GENERAL ASSEMBLY

JAVASCRIPT DEVELOPMENT

Sasha Vodnik, Instructor

HELLO!

1. Pull changes from the `svodnik/JS-SF-16-resources` repo to your computer:
   - Open the terminal
   - `cd` to the `Documents/JSD/JS-SF-16-resources` directory
   - Type **`git pull`** and press **return**
2. In your editor, open the following folder: `Documents/JSD/JS-SF-16-resources/03—conditionals-functions`

JAVASCRIPT DEVELOPMENT

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

- Array iterators
- 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

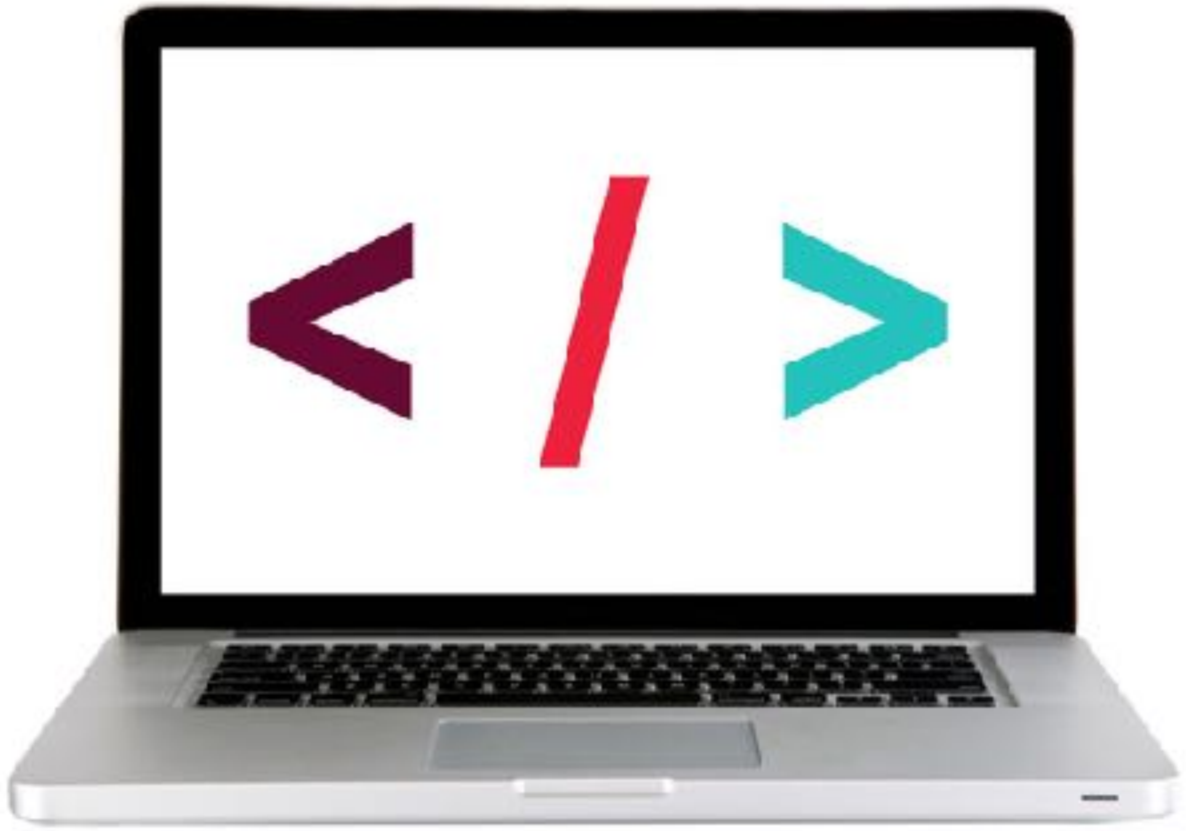
1. Besides coming in for office hours, are there any resources you recommend to practice the concepts we are learning?
2. Is having an adjacent browser window open alongside the code being written and referencing the console when changes are made to the apps.js file the common practice for testing and troubleshooting? Are there other ways people do this?
3. When would you use a array in a real life situation?
4. Is snake_case ever okay in Javascript? Is it a style choice or something more?
5. What would you call the last function covered during the forEach example does it have a specific name or terminology associated with those types of functions?

EXIT TICKET SUGGESTIONS

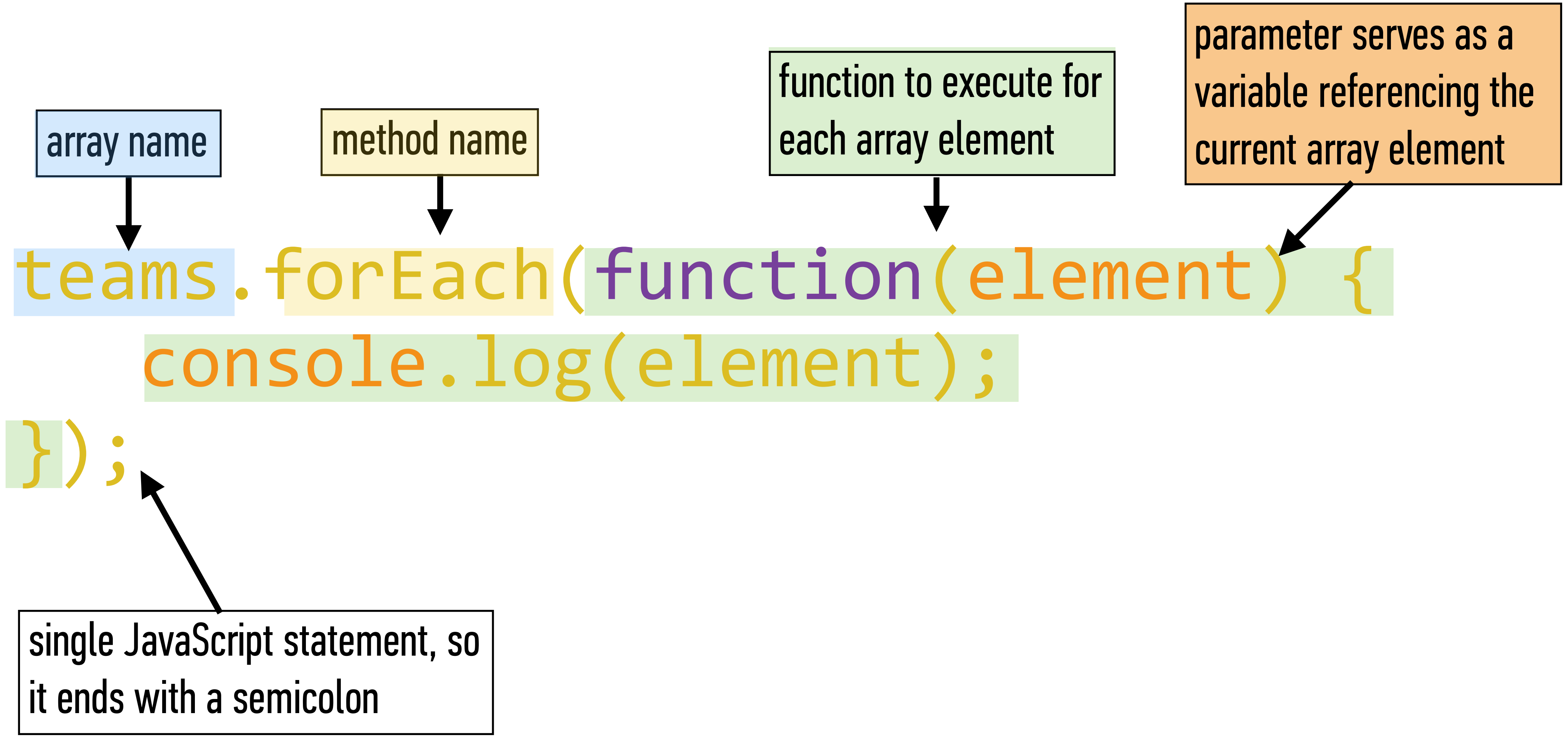
1. Please make sure to wait a few seconds after typing code to explain it. Otherwise i feel I'm copying it and don't get to hear/digest the context behind what we're doing.
2. I'm not sure how much the entire class fully interpreted the last exercise of the night, it seems like folks still had questions but we ran out of time to continue going over it.
3. slow down a little - especially as we have to build up on the knowledge.
4. I liked working the labs. but would like some more input.
5. It was nice when you checked-in or paused for a moment to make sure everyone was on the same page when we were doing exercises.

ARRAY ITERATOR METHODS

LET'S TAKE A CLOSER LOOK



forEach()



```
teams.forEach(function(element) {
    console.log(element);
});
```

array name
method name
function to execute for each array element
parameter serves as a variable referencing the current array element
single JavaScript statement, so it ends with a semicolon

`forEach()` EXAMPLE

```
let teams = ['Bruins', 'Bears', 'Ravens', 'Ducks'];

teams.forEach(function(element) {
    console.log(element);
});
```

LAB — ARRAY LOOPS

EXERCISE

TYPE OF EXERCISE

- Individual / Pair

LOCATION

- `starter-code > 1-arrays-loops-exercise`

TIMING

10 min

1. In the `app.js` file, complete questions 5-6.
2. As in the section you did earlier, your answers should be stored in variables called q1, q2 etc., and the variables logged to the console.
3. Answer these questions using `forEach()` loops, not `for` loops.

ARRAY ITERATOR METHODS

`forEach()`	Executes a provided function once per array element
`every()`	Tests whether all elements in the array pass the test implemented by the provided function
`some()`	Tests whether some element in the array passes the text implemented by the provided function
`filter()`	Creates a new array with all elements that pass the test implemented by the provided function
`map()`	Creates a new array with the results of calling a provided function on every element in this array

LAB — ARRAY LOOPS

EXERCISE

TYPE OF EXERCISE

- Individual / Pair

LOCATION

- `starter-code > 1-arrays-loops-exercise`

TIMING

5 min

1. In the `app.js` file, complete question 7.
2. As in the section you did earlier, your answer should be stored in a variable called q7 and the variable logged to the console.

LAB — PUTTING IT ALL TOGETHER!

EXERCISE

TYPE OF EXERCISE

- Individual / Pair

LOCATION

- `starter-code > 2-arrays-loops-exercise-2`

TIMING

until 9:25

1. Write code for a website shopping cart that calculates the sales tax for each item in a cart array and stores the result in a 2nd array. (Full instructions in the `app.js` file.)
2. Calculate the total with tax of all cart items and store the result in a new variable.
3. BONUS: Update your code to round each item to the nearest cent. (Hint: Read up on `Math.round`)
4. BONUS: Rewrite your code to use the `array.map` method.

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
}
```

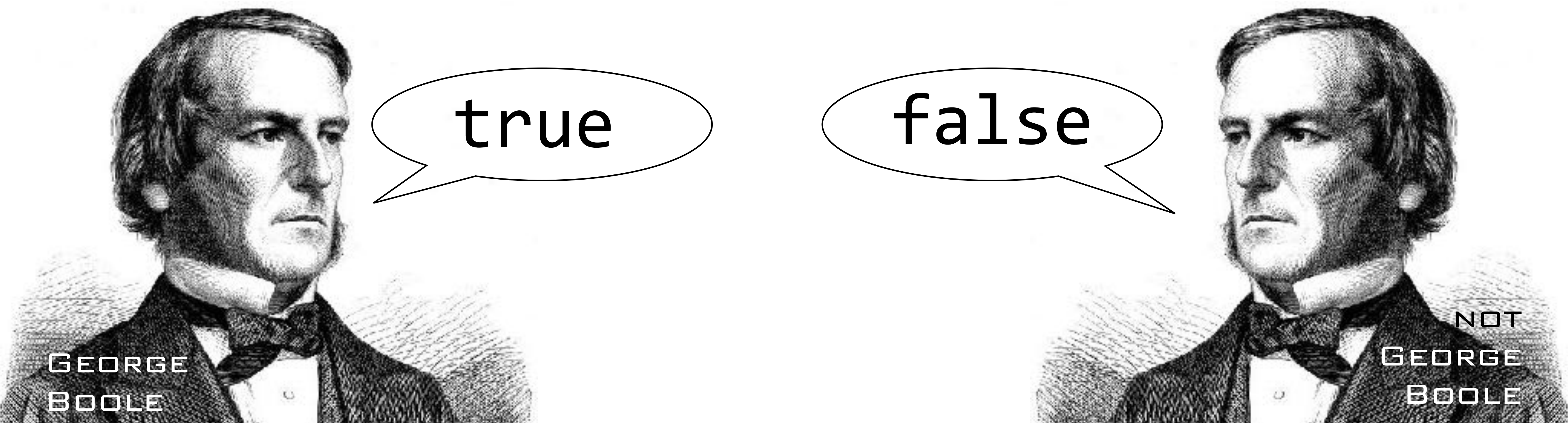
✓

```
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.

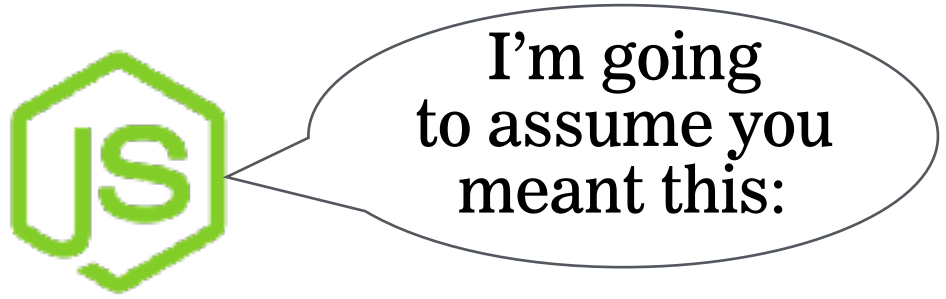
BOOLEAN VALUES



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



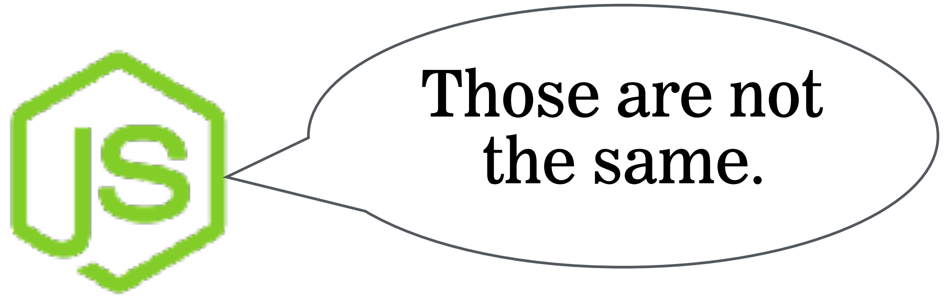
```
let total = “53”;
console.log(total == 53);
```

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

```
> true
```

TYPE COERCION

```
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 (checked === false)
if (checked === true)
```

✓ just use

```
if (!checked)
if (checked)
```

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 > 4-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

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

REUSABLE

We can use the same function multiple times

STORE STEPS

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

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

```
let speak = () => {
  console.log(“Hello!”);
};
```

CALLING A FUNCTION

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

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

$$\underbrace{\texttt{pickADescriptiveName();}}_{\text{Function name + parentheses}}$$

EXERCISE — WRITING FUNCTIONS

EXERCISE

KEY OBJECTIVE

- Practice defining and executing functions

TYPE OF EXERCISE

- Individual/paired

LOCATION

- `starter-code > 6-functions-exercise` (part 1)

EXECUTION

4 min 1. Follow the instructions under Part 1

PARAMETERS

DOES THIS CODE SCALE?

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

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

USING A PARAMETER

parameter

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

argument

```
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

```
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 > 6-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 > 6-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

```
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()`

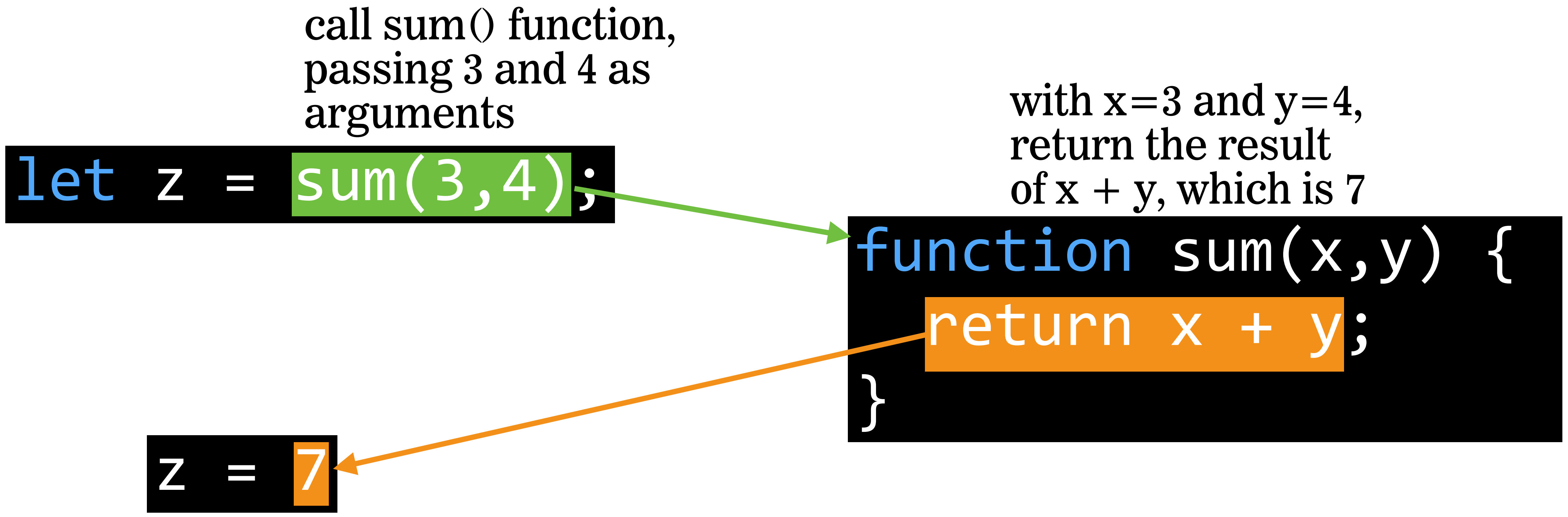
vs

`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 > 7-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