JAVASCRIPT DEVELOPMENT

Sasha Vodnik, Instructor

HELLO!

1. Pull changes from the `svodnik/JS-SF-15-resources` repo to your computer:

   ‣ Open the terminal

   ‣ `cd` to the `Documents/JSD/JS-SF-15-resources` **directory**

   ‣ Type **`git pull`** and press **return**

2. In your code editor, open the following folder: `Documents/JSD/JS-SF-15-resources/03—conditionals-functions`

CONDITIONALS & FUNCTIONS

LEARNING OBJECTIVES

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

‣ Use Boolean logic to combine and manipulate conditional tests.

‣ Use `if/else` conditionals to control program flow.

‣ Differentiate among true, false, truthy, and falsy.

‣ Describe how parameters and arguments relate to functions

‣ Create and call a function that accepts parameters to solve a problem

‣ Define and call functions defined in terms of other functions

‣ Return a value from a function using the `return` keyword

‣ Define and call functions with argument-dependent return values

AGENDA

‣ Comparison operators

‣ Logical operators

‣ Conditional statements

‣ Functions

WEEKLY OVERVIEW

WEEK 2	Arrays & Loops / Conditionals & Functions
WEEK 3	Scope & objects / Slack bot lab
WEEK 4	Objects & JSON / DOM & jQuery

EXIT TICKET QUESTIONS

1. I am curious about why the console.log portion is necessary in Javascript. What exactly is the function of that statement?

2. Is there anytime when a javascript method cannot be used such as in another method or a function.

3. Why is the array reduce method called the way it is (compared to its functionality)?

4. I would like to see some more on how math is different in JS. For example cartWithTax * salesTax is a lot different than 6 * 7

EXIT TICKET SUGGESTIONS

1. Asking more questions to the class before presenting answers. I'd like to be able to take a second to think what might happen before seeing what would happen.

How to you decide what to have for dinner?

‣ What factors do you consider?

‣ How do you decide between them?

CONDITIONALS

CONDITIONAL STATEMENTS

‣ Decide which blocks of code to execute and which to skip, based on the results of tests that we run

‣ Known as **control flow statements**, because they let the program make decisions about which statement should be executed next, rather than just going in order

`if` STATEMENT

```
if (expression) {
    code
}
```
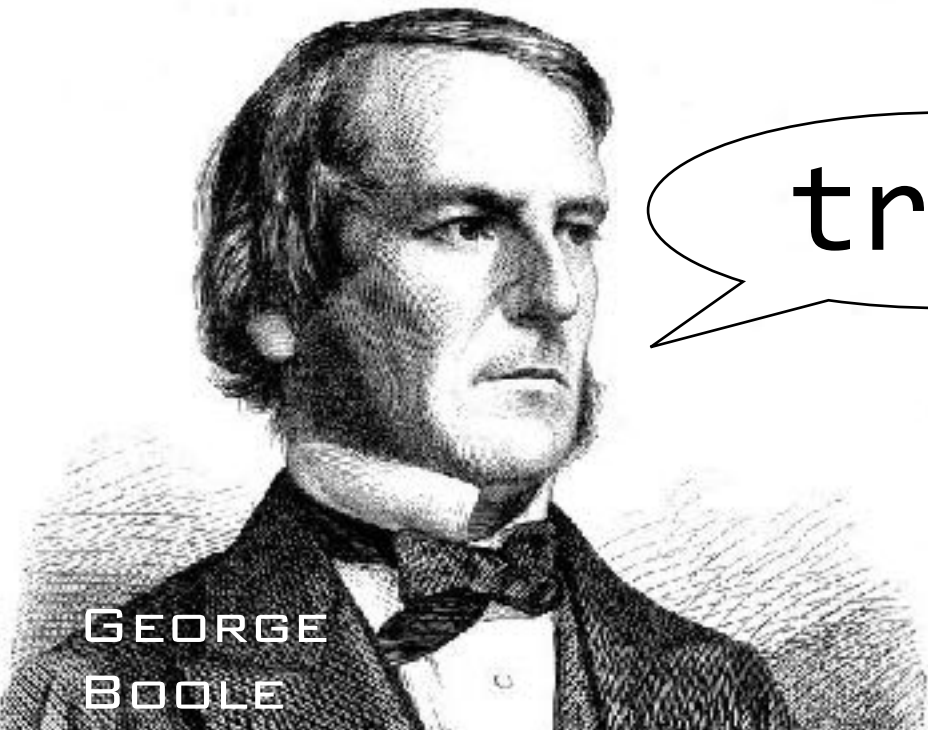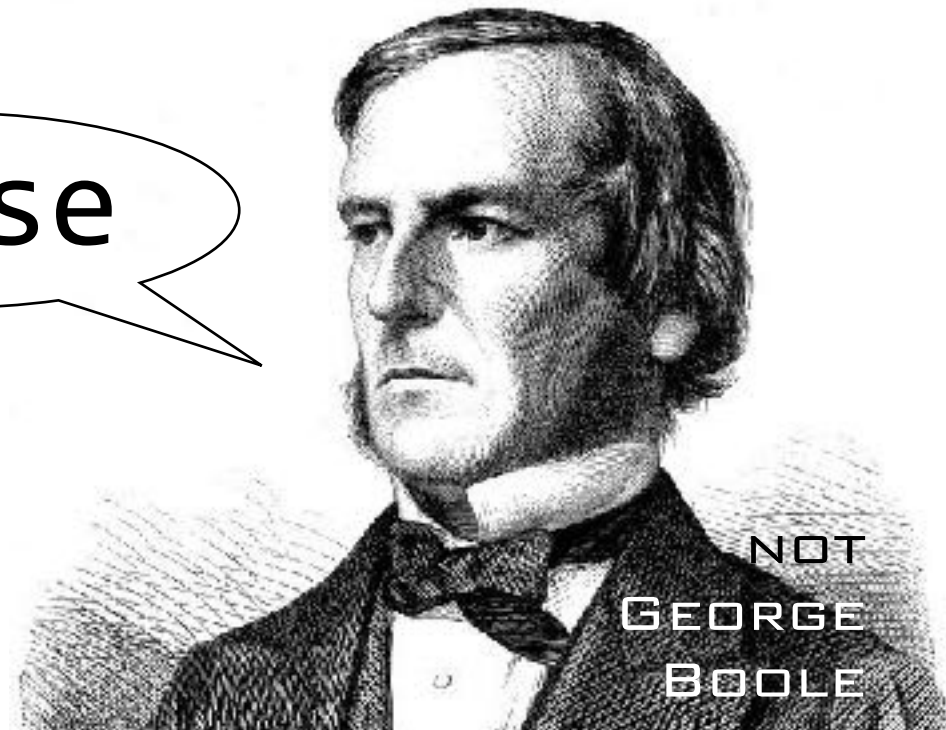✓

```
if (expression) { code }
```
✗

‣ JavaScript doesn't care about white space, so these are equivalent.
‣ **However**, putting block contents on a separate line is best practice for code readability.

BOOLEAN VALUES

COMPARISON OPERATORS

>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to
===	strict equal (use this one)
==	coercive equal (AVOID)
!==	strict not equal (use this one)
!=	coercive not equal (AVOID)

TYPE COERCION

I'm going
to assume you
meant this:

```
let total = "53";
console.log(total == 53);
```

```
let total = 53;
console.log(total == 53);
```

```
> true
```

😭

TYPE COERCION

Those are not the same.

```
let total = "53";
console.log(total === 53);
```

```
let total = "53";
console.log(total === 53);
```

```
> false
```

😎🥳

if STATEMENT

```
let weather = "sunny";

if (weather === "sunny") {
  console.log("Grab your sunglasses");
}
```

if/else STATEMENT

```
let weather = "sunny";

if (weather === "sunny") {
  console.log("Bring your sunglasses");
} else {
  console.log("Grab a jacket");
}
```

else if STATEMENT

```
let weather = "sunny";

if (weather === "sunny") {
  console.log("Bring your sunglasses");
} else if (weather === "rainy") {
  console.log("Take an umbrella");
} else {
  console.log("Grab a jacket");
}
```

TERNARY OPERATOR

(*expression*) ? *trueCode* : *falseCode*;

TERNARY OPERATOR

‣ Can produce one of two values, which can be assigned to a variable in the same statement

```
let name = (expression) ? trueCode : falseCode;
```

BLOCK STATEMENTS

‣ Statements to be executed after a control flow operation are grouped into a block statement

‣ A block statement is placed inside braces

```
{
    console.log("Grab your sunglasses.");
    console.log("Enjoy the beach!");
}
```

LOGICAL OPERATORS

‣ Operators that let you chain conditional expressions

&&	AND	Returns `true` when both left and right values are `true`
||	OR	Returns `true` when at least one of the left or right values is `true`
!	NOT	Takes a single value and returns the opposite Boolean value

TRUTHY AND FALSY VALUES

FALSY VALUES

‣ All of these values become `false` when converted to a Boolean:

```
false

0

""

NaN

null

undefined
```

‣ These are known as **falsy values** because they are equivalent to `false`

TRUTHY VALUES

‣ All values other than `false`, `0`, `""`, `NaN`, `null`, and `undefined` become `true` when converted to a Boolean

‣ All values besides these six are known as **truthy values** because they are equivalent to `true`

‣ '`0`' and '`false`' are both truthy! (Why?)

BEST PRACTICES FOR CONDITIONAL STATEMENTS

‣ Convert to an actual Boolean value

   ‣ Adding ! before a value returns the *inverse* of the value as a Boolean

   ‣ Adding !! before a value gives you the *original* value as a Boolean

```
zipCode     // number: 94104
            // truthy
!zipCode    // Boolean: false
!!zipCode   // Boolean: true
```

BEST PRACTICES FOR CONDITIONAL STATEMENTS

‣ Check a value rather than a comparison

❌ instead of
`if (name === false)`

✅ just use
`if (!name)`

TYPE CASTING

```
// Number() produces a number
let total = Number(sum)
```

```
// String() produces a string
let item = String(title)
```

```
// Boolean() produces a Boolean
let canAttend = Boolean(rsvp)
```

LAB — CONDITIONALS

EXERCISE

TYPE OF EXERCISE

▸ Pair

LOCATION

▸ `starter-code > 1-ages-lab`

TIMING

15 min

1. Write a program that outputs results based on users' age. Use the list of conditions in the `app.js` file.

2. BONUS 1: Rewrite your code to allow a user to enter an age value, rather than hard-coding it into your program. (Hint: Read up on the window.prompt method.)

3. BONUS 2: Rewrite your code to use a switch statement rather than if and else statements.

FUNCTIONS

FUNCTIONS

GROUP STEPS

REUSABLE

STORE STEPS

Allow us to group a series of statements together to perform a specific task

We can use the same function multiple times

Not always executed when a page loads. Provide us with a way to 'store' the steps needed to achieve a task.

CONDITIONALS & FUNCTIONS

DRY =
DON'T
REPEAT
YOURSELF

FUNCTION DECLARATION SYNTAX

```
function name(parameters) {
  // do something
}
```

FUNCTION DECLARATION EXAMPLE

```
function speak() {
  console.log("Hello!");
}
```

FUNCTION EXPRESSION SYNTAX

```
let name = function(parameters) {
  // do something
};
```

FUNCTION EXPRESSION EXAMPLE

```
let speak = function() {
  console.log("Hello!");
};
```

ARROW FUNCTION SYNTAX

```
let name = (parameters) => {
  // do something
};
```

ARROW FUNCTION EXAMPLE

```javascript
let speak = () => {
  console.log("Hello!");
};
```

CALLING A FUNCTION

```
function pickADescriptiveName() {
    // do something
}
```

To run the function, we need to *call* it. We can do so like this:

```
pickADescriptiveName();
```

Function name + parentheses

EXERCISE — WRITING FUNCTIONS

EXERCISE

KEY OBJECTIVE

▸ Practice defining and executing functions

TYPE OF EXERCISE

▸ Individual/paired

LOCATION

▸ `starter-code > 3-functions-exercise` (part 1)

EXECUTION

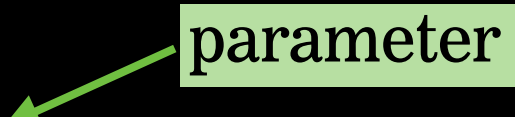*4 min*   1. Follow the instructions under Part 1

PARAMETERS

DOES THIS CODE SCALE?

```javascript
function helloVal() {
  console.log('hello, Val');
}

function helloOtto() {
  console.log('hello, Otto')
}
```

USING A PARAMETER

parameter

argument

```
function sayHello(name) {
  console.log('Hello ' + name);
}


sayHello('Val');
=> 'Hello Val'


sayHello('Otto');
=> 'Hello Otto'
```

USING MULTIPLE PARAMETERS

multiple parameter names
separated by commas
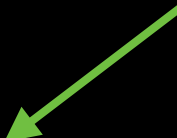
```
function sum(x, y, z) {
  console.log(x + y + z)
}


sum(1, 2, 3);
=> 6
```

USING DEFAULT PARAMETERS

default value to set for parameter if no argument is passed when the function is called

```javascript
function multiply(x, y = 2) {
  console.log(x * y)
}


multiply(5, 6);
=> 30 // result of 5 * 6 (both arguments)
multiply(4);
=> 8 // 4 (argument) * 2 (default value)
```

EXERCISE — READING FUNCTIONS

EXERCISE

KEY OBJECTIVE

▸ Given a function and a set of arguments, predict the output of a function

TYPE OF EXERCISE

▸ Groups of 2 - 3

LOCATION

▸ `starter-code > 3-functions-exercise` (part 2)

EXECUTION

3 min

1. Look at Part 2 A and B. Predict what will happen when each function is called.

EXERCISE — READING FUNCTIONS

EXERCISE

KEY OBJECTIVE

▸ Create and call a function that accepts parameters to solve a problem

TYPE OF EXERCISE

▸ Groups of 2 - 3

LOCATION

▸ `starter-code > 3-functions-exercise` (part 3)

EXECUTION

8 min

1. See if you can write one function that takes some parameters and combines the functionality of the *makeAPizza* and *makeAVeggiePizza* functions.

2. BONUS: Create your own function with parameters. This function could do anything!

THE return STATEMENT

`return` STATEMENT

‣ Ends function's execution

‣ Returns a value — the result of running the function

return STOPS A FUNCTION'S EXECUTION

```javascript
function speak(words) {
  return words;

  // The following statements will not run:
  let x = 1;
  let y = 2;
  console.log(x + y);
}
```

console.log() vs return

console.log() **VS** return

- Write a value at any point in a program to the browser console
- Helpful for developer in debugging
- Not seen by user or used by app

- Sends a value back wherever the current statement was triggered
- Can use a function to get a value and then use that value elsewhere in your app
- Does not appear in the console unless you're executing commands there

return in action

call sum() function, passing 3 and 4 as arguments

```
let z = sum(3,4);
```

with x=3 and y=4, return the result of x + y, which is 7

```
function sum(x,y) {
  return x + y;
}
```

```
z = 7
```

EXERCISE — FUNCTIONS LAB

EXERCISE

KEY OBJECTIVE

▸ Create and call a function that accepts parameters to solve a problem

TYPE OF EXERCISE

▸ Individual or pair

LOCATION

▸ `starter-code > 4-price-calculator`

EXECUTION

15 min

1. Write code to to calculate a customer's total cost in dollars based on product price, tax rate, shipping cost, and the currency they're using for the purchase (dollars or euros).

2. BONUS 1: Convert your function to assume a currency of "dollar" by default.

3. BONUS 2: Convert your code to use arrow functions.

Exit Tickets!

(Class #3)

LEARNING OBJECTIVES – REVIEW

‣ Use Boolean logic to combine and manipulate conditional tests.

‣ Use `if/else` conditionals to control program flow.

‣ Differentiate among true, false, truthy, and falsy.

‣ Describe how parameters and arguments relate to functions

‣ Create and call a function that accepts parameters to solve a problem

‣ Define and call functions defined in terms of other functions

‣ Return a value from a function using the `return` keyword

‣ Define and call functions with argument-dependent return values

NEXT CLASS PREVIEW

Scope & objects

‣ Determine the scope of local and global variables

‣ Describe what hoisting does

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

‣ Create JavaScript objects using object literal notation

Q&A