


Synchronous js

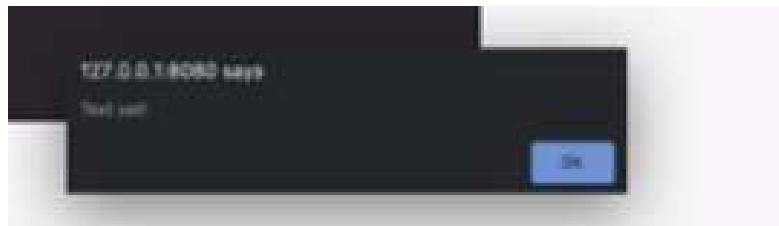
Synchronous code is executed line by line

Each line of code waits for previous line of code to finish




```
const p = document.querySelector('.p');
p.textContent = 'My name is Jonas!';
alert('Text set!');
p.style.color = 'red';
```

When we hit alert('Text set') line of code then we got a popup message .this alert window will block the code execution, right? So nothing will happen on the page until we click that OK Button.



the first line of code is still synchronous here, and we also move to the second line in a synchronous way. But here we encountered the set timeout function, which will basically start a timer in an asynchronous way. So this means that the timer will essentially run in the background without preventing the main code from executing. We also register a callback function, which will not be executed now, but only after the timer has finished running. And we have actually already done this many times before in practice, right? Now this callback function that I just mentioned is asynchronous JavaScript. And it is asynchronous because it's only going to be executed after a task that is running in the background finishes execution.

And in this case, that is the timer.



```
const p = document.querySelector('.p');
setTimeout(function () {
  p.textContent = 'My name Jonas!';
}, 5000);
p.style.color = 'red';
```

Example: Timer with callback

- 👉 Asynchronous code is executed **after** a task that runs in the “background” finishes
- 👉 Asynchronous code is **non-blocking**;
- 👉 Execution doesn't wait for an asynchronous task to finish its work;

But please understand this very important fact that callback functions alone do not make code asynchronous,

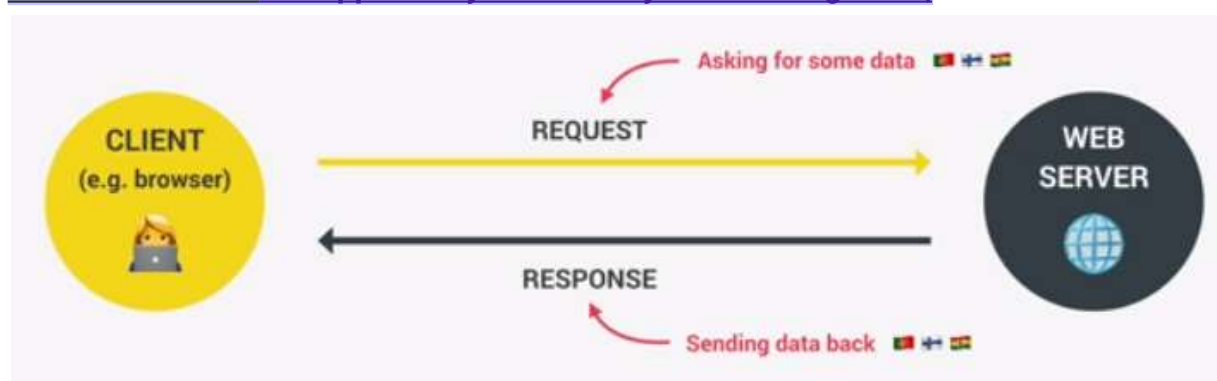
event listeners alone do not make code asynchronous, just like callback functions alone, do also not make code asynchronous.

For example, an event listener listening for a click on a button is not doing any work in the background. It's simply waiting for a click to happen,

And Ajax calls are probably the most important use case of asynchronous JavaScript.



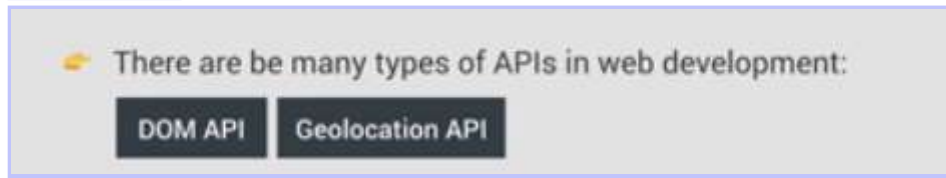
For now, let's quickly understand how Ajax works. So let's say that we have our JavaScript application running in the browser, which is also called the Client. And we want the application to get some data from a web server. And let's say the data about countries that I was talking about earlier. So with Ajax, we can do an HTTP request to the server, which has this data. And the server will then set back a response containing that data that we requested. And this back and forth between Client and server **all happens asynchronously in the background.**



Web Server usually contains web API And this API is the one that has the data [that we're asking for.](#)

API

an API is basically a piece of software that can be used by another piece of software in order to basically allow applications to talk to each other [and exchange information.](#)

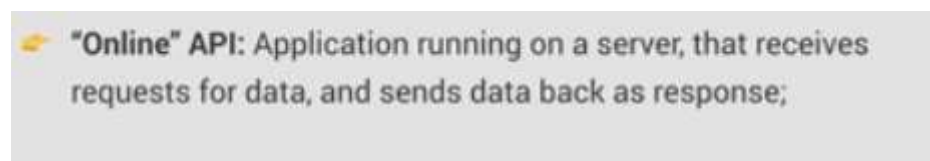


we can always implement a small and simple API in a class where we make some methods available as a public interface.

when we use Ajax.

And that are APIs

that I like to call Online APIs.



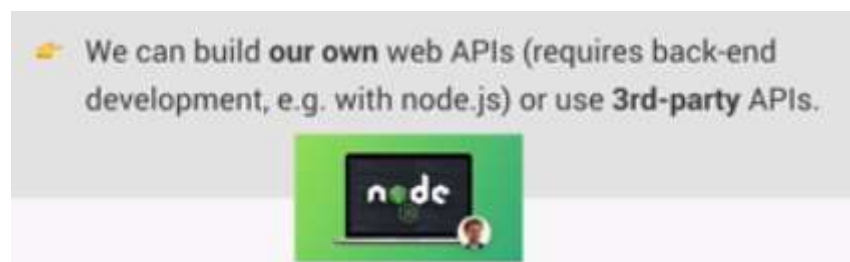
So an online API

is essentially an application running on a web server,

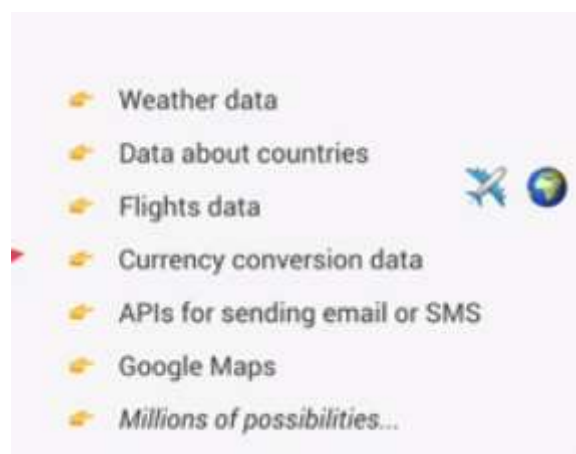
which receives requests for data,

then retrieves this data from some database

and then sends it back to the client.



there are really APIs for everything.



So Ajax stands for asynchronous JavaScript and XML.
Remember?

So the X there stands for XML
and XML is a data format,
which used to be widely used
to transmit data on the web.



So instead, most APIs these days
use the JSON data format.
So JSON is the most popular data format today
because it's basically just a JavaScript object,
but converted to a string.
And so therefore,
it's very easy to send across the web
and also to use in JavaScript
once the data arrives.

