EECS 368
Programming Language Paradigms

David O. Johnson

Fall 2022

Reminders

- Assignment 2 due: 11:59 PM, Monday, September 19
- Assignment 3 due: 11:59 PM, Monday, October 3

Any Questions?

In-Class Problem Solution

- 9-(9-14) In-Class Problem Solution.pptx

Any Questions?

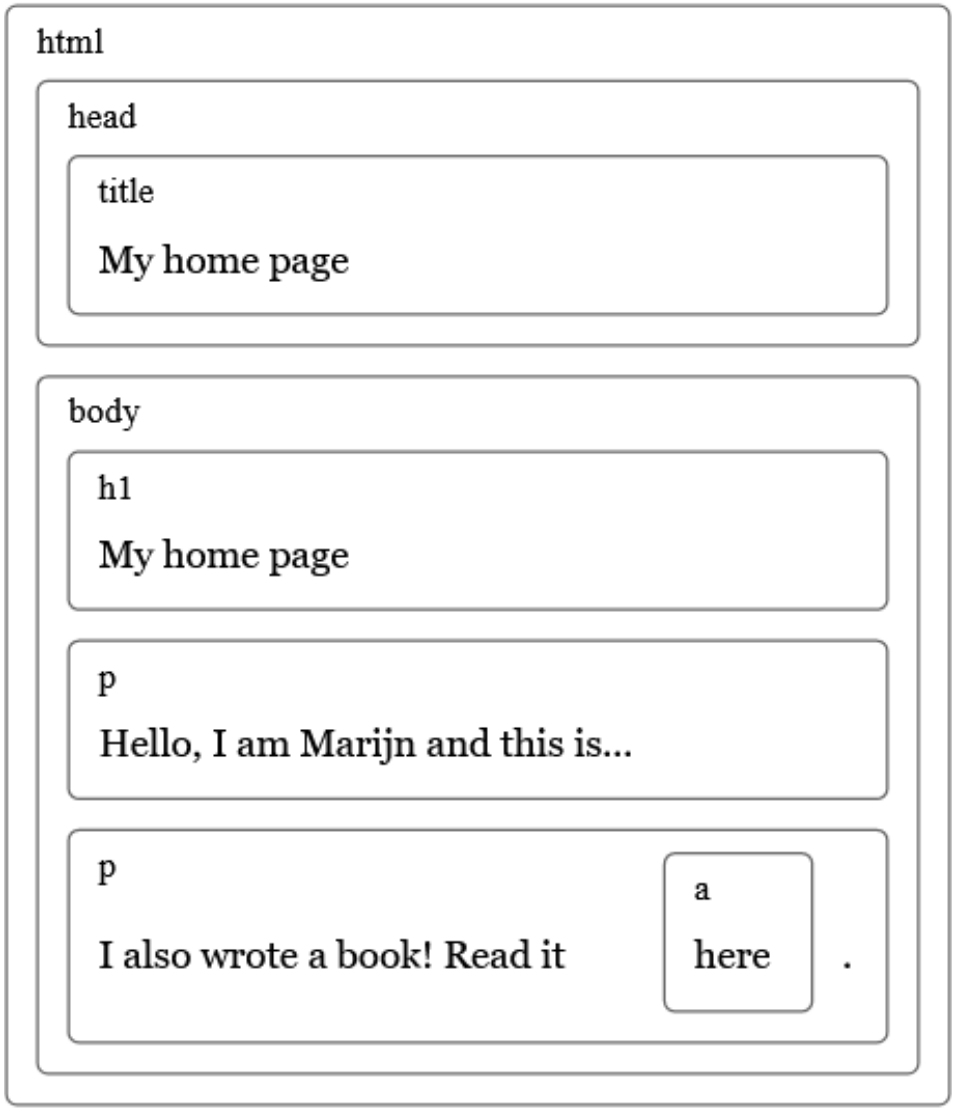
Document Object Model (DOM)

- You can imagine an HTML document as a nested set of boxes.
- Tags such as <body> and </body> enclose other tags, which in turn contain other tags or text.
- Here's an example document:

```
<!doctype html>
<html>
  <head>
    <title>My home page</title>
  </head>
  <body>
    <h1>My home page</h1>
    <p>Hello, I am Marijn and this is my home page.</p>
    <p>I also wrote a book! Read it
      <a href="http://eloquentjavascript.net">here</a>.</p>
  </body>
</html>
```

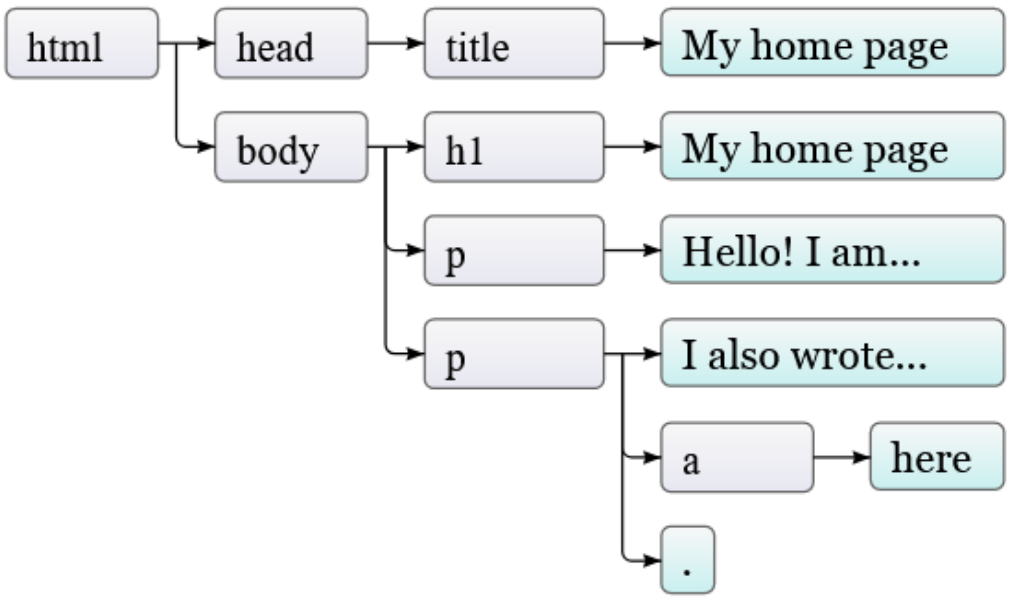
Document Object Model (DOM)

- The data structure the browser uses to represent the document follows this shape.
- For each box, there is an object, which we can interact with to find out things such as what HTML tag it represents and which boxes and text it contains.
- This representation is called the Document Object Model, or DOM for short.
- The JavaScript global binding, document, gives us access to these objects.
- Its documentElement property refers to the object representing the <html> tag.
- Since every HTML document has a head and a body, …
- … it also has head and body properties, pointing at those elements.



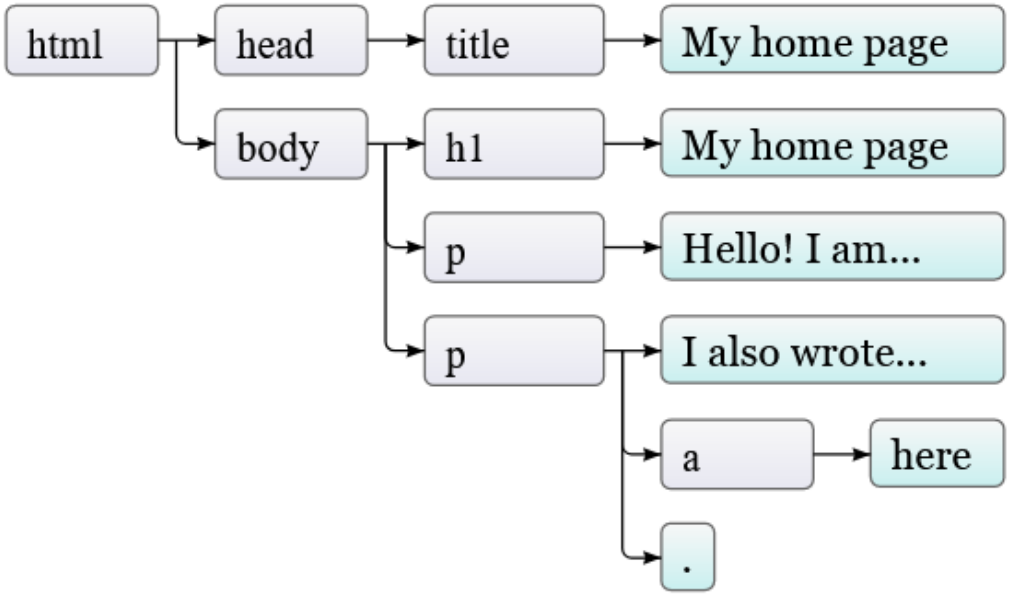
DOM Tree

- The Document Object Model (DOM) is a cross-platform and language-independent interface that treats an HTML document as a tree structure.
- Each node is an object representing a part of the document.
- The DOM represents a document with a logical tree.
- Each branch of the tree ends in a node, and each node contains objects.



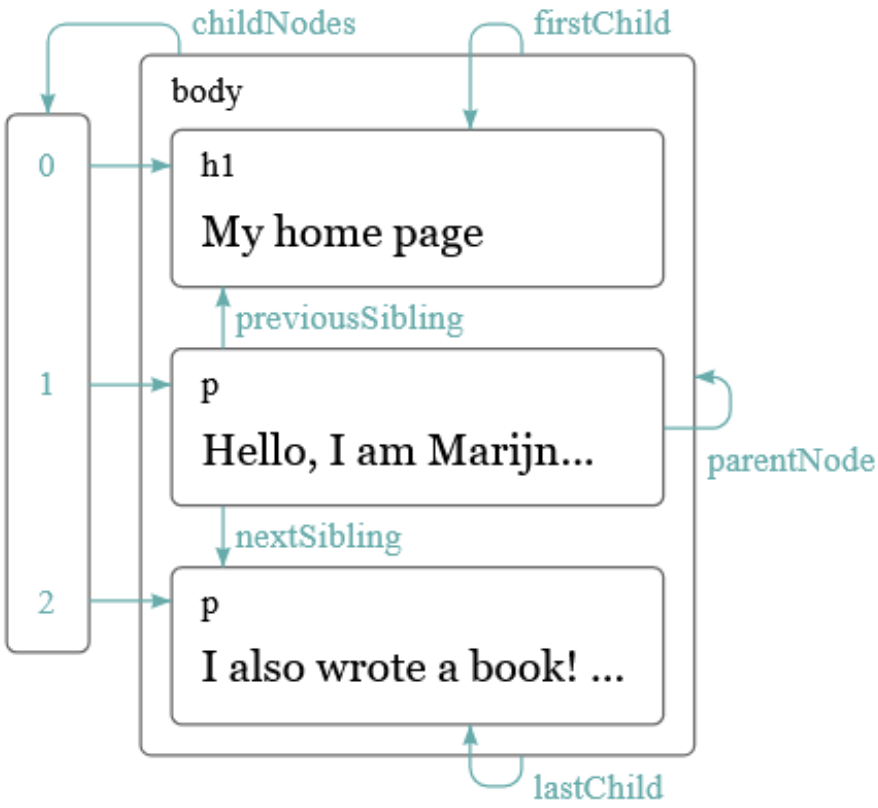
DOM Tree

- Each DOM node object has a nodeType property, which contains a code (number) that identifies 1 of 3 types of nodes:
- Element nodes:
  - HTML tags
  - Code 1 (which is also defined as the constant property Node.ELEMENT_NODE)
- Text nodes:
  - A section of text in the document
  - Code 3 (Node.TEXT_NODE)
- Comment Nodes:
  - Comments
  - Code 8 (Node.COMMENT_NODE)



DOM Tree Properties

- DOM nodes contain a wealth of links to other nearby nodes.
- Every node has a parentNode property that points to the node it is part of, if any.
- Likewise, every element node (node type 1) has a childNodes property that points to an array-like object holding its children.
- firstChild and lastChild properties point to the first and last child elements or have the value null for nodes without children.
- previousSibling and nextSibling point to adjacent nodes, which are nodes with the same parent that appear immediately before or after the node itself.
- For a first child, previousSibling will be null, and for a last child, nextSibling will be null.
- The children property is like childNodes but contains only element (type 1) children, not other types of child nodes.
- This can be useful when you aren't interested in text nodes.



Any Questions?

Changing the DOM

The DOM tree can be changed by:

- The remove method removes a node from its current parent node.
- The appendChild method adds a node to the end of the list of children.
- The insertBefore method inserts the node given as the first argument before the node given as the second argument.
- The replaceChild method replaces a child node with another one.
- All operations that insert a node somewhere will, as a side effect, cause it to be removed from its current position (if it has one).
- For example:

```
<script>
  let paragraphs = document.body.getElementsByTagName("p");
  document.body.insertBefore(paragraphs[2], paragraphs[0]);
</script>
```

paragraph[0]	One	Three
paragraph[1]	Two	One
paragraph[2]	Three	Two

Creating Element Nodes

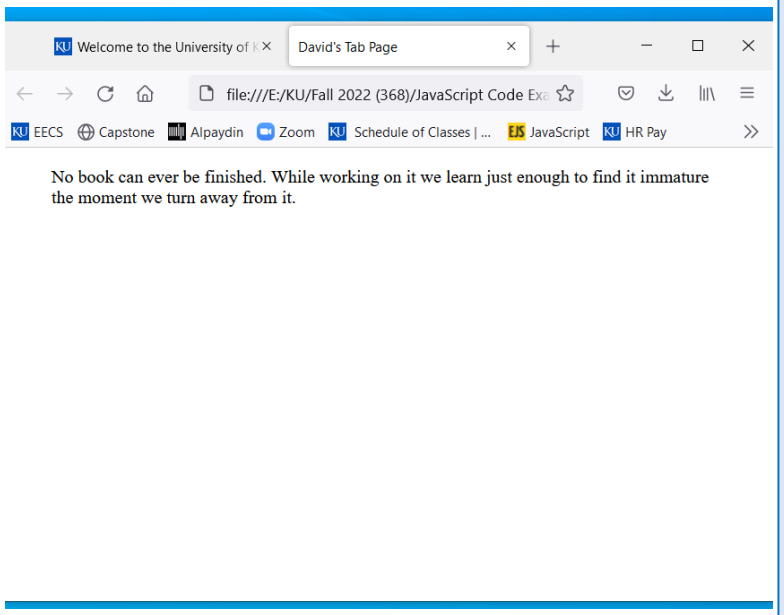
- To create element nodes, you can use the document.createElement method.
- This method takes a tag name and returns a new empty node of the given type.
- This example defines a utility elt, which creates an element node and treats the rest of its arguments as children to that node.
- This function is then used to add an attribution to a quote.

```
<blockquote id="quote">
  No book can ever be finished. While working on it we learn
  just enough to find it immature the moment we turn away
  from it.
</blockquote>

<script>
  function elt(type, ...children) {
    let node = document.createElement(type);
    for (let child of children) {
      if (typeof child != "string") node.appendChild(child);
      else node.appendChild(document.createTextNode(child));
    }
    return node;
  }

  document.getElementById("quote").appendChild(
    elt("footer", "—",
        elt("strong", "Karl Popper"),
        ", preface to the second edition of ",
        elt("em", "The Open Society and Its Enemies"),
        ", 1950"));
</script>
```

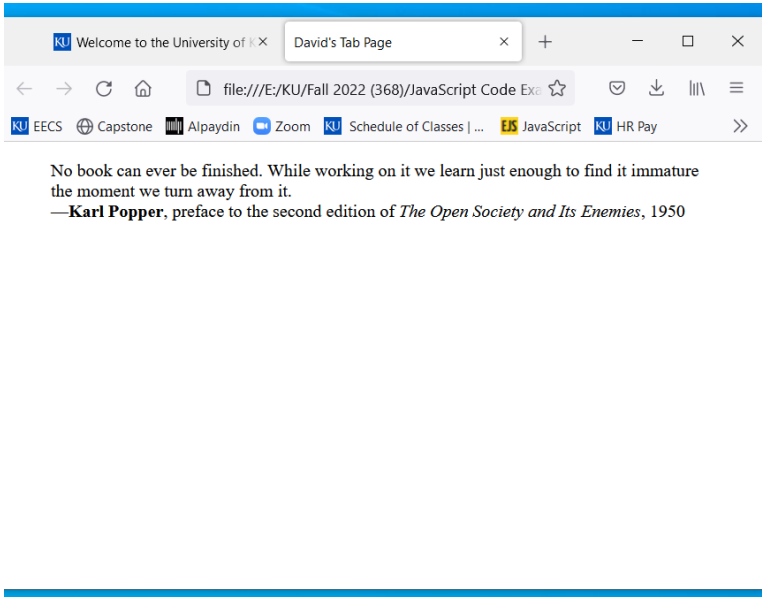
Creating Element Nodes




```
<blockquote id="quote">
  No book can ever be finished. While working on it we learn
  just enough to find it immature the moment we turn away
  from it.
</blockquote>
<!--
<script>
  function elt(type, ...children) {
    let node = document.createElement(type);
    for (let child of children) {
      if (typeof child != "string") node.appendChild(child);
      else node.appendChild(document.createTextNode(child));
    }
    return node;
  }

  document.getElementById("quote").appendChild(
    elt("footer", "—",
        elt("strong", "Karl Popper"),
        ", preface to the second edition of ",
        elt("em", "The Open Society and Its Enemies"),
        ", 1950"));
</script>
-->
```

Creating Element Nodes




```
<blockquote id="quote">
  No book can ever be finished. While working on it we learn
  just enough to find it immature the moment we turn away
  from it.
</blockquote>

<script>
  function elt(type, ...children) {
    let node = document.createElement(type);
    for (let child of children) {
      if (typeof child != "string") node.appendChild(child);
      else node.appendChild(document.createTextNode(child));
    }
    return node;
  }

  document.getElementById("quote").appendChild(
    elt("footer", "—",
        elt("strong", "Karl Popper"),
        ", preface to the second edition of ",
        elt("em", "The Open Society and Its Enemies"),
        ", 1950"));
</script>
```

Any Questions?

Creating Text Nodes

- To create text nodes, you can use the document.createTextNode method.
- This method creates a text node that we can insert into the document to make it show up on the screen.
- This example replaces all images (<img> tags) in the document with the text held in their alt attributes, which specifies an alternative textual representation of the image.
- This involves not only removing the images but adding a new text node to replace them.

```
<p>The <img src="img/cat.png" alt="Cat"> in the
  <img src="img/hat.png" alt="Hat">.</p>

<p><button onclick="replaceImages()">Replace</button></p>

<script>
  function replaceImages() {
    let images = document.body.getElementsByTagName("img");
    for (let i = images.length - 1; i >= 0; i--) {
      let image = images[i];
      if (image.alt) {
        let text = document.createTextNode(image.alt);
        image.parentNode.replaceChild(text, image);
      }
    }
  }
</script>
```

Creating Text Nodes

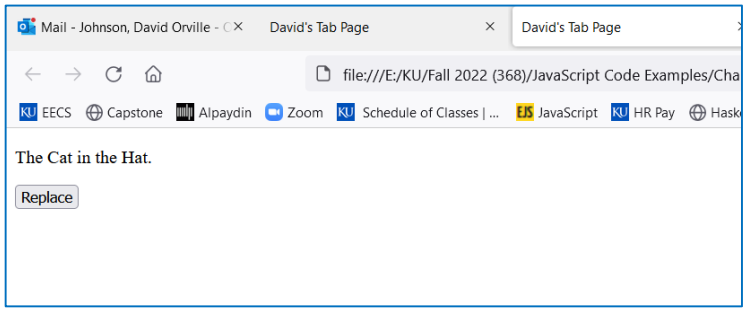
- The loop that goes over the images starts at the end of the list.
- This is necessary because the node list returned by a method like getElementsByTagName (or a property like childNodes) is live.
- That is, it is updated as the document changes.
- If we started from the front, removing the first image would cause the list to lose its first element so that the second time the loop repeats, where i is 1, it would stop because the length of the collection is now also 1.

```
<p>The <img src="img/cat.png" alt="Cat"> in the
  <img src="img/hat.png" alt="Hat">.</p>

<p><button onclick="replaceImages()">Replace</button></p>

<script>
  function replaceImages() {
    let images = document.body.getElementsByTagName("img");
    for (let i = images.length - 1; i >= 0; i--) {
      let image = images[i];
      if (image.alt) {
        let text = document.createTextNode(image.alt);
        image.parentNode.replaceChild(text, image);
      }
    }
  }
</script>
```

Creating Text Nodes




```
<p>The <img src="img/cat.png" alt="Cat"> in the
  <img src="img/hat.png" alt="Hat">.</p>

<p><button onclick="replaceImages()">Replace</button></p>

<script>
  function replaceImages() {
    let images = document.body.getElementsByTagName("img");
    for (let i = images.length - 1; i >= 0; i--) {
      let image = images[i];
      if (image.alt) {
        let text = document.createTextNode(image.alt);
        image.parentNode.replaceChild(text, image);
      }
    }
  }
</script>
```

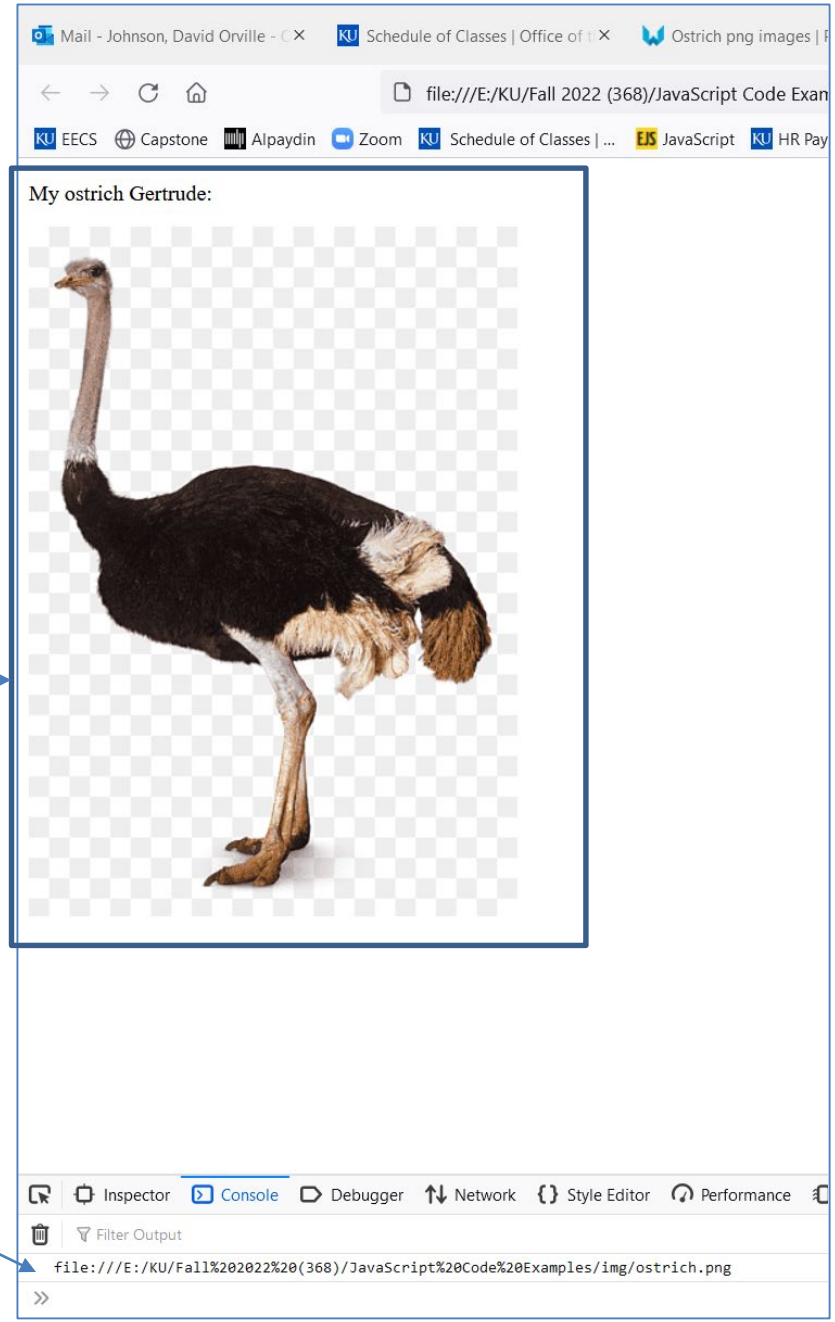
Any Questions?

Finding Elements (getElementById)

To find a specific single node, you can give it an id attribute and use document.getElementById.

```
<p>My ostrich Gertrude:</p>
<p><img id="gertrude" src="img/ostrich.png"></p>
```

```
<script>
  let ostrich = document.getElementById("gertrude");
  console.log(ostrich.src);
</script>
```



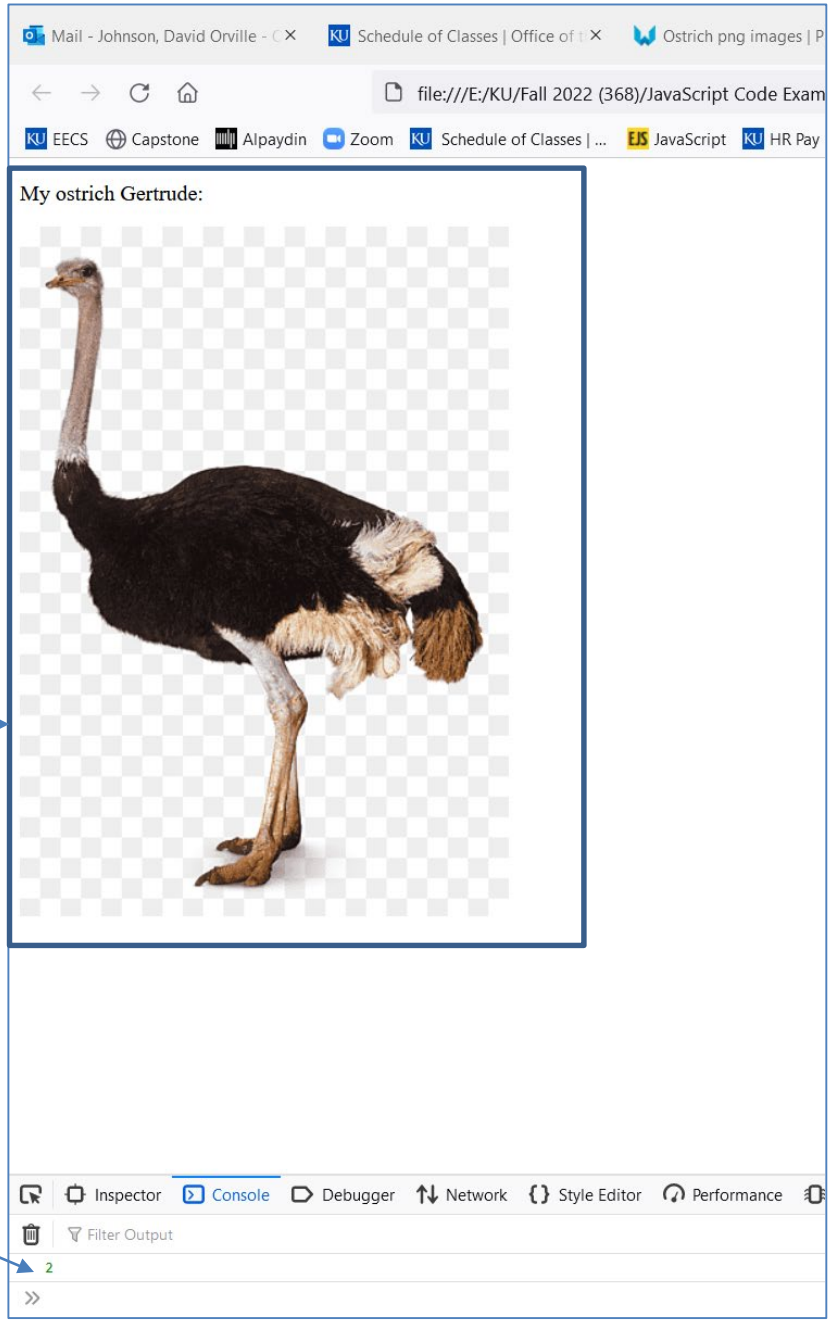
Finding Elements (getElementsByTagName)

The getElementsByTagName method collects all elements with the given tag name that are descendants of a node and returns them as an array-like object.

```
<p>My ostrich Gertrude:</p>
<p><img id="gertrude" src="img/ostrich.png"></p>
```

```
<script>
 let link = document.body.getElementsByTagName("p");
 console.log(link.length);
</script>
```

- link is an HTMLCollection
- HTMLCollection.length returns the number of items in the collection.

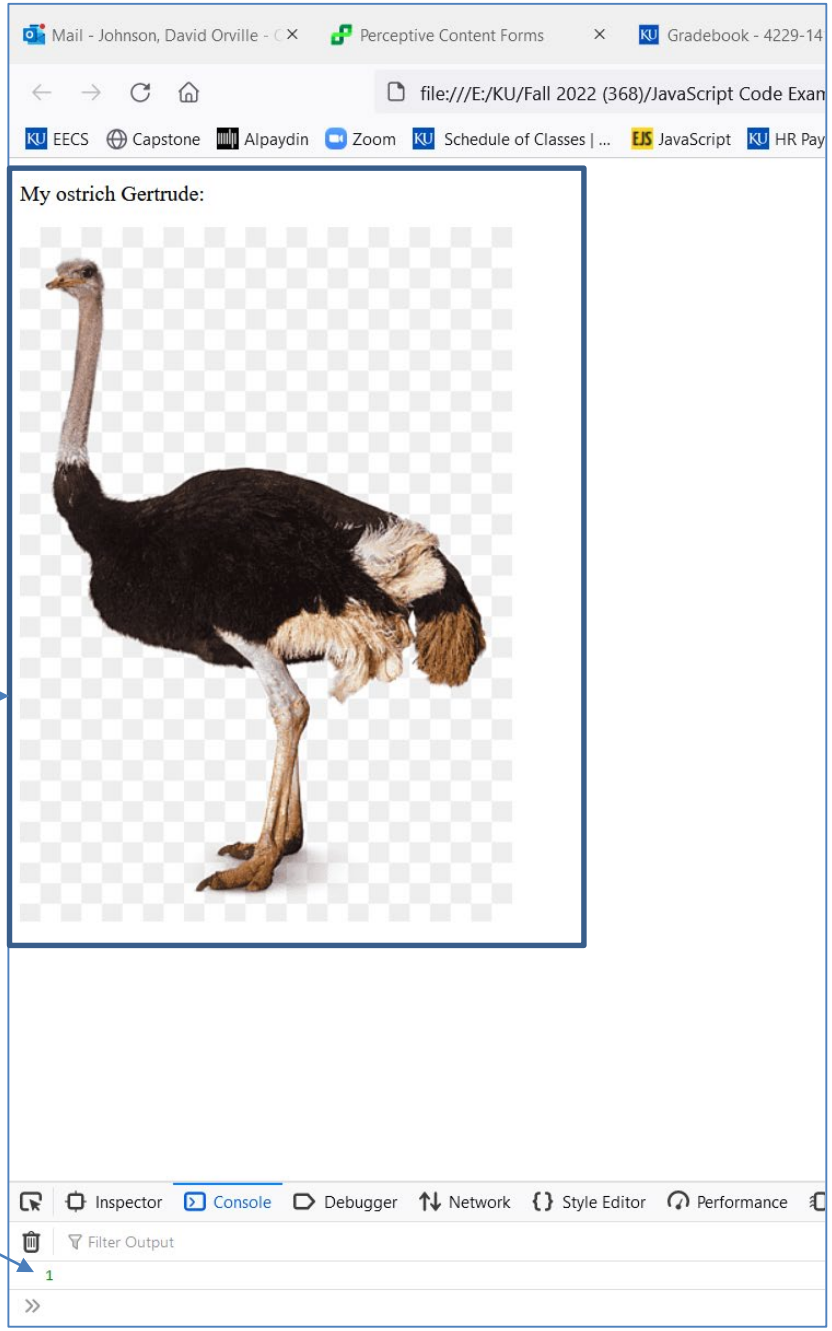


Finding Elements (getElementsByClassName)

The getElementsByClassName searches through the contents of an element node and retrieves all elements that have the given string in their class attribute and returns them as an array-like object.

```
<p>My ostrich Gertrude:</p>
<p class="test"><img id="gertrude“ src="img/ostrich.png"></p>

<script>
 let link = document.body.getElementsByClassName("test");
 console.log(link.length);
</script>
```



- link is an HTMLCollection
- HTMLCollection.length returns the number of items in the collection.

Any Questions?

Query Selectors (querySelectorAll)

- The querySelectorAll method, which is defined both on the document object and on element nodes, takes a selector string and returns a NodeList containing all the elements that it matches.
- Unlike methods such as getElementsByTagName, the object returned by querySelectorAll is not live.
- It won't change when you change the document.
- It is still not a real array, though, so you still need to call Array.from if you want to treat it like one.

```
<p>And if you go chasing
  <span class="animal">rabbits</span></p>
<p>And you know you're going to fall</p>
<p>Tell 'em a <span class="character">hookah smoking
  <span class="animal">caterpillar</span></span></p>
<p>Has given you the call</p>

<script>
  function count(selector) {
    return document.querySelectorAll(selector).length;
  }
  console.log(count("p"));          // All <p> elements
  // → 4
  console.log(count(".animal"));     // Class animal
  // → 2
  console.log(count("p .animal"));   // Animal inside of <p>
  // → 2
  console.log(count("p > .animal")); // Direct child of <p>
  // → 1
</script>
```

Query Selectors (querySelectorAll)

- The querySelectorAll method, which is defined both on the document object and on element nodes, takes a selector string and returns a NodeList containing all the elements that it matches.
- Unlike methods such as getElementsByTagName, the object returned by querySelectorAll is not live.
- It won't change when you change the document.
- It is still not a real array, though, so you still need to call Array.from if you want to treat it like one.

```
<p>And if you go chasing
  <span class="animal">rabbits</span></p>
<p>And you know you're going to fall</p>
<p>Tell 'em a <span class="character">hookah smoking
  <span class="animal">caterpillar</span></span></p>
<p>Has given you the call</p>

<script>
  function count(selector) {
    return document.querySelectorAll(selector).length;
  }
  console.log(count("p"));          // All <p> elements
  // → 4
  console.log(count(".animal"));     // Class animal
  // → 2
  console.log(count("p .animal"));   // Animal inside of <p>
  // → 2
  console.log(count("p > .animal")); // Direct child of <p>
  // → 1
</script>
```

Query Selectors (querySelectorAll)

- The querySelectorAll method, which is defined both on the document object and on element nodes, takes a selector string and returns a NodeList containing all the elements that it matches.
- Unlike methods such as getElementsByTagName, the object returned by querySelectorAll is not live.
- It won't change when you change the document.
- It is still not a real array, though, so you still need to call Array.from if you want to treat it like one.

```
<p>And if you go chasing
  <span class="animal">rabbits</span></p>
<p>And you know you're going to fall</p>
<p>Tell 'em a <span class="character">hookah smoking
  <span class="animal">caterpillar</span></span></p>
<p>Has given you the call</p>

<script>
  function count(selector) {
    return document.querySelectorAll(selector).length;
  }
  console.log(count("p"));          // All <p> elements
  // → 4
  console.log(count(".animal"));     // Class animal
  // → 2
  console.log(count("p .animal"));   // Animal inside of <p>
  // → 2
  console.log(count("p > .animal")); // Direct child of <p>
  // → 1
</script>
```

Query Selectors (querySelectorAll)

- The querySelectorAll method, which is defined both on the document object and on element nodes, takes a selector string and returns a NodeList containing all the elements that it matches.
- Unlike methods such as getElementsByTagName, the object returned by querySelectorAll is not live.
- It won't change when you change the document.
- It is still not a real array, though, so you still need to call Array.from if you want to treat it like one.

```
<p>And if you go chasing
  <span class="animal">rabbits</span></p>
<p>And you know you're going to fall</p>
<p>Tell 'em a <span class="character">hookah smoking
  <span class="animal">caterpillar</span></span></p>
<p>Has given you the call</p>

<script>
  function count(selector) {
    return document.querySelectorAll(selector).length;
  }
  console.log(count("p"));          // All <p> elements
  // → 4
  console.log(count(".animal"));     // Class animal
  // → 2
  console.log(count("p .animal"));   // Animal inside of <p>
  // → 2
  console.log(count("p > .animal")); // Direct child of <p>
  // → 1
</script>
```

Query Selectors (querySelector)

- The querySelector method (without the All part) works in a similar way.
- This one is useful if you want a specific, single element.
- It will return only the first matching element or null when no element matches.

Any Questions?

Summary

- JavaScript programs may inspect and interfere with the document that the browser is displaying through a data structure called the Document Object Model (DOM).
- The DOM is a cross-platform and language-independent interface that treats an HTML document as a tree structure.
- The tree consists of element (i.e., HTML tag), text, and comment nodes.
- The DOM tree can be changed by removing, appending, inserting, and replacing nodes with JavaScript methods.
- JavaScript methods can find nodes by accessing various parent, child, and sibling properties of a node.
- JavaScript methods can find nodes by HTML tag and HTML id.
- The JavaScript querySelectorAll method takes a selector string and returns a NodeList containing all the elements that it matches.
- The querySelector method (without the All part) works in a similar way, only returning the first matching element.

Any Questions?

In-Class Problem

1. Describe what the function byTagName is doing.
2. Describe what the function explore is doing.
3. What is the output for the following:

```
console.log(byTagName(document.body, "h1").length);

console.log(byTagName(document.body, "span").length);

let para = document.querySelector("p");
console.log(byTagName(para, "span").length);
```

```
<h1>Heading with a <span>span</span> element.</h1>
<p>A paragraph with <span>one</span>, <span>two</span>
 spans.</p>

<script>
 function byTagName(node, tagName) {
   let found = [];
   tagName = tagName.toUpperCase();

   function explore(node) {
     for (let i = 0; i < node.childNodes.length; i++) {
       let child = node.childNodes[i];
       if (child.nodeType == document.ELEMENT_NODE) {
         if (child.nodeName == tagName) found.push(child);
         explore(child);
       }
     }
   }

   explore(node);
   return found;
 }
</script>
```