GENERAL ASSEMBLY

JAVASCRIPT DEVELOPMENT

Sasha Vodnik, Instructor

HELLO!

1. Pull changes from the `JS-SF-14-resources` repo to your computer
2. Open the `13-prototypal-inheritance` folder in your code editor

PROTOTYPAL INHERITANCE

LEARNING OBJECTIVES

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

- Distinguish between classical and prototypal inheritance
- Explain the difference between literal and constructed objects.
- Write a constructor for a JavaScript object.
- Explain prototypal inheritance and its purpose.
- Create and extend prototypes.

AGENDA

- Objects and constructors
- Prototypal inheritance
- The `class` keyword

WEEKLY OVERVIEW

WEEK 8	Prototypal inheritance / Closures & `this`
WEEK 9	CRUD & Firebase / Deploying your app
WEEK 10	React / Final project lab

HOMEWORK REVIEW

HOMEWORK — GROUP DISCUSSION

EXERCISE

TYPE OF EXERCISE

- Groups of 2-3

TIMING

5 min

1. Share your solutions for the homework.
2. Share 1 thing you found challenging. If you worked it out, share how; if not, brainstorm with your group how you might approach it.

ACTIVITY

EXERCISE

KEY OBJECTIVE

- Check in on Feedr

TYPE OF EXERCISE

- Pairs

TIMING

6 min

1. Share your biggest accomplishment so far in working on Feedr
2. Share a challenge you've encountered. If you overcame it, share how; if not, brainstorm with your partner on an approach.
3. Share your next step.

EXIT TICKET QUESTIONS

- using template literals for formatting emails
- More about how to use APIs... I wonder how it works on the server side / what kind of code does it take to provide this sort of API.
- How to use tag templates? (tag function)

OBJECTS AND INHERITANCE

CLASS VS PROTOTYPE

CLASS-BASED LANGUAGE

class

- manufactures new objects
- defines behavior of manufactured objects

JAVASCRIPT

constructor

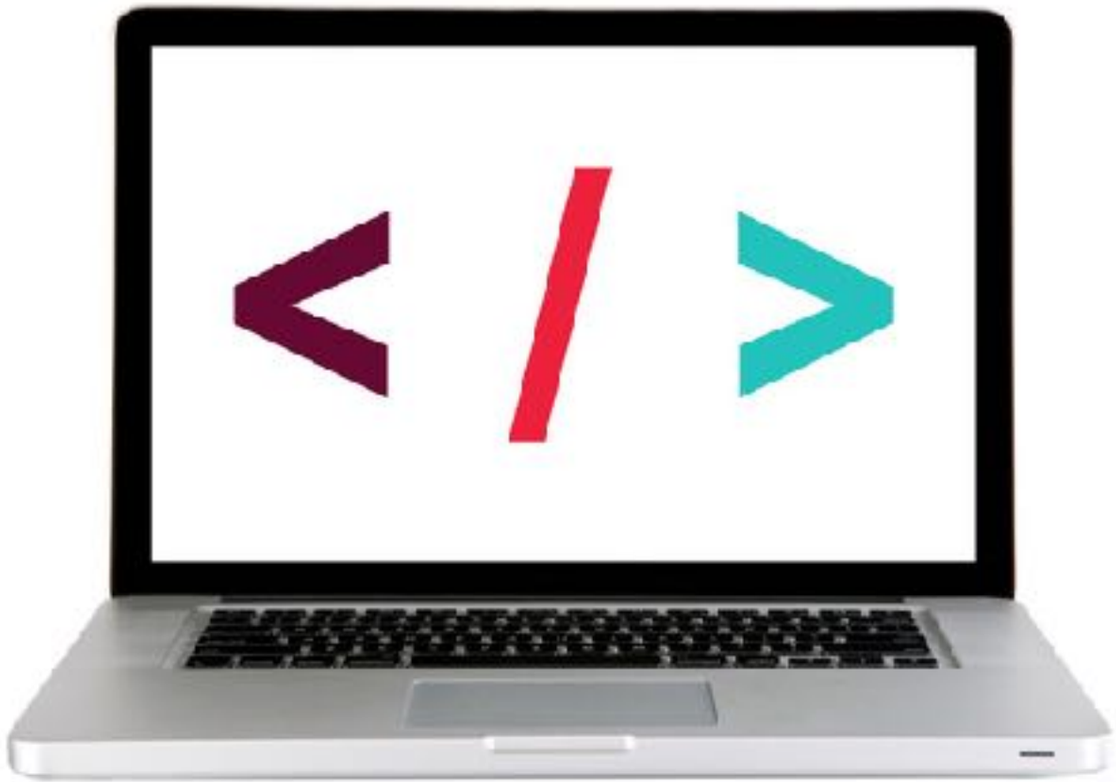
- manufactures new objects

prototype

- defines behavior of manufactured objects

CONSTRUCTORS

LET'S TAKE A CLOSER LOOK



EXERCISE — CREATE A MAKECAR FUNCTION

TYPE OF EXERCISE

- Individual/pair

LOCATION

- `start files > 1-make-car-exercise`

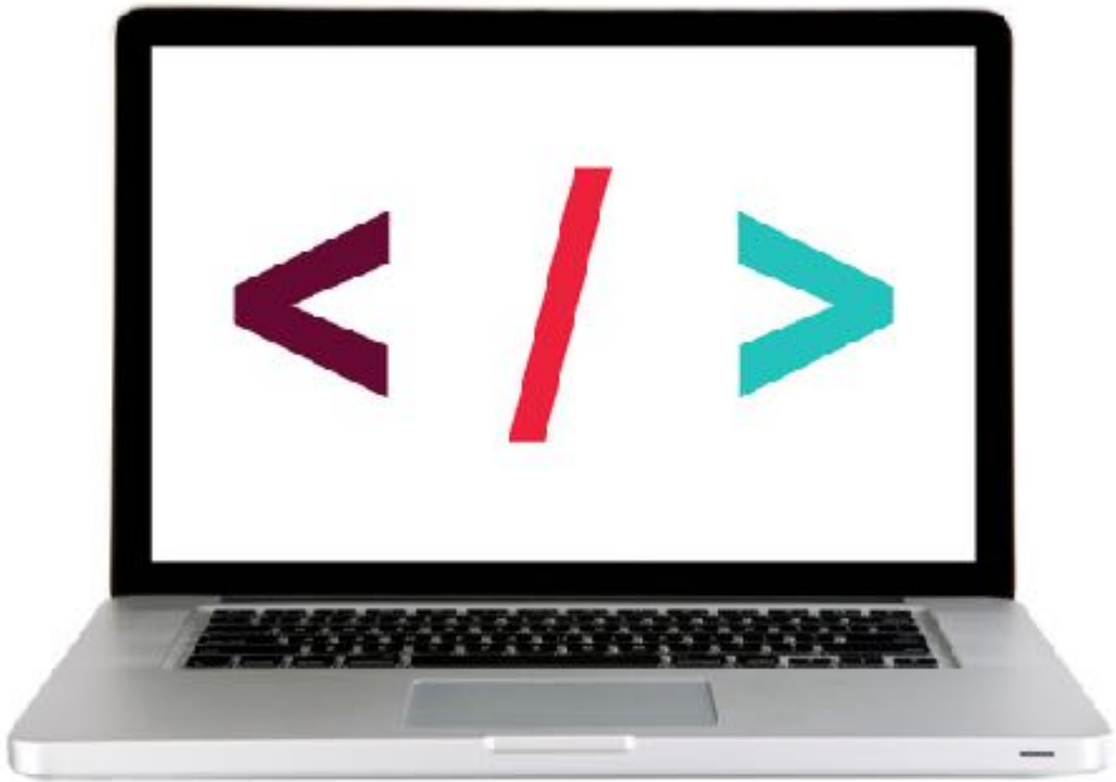
TIMING

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.

CONSTRUCTOR FUNCTIONS

LET'S TAKE A CLOSER LOOK



EXERCISE — MAKE A CAR CONSTRUCTOR FUNCTION

TYPE OF EXERCISE

- Individual/pair

LOCATION

- `start files > 3-constructor-exercise`

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 as in the 1-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.

PROTOTYPES

PROTOTYPES

- Every object in JS has a `prototype` property, which is a reference to another object
- The object that the `prototype` property points to is generally an instance of the constructor object
- Any properties/methods defined on an object's prototype are available on the object itself, without defining those properties/methods a second time
- The relationship between objects that have a prototypal relationship with each other is known as the **prototype chain**

Using the `prototype` property

```
function Dog(name, breed) {
  this.name = name;
  this.breed = breed;
}
Dog.prototype.species = “Canis Canis”;
Dog.prototype.bark = function() {
  return “Woof! I’m “ + this.name;
}
```

Dog.prototype

```
{
  species: “Canis Canis”,
  bark: function() {
    return “Woof! I’m “ +
      this.name;
  }
}
```

Using the `prototype` property

```
var spot = new Dog("Spot", "Beagle");
```

individual properties created by the constructor function

inherited from `Dog.prototype` object

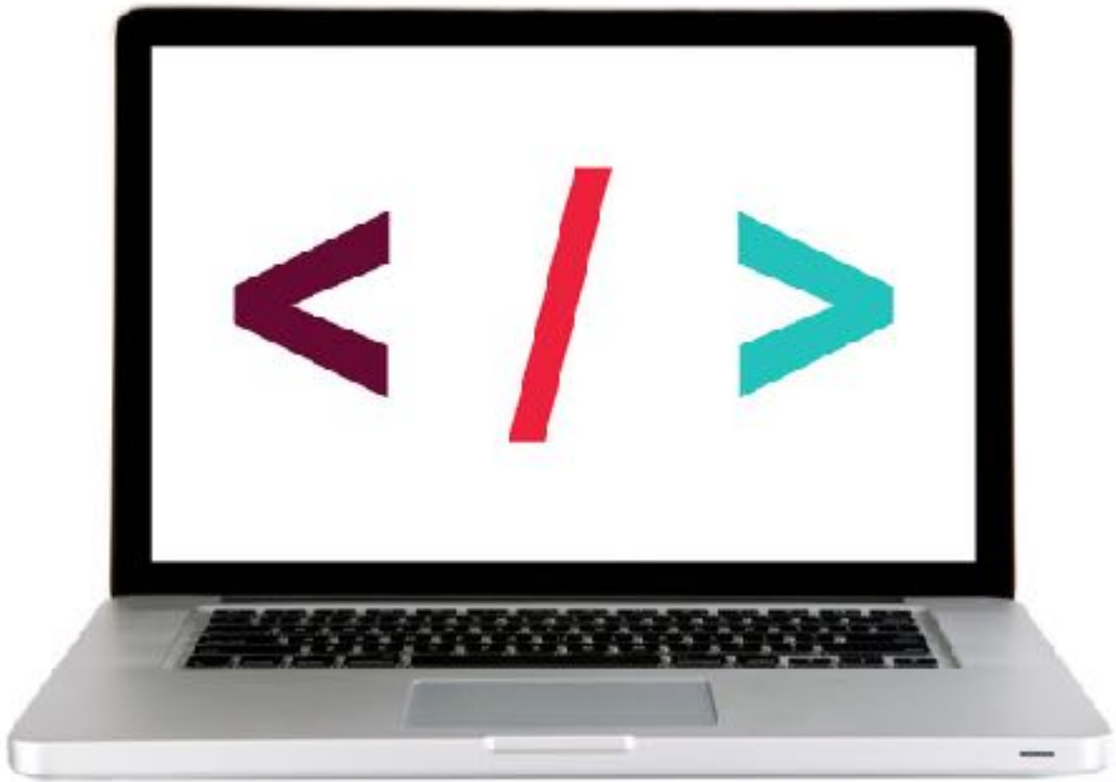
`spot` object (constructed)

```
{
  name: “Spot”,
  breed: “Beagle”,
  species: “Canis Canis”,
  bark: function() {
    return “Woof! I’m “ + this.name;
  }
}
```

PROTOTYPE TERMINOLOGY

- prototype: a model used to create instances
- `prototype` property: a reference to another object that is generally an instance of the constructor object
- `__proto__` (or “dunder proto”): a property used by web browsers that indicates an object’s parent in the prototype chain

LET'S TAKE A CLOSER LOOK



EXERCISE — MAKE A CAR CONSTRUCTOR FUNCTION

TYPE OF EXERCISE

- Individual/pair

LOCATION

- `start files > 6-prototypes-exercise`

TIMING

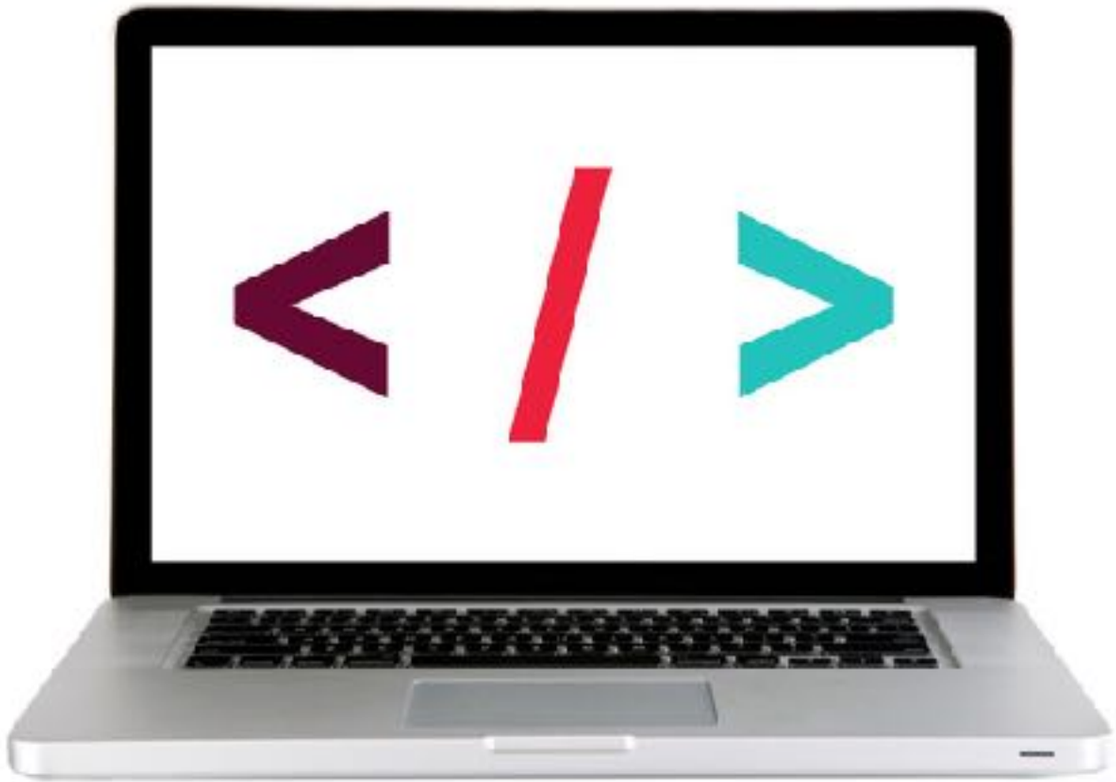
8 min

1. In app.js, create a Monkey constructor that meets the specs described.
2. Create 3 objects using your Monkey constructor and verify that all properties and methods of each have the expected values.

`Object.create()`

- Creates a new object
- Sets prototype of new object to be existing object
- Some differences under the hood, but essentially equivalent to using the `new` keyword
- Example:
  - `var me = Object.create(Person)`
  - equivalent to `var me = newPerson();`

LET'S TAKE A CLOSER LOOK



LAB - BUILD A PROTOTYPE CHAIN

EXERCISE

TYPE OF EXERCISE

- Individual/pair

LOCATION

- `start files > 9-prototypes-lab`

TIMING

12 min

1. Create an Item constructor using the specs in the start file.
2. Create Clothing and Household constructors and use Item as the prototype for each.
3. Test your work in the browser.
4. If you finish early, work on the bonus items described in app.js.

Exit Tickets!

(Class #13)

LEARNING OBJECTIVES - REVIEW

- Distinguish between classical and prototypal inheritance
- Explain the difference between literal and constructed objects.
- Write a constructor for a JavaScript object.
- Explain prototypal inheritance and its purpose.
- Create and extend prototypes.

NEXT CLASS PREVIEW

Closures & `this`

- Understand and explain Javascript context.
- Understand and explain closures.
- Instantly invoke functions.
- Predict what the `this` keyword refers to in different situations.

Q&A