Week 8

Name Circle

- Favorite halloween costume you saw this weekend

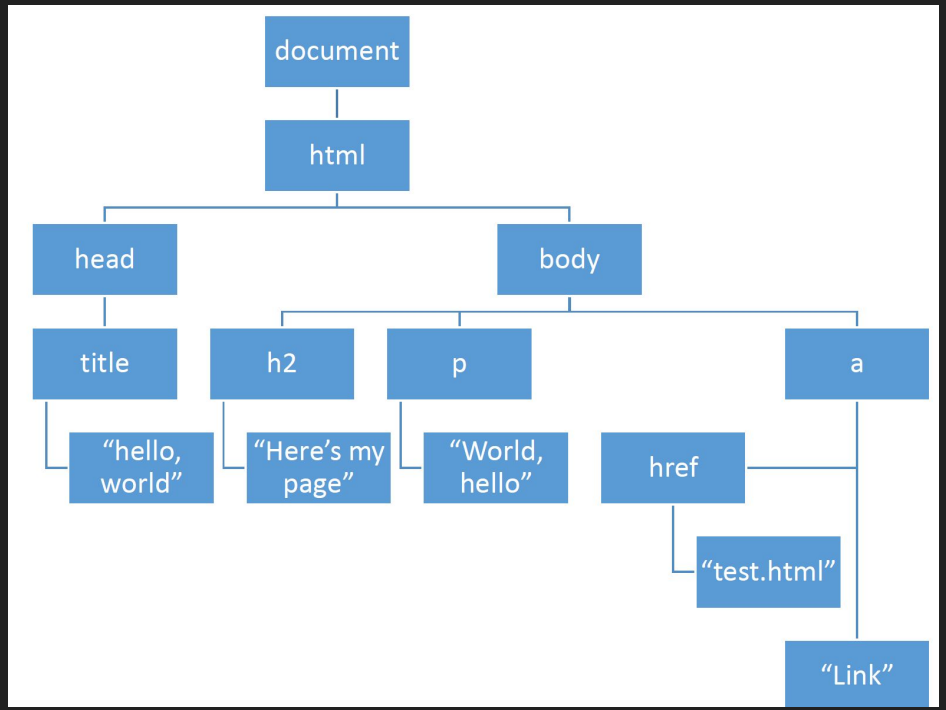
Preview

- This Week + Next Week: Normal schedule
- Test (11/15-11/21)
- Final Project
  - 11/14: Proposal
  - 11/29: Status Report
  - 12/7: Submission

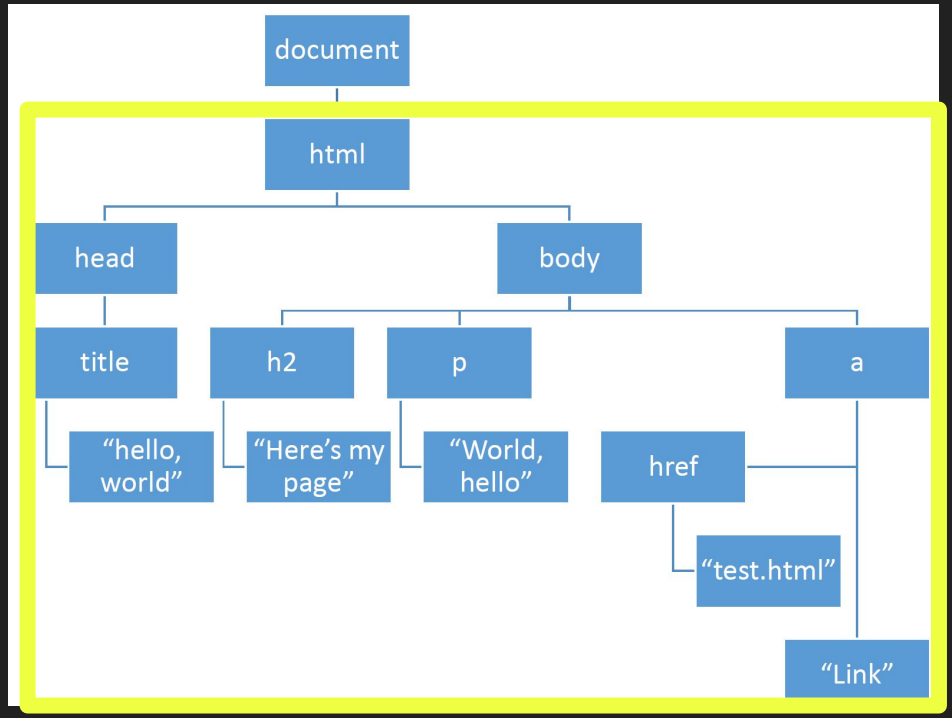
Content Recap

DOM

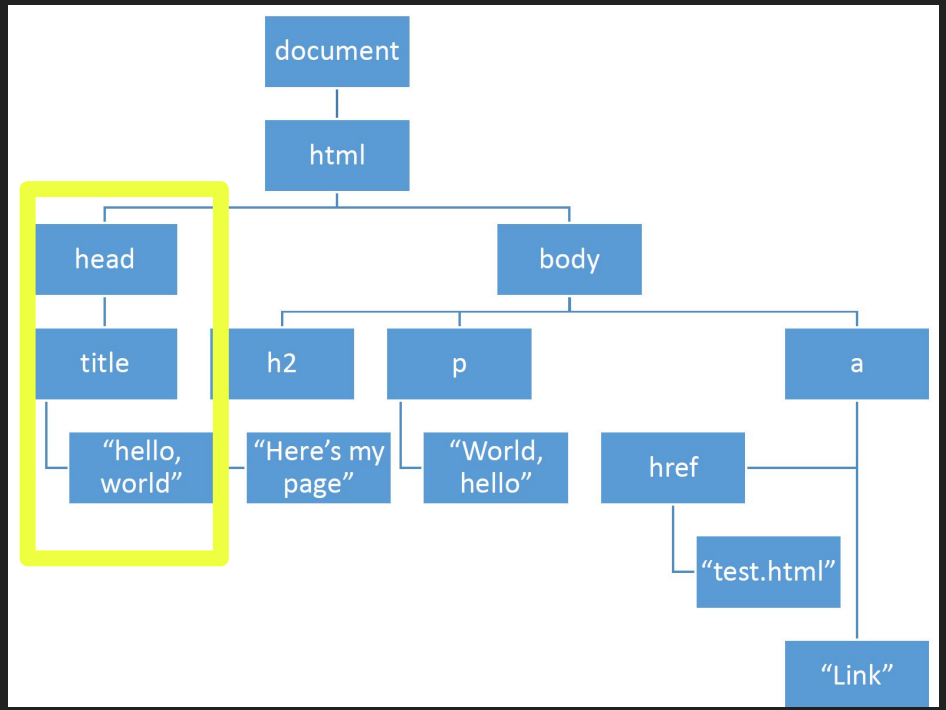
```
<html>
   <head>
      <title>Hello, world</title>
   </head>
   <body>
      <h2>Here's my page</h2>
      <p>World, hello</p>
      <a href="test.html">Link</a>
   </body>
</html>
```



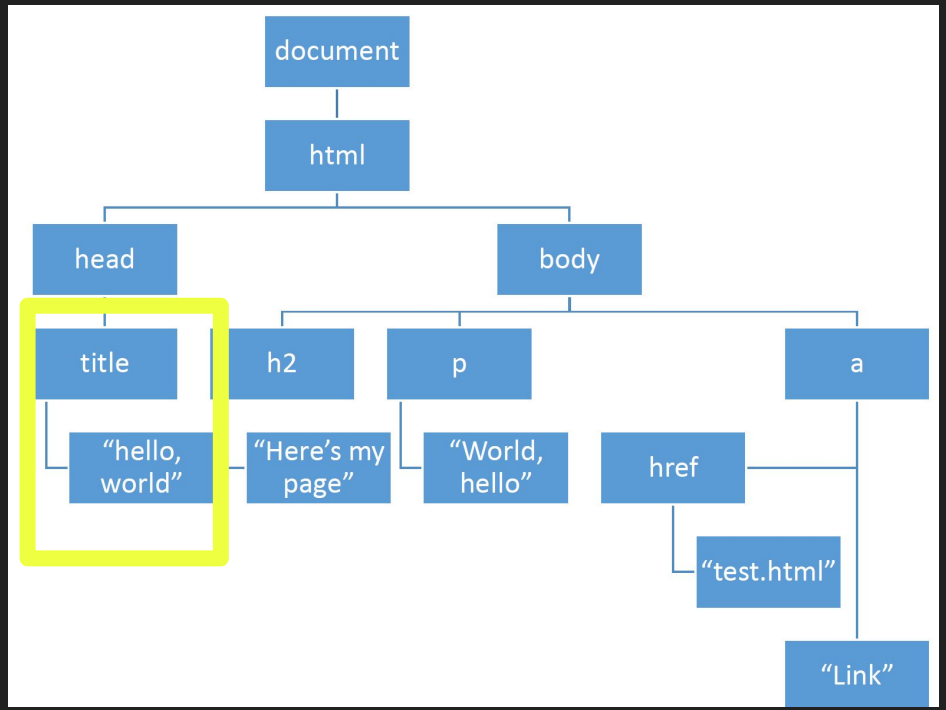

```
<html>
     <head>
          <title>Hello, world</title>
     </head>
     <body>
          <h2>Here's my page</h2>
          <p>World, hello</p>
          <a href="test.html">Link</a>
     </body>
</html>
```



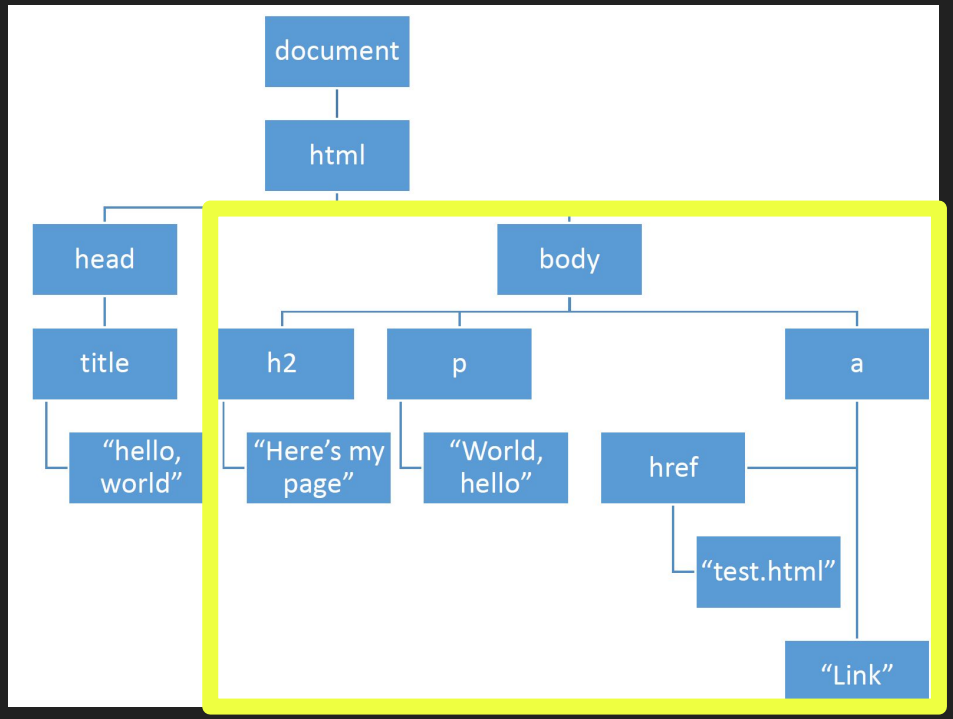

```
<html>
     <head>
          <title>Hello, world</title>
     </head>
     <body>
          <h2>Here's my page</h2>
          <p>World, hello</p>
          <a href="test.html">Link</a>
     </body>
</html>
```




```
<html>
     <head>
          <title>Hello, world</title>
     </head>
     <body>
          <h2>Here's my page</h2>
          <p>World, hello</p>
          <a href="test.html">Link</a>
     </body>
</html>
```




```
<html>
    <head>
        <title>Hello, world</title>
    </head>
    <body>
        <h2>Here's my page</h2>
        <p>World, hello</p>
        <a href="test.html">Link</a>
    </body>
</html>
```




```
<html>
     <head>
          <title>Hello, world</title>
     </head>
     <body>
          <h2>Here's my page</h2>
          <p>World, hello</p>
          <a href="test.html">Link</a>
     </body>
</html>
```

- Some examples of DOM *properties* include:

PROPERTY	DESCRIPTION
`innerHTML`	Holds the HTML inside a set of HTML tags.
`nodeName`	The name of an HTML element or element's attribute.
`id`	The "`id`" attribute of an HTML element
`parentNode`	A reference to the node one level up in the DOM.
`childNodes`	An array of references to the nodes one level down in the DOM.
`attributes`	An array of attributes of an HTML element.
`style`	An object encapsulating the CSS/HTML styling of an element.

- Some examples of DOM *methods* include:

METHOD	DESCRIPTION
`getElementById(id)`	Gets the element with a given ID below this point in the DOM.
`getElementsByTagName(tag)`	Gets all elements with the given tag below this point in the DOM.
`appendChild(node)`	Add the given node to the DOM below this point.
`removeChild(node)`	Remove the specified child node from the DOM.

- Some examples of DOM *methods* include:

METHOD	DESCRIPTION
`getElementById(id)`	Gets the element with a given ID below this point in the DOM.
`getElementsByTagName(tag)`	Gets all elements with the given tag below this point in the DOM.
`appendChild(node)`	Add the given node to the DOM below this point.
`removeChild(node)`	Remove the specified child node from the DOM.

- JavaScript to set the background color of the HTML element with ID `colorDiv` to be green:

```
document.getElementById("colorDiv").style.backgroundColor = "green";
```

- jQuery to set the background color of the HTML element with ID `colorDiv` to be green:

```
$("colorDiv").css("background-color", "green");
```

- jQuery to set the background color of the HTML element with ID `colorDiv` to be green:

```
$("colorDiv").css("background-color", "green");
```

- More information about the various potential use cases for jQuery at

`https://api.jquery.com`

JavaScript

- **JavaScript**, like Python, is much newer than C (1995 vs. 1972), but is also very heavily inspired by it.

- To start writing JavaScript, open a file with the .js file extension.

- Unlike Python which runs *server-side*, most JavaScript applications run *client-side*, on your own machine. (Modifications to JavaScript, such as the popular Node.js, is server-side.

- JavaScript, HTML, and CSS together basically comprise the backbone of the internet.

- Much like with CSS and `<style>` tags, you can directly write JS between `<script>` tags, but you can also link external JavaScript files (which is probably the preferred approach!) by way of the `src` attribute of `<script>` tags.

- Variables in JavaScript do not require a type specifier, and do not need to be declared in advance. But there is a special keyword for introducing them.

- Variables in JavaScript do not require a type specifier, and do not need to be declared in advance. But there is a special keyword for introducing them.

```
let x = 44;
```

- Variables in JavaScript do not require a type specifier, and do not need to be declared in advance. But there is a special keyword for introducing them.

```
let x = 44;
```

- Variables in JavaScript do not require a type specifier, and do not need to be declared in advance. But there is a special keyword for introducing them.

```
let x = 44;
```

- Conditionals are the same as C, and curly braces are used to delimit the blocks again.

- Conditionals are the same as C, and curly braces are used to delimit the blocks again.

```
if
else if
else
switch
?:
```

- Loops are the same as C, and curly braces are used to delimit the blocks again.

- Loops are the same as C, and curly braces are used to delimit the blocks again.

```
while
do-while
for
```

- Functions are introduced with the `function` keyword (basically equivalent to Python's `def`).

- JavaScript functions can be *anonymous*--you don't have to give them a name!
  - We'll revisit this idea shortly.
  - By the way, Python technically has this ability too!

- Declaring arrays (again called arrays in JavaScript) looks really similar to a Python list, and can contain mixed types as before.

- Declaring arrays (again called arrays in JavaScript) looks really similar to a Python list.

```
let nums = [1, 2, 3, 4, 5];
```

- As was the case with Python, JavaScript has the ability to behave as an object-oriented programming language, with properties contained within the object, and methods that apply only to objects that define those methods.

- JavaScript objects look a lot like Python dictionaries:

- As was the case with Python, JavaScript has the ability to behave as an object-oriented programming language, with properties contained within the object, and methods that apply only to objects that define those methods.

- JavaScript objects look a lot like Python dictionaries:

```
let herbie = { year: 1963, model: "Beetle"};
```

- Loops are the same as C, and curly braces are used to delimit the blocks again.

```
while
do-while
for
for ... in
```

- How do we iterate across all of the keys of an object?

```
let herbie = {  year: 1963,
                model: "Beetle",
                sound: "honk.mp3"
              };
```

- How do we iterate across all of the keys of an object?

```
let herbie = {  year: 1963,
                model: "Beetle",
                sound: "honk.mp3"
             };
```

```
for (let prop in herbie)
{
   console.log(herbie[prop]);
}
```

- How do we iterate across all of the keys of an object?

```
let herbie = {  year: 1963,
                model: "Beetle",
                sound: "honk.mp3"
             };
```

```
for (let prop in herbie)
{
   console.log(herbie[prop]);
}
```

1963
Beetle
honk.mp3

- How do we iterate across all of the keys of an object?

```
let herbie = {  year: 1963,
                model: "Beetle",
                sound: "honk.mp3"
             };
```

```
for (let prop in herbie)
{
   console.log(prop);
}
```

- How do we iterate across all of the keys of an object?

```
let herbie = {  year: 1963,
                model: "Beetle",
                sound: "honk.mp3"
             };
```

```
for (let prop in herbie)
{
   console.log(prop);
}
```

year
model
sound

- Loops are the same as C, and curly braces are used to delimit the blocks again.

```
while
do-while
for
for ... in
for ... of
```

- How do we iterate across all of the elements of an array?)

```
let wkArray = ["Mon", "Tue", "Wed"];
```

- How do we iterate across all of the elements of an array?)

```
let wkArray = ["Mon", "Tue", "Wed"];
for (let day of wkArray)
{
  console.log(day);
}
```

- How do we iterate across all of the elements of an array?)

```
let wkArray = ["Mon", "Tue", "Wed"];

for (let day of wkArray)
{
  console.log(day);
}
```

Mon
Tue
Wed

- Strings can be concatenated in JavaScript using the + operator… but be careful mixing types!

- Strings can be concatenated in JavaScript using the + operator… but be careful mixing types!

```
console.log(wkArray[day] + " is day number "
            + (day + 1) + " of the week!");
```

- Strings can be concatenated in JavaScript using the + operator… but be careful mixing types!

```
console.log(wkArray[day] + " is day number "
            + (day + 1) + " of the week!");
```

- Strings can be concatenated in JavaScript using the + operator… but be careful mixing types!

```
console.log(wkArray[day] + " is day number "
            + (parseInt(day) + 1) +
           " of the week!");
```

- Strings can be concatenated in JavaScript using the + operator… but be careful mixing types!

- As with Python, there are still underlying data types, we just don’t often have to worry about them. But here’s the tradeoff of losing the precise control we had in C!

- Arrays are a special case of an object (actually, everything in JavaScript in a special case of an object!). Many methods can be applied to them natively.

```
array.length()
  array.pop()
 array.push(x)
 array.shift()
  array.map()
```

- Arrays are a special case of an object (actually, everything in JavaScript in a special case of an object!). Many methods can be applied to them natively.

```
array.size()
array.pop()
array.push(x)
array.shift()
array.map()
```

- Arrays are a special case of an object (actually, everything in JavaScript in a special case of an object!). Many methods can be applied to them natively.

```
array.size()
array.pop()
array.push(x)
array.shift()
array.map()
```

- This one will give us a good way to introduce anonymous functions.

- `map()` accepts as its parameter a function to be applied to every element of the array. We could define the function in advance and pass it… or we could just define the function in our call to `map()`!

- `map()` accepts as its parameter a function to be applied to every element of the array. We could define the function in advance and pass it… or we could just define the function in our call to `map()`!

```
let nums = [1, 2, 3, 4, 5];
```

- `map()` accepts as its parameter a function to be applied to every element of the array. We could define the function in advance and pass it… or we could just define the function in our call to `map()`!

```
let nums = [1, 2, 3, 4, 5];

nums = nums.map()
```

- `map()` accepts as its parameter a function to be applied to every element of the array. We could define the function in advance and pass it… or we could just define the function in our call to `map()`!

```
let nums = [1, 2, 3, 4, 5];

nums = nums.map(function(num) {
   return num * 2;
});
```

- `map()` accepts as its parameter a function to be applied to every element of the array. We could define the function in advance and pass it… or we could just define the function in our call to `map()`!

```
let nums = [2, 4, 6, 8, 10];

nums = nums.map(function(num) {
   return num * 2;
});
```

- An **event** in HTML/JavaScript is a response to user interaction with the web page. (e.g. user clicked a button, a page has finished loading…)

- JavaScript supports **event handlers**, which are functions that respond to HTML events.

```
<html>
   <head>
      <title>Event Handlers</title>
   </head>
   <body>
      <button onclick="">Button 1</button>
      <button onclick="">Button 2</button>
   </body>
</html>
```

- An **event** in HTML/JavaScript is a response to user interaction with the web page (e.g. user clicked a button, a page has finished loading…)

- JavaScript supports **event handlers**, which are functions (usually called **callbacks**) that respond to HTML events.

- We can write a generic event handler here, which will automatically create an *event object*, that will tell us which of the two buttons was clicked.

Lab!

Test

- Release Monday 11/15 at 4:15pm
- Due Sunday 11/21 at 11:59am (This is noon!!!) (The day after harvard-yale!!!)
- A mixture of conceptual and coding questions
- Could cover anything we've talked about in class
- Sometimes includes extra readings/videos
- Cannot work with other students at all
- Cannot ask me or other TFs/CAs for help
- All questions should go to heads@cs50.harvard.edu
- These questions are more difficult and involved than the quizzes

https://cs50.harvard.edu/college/2021/fall/test/about/