

# JAVASCRIPT DEVELOPMENT

Sasha Vodnik, Instructor

# HELLO!

1. Pull changes from the `svodnik/JS-SF-10-resources` repo to your computer
2. Open the `06-json-dom > starter-code` folder in your code editor

---

**JAVASCRIPT DEVELOPMENT**

---

# **JSON & DOM**

# LEARNING OBJECTIVES

At the end of this class, you will be able to

- › Identify likely objects, attributes, and methods in real-world scenarios
- › Create JavaScript objects using object literal notation
- › Implement and interface with JSON data
- › Identify differences between the DOM and HTML.
- › Explain and use JavaScript methods for DOM manipulation.

# AGENDA

- Lab: Translate real world scenarios into objects
- Lab: Create objects
- JSON
- Lab: Work with JSON
- DOM intro

---

## JSON & DOM

---

# WEEKLY OVERVIEW

### WEEK 4

Slack bot lab / JSON & DOM

### WEEK 5

DOM & jQuery / Advanced jQuery

### WEEK 6

Ajax & APIs / Asynchronous JS & callbacks

# EXIT TICKET QUESTIONS

1. I didn't fully understand what a slack bot is.
2. How to balance finding interesting code on the internet and copying (and hoping for the best) -v- writing everything I cant to do from scratch?
3. Is it harder to write a bot for Slack without using Hubot?
4. How can I integrate a third-party API with my slackbot ?
5. Other than Slack, what other apps can Hubot access/integrate? Can you give some more example?

# BOTS — GROUP CHECKIN

---



## EXERCISE

### TYPE OF EXERCISE

---

- Groups of 3-4

### TIMING

---

*3 min*

#### 1. Share

- What you're planning for your bot to do
- How far you've gotten
- An outstanding question or challenge

#### 2. As a group, brainstorm possible next steps for each challenge described by a group member.



# EXERCISE — OBJECTS

---

## KEY OBJECTIVE

---

- ▶ Create JavaScript objects using object literal notation

## TYPE OF EXERCISE

---

- ▶ Groups of 2-3

## TIMING

---

*5 min*



EXERCISE

1. For the thing you've been assigned, make a list of attributes (descriptions) and actions (things it can do).
2. On your desk or on the wall, write a JavaScript statement to create a variable whose name corresponds to the thing you were assigned and whose value is an empty object.
3. Write another statement to add a property to the object and specify a value for the property.
4. Write another statement to add a method to the object, and specify a value for the method (use a comment or `console.log()` statement for the function body).
5. BONUS: Rewrite your answers for 2-4 as a single JavaScript statement.

## **REAL WORLD SCENARIO**

A user, browsing on a shopping website, searches for size 12 running shoes, and examines several pairs before purchasing one.

## OBJECTS = NOUNS

A **user**, browsing on a **shopping website**, searches for size 12 running shoes, and examines **several pairs** before purchasing one.

**implicit object:**

**shopping cart**

# PROPERTIES = ADJECTIVES

A user, browsing on a shopping website, searches for **size 12** **running** shoes, and examines several pairs before purchasing one.

**implicit properties:**

for each pair of shoes:

price  
color

for the shopping cart:

contents  
total  
shipping  
tax

## METHODS = VERBS

A user, browsing on a shopping website, **searches** for size 12 running shoes, and examines several pairs before purchasing one.

**implicit methods:**

for each pair of shoes:

add to cart

for the shopping cart:

calculate shipping  
calculate tax  
complete purchase  
remove item

---

## EXERCISE — REAL WORLD SCENARIOS & OBJECTS

---



### EXERCISE

#### KEY OBJECTIVE

---

- ▶ Identify likely objects, properties, and methods in real-world scenarios

#### TYPE OF EXERCISE

---

- ▶ Groups of 3-4

#### TIMING

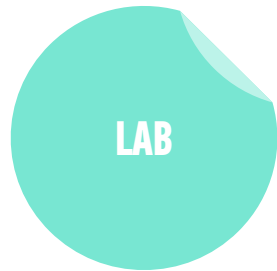
---

*5 min*

1. Read through your scenario together.
2. Identify and write down likely objects, properties, and methods in your scenario. (Remember to consider implicit objects as well as explicit ones.)
3. Choose someone to report your results to the class.

# LAB — OBJECTS

---



## KEY OBJECTIVE

---

- ▶ Create JavaScript objects using object literal notation

## TYPE OF EXERCISE

---

- ▶ Individual or pair

## TIMING

---

*10 min*

1. Open starter-code > 0-object-exercise > monkey.js in your editor.
2. Create objects for 3 different monkeys each with the properties name, species, and foodsEaten, and the methods eatSomething(thingAsString) and introduce.
3. Practice retrieving properties and using methods with both dot notation and bracket syntax.

# JSON



# JSON IS A DATA FORMAT BASED ON JAVASCRIPT

object

```
let instructor = {  
  firstName: 'Sasha',  
  lastName: 'Vodnik',  
  city: 'San Francisco',  
  classes: [  
    'JSD', 'FEWD'  
  ],  
  classroom: 7,  
  launched: true,  
  dates: {  
    start: 20180205,  
    end: 20180406  
  },  
};
```

JSON

```
{  
  "firstName": "Sasha",  
  "lastName": "Vodnik",  
  "city": "San Francisco",  
  "classes": [  
    "JSD", "FEWD"  
  ],  
  "classroom": 7,  
  "launched": true,  
  "dates": {  
    "start": 20180205,  
    "end": 20180406  
  }  
}
```

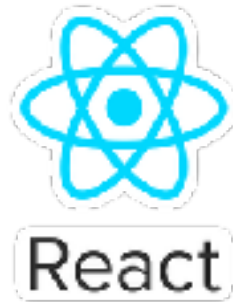
# JSON

- Easy for humans to read and write
- Easy for programs to parse and generate

```
{
  "firstName": "Sasha",
  "lastName": "Vodnik",
  "city": "San Francisco",
  "classes": [
    "JSD", "FEWD"
  ],
  "classroom": 7,
  "launched": true,
  "dates": {
    "start": 20180205,
    "end": 20180406
  }
}
```

# JSON IS NOT JAVASCRIPT-SPECIFIC

- Used across the web by programs written in many languages



ANGULARJS



# JSON RULES

- Property names must be double-quoted strings.
- Trailing commas are forbidden.
- Leading zeroes are prohibited.
- In numbers, a decimal point must be followed by at least one digit.
- Most characters are allowed in strings; however, certain characters (such as ' , " , \ , and newline/tab) must be 'escaped' with a preceding backslash ( \ ) in order to be read as characters (as opposed to JSON control code).
- All strings must be double-quoted.
- No comments!

## TO CONVERT AN OBJECT TO JSON

```
JSON.stringify(object);
```

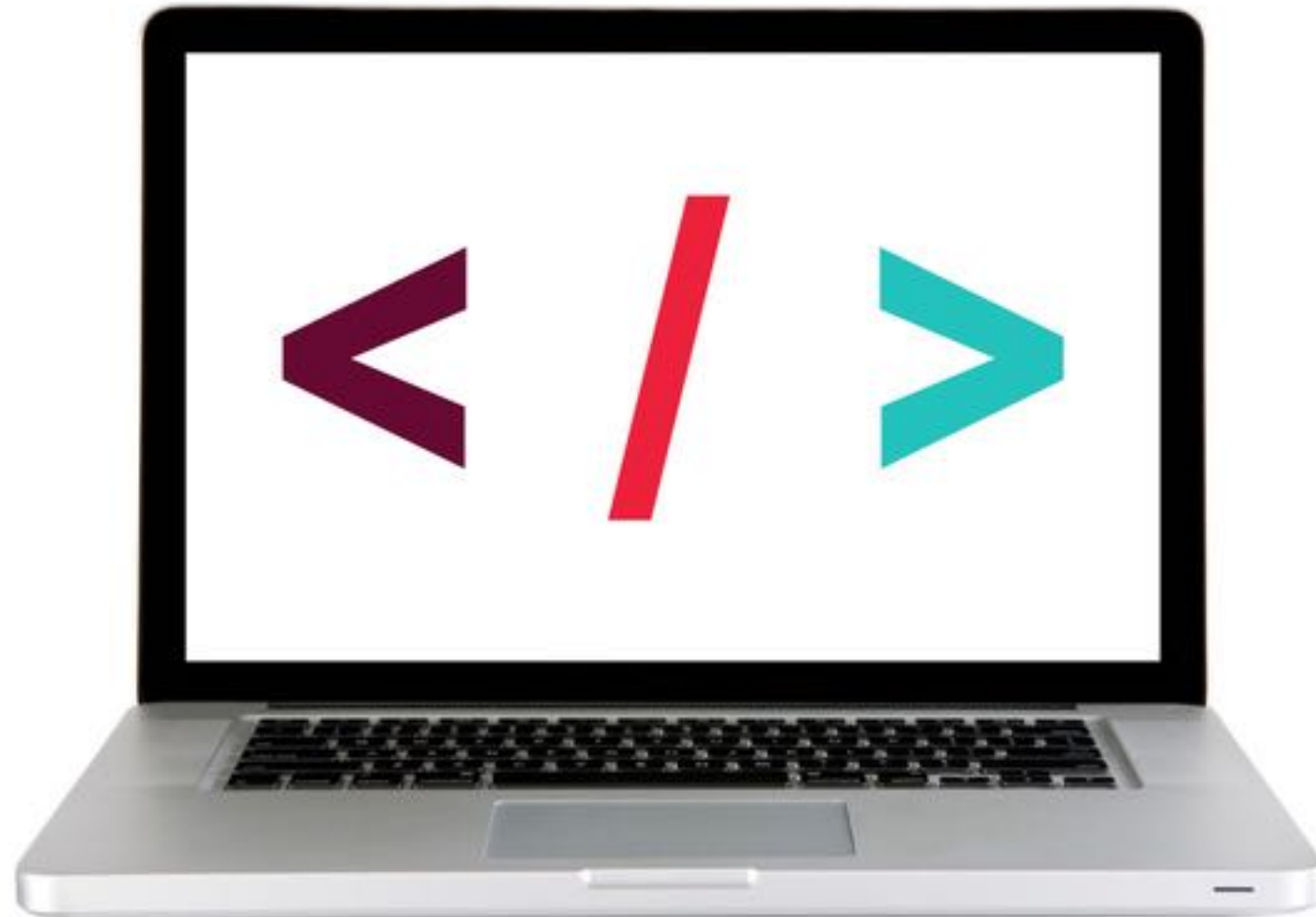
## **TO CONVERT JSON TO AN OBJECT**

**JSON.parse(*json*);**

---

**LET'S TAKE A LOOK**

---



# EXERCISE — JSON

---



## EXERCISE

### KEY OBJECTIVE

---

- Implement and interface with JSON data

### TYPE OF EXERCISE

---

- Groups of 2-3

### TIMING

---

*3 min*

1. Write JSON code that contains an error.
2. Write your code on the wall.
3. When everyone's code is done, we will look at the code together as a class and practice identifying errors.



# WORKING WITH NESTED DATA STRUCTURES

**YAY, I GOT SOME DATA!**

```
let person = '{"firstName":  
"Sasha","lastName": "Vodnik","city":  
"San Francisco","classes": ["JSD",  
"FEWD"],"classroom": 7,"launched":  
true,"dates": {"start": 20180205,"end":  
20180406}}';
```

**WAIT, WHAT?!**

# WORKING WITH NESTED DATA STRUCTURES

**1. PARSE THE JSON TO A JAVASCRIPT OBJECT (OR ARRAY!)**

**2. VIEW THE RESULTING DATA STRUCTURE**

**3. LOCATE THE DATA YOU WANT TO REFERENCE**

**4. USE DOT SYNTAX OR SQUARE BRACKET NOTATION TO MOVE DOWN A LEVEL, THEN REPEAT**

# WORKING WITH NESTED DATA STRUCTURES

## 1. PARSE THE JSON TO A JAVASCRIPT OBJECT (OR ARRAY!)

```
let person = '{"firstName":  
"Sasha","lastName": "Vodnik","city":  
"San Francisco","classes": ["JSD",  
"FEWD"],"classroom": 7,"launched":  
true,"dates": {"start": 20180205,"end":  
20180406}}';
```



```
let personObject = JSON.parse(person);
```

# WORKING WITH NESTED DATA STRUCTURES

## 2. VIEW THE RESULTING DATA STRUCTURE

```
let personObject = JSON.parse(person);  
console.log(personObject);  
>
```



```
city: "San Francisco"  
▼ classes: Array(2)  
  0: "JSD"  
  1: "FEWD"  
  length: 2  
  ► __proto__: Array(0)  
classroom: 8  
▼ dates:  
  end: 20171113  
  start: 20170906  
  ► __proto__: Object  
firstName: "Sasha"  
lastName: "Vodnik"  
launched: true
```

# WORKING WITH NESTED DATA STRUCTURES

## 3. LOCATE THE DATA YOU WANT TO REFERENCE


```
city: "San Francisco"
▼ classes: Array(2)
  0: "JSD"
  1: "FEWD"
  length: 2
  ► __proto__: Array(0)
classroom: 8
▼ dates:
  end: 20171113
  start: 20170906
  ► __proto__: Object
firstName: "Sasha"
lastName: "Vodnik"
launched: true
```

# WORKING WITH NESTED DATA STRUCTURES

## 4. USE DOT SYNTAX OR SQUARE BRACKET NOTATION TO MOVE DOWN A LEVEL, THEN REPEAT

direct property:

```
console.log(personObject.city);  
> "San Francisco"
```



```
city: "San Francisco"  
▼ classes: Array(2)  
  0: "JSD"  
  1: "FEWD"  
  length: 2  
  ► __proto__: Array(0)  
classroom: 8  
▼ dates:  
  end: 20171113  
  start: 20170906  
  ► __proto__: Object  
firstName: "Sasha"  
lastName: "Vodnik"  
launched: true
```

# WORKING WITH NESTED DATA STRUCTURES

4. USE DOT SYNTAX OR SQUARE BRACKET NOTATION TO MOVE DOWN A LEVEL, THEN REPEAT

```
city: "San Francisco"
▼ classes: Array(2)
  0: "JSD"
  1: "FEWD"
  length: 2
  ► __proto__: Array(0)
classroom: 8
▼ dates:
  end: 20171113
  start: 20170906
  ► __proto__: Object
firstName: "Sasha"
lastName: "Vodnik"
launched: true
```

direct property > array element

```
console.log(personObject.classes);
> ["JSD", "FEWD"]
```

```
console.log(personObject.classes[0]);
> "JSD"
```



# WORKING WITH NESTED DATA STRUCTURES

4. USE DOT SYNTAX OR SQUARE BRACKET NOTATION TO MOVE DOWN A LEVEL, THEN REPEAT

```
city: "San Francisco"
▼ classes: Array(2)
  0: "JSD"
  1: "FEWD"
  length: 2
  ► __proto__: Array(0)
classroom: 8
▼ dates:
  end: 20171113
  start: 20170906
  ► __proto__: Object
firstName: "Sasha"
lastName: "Vodnik"
launched: true
```

direct property > nested object property

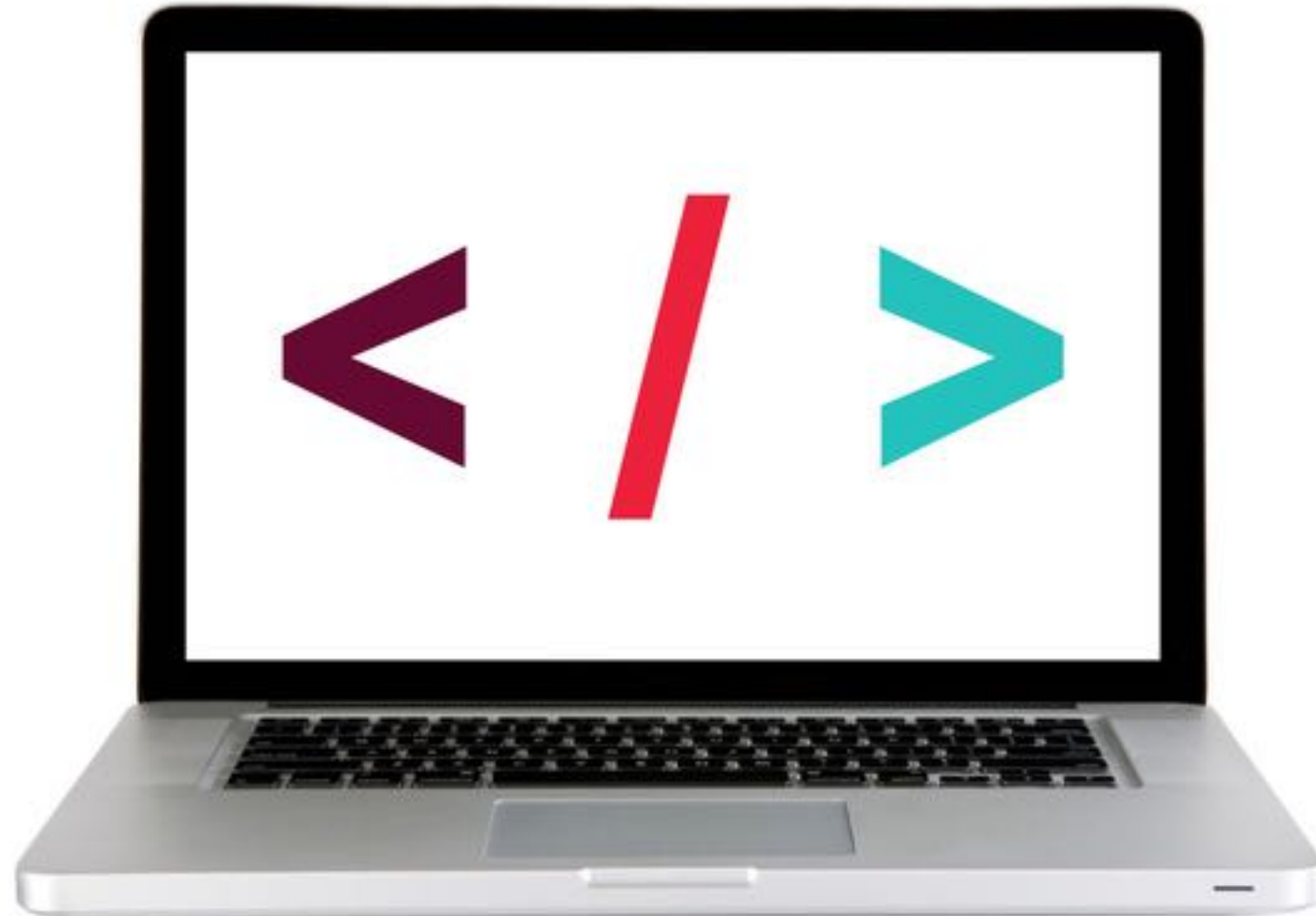
```
console.log(personObject.dates);
> {end:20171113,start:20170906}
```

```
console.log(personObject.dates.start);
> 20170906
```

---

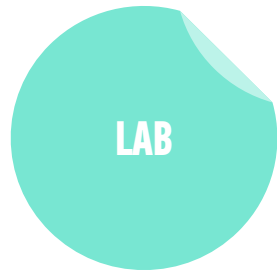
**LET'S TAKE A LOOK**

---



# LAB — JSON

---



## KEY OBJECTIVE

---

- Implement and interface with JSON data

## TYPE OF EXERCISE

---

- Individual or pair

## TIMING

---

*10 min*

1. Open `starter-code > 2-json-exercise > app.js` in your editor.
2. Follow the instructions to write code that produces the stated output.

# THE DOCUMENT OBJECT MODEL (DOM)

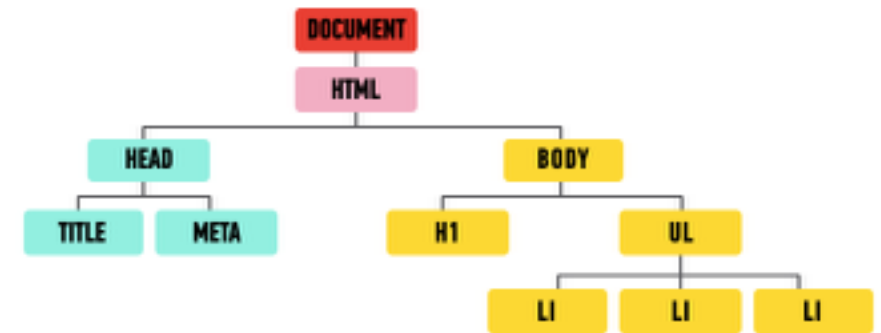
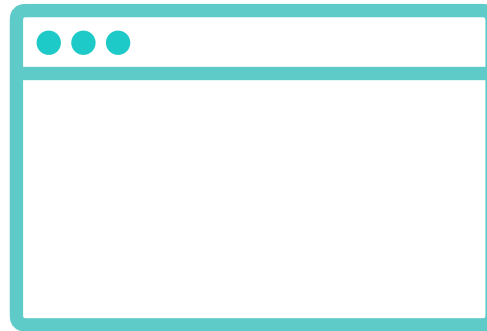
## DOM TREE — HTML FILE

```
index.html *
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>The Evolution of Denim</title>
6 </head>
7 <body>
8
9   <h1>The Evolution of Denim</h1>
10  <p>
11    Chambray retro plaid gentrify letterpress.
    Taxidermy ennui cliche Intelligentsia. Echo
    Park umami authentic before they sold out. <a
    href="https://placekitten.com/">Forage
    wayfarers</a> listicle Kickstarter, Pitchfork
    cray messenger bag fap High Life tilde pug
    Blue Bottle mumblecore.
12  </p>
13  <ul>
14    <li>Dark Wash</li>
15    <li>Stone Wash</li>
16    <li>Chambray</li>
17  </ul>
18
19 </body>
20 </html>
```

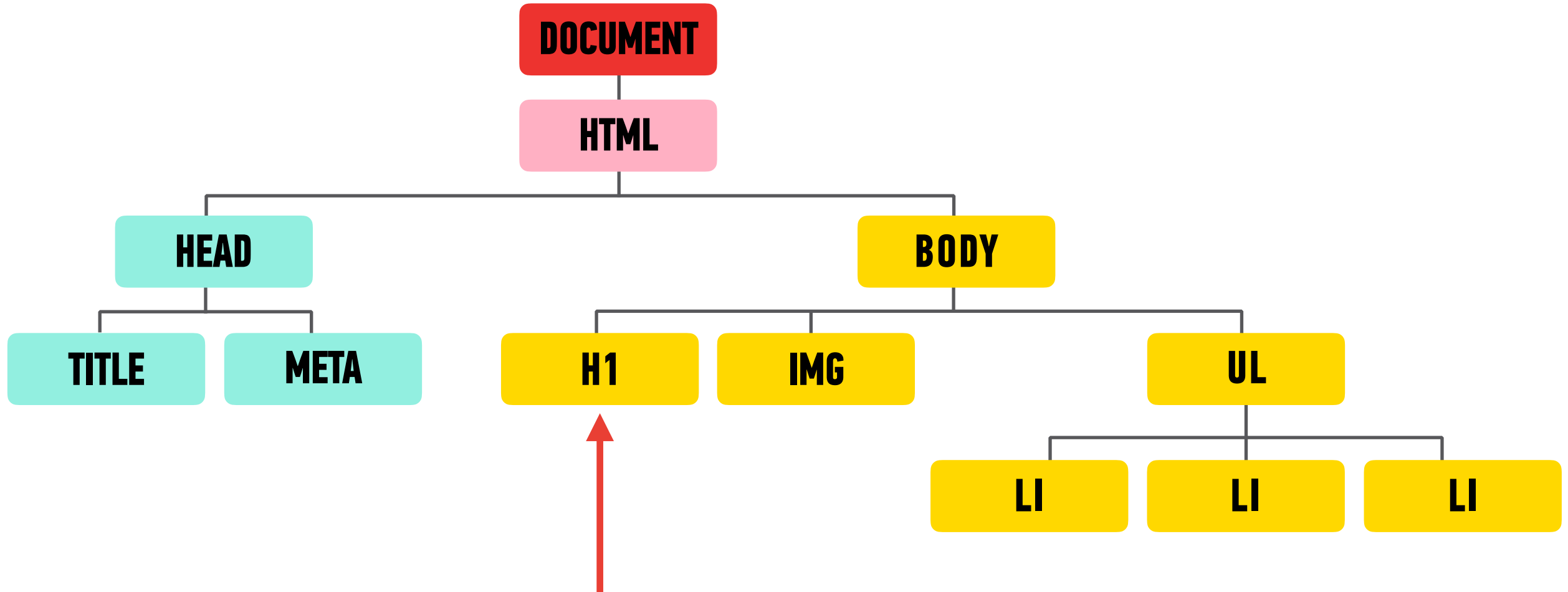
# DOM TREE

- ▶ The browser pulls in this HTML document, analyzes it, and creates an *object model* of the page in memory.
- ▶ This model is called the *Document Object Model (DOM)*.
- ▶ The DOM is structured like a tree, a DOM Tree, like in the model below:

```
index.html
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>The Evolution of Denim</title>
6 </head>
7 <body>
8   <h1>The Evolution of Denim</h1>
9   <p>
10    Chambray retro plaid gentrify letterpress.
11    Taxidermy ennui cliche Intelligentsia. Echo
12    Park umami authentic before they sold out. <a
13    href="https://placekitten.com/">Forage
14    wayfarers</a> listicle Kickstarter, Pitchfork
15    cray messenger bag fao High Life tilde pug
16    Blue Bottle mumblecore.
17  </p>
18  <ul>
19    <li>Dark Wash</li>
20    <li>Stone Wash</li>
21    <li>Chambray</li>
22  </ul>
23 </body>
24 </html>
```



# DOM TREE



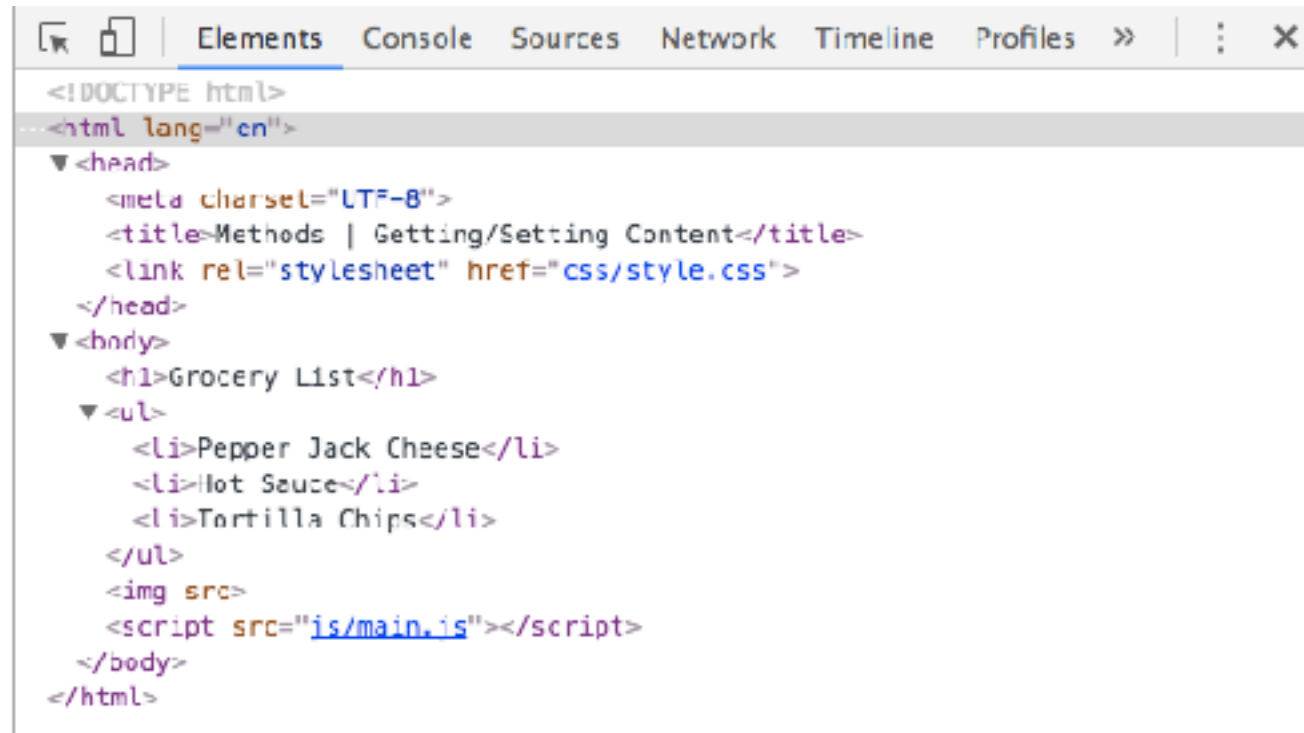
- ▶ Each element in the HTML document is represented by a *DOM node*.
- ▶ You can think of a node as a live object that you can access and change using JavaScript.
- ▶ When the model is updated, those changes are reflected on screen.

# DOM TREE

- ▶ In Chrome, you can go to View > Developer > Developer Tools and click on the Elements panel to take a look at the DOM tree.

## Grocery List

- Pepper Jack Cheese
- Hot Sauce
- Tortilla Chips



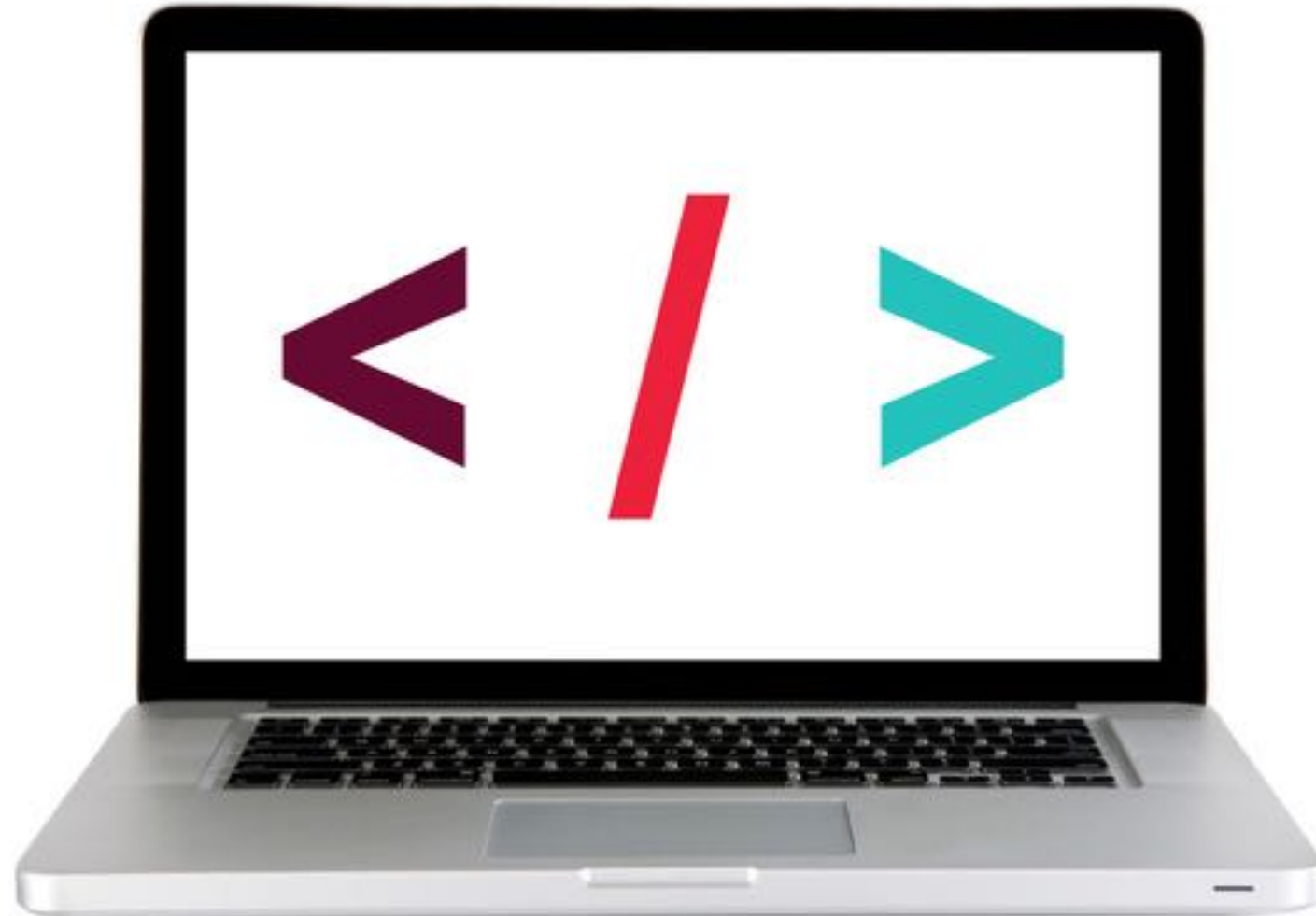
```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>Methods | Getting/Setting Content</title>
    <link rel="stylesheet" href="css/style.css">
  </head>
  <body>
    <h1>Grocery List</h1>
    <ul>
      <li>Pepper Jack Cheese</li>
      <li>Hot Sauce</li>
      <li>Tortilla Chips</li>
    </ul>
    </script>
  </body>
</html>
```



---

**LET'S TAKE A LOOK**

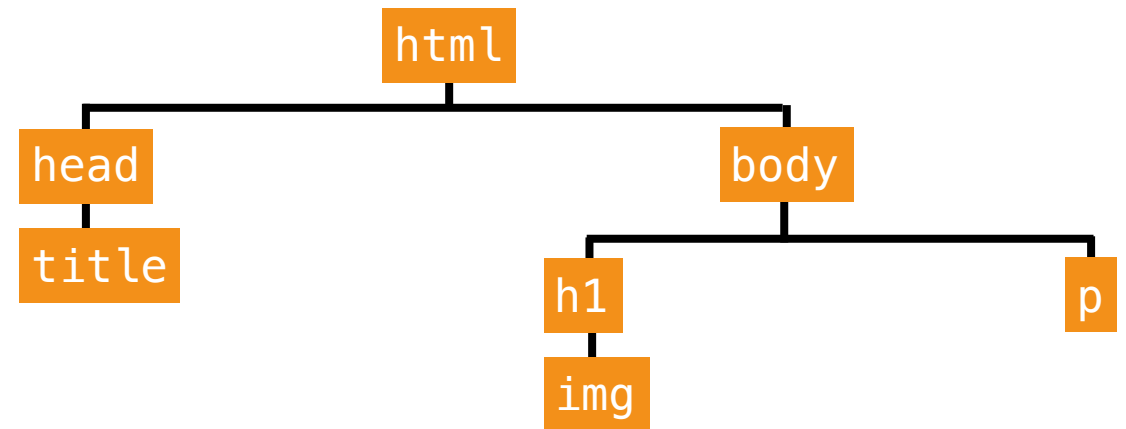
---



# Web page elements

```
<html>
  <head>
    <title>JavaScript Basics</title>
  </head>
  <body>
    <h1>
      
    </h1>
    <p>First, master HTML and CSS.</p>
  </body>
</html>
```

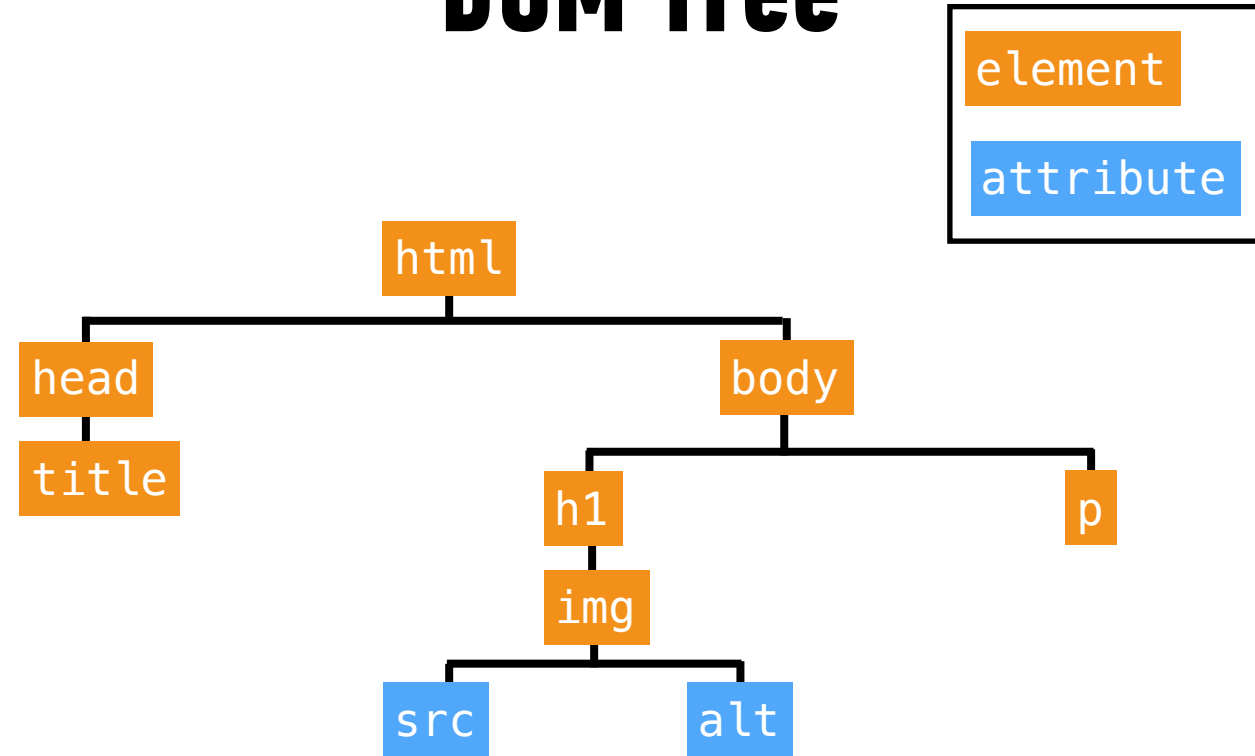
# DOM Tree



# Web page elements

```
<html>
  <head>
    <title>JavaScript Basics</title>
  </head>
  <body>
    <h1>
      
    </h1>
    <p>First, master HTML and CSS.</p>
  </body>
</html>
```

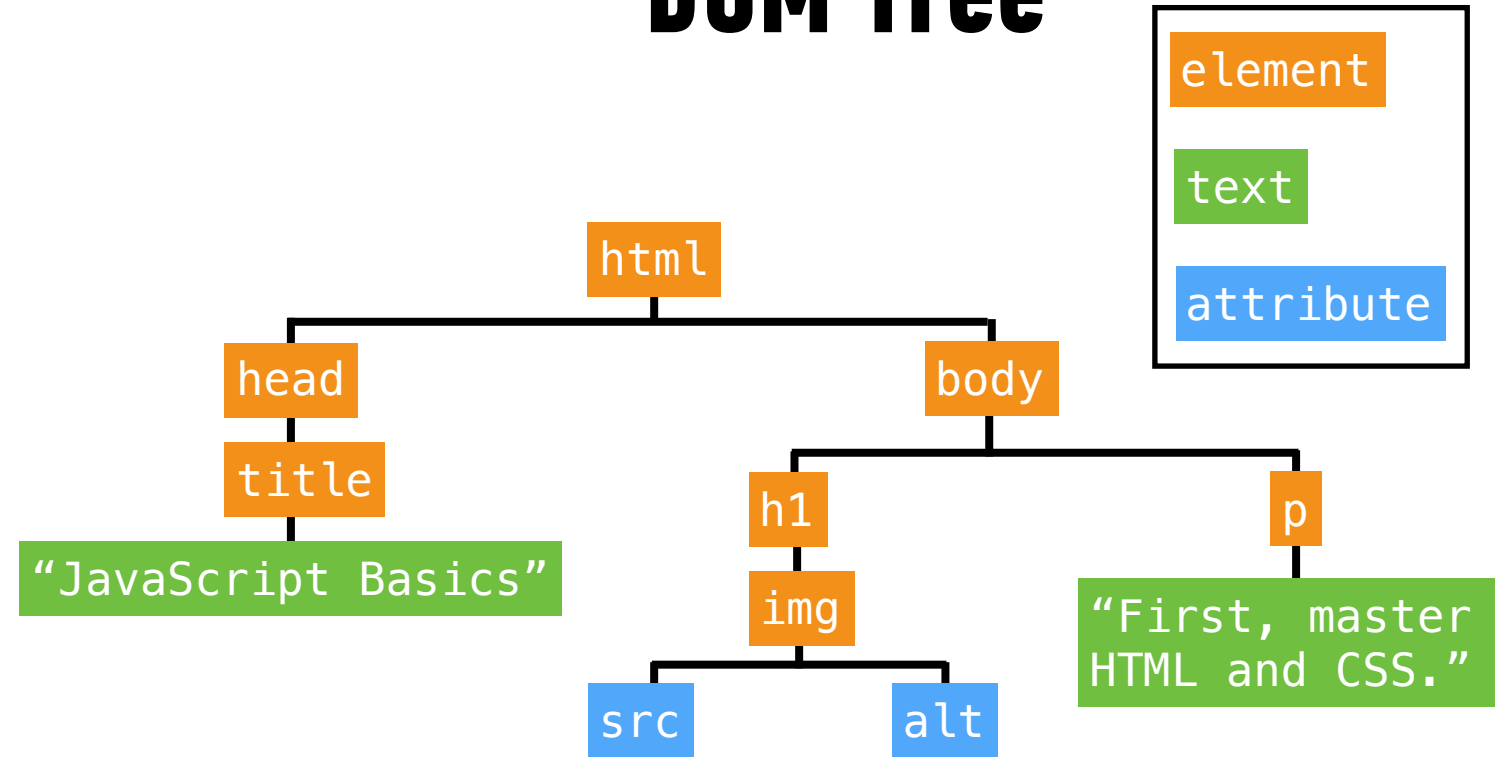
## DOM Tree



# Web page elements

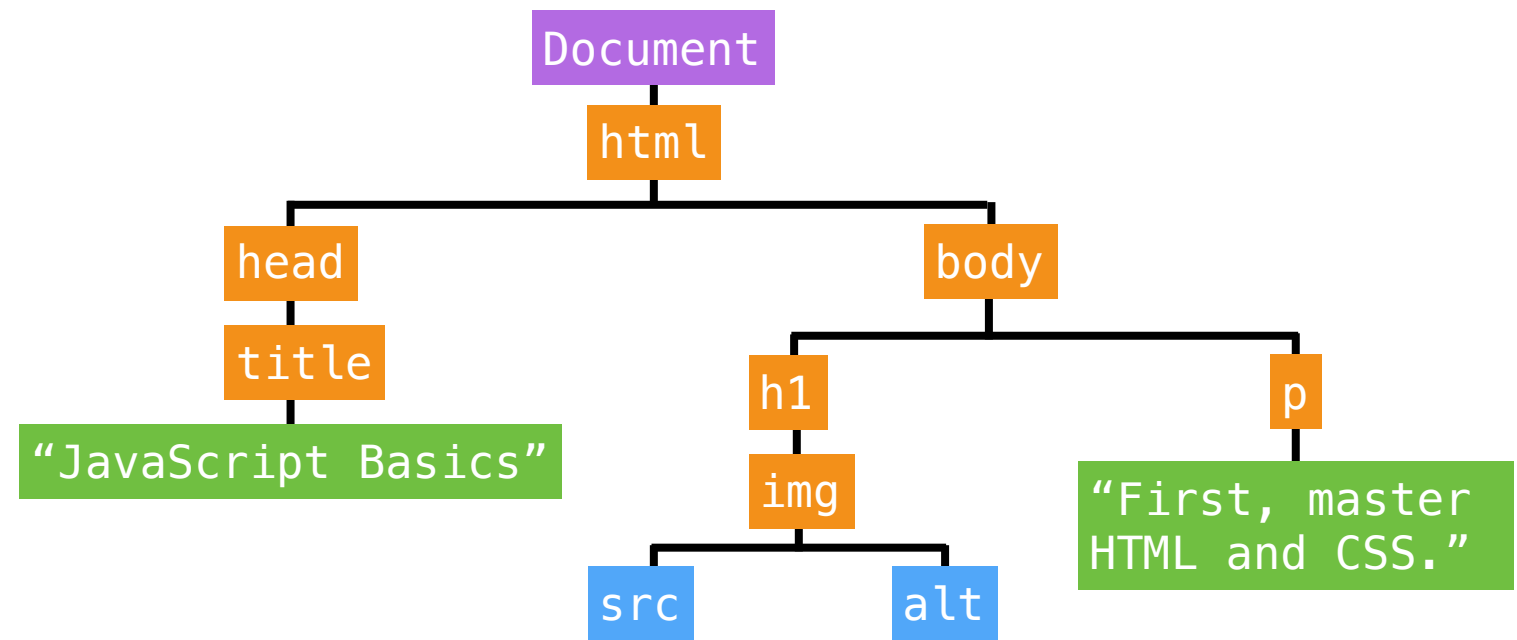
```
<html>
  <head>
    <title>JavaScript Basics</title>
  </head>
  <body>
    <h1>
      
    </h1>
    <p>First, master HTML and CSS.</p>
  </body>
</html>
```

## DOM Tree



# The Document object

- › Created by the browser
- › Contains all web page elements as descendant objects
- › Also includes its own properties and methods



---

# EXERCISE

---



## EXERCISE

### **KEY OBJECTIVE**

---

- Identify differences between the DOM and HTML

### **TYPE OF EXERCISE**

---

- Pairs

### **TIMING**

---

*2 min*

1. How is the DOM different from a page's HTML?

# REFERENCING A SCRIPT IN HTML

script element at the bottom of the  
body element

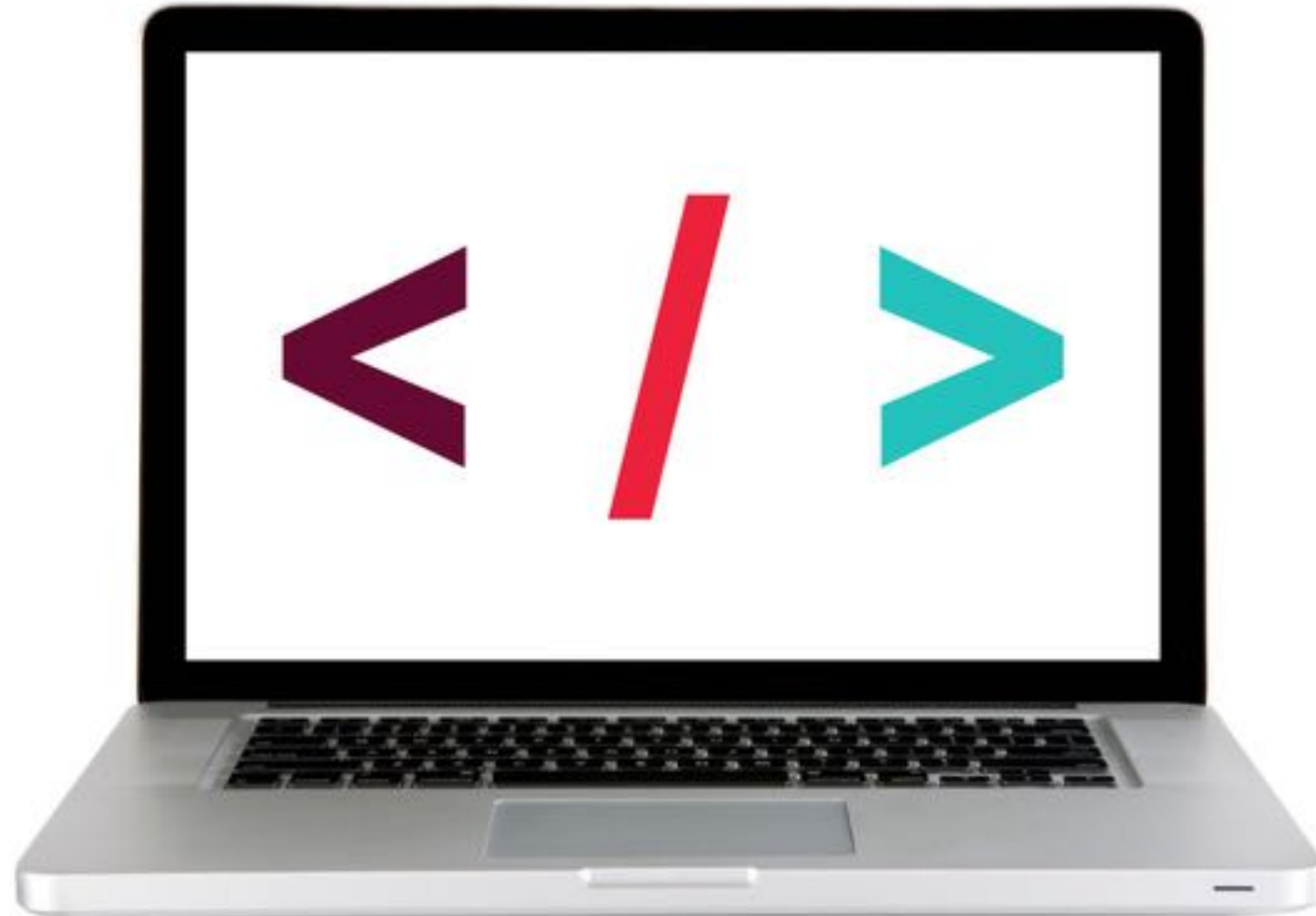
just before the closing `</body>` tag

```
<html>
  <head>
  </head>
  <body>
    <h1>JavaScript resources</h1>
    <script src="script.js"></script>
  </body>
</html>
```

---

**LET'S TAKE A LOOK**

---





# Selecting an element in the DOM

- `getElementById()`
- `getElementsByClassName()`
- `getElementsByTagName()`
- `querySelector()`
- `querySelectorAll()`

Let us select DOM elements  
using CSS selector syntax



# querySelector()

- Takes a single argument, a string containing CSS selector

## HTML

```
<body>
...
<p id="main">Lorem ipsum</p>
...
</body>
```

## JavaScript

```
document.querySelector('#main');
```

# querySelector()

- Selects the **first** DOM element that matches the specified CSS selector

```
<body>
...
<ul>
  <li>Lorem ipsum</li>
  <li>Lorem ipsum</li>
  <li>Lorem ipsum</li>
</ul>
...
</body>
```

JavaScript

```
document.querySelector('li');
```

# querySelectorAll()

- Takes a single argument, a string containing CSS selector
- Selects all DOM elements that match this CSS selector
- Returns a NodeList, which is similar to an array

```
<body>
...
<ul>
  <li>Lorem ipsum</li>
  <li>Lorem ipsum</li>
  <li>Lorem ipsum</li>
</ul>
...
</body>
```

JavaScript

```
document.querySelectorAll('li');
```

# **What can we do with a selected element?**

- Get and set its text content with the `innerHTML` property
- Get and set its attribute values by referencing them directly (`id`, `src`, etc.)

# innerHTML

- Gets the existing content of an element, including any nested HTML tags
- Sets new content in an element

```
var item = document.querySelector('li');  
  
console.log(item.innerHTML) // Gets value: "Lorem ipsum"  
  
item.innerHTML = 'Apples' // Sets value: 'Apples'
```

# className property

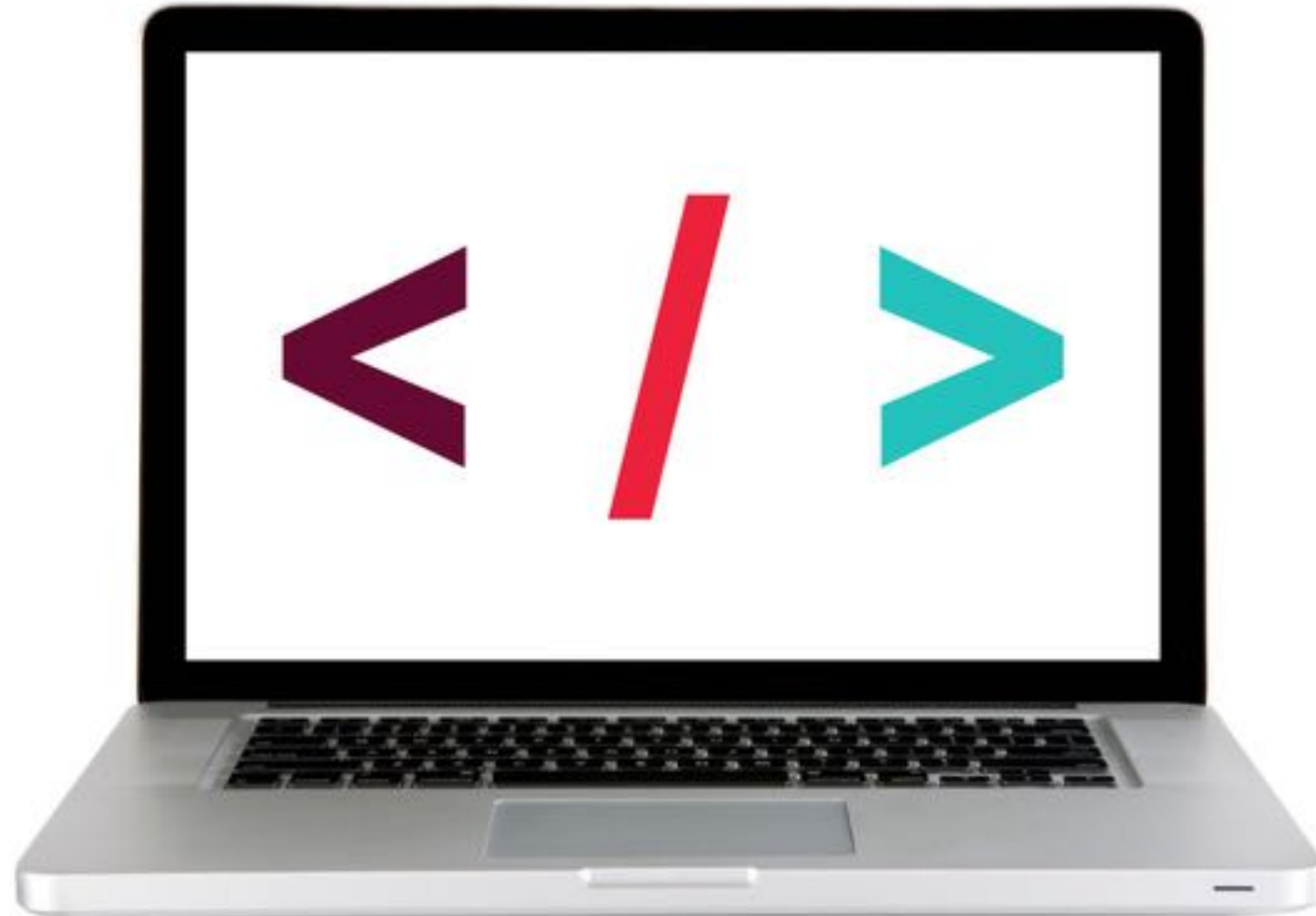
- Gets/sets an element's class attribute value
- CSS style sheet contains a style rule for each class
  - » Appearance of element changes based on which class is applied
  - » This is the best practice.

```
var item = document.querySelector('li');  
  
console.log(item.className) // Gets value: 'default'  
  
item.className = 'selected'  
// Sets value: 'selected'
```

---

**LET'S TAKE A LOOK**

---





# **Exit Tickets!**

**(Class #6)**

# **LEARNING OBJECTIVES – REVIEW**

- Identify likely objects, attributes, and methods in real-world scenarios
- Create JavaScript objects using object literal notation
- Implement and interface with JSON data
- Identify differences between the DOM and HTML.
- Explain and use JavaScript methods for DOM manipulation.

# **NEXT CLASS PREVIEW**

## **Intro to the DOM & jQuery**

- Explain and use JavaScript methods for DOM manipulation.
- Create DOM event handlers to respond to user actions
- Manipulate the DOM by using jQuery selectors and functions.
- Register and trigger event handlers for jQuery events.

# Q&A