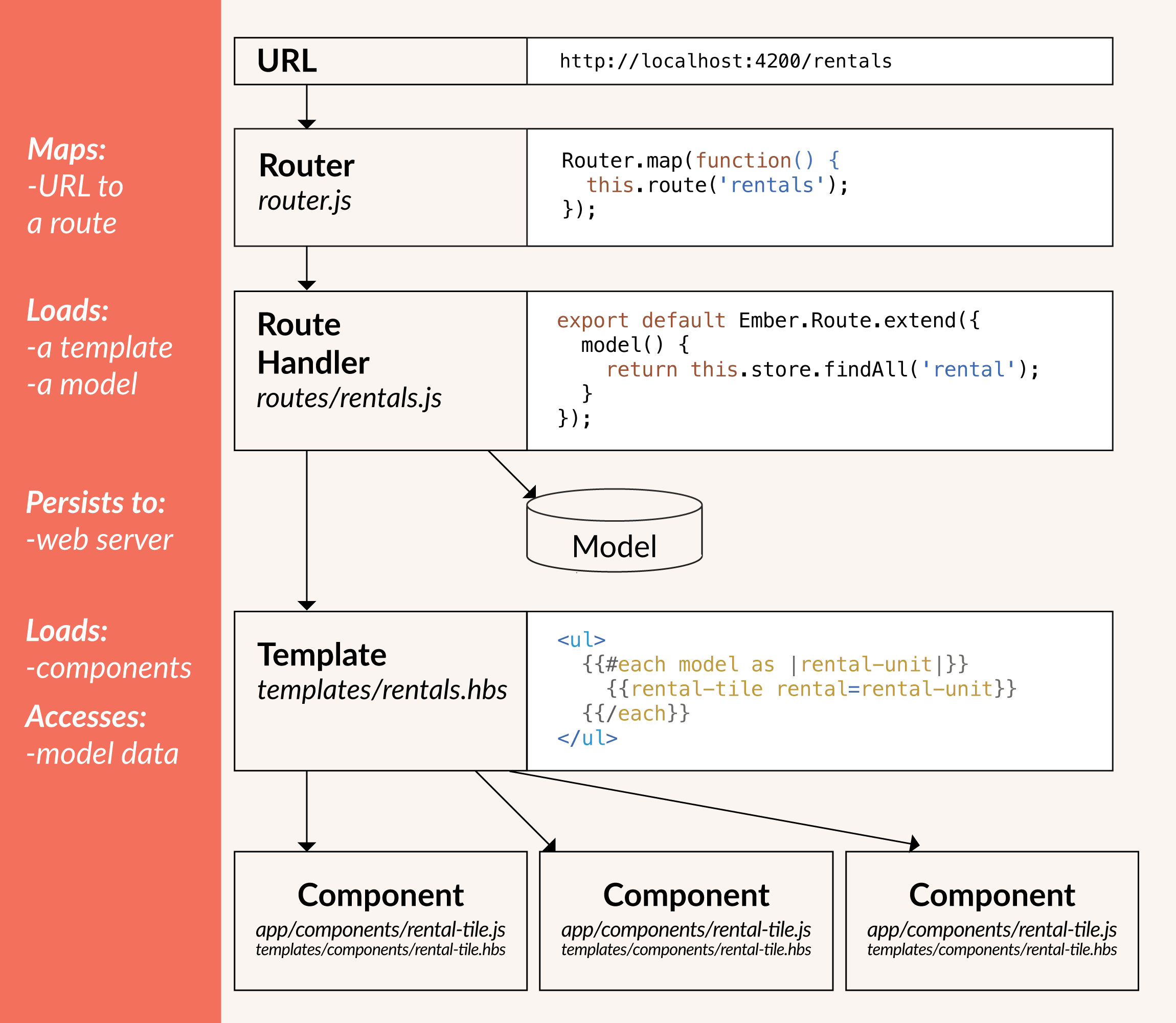
# Concepts

<https://guides.emberjs.com/v2.12.0/getting-started/core-concepts/>

How an Ember application works



## Router – maps the URL

One of the responsibilities of a route is to load a model.

The URL can be set in a few ways:

* The user loads the app for the first time.
* The user changes the URL manually, such as by clicking the back button or by editing the address bar.
* The user clicks a link within the app.
* Some other event in the app causes the URL to change.

Ember **router** **maps the URL** to a **route handler**.

## Route Handlers – page

The route handler then typically does two things:

* It **renders a template**.
* It **loads a model** that is then available to the template.

## Templates – HTML

Anything that is valid **Handlebars** syntax is valid Ember syntax

Templates’ context: a component or a route's controller. Provides properties - eg:{{name}}.

{{}} contain properties, helpers and components.

## Handlebars Helper

Decorate the data in our templates.

Built-in Helpers:

## Components – reusable

While templates describe how a user interface looks, components control how the user interface behaves.

* A template written in Handlebars – look.
* A source file written in JavaScript – component's behavior.

Web components completely isolated, are not directly effected by any surrounding CSS or JavaScript.

Other components can be called inside of a component's template.

To call a component in an Ember app, you must use {{curly-brace-syntax}}.

## Controllers

[Controllers](https://guides.emberjs.com/v2.13.0/controllers/) contain actions and properties available to the template of its corresponding route.

Ember will know that a controller with the name of rentals will apply to the route with the same name.

## Ember Data

Data management library called [Ember Data](https://github.com/emberjs/data) to help deal with persistent application data.

Ember Data requires you to **define the structure of the data** you wish to provide to your application by extending [DS.Model](http://emberjs.com/api/data/classes/DS.Model.html).

The [store service](http://emberjs.com/api/data/classes/DS.Store.html) is injected into all routes and components in Ember.

## Extend vs Mixin

<https://dockyard.com/blog/2015/11/09/best-practices-extend-or-mixin>

Code duplication

### Ember.Object.extend()

Ember.Mixin and Ember.Object.extend(). Both of these essentially achieve the same end result

// routes/index.js

export default Ember.Route.extend({

// properties and methods

});

IndexRoute **->** Ember.Route **->** Ember.Object **->** Ember.CoreObject

Any properties defined in the parent object will be shared among any child object. Inside the init constructor, each object instance gets its own unique set of properties.

### Ember.Mixin

Mixins don’t get extended. Any properties or functionality that is defined inside the mixin will be shared among all classes containing this mixin.

// mixins/foo.js

export default Ember.Mixin.create({

bars: []

)};

// components/A.js

import FooMixin from ‘../mixins/foo’;

export default Component.extend(FooMixin);

// components/B.js

import FooMixin from ‘../mixins/foo’;

export default Component.extend(FooMixin);

A and B sharing the **same** bars array. Any changes made to the array will be reflected in both components.

## Object Model

### Computed Properties

In a nutshell, computed properties let you declare functions as properties.

Person = Ember.Object.extend({  
 *// these will be supplied by `create`* **firstName**: **null**,  
 **lastName**: **null**,  
  
 **fullName**: Ember.computed(**'firstName'**, **'lastName'**, **function**() {  
 let firstName = **this**.get(**'firstName'**);  
 let lastName = **this**.get(**'lastName'**);  
  
 **return `**${firstName} ${lastName}**`**;  
 })  
 });  
  
let ironMan = Person.create({  
 **firstName**: **'Tony'**,  
 **lastName**: **'Stark'**});  
  
ironMan.get(**'fullName'**); *// "Tony Stark"*

This property was **cached**. Cache will be updated if params change.

The alternative syntax, with prototype extensions, might look like:

**fullName**: **function**() {  
 **return this**.get(**'firstName'**) + **' '** + **this**.get(**'lastName'**);  
}.property(**'firstName'**, **'lastName'**)

app/router.js

Router.map(function() {

this.route('about', { path: '/about' });

this.route('favorites', { path: '/favs' });

});

Nested

Router.map(function() {

this.route('posts', function() {

this.route('new');

});

});

Hbs

{{#link-to "index"}}<img class="logo">{{/link-to}}

<nav>

{{#link-to "about"}}About{{/link-to}}

{{#link-to "favorites"}}Favorites{{/link-to}}

</nav>

| templates/posts.hbs | |
| --- | --- |
| 1  2  3 | <h1>Posts</h1>  <!-- Display posts and other content -->  {{outlet}} |

# Project structure

|--app

|--config

|--node\_modules

|--public

|--tests

|--vendor

<other files>

ember-cli-build.js

package.json

README.md

testem.js

**app**: This is where folders and files for models, components, routes, templates and styles are stored. The majority of **your coding** on an Ember project happens in this folder.

**node\_modules / package.json**: This directory and file are from npm. npm is the **package manager** for Node.js. Ember is built with Node and uses a variety of Node.js modules for operation. The package.json file maintains the list of current npm **dependencies** for the app. Any Ember CLI add-ons you install will also show up here. Packages listed in package.json are installed in the node\_modules directory.

**public**: This directory contains assets such as images and fonts.

**vendor**: This directory is where front-end dependencies (such as JavaScript or CSS) that are not managed by Bower go.

Ember CLI uses ECMAScript 2015 (**ES2015** for short or previously known as **ES6**)

app/router.js

import Ember from 'ember';

import config from './config/environment';

const Router = Ember.Router.extend({

location: config.locationType,

rootURL: config.rootURL

});

Router.map(function() {

});

export default Router;

import Ember from 'ember'; gives us access to the actual Ember.js library as the variable Ember.

const is a way to declare a read-only variable to make sure it is not accidentally reassigned elsewhere.

export default Router; makes the Router variable defined in this file available to other parts of the app.

## Define a Route

For now, you can think of routes as being the different **pages** that make up your application.

ember generate route scientists

That's Ember telling you that it has created:

1. A template to be displayed when the user visits /scientists.
2. A Route object that fetches the model used by that template.
3. An entry in the application's router (located in app/router.js).
4. A unit test for this route.

### Loading the model with data for the page

| In a route's model() method, you return **whatever data** you want to make available to the template.  app/routes/scientists.js | |
| --- | --- |
| 1  2  3  4  5  6  7 | import Ember from 'ember';  export default Ember.Route.extend({  model() {  return ['Marie Curie', 'Mae Jemison', 'Albert Hofmann'];  }  }); |

If you need to fetch data asynchronously, the model() method supports any library that uses [JavaScript Promises](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise).

Ember will use the model object returned above and save it as an attribute called model, available to the rentals template we generated with our route in [Routes and Templates](https://guides.emberjs.com/v2.13.0/tutorial/routes-and-templates/#toc_a-rentals-route).

### To HTML

| app/templates/scientists.hbs | |
| --- | --- |
| 1  2  3  4  5  6  7 | <h2>List of Scientists</h2>  <ul>  {{#each model as |scientist|}}  <li>{{scientist}}</li>  {{/each}}  </ul> |

## Component

ember generate component people-list

* installing component
  + create app\components\people-list.js
  + create app\templates\components\people-list.hbs
* installing component-test
  + create tests\integration\components\people-list-test.js

Usage:

app/templates/scientists.hbs

{{**people-list title**=**"List of Scientists" people**=**model**}}

### Autocomplete-component

<https://guides.emberjs.com/v2.13.0/tutorial/autocomplete-component/>

ember g component list-filter

| app/templates/components/list-filter.hbs | |
| --- | --- |
| 1  2  3  4  5 | {{input value=value  key-up=(action **'handleFilterEntry'**)  class="light"  placeholder="Filter By City"}}  {{yield results}} |

The key-up property will be bound to the **handleFilterEntry** action.

app/components/list-filter.js

import Ember from 'ember';

export default Ember.Component.extend({

classNames: ['list-filter'],

value: '',

init() {

this.\_super(...arguments);

this.get('filter')('').then((results) => this.set('results', results));

},

actions: {

**handleFilterEntry**() {

let filterInputValue = this.get('value');

let filterAction = this.get('filter');

filterAction(filterInputValue).then((filterResults) => this.set('results', filterResults));

}

}

});

Usage:

app/templates/rentals.hbs

<div class="jumbo">

<div class="right tomster"></div>

<h2>Welcome!</h2>

<p>

We hope you find exactly what you're looking for in a place to stay.

</p>

{{#link-to 'about' class="button"}}

About Us

{{/link-to}}

</div>

{{#list-filter

filter=(action **'filterByCity'**) {{!-- controllers/rentals.js --}}

as |rentals|}}

<ul class="results">

{{#each rentals as |rentalUnit|}}

<li>{{rental-listing rental=rentalUnit}}</li>

{{/each}}

</ul>

{{/list-filter}}

ember g controller rentals

app/controllers/rentals.js

import Ember from 'ember';

export default Ember.Controller.extend({

actions: {

**filterByCity**(param) {

if (param !== '') {

return this.get('store').query('rental', { city: param });

} else {

return this.get('store').findAll('rental');

}

}

}

});

## Handlebars Helper

ember g helper rental-property-type

|  |  |
| --- | --- |
| 1  2  3  4 | installing helper  create app/helpers/rental-property-type.js  installing helper-test  create tests/unit/helpers/rental-property-type-test.js |

Ex: Let's use a handlebars helper to allow our users to quickly see if a property is "standalone" or part of a "Community".

rental-listing.hbs

<span>Type:</span> {{rental-property-type rental.propertyType}}

| app/helpers/rental-property-type.js | |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  **10**  11  12  13  14  15  16  17 | import Ember from 'ember';  const communityPropertyTypes = [  'Condo',  'Townhouse',  'Apartment'  ];  export function rentalPropertyType([propertyType]) {  if (communityPropertyTypes.includes(propertyType)) {  return 'Community';  }  return 'Standalone';  }  export default Ember.Helper.helper(rentalPropertyType); |

## Ember Data

ember g model rental

|  |  |
| --- | --- |
| 1  2  3  4 | installing model  create app/models/rental.js  installing model-test  create tests/unit/models/rental-test.js |

The [store service](http://emberjs.com/api/data/classes/DS.Store.html) is injected into all routes and components in Ember.

**import** Ember **from 'ember'**;  
  
**export default** Ember.Route.extend({  
 model() {  
 **return this**.get(**'store'**).findAll(**'rental'**);   
 }  
});

## Click Events

# Addons

### ember-cli-mirage

Mirage will allow us to create fake data to work with while developing our app and mimic a running backend server.

|  |  |  |
| --- | --- | --- |
|  | ember install ember-cli-mirage | |
| mirage/config.js | | | |
| 1  2  3  4  5  6  7  8  9  **10**  11  12  13  14  15  16  17  18  19  **20**  21  22  23  24  25  26  27  28  29  **30**  31  32  33  34  35  36  37  38  39  **40**  41  42 | | export default function() {  this.namespace = '/api';  this.get('/rentals', function() {  return {  data: [{  type: 'rentals',  id: 'grand-old-mansion',  attributes: {  title: 'Grand Old Mansion',  owner: 'Veruca Salt',  city: 'San Francisco',  "property-type": 'Estate',  bedrooms: 15,  image: 'https://upload.wikimedia.org/wikipedia/commons/c/cb/Crane\_estate\_(5).jpg'  }  }, {  type: 'rentals',  id: 'urban-living',  attributes: {  title: 'Urban Living',  owner: 'Mike Teavee',  city: 'Seattle',  "property-type": 'Condo',  bedrooms: 1,  image: 'https://upload.wikimedia.org/wikipedia/commons/0/0e/Alfonso\_13\_Highrise\_Tegucigalpa.jpg'  }  }, {  type: 'rentals',  id: 'downtown-charm',  attributes: {  title: 'Downtown Charm',  owner: 'Violet Beauregarde',  city: 'Portland',  "property-type": 'Apartment',  bedrooms: 3,  image: 'https://upload.wikimedia.org/wikipedia/commons/f/f7/Wheeldon\_Apartment\_Building\_-\_Portland\_Oregon.jpg'  }  }]  };  });  } | |

Usage:

**export default** Ember.Route.extend({  
 model() {  
 **return this**.get(**'store'**).findAll(**'rental'**);  
 }  
});

When we call findAll, Ember Data will attempt to fetch rentals from /api/rentals.

* model() is the same as writing model: function()
* Ember CLI uses ECMAScript 2015 (**ES2015** for short or previously known as **ES6**) npm is the **package manager** for Node.js. Ember is built with Node and uses a variety of Node.js modules for operation.
* A dash is required in every component name to avoid conflicting with a possible HTML element, so rental-listing is acceptable but rental isn't.
* The {{outlet}} helper tells Ember where content for our current route (such as about or contact) should be shown.
* An [actions hash](https://guides.emberjs.com/v2.13.0/templates/actions/) is an object in the component that contains functions. These functions are called when the user interacts with the UI, such as clicking.
* The [store service](http://emberjs.com/api/data/classes/DS.Store.html) is injected into all routes and components in Ember.
* The template contains an [{{input}}](https://guides.emberjs.com/v2.13.0/templates/input-helpers) helper that renders as a text field, in which the user can type a pattern to filter the list of cities used in a search. The value property of the input will be kept in sync with the value property in the component.

# JAVASCRIPT

## map()

<https://www.discovermeteor.com/blog/understanding-javascript-map/>

## Strict Mode

<https://www.w3schools.com/js/js_strict.asp>

"use strict"; //The "use strict" directive is only recognized at the **beginning** of a script or a function.

Scope global, local.

Mistyping a variable name creates a new global variable -> "bad syntax" into real errors.

In strict mode, any assignment to a non-writable property, a getter-only property, a non-existing property, a non-existing variable, or a non-existing object, will throw an error.

Not allowed:

* Using a variable, object, without declaring it.
* Deleting a variable (or object), function.
* Duplicating a parameter name.
* Octal numeric literals. var x = 010;
* Escape characters. var x = \010;
* Writing to a read-only property   
  "use strict";  
  var obj = {};  
  Object.defineProperty(obj, "x", {value:0, writable:false});  
    
  obj.x = 3.14;            // This will cause an error
* Writing to a get-only property  
  var obj = {get x() {return 0} };  
    
  obj.x = 3.14;            // This will cause an error
* …

## Method

A JavaScript **method** is a **property** containing a **function definition**.

Create:

*methodName : function() { code lines }*

Usage:

*objectName.methodName()*