

# JavaScript

Mark Tabor and Abby Chou

# Recap

## HTML

- Describes **content** and **structure**
- What exists? How is it organized?

```
<!DOCTYPE html>
<html>
  <head>
    <title>A Cool Webpage</title>
  </head>
  <body>
    <p>
      programming is just spicy Googling
    </p>
  </body>
</html>
```

# Recap

## HTML

- Describes **content** and **structure**
- What exists? How is it organized?

```
<!DOCTYPE html>
<html>
  <head>
    <title>A Cool Webpage</title>
  </head>
  <body>
    <p>
      programming is just spicy Googling
    </p>
  </body>
</html>
```

## CSS

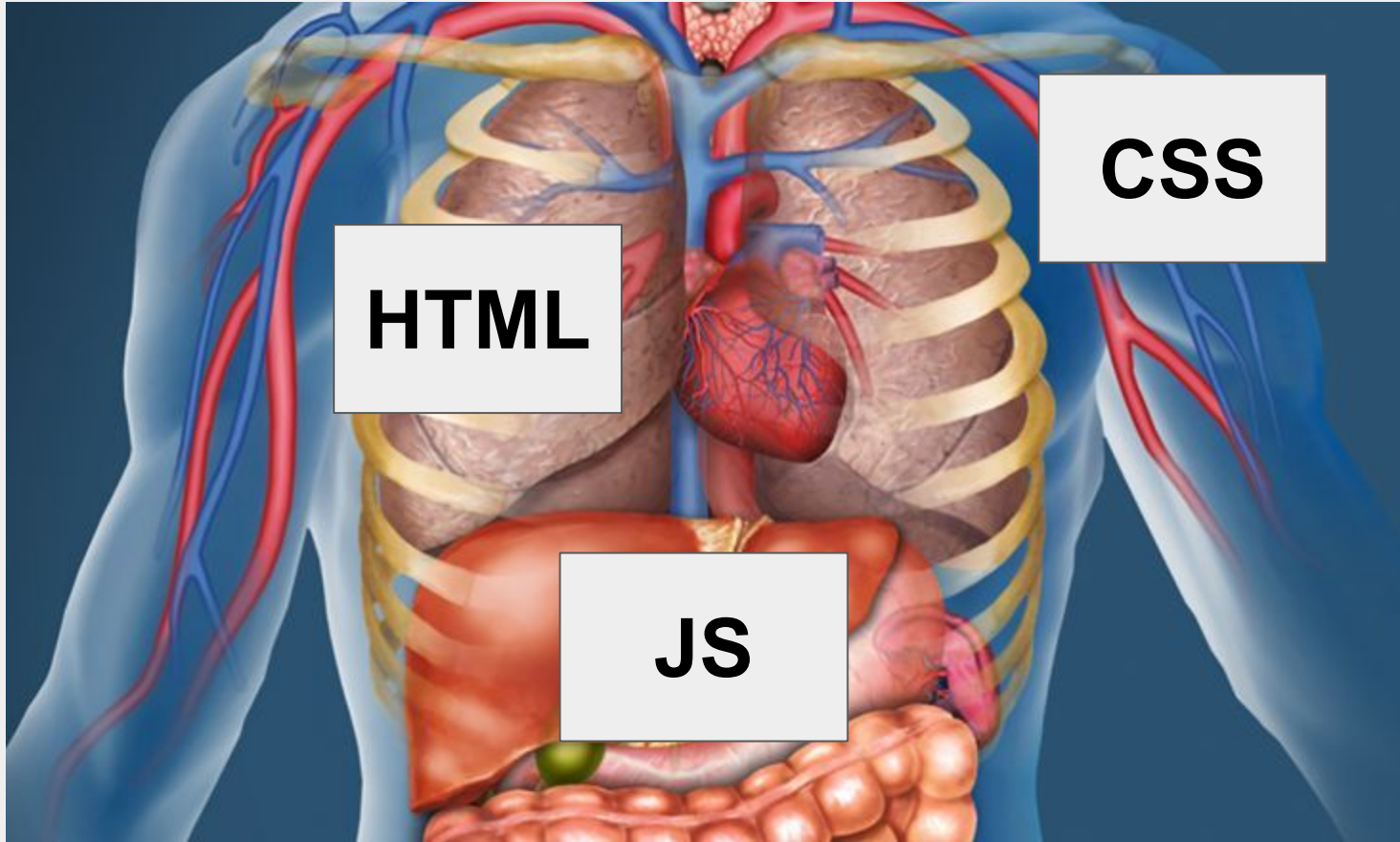
- Describes the **presentation**
- Colors! Fonts! Alignment, margins, borders, shading, and more

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

p {
  font-family: Helvetica;
  font-size: 16px;
}
```

# JavaScript is...

- ... a programming language that **manipulates** the content of a web page
- ... how we take HTML + CSS and make it **interactive!**
- ... used by a vast majority of websites and web applications
- ... not related to Java 🙄



**HTML**

**CSS**

**JS**

# Where does it go?

Where can we run JavaScript code?

1. The browser console
  - a. Windows
    - i. Chrome: Ctrl + Shift + J
    - ii. Firefox: Ctrl + Shift + J
  - b. Mac
    - i. Chrome: Cmd + Option + J
    - ii. Firefox: Cmd + Shift + J
2. Tied to our HTML file (more on that later!)

# How to JavaScript

# Types

JavaScript has 5 primitive data types:

- Boolean (true, false)
- Number (12, 1.618, -46.7, 0, etc.)
- String ("hello", "world!", "12", "", etc.)
- Null
- Undefined



# Operators

Things (mostly) work how you would expect:

```
> 5+4
< 9
> 8-2
< 6
> 3*7
< 21
> 1/3
< 0.3333333333333333
> "cool string" + "cooler string"
< "cool stringcooler string"
>
```

arithmetic operators

# Operators

Things (mostly) work how you would expect:

```
> 5+4
< 9

> 8-2
< 6

> 3*7
< 21

> 1/3
< 0.3333333333333333

> "cool string" + "cooler string"
< "cool stringcooler string"

>
```

arithmetic operators

```
> 2 === 2
< true

> 6 !== 7
< true

> 15 < 11
< false

> 8 > 3
< true

> 19 <= 19
< true

>
```

(note the triple equals sign!)



comparison operators

# Why we don't use ==

So, we use === to check equality in JavaScript.

But what does == do?

It performs *type coercion*  
(i.e. forces the arguments  
to be of the same type  
before comparing them)

```
2 === 2;    // true
```

```
2 === "2";  // false
```

```
2 == 2;     // true
```

```
2 == "2";   // also true!
```

tl;dr don't use ==

# Syntax

```
// this function finds the GCD of two numbers
const greatestCommonDivisor = (a, b) => {
  while (b !== 0) {
    const temp = b;
    b = a % b;
    a = temp;
  }
  return a;
}
```

```
const x = 50;
const y = 15;
const gcd = greatestCommonDivisor(x, y); // 5
```

Every statement in JavaScript ends with a semicolon;

Whitespace is ignored. (but can improve readability)

Curly braces denote where **blocks** begin and end.

These are **comments**. It doesn't affect how the code runs, but you should use them to keep your codebase readable!

# Defining variables

```
let myBoolean = true;  
let myNumber = 12;  
let myString = "Hello World!";  
  
myBoolean = false;  
myNumber = -5.6;  
myString = "";
```

# Defining variables

JavaScript convention  
is to name variables  
using **camelCase**.

```
let myBoolean = true;  
let myNumber = 12;  
let myString = "Hello World!";  
  
myBoolean = false;  
myNumber = -5.6;  
myString = "";
```

# Defining constants

To define a variable which *cannot* be re-assigned later:

```
const answerToLife = 6.148;
```

```
// this WILL NOT work!!!
```

```
answerToLife = 42;
```

# let vs. const

Why bother using `const` when `let` exists?



# let vs. const

Why bother using `const` when `let` exists?

Safe code practices! If something should never be changed, don't let it change :)

```
const secondsPerMinute = 60;  
// if this needs to be changed, then  
// we have bigger issues to address
```

# let vs. var

tl;dr please don't use **var**

```
let userLoggedIn = true;
```

```
var userLoggedIn = true;
```

# let vs. var

tl;dr please don't use **var**

```
let userLoggedIn = true;
```

```
var userLoggedIn = true;
```

technical details (Google it if you're interested):

**let** is block-scoped

**var** is function-scoped

**let** exists because people kept getting bugs when trying to use **var**

Questions?

# null vs. undefined

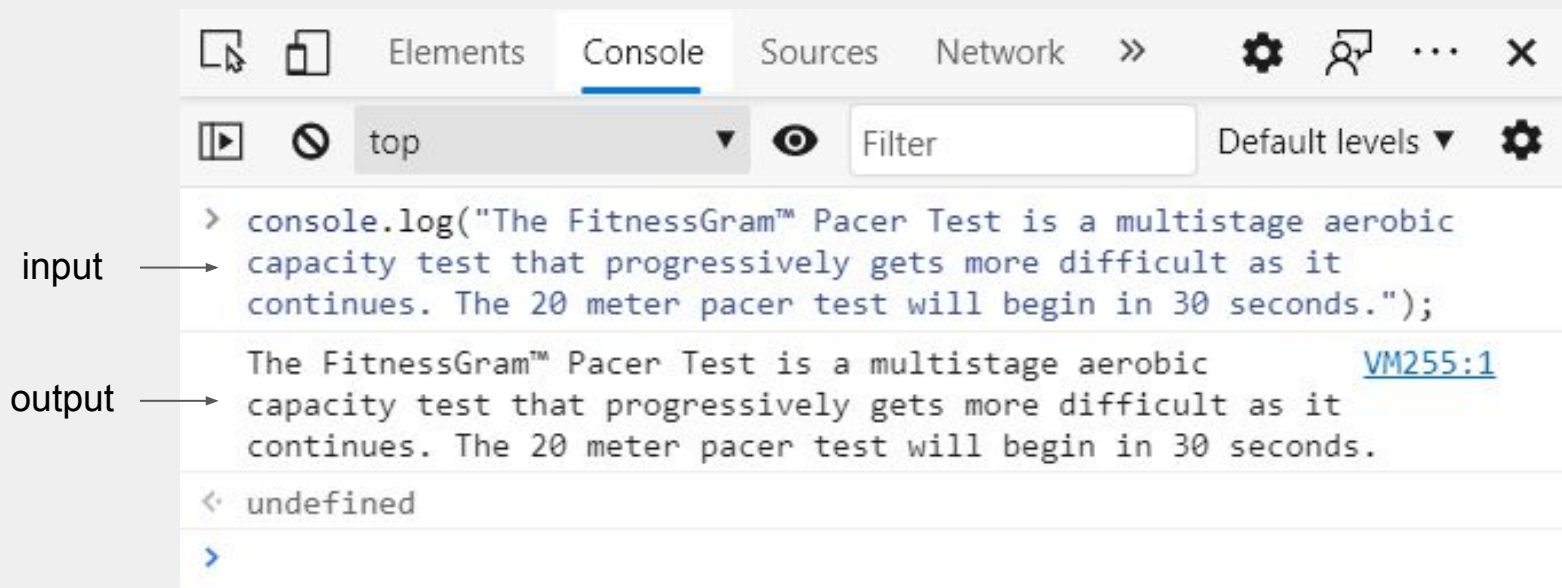
undefined means “declared but not yet assigned a value”

null means “no value”

```
let firstName;  
// currently, firstName is undefined  
  
firstName = "Albert";  
// firstName has now been assigned to a value  
  
firstName = null;  
// we can explicitly "empty" the variable
```

# Output

`console.log()` writes to the JavaScript console:



# Output

Handy for quick debugging!

```
let salary = 30000;  
salary = salary + 5000;  
salary = salary * 2;  
  
console.log(salary);  
// should output 70000
```

# Output

Can also console log with *template strings* for more descriptive logging

```
const a = 5;  
const b = 10;  
  
console.log(`a * b = ${a * b}`);
```



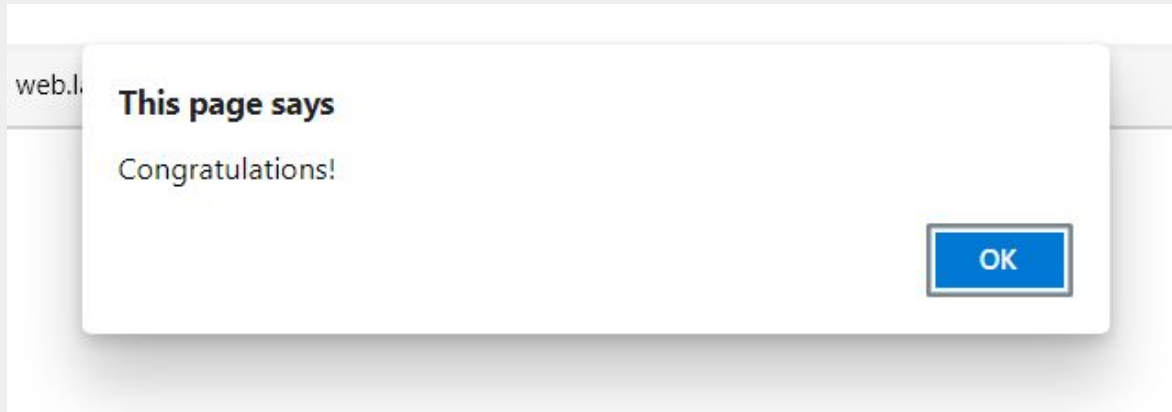
```
● marktabor@Marks-MacBook-Pro Lectures % node js1.js  
a * b = 50
```

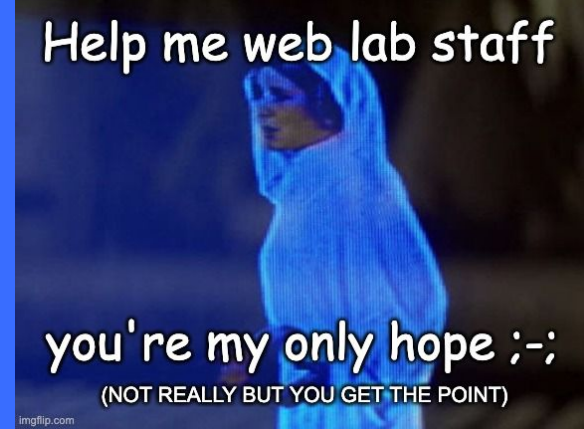
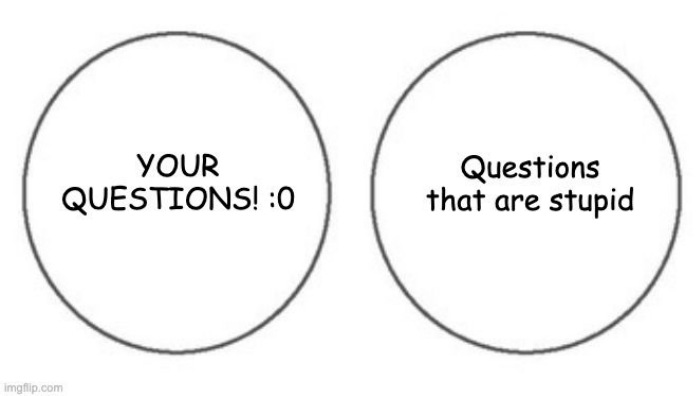


# Alerts

`alert()` generates a pop-up notification with the given content.

```
alert("Congratulations!");
```





# Questions?



# Arrays

For when you want to store a sequence of (ideally similar) items:

```
// initialize
let pets = ["flower", 42, false, "bird"]

// access
console.log(pets[3]); // "bird"

// replace
pets[2] = "hamster"; // ["flower", 42, "hamster", "bird"]
```

# Arrays

```
// initialize  
let pets = ["cat", "dog", "guinea pig", "bird"];  
  
// remove from end  
pets.pop(); // ["cat", "dog", "guinea pig"]  
  
// add to end  
pets.push("rabbit"); // ["cat", "dog", "guinea pig", "rabbit"]
```

# Conditionals

We often want to perform different actions in response to different conditions.

For this, we use the **conditional operators** `if`, `else`, and `else if`:


```
if (hour < 12) {  
    console.log("Good morning!");  
} else if (hour < 16) {  
    console.log("Good afternoon!");  
} else if (hour < 20) {  
    console.log("Good evening!");  
} else {  
    console.log("Good night!");  
}
```

# Conditionals

We often want to perform different actions in response to different conditions.

For this, we use the **conditional operators** `if`, `else`, and `else if`:

Note the indent (tab)!  
It's not necessary, but  
it will make your code  
much more readable.



```
if (hour < 12) {  
    console.log("Good morning!");  
} else if (hour < 16) {  
    console.log("Good afternoon!");  
} else if (hour < 20) {  
    console.log("Good evening!");  
} else {  
    console.log("Good night!");  
}
```

# While loops

What if we want to repeat an action *as long as* some condition is satisfied?

```
let z = 1;

while (z < 1000) {
  z = z * 2;
  console.log(z);
}
```

2

4

8

16

32

64

128

256

512

1024

# For loops

Useful when we want to iterate through indices:

I love my cat
I love my dog
I love my guinea pig
I love my bird

```
const pets = ["cat", "dog", "guinea pig", "bird"];

for (let i = 0; i < pets.length; i++) {
  const phrase = "I love my " + pets[i];
  console.log(phrase);
}
```



## For ... of ...

A more “pythonic” way of iterating:

I love my cat
I love my dog
I love my guinea pig
I love my bird

Requires the keyword **of** instead of **in**

```
const pets = ["cat", "dog", "guinea pig", "bird"];
```

```
for (const animal of pets) {
```

```
  const phrase = "I love my " + animal;
```

```
  console.log(phrase);
```

```
}
```

# Questions?

Comments?

Concerns?

Wait, we don't have a primitive data type for this



# Objects

A JavaScript **object** is a collection of **name:value** pairs.

```
const myCar = {  
  make    : "Ford",  
  model   : "Mustang",  
  year    : 2005,  
  color   : "red"  
};
```

# Accessing properties

There are two ways to access object properties, if you know the property name:

```
const myCar = {  
  make    : "Ford",  
  model   : "Mustang",  
  year    : 2005,  
  color   : "red"  
};  
  
console.log(myCar.model);    // "Mustang"  
console.log(myCar["color"]); // "red"
```

# Object destructuring

**Object destructuring** is a shorthand to obtain multiple properties at once.

# Object destructuring

**Object destructuring** is a shorthand to obtain multiple properties at once.

without object destructuring

```
const myCar = {  
  make    : "Ford",  
  model   : "Mustang",  
  year    : 2005,  
  color   : "red"  
};  
  
const make = myCar.make;  
const model = myCar.model;
```

# Object destructuring

**Object destructuring** is a shorthand to obtain multiple properties at once.

without object destructuring

```
const myCar = {  
  make    : "Ford",  
  model   : "Mustang",  
  year    : 2005,  
  color   : "red"  
};  
  
const make = myCar.make;  
const model = myCar.model;
```

with object destructuring

```
const myCar = {  
  make    : "Ford",  
  model   : "Mustang",  
  year    : 2005,  
  color   : "red"  
};  
  
const { make, model } = myCar;
```



# Equality...?

We use `===` to check if two *primitive* variables are equal in JavaScript.

```
2 === 2;           // true
2 === 3;           // false
"2" === "2";       // true
2 === "2";         // false
```

# Equality...?

We use `===` to check if two *primitive* variables are equal in JavaScript.

```
2 === 2;           // true
2 === 3;           // false
"2" === "2";       // true
2 === "2";         // false
```

But what does `===` mean for arrays and objects?

```
let arr1 = [1, 2, 3];
let arr2 = [1, 2, 3];

arr1 === arr2; // false!
```

```
let person1 = { name: "Bill Gates" };
let person2 = { name: "Bill Gates" };

person1 === person2; // false!
```

# Object references

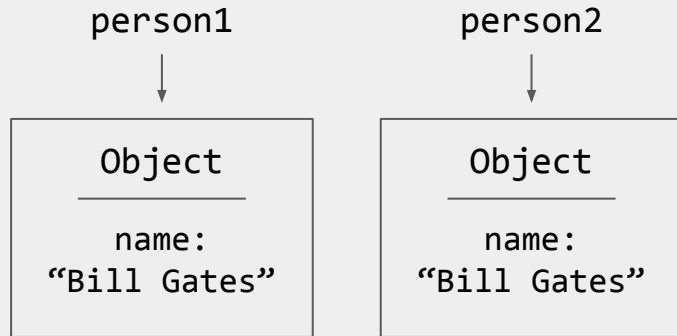
Object variables are **references** – they point to where the data is actually stored.

```
let person1 = { name: "Bill Gates" };  
let person2 = { name: "Bill Gates" };  
  
person1 === person2; // false!
```

=== checks if the *references* are equal.

Two objects created separately are stored separately, so their references are different!

Same goes for arrays – two arrays created separately have different references.



# How to copy arrays and objects

It's not as simple as

```
let arr = [1, 2, 3];  
let copyArr = arr;
```

(Why not?)

# How to copy arrays and objects

It's not as simple as

```
let arr = [1, 2, 3];  
let copyArr = arr;
```

(Why not?)

One way to copy arrays and objects is to use the **spread** operator (`...`) like so:

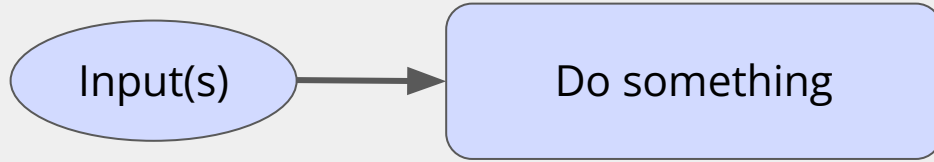
```
let arr = [1, 2, 3];  
let copyArr = [...arr];
```

```
let obj = { name: "Bill Gates" };  
let copyObj = { ...obj };
```

You could also manually copy over every item / property. But where's the fun in that?

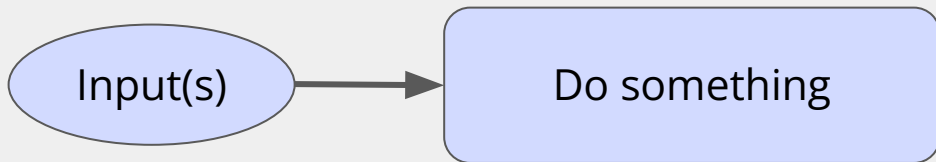
Questions?

# Functions



Sometimes, we want the function to **return** an output value.

# Functions



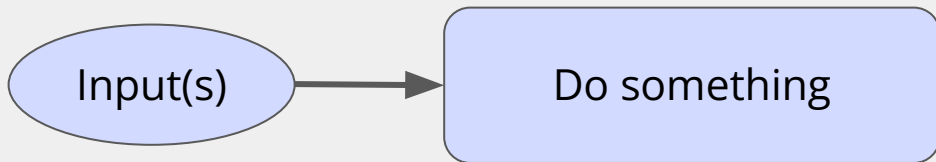
Sometimes, we want the function to **return** an output value.

JS Function Syntax:

```
(parameters) => { body };
```



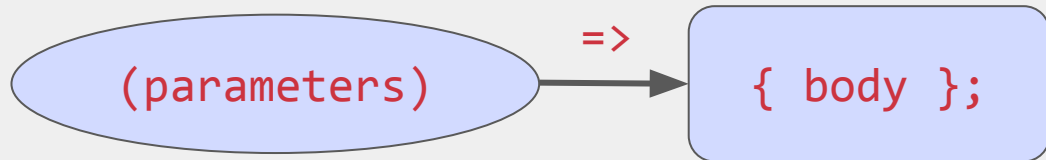
# Functions



Sometimes, we want the function to **return** an output value.

JS Function Syntax:

`(parameters) => { body };`



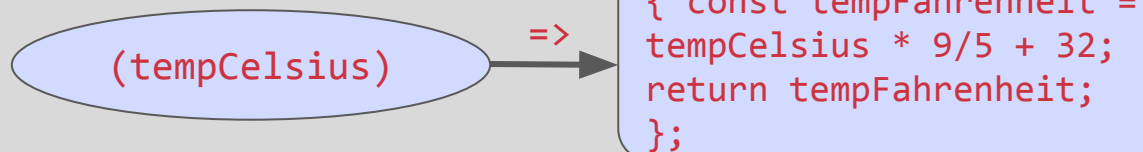
```
(tempCelsius) => {  
  const tempFahrenheit = tempCelsius * 9/5 + 32;  
  return tempFahrenheit;  
}
```

(tempCelsius)

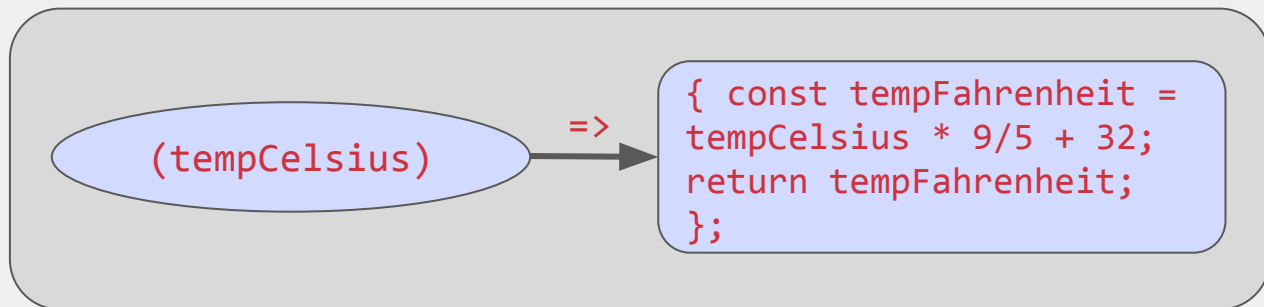
=>

```
{ const tempFahrenheit =  
tempCelsius * 9/5 + 32;  
return tempFahrenheit; };
```

```
(tempCelsius) => {  
  const tempFahrenheit = tempCelsius * 9/5 + 32;  
  return tempFahrenheit;  
}
```



```
(tempCelsius) => {  
  const tempFahrenheit = tempCelsius * 9/5 + 32;  
  return tempFahrenheit;  
}
```



What do we need to add so we can call this function?

```
(tempCelsius) => {  
  const tempFahrenheit = tempCelsius * 9/5 + 32;  
  return tempFahrenheit;  
}
```

celsiusToFahrenheit



(tempCelsius)

=>

```
{ const tempFahrenheit =  
tempCelsius * 9/5 + 32;  
return tempFahrenheit;  
};
```

```
(tempCelsius) => {  
  const tempFahrenheit = tempCelsius * 9/5 + 32;  
  return tempFahrenheit;  
}
```

celsiusToFahrenheit



(tempCelsius)

=>

```
{ const tempFahrenheit =  
tempCelsius * 9/5 + 32;  
return tempFahrenheit;  
};
```

```
const celsiusToFahrenheit = (tempCelsius) => {  
  const tempFahrenheit = tempCelsius * 9/5 + 32;  
  return tempFahrenheit;  
};
```

```
(tempCelsius) => {  
  const tempFahrenheit = tempCelsius * 9/5 + 32;  
  return tempFahrenheit;  
}
```

celsiusToFahrenheit



(tempCelsius)

=>

```
{ const tempFahrenheit =  
tempCelsius * 9/5 + 32;  
return tempFahrenheit;  
};
```

```
const celsiusToFahrenheit = (tempCelsius) => {  
  const tempFahrenheit = tempCelsius * 9/5 + 32;  
  return tempFahrenheit;  
};
```

```
(tempCelsius) => {  
  const tempFahrenheit = tempCelsius * 9/5 + 32;  
  return tempFahrenheit;  
}
```

celsiusToFahrenheit



(tempCelsius)

=>

```
{ const tempFahrenheit =  
tempCelsius * 9/5 + 32;  
return tempFahrenheit;  
};
```

```
const celsiusToFahrenheit = (tempCelsius) => {  
  const tempFahrenheit = tempCelsius * 9/5 + 32;  
  return tempFahrenheit;  
};
```



```
(tempCelsius) => {  
  const tempFahrenheit = tempCelsius * 9/5 + 32;  
  return tempFahrenheit;  
}
```

celsiusToFahrenheit



(tempCelsius)

=>

```
{ const tempFahrenheit =  
tempCelsius * 9/5 + 32;  
return tempFahrenheit;  
};
```

```
const celsiusToFahrenheit = (tempCelsius) => {  
  const tempFahrenheit = tempCelsius * 9/5 + 32;  
  return tempFahrenheit;  
};
```

```
(tempCelsius) => {  
  const tempFahrenheit = tempCelsius * 9/5 + 32;  
  return tempFahrenheit;  
}
```

celsiusToFahrenheit



(tempCelsius)

=>

```
{ const tempFahrenheit =  
tempCelsius * 9/5 + 32;  
return tempFahrenheit;  
};
```

```
const celsiusToFahrenheit = (tempCelsius) => {  
  const tempFahrenheit = tempCelsius * 9/5 + 32;  
  return tempFahrenheit;  
};
```

```
(tempCelsius) => {  
  const tempFahrenheit = tempCelsius * 9/5 + 32;  
  return tempFahrenheit;  
}
```

celsiusToFahrenheit



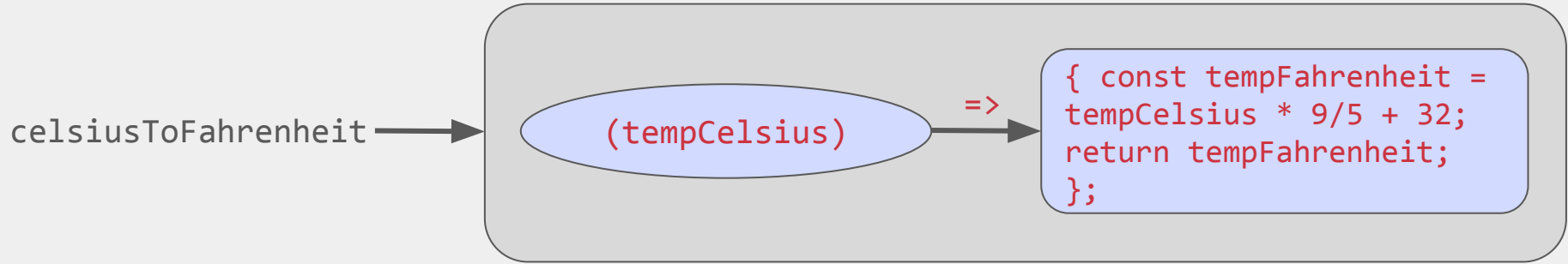
(tempCelsius)

=>

```
{ const tempFahrenheit =  
tempCelsius * 9/5 + 32;  
return tempFahrenheit;  
};
```

```
const celsiusToFahrenheit = (tempCelsius) => {  
  const tempFahrenheit = tempCelsius * 9/5 + 32;  
  return tempFahrenheit;  
};
```

Functions are values stored in memory, just like strings, ints, etc. So we can assign variables to point to them!



```
const roomTemp = celsiusToFahrenheit(26); // will assign 78.8 to roomTemp
```

# Callback functions

In JavaScript, functions can be passed around like any other values.

This means we can give a **“callback”** function as an argument to another function!

# Quick Practice!

Write a function that prints a simple message to the console.

Syntax reference:

`(parameters) => { body };` (but we don't need any parameters)

`console.log("your message");`

Remember to give your function a name!

## Example: setTimeout

```
const printSomething = () => {  
  console.log("i sleep");  
}
```

setTimeout() calls a function after a specified delay. It takes 2 parameters: the function to call (aka callback function), and the delay (in milliseconds).

## Example: setTimeout

```
const printSomething = () => {  
  console.log("i sleep");  
}
```

setTimeout() calls a function after a specified delay. It takes 2 parameters: the function to call (aka callback function), and the delay (in milliseconds).

How can we print our message after 5 seconds? (multiple-select)

A. `setTimeout(printSomething, 5000);`

B. `setTimeout(printSomething(), 5000);`

C. `setTimeout(() => { console.log("i sleep"); }, 5000);`



## Example: setTimeout

```
const printSomething = () => {  
  console.log("i sleep");  
};
```

setTimeout() calls a function after a specified delay. It takes 2 parameters: the function to call, and the delay (in milliseconds).

How can we print our message after 5 seconds?

- A. `setTimeout(printSomething, 5000);`
- B. `setTimeout(printSomething(), 5000);`
- C. `setTimeout(() => { console.log("i sleep"); }, 5000);`

DON'T DO THIS!  
printSomething() is  
whatever printSomething  
**returns**, not the function  
itself

# What's going on in Option C?

```
const tempC = 26;  
let tempF = celsiusToFahrenheit(tempC);  
  
tempF = celsiusToFahrenheit(26);
```

These are equivalent...



# What's going on in Option C?

```
const tempC = 26;  
let tempF = celsiusToFahrenheit(tempC);  
  
tempF = celsiusToFahrenheit(26);
```

These are equivalent...

```
const printSomething = () => {  
  console.log("i sleep");  
};  
setTimeout(printSomething, 5000);  
  
setTimeout(() => { console.log("i sleep"); }, 5000);
```

And so are these!

# What's going on in Option C?

```
const tempC = 26;  
let tempF = celsiusToFahrenheit(tempC);  
  
tempF = celsiusToFahrenheit(26);
```

These are equivalent...

```
const printSomething = () => {  
  console.log("i sleep");  
};  
  
setTimeout(printSomething, 5000);  
  
setTimeout(() => { console.log("i sleep"); }, 5000);
```

And so are these!  
(the highlighted parts  
are the same)

# setInterval() vs. setTimeout()

## setInterval()

Repeatedly calls a function with a delay between each call

Returns an intervalID which can be passed into **clearInterval()** to terminate

Arguments: func, delay, arg0, ..., argN

## setTimeout()

Sets a timer that executes a function when the timer ends

Returns an intervalID which can be passed into **clearTimeout()** to cancel the timer

Arguments: func, delay, arg0, ..., argN

Check out MDN docs for more info!

# Classes

Classes are used when we have multiple objects that all represent the same type of thing. They let us define the **attributes** and **methods** that every object of that type, or class, has.

```
class Rectangle {  
  constructor(width, height) {  
    this.width = width;  
    this.height = height;  
  }  
  
  getArea = () => {  
    return this.width * this.height;  
  };  
}
```

```
const rect = new Rectangle(6, 8);  
console.log(rect.getArea()); // 48
```

# Summary

JavaScript is how we make things happen!

- Declare variables using let, const.
- boolean, number, string, null, undefined
- functions, arrays, objects, classes
- if, else, while, for
- setInterval() vs. setTimeout()

**Up next:** hands-on JavaScript workshop!

Questions?



# Announcements

- Hw0 (Setup) [weblab.is/home](https://weblab.is/home)
  - Also has a link to flexbox froggy if you want to learn about/practice CSS flexbox!
  - Need to install Node by tomorrow!
- Office Hours tonight 7-9pm in 32-082!
  - Come for setup issues, or if you're new to programming and want to go over javascript or git!
- Milestone 0 (find a team and brainstorm 10 ideas) due eod Wednesday
- Recordings from today will be available as soon as we edit + post them (hopefully this evening). Slides and other links are all at [weblab.is/home](https://weblab.is/home)
- team formation mixer (right now! :D find teammates! )
- Give us feedback 👉👈 ([weblab.is/feedback](https://weblab.is/feedback))

Theme: "Send it or blend it"