Object-oriented JavaScript

tl;dr

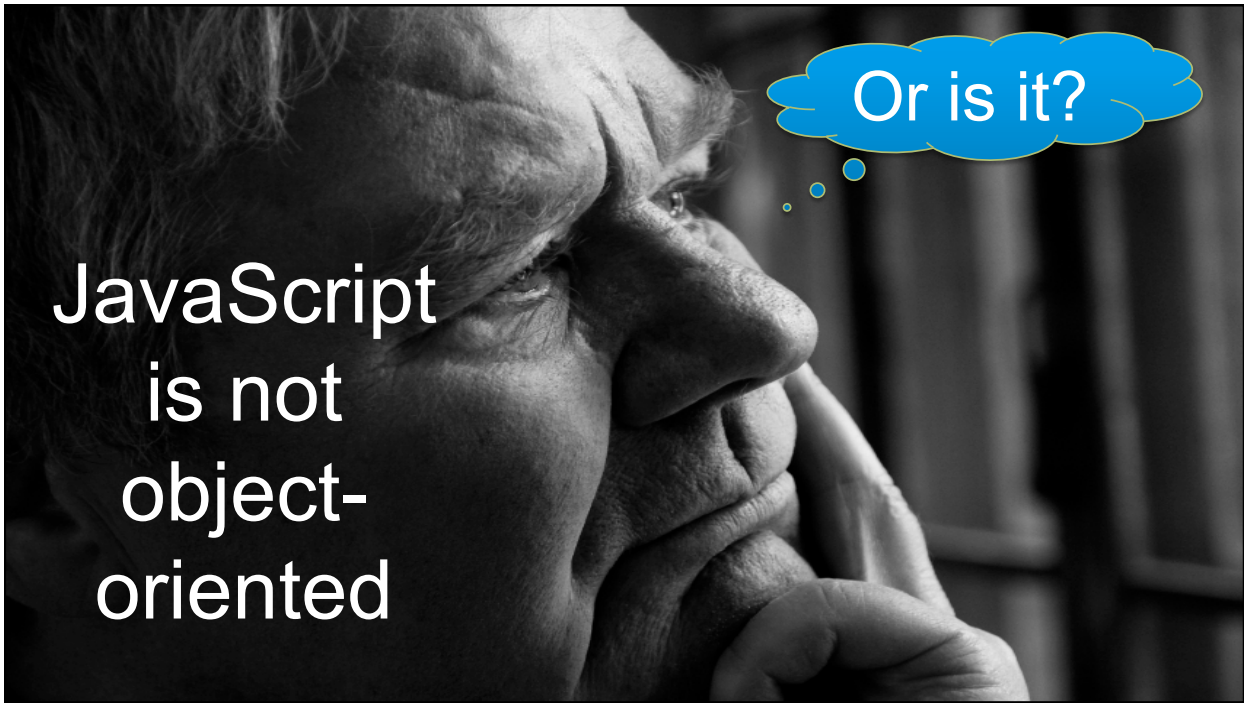
- JavaScript is not object-oriented natively but we can simulate most OO traits.
- We create objects, properties, and methods on the fly -- no class needed
- But the class keyword was added in ES2015 to make OO devs like JavaScript more
- It gives us classes, properties, methods, statics, and accessors
- But it does not give us traditional inheritance, JavaScript still only uses prototypal inheritance

To be OO, we need certain things

- ☐ Objects
- ☐ Classes
- ☐ Properties
- ☐ Methods
- ☐ Constructors
- ☐ Encapsulation
- ☐ Private members
- ☐ Public members
- ☐ Static members
- ☐ Accessor functions
- ☐ Overloading
- ☐ Inheritance
- ☐ Overriding
- ☐ Interfaces

JavaScript has no ...

- True classes
- Traditional inheritance
- Overloading
- Polymorphism
- Interfaces
- Namespaces



Declaring Objects

Overview

- There are generally two types of objects

1. Declared
   - One copy
   - Like a static object
2. Created
   - Is instantiated
   - Like an instance object

We create objects on the fly … no need for classes

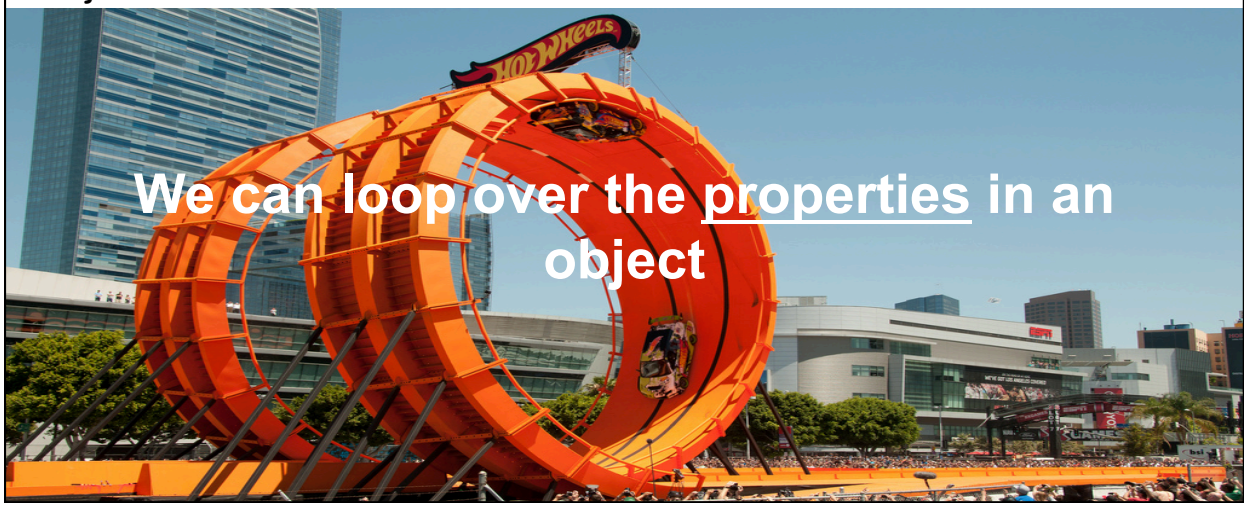
```
const obj = {
  firstName: "Meg",
  lastName: "Griffin",
  pathology: "Rejection",
};
```

Properties

Properties are similarly created dynamically

```
const person1 = {
  lastName: "Griffin",
  firstName: "Stewie",
  city: "Quahog",
  state: "RI",
};
document.write(`The person is
  ${person1.firstName} ${person1.lastName}`);
document.write("He is from " + person1["city"]);
var prop="state";
document.write("He lives in " + person1[prop]);
```

```
for (let propName in person1) {
  console.log(propName, person1[propName]);
}
```



Remember, objects can contain anything

```
var family = {
  name: "The Griffins",          // A string
  incomeInDollars: 34000,        // A number
  dog: brian,                    // Another object
  parents: [ peter, lois ],      // An array
  playTheme: function () {
    return "It seems to me " +
      "that all you see is ...";
  }  // A function
}
•
```

- How can I see if an object has a given property?
- The *in* operator
- if ('neighbor' in family) { ... }
- //It has a family.neighbor property
- else
- //It does not.
- Even if it is undefined, it is there.
- This is better than looking for truthiness since it would return a false negative if family.neighbor is 0 or ""

Property shorthands

```
const o = { foo, bar, ... }
```

- In a JavaScript object
- Note: no colons (:)

Traditional way

```
var a = 1; var b = 2;
var foo = {
  a: a,
  b: b,
  c: function (d) {
    // do stuff
  }
};
// foo has two properties that happen to be
named the same as variables and one function
```

New way

```
const a = 1; const b = 2;
const foo = {
  a,
  b,
  c (d) { // do stuff }
};
// foo has two properties that happen to be
named the same as variables and one function
```

Methods

Functions are objects themselves

- So they can be passed around like data

```
btn.addEventHandler('click',function () {
  // Do stuff
});
```

- They can be assigned to a variable

```
const doStuff = function () { // Do stuff });
```

- ... Even a property in a JSON object

```
const family = {
  ...
  playTheme: function () {
    return "It seems to me " +
      "that all you see is ...";
  }
}
```

So to run that last function, w
to go ...

```
let x = family.playTheme();
```

Classes

ES2015 gives us a different way to write this

```
class Person{
  attack() {
    // Do stuff in this method
  }
}
```

Note: methods don't need *this.* nor the function keyword

- Still not real classes, though!
- Just syntactic sugar on top of a constructor function

class is there to make OO devs feel better

All you're doing when you create a class is opting into a less powerful, less flexible mechanism and a whole lot of pitfalls and pain. (sic)

I think classes in ES6 are a bad part because they lock you into that paradigm.

Any time you can write a function instead of a class, do.

Each object knows it's constructor

```
function attack(otherPerson) {
  if (! (otherPerson instanceof Person)) {
    throw "That's not a person";
  }
  // Do stuff here
}
```

Note: "of" is not camel-cased

Syntax:

- object instanceof ConstructorFunction
- Returns a bool

getters and setters

```
class Person {
  ...
  get alias() { return this._alias; }
  set alias(newValue) {
    this._alias = newValue;
  }
  move(speed) { // Do stuff to move }
  attack(foe) {
    let d = `${this._alias} is attacking ${foe.alias}`;
    // Do other attacking stuff here
  }
}
const p = new Person();
p.alias = "Joker";  // Calls set
console.log(p.alias); // Calls get
```

formal constructors

```
class Person {
  constructor(first, last){
    this._first = first;
    this._last = last;
  }
  doStuff() {
    return `${this._first} ${this._last}
            doing stuff.`;
  }
}

const p = new Person("Talia", "Al-Ghoul");
console.log(p.doStuff());
```

static members

```
class Person {
  constructor(first, last){
    this._first = first;
    this._last = last;
  }
  static doStuff() {
    return `Doing stuff.`;
  }
}

const p = new Person("Tim", "Drake");
console.log(Person.doStuff());
```

Note: classes can only have ...

```
class foo {
  constructor() { ...} // Constructor
  get x() { return this._x; } // Getters
  set x(v) { this._x = v; } // Setters
  method1() { ... } // Methods
  static doIt { ... } // Static methods
}
```

- Can **not** have "this.", "let", "var" that would be a syntax error

Inheritance

JavaScript does not support traditional inheritance

- In c#

```
public class Employee : Person
{
  public decimal Salary { get; set; }
  public overrides string ToString() {
    return string.Format("{0}, {1}",
      this.Last, this.First);
  }
}
```

- But it does have these things called *prototypes*

prototype

/ˈprōdəˌtīp/
noun

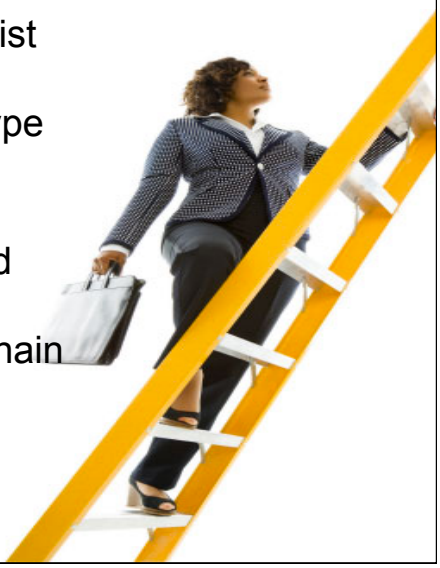
1. the original or model on which something is based or formed.
2. someone or something that serves to illustrate the typical qualities of a class; model; exemplar. ie: She is the prototype of a student activist.
3. something analogous to another thing of a later period. ie: a Renaissance prototype of our modern public housing.
4. Biology. an archetype; a primitive form regarded as the basis of a group.

From Random House Dictionary via Dictionary.com

The prototype serves as a 'backup' of sorts

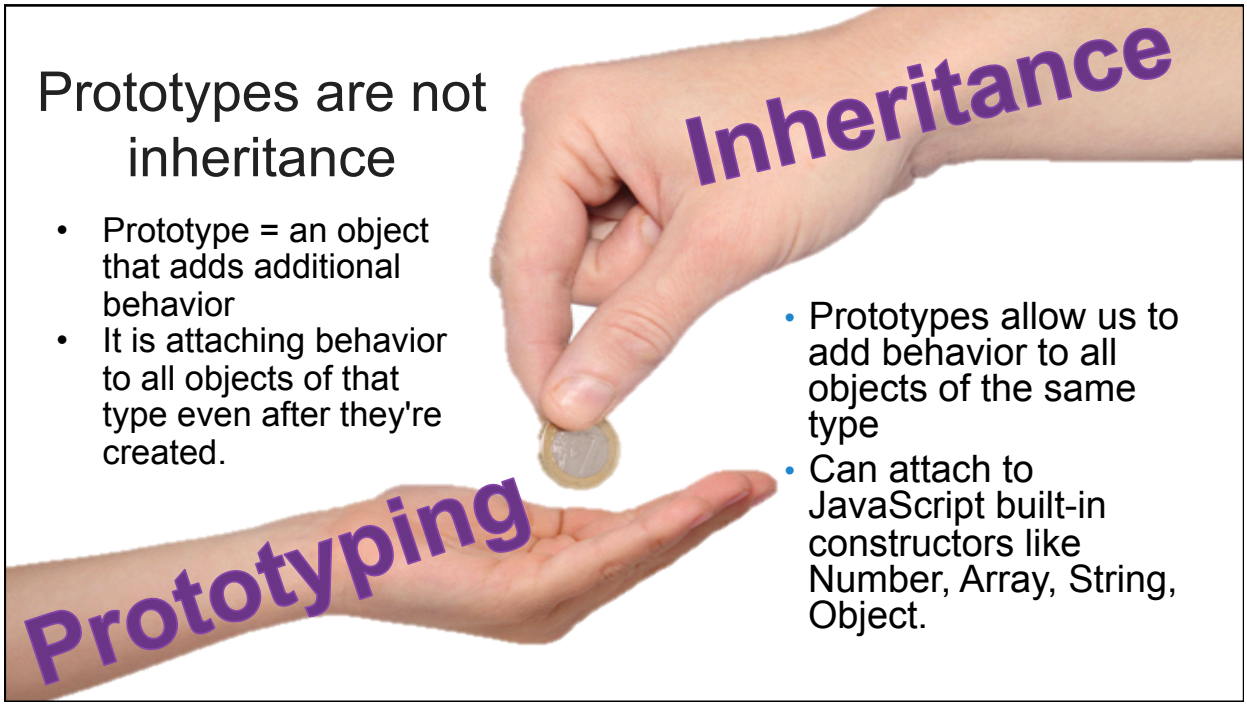
If we try to access a property which doesn't exist on the object itself,

1. JavaScript looks in its constructor's prototype for that property.
2. If it is present, that property is reported
3. If not, the prototype's prototype is searched
4. ... and so on and so on and so on
5. ... until JavaScript reaches the top of the chain
6. Then it reports that property as 'undefined'



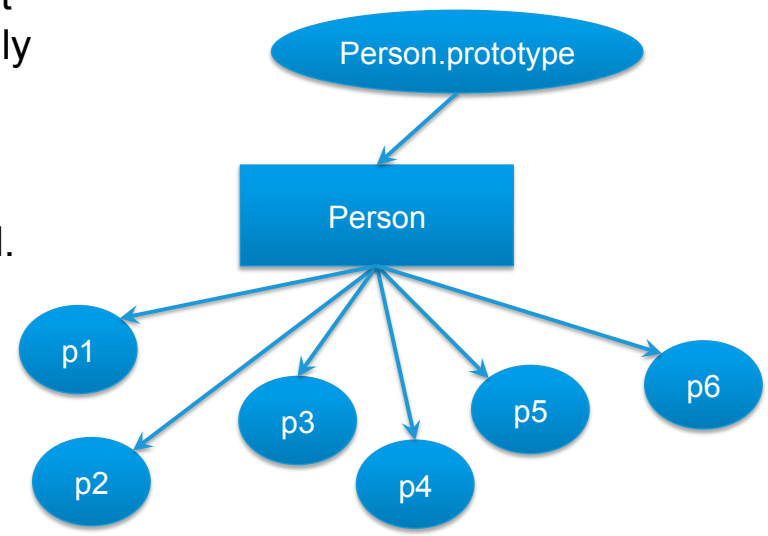
Prototypes are not inheritance

- Prototype = an object that adds additional behavior
- It is attaching behavior to all objects of that type even after they're created.



- Prototypes allow us to add behavior to all objects of the same type
- Can attach to JavaScript built-in constructors like Number, Array, String, Object.

- The object's prototype is never checked.
- Regular objects don't have a prototype, only Functions do.
- The object's constructor's prototype is checked.
- Thus you can add prototypes after instantiation and it is there.



The prototype on instances is read-only

```
var p = new Person();
console.log(p.firstname);
console.log(p.prototype.firstname);
p.firstname = "Joe";
p.prototype.firstname = "Joe";
p.__proto__.firstname = "Joe";
```

Person does not have a firstname, but its prototype does.

Works – pulls from the prototype

Does not work b/c objects don't have prototypes; Functions do

Works. Adds a firstname to p

Does not work. Can't set property 'firstname' of 'undefined'

Works, but really bad idea. Sets firstname for all instances. Breaks encapsulation

So you can see why this isn't exactly like traditional inheritance

ES2015 classes allow prototypal inheritance

- extends - Assigns a prototype
- super - Refers to the prototype

- Don't be tricked!
- These don't add any new capabilities. They're only more direct ways of working with prototypes.

extends registers a prototype

```
class Employee extends Person {
  constructor(first, last, salary) {
    super(first, last);
    this._salary = salary;
  }
}
const e = new Employee("Chris","Lee",75000);
console.log(e.doStuff()); // Chris Lee doing stuff
```

tl;dr

- JavaScript is not object-oriented natively but we can simulate most OO traits.
- We create objects, properties, and methods on the fly -- no class needed
- But the class keyword was added in ES2015 to make OO devs like JavaScript more
- It gives us classes, properties, methods, statics, and accessors
- But it does not give us traditional inheritance, JavaScript still only uses prototypal inheritance