ANGULARJS

SUPERHEROIC JAVASCRIPT MVW FRAMEWORK

Created by /

To properly explain AngularJS its necessary to understand the environment in which it was developed.

IN THE BEGINNING THERE WAS HTML...

HTML was originally intended to enable physicists to share and use documents; it contained 18 tags that could be used to indicate things like paragraphs, tables, and lists within a page.

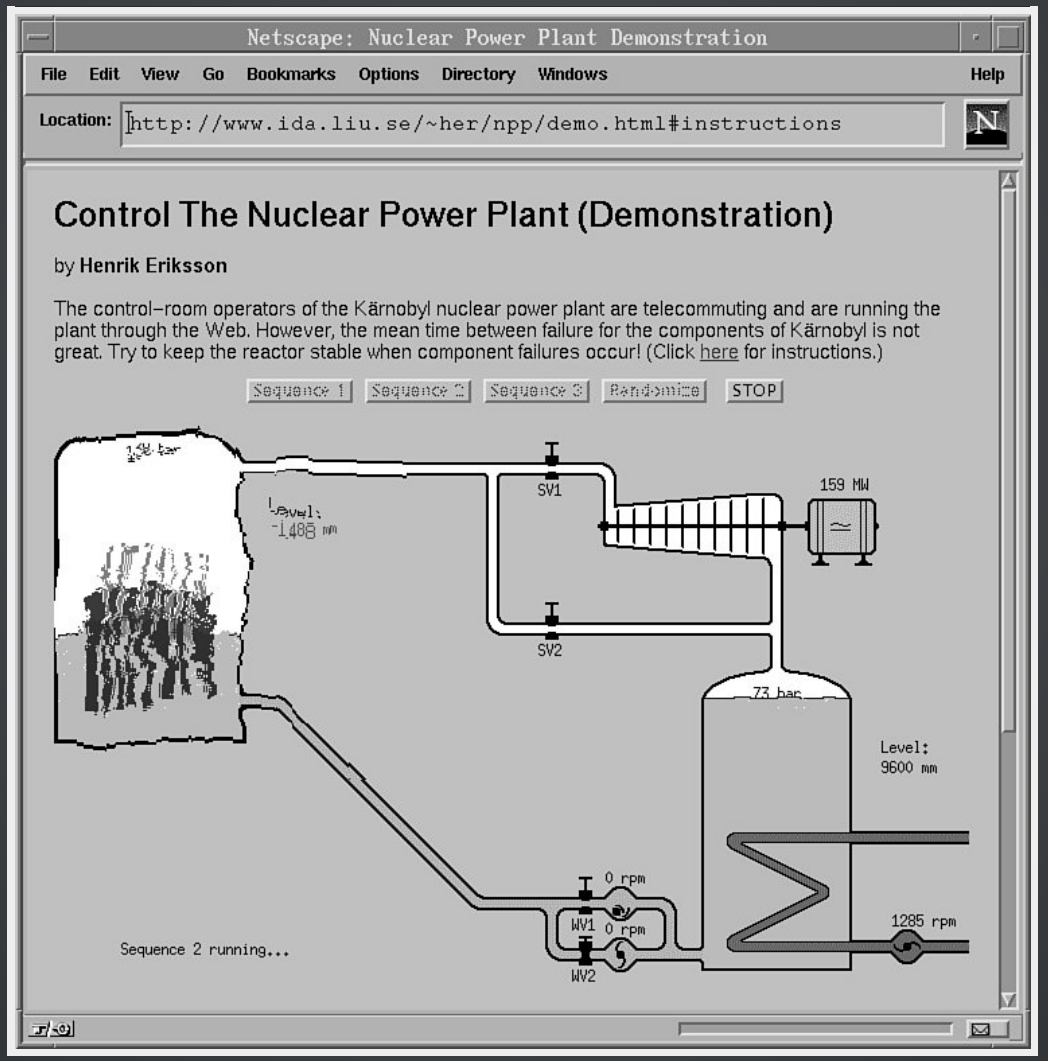
```
<p>
        My favorite elements
        <ol>
                <li>Helium</li>
                <li>Oxygen</li>
                <li>Nitrogen</li>
        </ol>
</p>
```

My favorite elements

1. *Helium*
2. *Oxygen*
3. *Nitrogen*

As the web grew in popularity developers hit a snag: users wanted more interactive applications but HTML could only deliver static content.

ONE IDEA WAS TO USE BROWSER PLUGINS LIKE APPLETS AND FLASH TO BRIDGE THE GAP



THERE WERE A LOT OF PROBLEMS WITH THIS SOLUTION

- Required that users install and maintain runtimes independent of their browser
- Pages took much longer to load
- Look and feel often differed from the rest of the page
- Resulted in pages had an "island of interactivity" surrounded by static content

Another idea was to use server-side frameworks like Coldfusion, JSP, or ASP to dynamically generate pages as they were requested.

These frameworks tended to be slow and very complicated.

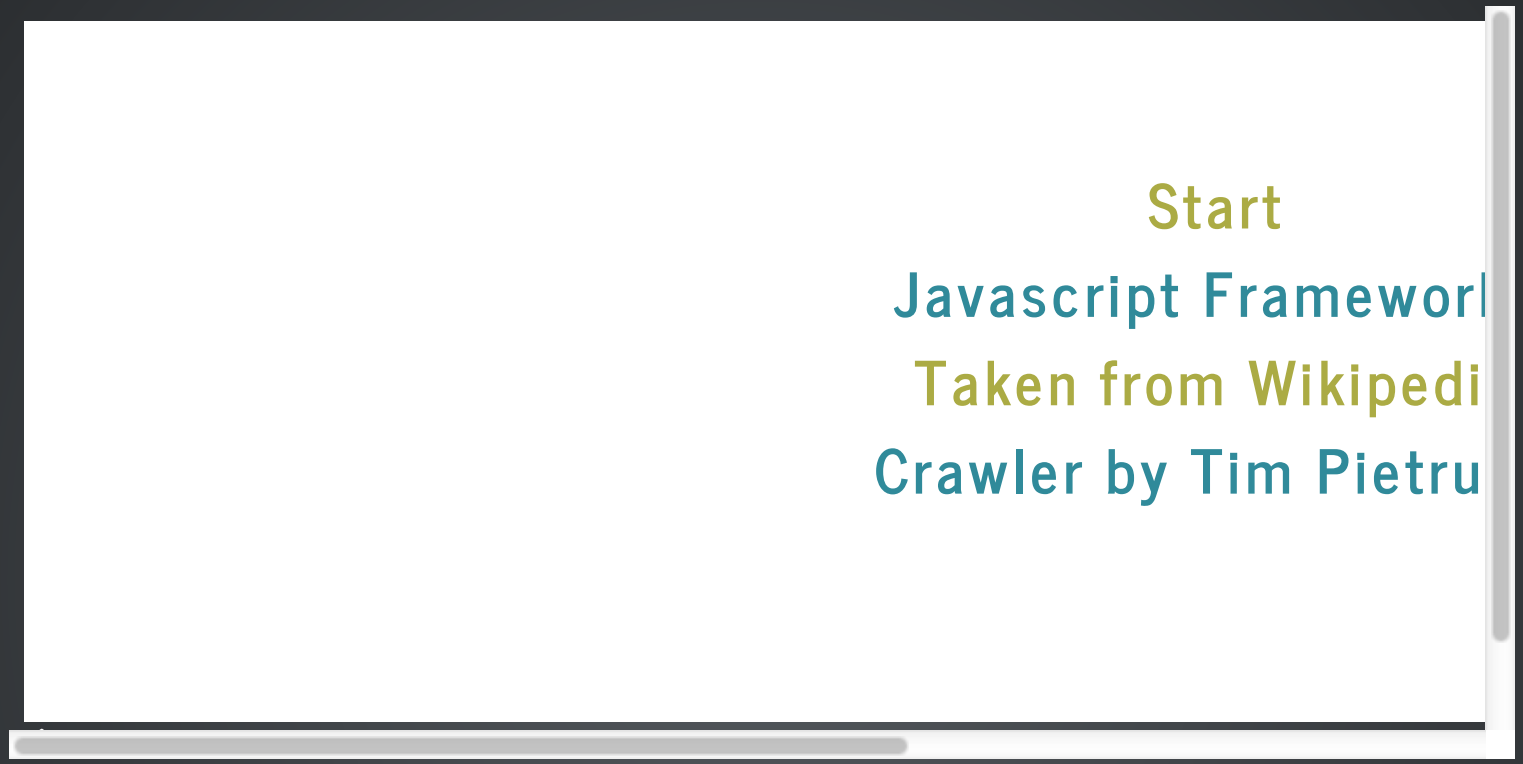
They could also result in inappropriate touching between the view and controller or spaghetti code - ultimately leading to maintainability issues.

```
<html>

  <%
    double num = Math.random();
    if (num > 0.95) {
  %>
      <h2>Welcome to the Healthcare.gov!</h2>
  <%
    } else {
  %>
      <h2>Sorry the site is busy, please try again later.</h2>
  <%
    }
  %>
  <a href="<%= request.getRequestURI() %>"><h3>Try Again</h3></a>

<html>
```

Then AJAX is invented (by IE!) and suddenly web development takes off.

Suddenly developers have the ability to deliver quick local updates to the page (like input validation) and reach back to a server for data (like table content).

There is an explosion of Javascript frameworks.



They share a similar goal of abstracting / normalizing how you interact with the browser. This allows you to write generic code that will execute across all major browsers.

Some frameworks went further and created "widget" frameworks that allow developers to be even more insulated from web development. ExtJS for example lets users build things that look eerily similar to Java Swing.

```
Ext.create('widget.window', {
                title: 'Layout Window',
                closable: true,
                width: 600,
                height: 350,
                layout: 'border',
                items: [{
                    region: 'west',
                    title: 'Navigation',
                    width: 200,
                    split: true,
                    collapsible: true,
                    floatable: false
                }, {
                    region: 'center',
                    xtype: 'tabpanel',
                    items: [{
                        title: 'Closable Tab',
                        html: 'Hello world',
                        closable: true
                    }]
                }]
            });
        }
```


COOL STORY, BRO

So if Javascript has all the tools needed to build rich internet applications, why are so many frameworks being invented?

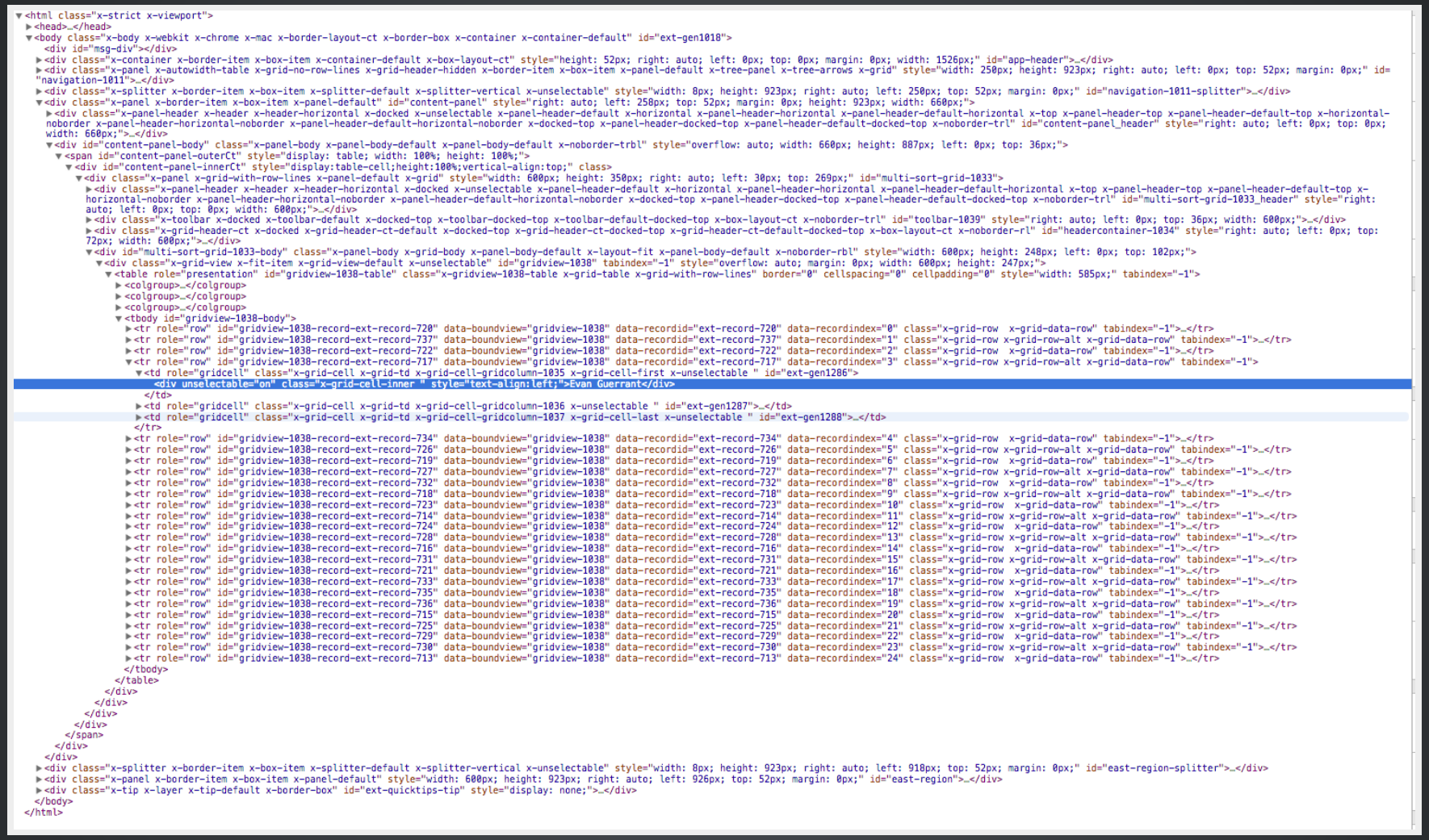
Are developers bored?

Are they looking for job security?

I think its because the display language available (HTML) lacks the ability to sufficiently express an application. We're no longer just creating lists of stuff.

```
<p>
        My favorite elements
        <ol>
                <li>Helium</li>
                <li>Oxygen</li>
                <li>Nitrogen</li>
        </ol>
</p>
```

We're creating dynamic user interfaces




ANGULARJS
by Google

AngularJS does a lot of things, however at its most basic it is a way of extending HTML's vocabulary so it can describe modern web applications.

For example, let's say you'd like to create a page of biographies for your company employees

To page divisions - t
coupling the model to

```
var printer = function(name,description){
  return "<div>"+
           "<div>"+employee.name+"</div>"+
           "<div>"+employee.description+"</div>"+
         "</div>";
};
$.ajax({
  url: "/myCompany/employees"
}).done(function(employees) {
  for(var i=0;i<employees.length;i++){
    var employee = employees[i];
     $('body').append(
             printEmployee(employee.name,employee.description)
     );
  }
});
```

We go from talking a
object called empl

This code creates an maintenance issue: since we're working with javascript strings rather than HTML its very easy to make a typo and break the view.

From a GOF standpoint we're violating separation of concerns: the code that retrieves employees shouldn't care how employees are rendered on the screen.

As a consequence of the disconnect between the model and the view developers have to constantly juggle multiple representations of their data:

- Javascript objects
- Rendered HTML fragments
- "Modified" data from user input
- Etc...

Wouldn't it make life easy if there was an "employee" HTML tag?

Even better, what if there was a "for loop" that we could just feed a list of employees to...

```
$scope.employees = $resource('/myCompany/employees');

...

<div ng:repeat="employee in employees">
        <employee information="employee"></employee>
</div>
```

- HTML
-
- CoffeeScript
- Result

Edit on

```
<html ng-app="hrApp">
  <body ng-controller="employeeListCtrl">
    <div ng:repeat="employee in employees">
      <employee employee="employee"></employee>
    </div>
  </body>
</html>

angular.module('directives', []).directive 'employee', ->
  restrict: 'E'
  replace: false
  scope:
    employee: "="
  template: "<div>{{employee.name}}</div>"
angular.module('controllers',[]).controller 'employeeListCtrl', ['$scope', '$timeout', ($scope, $timeout) ->
  $scope.employees = [
    {name: "Placeholder"}
  ]
  $timeout ->
    $scope.employees = [
```

These are known as

DIRECTIVES

and they are Angular's way of dynamically redefining and extending HTML. They form one of the 3 core ideas Angular is built upon.

THE THREE D'S IN ANGULARJS

- ____
  - markers on a DOM element that tell AngularJS to attach a specified behavior to that DOM element
- ____
  - mechanism AngularJS uses to associate dependencies with the code that needs them
- ____
  - automatic two-way synchronization of data between the model and view components

THE THREE D'S IN ANGULARJS

- ____
  - markers on a DOM element that tell AngularJS to attach a specified behavior to that DOM element
- ____
  - mechanism AngularJS uses to associate dependencies with the code that needs them
- ____
  - automatic two-way synchronization of data between the model and view components

Directives are used everywhere in AngularJS. They can take the form of either an element, an attribute, css class, or comment.

```
$scope.employees = $resource('/myCompany/empl

...

<div ng:repeat="employee in employees">
        <employee information="employee"
</div>
```

This an attribut
adds some behav
HTML

This an element dire
defines a new element
be used anywhere on

CSS AND COMMENT DIRECTIVES ARE RARELY USED:

CSS Directive

```
<div class="employee: information;"></div>
```

Comment Directive

```
<!-- directive: employee information -->
```

THE THREE D'S IN ANGULARJS

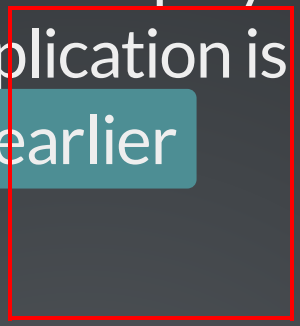
- ____
  - markers on a DOM element that tell AngularJS to attach a specified behavior to that DOM element
- ____
  - mechanism AngularJS uses to associate dependencies with the code that needs them
- ____
  - automatic two-way synchronization of data between the model and view components

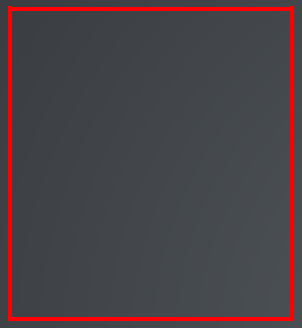
AngularJS's Dependency Injection does not have an opinion on how resources are loaded onto the page. It is only concerned with helping a consumer choose what resources to consume.

You will still need to use something like RequireJS or Browserify (or use script tags) to get your stuff on the page.

The real value of DI comes into play during testing
application. Say your application is structured li

Employee directive from earlier





A service that p
necessary data-m
the information

A provider to make the remote
connection and fetch the data

If you want to test this application you need to be able to simulate connecting to an external data source.

A traditional approach to testing this application would use something like Sinon.JS to stub any functions that make external calls

```
describe 'The employee directive', ->
  beforeEach ->
    sinon.stub $, 'ajax',(options)->
      deferred = $.Deferred()
      response = #some test input...
      deferred.resolve(JSON.parse(response))
      deferred.promise()
  afterEach ->
      $.ajax.restore()
```

THIS APPROACH HAS SOME LIMITATIONS

1. The override is global - if you're running many tests in parallel there's a chance of collision
2. You must properly clean up stubs once your test is done or risk leaking one test into another
3. Code must be modular enough to accept stubbed methods (either relying on global variables or allowing the appropriate arguments to be passed into the constructor)
4. All possible requests must be handled
5. More difficult to share common resource code across multiple tests
6. Becomes more difficult as the more transport mechanisms are introduced (a WebSocket that falls back to Long Polling, which can fall back to interval polling)

With dependency injection many of these problems can be alleviated...
we can simply provide a mock provider

WHEN THE APPLICATION IS INITIALIZED THE MOCK PROVIDER WILL BE INJECTED AUTOMATICALLY

```
describe 'The employee directive', ->
    beforeEach ->
      module ($provide)->
        $provide.provide 'provider', ->
          this.$get = ['$http', ($http)->
                #do test stuff...
          ]
```

THIS APPROACH HAS SEVERAL ADVANTAGES

1. You are not responsible for how the application is initialized in the test suite
2. The mocked module can depend on any other module in your application regardless of whether or not it is mocked
   - A mocked module that makes an external request can use a non-mocked module to process the data
3. It makes it much easier to test larger applications with deep dependency graphs
4. Can yield much higher code coverage since you tend to mock less to get a working test
5. When combined with aspects of AngularJS's test suite can shorten the feedback loop if any module is failing to properly render

How about a quick example?

VERTICAL SLIDES

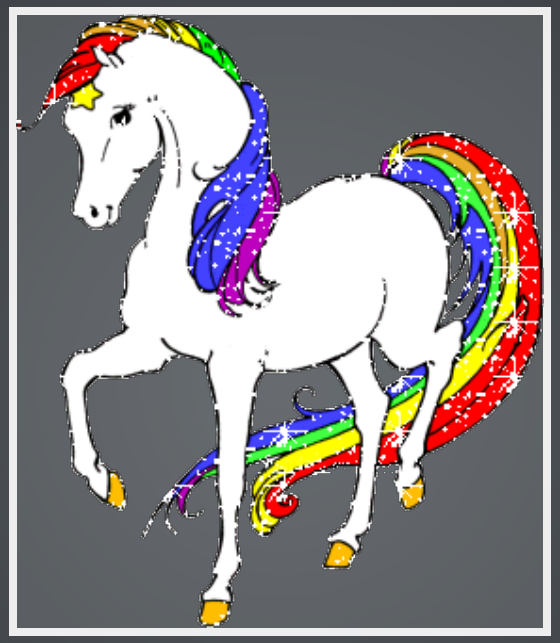
Slides can be nested inside of other slides, try pressing .

BASEMENT LEVEL 1

Press down or up to navigate.

BASEMENT LEVEL 2

Cornify



BASEMENT LEVEL 3

That's it, time to go back up.

SLIDES

Not a coder? No problem. There's a fully-featured visual editor for authoring these, try it out at .

POINT OF VIEW

Press to enter the slide overview.

Hold down alt and click on any element to zoom in on it using . Alt + click anywhere to zoom back out.

WORKS IN MOBILE SAFARI

Try it out! You can swipe through the slides and pinch your way to the overview.

MARVELOUS UNORDERED LIST

- No order here
- Or here
- Or here
- Or here

FANTASTIC ORDERED LIST

1. One is smaller than...
2. Two is smaller than...
3. Three!

MARKDOWN SUPPORT

For those of you who like that sort of thing. Instructions and a bit more info available .

```
<section data-markdown>
  ## Markdown support

  For those of you who like that sort of thing.
  Instructions and a bit more info available [here](https://github.com/hakime
</section>
```

TRANSITION STYLES

You can select from different transitions, like:

- - - - - - -

THEMES

Reveal.js comes with a few themes built in:

- - - - -

- -

* Theme demos are loaded after the presentation which leads to flicker. In production you should load your theme in the `<head>` using a `<link>`.

GLOBAL STATE

Set `data-state="something"` on a slide and `"something"` will be added as a class to the document element when the slide is open. This lets you apply broader style changes, like switching the background.

CUSTOM EVENTS

Additionally custom events can be triggered on a per slide basis by binding to the `data-state` name.

```
Reveal.addEventListener( 'customevent', function() {
        console.log( '"customevent" has fired' );
} );
```

SLIDE BACKGROUNDS

Set `data-background="#007777"` on a slide to change the full page background to the given color. All CSS color formats are supported.

IMAGE BACKGROUNDS

```
<section data-background="image.png">
```

REPEATED IMAGE BACKGROUNDS

```
<section data-background="image.png" data-background-repeat="repeat" data-ba
ckground-size="100px">
```

BACKGROUND TRANSITIONS

Pass reveal.js the `backgroundTransition: 'slide'` config argument to make backgrounds slide rather than fade.

BACKGROUND TRANSITION OVERRIDE

You can override background transitions per slide by using `data-background-transition="slide"`.

CLEVER QUOTES

These guys come in two forms, inline: *"The nice thing about standards is that there are so many to choose from"* and block:

> *"For years there has been a theory that millions of monkeys typing at random on millions of typewriters would reproduce the entire works of Shakespeare. The Internet has proven this theory to be untrue."*

PRETTY CODE

```
function linkify( selector ) {
  if( supports3DTransforms ) {

    var nodes = document.querySelectorAll( selector );

    for( var i = 0, len = nodes.length; i < len; i++ ) {
      var node = nodes[i];

      if( !node.className ) {
        node.className += ' roll';
      }
    }
  }
}
```

Courtesy of .

INTERGALACTIC INTERCONNECTIONS

You can link between slides internally, .

FRAGMENTED VIEWS

Hit the next arrow...

... to step through ...

1. `any type`
2. *of view*
3.

FRAGMENT STYLES

There's a few styles of fragments, like:

grow

shrink

roll-in

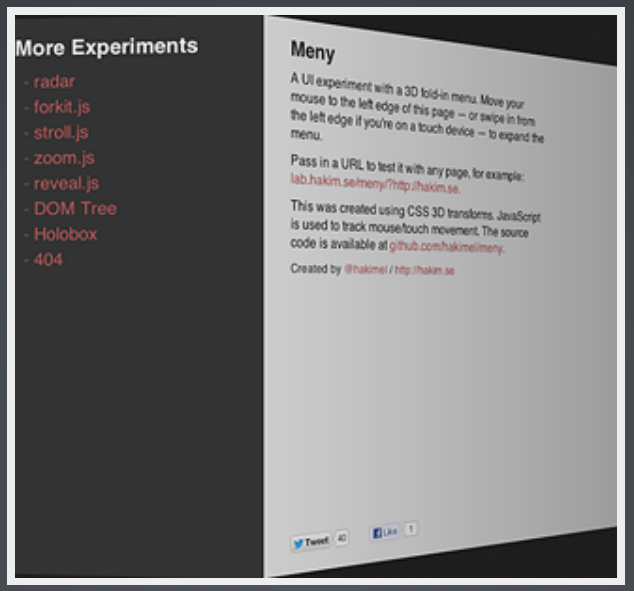
fade-out

highlight-red

highlight-green

highlight-blue

SPECTACULAR IMAGE!



EXPORT TO PDF

Presentations can be , below is an example that's been uploaded to SlideShare.

TAKE A MOMENT

Press b or period on your keyboard to enter the 'paused' mode. This mode is helpful when you want to take distracting slides off the screen during a presentation.

STELLAR LINKS

-
-
-

THE END

BY HAKIM EL HATTAB / HAKIM.SE