GENERAL ASSEMBLY

JAVASCRIPT DEVELOPMENT

Sasha Vodnik, Instructor

HELLO!

1. Pull changes from the `svodnik/JS-SF-7` repo to your computer
2. Open the `starter-code` folder in your code editor

JAVASCRIPT DEVELOPMENT

THE MODULE PATTERN & THIS

LEARNING OBJECTIVES

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

- Explain the difference between literal and constructed objects.
- Write a constructor for a JavaScript object.
- Implement the module pattern in your code.
- Understand and explain Javascript context.

AGENDA

- The module pattern
- Context and `this`

WEEKLY OVERVIEW

WEEK 8	Project 2 Lab / `this` & Module pattern
WEEK 9	CRUD & Firebase / Deploying your app
WEEK 10	(holiday) / Instructor/student choice

Checkin on Unit #2 Project - Feedr

- The **biggest challenge I have faced or anticipate facing** in completing the Feedr project is _____.
- (If you have already started on the Feedr project:) My **biggest accomplishment** so far in working on Feedr is _____.

How do you know what someone's talking about when they don't specifically name the thing?

EXERCISE — CREATE AN OBJECT LITERAL

TYPE OF EXERCISE

- Individual

TIMING

5 min

1. On your desk, on paper, or in your editor, write code that uses an object literal to create an object named `tortoise`.
2. Give your object a property named `mph` with a value of 1, and a property named `description` with a value of "slow and steady".

EXERCISE — CREATE A MAKECAR FUNCTION

TYPE OF EXERCISE

- Individual/pair

LOCATION

- start files > 01-make-car-function

TIMING

8 min

1. In app.js, Define a function called makeCar() that takes two parameters (model, color), makes a new object literal for a car using those params, and returns that object.
2. Be sure your function returns the fuel property and the drive and refuel methods that you worked with in the previous exercise.

EXERCISE — MAKE A CAR CONSTRUCTOR FUNCTION

TYPE OF EXERCISE

- Individual/pair

LOCATION

- start files > 03-car-constructor-function

TIMING

8 min

1. In app.js, write a constructor function to replace our makeCar function from earlier.
2. Your constructed objects should include the same properties and methods as in the 01-make-car-function exercise.

EXERCISE — LITERAL VS CONSTRUCTED OBJECTS

TYPE OF EXERCISE

- Groups of 2 or 3

TIMING

3 min

1. Spend 30 seconds thinking about the difference between literal and constructed objects.
2. Form a pair or group of 3, then take turns explaining how you understand the difference between the two.
3. Be prepared to share your thoughts with the class.

CLOSURES - REVIEW

- A **closure** is an inner function that has access to the outer (enclosing) function's variables.
- You create a closure by adding a function inside another function.
- A closure is also known as **lexical scope**

CLOSURES — KEY POINTS

- Closures have access to the outer function's variables (including parameters) **even after the outer function returns**.
- Closures store **references** to the outer function's variables.

THE MODULE PATTERN

- Using an IIFE to return an object literal
- The methods of the returned object can access the private properties and methods of the IIFE (closures!), but other code cannot do this
- This means specific parts of the IIFE are not available in the global scope

BENEFITS OF THE MODULE PATTERN

- Lets you keep some functions and variables private
- Lets you avoid polluting the global scope
- Keeps code organized into objects

EXERCISE — CREATE A MODULE

TYPE OF EXERCISE

- Pair

LOCATION

- start files > 05-module-exercise

TIMING

15 min

1. In app.js, complete the module so it exports methods for the behaviors described in the comment at the top of the file.
2. When your code is complete and works properly, the statements at the bottom of the file should all return the expected values in the console.
3. BONUS: Add a "tradeIn" method that lets you change the make of the car and refuels it. Be sure the `getMake` method still works after doing a tradeIn.

CONTEXT AND THIS

- Functions are always executed in relation to some object
- **Context** refers to whatever object is responsible for executing a function
- This object can be referenced using the keyword `this`
- In other words, `this` represents whatever object is in context when a function runs

HOW IS CONTEXT DECIDED?

- At runtime
- Based on how the function is called

CONTEXT RULES

situation	what `this` maps to
function invocation	default: the global object (`window`) strict mode: `undefined`
method invocation	the object that owns the method
constructor function	the newly created object
event handler	the element that the event was fired from

EXERCISE — PREDICT CONTEXT

TYPE OF EXERCISE

- Groups of 2 or 3

LOCATION

- start files > 09-this-exercise

TIMING

5 min

1. In app.js, read through the code without running it.
2. Predict the results of the two console.log statements.
3. Run the code and check the results against your predictions. If the results were different, identify why.

LEARNING OBJECTIVES - REVIEW

- Explain the difference between literal and constructed objects.
- Write a constructor for a JavaScript object.
- Implement the module pattern in your code.
- Understand and explain Javascript context.

NEXT CLASS PREVIEW

In-class lab: Intro to CRUD and Firebase

- Explain what CRUD is. (**Preview**: Create, Read, Update, Delete)
- Explain the HTTP methods associated with CRUD.
- Implement Firebase in an application.
- Build a full-stack app.

Exit Tickets!

Q&A