18 Javascript/JSON

CS6 Practical System Skills
Fall 2019
Leonhard Spiegelberg lspiegel@cs.brown.edu
18.01 History of Javascript

⇒ created by Brendan Eich in 1995 for Netscape Navigator

⇒ positioned originally as web "companion" for Java, though there is no connection between Java and Javascript

⇒ scripting language for webpages

→ typically used to manipulate documents on the client-side, i.e. javascript allows client-side computing.

→ also used for server-side scripting (node.js + more)
18.02 The big picture

**HTML content**
specify content with tags, e.g.

```
<p>this is a <b>bold</b> statement</p>
```

**CSS presentation**
rules to style the content

```
p { color: red; }
```

**Javascript behavior**
run logic to change content/presentation dynamically

```
alert("Hi!");
```
18.02 Why bother to learn another language?

Stackoverflow 2019 survey
https://insights.stackoverflow.com/survey/2019#overview

IEEE Spectrum 2019 survey

Javascript is among the most popular languages
18.02 Resources for Javascript

Book:


*Today:* Chapters 1-8

Web:

developer.mozilla.org/en-US/docs/Web/JavaScript

javascript.info
18.02 How to work with Javascript

⇒ Chrome/Firefox/Safari have a built-in Javascript console that can be used to execute/develop code in a REPL:

<table>
<thead>
<tr>
<th></th>
<th>Mac</th>
<th>Win</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chrome</td>
<td>Cmd + Opt + J</td>
<td>Ctrl + Shift + J</td>
</tr>
<tr>
<td>Firefox</td>
<td>Cmd + Opt + K</td>
<td>Ctrl + Shift + K</td>
</tr>
<tr>
<td>Safari</td>
<td>Cmd + Opt + C</td>
<td></td>
</tr>
</tbody>
</table>

⇒ very useful for developing small snippets are online services like jsfiddle.net or codepen.io/pen/
18.02 Basic Javascript

⇒ Javascript is a weakly typed dynamic language similar to Python
⇒ C-like statements (optionally) terminated with ;
⇒ boolean expressions with true/false and && / ||
⇒ increment ++ and decrement -- operators
⇒ C-like comments using // or /* ... */

1 + 3
4 * (5 - 3) ** 3
"There are " + 26 + " letters"
type && ("hello" > 'world')
10 & 3

no casting of 26 to string necessary
18.02 Hello world in Javascript

⇒ To write Javascript code as part of a HTML page, you have the following options:

1. embedded, write (similar to style tags for CSS) code within `<script>....</script>` tags

2. external scripts, i.e. put JavaScript code into a separate file and include it via `<script src="<path"></script>`

Note: code within tags will be ignored if src is given...

⇒ you can add as many script tags to a HTML page as you like!
18.02 Where to place javascript file?

⇒ script tag can be placed

1. in the **head** section
2. in the **body** section
3. after the `<body>` tag (i.e. before `</html>`) 

⇒ position of the script tags in the document determines when the script is executed.

—> **Best practice**: Include scripts in head, page specific code in body
18.02 Hello world in Javascript

⇒ we can output "Hello world" via document.write(...)
you can define variables using the following keywords in Javascript:

1. `let message = "Hello world"`
2. `const message = "Hello world"` (constant)
3. `var message = "Hello world"` (old style, don't use)
4. `message = "Hello world"` (implicit global, don't use)

variable names must only contain alphanumeric characters, $ or _. The first character must not be a digit.

→ I.e. `$ = 20` is legal JavaScript! We'll see this when working with jQuery.
18.04 Javascript functions

⇒ Functions are first-class objects in Javascript. They can be declared using the `function` keyword or as lambda/anonymous functions, e.g. via `=>`.

```javascript
function sum(a, b) {
    return a + b;
}

// function expression, assign to diff identifier
let diff = function(a, b) { return a - b; }

// you can either use an expression or a sequence of statements in { ... }
let pow = (a, b) => a ** b

let x = 10
const y = 42

document.write(sum(x, sum(y, 1) + diff(y, x) * pow(2, 3)))
```
⇒ Prefer to use `let`. There's a subtle difference in `var/let`: `let` declares block variables, i.e., they can be only accessed in the scope where they were declared. `var` declares global variables at function-level. (Details e.g. on [https://javascript.info/var](https://javascript.info/var)).

```javascript
{ 
  let message = "hello world";
}
document.write(message); // Uncaught ReferenceError: message is not defined

{ 
  var message = "hello world";
}
document.write(message) // works
```
⇒ JavaScript has C-like control structures: for/while/if

```javascript
for(let i = 0; i < 5; i++) {
    document.write(i * i + '<br>'
}

let counter = 0;
while(counter < 10) {
    console.log(counter)
    counter += 2
}

if(counter == 10)
    document.write('counter is 10')
else
    document.write('counter is not 10')
```
18.07 String formatting in Javascript

⇒ There's no built-in sprintf / format, however you can create strings using Javascript's implicit string conversions or via the toString method for each object.

⇒ to convert a string object to int or float type use

parseInt / parseFloat or constructors Boolean / Number
 ⇒ Arrays can be declared using \([ \ldots \) similar to Python.

```javascript
let arr = [1, 2, 3, 4, 5, 'hello']

arr.length

arr[3] // 0, ..., length-1 indices are allowed

arr[1] = 42

arr.concat([1, 2, 3])
```
Everything in Javascript is an object. An object can have one or more properties to which a value can be assigned to.

```javascript
let hotel = new Object(); // create new, empty object

hotel.name = 'Quay'
hotel.rooms = 40
hotel.booked = 25

hotel.checkAvailability = function() { return this.rooms - this.booked; }

hotel.rooms // access rooms via . syntax
hotel['rooms'] // or via [key]

delete hotel.name // remove property from object

hotel.stars = 5 // add new property
hotel['stars'] = 5 // alternative
```
⇒ Instead of assigning properties in statements, Objects can be also constructed using literal syntax:

```javascript
let user = {
  name: 'Tux',
  profession: 'penguin',
  age: 30
}
```

Note: This also works with functions!
to define custom structures/objects, a constructor syntax can be used:

```javascript
function Hotel(name, rooms, booked) {
    this.name = name;
    this.rooms = rooms;
    this.booked = booked;
    this.checkAvailability = function() {
        return this.rooms - this.booked;
    };
}

// use constructor
let quayHotel = new Hotel('Quay', 40, 25);
let parkHotel = new Hotel('Park', 120, 77);
```
DOM manipulation
18.10 DOM manipulation using Javascript

DOM = Document Object Model, every element in a HTML page is represented as Object that can be manipulated

⇒ the objects are organized as nodes in a tree, the DOM-tree

⇒ via Javascript nodes can be added, altered, removed

⇒ root node is document
18.10 DOM tree example

Example taken from: https://data-flair.training/blogs/javascript-dom/
18.10 Basic DOM manipulation

⇒ Javascript provides several functions to create & place nodes

Example:

```javascript
let paragraph = document.createElement("p");

let content = document.createTextNode("This is some text the paragraph contains....");

paragraph.appendChild(content);

// append paragraph tag after body tag
document.body.appendChild(paragraph);
```
### 18.10 Javascript DOM functions

<table>
<thead>
<tr>
<th>accessing/finding elements</th>
<th>creating elements/manipulating element</th>
<th>manipulating the tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>document.getElementById(id)</td>
<td>document.createElement (name)</td>
<td>parentNode. appendChild (node)</td>
</tr>
<tr>
<td>document.getElementsByTagName (name)</td>
<td>parentNode. appendChild (node)</td>
<td>parentNode. remove Child (node)</td>
</tr>
<tr>
<td>document.querySelector(selector)</td>
<td>element.innerHTML</td>
<td>parentNode. replace Child (old, new)</td>
</tr>
<tr>
<td>document.querySelectorAll(selector)</td>
<td>element.style.left</td>
<td></td>
</tr>
<tr>
<td></td>
<td>element.setAttribute ()</td>
<td></td>
</tr>
<tr>
<td></td>
<td>element.getAttribute ()</td>
<td></td>
</tr>
<tr>
<td></td>
<td>element.innerHTML</td>
<td></td>
</tr>
<tr>
<td></td>
<td>element.style.left</td>
<td></td>
</tr>
<tr>
<td></td>
<td>element.setAttribute ()</td>
<td></td>
</tr>
<tr>
<td></td>
<td>element.getAttribute ()</td>
<td></td>
</tr>
<tr>
<td></td>
<td>element.addEventListener ()</td>
<td></td>
</tr>
</tbody>
</table>

18.11 Javascript event handlers

⇒ Nodes in the DOM tree can have event handlers to specific events assigned. I.e. if a certain event happens, some code is executed.

⇒ The general syntax is `<element event="some code">`

  ⇒ `element` is some HTML tag

  ⇒ `event` is one specific event type, e.g. `onload`, `onclick`, ...

⇒ complete list of events is available at

⇒ alternatively, an event handler can be assigned directly via e.g.
  `node.onload = ... or via addEventListener`
18.11 Javascript DOM events example

```html
<button id="btn" onclick="click_btn();">Click me!</button>
<p id="btn-target"></p>
<span>Some text...</span>

<script type="text/javascript">
function click_btn() {
    let p = document.getElementById("btn-target");
    p.textContent = "You clicked a button, awesome"
}

// add event listener to span elements
let span = document.querySelector('span')
span.addEventListener('mouseover', function() {
    this.style.backgroundColor = "#ff0000"; });
span.addEventListener('mouseout', function() {
    this.style.backgroundColor = "#00ff00"; });
</script>
```
jQuery
18.12 What is jQuery?

⇒ "write less, do more" library

⇒ makes life easier by helping with
  - HTML/DOM manipulation
  - CSS manipulation
  - HTML DOM events
  - AJAX (requests!)

⇒ >25% of all websites use jQuery (still).
18.12 Why do we need a library?

⇒ **Problem:** Browser incompatibility. Versions/Vendors/Devices/…

⇒ There is not one javascript standard, different browsers support and provide different functions.

⇒ jQuery to provide standard interface across browsers.
Basic Usage: `$('btn-target').addClass('highlight')`
18.12 Where to put jQuery code

⇒ Loading a website might take a while. However, often adding event listeners or manipulating the DOM makes only sense when the website is fully loaded.

⇒ jQuery provides a special `.ready()` method which will execute its argument when the document is "ready"

```javascript
$(document).ready(function() {
  // write all code here...
});

// alternative:
$(function() {
  // write all code here
});
```
18.13 Basic jQuery methods

⇒ `.html()` allows to retrieve the html of the first matched element,
  `.html(...)` allows to set the html of the first matched element

⇒ `.text()` / `.text(...)` returns the text content of the element and its children.

⇒ `.before()`/.`after()` allow to insert content before/after an element

⇒ `.prepend()`/.`append()` insert content inside the element

⇒ `.attr(name)` retrieves value of attribute name `.attr(name, value)` allows to update the attribute name with value

⇒ `.addClass(cls)` / `.removeClass(cls)` add/remove CSS class to element.

⇒ `.css(...)` allows to get or change css rules.
Passing data
18.14 Backend vs. Frontend

**Backend:** what happens on the server side

**Frontend:** what happens on the local machine (i.e. in the browser)

⇒ We've seen that we can manipulate the DOM tree with Javascript on the client side, i.e. all logic runs in the browser once the script is downloaded.

⇒ How can we pass data from the backend to the frontend?

⇒ How can we request data from the backend?

⇒ In which format shall we pass data?
18.14 Backend meets Frontend and vice versa

⇒ How can we pass data from the backend to the frontend?

   HTTP requests! GET/POST/… to an URI and process the response or via templating when generate the initially requested page.

⇒ How can we request data from the backend?

   issue HTTP requests via Javascript and process the response

⇒ In which format shall we pass data?

   Good question...
To pass data between two actors, we need to exchange it in some format because each side may have a different representation and has the data scattered in main memory (not in one location).

Ideally, both actors can convert their representation quickly to the format (serialization) and convert the format quickly to their representation (deserialization).

Javascript brings a default serialization format called JSON = Javascript Object Notation to the table.
18.14 JSON = JavaScript Object Notation

⇒ details under https://www.json.org/

⇒ encode data similar to python dictionaries as

   { "key" : value, ... }

⇒ Arrays via [...]

⇒ can be nested

Note: In JSON keys are always strings. Strings need to be always quoted with " (escape " via \")

Example:

```json
{
    "color" : "purple",
    "id" : 210,
    "composition" : {
        "R" : 70,
        "G" : 39,
        "B" : 89
    },
    "names" : ["violet", "lilac"]
}
```
18.14 JSON in Javascript/Python

⇒ Javascript and python both have support for JSON already built in

```javascript
let msg = {name: "tux", profession: "penguin", location: 'antarctica'}

// serialize
// yields "{"name":"tux","profession":"penguin","location":"antarctica"}"
let str = JSON.stringify(msg)

// deserialize
// yields {name: "tux", profession: "penguin", location: 'antarctica'}
JSON.parse(str)
```

```python
import json
msg = '{"name":"tux","profession":"penguin","location":"antarctica"}'
# yields {'name': 'tux', 'profession': 'penguin', 'location': 'antarctica'}
user = json.loads(msg)
# yields '{"name": "tux", "profession": "penguin", "location": "antarctica"}'
json.dumps(user)
```
18.14 Connecting via Ajax Requests

Ajax = asynchronous javascript and xml

⇒ allows to make HTTP requests via JavaScript whose response can be used to alter the webpage after the request succeeded.

⇒ easiest to do with jQuery which provides $.get(...) and $.post(...) functions to perform GET or POST requests.

⇒ Often, requests are made to endpoints which return JSON data (MIME: application/json).

⇒ Next lecture: RESTful APIs/design - usually based on JSON.
Demo

Examples from today available under github.com/browncs6/JSExamples
End of lecture.

Next class: Tue, 4pm-5:20pm @ CIT 477