Week 12: JavaScript Continued

Reviewing Last Week

- Console
    - console.log()
    - For debugging
- Data Types
    - strings, numbers, boolean (true/false)
    - concatenation (myVar + “ “ + myVar2);
- Syntax
    - Semicolon optional
    - // for commenting out
- Variables
    - keep things flexible
    - const vs. let
- Operators
    - equal to, not equal to
    - *, +, -, etc.
    - === or ==?
    - You can also use: <=, >=, etc.
- Arrays
    - Lists
    - zero-indexed (the index of the first element is zero)
    - Properties include length; so you can write: array.length() // returns: number of elements
- Random
    - Math.random()
    - Math.floor()

Conditional Logic

- if this, then that

    - if, if else, else

Conditional Statements

Conditional statements allow your code to perform certain commands *only if* certain conditions are met. These conditions can be based on:

- a user's input (is the password entered correct?)
- the current state (is it day or night?)
- the value of a certain element (is this person older than 18?)

Conditional statements are written as **if statements** with the question wrapped in `()`parentheses and the code to check if the question is true in `{}`curly braces.

```
let isItPartyTime = true;
if (isItPartyTime) {
  console.log("bye");
}

let partyTime = 5;

if (partyTime == 5) {
  console.log("Everybody dance!");
}
```

Sometimes we use conditional statements to compare values. You use the operators we looked at before to do so. Here they are again.

Comparison Operators

==	check is this equal to (2=="2" would be true, because the string gets converted to a number)
===	equal value and equal data type (2==="2" would be false, because the second 2 is a string)
!=	not equal

!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to

We can add a condition for when the condition isn't true using an `else` statement.

```
let partyTime = 7;

if (partyTime == 7) {
  console.log("peace out");
} else {
  console.log("stay seated...");
}
```

You can also add an `else if` statement for further control.

```
// :) :) :) :) :) :) :) :) :) :) :) :)

let guess = 2;

// how many emoticons are listed above?
if (guess < 5) {
  console.log("Too low");
} else if (guess > 5) {
  console.log("Too high");
} else if (guess == 5) {
  console.log("That's right!");
}
```

Try this out in a CodePen or your Console:

- Create a variable for the current time.
  - Hint, try typing in `new Date()` in the console versus `Date.now()`
  - Hint #2: Once you have a variable, add `.getHours();`
- Using the variable, you just created, have the console log “peace out” if it’s 7pm. Otherwise log, “Nope not yet”
- Now create a new conditional statement that uses the same variable. If it’s 7pm, have the console log “peace out”, else, if it’s between 4 and 7pm log “Working hard,” otherwise log “Class is not in session!” (hint: use `&&` to indicate two rules at once)

Let’s Add to Our JavaScript Vocabulary: Functions

A function is a set of instructions.

For example, let’s write our morning routine in code. In other words, let’s “codify it.”

Mine: I get up around 7am (unless I went to bed late the night before or it’s a weekend), and then I: brush my teeth, have a coffee (flat white with almond if it’s cold, cold brew if it’s hot), read the news, have a shower, pick an outfit at random, check my email.

- Data Types?
- Variables?
- Operators?
- Conditional Logic?

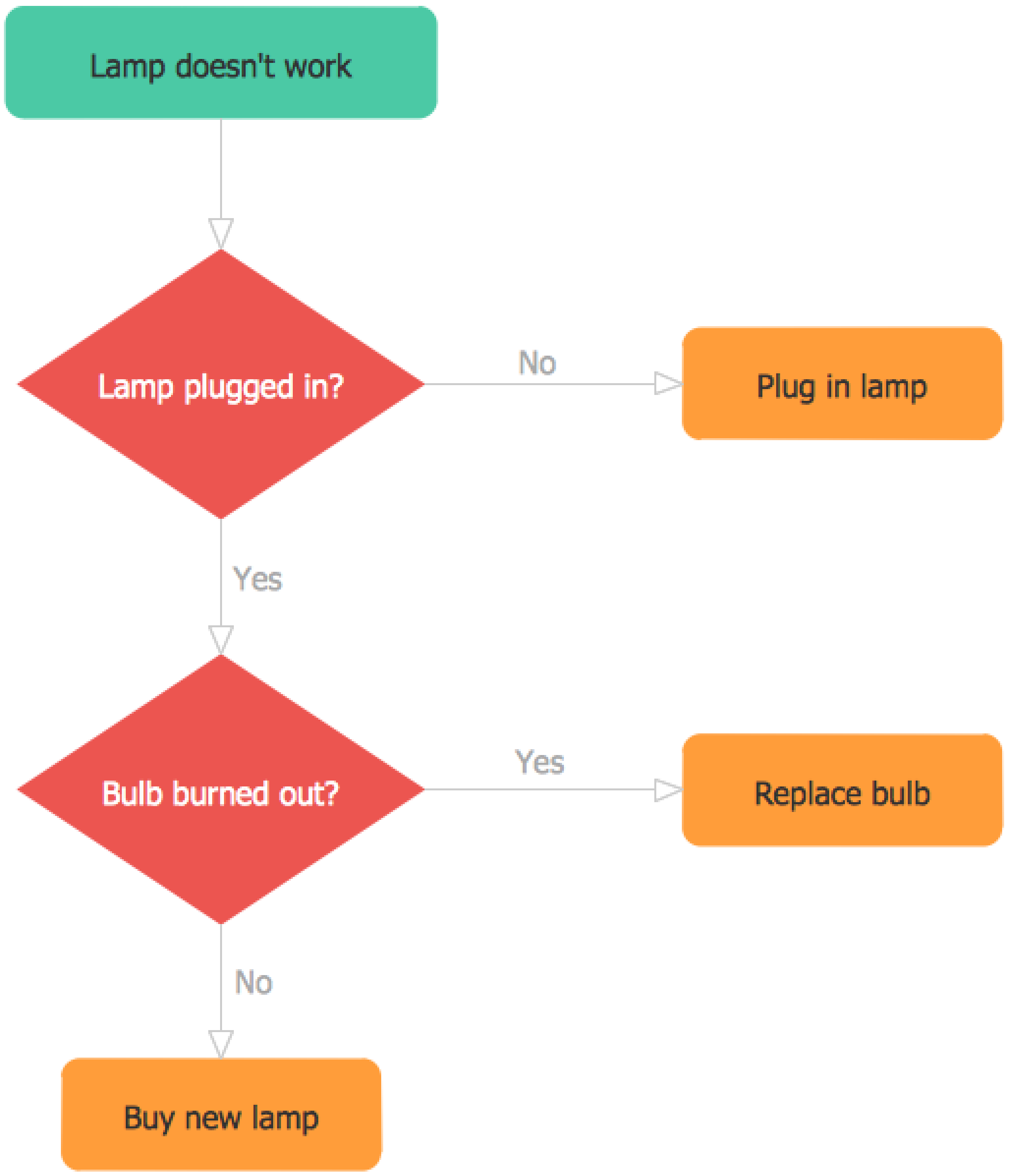
What else? Lists? Randomization?

How about something else? Passing someone you know on the street? Saying hello?

- Variables?
- Conditional logic?

It’s sad to think we’re programmatic and I don’t think we are (maybe without coffee ;)) . But computers are 🙃

Ever seen these flow charts, systems diagrams?



Everyday routines to explain Algorithms & Flowcharts. | by Diana Vilé | Medium

Now, how do we translate something like this to a function?

- Functions perform specific tasks and allow us to execute them at another point.

- Values can be passed into functions and used within them. We call these values arguments.
    - When defining a function, we add parameters, which are placeholders for the arguments.
- Functions always return a value. If no return value is specified, the function will return undefined.
- Functions need to be defined and called.

Defining Functions

```
let functionName = (parameterX, parameterY) => {
   // function instructions go here
}

let morningRoutine = (sleepTime, time) => {
  // function instructions go here
}

let greeting = (name) => {
  console.log("Hello" + name + "!");
  console.log(`Hello ${name}!`);
}

let toggleLightStatus = (currentLight) => {
    currentLight = !currentLight;
}

let toggleLightStatusTakeTwo = (lightOn) => {
  if (lightOn) {
     lightOn = false;
  } else {
     lightOn = true;
  }
}
```

```
let name = "Linda";
// console.log(`Hi, ${name}!`);

let greetStudent = (studentName, className) => {
  console.log(`Hi, ${studentName}. Welcome to ${className}`);
};

greetStudent(name, "Interaction");
```

- `toggleLightStatus`: The easy way to toggle a boolean is by assigning it to the opposite of its current value. it's a shorthand shortcut and concise.
    - The function `toggleLightStatusTakeTwo` is doing the same exact thing, but it's spelling it out explicitly with conditional statements.
- Another way to write strings with variables is by using **template literals** `` `${variable}` ``.
    - It's a backtick and then ${} to contain your variables.
    - It's a more legible way of the "" + "" + "" method we've been using.
    - Try changing your in-class exercise from last week to write out the answer using template literals.

Calling Functions

```
functionName(parameterX, parameterY);
```

```
// Define variables
let date = new Date;
let time = date.getHours();
let sleepTime = 23;

// Call Function
morningRoutine(sleepTime, time);
```

- I know you'd rather have the variable `SleepTime` change, and not always be 23 (i.e. 11pm). That's why it's a variable! We'll get into different ways to do this during this class and next.

```
// Defining variables
let name = "Julia"

// Call Function
greeting(name);
```

Defining the variables:

```
let names = [ "Emma", "Annie", "Will", "Jui", "Hanna", "Coral
", "Apollonia"];
let randomIndex = Math.floor(Math.random() * names.length);
let randomName = names[randomIndex];
```

Defining the function:

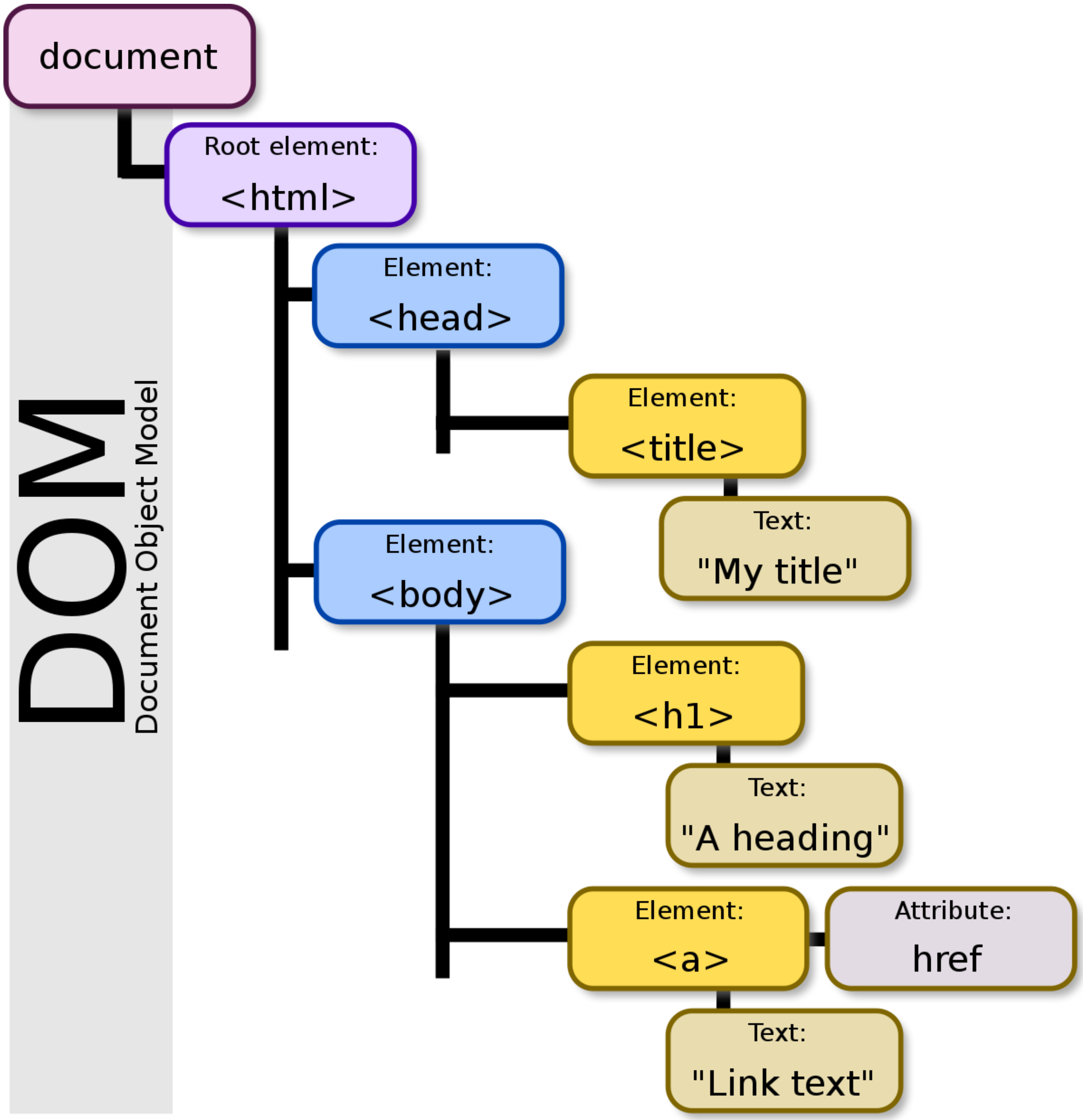
```
let greeting = (name) => {
  console.log("Hello" + name + "!");
  console.log(`Hello ${name}!`);
}
```

Calling the function:

```
greeting(randomName);
```

- randomName is the **argument**; name is the **parameter**
    - randomName is the value; name is the placeholder
- If a function has two parameters, the arguments are matched with the parameters in the order they appear in the function call.

Applying these rules to manipulate your Document Object Model (DOM)



DOM tree and its components. Source: Wikipedia 2020.

The **Document Object Model (DOM)** is a programming interface for web documents.

- It represents the page so that programs can change the document structure, style, and content.
- The DOM represents the document as nodes and objects; that way, programming languages can interact with the page.

But how is this different from HTML?

- **HTML** is the *static* code that structures your content.
- **DOM** is the live, *dynamic* representation of that HTML Document, including the applied styles (CSS)

```
[Document]
  ├── [HTML]
  │     ├── [Head]
  │     │     └── [Style] (contains CSS)
  │     └── [Body]
  │           ├── [H1] { color: blue, text-align: center }
  │           ├── [P id="greeting"] { color: green, font-size: 2
0px }
  │           └── [Button]
```

It's not too important to know the difference. Just know they are different, and that you're making changes to the DOM when you use JavaScript, not directly to the HTML.

Let's Jump Into Query Selectors

- We use `querySelector` to interact with HTML elements.
- This is a tool to "query" or search the document by a CSS selector.
- To search your whole page, refer to it as the `document`.

```
let myNav = document.querySelector(".nav");
```

(Where else have we seen this syntax?)
(How do you know what language this is?)

HTML:

```
<div class="nav">
  ...
</div>
```

JS

```
let myNav = document.querySelector(".nav");
console.log("my nav element:", myNav);
```

CSS

```
.special {
  font-family: arial;
  font-size: 100px;
  color: red;
}
```

We can use the `classList` attribute to add, remove, and check whether an element has a class.

```
let navElement = document.querySelector(".nav");
navElement.classList.add("special");
```

Adding interactions with event listeners

We can create an interaction and trigger the above after an event, like a click.

To do so, we *listen* for user events with the `addEventListener` method of an element you select with `querySelector`.

`addEventListener` needs two arguments – the event you're "listening" for, and what happens when that event happens.

HTML: CodePen

```
<section>
    <div class="button">
      Click Me
    </div>
</section>
```

CSS

```
body {
  background-color:  black;
}

section {
  display:  flex;
  flex-direction:  column;
  justify-content: center;
  align-items:  center;
  height:  100vh;
}

.button {
  padding: 20px 50px;
  border-radius: 30px;
  background-color:  white;
  color:  black;
  font-size:  60px;
  font-size: 20px;
  text-align:  center;
  transition:  all 100ms;
}
```

```
.button:hover {
  background-color: #DEB887;
  cursor:  pointer;
}
```

JS

```
turnOnLight = () => {
  console.log("Lights on!");
}

let button = document.querySelector(".button");
button.addEventListener("click", turnOnLight);
```

In the console you'll see that every time you click on the button, the console counts the click.

JS

```
let body = document.body;

let turnOnLight = () => {
  body.classList.add("light");
  button.classList.add("buttonOn");
}

let button = document.querySelector(".button");
button.addEventListener("click", turnOnLight);
```

If you want to make it turn on and off, replace "add" with "toggle"

```
let body = document.body;

function turnOnLight() {
  body.classList.toggle("light");
  button.classList.toggle("buttonOn");
}

let button = document.querySelector(".button");
button.addEventListener("click", turnOnLight);
```

👉You try

Take a few minutes and create your own event on click.

Demos

1. Click button to append content

When you click the button, use the `appendChild()` function to add new content into some other `div` to represent the content area.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-s
cale=1">
  <link rel="stylesheet" type="text/css" href="main.css">
  <title>JavaScript Day 2</title>
</head>
<body>
  <div class="wrapper">
```

```
    <div class="button">
      Click Me
    </div>
  </div>
  <script src="main.js" type="text/javascript"></script>
</body>
</html>
```

```
body {
  background-color:  black;
  color:  white;
  font-family: arial;
}

.wrapper {
  display: flex;
  justify-content: center;
  align-items: center;
  height: 100vh;
}

h1 {
  font-weight:  normal;
}

.button {
  background-color: white;
  border-radius: 20px;
  padding: 20px 40px;
  color: black;
  font-size: 36px;
}

.button:hover {
  opacity: .8;
  cursor: pointer;
  transition: 90ms ease;
}
```

```


// If you use a <button> tag you can query the button directl
y. Otherwise, you can use a CSS class on the element you clic
k, and querySelector that class.
let button = document.querySelector(".button");
let wrapper = document.querySelector(".wrapper");

let myFunction = (event) => {
  console.log(event, event.target);
  let newItem = document.createElement("div");
  newItem.classList.add("new-content");
  newItem.innerHTML = "But wait, theres more ";
  wrapper.appendChild(newItem);
}

button.addEventListener("click",
});
```

Random Background Color From Array On Click

```
<!DOCTYPE html>
```

```
<html>
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-s
cale=1">
  <link rel="stylesheet" type="text/css" href="main.css">
  <title>Random BG Color on Click</title>
</head>
<body>
  <div class="wrapper">
    <div class="button">
      New Color
    </div>
  </div>
  <script src="main.js" type="text/javascript"></script>
</body>
</html>
```

```
body {
 background-color: #647e6b;
}

.wrapper {
  display: flex;
  justify-content: center;
  align-items: center;
  height: 100vh;
}

.button {
  background-color: black;
  color: white;
  font-size: 30px;
  padding: 30px;
  border-radius: 100px;
}

.button:hover {
  cursor: pointer;
  opacity: .8;
  transition: 8ms ease-out;
}
```

```
let button = document.querySelector(".button");
let colors = ["#255c34", "#726756", "#703a56", "#953e00"];
let body = document.querySelector("body");

function newColor() {
  let randIndex = Math.floor(Math.random() * colors.length);
  let randColor = colors[randIndex];
  body.style.backgroundColor = randColor;
}

button.addEventListener("click", newColor);
```

Let’s practice linking these via script files. Open up your project.

Tasks

Manipulate the Demos: Working by yourself or with a partner, manipulate each of the above examples to do something that builds on each interaction.