Cubstart Web Lab 3

[start recording]

Administrivia

- HW2 is due today, Sep. 29 at midnight
- HW2 walkthrough at the end of lab! (~5:45 pm)
- HW3 is due next Friday, Oct. 6

Recap: JavaScript

Where to put JS Code?

- Just like CSS, we want to put our JS in a separate file (.js)
- Put this tag at the **bottom** of your HTML file

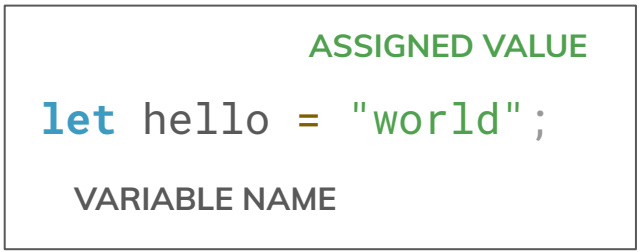
```
<script src="filename.js"></script>
```

- Why at the bottom? Your JS might rely on your HTML elements existing. If JS is at top, browser will run it before the rest of your HTML is rendered, causing errors.

Declaring Variables

Here are a couple ways to **declare** variables and store values with unique names for use:

- `let hello = "world";`
- `const world = 100;`
- `var random = false; // Don't use this!`



let vs. const vs. var

let

- block scoped
- can be reassigned

const

- block scoped
- cannot be reassigned

var

- globally/function scoped
  - Accessible within the entire function body
- can be reassigned

Primitive types in JS

- **number:** `1, 200, 1.8, 0.5, -1000, 10e5, ...`
- **string:** `"hello", ‘world’, ...`
- **boolean:** `true, false`
- **null** `//empty or blank value`
- **undefined** `//variable has been declared, but not defined`

Everything else is an “Object”, including arrays and functions!

Another type: `arrays/lists`

Lists can be used to store multiple values.

For example:

```
let lst = ["go", "bears", ":)"];
```

Types in a list can be different:

```
let vals = [true, “hi”, 3.9];
```

Some useful operations you can do on arrays.

Getting values from lists

```
console.log(lst[1]);

// Output: bears
```

Setting values in lists

```
lst[0] = "hi!"

console.log(lst[0]);

// Output: hi!
```

Another type: `objects`

Objects are basically dictionaries/maps. They have a set of “keys”, which each correspond to a “value”. Values can be anything: primitives, functions, lists, even other objects. Keys can only be strings or symbols.

For example:

```
let obj = { a: 'go', b: 'bears' };
```

VALUE VALUE

KEY KEY

```
console.log(obj.a); // go

obj.b = 'fish'

console.log(obj.b); // fish
```

Some useful operations you can do on objects.

Arguments = data that you pass into a function

```
function functionName(arg1,...) {
    // Do something...
    return returnValue;
}
```

```
function magic(num) {
    return num * 2 + 10;
}

console.log(magic(5)); // 20
console.log(magic(28)); // 66
console.log(magic(14)); // 38
```

Functions

Functions let you reuse a chunk of code without having to write the code again.

They help reduce repetitions in your code.

Basic Math

Some basic math operations

- General: `+, -, *, /`
- Power: `base ** exponent`
- Modulo: `dividend % divisor`
    - Example: `7 % 3 === 1` (Floor div: `7 / 3 === 2`)

Basic Strings

`split`: splits the string into an array of substrings using the delimiter

`includes`: checks if string includes specific character(s)

`toUpperCase & toLowerCase`: converts the string to upper or lowercase

`slice`: returns the specified part of the string (substring)

`replace`: returns the same string, but with the specified replacements

`indexOf`: returns the index of the character(s) within the string and -1 if the character(s) don't exist in the string

`charAt`: returns the character of the string at a given index

For more: check out the MDN page on JS string methods

Control Flow

Conditionals, loops, and more functions

```
if (condition) {
    // Do something...
} else {
    // Do something else...
}
```

```
if (condition1) {
    /* logic if condition1 is true */
} else if (condition2) {
    /* logic if condition1 is false
and condition2 is true */
} else {
    /* logic if condition1 and
    condition2 are false */
}
```

If Statements

`if, else if, else` statements

If statements allow you to execute code based on certain **conditions**

`if` runs the block if the condition is **truthy.**
Falsey values:

- `false`
- `0`
- `null`
- `undefined`
- `‘’`
- `NaN`

Everything else is truthy!

Conditionals: loose equality operators

== and != are **loose equality** operators (they don't compare the value types, only the values).

Some examples:

```
true == 1 // true

[] == 0 // true

6 == '6' // true

5 == 6 // false
```

=== and !== are **strict equality** operators (they compare the type of the object and it's value).

Some examples:

```
true === 1 // false

[] === 0 // false

6 === '6' // false

6 === 6 // true
```

Conditionals: && and | |

```
condition1 && condition2
```

&&: *and* operator evaluates first falsey value, or the last value if there are none

Example:

```
true && 1 // 1
```

```
0 && false // 0
```

```
5 < 6 && 4 + 5 == 1 // false
```

```
condition1 || condition2
```

| |: *or* operator evaluates to first truthy value, or the last value if there are none

Example:

```
true || false // true
```

```
5 == 6 || 1 + 1 == 2 // true
```

```
5 == 6 || 4 == 6 // false
```

Check-in: Conditionals

What would the following print?

```
let lst = [0, 1]
if (lst) {
    console.log('nice')
} else {
    console.log('oops')
}

console.log(lst[0] && lst[1])

console.log(lst[1] == '1')
```

While Loops

```
while (condition) {
    console.log("Condition true");
}
console.log("Condition false");
// Output
// Condition true
// Condition true
// ...
// Condition false
```

- While loops are used to iterate as long as a certain **condition is true**
- When the condition becomes false we exit the loop stop executing the code inside

For Loops

```
for (let i = 0; i < n; i++) {
    console.log("Num: " + i);
}
// Output
// Num: 0
// Num: 1
// ...
// Num: n
```

Really just a fancy while loop.

- Mostly used when you are iterating a certain **number of times**
- Examples...
  - Printing numbers from 0 to n
  - Iterating through every value in a list

for...of loop: arrays

```
let dogs = [“Leopard”, “Tiger”]

for (const dog of dogs) {
    console.log(dog);
}

// Output
// Leopard
// Tiger
```

Short-hand syntax to loop through a array or array-like object

Check-in: Loops

What would the following print?

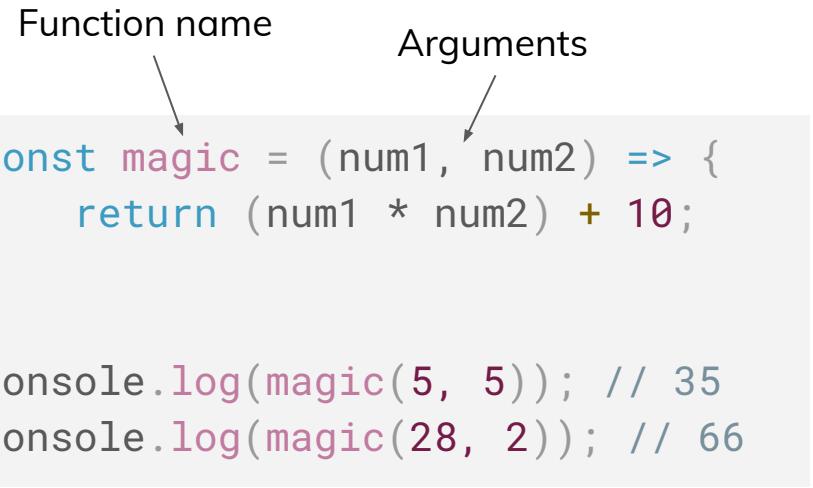
```
let a = 1
while (a < 10) {
    a = a + 1
}
console.log(a)

for (let b = 0; b < 10; b++) {
    a += b
}
console.log(a)
```

Arrow Function Expression =>

```
const magic = (num) => {
    return (num * 2) + 10;
}

console.log(magic(5)); // 20
console.log(magic(28)); // 66
```



Arrow Functions in Use

```
let lst = [1, 5, 10, 15, 20];
```

```
const helper = (element) => {

    return element > 5;

};

console.log(lst.find(helper));

// Output: 10
```

```
console.log(lst.find((element) => { return elt > 5 }));

// Output: 10
```

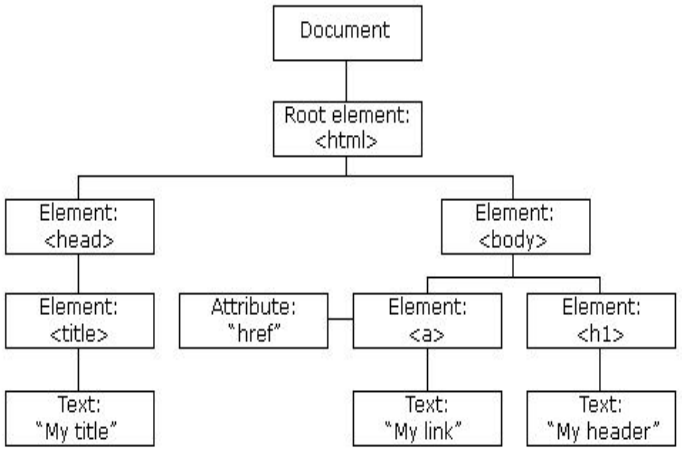
Review: Document Object Model (DOM)

Representation of the webpage elements/HTML

Can be manipulated by JavaScript dynamically!

Document Object Model (DOM)

- How the browser “thinks” about elements on the webpage
- Browser generates DOM tree from HTML
- Hierarchy between parent and children (hence a tree)



Accessing/Selecting Elements

Access elements in the DOM by using:

```
document.getElementById(id); // returns one element
```

Or:

```
document.getElementsByClassName(className);

document.getElementsByTagName(tag);

// returns “arrays” of elements
```

Creating Elements

Create elements in the DOM using... `createElement`

```
document.createElement(tag);
```

For example, here's how to create a `div` element:

```
document.createElement('div');
```

Note: this returns an Element object in JS, but doesn't attach it to the DOM yet...

Adding Elements

Add one DOM element as a child of another using `appendChild`

Syntax

```
parentElement.appendChild(childEl);
```

Now we can create and add our own elements to the page like this:

```
let button = document.createElement('button');

let box = document.getElementById('boxId');

box.appendChild(button);
```

Removing or Replacing Elements

Remove an element from a parent element using `removeChild`

Syntax

```
parent.removeChild(child);
```

Replace an element inside a parent element using `replaceChild`

Syntax

```
parent.replaceChild(newChild, oldChild);
```

Check-in: Modifying the DOM

How do I add a paragraph element to the document body?

A: `document.body.appendElement(document.createChild('p'))`

B: `document.body.createElement(document.appendChild('p'))`

C: `document.body.appendChild(document.createElement('p'))`

D: `document.body.createChild(document.appendElement('p'))`

DOM Element Properties

Here are some useful properties of DOM elements:

- **`.className`** returns the class property of the element
  - `<span class="someClass"></span>`
- **`.id`** returns the id of the element
  - `<span id="someId"></span>`
- **`.textContent`** returns the text contained within the element
  - `<span>"Element text"</span>`
- **`.style`** returns the style property of the element
  - `<span style="color: red"></span>`
- **`.value`** returns the value of input elements
  - `<input value={someValue}>`

DOM: Events

More at https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Events

What is an event?

Something that happens on the webpage, that you can react to

Example Events

Most elements: “**click**”, “**mouseover**”, “**mouseout**”

Inputs: “**change**”

Full reference: https://developer.mozilla.org/en-US/docs/Web/Events

Handling Events

Event handlers are used to trigger **actions** (execute functions) when an **event** occurs on a **target** (element), e.g. when a user clicks a button.

Syntax

```
target.addEventListener('eventName', functionName);
```

```
let button = document.getElementById('btn');

function doAction() {
    console.log('Button clicked!');
}

button.addEventListener('click', doAction);
```

Demo from Lab 3

```
const btn = document.getElementById("btn")

function whenClicked(event) {
    alert("hello world!")

    console.log(event.target)

    event.target.textContent = "can't click anymore"

    btn.removeEventListener("click", whenClicked)
}

btn.addEventListener("click", whenClicked)
```

Removing Event Handlers

Syntax

```
target.removeEventListener('eventName', functionName);
```

```
let button = document.getElementById('btn');

function doAction() {
    console.log('Button clicked!');
    button.removeEventListener('click', doAction);
}

button.addEventListener('click', doAction);
```

Event Object

The event handler function takes one parameter, which contains info about the event

event.target is the element that the event is fired on

```
let button = document.getElementById('btn');

function doAction(event) {
    console.log('Button clicked!');
    event.target.style.backgroundColor = rndCol;
}

button.addEventListener('click', doAction);
```

Example: Input Element

- Listening to the “**change**” event
- Using **event.target.value** to get the value of the input

[stop recording]

Secret word:

https://forms.gle/VZZYR7R9nCQySgAP9