JavaScript and the DOM

Class 18

Checkboxes

- `<input type="checkbox"` defines a multiple-choice input that allows multiple selections
- a single thing with multiple values requires an array to store the values so the name attribute must have square brackets

```
<form action="foo.php" method="post">
  <input type="checkbox" name="colors[]" value="red" id="color_red" />
  <label for="color_red">Red</label>

  <input type="checkbox" name="colors[]" value="green" id="color_green" />
  <label for="color_red">Green</label>

  <input type="checkbox" name="colors[]" value="blue" id="color_blue" />
  <label for="color_red">Blue</label>
  <input type="submit" value="Submit">
</form>
```

Checkboxes

- in PHP, checkbox value are in `$_POST` like normal
- but `$_POST['colors']` is an array
- if no checkbox is checked, then `$_POST['colors']` is not set
- if only one checkbox is checked, `$_POST['colors']` is still an array

Calculate a Tip

tip calculator example

Calculate a Tip

tip calculator example

a callback function attached to an element can refer to the element by this

Style

the DOM includes style information for each element

rick example

Separation of Duties

there's a big problem with the rick example

just as we separate style and content

we must also separate style and behavior

JS is supposed to be about behavior, but rick.js is full of style

Style, Take 2

all the style should be in CSS, none in HTML or JS

instead of hard-coding a style in JS, we can style unobtrusively by giving an element a class that already has a pre-defined style in the CSS file

style example

ClassList Methods

```
element.classList.add("classname")
element.classList.contains("classname")
element.classList.remove("classname")
element.classList.toggle("classname")
```

The DOM Tree

the DOM is a hierarchical framework

the DOM has three kinds of nodes

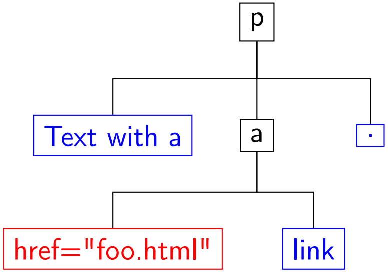
- an element node represents an HTML tag
- it can have children and attributes

- a text node represents inline text inside a block element
- it always occurs as a child within an element node
- it cannot have any children or attributes

- an attribute node represents an attribute name-value pair in an element's opening tag
- it is connected to the element node but is not a child of that node.
- it cannot have any children or attributes.

DOM Tree Example

```
<p>Text with a <a href="foo.html">link</a>.</p>
```



DOM Nodes

- attribute nodes are not children and are not traversed when navigating the DOM
- text nodes are children and must be considered
- element nodes are usually what we want to work with

- text and element nodes are considered navigational nodes

Traversal

every navigational node has properties that connect it to those around it

parentNode	is the element one level up that contains this node (null if none exists)
previousSibling	is the the element that comes before this one on the same level (null if there is none)
nextSibling	is the the element that comes after this one on the same level
firstChild	is the leftmost element that this node contains, one level down
lastChild	is the rightmost element that this node contains, one level down
childNodes	is a list of all of this node's children, one level down (empty if there are none)

Navigation Example

```
<p id="myp">Text with a <a href="foo.html">link</a>.</p>
```

```
document.getElementById("myp").
   firstChild.nextSibling.firstChild.
   textContent = "Fred";
```

produces:

```
<p id="myp">Text with a <a href="foo.html">Fred</a>.</p>
```

Navigation

this works, but is tedious and error-prone

for example, how many children does the div below have?

```
<div id="foo">
  <p id="bar">
    This is a paragraph of text with a
    <a href="anotherpage.html">link</a>
  </p>
</div>
```

Selecting Groups of Elements

the DOM has several powerful methods of selecting groups of elements

getElementById(id)
: we've seen before returns the DOM element with the given HTML ID

getElementsByTagName(tag)
: returns a list of DOM elements with the given HTML element tag, e.g., div

getElementsByName(name)
: returns a list of DOM elements with the given name attribute, e.g., all radio buttons with this name. Returns element nodes, not attribute nodes.

querySelector(selector)
: returns the first element that matches the given CSS-type selector string

querySelectorAll(selector)
: returns a list of all the elements that match the given CSS-type selector string

Tag Name Example

```
// change the class of all paragraphs in the document
let allParagraphs = document.getElementsByTagName("p");
for (let i = 0; i < allParagraphs.length; i++)
{
  allParagraphs[i].classList.toggle("specialStatus");
}
```

Selectors By Level

except for `getElementById`, all the selectors work at any level

```
<p>Not affected</p>
<footer>
  <p>&copy; 2021</p>
  <p>All rights reserved</p>
</footer>
```

```
let footer = document.getElementsByTagName("footer")[0];
let footerps = footer.getElementsByTagName("p");
for (let i = 0; i < footerps.length; i++)
{
  footerps[i].classList.add("specialStatus");
}
```

querySelectorAll

the previous example could also be accomplished with querySelectorAll:

```
let footerps = document.querySelectorAll("footer p");
for (let i = 0; i < footerps.length; i++)
{
  footerps[i].classList.add("specialStatus");
}
```

Adding and Removing Nodes

sometimes we need to add an element to or remove an element from a document

```
let new_heading = document.createElement("h2");
new_heading.classList.add("alert");
new_heading.innerHTML = "This is an alert heading";
```

Adding an Element Node

but just creating the element doesn't make it appear on the page

appendChild(node)
: places the given node at the end of a node's child list

insertBefore(newNode, existingNode)
: places the given new node in this node's child list before the given existing node

removeChild(node)
: removes the given node form this node's child list

replaceChild(newNode, oldNode)
: replaces the given old node with the new node

add paragraphs example