JavaScript, DOM, and events

Michael Chang

Spring 2020

Plan for today

More JS language features

Arrays, Objects, iteration

Inputs and events

`<input>` and `<button>`, event handlers

Example: calculator

Adding/removing DOM elements

JS Arrays

Array syntax

```
let arr = [10, 20, 30];
/* Usual indexed for loop */
for (let i = 0; i < arr.length; i++)
  console.log(arr[i]);
/* Loop over elements */
for (let elem of arr) console.log(elem);
```

Caution: `for ... of` is very different than `for ... in`

JS Array operations

Useful Array operations

`arr.push(elem1[, elem2, ...])`

Add element(s) to an array

`arr.indexOf(value[, start])`

Get index of value in arr (-1 if not found)

`arr.splice(index, delCount[, newElem1, ...])`

Insert and/or remove elements at index

Warning: `delete arr[i]` doesn't work!

JS Objects

(Plain) Object is a key-value store

- Keys must be strings (one exception later)
- Values can be anything

Syntax

```
let obj = {
  binky: 42,
  winky: "Hello",
  "key w/ $pecial_chars": []
};
console.log(obj["binky"]);
```

JS Objects

Shorthand syntax

If key is a valid identifier, can use dot

```
console.log(obj.binky);
obj.dinky = 193;
```

Best practice: Use dot when possible

JS Object operators

Operators

`"key" in obj`

Check membership

Note: `obj.nonexistentKey -> undefined`

`if (!obj.nonexistentKey)` is common/useful, but be careful of falsy values

`delete obj.key`

Remove key/value pair

JS Object functions

Functions

("static" on Object, not methods on individual objects)

Key/value pairs iterated in insertion order

`Object.keys(obj)`

Array of object keys (insertion order)

`Object.values(obj)`

Array of object values

`Object.entries(obj)`

Array of pairs (arrays with length 2) of [key, value]

JS Object iteration

```
for (let key of Object.keys(obj))
  console.log(key + ": " + obj.key);

for (let [key, value] of Object.entries(obj))
  console.log(key + ": " + value);
```

`for ... in` can also iterate Object keys

- Recommendation: Avoid `for ... in` because it's confusing

Note: object references

Arrays and Objects are mutable

Variables and argument store references

```
const addElem = (arr) => {
  arr.push(42);
};
let arr = [1,2,3];
addElem(arr);
console.log(arr); // [1, 2, 3, 42]
```

Aside: newer language features

"Destructuring": assign to multiple vars

```
/* Get first and second elems of arr */
let [first, second] = arr;
/* Variable name matters here! */
let { binky, winky } = obj;
/* Fancier technique, "rest" value */
let [first, ...rest] = arr;
```

Template strings

```
for (let [key, value] of Object.entries(obj))
  console.log(`${key}: ${value}`);
```

Can contain any JS expression

So far

More JS language features

- Arrays, Objects, iteration

Inputs and events

- `<input>` and `<button>`, event handlers

Example: calculator

Adding/removing DOM elements

A couple more DOM things

Don't forget defer on script tags

`document.querySelectorAll("selector")`

Return a list of elements matching selector

`elem.querySelector(...)`

Search only in descendants of elem

`elem.textContent`

Get the text inside a node

Best practice: avoid `elem.innerHTML`

Lets you get/set raw HTML from JS, leads to security issues

HTML interactors

`<input>`: get user input

- Leaf element (no closing tag)
- type determines input type
  - text, checkbox, radio
  - Best practice: many newer types: number, email, date, ...
- Default to text

`<button>`: a button

- Best practice: don't use `<input type="button">`
- Children can be anything (text, images)

JS: handling events

`elem.addEventListener(type, fn)`

type is the event to handle (e.g. click)

fn is a function to handle the event

Note: functions can be passed as values!

Event types

Mouse: click, mouseenter, mouseleave

Keyboard: keydown, keyup, keypress

Interaction: change, focus, blur

Best practice: semantic elements (again)

E.g. any element can have a click event, but use button/link/etc. when possible

JS: handling events

```
const handleClick = (event) => {
  /* ... */
};

let button =
    document.querySelector("#clickme");
button.addEventListener(
    "click", handleClick);
```

JS: event argument

Get info about the event

`event.currentTarget`

The element that triggered the event

Aside: `event.target` is different

```
const handleClick = (event) => {
  let elem = event.currentTarget;
  elem.textContent = "I was clicked!";
};
```

So far

More JS language features

Arrays, Objects, iteration

Inputs and events

`<input>` and `<button>`, event handlers

Example: calculator

Adding/removing DOM elements

Technique: display: none

Approach

- Write all the elements into the HTML up front
- Set `display: none` (e.g. use a "`hidden`" class) on unused elements
- Add/remove display from JS

To be clear, this is a good approach

But what if we need more control?

- Many similar elements, don't want to copy/paste
- Variable number of elements

Creating elements

`document.createElement(tag)`

Create new element with tag (e.g. "img")

`node.cloneNode(deep)`

Shallow or deep copy of node

Should always pass arg (default changed)

Not added to tree

Set attributes, add children

Then add to tree in desired location

Traversing and changing the tree

`node.parentNode`

`node.childNodes`

Traverse the DOM tree

`node.appendChild(childNode)`

Add childNode to the end of node

`node.remove()`

Remove node from the tree (still valid object)

Aside: Node vs. Element

An Element is a type of Node

- Represents an HTML tag

Another type is a Text node

- Represents the text inside elements

Often most useful to traverse elements

- `elem.children`: return the **elements** under elem
- StackOverflow: childNodes vs. children

I'll probably use them interchangeably

Aside: frontend frameworks/libraries

Some are very useful for large projects

- E.g. React uses HTML-like syntax directly in JS and builds the nodes for you

Some are a product of an older time

- jQuery used to be very popular
- Provides many functions for accessing HTML/CSSS attributes, defining events, etc.
  - E.g. $("selector") is like querySelector
- But now unnecessary given broad support for standard techniques,
- Best practice: avoid jQuery
  - There's still a lot in the wild (SO answers, etc.)

Summary

So far

- Dynamic web pages through DOM manipulation
- User input and event handling

Before next time

- assign1 due tomorrow
- assign2 + project proposal out this weekend

Next week

- More event examples and issues
- JS classes and modules