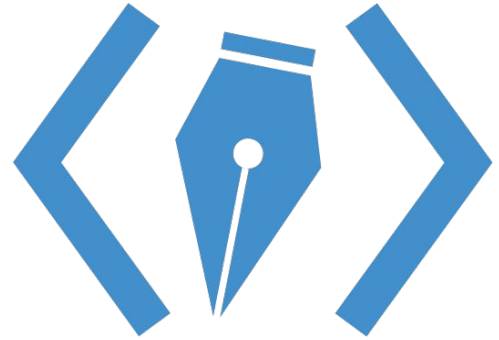


Week 9

How JavaScript Meets HTML/CSS



Announcements

Homework 7 due tomorrow at 7pm!

Homework 8 will be released tonight

Start the [Final Project](#)!

Check-in due **November 12th**

Project due **December 1st**

Your project can be featured on our [showcase](#) website!

Final Project Submission: wdd.io/go/project-submit

Come to office hours via Piazza or wdd.io/go/OH

Give us anonymous feedback at wdd.io/go/feedback

Review variables & functions

New concept: Objects

JS & HTML meet → The DOM

Adding/removing CSS Classes with JS

Review variables & functions

New concept: Objects

JS & HTML meet → The DOM

Adding/removing CSS Classes with JS

Review: Variables

Variables are like in math! We use a variable when we want to store the value of something and give it a name.

```
let name = "Ajia";
```

```
let age = 21;
```

“let” means that we’re creating a variable

Make names meaningful! Usually a noun.

value to store, in this case, a number.

IMPORTANT: if we’re storing text, it must be in **quotes**. This is called a **string**.

Review: Variables

If we do something with a variable, we want to store the result so we can use it later!

```
let age = 20 + 5;
```

```
age - 2.5;
```

First of all, this is **incorrect**, and the browser would give you an error. Second of all, it does not make sense, because the `age - 2.5` value would be lost forever, since it's not stored anywhere

```
age = age * 2;
```

This is correct! When you modify a variable, remember to *re-assign* it, or put it in a new variable.

Review: Using Functions

defining the function - the code is not run, but is now ready to be used!

```
function showGreeting(name) {  
    let greeting = "howdy " + name + "!";  
    alert(greeting);  
}
```

calling the function - actually runs the code in the function definition

```
showGreeting("Myles");
```

Review: Using Functions

```
function square(x) {  
    return x * x;  
}
```

“return” means,
this is the final
output value

```
let y = square(5);  
let z = square(y);
```

So now the
value of **y** is 25.

A variable can be the
function input! This is the
equivalent of **square(25)**.

Review variables & functions

New concept: Objects

JS & HTML meet → The DOM

Adding/removing CSS Classes with JS

New concept: Objects

- Before we go on - we need to introduce a new concept
- When we write code we're trying to convert the world and how we describe it in English, to a representation that computers can understand
- We've only talked about numbers and strings, but what if we want something more complex?
- **Theme:** complex things are just composed of simple things

New concept: Objects

- Objects are JavaScript's way of representing more complex things.
- Objects are a collection of **methods (functions)** and **properties (variables)**
- **Theme:** think about what you're trying to do in English before converting to code
- **Example:** a car
 - What are **properties** of a car?
 - color, model, current mph
 - What are **functions** a car can do?
 - drive straight, brake, change color



Car object

- **Properties:** color, model, current mph
- **Functions:** drive straight, brake, change color

Getting closer to code..

Properties

- `color`: string
- `model`: string
- `mph`: number

Functions

- `driveStraight()`
- `brake()`
- `changeColor(color)`



Car object: in JavaScript

Properties

- `color`: string
- `model`: string
- `mph`: number

Functions

- `driveStraight()`
- `brake()`
- `changeColor(color)`

```
let car = {  
  color: "red",  
  model: "Tesla Model S",  
  mph: 0,  
  driveStraight: function() {  
    // ...  
  },  
  brake: function() {  
    // ...  
  },  
  changeColor: function(newColor) {  
    // ...  
  },  
};
```

Car object: in JavaScript

Properties

- `color`: string
- `model`: string
- `mph`: number

Functions

- `driveStraight()`
- `brake()`
- `changeColor(color)`

```
let car = {  
  color: "red",  
  model: "Tesla Model S",  
  mph: 0,  
  driveStraight: function() {  
    // ...  
  },  
  brake: function() {  
    // ...  
  },  
  changeColor: function(newColor) {  
    // ...  
  },  
};
```

Car object: in JavaScript

Properties

- `color`: string
- `model`: string
- `mph`: number

Functions

- `driveStraight()`
- `brake()`
- `changeColor(color)`

```
let car = {  
  color: "red",  
  model: "Tesla Model S",  
  mph: 0,  
  driveStraight: function() {  
    // ...  
  },  
  brake: function() {  
    // ...  
  },  
  changeColor: function(newColor) {  
    // ...  
  },  
};
```

Car object: in JavaScript

Wow, this is a lot of new syntax!
We won't really be writing our own objects, so **don't worry** too much about this.

Just be able to recognize how variables can now contain not just a number or a string, but also **objects** that have properties and functions.

```
let car = {  
  color: "red",  
  model: "Tesla Model S",  
  mph: 0,  
  driveStraight: function() {  
    // ...  
  },  
  brake: function() {  
    // ...  
  },  
  changeColor: function(newColor) {  
    // ...  
  },  
};
```


Using object properties and functions: dot notation

```
let car = {  
  color: "red",  
  model: "Tesla Model S",  
  mph: 0,  
  driveStraight: function() {  
    // ...  
  },  
  changeColor: function(newColor) {  
    // ...  
  },  
};
```

```
console.log(car.color);
```

```
// result: "red"
```

```
car.driveStraight();
```

```
car.changeColor("blue");
```

IMPORTANT: To call an object's function, or get the value stored in an object's property, we use a dot!

Now back to how HTML & JS meet..

Review variables & functions

New concept: Objects

JS & HTML meet → The DOM

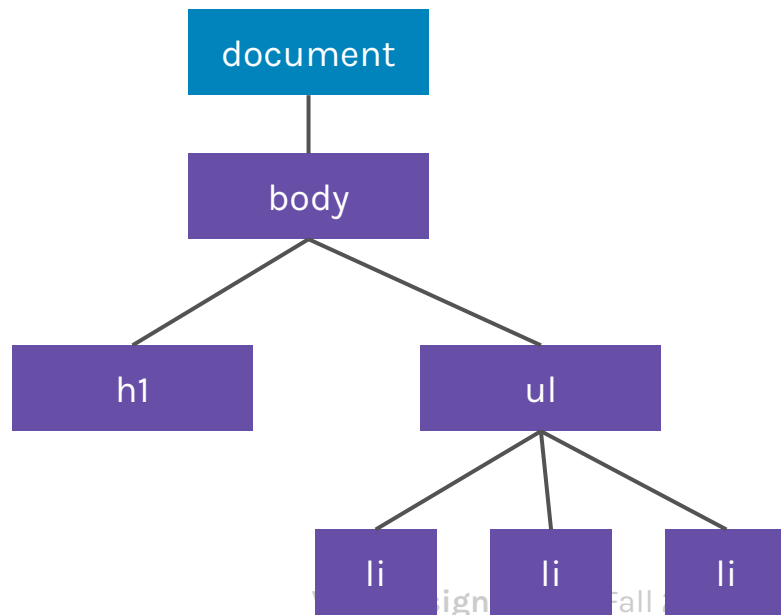
Adding/removing CSS Classes with JS

HTML & JS meet!

index.html

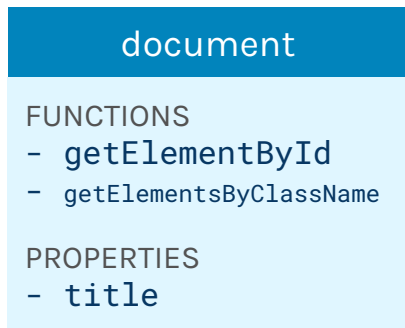
```
<html>
<body>
  <h1>Food Places</h1>
  <ul id="food-list" class="blue-box">
    <li>Mezzo</li>
    <li>Pho K&K</li>
    <li class="highlight">Taco Bell</li>
  </ul>
</body>
</html>
```

Diagram of the JavaScript side constructed by the web browser, very simplified



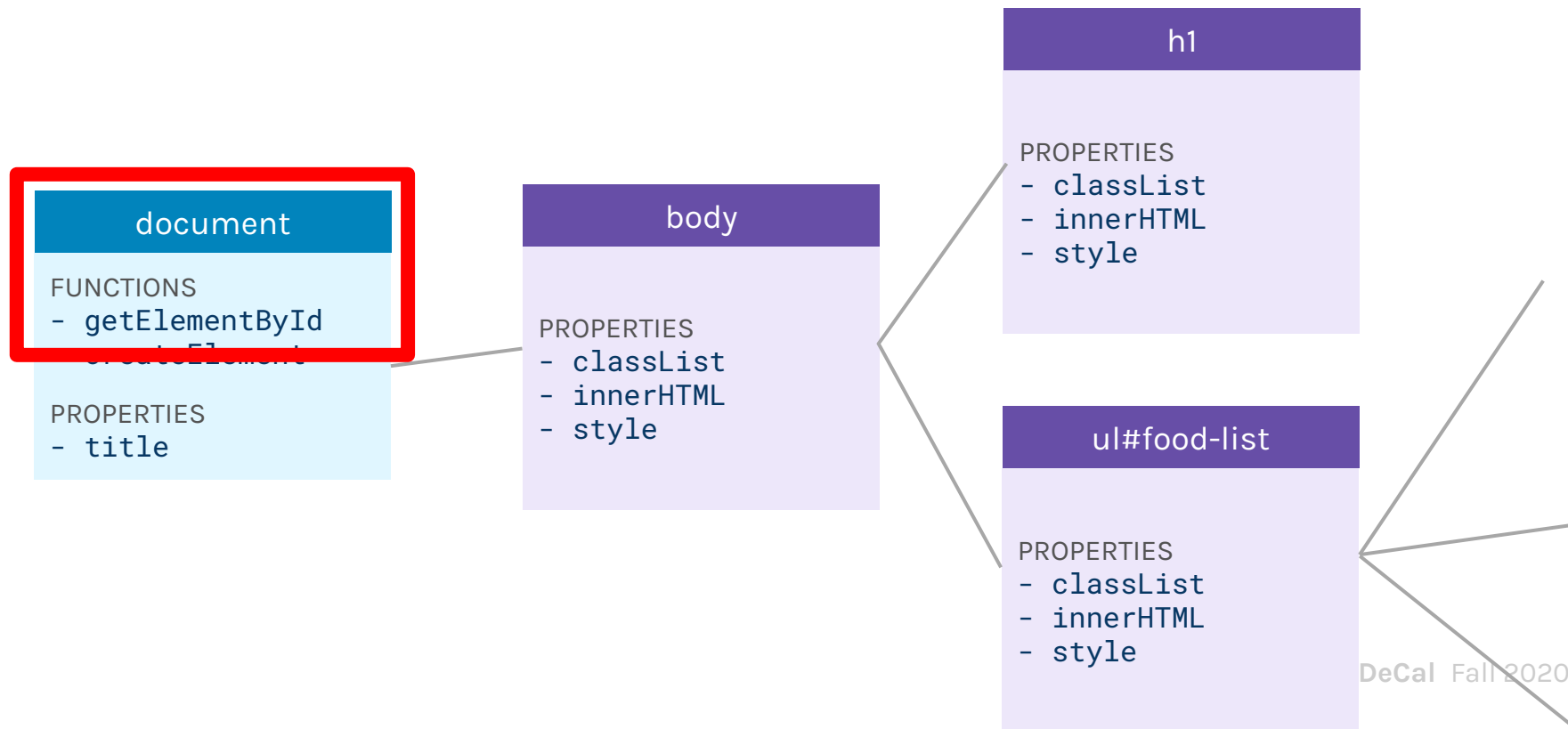
The DOM: Document Object Model

When the HTML page is loaded, JavaScript creates its own representation of the page, which is made up of objects. Each HTML tag is an object.



Only a few (out of like a hundred..) functions and properties are listed here, and all HTML elements on the page have the same set. We won't go over all of these in class, but this diagram is a sampler.

Our first focus: `document.getElementById`



What is this function, `document.getElementById()`?

- Takes one argument, a string, which is the id of the HTML element we want
- Returns the element object

What is this function?

```
let listElement = document.getElementById("food-list");
```



```
<ul id="food-list" class="blue-box">  
  <li>Mezzo</li>  
  <li>Pho K&K</li>  
  <li class="highlight">Taco Bell</li>  
</ul>
```


Examples: properties of element objects

```
let foodList = document.getElementById("food-list");
```

```
console.log(foodList.classList);
```

```
-> result: ["blue-box"]
```

```
console.log(foodList.innerHTML);
```

```
-> result: "<li>Mezzo</li><li>Pho K&K</li>  
          <li class='highlight'>Taco Bell</li>"
```

```
console.log(foodList.children);
```

```
-> result: [li, li, li]
```

Remember, this is what is available to us:

ul#food-list

PROPERTIES

- classList
- innerHTML
- style
- children

w

Why are we doing this again?

Recall some of our examples of interactive features from last lecture:

- When the user clicks on `this image`, then `increase its size`.
 - “increase its size” → *change its CSS `width`*
- When the user clicks on `this button`, then `change colors to dark mode`.
 - “change colors to dark mode” → *change CSS `color` and `background-color`*

We want to convert these to code:

`"this button" → document.getElementById("dark-mode-button")`

`"change CSS" → see next slides! we're almost there!!`

Another method: `document.querySelector()`

- Takes one argument, a string, which is a **CSS selector** of the HTML element we want
- Returns the **first** element object that matches the selector name
- Can be the same as `document.getElementById()`

What is this function?

```
let listElement = document.querySelector("#food-list");
```



```
<ul id="food-list" class="blue-box">  
  <li>Mezzo</li>  
  <li>Pho K&K</li>  
  <li class="highlight">Taco Bell</li>  
</ul>
```

Another way??

```
let listElement = document.querySelector("ul");
```



ul#food-list

PROPERTIES

- classList
- innerHTML
- style
- children

```
<ul id="food-list" class="blue-box">  
  <li>Mezzo</li>  
  <li>Pho K&K</li>  
  <li class="highlight">Taco Bell</li>  
</ul>
```

Multiple repeated names?

```
<div class="child">  
  Child 1  
</div>  
<div class="child">  
  Child 2  
</div>
```

```
let element = document.querySelector(".child");  
  
/* element is now the first child div. any changes to  
"element" will only change the first div! */
```

Demo

Review variables & functions

New concept: Objects

JS & HTML meet → The DOM

Adding/removing CSS Classes with JS

Properties of properties

Turns out that the result of `foodList.classList` is not just a list, but another object! So now our code goes like:

Properties of properties

Turns out that the result of `foodList.classList` is not just a list, but another object! So now our code goes like:

`document`

`document`

FUNCTIONS

- `getElementById`
- `createElement`

PROPERTIES

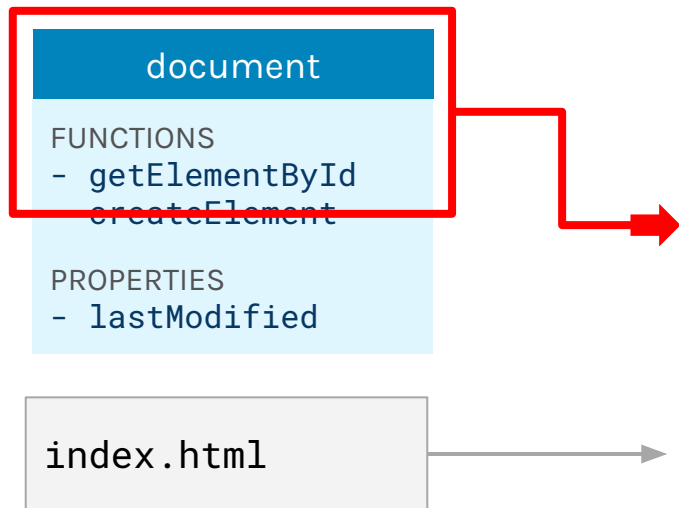
- `lastModified`

`index.html`

Properties of properties

Turns out that the result of `foodList.classList` is not just a list, but another object! So now our code goes like:

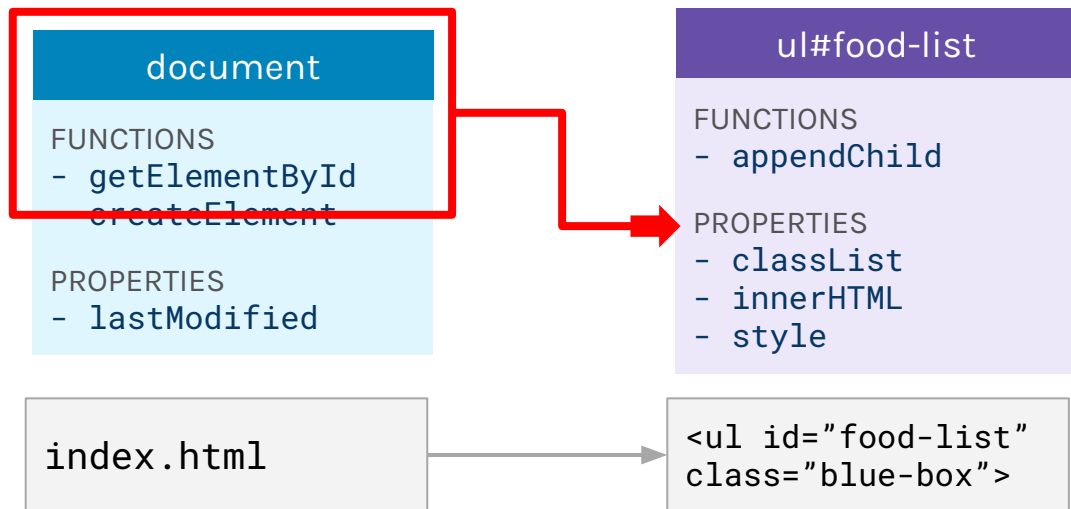
```
document.getElementById("food-list")
```



Properties of properties

Turns out that the result of `foodList.classList` is not just a list, but another object! So now our code goes like:

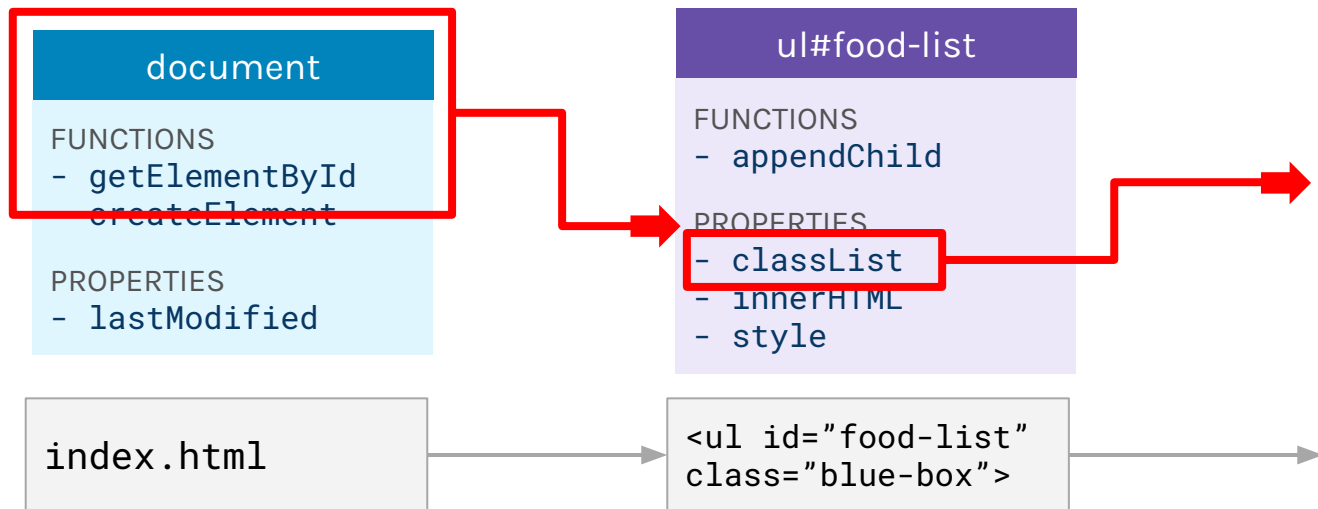
```
document.getElementById("food-list")
```



Properties of properties

Turns out that the result of `foodList.classList` is not just a list, but another object! So now our code goes like:

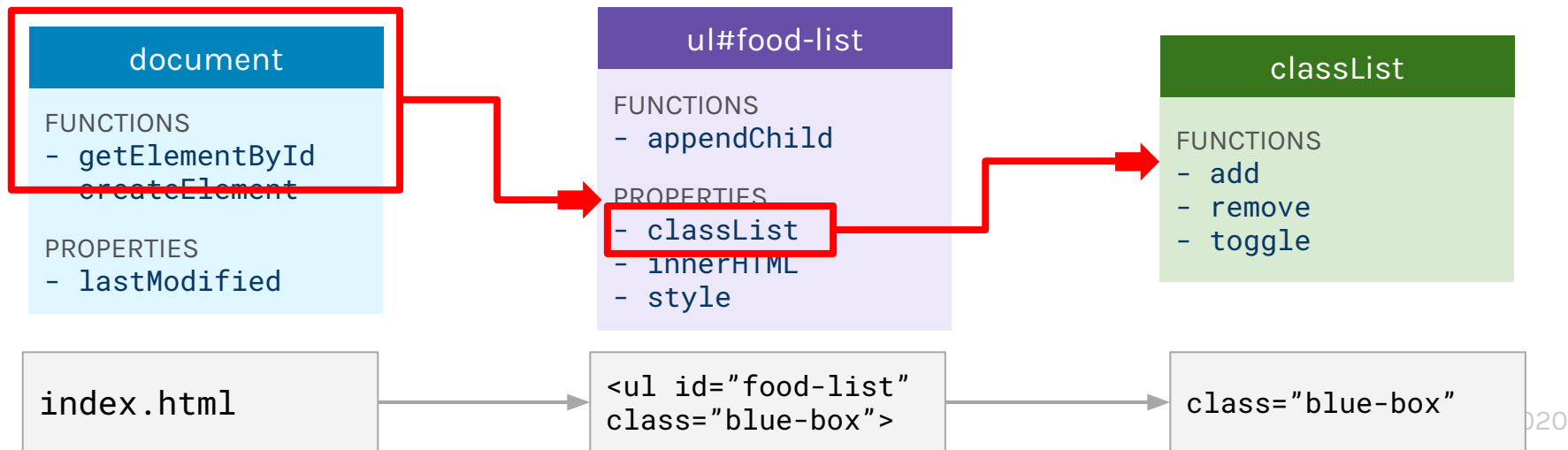
```
document.getElementById("food-list").classList
```



Properties of properties

Turns out that the result of `foodList.classList` is not just a list, but another object! So now our code goes like:

```
document.getElementById("food-list").classList
```



Ok here we are! Changing CSS with JS!

```
let foodList = document.getElementById("food-list");
```

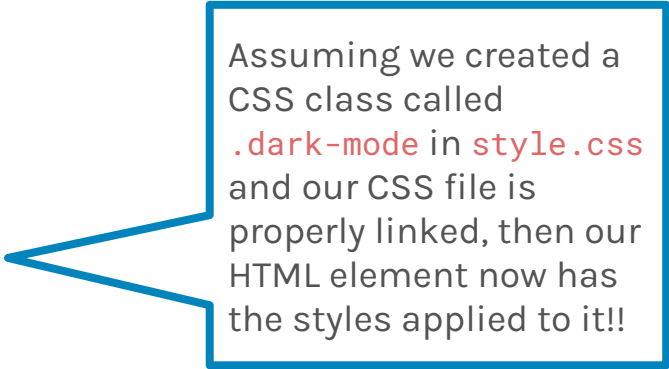
```
console.log(foodList.classList);
```

```
-> result: ["blue-box"]
```

```
foodList.classList.add("dark-mode");
```

```
console.log(foodList.classList);
```

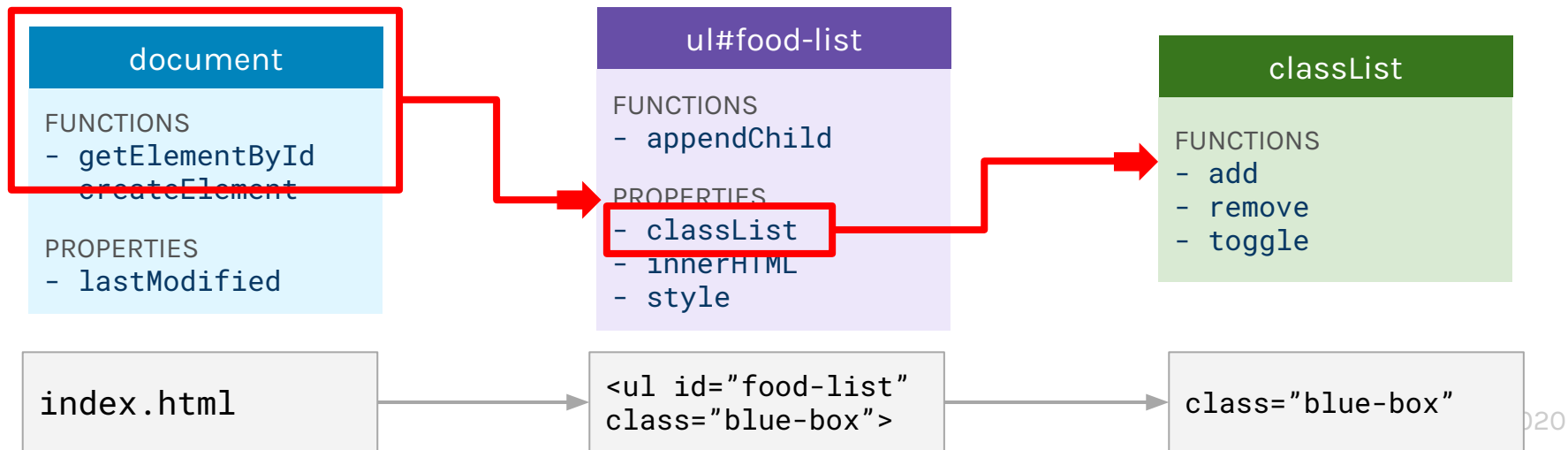
```
-> result: ["blue-box", "dark-mode"]
```



Assuming we created a CSS class called `.dark-mode` in `style.css` and our CSS file is properly linked, then our HTML element now has the styles applied to it!!

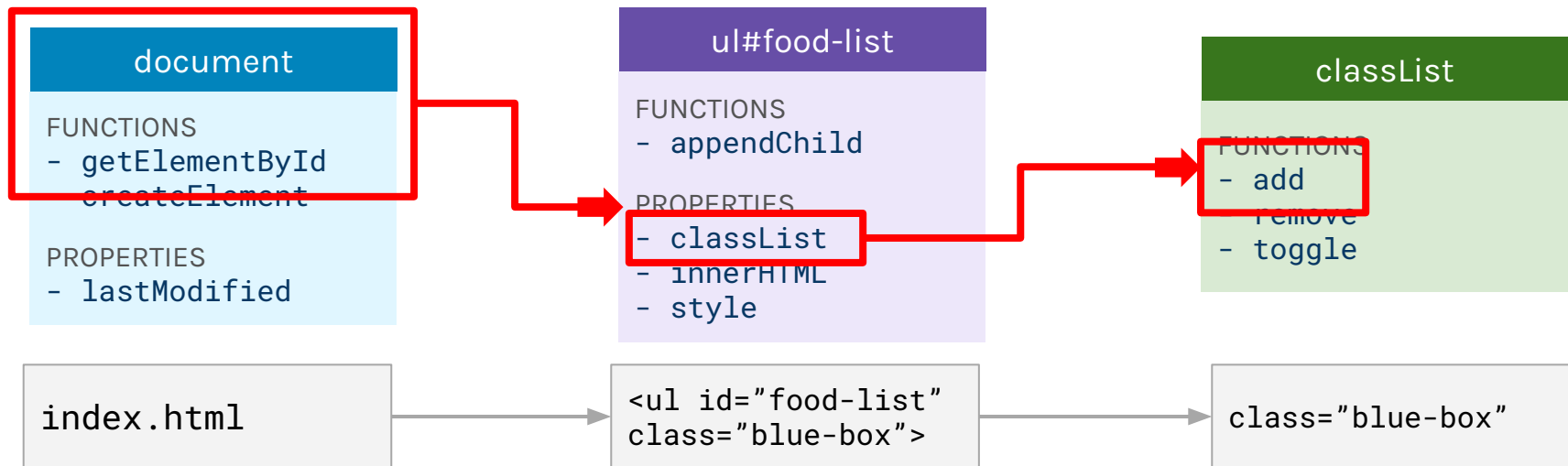
Properties of properties

```
document.getElementById("food-list").classList.add("dark-mode")
```



Properties of properties

```
document.getElementById("food-list").classList.add("dark-mode");
```



Removing a class

```
foodList.classList.add("dark-mode");  
console.log(foodList.classList);
```

```
-> result: ["blue-box", "dark-mode"]
```

```
foodList.classList.remove("dark-mode");  
console.log(foodList.classList);
```

```
-> result: ["blue-box"]
```

Demo

Thinking about this interactive functionality..

We'll think of a feature to add to our webpage like:

When the user clicks on *[this button/link/image]*,
then *[add/change/remove]* *[this HTML/CSS]*.

Thinking about this interactive functionality

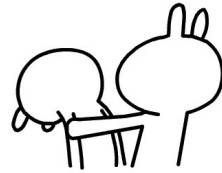
Next lecture we'll put this all together!

We can now specify the element if it has an id!

think of a feature to add to our page like:

When the user clicks on `[this button/link/image]`,
then `[add/change/remove]` `[this HTML/CSS]`.

We can now
add/change/remove CSS!



Questions?