EECS 368
Programming Language Paradigms

David O. Johnson

Fall 2022

Reminders

- Assignment 2 due: 11:59 PM, Monday, September 19
- Assignment 3 due: 11:59 PM, Monday, October 3

Any Questions?

In-Class Problem Solution

- 6-(9-7) In-Class Problem Solution.pptx

Any Questions?

JavaScript Objects

Recall:

- A JavaScript object is an arbitrary collections of properties.
- One way to create an object is by using braces as an expression.
- Inside the braces, there is a list of properties separated by commas.
- Each property has a name followed by a colon and a value.

```
let day1 = {
  squirrel: false,
  events: ["work", "touched tree", "pizza", "running"]
};
console.log(day1.squirrel);
// → false
console.log(day1.events);
// → Array(4) [ "work", "touched tree", "pizza", "running" ]
```

Methods

Recall:

- Properties that contain functions are called methods.

- Here is a simple method:

```
let rabbit = {};
rabbit.speak = function(line) {
  console.log(`The rabbit says '${line}'`); //Note the Template Literal here.
};

rabbit.speak("I'm alive.");
// → The rabbit says 'I'm alive.'
```

Recall:
When you write something inside ${} in a Template Literal, its result will be computed, converted to a string, and included at that position.

this Keyword

- Usually a method needs to do something with the object it was called on.
- When a function is called as a method the binding called this in its body automatically points at the object that it was called on.

```
function speak(line) {
  console.log(`The ${this.type} rabbit says '${line}'`);
}
let whiteRabbit = {type: "white", speak};
let hungryRabbit = {type: "hungry", speak};

whiteRabbit.speak("Oh my ears and whiskers, " +
                  "how late it's getting!");
// → The white rabbit says 'Oh my ears and whiskers, how
//   late it's getting!'
hungryRabbit.speak("I could use a carrot right now.");
// → The hungry rabbit says 'I could use a carrot right now.'
```

- Alternately, you can use the call method:
  - speak.call(hungryRabbit, "I could use a carrot right now.");
  - // → The hungry rabbit says 'I could use a carrot right now.'

this Keyword

- Arrow functions are different.
- They do not bind their own this but can see the this binding of the scope around them.
- Thus, you can do something like the following code, which references this from inside a local function:

```
function normalize() {
  console.log(this.coords.map(n => n / this.length));
}
normalize.call({coords: [0, 2, 3], length: 5});
// → [0, 0.4, 0.6]
```

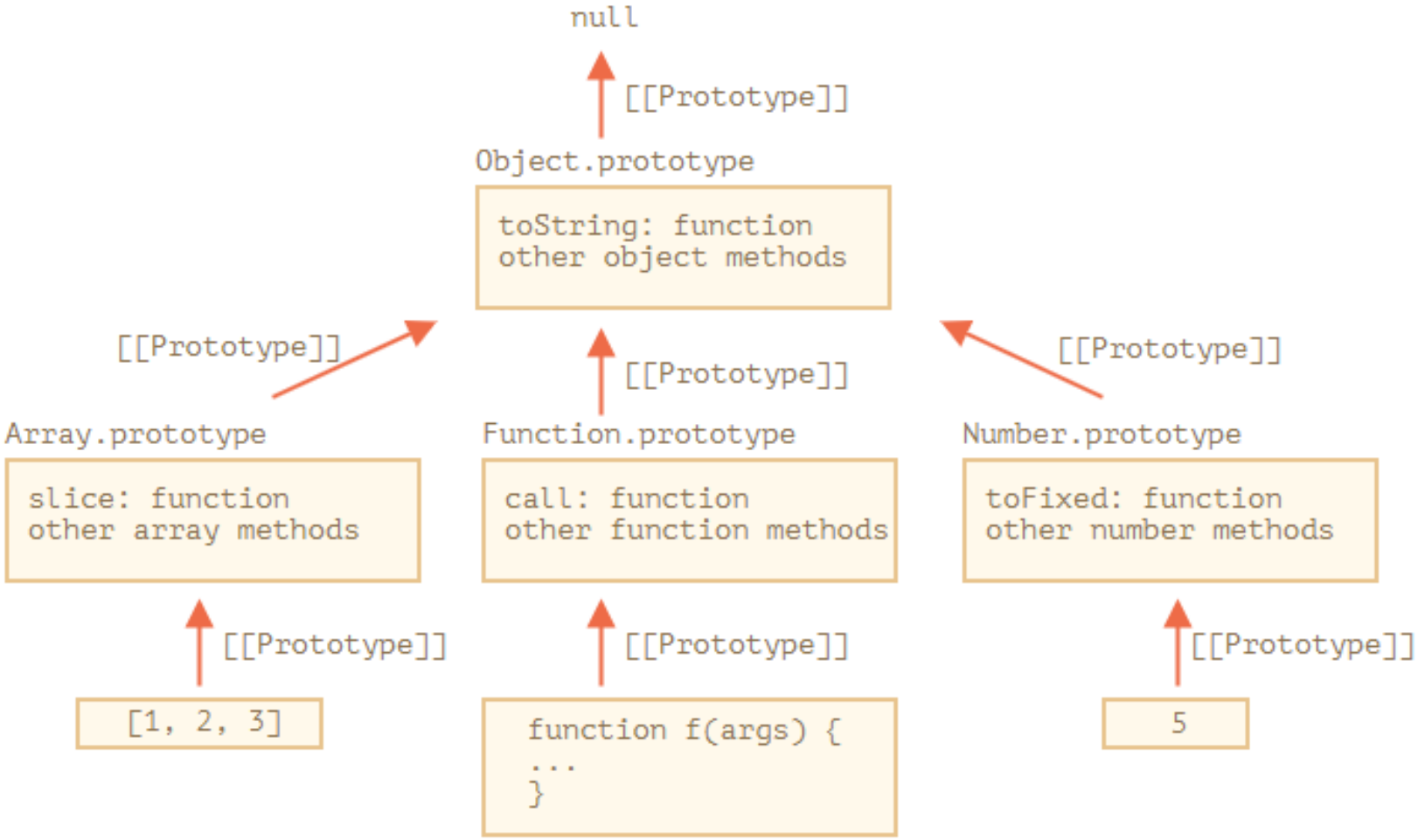
- If I had written the argument to map using the function keyword, the code wouldn't work.

Any Questions?

Prototypes

- In addition to their set of properties, most objects also have a prototype.
- A prototype is another object that is used as a fallback source of properties.
- When an object gets a request for a property that it does not have, its prototype will be searched for the property, then the prototype's prototype, and so on.
- This is inheritance.
- The prototype relations of JavaScript objects form a tree-shaped structure, and at the root of this structure sits Object.prototype.
- It provides a few methods that show up in all objects, such as toString, which converts an object to a string representation.
- Function.prototype, Array.prototype, and Number.prototype are built-in prototypes with their own set of default properties and methods.

Prototype Inheritance



Prototypes

- You can create your own prototypes with Object.create:

```
let protoRabbit = {
  speak(line) {
    console.log(`The ${this.type} rabbit says '${line}'`);
  }
};

let killerRabbit = Object.create(protoRabbit);
killerRabbit.type = "killer";
killerRabbit.speak("SKREEEE!");
// → The killer rabbit says 'SKREEEE!'
```

- A property like speak(line) in an object expression is a shorthand way of defining a method.
- It creates a property called speak and gives it a function as its value.
- protoRabbit acts as a container for the properties that are shared by all rabbits.
- killerRabbit contains properties that apply only to itself (i.e., its type) and derives shared properties from its prototype, protoRabbit.

Prototypes

- Prototypes can also contain data properties:

```
let protoRabbit = {
  city: 'Madrid'
  speak(line) {
    console.log(`The ${this.type} rabbit says '${line}'`);
  }
};

let killerRabbit = Object.create(protoRabbit);
killerRabbit.type = "killer";
killerRabbit.speak("SKREEEE!");
// → The killer rabbit says 'SKREEEE!’
killerRabbit.city
// → 'Madrid'
```

Any Questions?

Constructors

- JavaScript's prototype can be interpreted as a somewhat informal object-oriented class.
- A class defines the shape of an object: What methods and properties it has.
- As does a JavaScript prototype.
- Recall, in object-oriented-programming an object is an instance of a class created by a constructor.
- In JavaScript, an object is an instance of a prototype created by Object.create.
- Thus, Object.create is a constructor.

- JavaScript also treats a function call with the keyword new in front of it as a constructor.

```
function Rabbit(type) {
  this.type = type;
}
Rabbit.prototype.speak = function(line) {
  console.log(`The ${this.type} rabbit says '${line}'`);
};

let weirdRabbit = new Rabbit("weird");
console.log(weirdRabbit.speak("Hello World!"))

// → The weird rabbit says 'Hello World!'
```

By convention, the names of constructors are capitalized so that they can easily be distinguished from other functions.

Class Notation

- In 2015, JavaScript introduced an easier way to create classes with the class keyword:

```
class Rabbit {
  constructor(type) {
    this.type = type;
  }
  speak(line) {
    console.log(`The ${this.type} rabbit says '${line}'`);
  }
}

let killerRabbit = new Rabbit("killer");
console.log(killerRabbit.speak("Hello World!"))
// → The killer rabbit says 'Hello World!’
```

Class Notation

- Class declarations currently allow only methods.
- This can be somewhat inconvenient when you want to save a non-function value in there.
- The next version of the language will probably improve this.
- For now, you can create such properties by directly manipulating the prototype after you've defined the class.

- Like function, class can be used both in statements and in expressions.
- When used as an expression, it doesn't define a binding but just produces the constructor as a value.

```
let object = new class { getWord() { return "hello"; } };
console.log(object.getWord());
// → hello
```

Any Questions?

Adding or Overriding Object Properties

- You can add a property to an object, whether it is present in the prototype or not.
- If there was already a property with the same name in the prototype, this property will be overwritten.
- Overriding properties that exist in a prototype can be a useful thing to do.
- Overriding can be used to express exceptional properties in instances of a more generic class of objects, …
- … while letting the nonexceptional objects take a standard value from their prototype.

```
//Adding a property
Rabbit.prototype.teeth = "small";
console.log(killerRabbit.teeth);
// → small

//Overriding a property
killerRabbit.teeth = "long, sharp, and bloody";
console.log(killerRabbit.teeth);
// → long, sharp, and bloody
console.log(blackRabbit.teeth);
// → small
console.log(Rabbit.prototype.teeth);
// → small
```

JavaScript Polymorphism

- The polymorphism is a core concept of an object-oriented paradigm that provides a way to perform a single action in different forms.
- It provides an ability to call the same method on different JavaScript objects.

```
Rabbit.prototype.toString = function() {
  return `a ${this.type} rabbit`;
};

console.log(blackRabbit.toString());
// → a black rabbit
```

Any Questions?

Summary

- When a function is called as a method the binding called this in its body automatically points at the object that it was called on.

- In addition to their set of properties, most objects also have a prototype.
- A prototype is another object that is used as a fallback source of properties.
- JavaScript's prototype can be interpreted as a somewhat informal object-oriented class.
- The new key word can be used as a constructor to create a new instance of a prototype.
- There's a class notation that provides a clear way to define a constructor and its prototype.

- JavaScript also supports:
  - Adding or Overriding Object Properties
  - Polymorphism – overriding object methods

Any Questions?

In-Class Problem

- Use polymorphism to change the Rabbit class, so that this code:

```
let killerRabbit = new Rabbit("killer");
killerRabbit.speak("Hello World!")
```

- prints out:

The rabbit of type killer says 'Hello World!'

- Do not re-write the existing Rabbit class.

```
class Rabbit {
  constructor(type) {
    this.type = type;
  }
  speak(line) {
    console.log(`The ${this.type} rabbit says '${line}'`);
  }
}
```