GENERAL ASSEMBLY

JAVASCRIPT DEVELOPMENT

Sasha Vodnik, Instructor

HELLO!

1. Pull changes from the `svodnik/JS-SF-10-resources` repo to your computer
2. Open the `14-closures-module-pattern > starter-code` folder in your code editor
3. **To submit your Feedr project**,
   - Remove your API keys and push your code to GitHub
   - DM the URL of your GitHub fork of the Feedr repo to Sasha & Aurielle

CLOSURES & THE MODULE PATTERN

LEARNING OBJECTIVES

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

- Describe the difference between functional programming and object oriented programming.
- Understand and explain closures.
- Instantly invoke functions.
- Implement the module pattern in your code.

AGENDA

- Objects
- Closures
- IIFEs
- Module pattern

WEEKLY OVERVIEW

WEEK 9	Closures & the module pattern / CRUD & Firebase
WEEK 10	Deploying your app / Final Project Lab
WEEK 11	Student-Instructor Choice / Final Project Presentations!

HOMEWORK REVIEW

ACTIVITY

KEY OBJECTIVE

- Review Feedr project and show off your work

TYPE OF EXERCISE

- Groups of 3-4

TIMING

10 min

1. Open Feedr sites on laptops and display them proudly!
2. Give feedback to your peers: "I like" and "I wish/wonder”
3. Share a challenge you ran into in your project and discuss how other group members may have worked with it.
4. Did you incorporate template literals in your project? Show your group how you did it!

ACTIVITY

EXERCISE

KEY OBJECTIVE

- Check in on final projects

TYPE OF EXERCISE

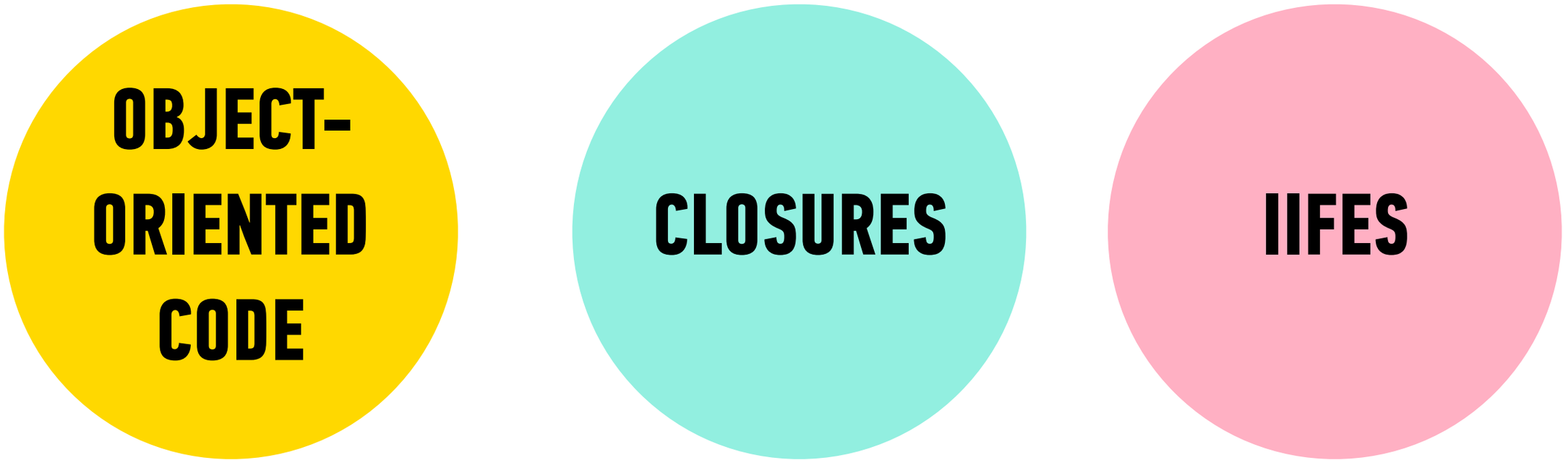
- Groups of 3-4

TIMING

2 min

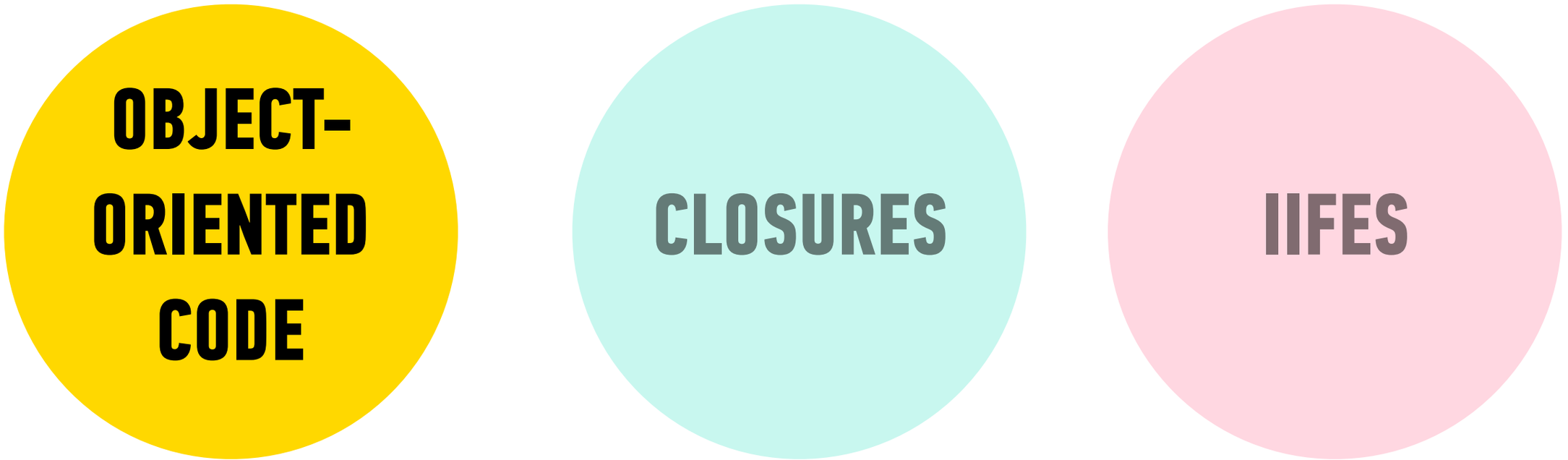
1. Describe your idea for your final project — what will your app do?
2. Share your next step. If you're not sure, share where you are right now and brainstorm with your group what next steps might look like.

THE MODULE PATTERN

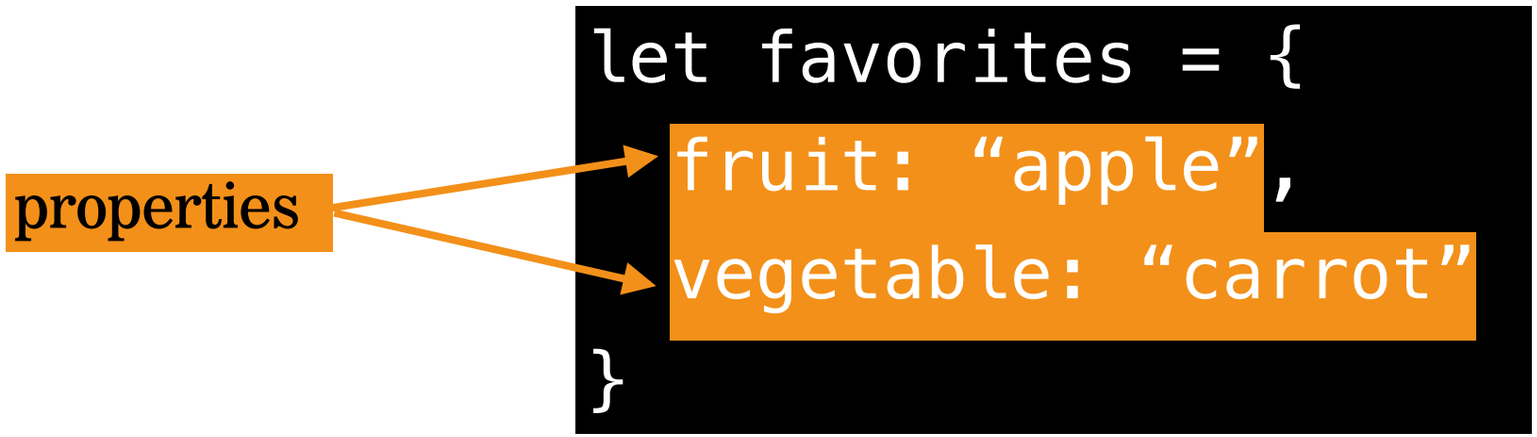


FUNCTIONS & OBJECTS

THE MODULE PATTERN



AN OBJECT IS A COLLECTION OF PROPERTIES



PROPERTY = KEY & VALUE

- A **property** is an association between a key and a value
  - **key**: name (often descriptive) used to reference the data
  - **value**: the data stored in that property

EXERCISE — CREATE AN OBJECT LITERAL

TYPE OF EXERCISE

- Individual

TIMING

2 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".

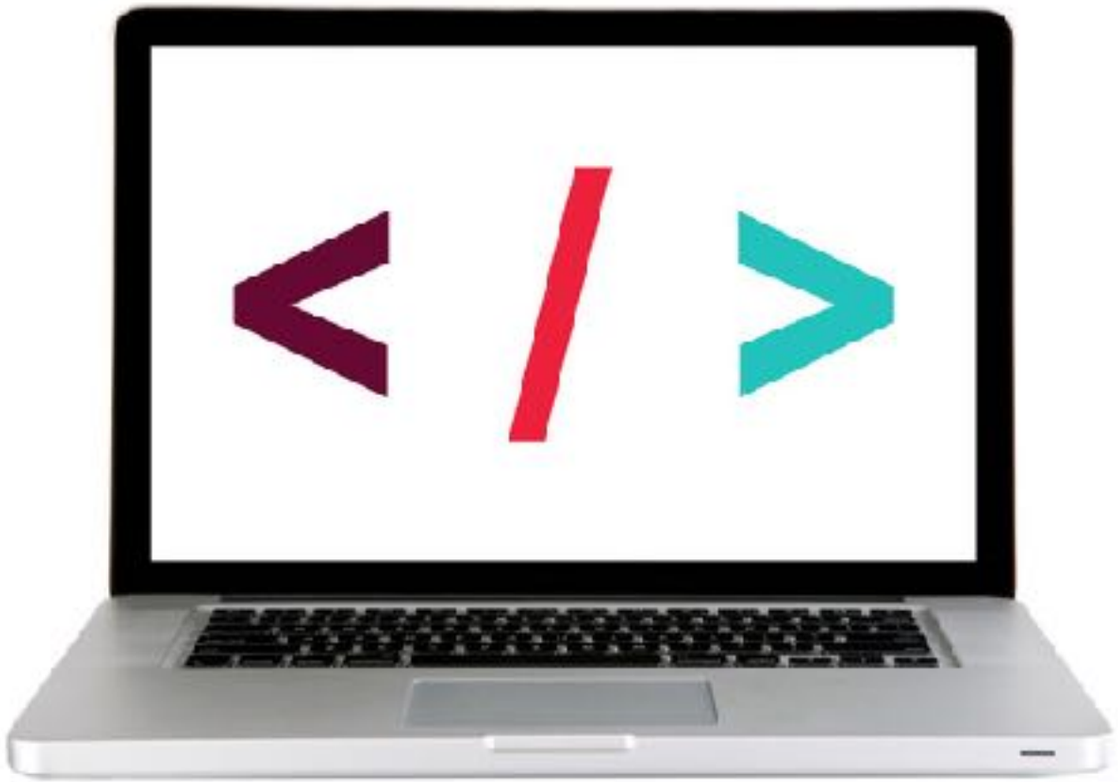
functional code

```
const taxRate = 0.0875;
let items = [];

function addToCart() {
  // do something
}

function calcTax() {
  // do something
}

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

object oriented code

```
let cart = {
  taxRate: 0.0875,
  items: [],
  addToCart: function() {
    // do something
  },
  calcTax: function() {
    // do something
  },
  calcTotal: function() {
    // do something
  }
};
```

LET'S TAKE A CLOSER LOOK



EXERCISE — CREATE A MAKECAR FUNCTION

TYPE OF EXERCISE

- Individual/pair

LOCATION

- start files > 1-make-car-function

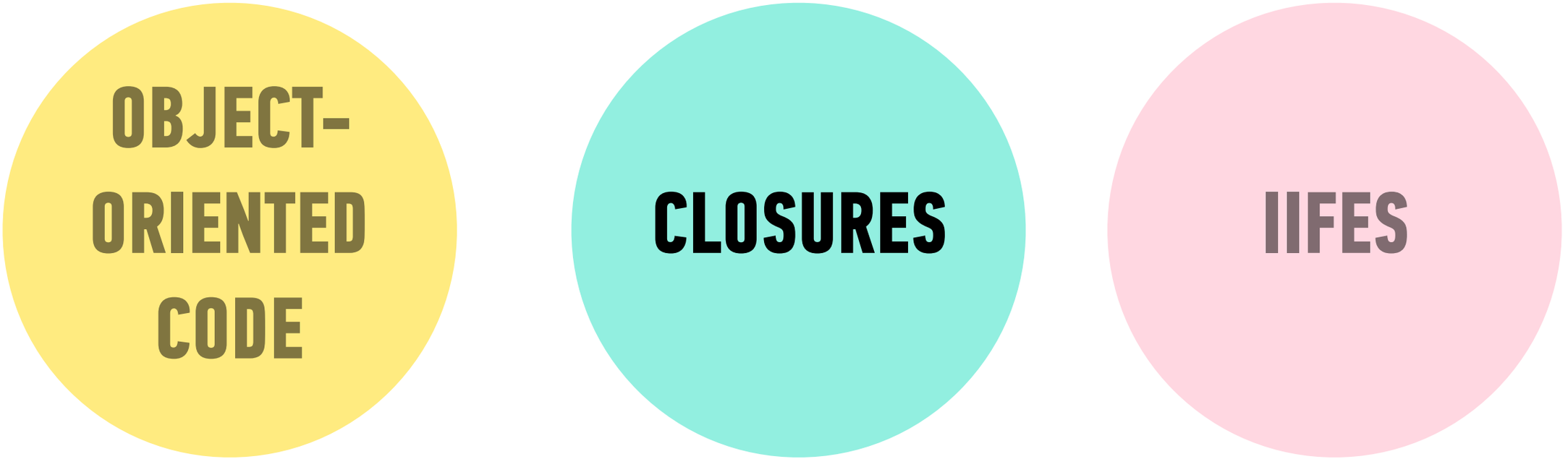
TIMING

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.

CLOSURES

THE MODULE PATTERN



SCOPE

- Describes the set of variables you have access to

GLOBAL SCOPE

- A variable declared outside of a function is accessible everywhere, even within functions. Such a variable is said to have **global scope**.

a variable declared outside of the function is in the global scope

```
let temp = 75;
function predict() {
  console.log(temp); // 75
}
console.log(temp); // 75
```

LOCAL SCOPE

- A variable declared within a function is not accessible outside of that function. Such a variable is said to have **local scope**.

```
let temp = 75;
function predict() {
  let forecast = ‘Sun’;
  console.log(temp + " and " + forecast); // 75 and Sun
}
console.log(temp + " and " + forecast);
// ‘forecast’ is undefined
```

a variable declared within a function is in the local scope of that function

a local variable is not accessible outside of its local scope

BLOCK SCOPE

- A variable created with `let` or `const` creates local scope within any block, including blocks that are part of loops and conditionals.
- This is known as **block scope**.

```
let temp = 75;
if (temp > 70) {
  let forecast = 'It's gonna be warm!';
  console.log(temp + "! " + forecast); // 75! It's gonna be warm!
}
console.log(temp + "! " + forecast); // 'forecast' is undefined
```

`let` creates a local variable within any block, such as an `if` statement

a variable with block scope is not accessible outside of its block

BUILDING BLOCKS OF CLOSURES

1. **nested functions**
2. **scope**

   inner function has access to outer function's variables

3. **`return` statements**

   outer function returns reference to inner function

CLOSURES

- A **closure** is an inner function that has access to the outer (enclosing) function's variables.

```
function getTemp() {
  let temp = 75;
  let tempAccess = function() {
    console.log(temp);
  }
  return tempAccess;
}
```

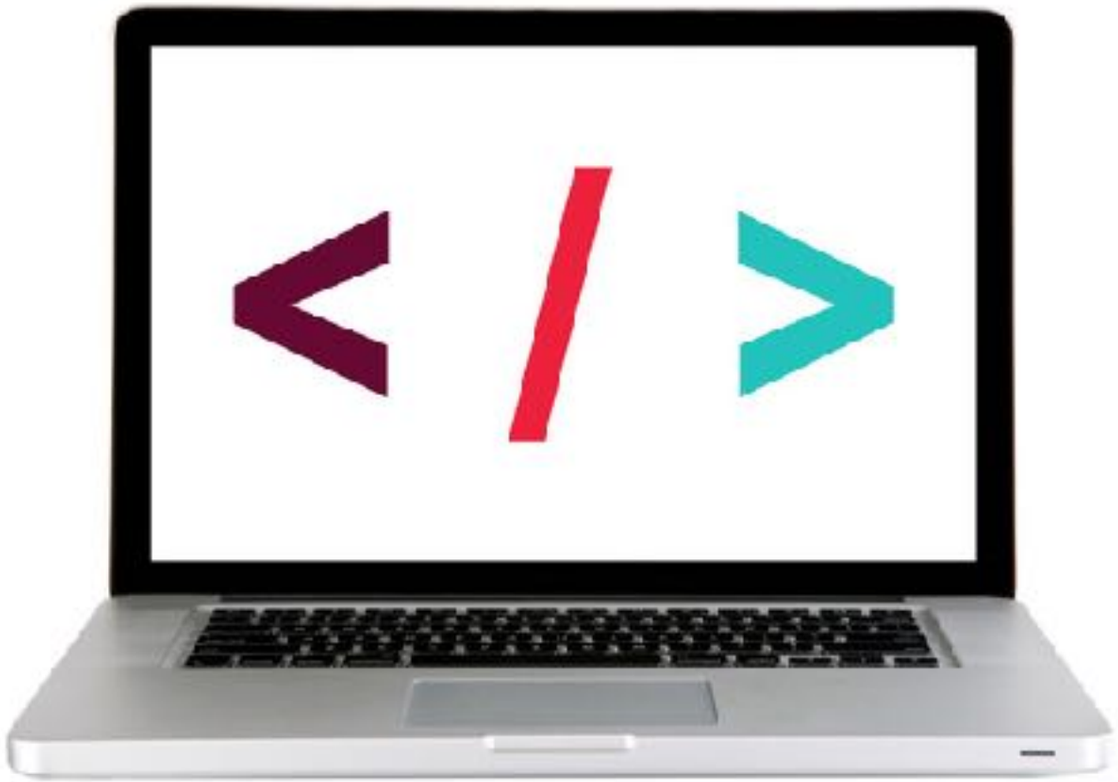
the `tempAccess()` function is a closure

outer function `getTemp()` returns a reference to the inner function `tempAccess()`

CLOSURES

- A **closure** is an inner function that has access to the outer (enclosing) function's variables.
- You create a closure by nesting a function inside another function.

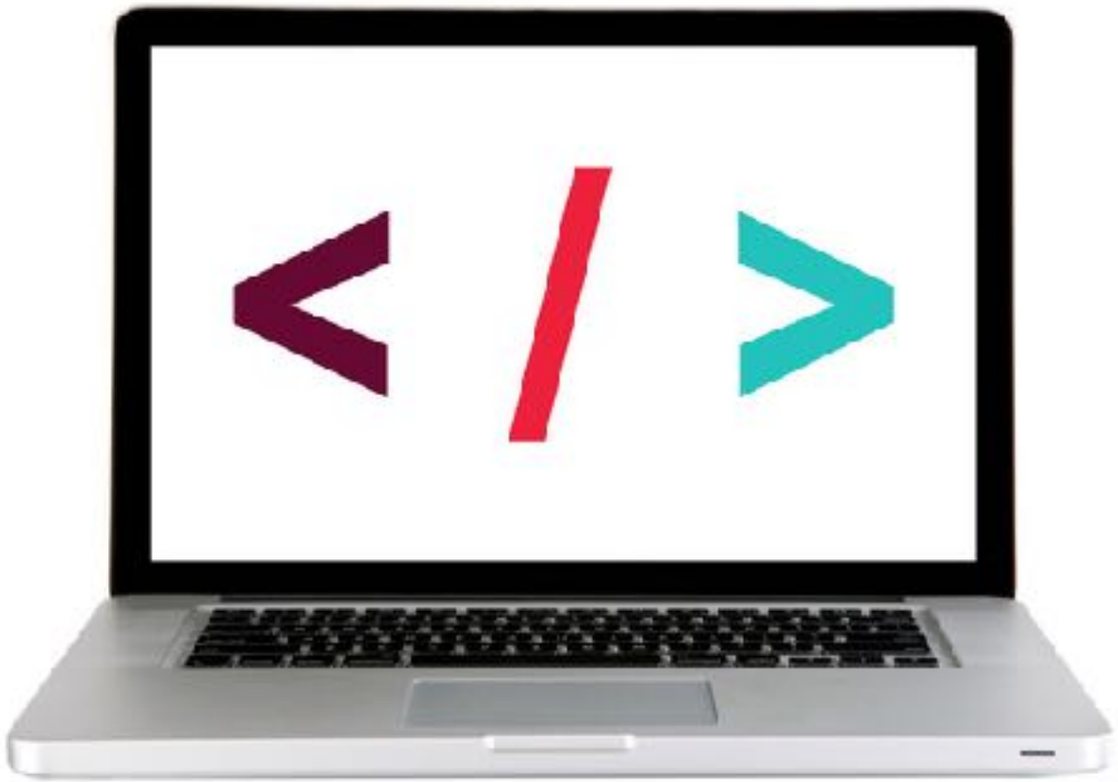
LET'S TAKE A CLOSER LOOK



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, not the actual values.

LET'S TAKE A CLOSER LOOK



WHAT ARE CLOSURES USED FOR?

- Turning an outer variable into a private variable
- Namespacing private functions

LAB — CLOSURES

KEY OBJECTIVE

- Understand and explain closures

TYPE OF EXERCISE

- Pairs

LOCATION

- starter-code > 3-closures-lab

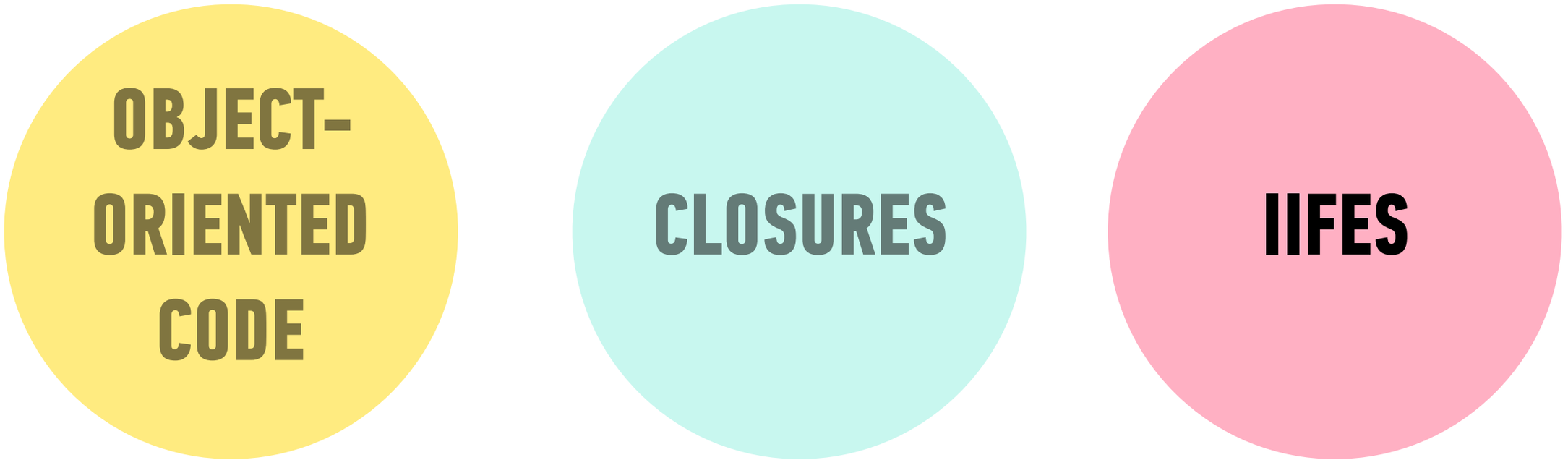
EXECUTION

15 min

1. Follow the instructions in app.js to build and test code that uses a closure.

Immediately-invoked function expressions

THE MODULE PATTERN



Immediately-invoked function expression (IIFE)

- A function expression that is executed as soon as it is declared
- Pronounced “iffy”

IIFE based on a function expression

- Make a function expression into an IIFE by adding () to the end (before the semicolon)

```
let countDown = function() {
  let counter;
  for(counter = 3; counter > 0; counter--) {
    console.log(counter);
  }
}();
```

IIFE based on a function expression

- Make a function expression into an IIFE by adding () to the end (before the semicolon)

```
let countDown = function() {
  let counter;
  for(counter = 3; counter > 0; counter--) {
    console.log(counter);
  }
}();
```

IIFE based on a function declaration

- Make a function declaration into an IIFE by adding
  `(` at the start and
  `)();` to the end

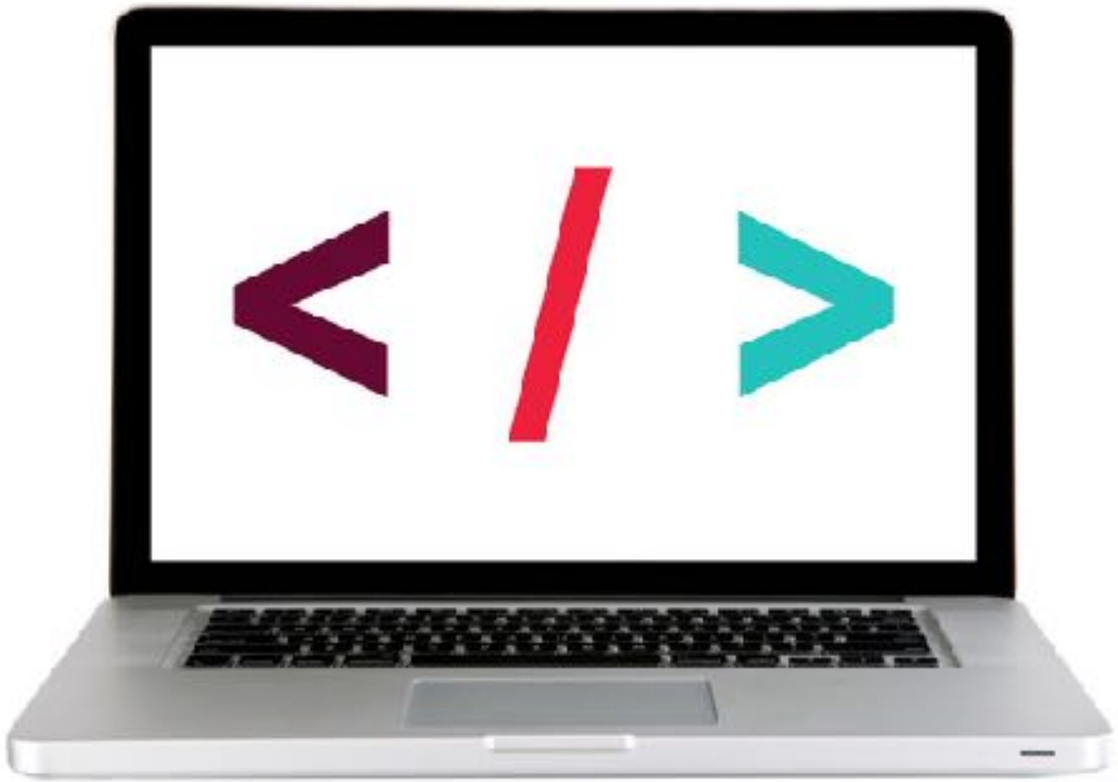
```
(function countDown() {
  let counter;
  for(counter = 3; counter > 0; counter--) {
    console.log(counter);
  }
})();
```

IIFE based on a function declaration

- Make a function declaration into an IIFE by adding
  `(` at the start and
  `)();` to the end

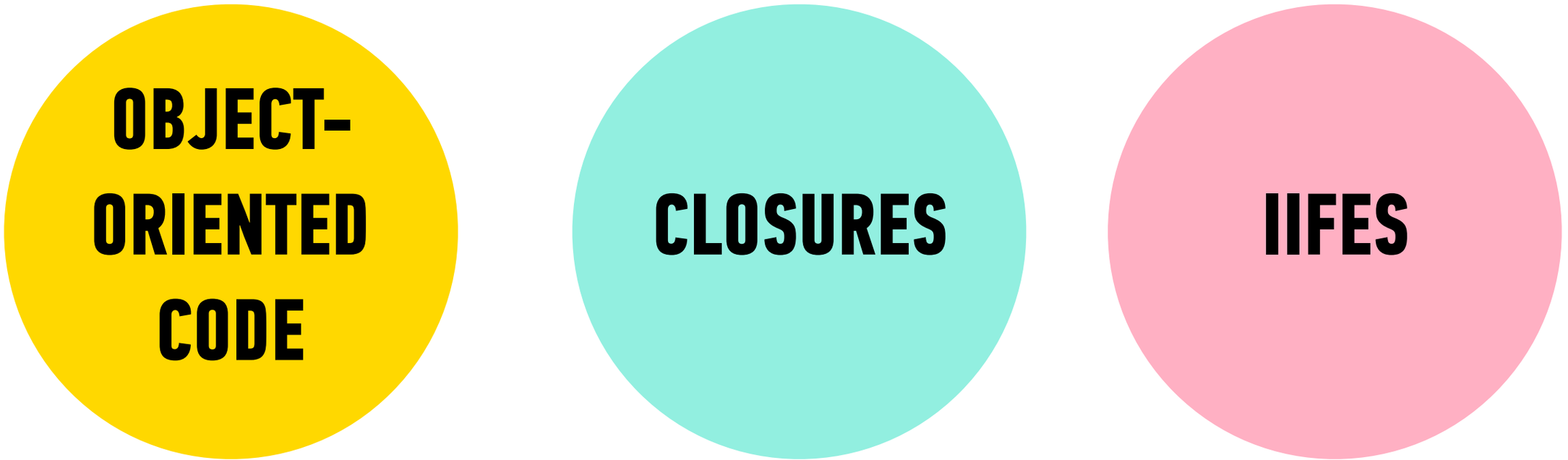
```
(function countDown() {
  let counter;
  for(counter = 3; counter > 0; counter--) {
    console.log(counter);
  }
})();
```

LET'S TAKE A CLOSER LOOK



THE MODULE PATTERN

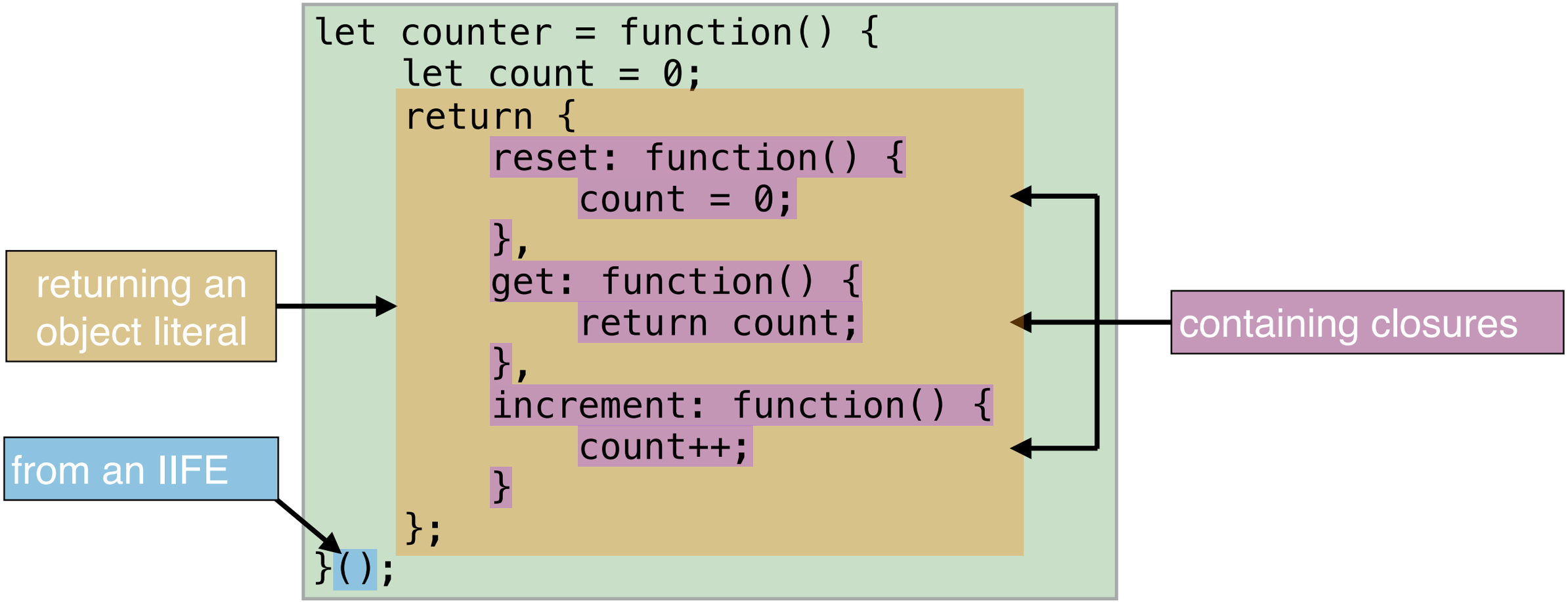
PUTTING IT ALL TOGETHER!



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

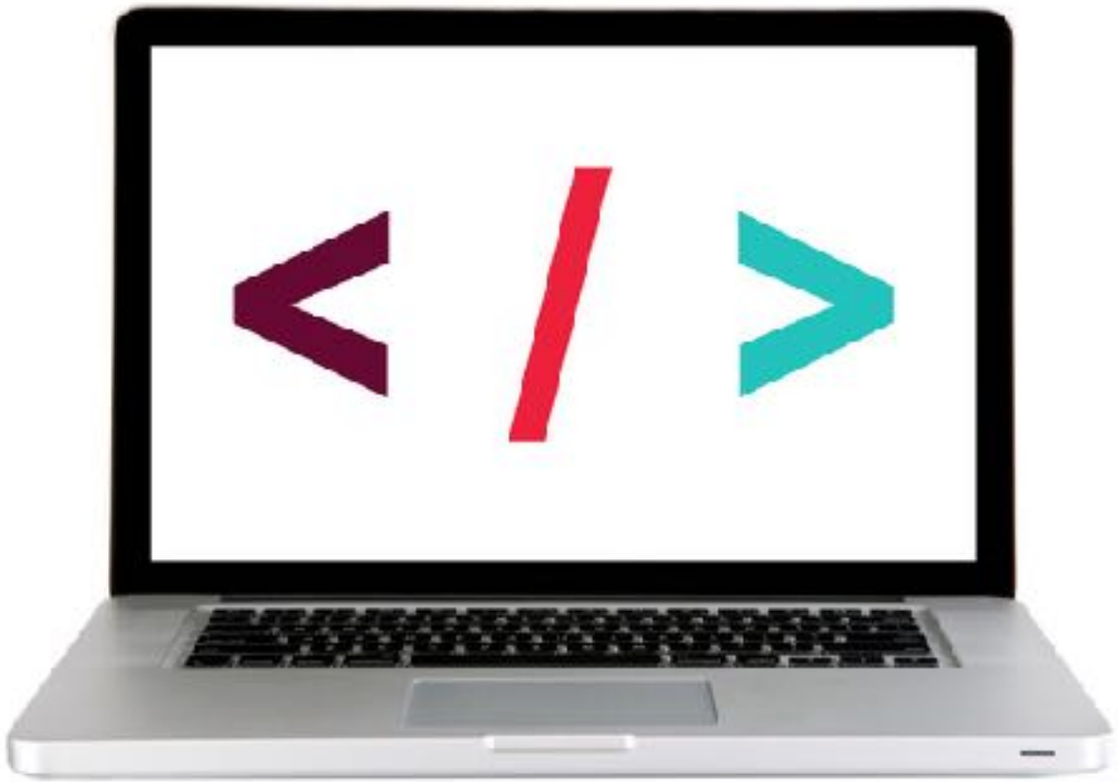
BUILDING A MODULE



BENEFITS OF THE MODULE PATTERN

- Keeps some functions and variables private
- Avoids polluting the global scope
- Organizes code into objects

LET'S TAKE A CLOSER LOOK



EXERCISE — CREATE A MODULE

TYPE OF EXERCISE

- Pair

LOCATION

- start files > 6-modules-exercise

TIMING

until 9:20

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.

Exit Tickets!

(Class #14)

LEARNING OBJECTIVES - REVIEW

- Describe the difference between functional programming and object oriented programming.
- Understand and explain closures.
- Instantly invoke functions.
- Implement the module pattern in your code.

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.

Q&A