Let's talk about prototypes!



Initial questions 😊

- 1. How many primitives have JS?
- 2. Everything in the language is an object?
- 3. Which is the result of typeof null??

"Complex primitives"

- 1. Function
- 2. String
- 3. Number
- 4. ...

Common situation

```
...
} catch (e) {
  return Promise.reject(new Error("Hi there little boy, I'm an error"))
}
```

Lets remember some things!

```
function Particle() {
  this.x = 50;
  this.y = 100;
}

var newParticle = new Particle();
```

What is doing the new keyword??

Objects in the soup

What if I want to add a function to my object??

```
function Particle() {
  this.x = 50;
  this.y = 100;
  this.position = function () {
    console.log("x: " + this.x + " - " + "y: " + this.y);
  };
};
```

Inherit from this, from that...

• Inheritance avoid redefine some common beahivors

```
class Vehicle {
  turnOn() {
    console.log("I'm alive!!");
  }
}
class Car extends Vehicle {}
```

Polymorphism for the winners

 Describes the idea that a general behavior from a parent class can be overridden in a child class to give it more specifics

```
class Vehicle {
  turnOn() {
    console.log("I'm alive!!");
  }
}

class Car extends Vehicle {
  turnOn() {
    console.log("I'm alive, but as a car");
  }
}
```

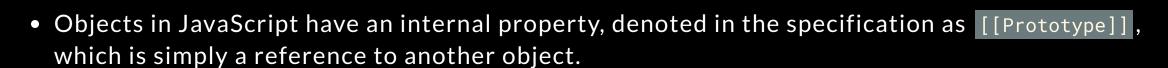
Classes in JS are sugar 🚉!



Mixins (like a michelada)

```
// vastly simplified `mixin(..)` example:
function mixin(sourceObj, targetObj) {
  for (var key in sourceObj) {
    // only copy if not already present
    if (!(key in targetObj)) {
      targetObj[key] = sourceObj[key];
  return targetObj;
var Vehicle = {
  turnOn: function () {
    console.log("I'm alive!!");
// This overrides turnOn implementation
var Car = mixin(Vehicle, {
  turnOn: function () {
    console.log("I'm alive, but as a car");
});
```

All cool but..where are the protoypes?!



```
{
    ...
    __proto__: Object
}
```

Prototypes in action 💡

Let's take the snippet in slide 5

```
function Particle() {
  this.x = 50;
  this.y = 100;
  this.position = function () {
    console.log("x: " + this.x + " - " + "y: " + this.y);
  };
};
var p1 = new Particle();
var p2 = new Particle();
```

What it's going to happen?

Prototypes in action x2

What if instead we define the function in a common place? What if...

```
function Particle() {
  this.x = 50;
  this.y = 100;
}

Particle.prototype.show = function () {
  console.log("x: " + this.x + " - " + "y: " + this.y);
};
```

Let's console it!

Prototypes in action x3

Why is this useful?

```
var superDuperLibrary = require("superDuperLibrary");
var foo = superDuperLibrary.fooObject();

/**
    * > Uncaught TyperError
    * superDuperLibrary.dummy is not a function
    **/
foo.dummy();
```

Guess what!! We can do this...

```
superDuperLibrary.fooObject.prototype.dummy = function() { ... }
```

Prototype chain, to the infinite and beyond...

```
var anotherObject = {
   a: 2,
};

// create an object linked to `anotherObject`
var myObject = Object.create(anotherObject);

myObject.a; // 2
```

mmm ok, let's make us some questions:

- 1. myObject.a exists??
- 2. What its going to happen if some property doesn't exists?
- 3. How deep its going to be the look-up process?

Inheritance with...prototypes?? Wait, what??

Remember the Vehicle, Car classes? Let's use them in prototype approach

```
function Vehicle() {
  this.engine = "engine";
  this.wheels = 4:
Vehicle.prototype.turnOf = function () {
  console.log("I'm dead");
Vehicle.prototype.turnOn = function () {
  console.log("I'm alive!!");
function Car() {
  Vehicle.call(this);
var car = new Car();
```

What happened??! Why we are not seeing turn0n and turn0f in __prototype__?

KUniversity

Inheritance with...prototypes?? Wait, what?? x2

What its going to happen if we do this?

Car.prototype = Vehicle.prototype;

Soooooo TL;DR

- The prototype is an object that have some cool basic and underground functionality
- All normal objects have the built-in Object.prototype as the top of the prototype chain
- Objects end up linked to each other via an internal [[Prototype]] chain.
- For a variety of reasons, not the least of which is terminology precedent, inheritance (and prototypal inheritance) and all the other OO terms just do not make sense when considering how JavaScript actually works (not just applied to our forced mental models).

Links

- 1. 9.19: Prototypes in JS
- 2. 9.20: Inheritance with Prototype in JS
- 3. <u>If you want to go deeper!</u>

GIVE ME THE CODE!!