L5: JavaScript

Web Engineering

188.951 2VU SS20

Jürgen Cito

TU WIEN Informatics

L5: JavaScript

- Core Language Constructs
- Standard Library/Common APIs
- Event-driven and Asynchronous Programming

Learning Goals

- Become proficient in building client-side functionality in JavaScript
- Understand asynchronous/event-driven programming in the browser
- Be able to interact with common APIs in the browser
  (Document Object Model, Browser Object Model, HTTP, Local Storage)

JavaScript

JavaScript is an interpreted scripting language

- Originally developed by Netscape in 1995
- Standardized as ECMAScript language in the ECMA-262
  - Other ECMAScript implementations: JScript, ActionScript, QtScript, …
- **Focus on this course on newest standard: ECMAScript 6**
- Integration into HTML
  - Internal in <head> – using the <script> element
  - Internal in <body> – using the <script> element
  - External – linking to an external JavaScript file in <head>

```
<script type="text/javascript">
  statement;
  statement;
  ...
</script>
```

- JavaScript is executed as soon as it is read!

defer enables asynchronous loading

```
<script type=“text/javascript"
src=“script.js” defer />
```

```
<script type=“module">
  statement;
  statement;
  ...
</script>
```

type=“module”

is ‘defer’ by default

JavaScript

Variables have dynamic types

- Changing data changes type
- String, Number, Boolean, Array
- undefined if there is no value
- null empties the variable

```
var data;              // data undefined
data = true;           // Boolean object
data = '7';            // String object
data = Number(data);   // Number object
data = new Array();    // Array object
data[0] = "A";         // 0-based
data = ["A"];          // literal
data = null;           // data is empty
```

Objects (classically) have no underlying class!

- (Almost) everything is an object!
- Objects are associative arrays
  - key = property
  - Added when value assigned

```
var student = new Object();
student.nr = 'e120...';
student.name = 'A Student';
student.age = 19;
student.hasSTEOP = true;

student = { nr : 'e120...',
            name : 'A Student',
            age : 19,
            hasSTEOP : true };
```

Events

- List of pre-defined events for objects
- Mouse, keyboard, forms, loading, ...

```
<img src="..." alt="..."
     onclick="alert('Hello');" />
```

JavaScript

Declaration: `var/const/let <name> = <value>`

Variable literals

Number	String	Boolean	Array	Object	Function	Special
10.50 105	'Text' "Text"	true false	[1,2,3]	{a: 2, b: '4'}	**function** a() {}	null undefined

Dynamic type: **typeof true === "boolean"**

Type conversion functions

Number	Integer	String	Boolean
+"10"	parseInt("10")	5.toString()	Boolean("false")
10	10	'5'	true*

Every non-empty (`0`,`""`,`null`)-value is considered as true

Type conversions performed automatically

Lots of caveats, be careful!

```
x = 10 + 5          //15
x = 10 + "5"        //"105"
x = 10 * "5"        //50
x = [10] + 5        //"105"
x = true + 5        //6
x = +"10" + 5       //15
```

JavaScript
Operators

Typical Operators you expect

- Numerical operators: `+,-,*,/,++,--,%`
- Comparison operators: `>,>=,<,<=,==,!=,===,!==`

Two kinds of comparison operators

- Equality with type conversion `==, !=`
  - True if both expressions, converted to the same type, are equal
- (True) quality without type conversion `===, !==`

```
x = 0
if (x = 10) { //Assignment, returns true
   console.log("I: x has 10 assigned to it");
}
if (x == "10") { //Equality with type conversion, returns true now
   console.log("II: x is 10");
}
if (x === "10") { //Equality without type conversion, returns false
   console.log("III: x is 10");
}
```

JavaScript
Arrays (Heterogenous Lists)

Zero-based arrays with length,
mixed typed elements allowed

Add/remove elements on start/end
- `push/pop/shift/unshift`

Sort array
- `sort/reverse`
- `sort` with comparison function

Arrays can be concatenated with `concat`

String conversion with `join`(joinString)

Beware: Certain functions on array change the structure, others don't

```
x = new Array(2,3,1) //don't use
x = [2,3.6,1]
x.length            //3
x[1]                //3
x.push(0)           //[2,3.6,1,0]
x[x.length] = 6     //[2,3.6,1,0,6]
x.pop()             //[2,3.6,1,0]
x.shift()           //[3.6,1,0]
x.unshift(4)        //[4,3.6,1,0]
x.sort()            //[0,1,3.6,4]
x.sort(function(a,b){return b-a;})
                    //[4,3.6,1,0]
x.reverse()         //[4,3.6,1,0]
delete x[1]         //[4,undefined,1,0]
x[1] = 5            //[4,5,1,0]
x = x.concat([5,6])//[4,9,8,7,0,5,6]
x.join("-")         //4-9-8-7-0-5-6
```

JavaScript
Objects

Objects are associative arrays (dictionaries)

- Support dynamically adding/deleting values
- May contain functions

```
x = {name: "John Doe", nr: 123456}
x.name                  //"John Doe"
x["nr"]                 //123456
x.name = "Jane Doe"//x = {name: “Jane Doe" …}
delete x["name"]   //x = {nr: 123456}
x["a"] = 1         //x = {a: 1, nr: 123456}
x.func = function(x) {alert('Hello '+x);}
x.func(5);         //Displays 'Hello5'
x = {name: "John Doe”,…, func : function {…}}
```

JSON serialization

- JavaScript Object Notation
- Serialized JS objects as data exchange format
- Objects can be directly translated into JSON (`stringify`)
- JSON strings can be loaded as JS objects (`parse`)
- Limitations: Cannot de/serialize JS functions

```
x = {name: "John Doe", nr: 123456}
JSON.stringify(x)
x = JSON.parse("{'nr': 5}")
```

JavaScript
Strings / Template Literals

Contained in single or double quote (“…”, ‘…’)

- Backslash escapes (\)

String concatenation with + operator

- `"Dear " + s1.name + ", thank you for…”`

JavaScript Template Literals

- Use backquote (`) as delimiter
- Can span multiple lines (new line included in the string)
- Can have embedded expressions
- ${expression} — computes expression and replaces it with string value
- Replaces concatenation operations and complex expressions

```
message = `Dear ${student.name},
Your GPA, ${gpa(student)}, is
lower than the average: $
{averageGPA}`
```

JavaScript
Control Structures

Java-like control structures

- Condition, Switch/Case
- For-Loops
  - For: “Classic” for loops through a block a number of specified times
  - For-in: Loop through object keys/array indices
  - For-of: Loop through values of an iterable object
- While-Loop

Debug messages

- Display only in developer console: **console.log**(str)
- Popup message: **alert**(str)

```
if (a > 5) {
     console.log('Larger a!');
} else {
     console.log('Smaller a!');
}
```

```
switch(a) {
 case 0: alert('a is 0!'); break;
 default: alert('a is different!');
}
```

Uses strict comparison
===

```
x = [1,'a',3]//Iterates over values
for(let i of x) {
  console.log(i);
} //logs 1, a, 3
```

```
x = [1,2,3] //Iterates over indices
for(let i in x) {
  x[i] = x[i] * 4;
} //x === [4,8,12]
```

```
x = [1,2,3] // C/Java-like For Loop
for(let i = 0; i < x.length; ++i) {
  x[i] = x[i] * 4;
} //x === [4,8,12]
```

```
student = {name: "Jane Doe", nr: 123]
//Iterating over objects keys
for(let key in student)
  console.log(key); //logs name, nr
```

```
x = 3;
while(x > 0) {
 console.log(x*x); x--;
} //logs 9, 4, 1
```

```
x = 4;
do {
 console.log(x*x); x--;
} while(x > 10) //logs 16
```

JavaScript
Functions

Declared with keyword `function`

- Can have arguments (no predefined types)
- Can have a return value (no predefined return type)
- Function call only matched by name, not parameters
  - Missing parameters replaced by `undefined`
  - Additionally passed parameters are ignored
- Since ES6: Optional parameters by specifying default
  - Example: `function hasTopGrades(student, threshold=1.5)`
  - Before achieved through check for `undefined` in method body `if(threshold === undefined){threshold=1.5}`
- Functions can be nested (difficult variable scoping)

```
function addition(a, b) {
   return a + b;
}
//Can't control types
x = addition('A ', 'Student');
//b becomes undefined,
//7+undefined returns NaN
x = addition(7);
```

```
function counter(a) {
   function plusOne() {
     return a + 1;
   }
   plusOne();
   return a + 1;
}
```

```
function gpa(student) {
   if(student.grades.length == 0) return 0;
   let sum = 0;
   for(grade of student.grades) {
      sum += grade;
   }
   return sum / student.grades.length;
}
function topGrade(student, threshold=1.5) {
   return gpa(student) <= threshold;
}
s1 = { grades : [1,2,2,1] };//gpa(s1)===1.5
s2 = { grades : [1,2,2,2] };//gpa(s2)===1.75
topGrade(s1); //returns true
topGrade(s2); //returns false
topGrade(s1, 1.8); //returns true
```

JavaScript
Higher-Order Functions

Functions are first-class citizens

- Functions can be passed as parameters
- Functions can return other functions

Anonymous functions are functions without a name

- Also known as “Lambdas” in other languages
- Only works when passing functions as parameters or assigning them to a variable
- Syntactic Sugar: Fat Arrow Syntax =>
  - One parameter, one expression in body as return value
    ```
    student => gpa(student.grades)
    ```
  - Multiple parameter, one expression in body
    ```
    (a, b) => a + b
    ```
  - Multiple parameters, multiple expressions in body
    ```
    (a, b, message) => { console.log(message);
                         return a + b; }
    ```

```
s1 = { grades : [1,2,2,1] };
s2 = { grades : [1,2,2,2] };
let students = [ s1, s2 ];

function assignGPA(student) {
   student.gpa = gpa(student)
   return student;
}

let gpa_list = students.map(assignGPA)
//returns new map
[ { grades: [ 1, 2, 2, 1 ], gpa: 1.5 },
  { grades: [ 1, 2, 2, 2 ], gpa: 1.75 } ]

gpa_list.filter(
    student => student.gpa <= 1.5
)
//[ { grades: [ 1, 2, 2, 1 ], gpa: 1.5 } ]
```

JavaScript
Functions as Objects

Functions as objects

Functions (`function`) are first-class objects

- Functions are objects too
  - As objects, they can have properties and methods
  - Difference to pure objects:
    They can be called and can have return a value

"Methods"

- Definition assigned as properties (use `new` and `this`)

```
function Student(nr, name, age, hasSteop) {
   this.nr = nr;
   this.name = name;
   this.age = age;
   this.hasSteop = hasSteop;

   this.finishSteop = function() {
     this.hasSteop = true;
   }
}
var jc = new Student('e0828…',
                'Jurgen Cito', 29, false);
jc.finishSteop();
```

```
class Student {
   constructor(nr, name, age, hasSteop) {
      this.nr = nr;
      this.name = name;
      this.age = age;
      this.hasSteop = hasSteop;
   }

   finishSteop() {
     this.hasSteop = true;
   }
}
var jc = new Student('e0828…',
                'Jurgen Cito', 29, false);
jc.finishSteop();
//Student { nr: 'e0828112', name: 'Jurgen
Cito', age: 29, hasSteop: true }
//typeof(jc) === 'object'
```

ES6 Class is syntactic sugar for functions as objects

JavaScript
Object Prototypes

All objects have a prototype

- All prototype are object, any object can be a prototype
- Objects inherit properties and methods from prototype
- Object.prototype is top of prototype chain (its prototype is null)

```
var aStudent = { nr : 'e120...', ... };     // aStudent  -> Object.prototype
var students = [ aStudent, ... ];           // students  -> Array.prototype    -> Object.prototpye
var jc = new Student('e08…', ...);          // aoStudent -> Student.prototype  -> Object.prototype
function print() { ... };                   // print     -> Function.prototype -> Object.prototype
```

Existing prototypes can be extended at any time

- Beware of monkey patching!

```
String.prototype.distance = function() { ... };   // monkey patching


function Student(nr, name, age, hasSteop) { ... } // as before
Student.prototype.university = 'TU Wien';         // add property
Student.prototype.summary = function() {          // add "method"
   return this.nr + ' ' + this.name;
};
```

JavaScript
Variable Declaration and Scoping

Declaring variables is possible with keywords
`var, let, const`

- Function-scope: `var`
  - When a variable is declared within a function with `var`, it will only live in that function
  - If the variable has been declared outside the function, it lives in the outer scope
  - Function scopes can nest with nested functions
- Block-scope: `let, const`
  - Variables declared with `let` or `const` will only exist within its block
  - `const` identifiers cannot change
- Global Scope: Variables are global by default
  - If no keyword is used, the variable is available anywhere (globally)

```
students = […]
function GPAMessage(student) {
   let message = '';

   // no declaration of students required
   const averageGPA = avgGPA(students);
   const studentGPA = gpa(student)
   if(studentGPA < averageGPA) {
     const GPADifference = studentGPA - averageGPA;
     message = `Great job, ${student.name}.
                Your GPA is ${GPADifference}
                higher than the average`
   } else {
     // GPADifference is now not
     // available any longer in this scope
     message = `Dear ${student.name},
                Your GPA, ${studentGPA}, is lower
                than the average: ${averageGPA}`
   }
   return message;
}
```

JavaScript
Modules (ES6)

Establish namespace in separate module files

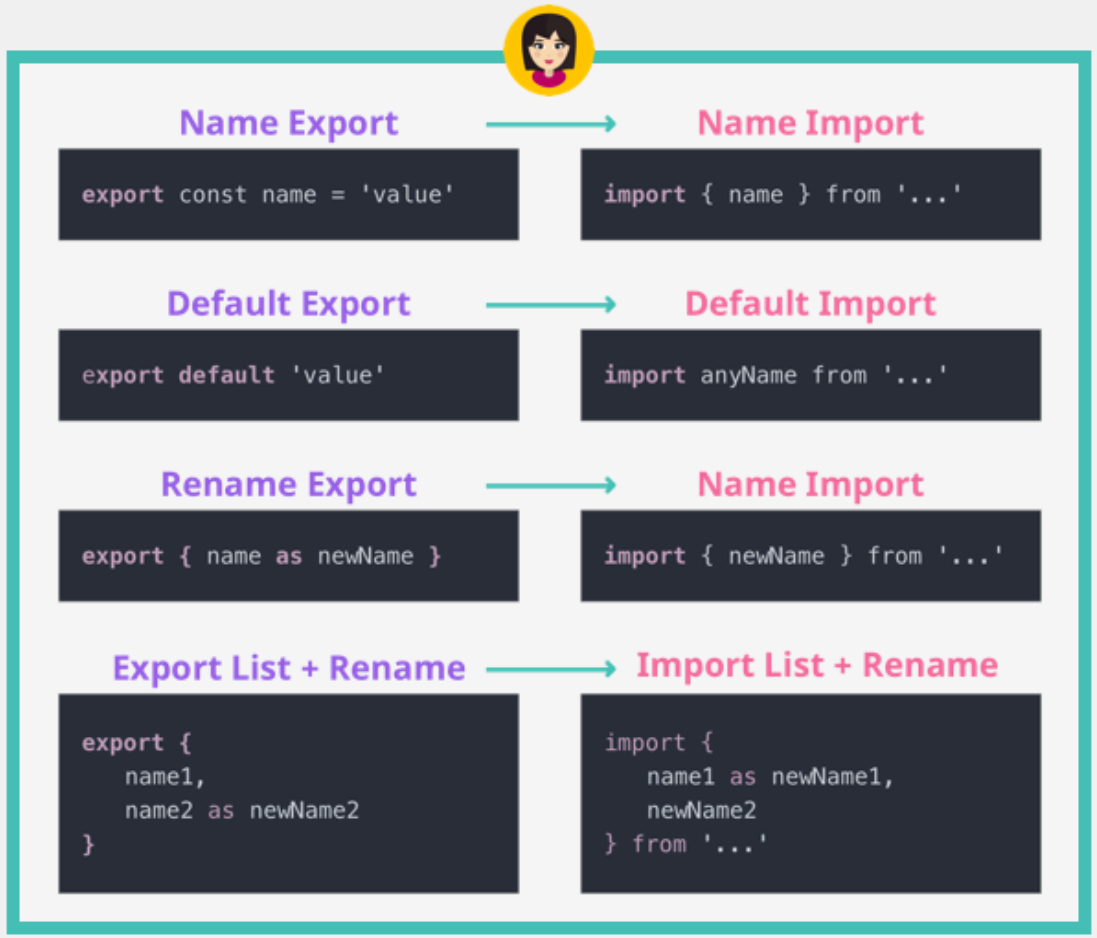
- One module per file, one file per module
- Ability to create named export or default exports

Export specific elements

- `export function gpa(student) {…}`
- `function gap(student) {…}; export gpa;`

Import in other files

- ```
  import * as Student from ‘./student.js';
  s = {…}; s.gpa = Student.gpa(s);
  ```
- ```
  import { gpa } from ‘./student.js’;
  s = {…}; s.gpa = gpa(s);
  ```



Source: samanthaming.com

Browser/Web APIs

- Browser Object Model (window)
- Storage (Local Storage, Cookies)
- Document Object Model
- (many more we are not discussing)

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

Browser Object Model (BOM)

- Allows access to browser objects
- Not standardized! (But very similar in all modern browsers)

Window is the global object

- All global objects, functions, and variables are members of window

Object	Property and Methods
window	Other global objects, open(), close(), moveTo(), resizeTo()
screen	width, height, colorDepth, pixelDepth, …
location	hostname, pathname, port, protocol, assign(), …
history	back(), forward()
navigator	userAgent, platform, systemLanguage, …
document	body, forms, write(), close(), getElementById(), …
Popup Boxes	alert(), confirm(), prompt()
Timing	setInterval(func,time,p1,…), setTimeout(func,time)

Storing Data

Cookies

- String/value pairs, Semicolon separated
- Cookies are transferred on to every request

```
document.cookie = "name=Jane Doe; nr=1234567;
expires="+date.toGMTString()
```

Web Storage (Local and Session Storage)

- Store data as key/value pairs on user side
- Browser defines storage quota

Local Storage (`window.localStorage`)

- Store data in users browser
- Comparison to Cookies: more secure, larger data capacity, not transferred
- No expiration date

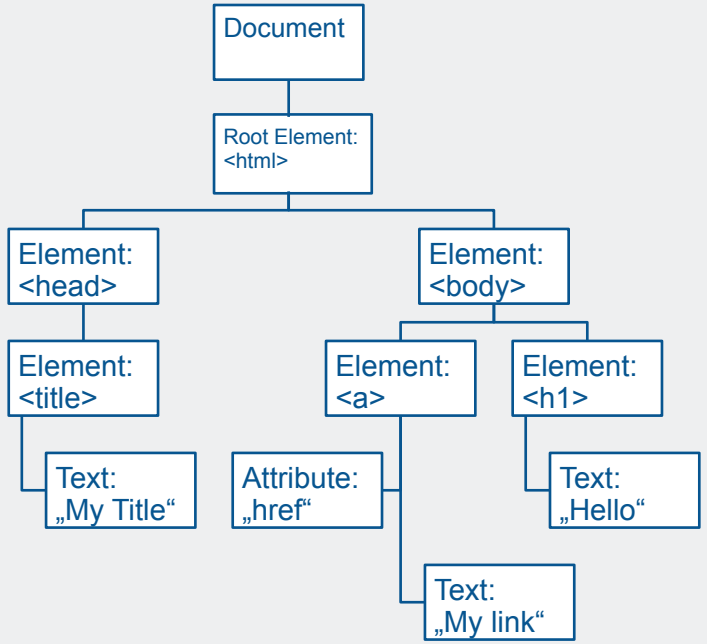
Session Storage (`window.sessionStorage`)

- Store data in session
- Data is destroyed when tab/browser is closed

```
let storage = permanent ? window.localStorage :
                          window.sessionStorage;
if(!storage["name"]) {
   storage["name"] = "A simple storage"
}
alert("Your name is " + storage["name"]);
```

Document Object Model (DOM)

- Tree structure for interacting with (X)HTML and XML documents
  - HTML elements as objects with properties, methods and events
- Standardized by the W3C
  - Platform- and language-independent
  - Levels: Level 1 ⊆ Level 2 ⊆ Level 3



Document Object Model (DOM)

Retrieving Elements

- ID, tag name, class name
- Document property
- ES6: Selector-based Access

Change Elements

- Content (innerHTML)
- Element attributes
- Element style
- Element class
  `(className, classList)`

Manipulating DOM Nodes

- Create, append, remove, …

DOM Traversal on Element

- `parentElement, nextElementSibling, previousElementSibling, childNodes`

```
let title = document.getElementById("title");
let links = document.getElementsByTagName("a");
let greens = document.getElementsByClassName("green");
let imgs = document.images;
let firstParaBox = document.querySelector(“p.box”);
let allBoxes = document.querySelectorAll(“p.box,div.box”);
```

```
title.innerHTML = "newTitle";
links[0].href = "http://...";
links[0].setAttribute("href",…)
greens[0].style.color = “red";
greens[0].className = “red"
greens[0].classList.add(“dangerzone”)
```

```
let header = document.createElement("h2");
let text = document.createTextNode("SubTitle");
header.appendChild(text);
document.removeChild(title);
document.replaceChild(title, header);
```

DOM Updates and Accessibility

Changing the DOM can introduce accessibility issues

- (Frequent) updates can confuse screen readers
- Updates may not be able in high magnification
- Certain updates may be invisible (e.g., introducing a red border for errors)
- Updates may come too fast (before pre-update part of the page was read)

Guidelines

- If the content updates for more than 5 seconds, provide ability to pause, stop, or hide them
- Inform users of changes: Setting focus, Highlight, Alert, Live Region (ARIA Term)
- Communicate to user that page is dynamic
- Provide static HTML page alternatives

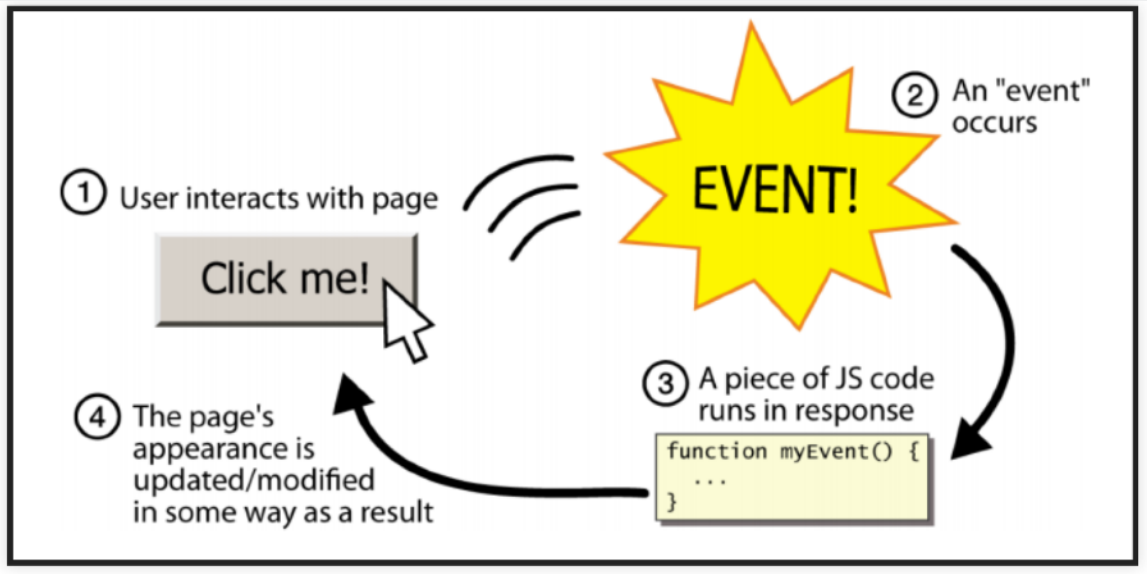
Careful testing necessary (screen reader testing, proxy testing with text browser)

Source: Steven Reiss, Brown University, CS1320 “Dynamic Web Pages”

Event-driven and Asynchronous Programming

- Events in HTML/DOM
- Asynchronous Programming: Callbacks vs. Promises
- `async/await` Syntax
- HTTP (Fetch)

Event-driven JavaScript Programming



Event-driven Programming:

- Flow of the program is determined by responding to user actions called **events**
- Writing programs driven by user events

Source: Hovik, University of Washington, CSE154, https://courses.cs.washington.edu/courses/cse154/19sp/sections/week03-tues/slides/index.html#/4

DOM Events

Event callback attached to HTML elements

```
<button onclick="alert('Test!')">
   Test me!
</button>
```

```
let button = document.getElementsByTagName("button")[0]
header.click(); //Execute predefined event
header.onclick = function(){alert('Clicked!');}
  //Set event listener - only one listener supported
let func = function() {alert('Clicked!');}
header.addEventListener("click", func)
header.removeEventListener("click", func)
```

Event types (selection)

Event	Description
load,unload	User enters/leaves page
change	Form input field changes
focus/blur	User focuses/unfocuses an input field
Submit	Form is submitted
mouseover, mouseout	Mouse enters/leaves region
mousedown/mouseup/click	Mouse click events
keydown/keyup/keypress	Keyboard events
drag	User drags an element

Sending Asynchronous Requests (Callbacks)
HTTP (XMLHttpRequest)

```
const API_BASE_URL = 'https://pokeapi.co/api/v2';
const pokemonXHR = new XMLHttpRequest();
pokemonXHR.responseType = 'json';
pokemonXHR.open('GET', `${API_BASE_URL}/pokemon/1`);
pokemonXHR.send();

pokemonXHR.onload = function () {
    const moveXHR = new XMLHttpRequest();
    moveXHR.responseType = 'json';
    moveXHR.open('GET', this.response.moves[0].move.url);
    moveXHR.send();
    moveXHR.onload = function () {
        const machineXHR = new XMLHttpRequest();
        machineXHR.responseType = 'json';
        machineXHR.open('GET', this.response.machines[0].machine.url);
        machineXHR.send();
        machineXHR.onload = function () {
            const itemXHR = new XMLHttpRequest();
            itemXHR.responseType = 'json';
            itemXHR.open('GET', this.response.item.url);
            itemXHR.send();
            itemXHR.onload = function () {
               itemInfo = this.response;
               console.log('Item', itemInfo);
            }
        }
    }
}
```

Classic network request API (XMLHttpRequest)

- Used callbacks - a mechanism to provide a function that gets called once you receive a response from HTTP
- Callbacks resulted in increasingly nested callback chains dubbed “callback hell” http://callbackhell.com/

Code adapted from Wendell Adriel: https://medium.com/@wendell_adriel/from-highway-to-callback-hell-to-stairway-to-async-await-heaven-e90f73309ed4

Sending Asynchronous Requests (Promises)
HTTP (fetch)

`fetch` API allows to process HTTP requests/ responses using **promises:**

- Promises are a general wrapper around asynchronous computations and callbacks
- They represent how to get a value - you tell it what to do as soon as it receives the value
- `Promise` is a proxy object for a value that is not yet known
  It is modelled with the following states
  - Pending (inital state)
  - Fulfilled (execution successful)
  - Rejected (operation failed)

```
fetch('./movies.json')
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(err => console.log(err));
```

A promise defines a `function (resolve, reject)` asynchronously loading data

- **resolve**(*data*) on success
- **reject**(*errorObject*) otherwise

The callback is defined with `then(success, failure)`

- **success**(*result*)
- **failure**(*errorObject*)

`catch(failure)` is a shortcut for `then(undefined, failure)`

Sending Asynchronous Requests (async/await)
HTTP (fetch)

`async/await` is a special syntax to work with promises

- `async` is a keyword around a function that wraps a promise around its return value.

```
async function f() { return 1; }

f().then(alert); //requires then to resolve result
```

- `await` is a keyword that makes JavaScript wait until the promise is resolved and can the return the value (*only works within async functions!*)

```
let response = await fetch("./movies.json")
```

```
async function showAvatar(username) {
  // read github user
  let githubResponse = await fetch(`https://
api.github.com/users/${username}`);
  let githubUser = await githubResponse.json();

  // show the avatar
  let img = document.createElement('img');
  img.src = githubUser.avatar_url;
  img.className = "promise-avatar-example";
  document.body.append(img);

  // wait 3 seconds
  await new Promise((resolve, reject) =>
                    setTimeout(resolve, 3000));

  img.remove();
  return githubUser;
}
```

Adapted from https://javascript.info/async-await