



Errors, Storages

October 2022

Agenda

1 INTRO

2 ERROR OBJECT

3 THROW AN EXCEPTION

4 TRY, CATCH, FINALLY

5 COOKIES

6 LOCAL AND SESSION STORAGES

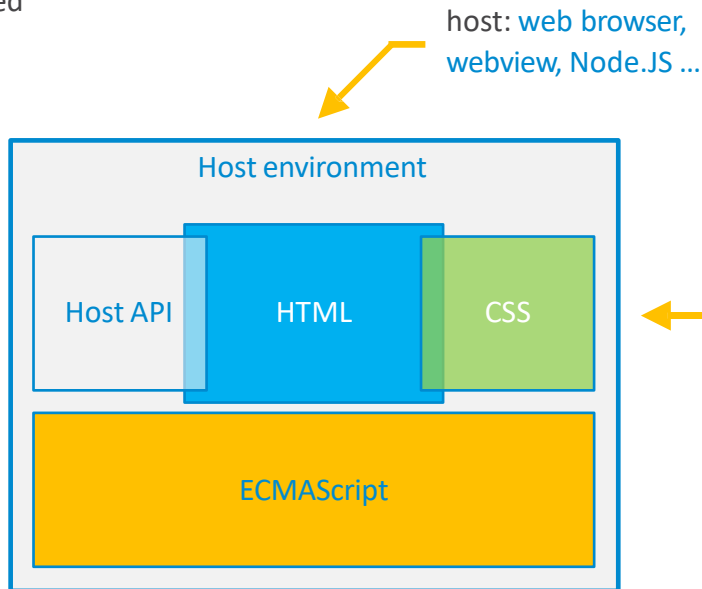


Introduction

The web uses different technologies and standards. We could be already familiar with [HTML](#), [CSS](#) and [JavaScript](#) (these are all well defined standards).

However, these are just really the top of the iceberg, and boundaries between these are not always clear.

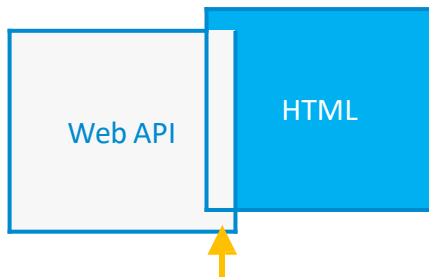
Console API is part of the browser's [Web API](#)



Core JavaScript vs Web API

Usually, we refer JavaScript as a code, written in the JavaScript language. However, there are parts provided by the standard (core JavaScript), yet others are added by the host environment.

In this lecture, we will depart from the core JS and learn about an important parts of the [Web API: Cookies](#) and [Web storage](#). Also, we will get familiar with the [built-in Error](#) object of the core JavaScript.



Storage is a [Web API](#), however, also a part of the [HTML Standard](#) as well.

[Error](#) is a built-in object in JavaScript, such as Array, Number, Boolean, etc.

[setTimeout](#)*, however, is not part of the JS core, it is added by the browser.

```
> setTimeout(function() {  
    throw new Error("A déjà vu is usually a glitch in the Matrix.");  
}, 0);  
< 42
```

✖ ▶ Uncaught Error: A déjà vu is usually a glitch in the Matrix. [VM7632:2](#)
at <anonymous>:2:11

* as you may guessed, there is a `setTimeout` in Node.JS, slightly different, though

Host API

It is easier to reason about the host APIs, if we imagine that the browser declares these before running our code, like this:

→

```
window.setTimeout = function() { ... }  
window.localStorage = { ... }  
window.sessionStorage = { ... }  
window.console = { ... };  
...  
  
// here comes your code
```

This will be very important when you try to write unit tests, because those are running in Node.JS, therefore the browser API needs to be [emulated](#) to be able to understand your code.

```
➤ global.setTimeout  
[Function: setTimeout] {  
  [Symbol(nodejs.util.promisify.custom)]: [Function]  
}  
➤ global.sessionStorage  
undefined  
➤
```

in Node.JS, setTimeout added to the [global](#) object, but there is no support for the storage API

ERROR OBJECT

Error object

Error is an object

As you may already get used to it, important parts of the core JavaScript can be accessed via built-in objects. [Error](#) is one of them.

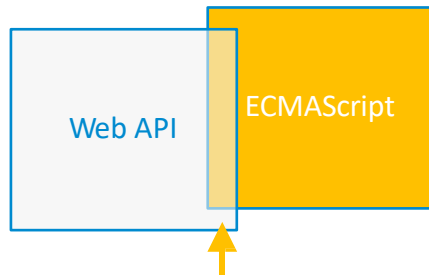
When you see an error in a console, that is essentially the result of:

- creating a new error object instance
- throwing that
- (and not catching the error)

```
> let neo = {  
  contacts: {  
    rhineheart: {},  
  }  
}  
  
// later...  
  
neo.contacts.morpheus.talksTo();
```

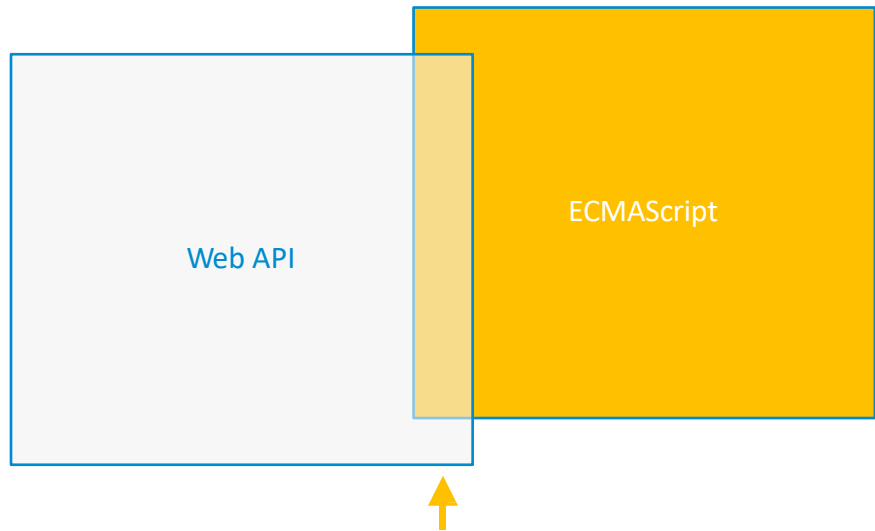
✖ ▶ Uncaught TypeError: Cannot read property 'talksTo' of undefined
at <anonymous>:9:23

nah, not yet...



While Error is a built-in object, defined by the ECMA Standard, implementations are slightly in different browsers.

Browser compatibility



While Error is a built-in object, defined by the ECMA Standard, implementations are slightly different in browsers.

And differences in browser implementations are always an infinite source of lot of fun nightmare in web development.

Error types

There are several types of **predefined error objects** exist, and a custom error could be created, as well.

RangeError

a numeric variable or parameter is outside of its valid range

ReferenceError

referencing an invalid reference

SyntaxError

represents a syntax error

TypeError

a variable or parameter is not of a valid type

URIError

encodeURI() or decodeURI() are passed invalid parameters

AggregateError

represents several errors wrapped in a single error when multiple errors need to be reported by an operation (e.g., Promise.any()).

Examples

> Array(-1)

✗ ▶ Uncaught RangeError: Invalid array length
at <anonymous>:1:1

> equilibrium++;

✗ ▶ Uncaught ReferenceError: equilibrium is not defined
at <anonymous>:1:1

> Wake up Samurai, the Matrix is everywhere. It is all around us.

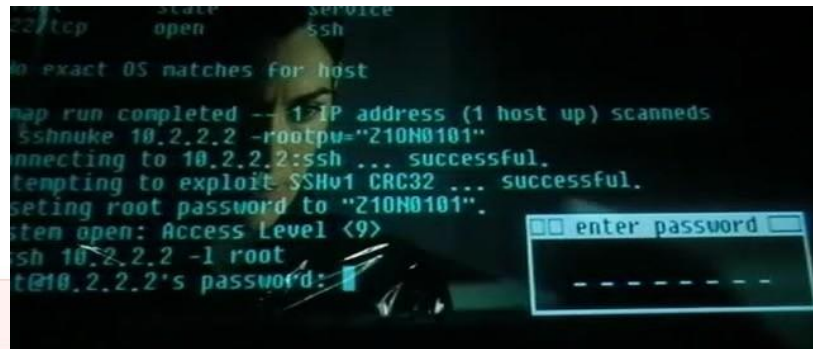
✗ Uncaught SyntaxError: Unexpected identifier

> ({ choice: { is: { an: "illusion" } } }).choice.is.not.illusion;

✗ ▶ Uncaught TypeError: Cannot read property 'illusion' of undefined
at <anonymous>:1:56

> decodeURIComponent('%root#10.2.2.2');

✗ ▶ Uncaught URIError: URI malformed
at decodeURIComponent (<anonymous>)
at <anonymous>:1:1



THROW AN EXCEPTION

How to create ~~bugs~~ errors

Error occurs in predefined cases, however, errors can be triggered programmatically, too.

All you need is to create a [new instance](#) of your choice of error objects:

the [error constructors](#) return the error object even when called without *new*, so it can be used both ways



```
> let pillColor = "yellow";  
    switch (pillColor) {  
      case "red":  
        message = "you wake up in your bed and believe whatever you want to believe.";  
        break;  
      case "blue":  
        message = "you stay in Wonderland and I show you how deep the rabbit hole goes.";  
        break;  
      default:  
        throw new RangeError("Remember, all I'm offering is the truth, nothing more");  
    }
```

✖ ▶ Uncaught RangeError: Remember, all I'm offering is the truth, nothing more
at <anonymous>:11:15

Throwing an exception

That being said, we can **throw an exception without an error object** as well.

*Execution will stop (the statements after throw won't be executed), and control will be passed to the first catch block in the call stack. If **no catch block** exists among caller functions, the program **will terminate**.*

```
throw {  
  name: "ChickenError",  
  message: "I can't do this"  
}
```

```
> (function goToTheScaffold() {  
  let whiteRabbit = "\u{0001F407}";  
  throw "I can't do this";  
  console.log(whiteRabbit);  
})();
```

this **will not** run →

nor this →

no rabbit, no real world :/ →

```
console.log("Welcome to the real world");
```

```
✖ ▶ Uncaught I can't do this
```

the throw value can be any expression

let's have a catch block then...

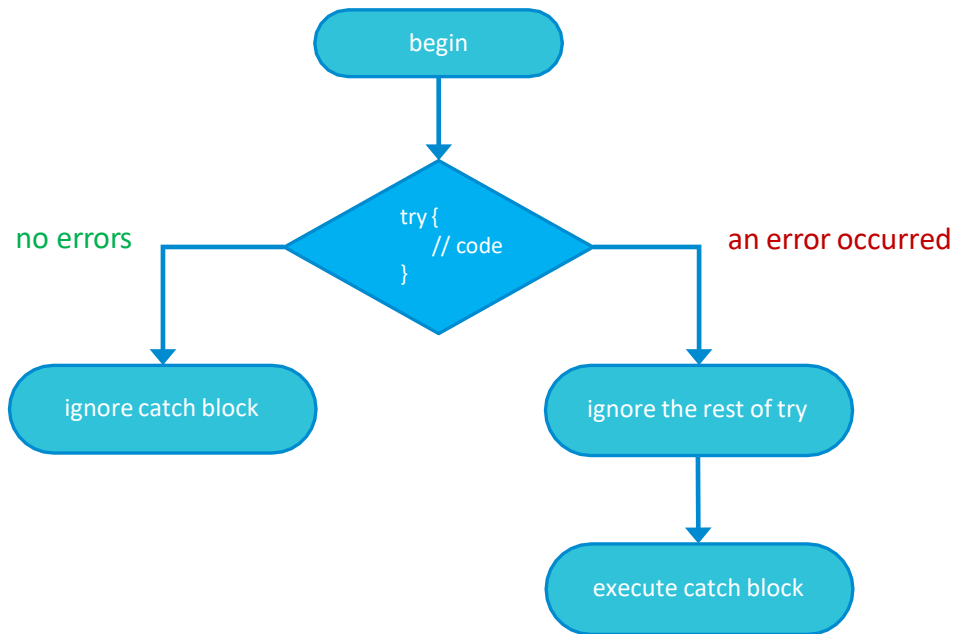
TRY, CATCH, FINALLY

Try ... Catch

Throwing errors is just the half of the story

An uncaught error leads to less user-friendly error message in the console, and generally is a sign that the developer team left some loose ends.

The mechanism for handling errors is the try-catch.



[MDN: try...catch](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/try...catch)

Try ... Catch

The `try ... catch` block allows to "catch" errors, and instead of terminating, do something more reasonable.

we have an exception here →

... so this won't run →

but we handle it →

... so we won't miss the important part →

```
> try {
  (function goToTheScaffold() {
    let whiteRabbit = "\u{0001F407}";

    throw "I can't do this";

    console.log(whiteRabbit);
  })();
} catch {
  console.log("Trinity removes the bug");
}

console.log("Welcome to the real world");
Trinity removes the bug
Welcome to the real world
```

finally...

Try ... Catch ... Finally

We could also have a **finally block**, which will run in either case.

```
> try {  
  let whiteRabbit = "\u{0001F407}";  
  throw "I can't do this";  
  console.log(whiteRabbit);  
}  
catch {  
  console.log("Trinity removes the bug");  
}  
finally {  
  console.log("Got the Red pill");  
}
```

```
console.log("Welcome to the real world");
```

```
Trinity removes the bug
```

```
Got the Red pill
```

```
Welcome to the real world
```

```
< undefined
```

we could throw,
or not

the **finally** block
runs anyway

```
> try {  
  let whiteRabbit = "\u{0001F407}";  
  // throw "I can't do this";  
  console.log(whiteRabbit);  
}  
catch {  
  console.log("Trinity removes the bug");  
}  
finally {  
  console.log("Got the Red pill");  
}
```

```
console.log("Welcome to the real world");
```



```
Got the Red pill
```

```
Welcome to the real world
```

```
< undefined
```

so why do we need for an error object then?...

Differential error handling

Errors being an object is useful, because we can handle **different errors in different ways**.

Also, an error object can provide useful **information for debugging** in its properties.

also, movieMismatchErrors



```
> let chickenError = {
  name: "ChickenError",
  message: "I can't do this"
}

try {
  let whiteRabbit = "\u{0001F407}";

  throw chickenError;

  console.log(whiteRabbit);
}

catch (error) {
  switch (error.name) {
    case "RangeError":
      console.log("Got the wrong pill");
      break;
    case "ChickenError":
      console.log("What's wrong McFly? Chicken!");
      break;
    default:
      console.log("Houston, we have a problem");
  }
}

What's wrong McFly? Chicken!
```

Do we use try ... catch for error handling?

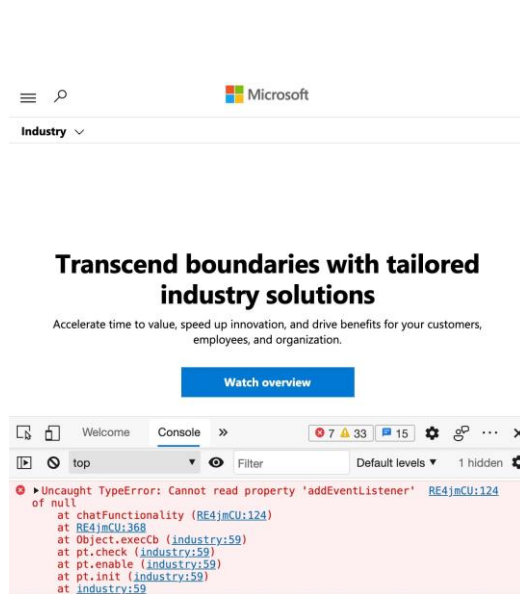
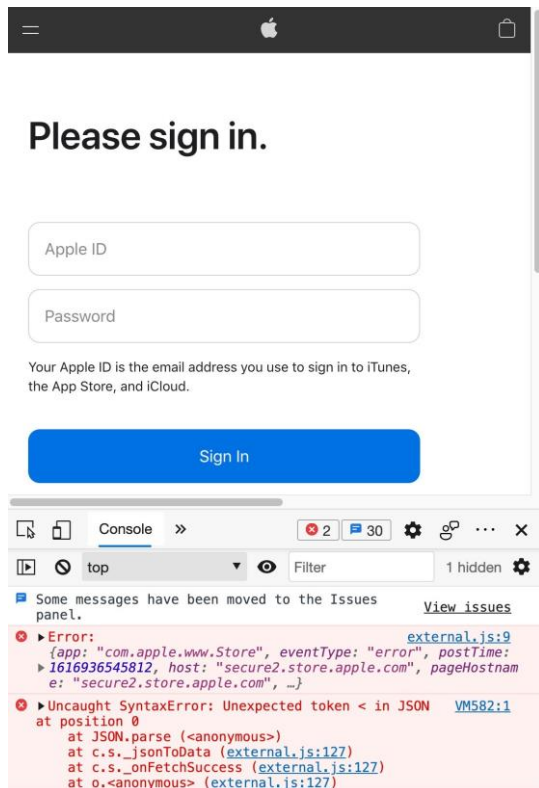
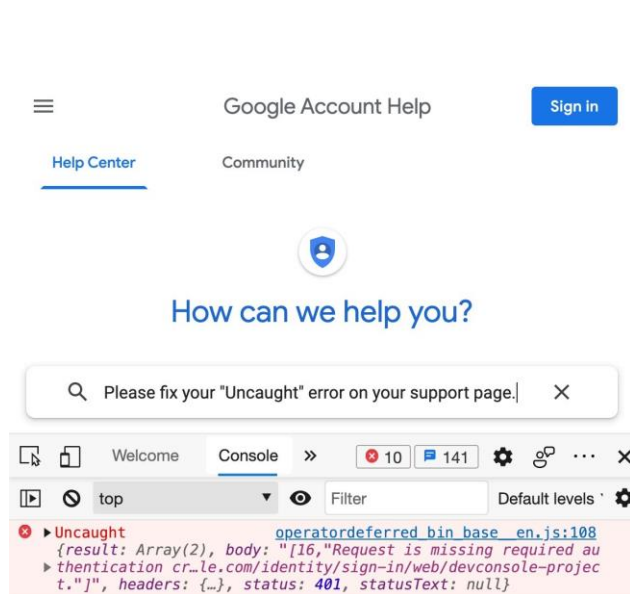
Depends on the project. Generally speaking, *error handling* is one of the most complex areas, and different projects require different approaches.

That being said, having errors on the console reflects less competent engineering than expected.

Interestingly, relying on TypeScript (false) safety in runtime could lead to more `TypeError`s if backend data is not validated properly (because you probably won't *check?.the?.existence?.of?.every?.property*).

No worries, though, *you will have errors* on console (see the next slide), and if you are prepared for that, you can handle as well.

It happens with the best...

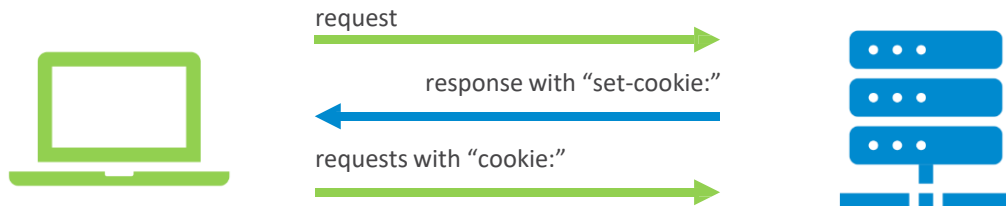


COOKIES

Cookies

Cookie is a string containing a semicolon-separated list of *key = pair* values

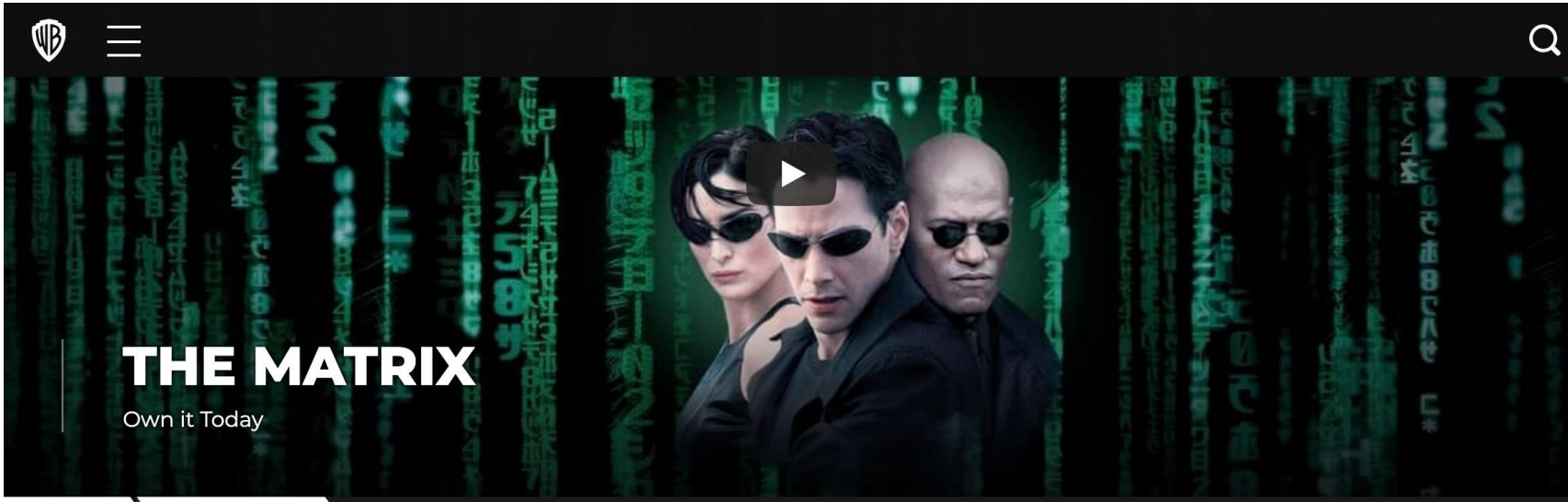
How do cookies work? **Cookies are transferred** in the http header (you can check at the network tab) **between the server and the client**. After a cookie was set, it will be stored in the browser for a specified time. The cookie then will be sent to server with **every subsequent requests**, even with requests for images*.



mr_anderson=neo; neo=not_the_one;

**because of this, cookies can be used for tricky activities, such as analyzing website traffic, communication between iframes, etc.*

[MDN: Cookies](#)



Application

Manifest
Service Workers
Storage

Storage

- Local Storage
- Session Storage
- IndexedDB
- Web SQL
- Cookies

https://www.warnerbros.com

Filter

☐ Only show cookies with an issue

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Priority
mr_smith	application error	www.warnerbros.com	/	Session	25				Medium
mr_anderson	neo	www.warnerbros.com	/	Session	14				Medium

don't believe your eye (#1): a cookie value cannot contain whitespaces

Cookie - attributes

Different attributes
can be set for a cookie:

`;path=path`
`;domain=domain`
`;max-age=max-age-in-seconds`
`;expires=date-in-GMTString-format`
`;secure`
`;httponly`
`;samesite`

*In case you wondered: there is a standard
for cookies – here is the new, the draft
version, signed by Google and Apple:*

Workgroup:	HTTP
Internet-Draft:	draft-ietf-httpbis-rfc6265bis-07
Obsoletes:	6265 (if approved)
Published:	7 December 2020
Intended Status:	Standards Track
Expires:	10 June 2021
Authors:	M. West, Ed. J. Wilander, Ed. Google, Inc Apple, Inc



[Cookies: HTTP State Management Mechanism](#)


Cookie – accessing from JavaScript

Cookies can be get/set with JavaScript as well (except for cookies with *;httponly* attributes) - one cookie at once, only: it's not a data property, it's an accessor (getter/setter).

An assignment to it is treated specially.

```
> document.cookie="WMF-Last-Access-Global=Because as we both know, without  
purpose, we would not exist.; domain=.wikipedia.org; path="/";  
< "WMF-Last-Access-Global=Because as we both know, without purpose, we would  
not exist.; domain=.wikipedia.org; path="/"
```

don't believe your eye (#2): it is the value of the assignment operator, not the new value of the cookie: it is an *httponly* cookie on wikipedia, so cannot be set; also whitespaces.



LOCAL AND SESSION STORAGES

Web storage

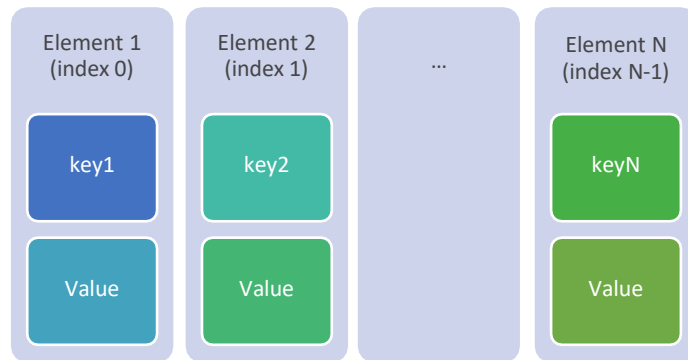
The Web Storage API provides mechanisms by which browsers can securely store key/value pairs.

Storage elements are simple key-value stores (strings).

2 different storages exist:

[sessionStorage](#) a separate storage area for the duration of the page session, including page reloads.

[localStorage](#) does the same thing but persists even when the browser is closed and reopened.



[MDN: Web storage API](#)

Storage – properties

Method	Description
length	The number of elements in the storage
key(n)	Returns the name of the n th key in the storage
getItem(keyname)	Returns the value of the specified key name
setItem(keyname, value)	Adds that key to the storage, or updates that key's value if it already exists. Also, it will throw an exception if the size quota is reached.
removeItem(keyname)	Removes that key from the storage
clear()	Empties all key out of the storage

Storage - examples

```
> const storageSupportedStr = window.sessionStorage && window.localStorage ? "is" : "is not";  
  `Storage API ${storageSupportedStr} supported`;  
⏏ "Storage API is supported"
```

```
> localStorage.setItem("We need guns", "Lots of guns");  
⏏ undefined
```

```
> Array.from({length:localStorage.length}, function(_,i) { return localStorage.key(i)})  
⏏ ▶ (10) ["as_tex", "ls-opt-out", "as-fcs1", "We need guns", "ac-storage-ac-store-cache",  
  "as-fcs2", "mk_epub_expiry", "test", "fl_products_507829", "mk_epub"]
```

```
> localStorage.getItem("We need guns");  
⏏ "Lots of guns"
```

```
> localStorage.removeItem("We need guns");  
⏏ undefined
```

```
> localStorage.getItem("We need guns");  
⏏ null
```

```
> localStorage.clear();  
⏏ undefined
```

```
> Array.from({length:localStorage.length}, function(_,i) { return localStorage.key(i)})  
⏏ ▶ []
```

Storage – storage event


Also, there is a special event for storages, it's called “storage”.

This storage event is triggered each time a value in a storage is modified from [another page](#).

```
> window.addEventListener(
  'storage',
  function({ url, key, newValue: value }) {
    console.log(`${key} => '${value}' on '${url}'`);
  }
);
< undefined
'Neo' => 'I know Kung Fu!' on 'https://www.epam.com/' VM424:4
```

```
> localStorage.setItem('Neo', 'I know Kung Fu!');
< undefined
>
```

different browser
window / tab



Cookies vs Storage API

Cookie	localStorage / sessionStorage
~ 4 kB	~ 5 MB
data lives until the end of expiration date / until it is deleted	data lives until deleted (sessionStorage: until the end of the session)
no event	storage event
document.cookie setter/getter	methods for writing and reading
automatically sent to the server	-
does not rely on JavaScript	requires JavaScript

Cookies are obsolete?

Far from it. Because of their special nature (they sent to the servers automatically), complex sites depends on cookies a lot. They won't go anywhere. However, exactly because of that, cookies are usually server-side concerns: they send and analyze them. 3rd party scripts also create cookies, but your main task with them will be to integrate to the side – their internal is not the business of a developer.

For storing data on client side, please use the Storage API.

A typical interview question: what to store in cookies / localStorage.
And a typical wrong answer is “username”. Why?

THANK YOU!