Getting Started

First be sure you have MongoDB and Node.js installed.

Next install Mongoose from the command line using `npm`:

```
$ npm install mongoose
```

Now say we like fuzzy kittens and want to record every kitten we ever meet in MongoDB. The first thing we need to do is include mongoose in our project and open a connection to the `test` database on our locally running instance of MongoDB.

```
// getting-started.js
var mongoose = require('mongoose');
mongoose.connect('mongodb://localhost/test');
```

We have a pending connection to the test database running on localhost. We now need to get notified if we connect successfully or if a connection error occurs:

```
var db = mongoose.connection;
db.on('error', console.error.bind(console, 'connection error
db.once('open', function() {
  // we're connected!
});
```

Once our connection opens, our callback will be called. For brevity, let's assume that all following code is within this callback.

With Mongoose, everything is derived from a Schema. Let's get a reference to it and define our kittens.

```
var kittySchema = mongoose.Schema({
    name: String
});
```

mongoose

- quick start
- contributors
- prior releases

So far so good. We've got a schema with one property, `name`, which will be a `String`. The next step is compiling our schema into a Model.

```
var Kitten = mongoose.model('Kitten', kittySchema);
```

A model is a class with which we construct documents. In this case, each document will be a kitten with properties and behaviors as declared in our schema. Let's create a kitten document representing the little guy we just met on the sidewalk outside:

```
var silence = new Kitten({ name: 'Silence' });
console.log(silence.name); // 'Silence'
```

Kittens can meow, so let's take a look at how to add "speak" functionality to our documents:

```
// NOTE: methods must be added to the schema before compiling
kittySchema.methods.speak = function () {
  var greeting = this.name
    ? "Meow name is " + this.name
    : "I don't have a name";
  console.log(greeting);
}

var Kitten = mongoose.model('Kitten', kittySchema);
```

Functions added to the `methods` property of a schema get compiled into the `Model` prototype and exposed on each document instance:

```
var fluffy = new Kitten({ name: 'fluffy' });
fluffy.speak(); // "Meow name is fluffy"
```

We have talking kittens! But we still haven't saved anything to MongoDB. Each document can be saved to the database by calling its save method. The first argument to the callback will be an error if any occured.

mongoose

- home
- FAQ
- plugins
- change log
- support
- fork
- guide
- API docs

```
fluffy.save(function (err, fluffy) {
  if (err) return console.error(err);
  fluffy.speak();
});
```

Say time goes by and we want to display all the kittens we've seen. We can access all of the kitten documents through our Kitten model.

```
Kitten.find(function (err, kittens) {
  if (err) return console.error(err);
  console.log(kittens);
})
```

We just logged all of the kittens in our db to the console. If we want to filter our kittens by name, Mongoose supports MongoDBs rich querying syntax.

```
Kitten.find({ name: /^fluff/ }, callback);
```

This performs a search for all documents with a name property that begins with "Fluff" and returns the result as an array of kittens to the callback.

Congratulations

That's the end of our quick start. We created a schema, added a custom document method, saved and queried kittens in MongoDB using Mongoose. Head over to the guide, or API docs for more.

mongoose