Events, callbacks and event loop

How events work

```
// Event handler
function sayHello(){
        document.getElementById("output").innerHTML = "Hello World";
}

// Target element to bind event
var elementObj = document.getElementById('myElement');

// Bind this way
elementObj.addEventListener("click", sayHello);

// Or better this way
elementObj.onclick = sayHello;
```

Mouse Events

`onclick`	user presses/releases mouse button on the element
`ondblclick`	user presses/releases mouse button twice on the element
`onmousedown`	user presses down mouse button on the element
`onmouseup`	user releases mouse button on the element movement
`onmouseover`	mouse cursor enters the element's box
`onmouseout`	mouse cursor exits the element's box
`onmousemove`	mouse cursor moves around within the element's box

The event object

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

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

`target`	The element on which the event handler was registered
`preventDefault()`	Prevents browser from performing its usual action in response to the event
`stopPropagation()`	Prevents the event from bubbling up further
`stopImmediatePropagation()`	Prevents the event from bubbling and prevents any other handlers from being executed

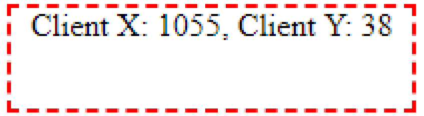
Stopping an event's browser behavior

To abort a form submit or another event's default browser behavior, call jQuery's **`preventDefault()`** method on the event

```
<form id="exampleform" action="">...</form>

$(function() {
        $("#exampleform").submit(validateData);
});

function validateData(event) {
        if ($("#city").val() == "" || $("#state").val().length != 2) {
                alert("Error, invalid city/state."); // show error message
                event.preventDefault();
        }
}
```

Using event object

```
<div style="width: 200px; height: 50px;
 border: dashed red 2px;
 margin: auto; text-align: center" id="output">
        Move over me!
</div>
```



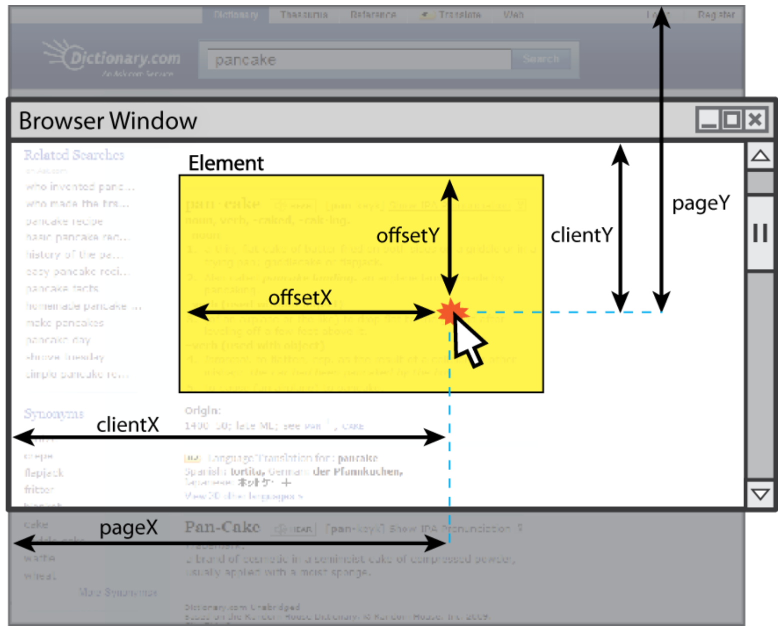
Client X: 1055, Client Y: 38

```
window.onload = function () {
    document.getElementById("output").onmousemove = displayMouseCords;
}

function displayMouseCords(e) {
    document.getElementById("output").innerHTML = "Client X: " + e.clientX +
        ", Client Y: " + e.clientY;
}
```

Mouse Event Object

The **event** object is passed to a mouse handler has these properties:

`clientX, clientY`	coordinates in browser window
`screenX, screenY`	coordinates in screen
`offsetX, offsetY`	coordinates in element (non-standard)
`pageX, pageY`	coordinates in entire web page
`which`	which mouse button was clicked



Page/window events

`onload`, `onunload`	the browser loads/exits the page
`onresize`	the browser window is resized
`onerror`	an error occurs when loading a document or an image
`oncontextmenu`	the user right-clicks to pop up a context menu

The above can be handled on the `window` object.

Form events

`onsubmit`	form is being submitted
`onreset`	form is being reset
`onchange`	the text or state of a form control has changed

Keyboard/text events

`onkeydown`	user presses a key while this element has keyboard focus
`onkeyup`	user releases a key while this element has keyboard focus
`onkeypress`	user presses and releases a key while this element has keyboard focus
`onfocus`	this element gains keyboard focus
`onblur`	this element loses keyboard focus
`onselect`	this element's text is selected or deselected)

Keyboard event object properties

`which`	ASCII integer value of key that was pressed (convert to char with `String.fromCharCode`)
`altKey`, `ctrlKey`, `shiftKey`	true if Alt/Ctrl/Shift key is being held

Event Bubbling

Which element gets the event?

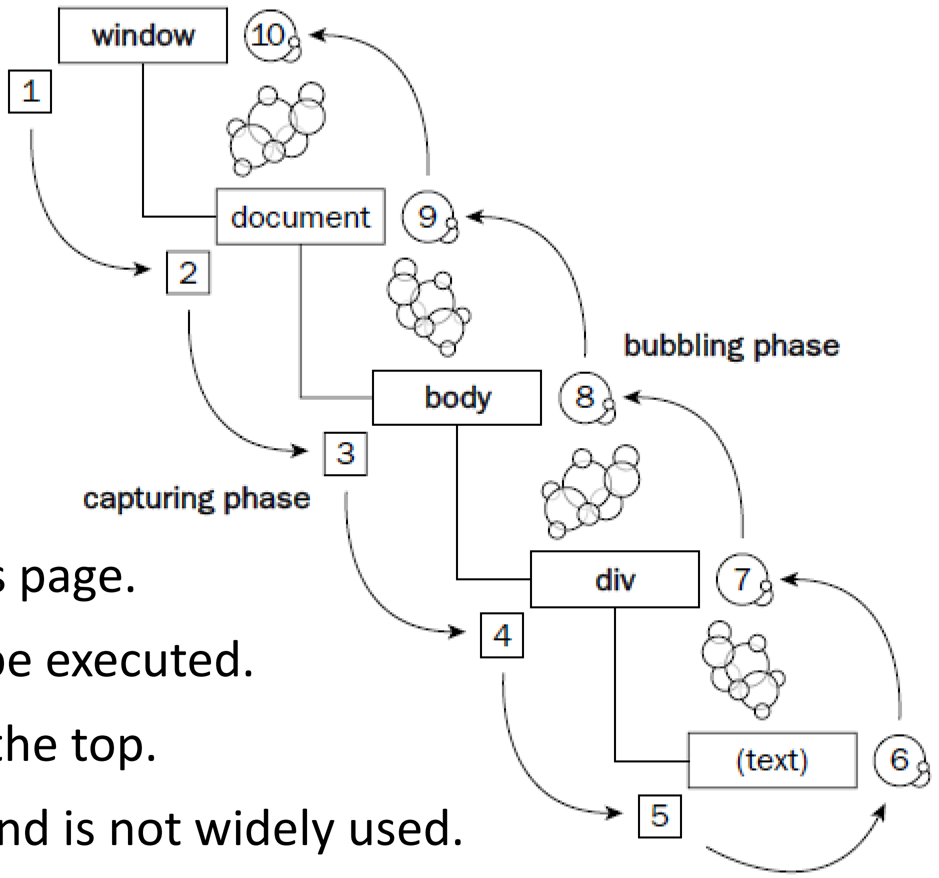
```
<body>
    <div>
        <p> Events are <em>crazy</em>! </p>
    </div>
</body>

$(function() {
       $("body, div, p, em").click(hello);
});

function hello() {
       alert("You clicked on the " + this.nodeName);
}
```

What happens when I click on the em? Which element will get the event?

Answer: All of them!

Bubbling vs Capturing

```
<body>
    <div>
        <p> Events are <em>crazy</em>! </p>
    </div>
</body>
```



- Clicking the em is actually a click on every element in this page.
- Therefore it was decided that all of the handlers should be executed.
- The events **bubble** from the bottom of the DOM tree to the top.
- The opposite model (top to bottom) is called **capturing** and is not widely used.

Stopping an event from bubbling

Use the **`stopPropagation()`** method of the jQuery event to stop it form bubbling up.

```
<body>
    <div>
        <p> Events are <em>crazy</em>! </p>
    </div>
</body>
```

```
$(function() {
       $("body, div, p, em").click(hello);
});

function hello(evt) {
       alert("You clicked on the " + this.nodeName);
       evt.stopPropagation();
}
```

Run Example

Multiple handlers

```
<body>
    <div>
        <p> Events are <em>crazy</em>! </p>
    </div>
    <p>Another paragraph!</p>
</body>
```

```
$(function() {
       $("body, div, p, em").click(hello);
       $("div > p").click(anotherHandler);
});

function hello() { alert("You clicked on the " + this.nodeName); }
function anotherHandler() { alert("You clicked on the inner P tag"); }
```

What happens when the first p tag is clicked?   Run Example

Stopping an event right now

- Use **`stopImmediatePropagation()`** to prevent any further handlers from being executed.
- Handlers of the same kind on the same element are otherwise executed in the order in which they were bound.

```
function anotherHandler(evt) {
	alert("You clicked on the inner P tag");
	evt.stopImmediatePropagation();
}
```

Run Example

stopImmediatePropogation()

```
$("div a").click(function() {
                      // Do something
                      });
$("div a").click(function(evt) {
                      // Do something else
                      evt.stopImmediatePropagation();
                      });
$("div a").click(function() {
                      // THIS NEVER FIRES
                      });
$("div a").click(function() {
                      // THIS NEVER FIRES
                      });
```

Only the first two handlers will ever run when the `anchor` tag is clicked.

jQuery handler return value

jQuery does something special if you return `false` in your event handler

1. prevents the default browser action, eg evt.preventDefault()
2. stops the event from bubbling, eg evt.stopPropagation()

```
<form id="myform"> ... <button type="submit">Done</button> </form>
```

```
$(function() {
        $("#myform").submit(cleanData);
       $("button").click(checkData);
});
function checkData() {
        if ($("#city").val() == "" || $("#state").val().length != 2) {
        alert("Error, invalid city/state."); // show error message
                return false;
        }
}
```

Main Point

Event handlers take callback functions that are executed later when the event occurs.

Science of Consciousness: Callbacks are a form of memory for an action that is automatically executed when an event happens. When we act from deep levels of awareness we are more likely to activate appropriate memories and reactions (event handlers).

JS Timers (review)

```
setTimeout(function, delayMS); // arranges to call given function after given delay in ms

setInterval(function, delayMS); // arranges to call function repeatedly every delayMS ms
```

Both **`setTimeout`** and **`setInterval`** return an ID representing the timer, this ID can be passed to **`clearTimeout(timerID)`** and **`clearInterval(timerID)`** to stop the given timer.

Note: If **`function`** has parameters: **`setTimeout(function, delayMS, param1, param2 ..etc);`**

Callbacks and Events Queue

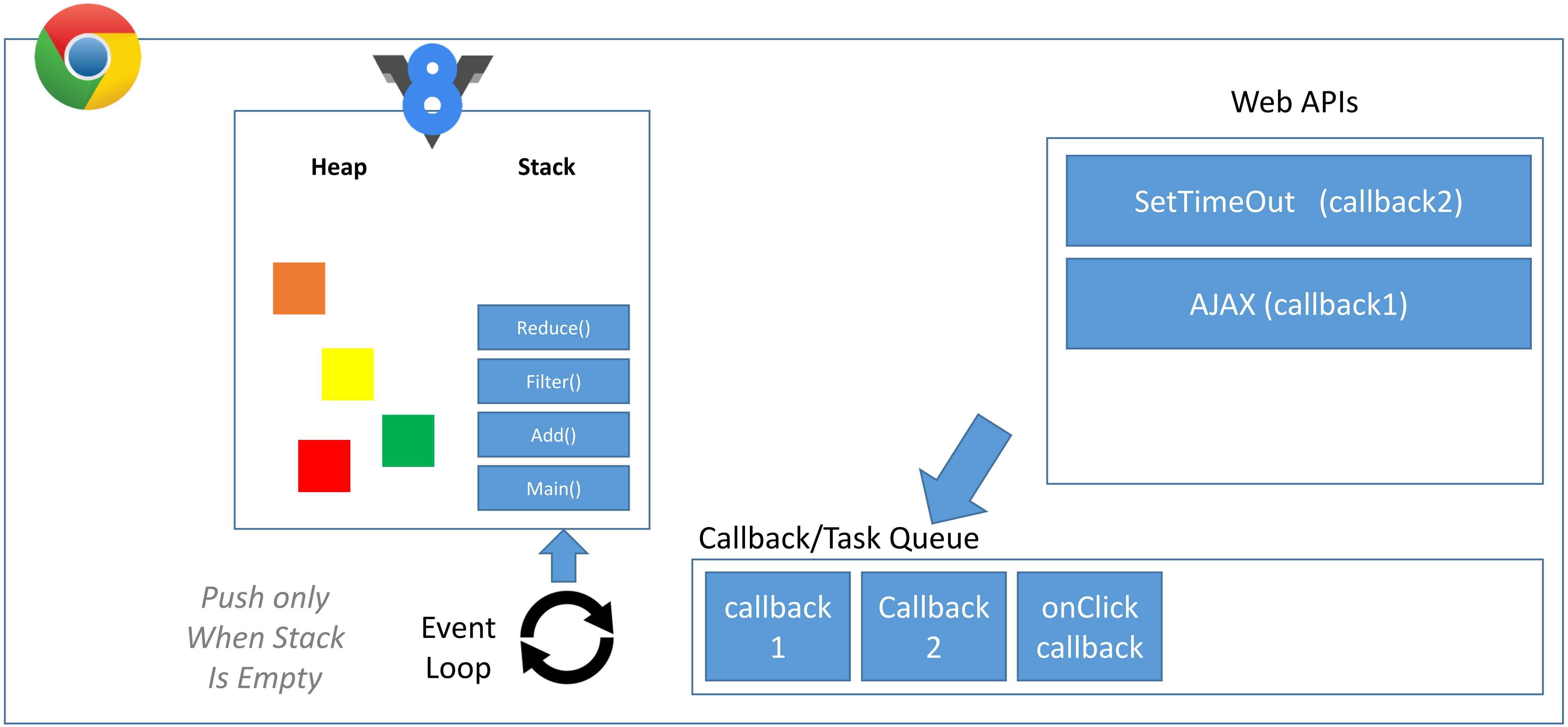
```
// In what order the results will be printed and why?

function logger(f){
  f();
}

console.log(1);
setTimeout(function(){ console.log(2); }, 1000);
logger(function(){console.log(3)});
setTimeout(function(){ console.log(4); }, 0);
console.log(5);
```

Callbacks & Asynchronous Callbacks

A callback function is a function you give to another function, to be invoked later when the other function is finished (desired).

All Callback functions may not execute immediately, instead they may be registered in the browser and run later when desired from the browser **Event/Task Queue**.

Chrome – Concurrency & the Event Loop



Main Point

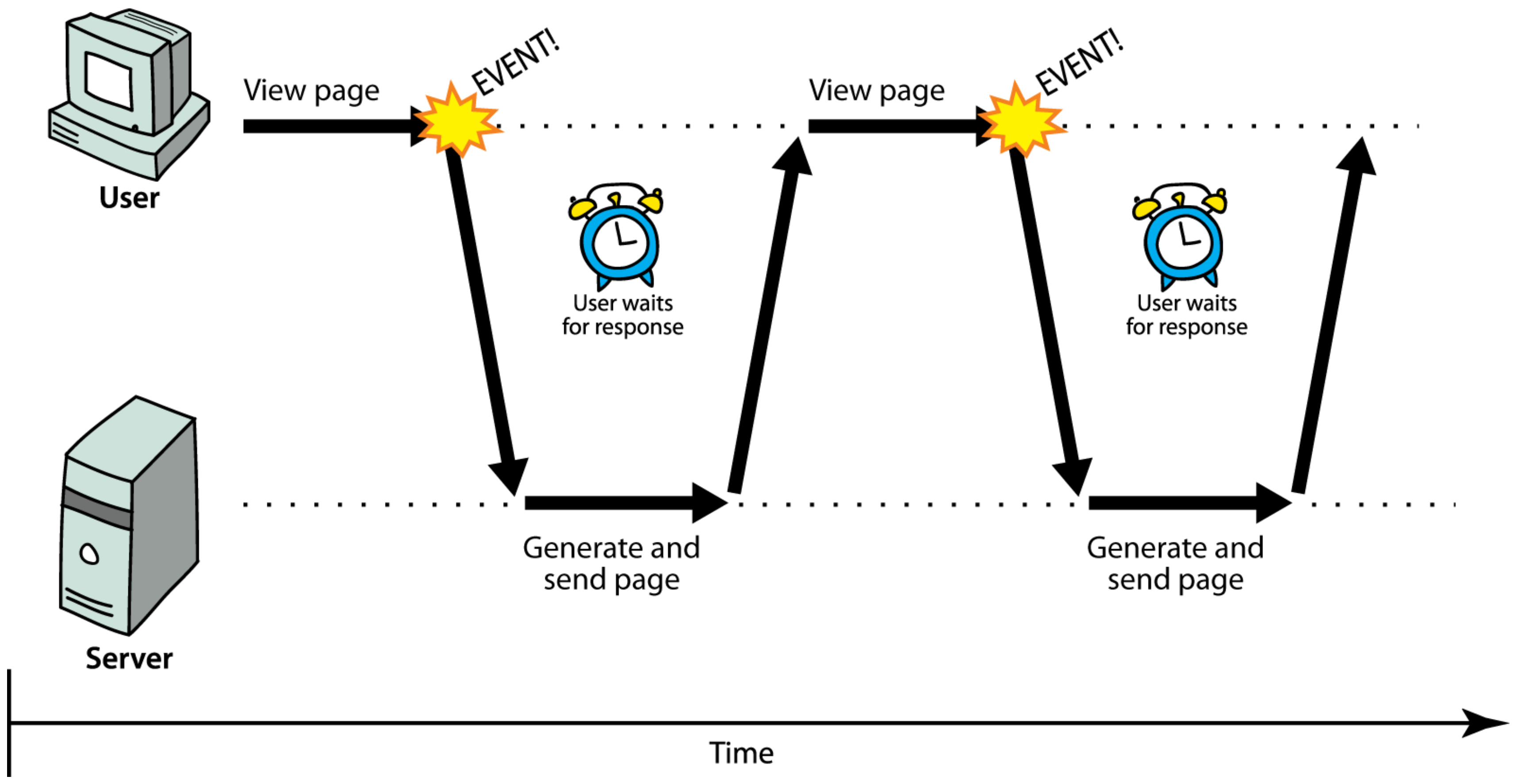
JavaScript is single threaded. It handles asynchronous events by storing them and cycling through them in an ‘event loop’.

Science of Consciousness: The event loop gives the appearance of multitasking even though there is only ever a single task and thread of execution. The universe appears to be infinitely diverse even though there is only a single unified field.

AJAX

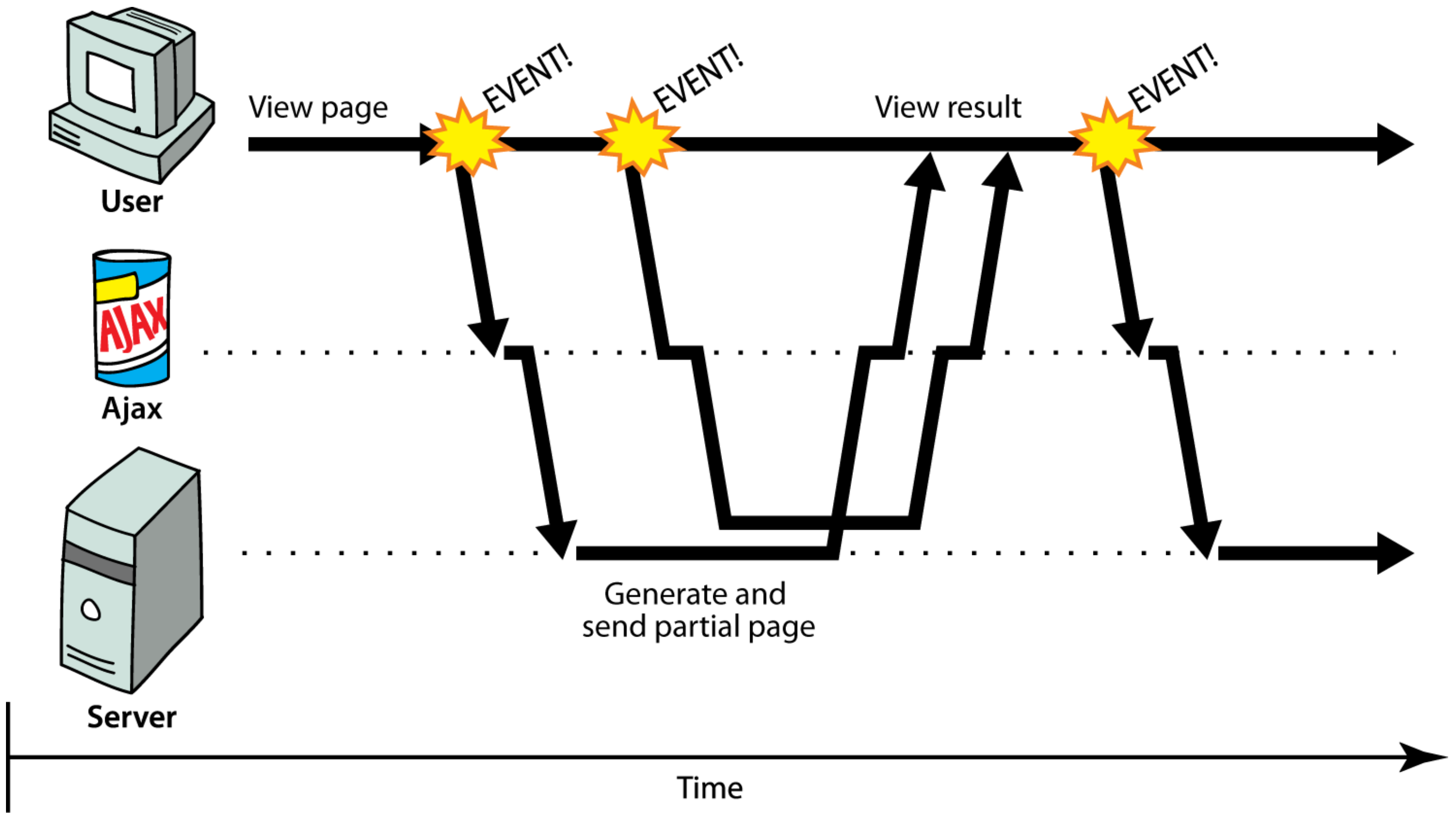
Synchronous web communication

Synchronous: user must wait while new pages load (click, wait, refresh)



Asynchronous web communication

Asynchronous: user can keep interacting with page while data loads



Web applications and Ajax

- **Web Application**: a dynamic web site that mimics the feel of a desktop app
  - Presents a continuous user experience rather than disjoint pages
  - Examples: Gmail, Google Maps, Google Docs and Spreadsheets
- **Ajax**: Asynchronous JavaScript and XML
  - Not a programming language; a particular way of using JavaScript
  - Downloads data from a server in the background
  - Allows dynamically updating a page without making the user wait
  - Avoids the "click-wait-refresh" pattern
  - Example: Google Suggest

XMLHttpRequest

- JavaScript includes an **`XMLHttpRequest`** object that can make requests to a web server
- Supported in IE5+, Safari, Firefox, Opera, Chrome, etc. (with minor incompatibilities)
- It can do this asynchronously (in the background, transparent to user)
- The contents of the fetched file can be put into current web page using the DOM
- Sounds great!...

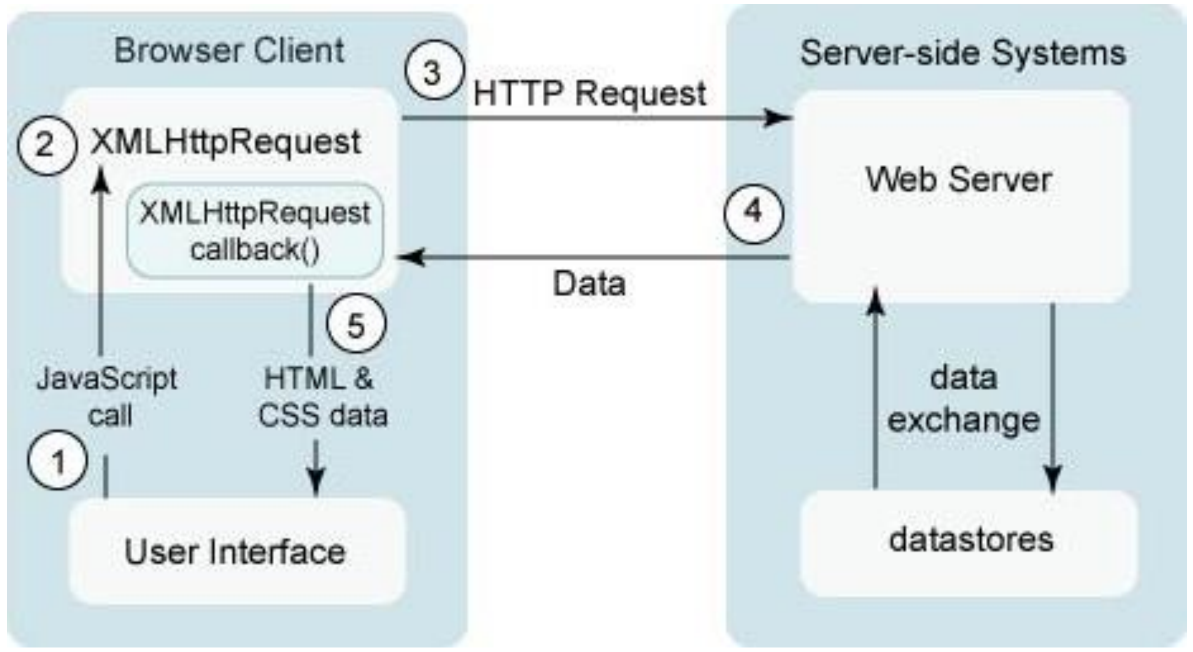
XMLHttpRequest *(and why we won't use it)*

- It is clunky to use, and has various browser incompatibilities
  - jQuery provides a better wrapper for Ajax, so we will use that instead

- New Fetch API provides a more powerful and flexible feature set.

A typical Ajax request

1. User clicks, invoking an event handler
2. Handler's code creates an XMLHttpRequest object
3. XMLHttpRequest object makes partial request to the server and wait.
4. Server responds (success or fail)
5. XMLHttpRequest fires callback associated with the event.
6. Your callback (event handler function) processes the data or error and partially refreshes the view.



Main Point

The key component that a browser provides to enable Ajax is the `XMLHttpRequest` object, which is supported by all modern browsers. This object opens a connection with a server, sends a message, waits for the response, and then activates a given callback method. **Science of Consciousness:** The TM Technique is supported by any human nervous system. It allows us to connect with the source of thought, experience restful alertness, and then return to activity with that influence of calm alertness.

JavaScript Object Notation (JSON)

JavaScript Object Notation (JSON): Data format that represents data as a set of JavaScript objects

- Natively supported by all modern browsers (and libraries to support it in old ones)
- Becoming more popular than XML due to its simplicity and ease of use

http://www.json.org/

XML vs JSON

```xml
<?xml version="1.0" encoding="UTF-8"?>
<note private="true">
    <from>Alice Smith (alice@example.com)</from>
    <to>Robert Jones (roberto@example.com)</to>
    <to>Charles Dodd (cdodd@example.com)</to>
    <subject>Tomorrow's "Birthday Bash" event!</subject>
    <message language="english">
        Hey guys, don't forget to call me this weekend!
    </message>
</note>
```

```json
{
  "private": "true",
  "from": "Alice Smith (alice@example.com)",
  "to": [
    "Robert Jones (roberto@example.com)",
    "Charles Dodd (cdodd@example.com)"
  ],
  "subject": "Tomorrow's \"Birthday Bash\" event!",
  "message": {
    "language": "english",
    "text": "Hey guys, don't forget to call me this weekend!"
  }
}
```

JavaScript Object Notation (JSON)

JSON is a syntax for storing and exchanging data and an efficient alternative to XML

```
{"employees":[
    {"firstName":"John", "lastName":"Doe"},
    {"firstName":"Anna", "lastName":"Smith"},
    {"firstName":"Peter", "lastName":"Jones"}
]}
```

A name/value pair consists of a field name **(in double quotes)**, followed by a colon, followed by a value.

JSON values can be:

- A number (integer or floating point)
- A string (in double quotes)
- A Boolean (true or false)
- An array (in square brackets)
- An object (in curly braces)
- null

Browser JSON methods

- You can use Ajax to fetch data that is in JSON format
- Then call `JSON.parse` on it to convert it into an object
- Then interact with that object as you would with any other JavaScript object

method	description
`JSON.parse(`*`string`*`)`	converts the given string of JSON data into an equivalent JavaScript object and returns it
`JSON.stringify(`*`object`*`)`	converts the given object into a string of JSON data (the opposite of JSON.parse)

JSON expressions exercise

Given the JSON data at right, what expressions would produce:

- The window's title?

  ```
  var title = data.window.title;
  ```
- The image's third coordinate?

  ```
  var coord = data.image.coords[2];
  ```
- The number of messages?

  ```
  var len = data.messages.length;
  ```
- The y-offset of the last message?

  ```
  var y = data.messages[len - 1].offset[1];
  ```

```
var jsonString = '{
  "window": {
    "title": "Sample Widget",
    "width": 500,
    "height": 500
  },
  "image": {
    "src": "images/logo.png",
    "coords": [250, 150, 350, 400],
    "alignment": "center"
  },
  "messages": [
    {"text": "Save", "offset": [10, 30]}
    {"text": "Help", "offset": [ 0, 50]},
    {"text": "Quit", "offset": [30, 10]},
  ],
  "debug": "true"
}';
var data = JSON.parse(jsonString);
```

Main Point

JSON has become more widely used for Ajax data representations than XML because JSON is easier to write and read and is almost identical to JavaScript object literal syntax. **Science of Consciousness:** We always prefer to do less and accomplish more. Actions arising from deep levels of consciousness are more efficient and effective.

jQuery's ajax method

- Call the `$.ajax()` method
- Constructor accepts an object literal full of options that dictate the behavior of the AJAX request:
  - The url to fetch, as a String,
  - The type of the request, GET or POST.. etc
- Hides icky details of the raw `XMLHttpRequest`; works well in all browsers

```
$.ajax({
  "url": "http://foo.com",
  "option" : "value",
  "option" : "value",
  ...
  "option" : "value"
});
```

$.ajax() options

option	description
url	The URL to make a request from
type	whether to use POST or GET
data	an object literal filled with query parameters and their values
dataType	The type of data you are expecting to receive, one of: "text", "html", "json", "xml"
timeout	an amount of time in seconds to wait for the server before giving up
success	*event:* called when the request finishes successfully
error	*event:* called when the request fails
complete	*event:* called when the request finishes successfully or erroneously

jQuery AJAX example

```
$.ajax({
        "url": "foo/bar/mydata.txt",
        "type": "GET",
        "success": myAjaxSuccessFunction,
        "error": ajaxFailure
});

function myAjaxSuccessFunction(data) {
        // do something with the data
}
function ajaxFailure(xhr, status, exception) {
         console.log(xhr, status, exception);
}
```

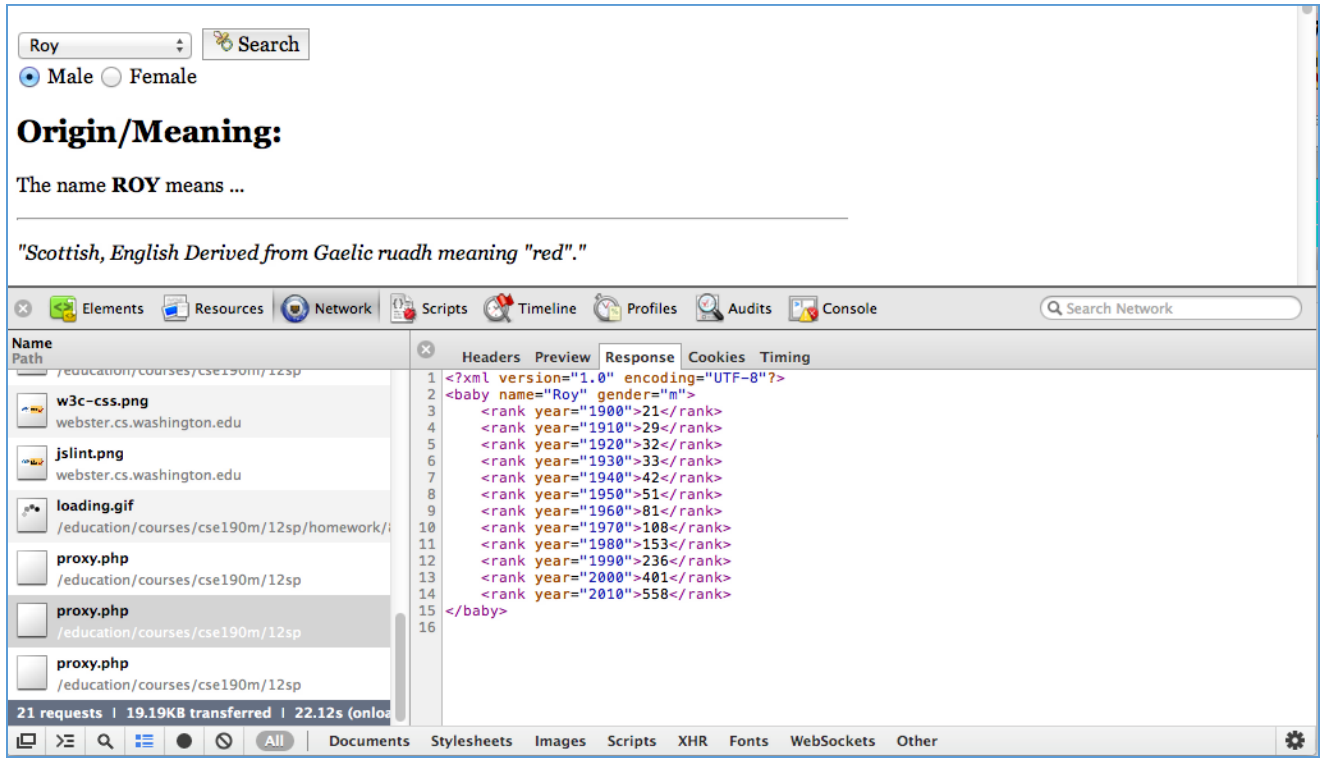
Examples – Check in DevTools!

Adding AJAX code to load in a hw output file into the textarea (homework page)

Another example: The quotes page

Debugging AJAX code

- Chrome DevTool's **Network** tab shows each request, parameters, response, errors
- expand a request by clicking on it and look at **Response** tab to see Ajax result
- check the **Console** tab for any errors that are thrown by requests



Roy

[✓ Male] Female

Origin/Meaning:

The name **ROY** means ...

"Scottish, English Derived from Gaelic ruadh meaning "red"."

Name / Path
/education/courses/cse190m/12sp
w3c-css.png webster.cs.washington.edu
jslint.png webster.cs.washington.edu
loading.gif /education/courses/cse190m/12sp/homework/
proxy.php /education/courses/cse190m/12sp
proxy.php /education/courses/cse190m/12sp
proxy.php /education/courses/cse190m/12sp

21 requests | 19.19KB transferred | 22.12s (onlo

```
<?xml version="1.0" encoding="UTF-8"?>
<baby name="Roy" gender="m">
    <rank year="1900">21</rank>
    <rank year="1910">29</rank>
    <rank year="1920">32</rank>
    <rank year="1930">33</rank>
    <rank year="1940">42</rank>
    <rank year="1950">51</rank>
    <rank year="1960">81</rank>
    <rank year="1970">108</rank>
    <rank year="1980">153</rank>
    <rank year="1990">236</rank>
    <rank year="2000">401</rank>
    <rank year="2010">558</rank>
</baby>
```

Better jQuery AJAX

Using these event handler function calls done() and fail() instead

```
$.ajax("http://foo.com", { "type": "GET" })
        .done(functionName)
        .fail(ajaxFailure);
```

Passing query parameters to a request

```
$.ajax("lookup_account.php", { "type": "GET",
                                  "data": { "name": "David Lynch",
                                            "height": 180,
                                            "password": "abcdef" },
                                }) .done(function)
                                   .fail(function);
```

- Don't concatenate the parameters onto the URL yourself with "?" + ...
  - won't properly URL-encode the parameters
  - won't work for POST requests
- Query parameters are passed as a data object literal

  (the above is equivalent to: "name=David+Lynch&height=180&password=abcdef")

Creating a POST request

`type` should be changed to "`POST`" (`GET` is default)

```
$.ajax("url", {
        "type": "POST",
        "data": {
                "name": value,
                "name": value,
                    ...,
                "name": value
        }
}) .done(function)
   .fail(function);
```

`$.get()` and `$.post()` shortcuts

Often you don't need the flexibility of `$.ajax()` function

- The options are hard to remember
- You don't usually need all of the options

These shortcut functions are preferred when additional options are not needed.

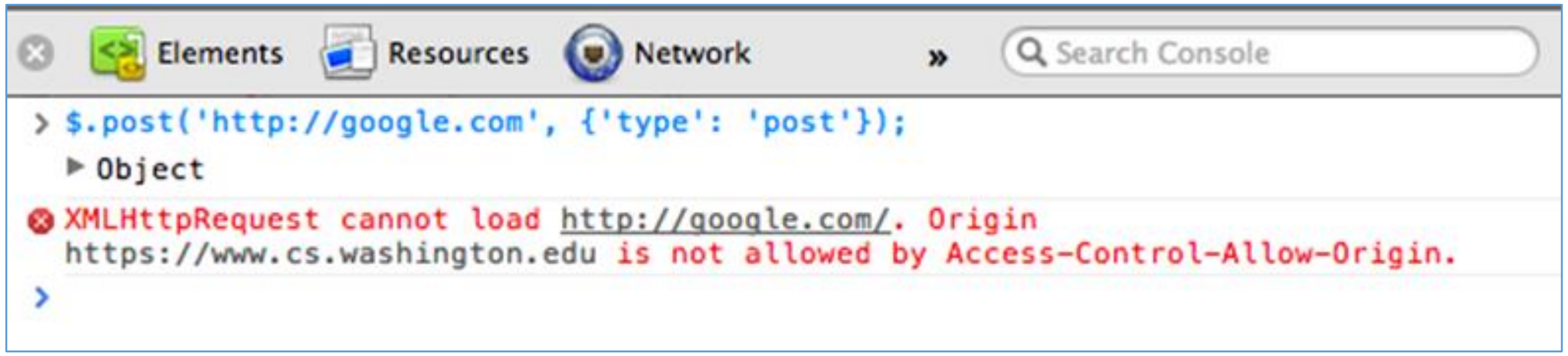
```
$.get(url, {data})
        .done(function)
        .fail(function);

$.post(url, {data})
        .done(function)
        .fail(function);
```

Main Points

- The browser console window is critical to debugging Ajax code since it allows the developer to easily see the request that was sent and the response that was returned. **Science of Consciousness:** TM Checking is critical to ensure that we are practicing our TM Technique easily and effortlessly.

`XMLHttpRequest` security restrictions

- The same-origin policy restricts how a document or script loaded from one origin can interact with the resource from another origin.
- It is a critical security mechanism for isolating potentially malicious documents.
  - Origin: combination of protocol, host and port



```
> $.post('http://google.com', {'type': 'post'});
  ▶ Object
⊗ XMLHttpRequest cannot load http://google.com/. Origin
  https://www.cs.washington.edu is not allowed by Access-Control-Allow-Origin.
>
```

Main Points

The same origin policy is a security constraint on browsers that restricts scripts to only contact a site with the same domain name, application protocol, and port. This means that browsers only allow Ajax calls to the same web server from which the page originated. **Science of Consciousness:** If we are calm and alert then we are more secure from disruptions by external distractions or deceptions.

AJAX user feedback

- Ajax are silent calls, no visual action to users.
- Users don't like unresponsiveness
- Often you show some sort of loader image or text while the request is made

User feedback with `always()`

The general technique is to `show()` some feedback when the AJAX request starts and `hide()` it again once it finishes.

- The `always()` function is an event that the AJAX request fires every time the request finishes, whether it was successful or not
- This might be some typical user feedback code (example)

Async

```
$("#mybutton").click(function () {
        $.get("delay.php")
            .done(display)
            .fail(ajaxFailure)
            .always(function () {
                $("#loader").hide();
            });

        $("#loader").show();
    });
```

AJAX using Fetch API

- The Fetch API provides global `fetch()` method that provides an easy, logical way to fetch resources asynchronously across the network.
- Fetch provides a better alternative to XMLHttpRequest.
- The fetch spec differs from `jQuery.ajax()`
  - The Promise returned from `fetch()` won't reject on HTTP error status even if the response is an HTTP 404 or 500. Instead, it will resolve normally (with ok status set to false), and it will reject failure only if anything prevented the request from completing.

Basic fetch request

```
fetch('http://example.com/movies.json')
    .then(function(response) {
        return response.json(); // response.json() returns Promise as well
    })
    .then(function(myJson) {
        console.log(JSON.stringify(myJson));
    });
```

Promise

- A Promise is an object representing the *eventual* completion or failure of an asynchronous operation.
- Most of the time in our application, we consume promises returned from calling some other APIs like `fetch()` or `json()`
  - But, you can easily create and return Promise from your own APIs.

Demo: Promise Constructor

```
let promise1 = new Promise(function (resolve, reject) {
    let r = Math.ceil(Math.random() * 10);
    if (r % 2 === 0) {
        setTimeout(function () {
            resolve('even');
        }, 300);
    } else {
        setTimeout(function () {
            reject('odd');
        }, 300);
    }
});

promise1
    .then(function (value) {
        console.log("Success: "+value);
    })
    .catch(function (err) {
        console.log("Error: "+ err);
    });

console.log(promise1); // this line will execute before promise resolves or rejects
```

Promise Constructor

- Syntax

```
new Promise(executor);
```

  - Executer is a function that is passed with the arguments `resolve` and `reject`.
  - The `executer` function is executed immediately by the `Promise` implementation, passing `resolve` and `reject` functions.
  - The resolve and reject functions, when called, `resolve` or `reject` the promise, respectively.
  - The executer normally initiates some asynchronous work, and then, once that completes, either call the `resolve` function to resolve the promise or else rejects it if an error occurred.

Description

- A `Promise` is a proxy for a value not necessarily known when the promise is created.
    - It allows you to associate handlers with an asynchronous action's eventual success value or failure reason.
    - This lets asynchronous methods return values like synchronous methods: instead of immediately returning the final value, the asynchronous method returns a promise to *supply* the value at some point in future.

Promise states

- A promise is in one of these states:
  - *pending:* initial state, neither fulfilled nor rejected.
  - *fulfilled*: meaning that the operation completed successfully.
  - *rejected*: meaning that the operation failed.

Fetch with POST data

```
let url = 'https://example.com/profile';
let data = {username: 'example'};

fetch(url, {
    method: 'POST', // or 'PUT'
    body: JSON.stringify(data),
    headers:{
        'Content-Type': 'application/json'
    }
}).then(res => res.json())
    .then(response => console.log('Success:', JSON.stringify(response)))
    .catch(error => console.error('Error:', error));
```

AJAX summary

- So, what makes AJAX, well… AJAX?
    - Asynchronous request.
    - Partial page rendering.

Exercise: Parsing JSON

Suppose we have a service http://jsonplaceholder.typicode.com about blogs.

- Write a page that processes this JSON blog data.
  - To display user information /users/1
  - Display all posts from selected user /posts?userId=1
  - Display all comments from selected post /comments?postId=1

- Create one page with input form to take userId from the browser
- Display user name and email and address and all posts belongs to this userId
- For every post, you need to show a button (show comments) once clicked you need to show all comments for the specific post.

CONNECTING THE PARTS OF KNOWLEDGE WITH THE WHOLENESS OF KNOWLEDGE

Ajax and Advanced techniques

1. Client side programming with JavaScript is useful for making web applications highly responsive.
2. Ajax allows JavaScript to access the server in a very efficient manner using asynchronous messaging and partial page refreshing.

3. **Transcendental consciousness** is the experience of the home of all the laws of nature where all information is available at every point.

4. **Impulses within the transcendental field**: Communication at this level is instantaneous and effortless.

5. **Wholeness moving within itself**: In unity consciousness daily life is experienced in terms of this frictionless and effortless flow of information.

References

- https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Using_promises
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise