Using Prototypes

[(https://github.com/learn-co-curriculum/phase-1-using-prototypes)](https://github.com/learn-co-curriculum/phase-1-using-prototypes) [(https://github.com/learn-co-curriculum/phase-1-using-prototypes/issues/new)](https://github.com/learn-co-curriculum/phase-1-using-prototypes/issues/new)

Learning Goals

- Identify inefficiency in constructor function object orientation
- Recognize the prototype as a means for reducing inefficiency

Introduction

```
function User(name, email) {
  this.name = name;
  this.email = email;
  this.sayHello = function () {
    console.log(`Hello everybody, my name is ${this.name} whom you've been
mailing at ${this.email}!`);
  };
}

const lauren = new User("lauren", "lauren@gmail.com");
lauren.sayHello();
// => Hello everybody, my name is lauren whom you've been mailing at lauren@gmail.com!
```

Let's assume the following. Let's assume that the `name` property costs 32 bytes of space. Let's also assume that the `email` property costs 32 bytes of space. Let's lastly assume that a function costs 64 bytes of space.

So to create our `lauren` instance we pay a cost of `32 + 32 + 64 = 128` bytes. But now let's assume that we want to create many more `User` s - Facebook numbers of users. Lets suppose a paltry 1 million users. That would be: 128 million bytes of space. While memory and disk are getting bigger and cheaper all the time, we'd like to be efficient whenever possible.

The key to gaining efficiency is the *prototype*.

Identify Inefficiency In Constructor Function Object Orientation

In our example the `name` s vary, so we can't economize there. The `email` s vary, so we can't economize there either. But the method, `sayHello` is the same in every instance: "in my current context return a template string with this current context's values."

We would like to tell all instances of `User` that they have a shared place to find methods. That place is called the "prototype."

Recognize the Prototype as a Means for Reducing Inefficiency

We access the prototype of a constructor function by typing the constructor function's name, and adding the attribute `.prototype` . So for `User` it's `User.prototype` . Attributes that point to functions in this JavaScript `Object` will be shared to all instances made by that constructor function.

```
function User(name, email) {
  this.name = name;
  this.email = email;
}

User.prototype.sayHello = function () {
  console.log(`Hello everybody, my name is ${this.name}`);
};

const sarah = new User("sarah", "sarah@example.com");
const lauren = new User("Lauren", "lauren@example.com");

sarah.sayHello(); //=> // "Hello everybody, my name is sarah!"
lauren.sayHello(); //=> // "Hello everybody, my name is Lauren!"
```

The prototype is just a JavaScript `Object` .

```
User.prototype;
// {sayHello: f, constructor: f}
typeof User.prototype;
// object
```

To prove the efficiency of sharing methods via prototype:

```
lauren.sayHello === sarah.sayHello; //=> true
```

Conclusion

In this lesson you've seen the constructor' prototype and seen how it can be used to share functionality between instances.

Your pattern for writing OO in JavaScript (Prototype-based) is the following:

```
function User(name, email) {
  this.name = name;
  this.email = email;
}

User.prototype.sayHello = function () {
  console.log(`Hello everybody, my name is ${this.name}`);
};

const sarah = new User("sarah", "sarah@example.com");
```