIDE VALIDATION

- This is typically done using JavaScript.
- However, HTML5 introduced some basic validation already built into the browser using the `type`, `pattern` and `required` attributes to input elements.

HTML5 REQUIRED ATTRIBUTE

```
<input type="text" name="firstname" required />
```

HTML5 TYPE ATTRIBUTE

```
<input type="number" />

<input type="tel" />

<input type="url" />

<input type="email" />
```

Some types come with built-in validation in the browser.

HTML5 PATTERN ATTRIBUTE

```
<input type="tel" name="telephone"
pattern="^\d{3}-\d{3}-\d{4}$" />
```

Uses JavaScript Regular Expressions (RegEx).

JavaScript can also be used to do client-side form validation.

SOME JAVASCRIPT PROPERTIES/FUNCTIONS

- `trim()` function
- `length` property
- Regular Expression `test()` method

TRIM EXAMPLE

```
let stringWithSpaces = "   hello   ";
console.log(stringWithSpaces.trim());
//-> hello
```

`trim()` removes any whitespace at the start and end of a string.

LENGTH EXAMPLE

```
let someString = "hello";
console.log(someString.length);
//-> 5
```

`length` will return the number of characters in a string.

REGEX TEST EXAMPLE

```
let telephoneNumber = "876-999-1234";

let exp = /^\d{3}-\d{3}-\d{4}$/;

console.log(exp.test(telephoneNumber));

//-> true
```

`test()` will check if our string matches our regular expression.

REGULAR EXPRESSIONS (REGEX)

REGULAR EXPRESSIONS ARE PATTERNS USED TO MATCH CHARACTER COMBINATIONS IN STRINGS.

https://developer.mozilla.org/en/docs/Web/JavaScript/Guide/Regular_Expressions

OVERVIEW OF REGULAR EXPRESSIONS

- You can also use regex's for search/replace
- They usually start and end with "**/**" e.g. **/abc/**
- A dot **.** matches any character except a line break (\n)
- A trailing **i** at the end of a regex signifies case-insensitive matching. (e.g. **/howdy/i**, can match "Howdy", "hOwDy", etc.)
- **|** stands for **OR** (e.g. **/red|blue/** will match "red" or "blue"

OVERVIEW OF REGULAR EXPRESSIONS

- `( )` for grouping e.g. `/(John|Jane) Doe/` matches "John Doe" or "Jane Doe"
- `^` means begins with (e.g. `/^A/` will match the A in "Apple" but not "an Apple".
- `$` means ends with. (e.g. `/t$/` will match the t in "hot" but not "hotter".
- `\` begins an escape sequence since some characters are special to regexes. So if you want them to match them literally you will need to escape them. (e.g. `/\$100\.00/`)

OVERVIEW OF REGULAR EXPRESSIONS

- `*` means 0 or more occurrences (e.g. `/abc*/` matches "ab", "abc", "abcc", "abccc", ...)
- `+` means 1 or more occurrences (e.g. `/Goo+gle/` matches "Google", "Gooogle", "Goooogle", ...)
- `?` means 0 or 1 occurrence (e.g. `/a(bc)?/` matches "a" or "abc")
- `{min, max}`, minimum and maximum occurrences. e.g. `/(ha){2,4}/` would match "haha" or "hahaha" or "hahahaha".
- `[]` group in a set; So it will match any single character from the set. e.g. `/[bcm]at/` would match strings containing "bat", "cat", "mat"

OVERVIEW OF REGULAR EXPRESSIONS

- Ranges can be done with **`/[a-zA-Z0-9]/`**. That will match any lowercase, uppercase letter or digit.
- **`/[^abcd]/`** matches anything except a, b, c or d. The **`^`** in **`[]`** represents **`NOT`**.
- **`\d`** matches any digit (same as **`[0-9]`**); **`\D`** any non-digit (**`[^0-9]`**)
- **`\w`** matches any word character (same as **`[a-zA-Z_0-9]`**); \W any non-word char
- **`\s`** matches any whitespace character (**`, \t, \n,`** etc.); **`\S`** any non-whitespace

OVERVIEW OF REGULAR EXPRESSIONS

To learn about and experiment more with Regular Expressions you can view http://regexr.com/

SERVER-SIDE FORM VALIDATION

SOME PHP FUNCTIONS TO HELP WITH VALIDATION

- `isset()`
- `empty()`
- `filter_input()`
- `filter_var()`
- `strlen()`
- `htmlentities()`
- `preg_match()`

EXAMPLE OF EMPTY()

```php
$name = $_POST['fname'];

if (empty($name)) {
    echo "Name is empty";
} else {
    echo $name;
}
```

If **$name** is **""**, an empty **array()**, **null**, **false**, or is not set it would be considered empty.

EXAMPLE OF FILTLER_INPUT()

```
if (!filter_input(INPUT_GET, "email",
FILTER_VALIDATE_EMAIL)) {
    echo("Email is not valid");
} else {
    echo("Email is valid");
}
```

Used to validate variables from insecure sources, such as user input from forms. You could also use **INPUT_POST** and also some different predefined filters.

EXAMPLE OF FILTER_VAR()

```php
$myEmail = "firstname.lastname@mymona.uwi.edu";
if (!filter_var($myEmail, FILTER_VALIDATE_EMAIL)) {
    echo("Email is not valid");
} else {
    echo("Email is valid");
}
```

Similar to **`filter_input`**, but instead filters a variable.

EXAMPLE OF HTMLENTITIES()

```php
$str = "A 'quote' is <b>bold</b>";

echo htmlentities($str);
// Outputs: A 'quote' is &lt;b&gt;bold&lt;/b&gt;

echo htmlentities($str, ENT_QUOTES);
// Outputs: A &#039;quote&#039; is &lt;b&gt;bold&lt;/b&gt;
```

This function will convert special characters to their HTML entities.

EXAMPLE OF PREG_MATCH

```php
$telephone = "876-999-1234";

if (!preg_match("/^\d{3}-\d{3}-\d{4}$/", $telephone)) {
    echo 'That telephone number is not valid!';
}
```

Performs a regular expression match

RESOURCES

- jQuery - http://jquery.com/
- Learn jQuery - https://learn.jquery.com/
- AJAX - https://developer.mozilla.org/en-US/docs/AJAX
- jQuery AJAX method - http://api.jquery.com/jquery.ajax/
- Using the Fetch API - https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch
- Using Promises - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Using_promises
- Async/Await - https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Asynchronous/Async_await

RESOURCES

- Regular Expressions - http://regexr.com/
- Form Data Validation - https://developer.mozilla.org/en-US/docs/Learn/HTML/Forms/Form_validation
- jQuery Validation Plugin - https://jqueryvalidation.org/
- PHP Docs - http://php.net/docs.php (to look up any php functions)