HackerU PRO UPDATE NEEDED

Professional training solutions in the fields of high tech and information technology

Web Training

Itay Hauptman

Bottomline

HackerU Pro is the Business Division of HackerU

HackerU Pro specializes in computer and information technology training for Hi-Tech professionals with innovative up-tp-date content by industry leading experts.

Moriya Hadad
Account Manager
052-4845189 | Moriya@hackerupro.co.il


Bootstrap 4
HackerU Pro Bootstrap 4 Day

Where do I download BootStrap?

Bootstrap 4 Starter Template

Installing Bootstrap 4 via NPM

Nodejs download page

Our first container

The starting point of your Bootstrap 4 project is almost always going to revolve around the grid container. The .container class allows you to horizontally center your layout. Alternatively, for a fluid layout (100% width), you can use .container-fluid.

```
<body>
    <div class="container">
        test
    </div>

    ...3 script tags omitted
</body>
```

If you look at your browser and get out the inspector (CTRL-SHIFT-i on Chrome), click on the element selector icon, and hover over the "test" text, you will see the outline of the container. This is a responsive container, so, you can shrink the width of your browser and observe how it reacts.

Media breakpoint

```
$bg-color: white;

body {
   background: $bg-color;
}
@import "../../node_modules/bootstrap/scss/_functions.scss";
@import "../../node_modules/bootstrap/scss/_variables.scss";
@import "../../node_modules/bootstrap/scss/mixins/_breakpoints.scss";

@include media-breakpoint-down(xs) { }
@include media-breakpoint-down(sm) { }
@include media-breakpoint-down(md) { }
@include media-breakpoint-down(lg) { }
@include media-breakpoint-down(md) {
   body {
      background: purple;
   }
 }
```

in **Src/Scss** folder update: **styles.scss**

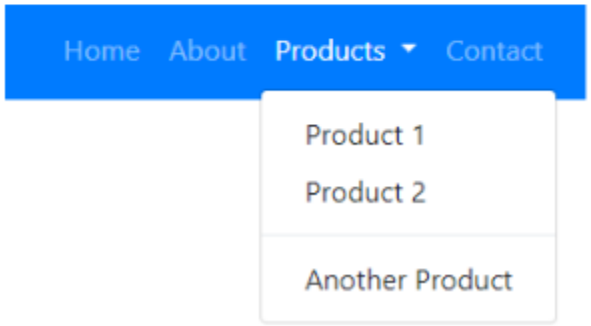
...now resize the window to get **PURPLE** background

Navbar

```
<div class="container">
  <nav class="navbar navbar-expand-lg ... choose your colors ...">
    <a class="navbar-brand" href="#">Write your CompanyName</a>

    <div class="collapse navbar-collapse" id="navbarSupportedContent">
      <ul class="navbar-nav ml-auto">
        <li class="nav-item">
          <a class="nav-link" href="#">Home</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="#">About</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="#">Products</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="#">Contact</a>
        </li>
      </ul>
    </div>
  </nav>
</div>
```

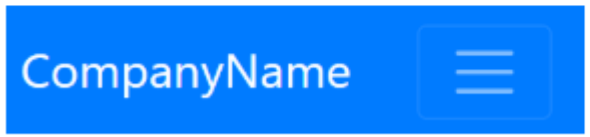
Navbar - dropdown items

```
<li class="nav-item dropdown">
    <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" data-toggle="dropdown">
      Products
    </a>
    <div class="dropdown-menu">
      <a class="dropdown-item" href="#">Product 1</a>
      <a class="dropdown-item" href="#">Product 2</a>
      <div class="dropdown-divider"></div>
      <a class="dropdown-item" href="#">Another Product</a>
    </div>
  </li>
```



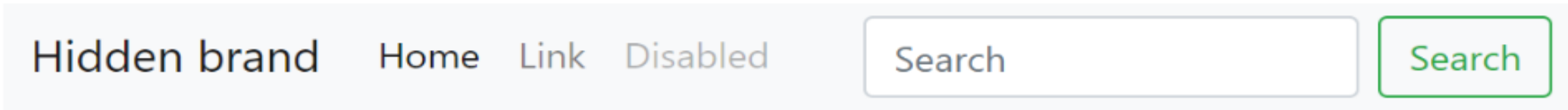
Navbar - hamburger menu

```
<a class="navbar-brand" href="#">CompanyName</a>

<button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarSupportedContent">
    <span class="navbar-toggler-icon"></span>
</button>

.....
```



Navbar - hamburger menu (full size)

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarTogglerDemo01" aria-
controls="navbarTogglerDemo01" aria-expanded="false" aria-label="Toggle navigation">
    <span class="navbar-toggler-icon"></span>
  </button>
  <div class="collapse navbar-collapse" id="navbarTogglerDemo01">
    <a class="navbar-brand" href="#">Hidden brand</a>
    <ul class="navbar-nav mr-auto mt-2 mt-lg-0">
      <li class="nav-item active">
        <a class="nav-link" href="#">Home <span class="sr-only">(current)</span></a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href="#">Link</a>
      </li>
      <li class="nav-item">
        <a class="nav-link disabled" href="#" tabindex="-1" aria-disabled="true">Disabled</a>
      </li>
    </ul>
    <form class="form-inline my-2 my-lg-0">
      <input class="form-control mr-sm-2" type="search" placeholder="Search" aria-label="Search">
      <button class="btn btn-outline-success my-2 my-sm-0" type="submit">Search</button>
    </form>
  </div>
</nav>
```



Buttons

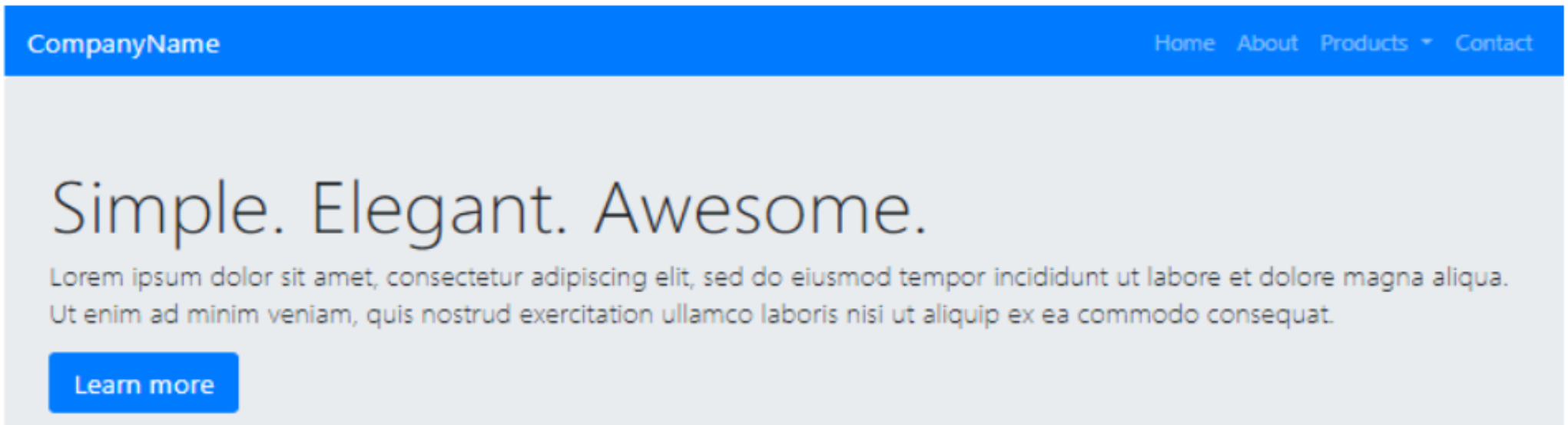
```
<button type="button" class="btn btn-primary">Primary</button>
<button type="button" class="btn btn-secondary">Secondary</button>
<button type="button" class="btn btn-success">Success</button>
<button type="button" class="btn btn-danger">Danger</button>
<button type="button" class="btn btn-warning">Warning</button>
<button type="button" class="btn btn-info">Info</button>
<button type="button" class="btn btn-light">Light</button>
<button type="button" class="btn btn-dark">Dark</button>
<button type="button" class="btn btn-link">Link</button>
```

Buttons - more...

```
<button type="button" class="btn btn-outline-primary">Primary</button>
<button type="button" class="btn btn-outline-secondary">Secondary</button>
<button type="button" class="btn btn-outline-success">Success</button>
<button type="button" class="btn btn-outline-danger">Danger</button>
<button type="button" class="btn btn-outline-warning">Warning</button>
<button type="button" class="btn btn-outline-info">Info</button>
<button type="button" class="btn btn-outline-light">Light</button>
<button type="button" class="btn btn-outline-dark">Dark</button>
```

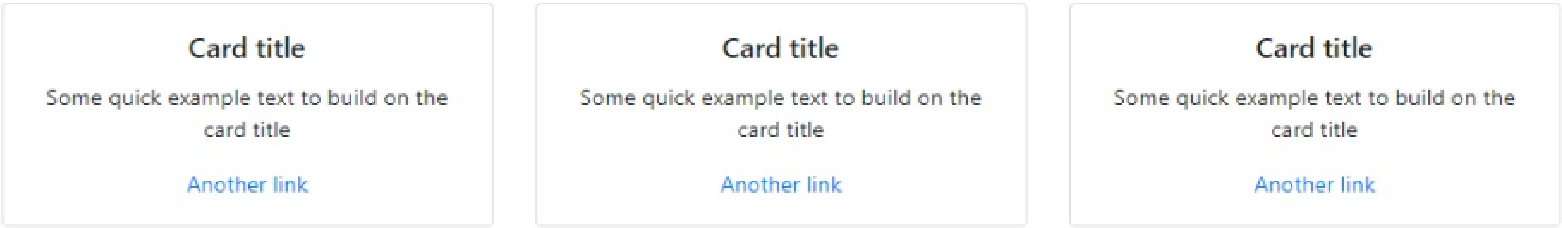
Jumbotron

```
        <div class="container">

            ... your nav bar ...

            <div class="jumbotron">
                <h1 class="display-4">Simple. Elegant. Awesome.</h1>
                <p class="lead">Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. </p>

                <p class="lead">
                  <a class="btn btn-primary btn-lg" href="#" role="button">Learn more</a>
                </p>
              </div>
          </div>
```



Cards

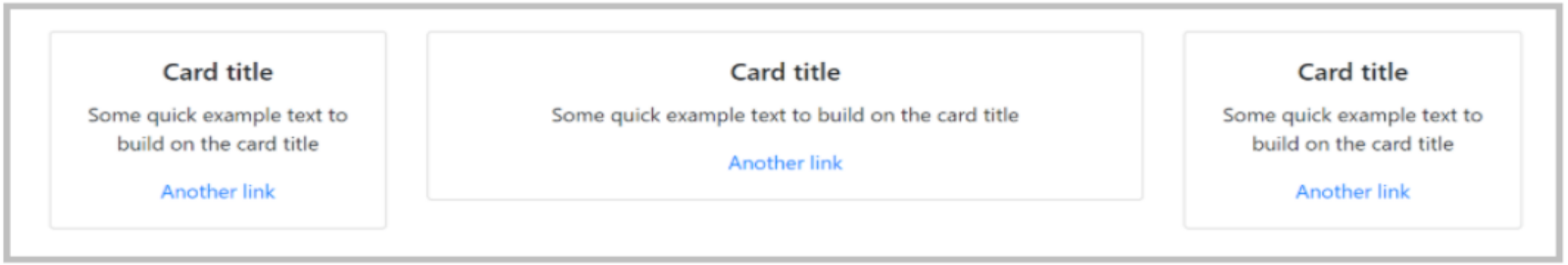
```
        <div class="row">
            <div class="col">
                <div class="card">
                    <div class="card-body text-center">
                        <h5 class="card-title">Card title</h5>
                        <p class="card-text">Some quick example text to build on the card title</p>
                        <a href="#" class="card-link">Another link</a>
                    </div>
                </div>
            </div>
            <div class="col">
                <div class="card">
                    <div class="card-body text-center">
                        <h5 class="card-title">Card title</h5>
                        <p class="card-text">Some quick example text to build on the card title</p>
                        <a href="#" class="card-link">Another link</a>
                    </div>
                </div>
            </div>
            <div class="col">
                <div class="card">
                    <div class="card-body text-center">
                        <h5 class="card-title">Card title</h5>
                        <p class="card-text">Some quick example text to build on the card title</p>
                        <a href="#" class="card-link">Another link</a>
                    </div>
                </div>
            </div>
        </div>
```



Bootstrap Grid System

```
<div class="row">
    <div class="col">
        ...
    </div>
    <div class="col-6">
        ...
    </div>
    <div class="col">
        ...
</div>
```

And the grid will automatically adjust the width of the remaining columns like so:



Bootstrap Grid System

Let's say for instance that we want our very first card in the first column to span 100% width of the viewport (assume all 12 columns) on small viewports only:

```
<div class="row">
    <div class="col-sm-12 col-md-4">
        <div class="card">
```

We first specify .col-sm-12, which means that on *small* viewports, this column will take up all 12 columns.

Then, we specify .col-md-4 which means that on *medium* viewports and larger, it will assume 4 columns

Margin and Padding

If you ever need to add either margin (*the space outside of an element*) or padding (*the space inside of an element*) you can use the margin and padding helper classes.

It works like this:

- You use **m** margin or **p** for padding
- Following *m* or *p* you add either: t (top), b (bottom), l (left), r (right), x (left and right), y (top and bottom), or nothing for all 4 sides.
- After a hyphen, you specify sizes 0 through 5 (5 being the largest amount of spacing).

So, in our example, it looks like we need to use margin and bottom to push away the cards beneath it.

On the first card container, add mb-4:

```
<div class="card mb-4">
```

Typography

Bootstrap 4 has a Typography section in their Documentation that will provide you with all of the type-based helper classes. It's fairly straightforward. We're using .lead to place emphasis on a subheading underneath our *h3* element.

There is also a Text utilities section in their Documentation that provides you with text alignment options, transform, italics and font weights.

On small viewports, let's say that we want our type to be centered instead of the default behavior, which is left aligned.

We would add the following class:

```
<div class="col-sm-12 col-md-8 text-sm-center text-md-left">
```

Cuck Here

Typography - in our page

```
            <div class="row mt-sm-4 mt-md-0">
                <div class="col-sm-12 col-md-8">
                    <h3>An important heading</h3>
                    <p class="lead">A sort of important subheading can go here, which is larger and gray.</p>

                    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.</p>
                    <p>Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.</p>
                </div>

                <div class="col-sm-12 col-md-4">
                    ..vertical navigation shortly..
                </div>
            </div>
```

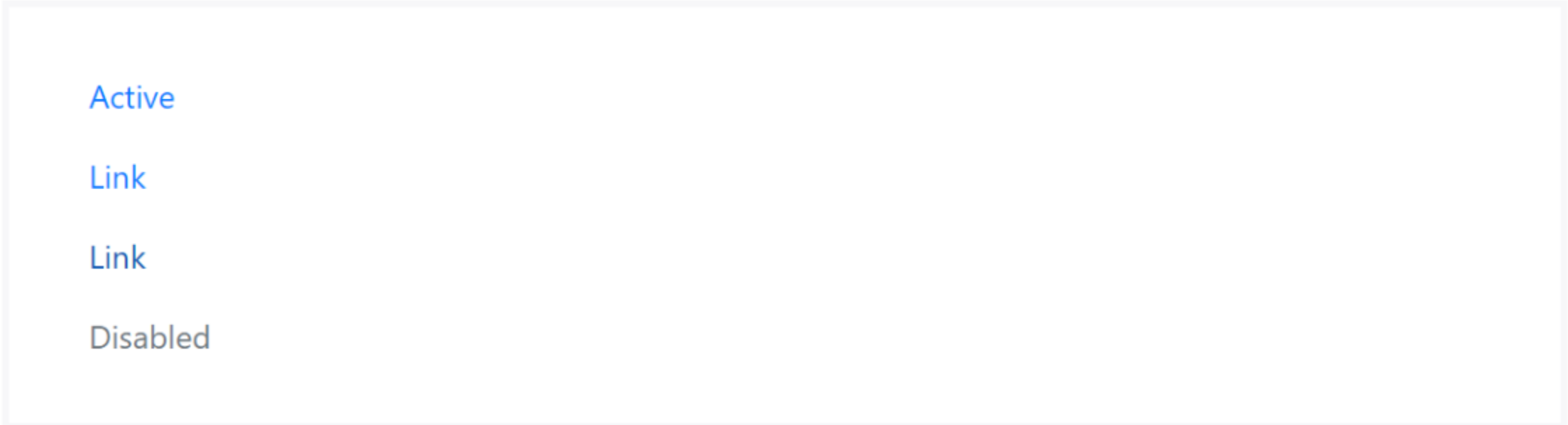
An important heading

A sort of important subheading can go here, which is larger and gray.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.

Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.

Vertical menu (tabs/ pills/ and more)




```
<nav class="nav flex-column">
  <a class="nav-link active" href="#">Active</a>
  <a class="nav-link" href="#">Link</a>
  <a class="nav-link" href="#">Link</a>
  <a class="nav-link disabled" href="#">Disabled</a>
</nav>
```

Click Here

Vertical menu - in our page

```
<h3 class="mb-4">Secondary Menu</h3>

<ul class="nav flex-column nav-pills">
  <li class="nav-item">
    <a class="nav-link active" href="#">Active</a>
  </li>
  <li class="nav-item">
    <a class="nav-link" href="#">Link</a>
  </li>
  <li class="nav-item">
    <a class="nav-link" href="#">Link</a>
  </li>
  <li class="nav-item">
    <a class="nav-link disabled" href="#">Disabled</a>
  </li>
</ul>
```

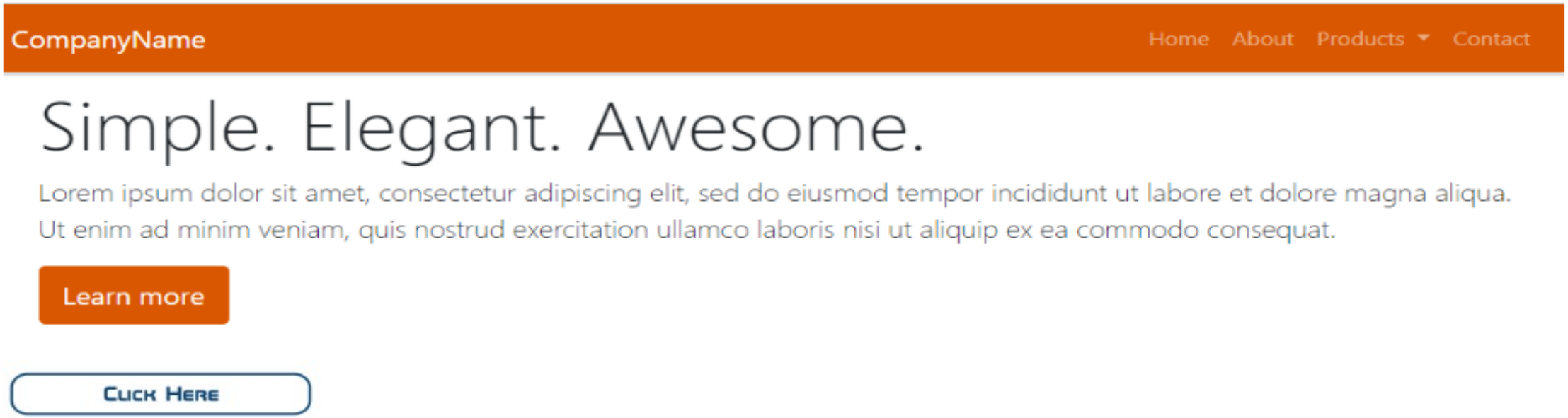
Secondary Menu

- Active
- Link
- Link
- Disabled

Changing Theme

in **Src/Scss** update: **styles.scss**

```
// Variable Overrides
$theme-colors: (
  "primary": #d95700
);
$body-bg: #ededed;

.jumbotron {
    background-color: #ffffff !important;
    border-top: 3px solid rgb(219, 219, 219);
}

@import "../../node_modules/bootstrap/scss/bootstrap";
```



Live css editor

chrome web store

Home > Extensions > Live CSS Editor

Live CSS Editor

Offered by: www.livecsseditor.com

★★★★☆ 207 | Developer Tools | 41,952 users

Add to Chrome

Overview

Compatible with your device

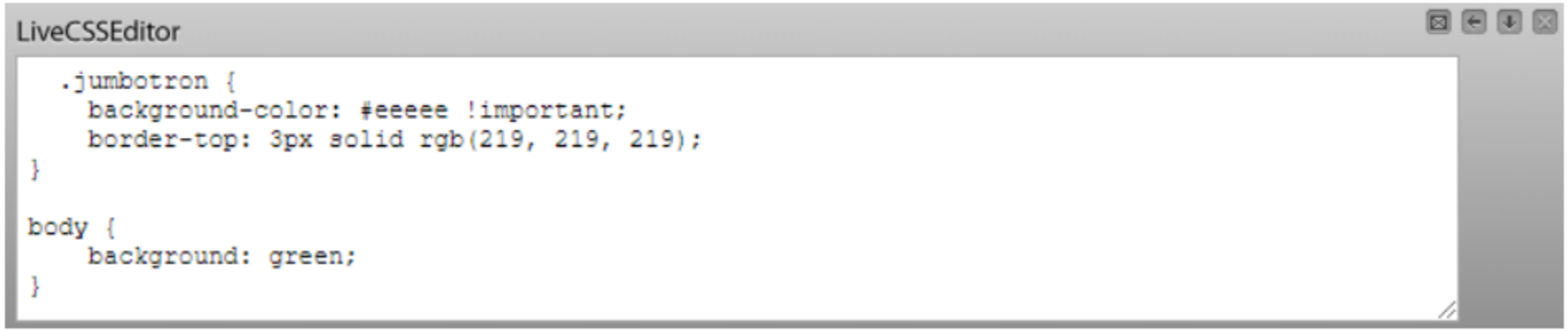
Live Write CSS onto any page

This extension provides a text box on any HTML page so that you can write CSS freely and immediately see the results. I use it every day for quick edits, product demos and other testing out new ideas.

The extension options allow you to set a default key command to open and close the editor; turn off the warning that is shown when you close the editor; and enable or disable the automatic saving of your changes when you close the editor.

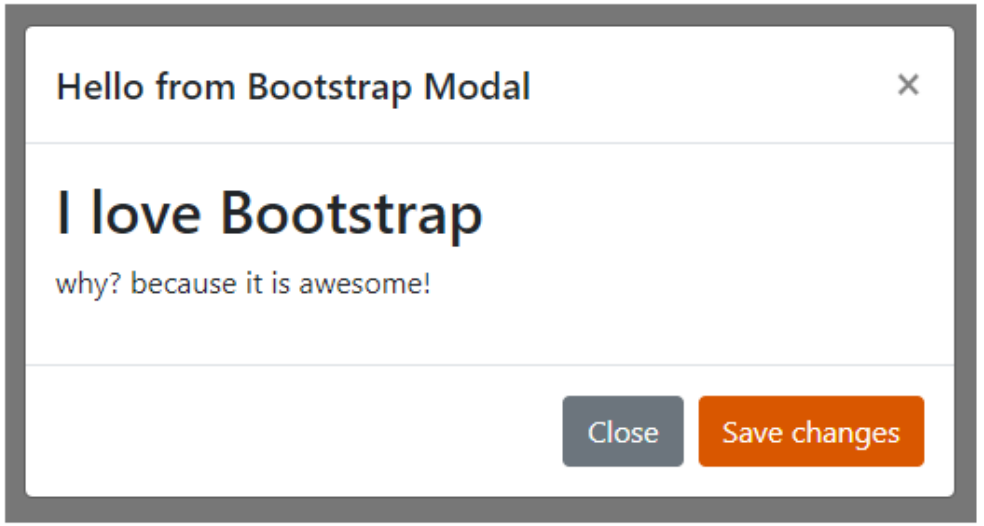
Live css editor

```
.jumbotron {
    background-color: #eeeee !important;
    border-top: 3px solid rgb(219, 219, 219);
}

body {
    background: green;
}
```



Modal messages

```
<!-- Button trigger modal -->
<button type="button" class="btn btn-primary" data-toggle="modal"
data-target="#myModal">
      Launch demo modal
     </button>
     <!-- Modal -->
     <div class="modal fade" id="myModal" tabindex="-1" role="dialog"
     arialabelledby="exampleModalLabel"
       aria-hidden="true">
      <div class="modal-dialog" role="document">
      <div class="modal-content">
      <div class="modal-header">
      <h5 class="modal-title" id="exampleModalLabel">Hello from Bootstrap Modal</h5>
      <button type="button" class="close" data-dismiss="modal" aria-label="Close">
      <span aria-hidden="true">×</span>
      </button>
      </div>
      <div class="modal-body">
      <h2>I love Bootstrap</h2>
      <p>why? because it is awesome!</p>
      </div>
      <div class="modal-footer">
      <button type="button" class="btn btn-secondary" data-dismiss="modal">Close</
     button>
      <button type="button" class="btn btn-primary">Save changes</button>
      </div> </div></div></div>
```



Forms Layout

Bootstrap provides you with following types of form layouts:

- Vertical (default) form
- Inline form
- Horizontal form

BASIC FORM

The basic form structure comes with Bootstrap; individual form controls automatically receive some global styling.
To create a basic form do the following:

- Add a role *form* to the parent <form> element.
- Wrap labels and controls in a with class *.form-group*. This is needed for optimum spacing.
- Add a class of *.form-control* to all textual <input>, <textarea>, and <select> elements.

Vertical Forms

```
<form style="margin-left: 20px">
    <div class="form-group">
      <label for="inputUserName">Username</label>
      <input class="form-control" placeholder="Login Username"
          type="text" id="inputUserName" />
    </div>
    <div class="form-group">
      <label for="inputPassword">Password</label>
      <input class="form-control" placeholder="Login Password"
          type="password" id="inputPassword" />
    </div>
    <button type="submit" class="btn btn-default">Login</button>
  </form>
```

Username

Login Username

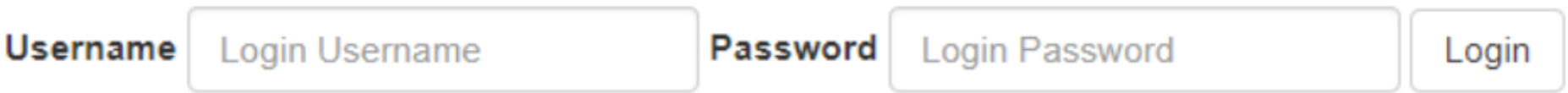
Password

Login Password

Login

Inline Forms

```
<form style="margin-left: 20px" class="form-inline">
    <div class="form-group">
      <label for="inputUserName">Username</label>
      <input class="form-control" placeholder="Login Username"
          type="text" id="inputUserName" />
    </div>
    <div class="form-group">
      <label for="inputPassword">Password</label>
      <input class="form-control" placeholder="Login Password"
          type="password" id="inputPassword" />
    </div>
    <button type="submit" class="btn btn-default">Login</button>
  </form>
```



Horizontal Forms

```
<form class="form-horizontal" style="margin-left: 20px">
    <div class="form-group">
      <label for="inputUserName" class="control-label col-sm-2">Username</label>
      <div class="col-sm-6">
        <input class="form-control" placeholder="Login Username"
            type="text" id="inputUserName" />
      </div>
    </div>
    <div class="form-group">
      <label for="inputPassword" class="control-label col-sm-2">Password</label>
      <div class="col-sm-6">
        <input class="form-control" placeholder="Login Password"
            type="password" id="inputPassword" />
      </div>
    </div>
    <div class="form-group">
      <div class="col-sm-offset-2 col-sm-10">
        <button type="submit" class="btn btn-default">Login</button>
      </div>
    </div>
  </form>
```

Username Login Username

Password Login Password

Login

Form validation with JS

```
<form class="needs-validation" novalidate>
      <div class="form-row">
       <div class="col-md-4 mb-3">
        <label for="validationCustom01">First name</label>
        <input type="text" class="form-control" id="validationCustom01" placeholder="First name" value="Mark" required>
        <div class="valid-feedback">
         Looks good!
        </div>
       </div>
       <div class="col-md-4 mb-3">
        <label for="validationCustom02">Last name</label>
        <input type="text" class="form-control" id="validationCustom02" placeholder="Last name" value="Otto" required>
        <div class="valid-feedback">
         Looks good!
        </div>
       </div>
       <div class="col-md-4 mb-3">
        <label for="validationCustomUsername">Username</label>
        <div class="input-group">
         <div class="input-group-prepend">
          <span class="input-group-text" id="inputGroupPrepend">@</span>
         </div>
```

Last name

Hau

Looks good!

Username

@ Username

Please choose a username.

Tables striped

```
<table class="table table-striped">
  <thead>
    <tr>
      <th scope="col">#</th>
      <th scope="col">First</th>
      <th scope="col">Last</th>
      <th scope="col">Handle</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <th scope="row">1</th>
      <td>Mark</td>
      <td>Otto</td>
      <td>@mdo</td>
    </tr>
    <tr>
      <th scope="row">2</th>
      <td>Jacob</td>
      <td>Thornton</td>
      <td>@fat</td>
    </tr>
    <tr>
      <th scope="row">3</th>
      <td>Larry</td>
      <td>the Bird</td>
      <td>@twitter</td>
    </tr>
  </tbody>
</table>
```

#	First	Last	Handle
1	Mark	Otto	@mdo
2	Jacob	Thornton	@fat
3	Larry	the Bird	@twitter

Font Awesome

```
<button type="button" class="btn btn-default btn-lg">
  <i class="fa fa-star"></i> Star
</button>
<div class="btn-toolbar" role="toolbar">
  <div class="btn-group">
    <button type="button" class="btn btn-default">
      <i class="fa fa-align-left"></i>
    </button>
    <button type="button" class="btn btn-default">
      <i class="fa fa-align-center"></i>
    </button>
    <button type="button" class="btn btn-default">
      <i class="fa fa-align-right"></i>
    </button>
    <button type="button" class="btn btn-default">
      <i class="fa fa-align-justify"></i>
    </button>
  </div>
</div>
```

Home Site!

★ Star

Email address

Password

Block Quote

```
<blockquote class="blockquote">
    <p class="mb-0">Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer posuere erat a ante.</p>
</blockquote>
```

Blockquotes

For quoting blocks of content from another source within your document. Wrap `<blockquote class="blockquote">` around any HTML as the quote.

> Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer posuere erat a ante.

Footer

```
<blockquote class="blockquote">
  <p class="mb-0">Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer posuere erat a ante.</p>
  <footer class="blockquote-footer">Someone famous in <cite title="Source Title">Source Title</cite></footer>
</blockquote>
```

Add a `<footer class="blockquote-footer">` for identifying the source. Wrap the name of the source work in `<cite>`.

> Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer posuere erat a ante.
> — Someone famous in *Source Title*

Inline text elements style (for HTML5)

```
<p>You can use the mark tag to <mark>highlight</mark> text.</p>
<p><del>This line of text is meant to be treated as deleted text.</del></p>
<p><s>This line of text is meant to be treated as no longer accurate.</s></p>
<p><ins>This line of text is meant to be treated as an addition to the document.</ins></p>
<p><u>This line of text will render as underlined</u></p>
<p><small>This line of text is meant to be treated as fine print.</small></p>
<p><strong>This line rendered as bold text.</strong></p>
<p><em>This line rendered as italicized text.</em></p>
```

You can use the mark tag to highlight text.

~~This line of text is meant to be treated as deleted text.~~

~~This line of text is meant to be treated as no longer accurate.~~

This line of text is meant to be treated as an addition to the document.

This line of text will render as underlined

<small>This line of text is meant to be treated as fine print.</small>

This line rendered as bold text.

This line rendered as italicized text.

Scrollspy

```
<!DOCTYPE html>
<html>
<head>
 <title>Bootstrap Example</title>
 <meta charset="utf-8">
 <meta name="viewport" content="width=device-width, initial-scale=1">
 <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/css/bootstrap.min.css">
 <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
 <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/js/bootstrap.min.js"></script>
 <style>
 body {
    position: relative;
 }
 #section1 {padding-top:50px;height:500px;color: #fff; background-color: #1E88E5;}
 #section2 {padding-top:50px;height:500px;color: #fff; background-color: #673ab7;}
 #section3 {padding-top:50px;height:500px;color: #fff; background-color: #ff9800;}
 #section41 {padding-top:50px;height:500px;color: #fff; background-color: #00bcd4;}
 #section42 {padding-top:50px;height:500px;color: #fff; background-color: #009688;}
 </style>
</head>
```

Automatically update Bootstrap navigation or list group components based on scroll position to indicate which link is currently active in the viewport.

Navbar

- Item 1
  - Item 1-1
  - Item 1-2
- Item 2
- Item 3
  - Item 3-1
  - Item 3-2

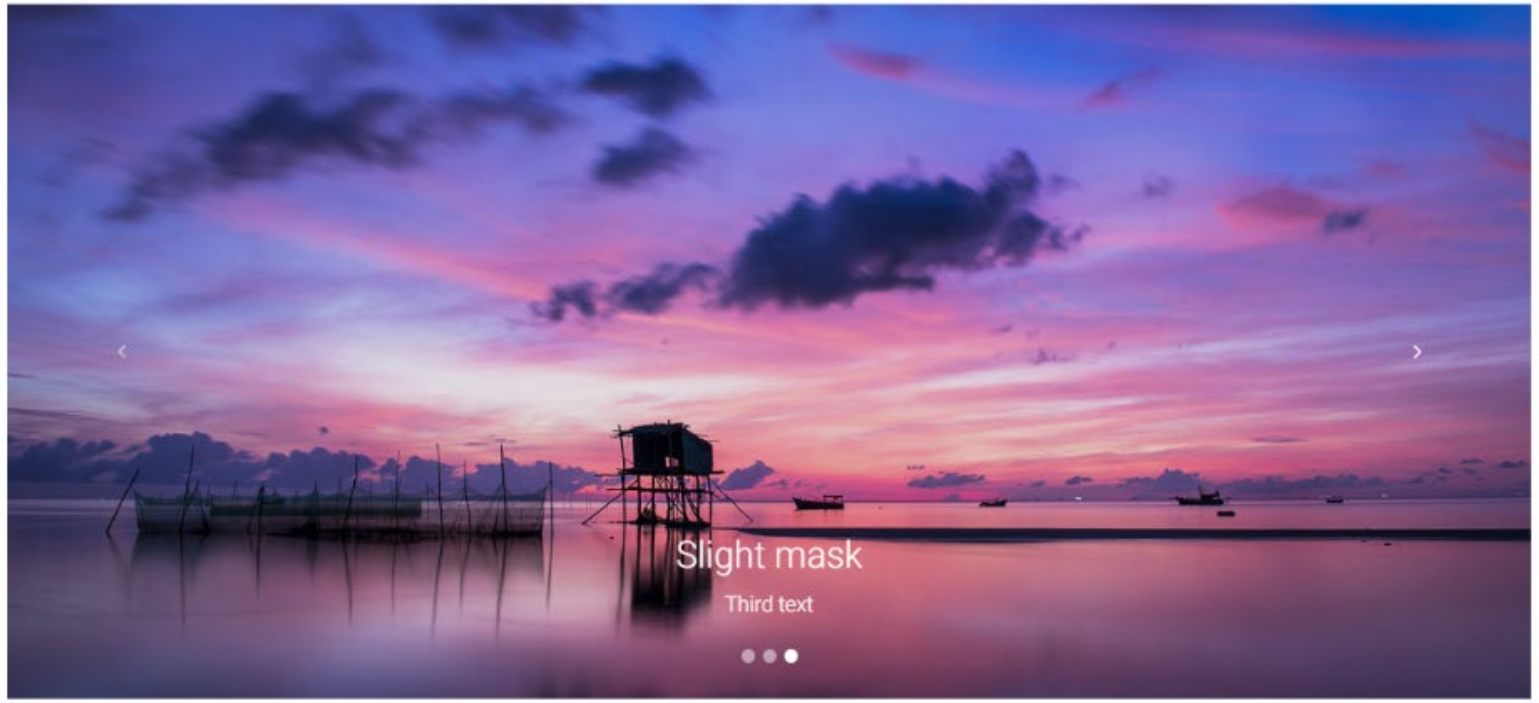
dolore anim exercitation aute fugiat labore voluptate cillum do laboris labore. Ex velit exercitation nisi enim labore reprehenderit labore nostrud ut ut. Esse officia sunt duis aliquip ullamco tempor eiusmod deserunt irure nostrud irure. Ullamco proident veniam laboris ea consectetur magna sunt ex exercitation aliquip minim enim culpa occaecat exercitation. Est tempor excepteur aliquip laborum consequat do deserunt laborum esse eiusmod irure proident ipsum esse qui.

Item 3-2

Labore sit culpa commodo elit adipisicing sit aliquip elit proident voluptate minim mollit nostrud aute reprehenderit do. Mollit excepteur eu Lorem ipsum anim commodo sint labore Lorem in exercitation velit incididunt. Occaecat consectetur nisi in occaecat proident minim enim sunt reprehenderit exercitation cupidatat et do officia. Aliquip consequat ad labore labore mollit ut amet. Sit pariatur tempor proident in veniam culpa aliqua excepteur elit magna fugiat eiusmod amet officia.

Carousel

```
        <!--Carousel Wrapper-->
<div id="carousel-example-2" class="carousel slide carousel-fade" data-ride="carousel">
    <!--Indicators-->
    <ol class="carousel-indicators">
      <li data-target="#carousel-example-2" data-slide-to="0" class="active"></li>
      <li data-target="#carousel-example-2" data-slide-to="1"></li>
      <li data-target="#carousel-example-2" data-slide-to="2"></li>
    </ol>
    <!--/.Indicators-->
    <!--Slides-->
    <div class="carousel-inner" role="listbox">
      <div class="carousel-item active">
        <div class="view">
          <img class="d-block w-100" src="https://mdbootstrap.com/img/Photos/Slides/img%20(68).jpg" alt="First slide">
          <div class="mask rgba-black-light"></div>
        </div>
        <div class="carousel-caption">
          <h3 class="h3-responsive">Light mask</h3>
          <p>First text</p>
        </div>
```



Gradient colors

styles.scss:

```
$enable-gradients: true;
```

HTML:

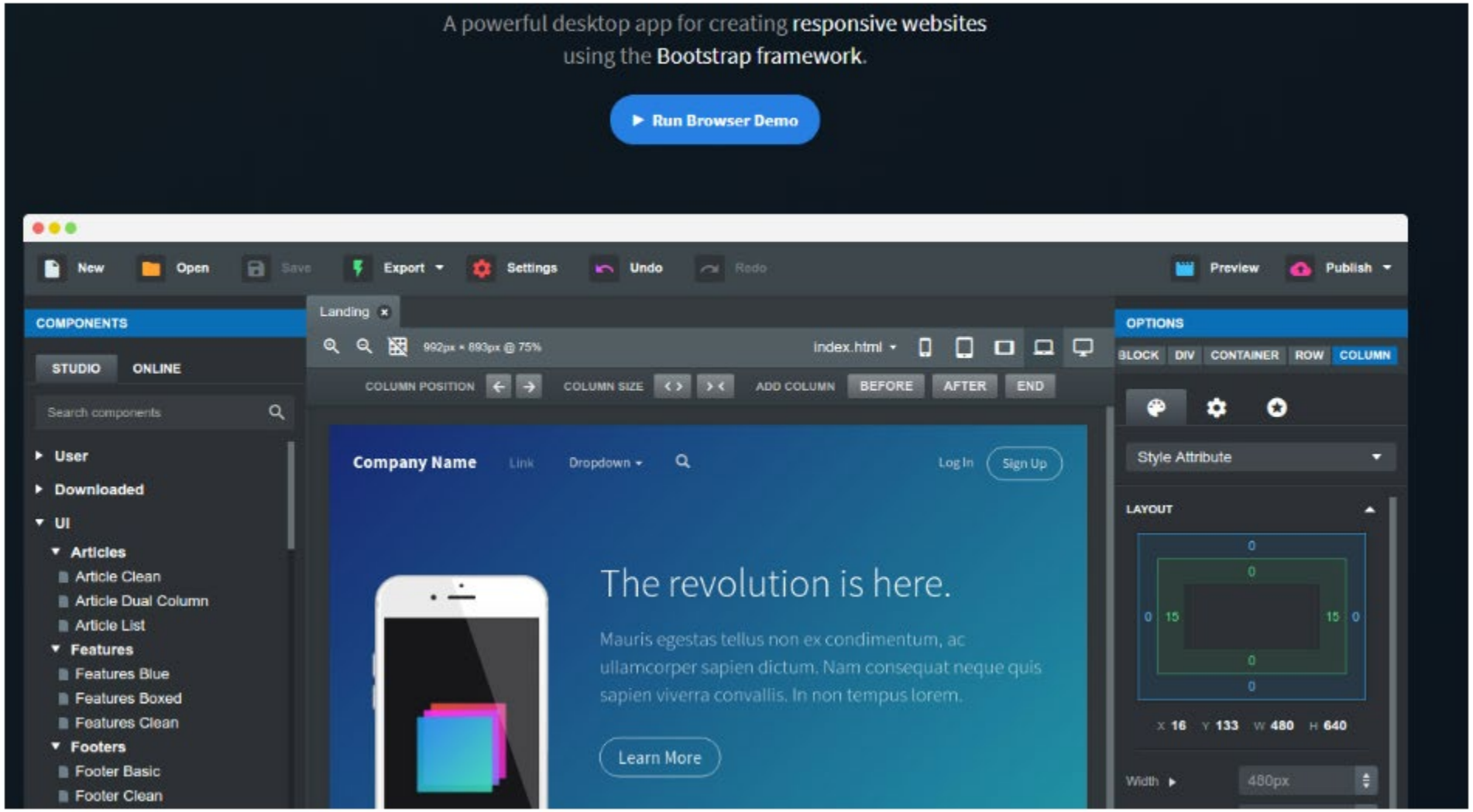
```
<div class="p-3 mb-2 bg-gradient-primary text-white">.bg-gradient-primary</div>
<div class="p-3 mb-2 bg-gradient-secondary text-white">.bg-gradient-secondary</div>
<div class="p-3 mb-2 bg-gradient-success text-white">.bg-gradient-success</div>
<div class="p-3 mb-2 bg-gradient-danger text-white">.bg-gradient-danger</div>
<div class="p-3 mb-2 bg-gradient-warning text-dark">.bg-gradient-warning</div>
<div class="p-3 mb-2 bg-gradient-info text-white">.bg-gradient-info</div>
<div class="p-3 mb-2 bg-gradient-light text-dark">.bg-gradient-light</div>
<div class="p-3 mb-2 bg-gradient-dark text-white">.bg-gradient-dark</div>
```

More global options

You can find and customize these variables for key global options in our `_variables.scss` file.

Variable	Values	Description
`$spacer`	`1rem` (default), or any value > 0	Specifies the default spacer value for our spacer utilities.
`$enable-rounded`	`true` (default) or `false`	Enables predefined `border-radius` styles on various components.
`$enable-shadows`	`true` or `false` (default)	Enables predefined `box-shadow` styles on various components.
`$enable-gradients`	`true` or `false` (default)	Enables predefined gradients via `background-image` styles on various components.
`$enable-transitions`	`true` (default) or `false`	Enables predefined `transition`s on various components.
`$enable-hover-media-query`	`true` or `false` (default)	...
`$enable-grid-classes`	`true` (default) or `false`	Enables the generation of CSS classes for the grid system (e.g., `.container`, `.row`, `.col-md-1`, etc.).
`$enable-print-styles`	`true` (default) or `false`	Enables styles for optimizing printing.

Bootstrap Studio



Summary

Breakpoints

Breakpoints

Extra small < 544px

Small ≥ 544px

Medium ≥ 768px

Large ≥ 992px

Extra large ≥ 1200px

Typography

.text-left Left aligned text

.text-center Center aligned text

.text-right Right aligned text

.text-justify Justified text

.text-nowrap No wrap text

.text-lowercause Lowercase text

.text-uppercase Uppercase text

.text-capitalize Capitalized text

.lead Good for first paragraph of article

Blockquote

```
<blockquote class="blockquote">
<p class="m-b-0">Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer posuere erat a ante.</p>
</blockquote>
```

Headings

```
<h1>h1. Bootstrap heading <small>Secondary text</small></h1>
<p class="h1">Paragraph that looks like heading</p>
```

Colors

.text-muted

.text-primary

.text-success

.text-info

.text-warning

.text-danger

.bg-primary

.bg-success

.bg-info

.bg-warning

.bg-danger

.bg-inverse

.bg-faded

Lists

.list-unstyled Removes default list margin

.list-inline Makes list items inline

.dl-horizontal Makes list items two columns

Images

.img-fluid Make an image responsive

.img-rounded Adds rounded corners to image

.img-circle Crops image to be circle

.img-thumbnail Adds rounded corner + border

Floats

.pull-left Floats item left

.pull-right Floats item right

.center-block Set an elemennt to block with auto left and right margin

.clearfix Clear floats by adding this class to the parent container

Starter Template

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

    <!-- Bootstrap CSS -->
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-alpha.6/
css/bootstrap.min.css">

    <!-- Main CSS -->
    <link rel="stylesheet" href="css/main.css">

  </head>
  <body>
    <div class="container">
      <h1>Hello, world!</h1>
      <div class="row">
            <div class="col-sm-6">Left Column</div>
            <div class="col-sm-6">Right Column</div>
      </div>
    </div>

    <!-- jQuery first, then Tether, then Bootstrap JS. -->
    <script src="https://code.jquery.com/jquery-3.1.1.slim.min.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/tether/1.4.0/js/tether.min.js"></
script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-alpha.6/js/bootstrap.min.
js"></script>
    <!-- Main JS -->
    <script src="js/main.js"></script>
  </body>
```

One Column Centered Grid

```
<div class="container">
   <div class="row justify-content-center">
      <div class="col-md-6"></div>
   </div>
</div>
```

Two Column Grid

```
<div class="container">
   <div class="row">
      <div class="col-sm-6"></div>
      <div class="col-sm-6"></div>
   </div>
</div>
```

Three Column Grid

```
<div class="container">
   <div class="row">
      <div class="col-sm-4"></div>
      <div class="col-sm-4"></div>
      <div class="col-sm-4"></div>
   </div>
</div>
```

Four Column Grid

```
<div class="container">
   <div class="row">
      <div class="col-sm-3"></div>
      <div class="col-sm-3"></div>
      <div class="col-sm-3"></div>
      <div class="col-sm-3"></div>
   </div>
</div>
```

Navbar

```
<nav class="navbar navbar-toggleable-md bg-faded navbar-inverse bg-primary">
  <button class="navbar-toggler navbar-toggler-right" type="button" data-toggle="collapse"
data-target="#navbarNavDropdown" aria-controls="navbarNavDropdown" aria-expanded="false"
aria-label="Toggle navigation">
    <span class="navbar-toggler-icon"></span>
  </button>
  <a class="navbar-brand" href="#">Navbar</a>
  <div class="collapse navbar-collapse" id="navbarNavDropdown">
    <ul class="navbar-nav">
      <li class="nav-item active">
        <a class="nav-link" href="#">Home <span class="sr-only">(current)</span></a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href="#">Features</a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href="#">Pricing</a>
      </li>
      <li class="nav-item dropdown">
        <a class="nav-link dropdown-toggle" href="http://example.com"
id="navbarDropdownMenuLink" data-toggle="dropdown" aria-haspopup="true" aria-
expanded="false">
          Dropdown link
        </a>
        <div class="dropdown-menu" aria-labelledby="navbarDropdownMenuLink">
          <a class="dropdown-item" href="#">Action</a>
          <a class="dropdown-item" href="#">Another action</a>
          <a class="dropdown-item" href="#">Something else here</a>
        </div>
      </li>
    </ul>
  </div>
</nav>
```

Modal

```
<!-- Button trigger modal -->
<button type="button" class="btn btn-primary" data-toggle="modal" data-target="#myModal">
  Launch demo modal
</button>

<!-- Modal -->
<div class="modal fade" id="myModal" tabindex="-1" role="dialog" aria-
labelledby="exampleModalLabel" aria-hidden="true">
  <div class="modal-dialog" role="document">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title" id="exampleModalLabel">Modal title</h5>
        <button type="button" class="close" data-dismiss="modal" aria-label="Close">
          <span aria-hidden="true">&times;</span>
        </button>
      </div>
      <div class="modal-body">
        <h2>Modal body heading</h2>
        <p>Modal body text description</p>
      </div>
      <div class="modal-footer">
        <button type="button" class="btn btn-secondary" data-dismiss="modal">Close</
button>
        <button type="button" class="btn btn-primary">Save changes</button>
      </div>
    </div>
  </div>
</div>
```

Forms

```
<form>
  <div class="form-group">
    <label for="exampleInputEmail1">Email address</label>
    <input type="email" class="form-control" id="exampleInputEmail1" aria-
describedby="emailHelp" placeholder="Enter email">
    <small id="emailHelp" class="form-text text-muted">We'll never share your email with
anyone else.</small>
  </div>
  <div class="form-group">
    <label for="exampleInputPassword1">Password</label>
    <input type="password" class="form-control" id="exampleInputPassword1"
placeholder="Password">
  </div>
  <div class="checkbox">
    <label>
      <input type="checkbox"> Check me out
    </label>
  </div>
  <button type="submit" class="btn btn-primary">Submit</button>
</form>
```

Buttons

.btn Needs to be added to all buttons because it adds padding and margin
.btn-default The default button style
.btn-primary The button that has the primary action in a group
.btn-success Could be used on the last submit button in a form
.btn-info Informational button
.btn-link Removes background color and add text color
.btn-lg Large buttom
.btn-sm Smaller than default button
.btn-xs Even smaller
.btn-block Button that spans full width of parent

```
<a class="btn btn-default"
href="#" role="button">Link</a>
<button class="btn btn-primary"
type="submit">Button</button>
```

Carousel

```
<div id="carousel-name" class="carousel slide" data-ride="carousel">
  <ol class="carousel-indicators">
    <li data-target="#carousel-name" data-slide-to="0" class="active"></li>
    <li data-target="#carousel-name" data-slide-to="1"></li>
    <li data-target="#carousel-name" data-slide-to="2"></li>
  </ol>
  <div class="carousel-inner" role="listbox">
    <div class="carousel-item active">
      <img class="d-block img-fluid" src="https://dummyimage.com/900x340/563d7c/
fff&text=+" alt="First slide">
        <div class="carousel-caption d-none d-md-block">
          <h3>Carousel Slider Title</h3>
          <p>Description text to further describe the content of the slide image</p>
          <a href="" class="btn btn-primary">Call to Action</a>
        </div>
    </div>
    <div class="carousel-item">
      <img class="d-block img-fluid" src="https://dummyimage.com/900x340/563d7c/
fff&text=+" alt="Third slide">
        <div class="carousel-caption d-none d-md-block">
          <a href="" class="btn btn-primary">Call to Action</a>
        </div>
    </div>
  </div>
  <a class="carousel-control-prev" href="#carousel-name" role="button" data-slide="prev">
    <span class="carousel-control-prev-icon" aria-hidden="true"></span>
    <span class="sr-only">Previous</span>
  </a>
  <a class="carousel-control-next" href="#carousel-name" role="button" data-slide="next">
    <span class="carousel-control-next-icon" aria-hidden="true"></span>
    <span class="sr-only">Next</span>
  </a>
</div>
```

Jumbotron

```
<div class="jumbotron jumbotron-fluid">
  <div class="container">
    <h1 class="display-3">Fluid jumbotron</h1>
    <p class="lead">This is a modified jumbotron that occupies the entire horizontal space of its parent.</p>
  </div>
</div>
```

Card

```
<div class="card" style="width: 20rem;">
  <img class="card-img-top w-100" src="https://dummyimage.com/600x400/563d7c/fff" alt="Card image cap" >
  <div class="card-block">
    <h4 class="card-title">Card title</h4>
    <p class="card-text">Some quick example text to build on the card title and make up the bulk of the card's content.</p>
    <a href="#" class="btn btn-primary">Go somewhere</a>
  </div>
</div>
```

Breadcrumbs

```
<ol class="breadcrumb">
  <li><a href="#">Home</a></li>
  <li><a href="#">Library</a></li>
  <li class="active">Data</li>
</ol>
```

Responsive embed

```
<div class="embed-responsive embed-responsive-16by9">
  <iframe class="embed-responsive-item" src="..."></iframe>
</div>
```

Tables

```
<table class="table">
  <thead class="thead-default">
    <tr>
      <th>#</th>
      <th>thead-default</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <th scope="row">1</th>
      <td>Nina</td>
    </tr>
  </tbody>
</table>

<table class="table">
  <thead class="thead-inverse">
    <tr>
      <th>#</th>
      <th>thead-inverse</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <th scope="row">1</th>
      <td>Nina</td>
    </tr>
  </tbody>
</table>
```

Component CSS Format and Media Queries

```
/*
 * Component section heading
 *
 * Component description and use
 */

/* base - shared styles */
.component { width: 220px; }

/* Sub-component with component name as a prefix to isolate styles and break
the cascade. */

.component-heading {
  display: block;
  width: 100px;
  font-size: 1rem;
}

/* variant - alert color */
.component-alert {
  color: #ff0000;
}

/* variant - success color */
.component-success {
  color: #00ff00;
}

/* Add media queries below components instead of a separate stylesheet or
section to make updating easier */

@media (min-width: 480px) {
  .component-heading { width:auto; }
}
```

Convert pixels to REMS

Pixels	REMS	Pixels	REMS
1 px	0.0625	26	1.625
2 px	0.125	27	1.6875
3 px	0.1875	28	1.75
4 px	0.25	29	1.8125
5 px	0.3125	30	1.875
6 px	0.375	31	1.9375
7 px	0.4375	32	2
8 px	0.5	33	2.0625
9 px	0.5625	34	2.125
10 px	0.625	35	2.1875
11 px	0.6875	36	2.25
12 px	0.75	37	2.3125
13 px	0.8125	38	2.375
14 px	0.875	39	2.4375
15 px	0.9375	40	2.5
Default Bootstrap 4 font size 16 px	1	41	2.5625
17 px	1.0625	42	2.625
18 px	1.125	43	2.6875
19 px	1.1875	44	2.75
20 px	1.25	45	2.8125
21 px	1.3125	46	2.875
22 px	1.375	47	2.9375
23 px	1.4375	48	3
24 px	1.5	49	3.0625
25 px	1.5625	50	3.125

Multiples of common unites of measure

Multiples of 15		Multiples of 30	
15	405	30	810
30	420	60	840
45	435	90	870
60	450	120	900
75	465	150	930
90	480	180	960
105	495	210	990
120	510	240	1020
135	525	270	1050
150	540	300	1080
165	555	330	1110
180	570	360	1140
195	585	390	1170
210	600	420	1200
225	615	450	1230
240	630	480	1260
255	645	510	1290
270	660	540	1320
285	675	570	1350
300	690	600	1380
315	705	630	1410
330	720	660	1440
345	735	690	1470
360	750	720	1500
375	765	750	1530
390	780	780	1560

ES6 refers to version 6 of the ECMA Script programming language. ECMA Script is the standardized name for **JavaScript**, and version 6 is the next version after version 5, which was released in 2011. It is a major enhancement to the JavaScript language, and adds many more features intended to make large-scale software development easier.

ECMAScript, or ES6, was published in June 2015. It was subsequently renamed to ECMAScript 2015. Web browser support for the full language is not yet complete, though major portions are supported. Major web browsers support some features of ES6. However, it is possible to use software known as a **transpiler** to convert ES6 code into ES5, which is better supported on most browsers.

`let` Declarations

we can now create declarations which are bound to any block, called (unsurprisingly) *block scoping*. This means all we need is a pair of `{ .. }` to create a scope. Instead of using `var`, which always declares variables attached to the enclosing function (or global, if top level) scope, use `let`:

```
var a = 2;

{
    let a = 3;
    console.log( a );   // 3
}

console.log( a );       // 2
```

It's not very common or idiomatic thus far in JS to use a standalone `{ .. }` block as shown there, but it's always been totally valid. And developers from other languages that have *block scoping* will readily recognize that pattern.

`const` Declarations

There's one other form of block-scoped declaration to consider, the `const`, which creates *constants*.

What exactly is a *constant*? It's a variable that's read-only after its initial value is set. Consider:

```
{
    const a = 2;
    console.log( a );   // 2

    a = 3;              // TypeError!
}
```

You are not allowed to change the value of the variable once it's been set, at declaration time. A `const` declaration must have an explicit initialization. If you wanted a *constant* with the `undefined` value, you'd have to declare `const a = undefined` to get it.

Constants are not a restriction on the value itself, but on the variable assignment of that value. In other words, the value is not frozen, just the assignment of it. If the value is complex, such as an object or array, the contents of the value can still be modified:

```
{
    const a = [1,2,3];
    a.push( 4 );
    console.log( a );       // [1,2,3,4]

    a = 42;                 // TypeError!
}
```

Block-Scoped Declarations

You're probably aware that the fundamental unit of variable scoping in JavaScript has always been the `function`. If you needed to create a block of scope, the most prevalent way to do so was the IIFE (immediately invoked function expression), such as:

```
var a = 2;

(function IIFE(){
    var a = 3;
    console.log( a );   // 3
})();

console.log( a );       // 2
```

Template Literals

```
var name = "Kyle";

var greeting = `Hello ${name}!`;

console.log( greeting );            // "Hello Kyle!"
console.log( typeof greeting );     // "string"
```

Consider:

```
function upper(s) {
    return s.toUpperCase();
}

var who = "reader"

var text =
`A very ${upper( "warm" )} welcome
to all of you ${upper( `${who}s` )}!`;

console.log( text );
// A very WARM welcome
// to all of you READERS!
```

Arrow Functions

It's important to understand the frustrations that `this` based programming with normal functions brings, because that is the primary motivation for the new ES6 `=>` arrow function feature.

Let's first illustrate what an arrow function looks like, as compared to normal functions:

```
function foo(x,y) {
    return x + y;

}

// versus

var foo = (x,y) => x + y;
```

The arrow function definition consists of a parameter list (of zero or more parameters, and surrounding `( .. )` if there's not exactly one parameter), followed by the `=>` marker, followed by a function body.

So, in the previous snippet, the arrow function is just the `(x,y) => x + y` part, and that function reference happens to be assigned to the variable `foo`.

Arrow Functions

The body only needs to be enclosed by { .. } if there's more than one expression, or if the body consists of a non-expression statement. If there's only one expression, and you omit the surrounding { .. }, there's an implied `return` in front of the expression, as illustrated in the previous snippet.

Here's some other arrow function variations to consider:

```
var f1 = () => 12;
var f2 = x => x * 2;
var f3 = (x,y) => {
    var z = x * 2 + y;
    y++;
    x *= 3;
    return (x + y + z) / 2;
};
```

Arrow functions are *always* function expressions; there is no arrow function declaration. It also should be clear that they are anonymous function expressions — they have no named reference for the purposes of recursion or event binding/unbinding

Not Just Shorter Syntax, But `this`

So now we can conclude a more nuanced set of rules for when => is appropriate and not:

- If you have a short, single-statement inline function expression, where the only statement is a `return` of some computed value, *and* that function doesn't already make a `this` reference inside it, *and* there's no self-reference (recursion, event binding/unbinding), *and* you don't reasonably expect the function to ever be that way, you can probably safely refactor it to be an => arrow function.
- If you have an inner function expression that's relying on a `var self = this` hack or a `.bind(this)` call on it in the enclosing function to ensure proper `this` binding, that inner function expression can probably safely become an => arrow function.
- If you have an inner function expression that's relying on something like `var args = Array.prototype.slice.call(arguments)` in the enclosing function to make a lexical copy of `arguments`, that inner function expression can probably safely become an => arrow function.
- For everything else — normal function declarations, longer multi-statment function expressions, functions which need a lexical name identifier self-reference (recursion, etc.), and any other function which doesn't fit the previous characteristics — you should probably avoid => function syntax.

Bottom line: => is about lexical binding of `this`, `arguments`, and `super`. These are intentional features designed to fix some common problems, not bugs, quirks, or mistakes in ES6.

`class`

Even though JS's prototype mechanism doesn't work like traditional classes, that doesn't stop the strong tide of demand on the language to extend the syntactic sugar so that expressing "classes" looks more like real classes. Enter the ES6 `class` keyword and its associated mechanism.

This feature is the result of a highly contentious and drawn out debate, and represents a smaller subset compromise from several strongly-opposed views on how to approach JS classes. Most developers who want full classes in JS will find parts of the new syntax quite inviting, but will find important bits still missing. Don't worry, though. TC39 is already working on additional features to augment classes in the post-ES6 timeframe.

At the heart of the new ES6 class mechanism is the `class` keyword, which identifies a *block* where the contents define the members of a function's prototype. Consider:

```
class Foo {
    constructor(a,b) {
        this.x = a;
        this.y = b;
    }

    gimmeXY() {
        return this.x * this.y;
    }
}
```

Subclass Constructor

Constructors are not required for classes or subclasses; a default constructor is substituted in both cases if omitted. However, the default substituted constructor is different for a direct class versus an extended class.

Specifically, the default subclass constructor automatically calls the parent constructor, and passes along any arguments. In other words, you could think of the default subclass constructor sort of like this:

```
constructor(...args) {
    super(...args);
}
```

This is an important detail to note. Not all class languages have the subclass constructor automatically call the parent constructor. C++ does, but Java does not. But more importantly, in pre-ES6 classes, such automatic "parent constructor" calling does not happen. Be careful when converting to ES6 `class` if you've been relying on such calls *not* happening.

Another perhaps surprising deviation/limitation of ES6 subclass constructors: in a constructor of a subclass, you cannot access `this` until `super(..)` has been called. The reason is nuanced and complicated, but it boils down to the fact that the parent constructor is actually the one creating/initializing your instance's `this`. Pre-ES6, it works oppositely; the `this` object is created by the "subclass constructor", and then you call a "parent constructor" with the context of the "subclass" `this`.

Destructuring

ES6 introduces a new syntactic feature called *destructuring*, which may be a little less confusing sounding if you instead think of it as *structured assignment*. To understand this meaning, consider:

```
function foo() {
    return [1,2,3];
}

var tmp = foo(),
    a = tmp[0], b = tmp[1], c = tmp[2];

console.log( a, b, c );             // 1 2 3
```

As you can see, we created a manual assignment of the values in the array that `foo()` returns to individual variables `a`, `b`, and `c`, and to do so we (unfortunately) needed the `tmp` variable.

We can do similar with objects:

```
function bar() {
    return {
        x: 4,
        y: 5,
        z: 6
    };
}

var tmp = bar(),
    x = tmp.x, y = tmp.y, z = tmp.z;

console.log( x, y, z );             // 4 5 6
```

The `tmp.x` property value is assigned to the `x` variable, and likewise for `tmp.y` to `y` and `tmp.z` to `z`.

`Array.of(..)` Static Function

There's a well known gotcha with the `Array(..)` constructor, which is that if there's only one argument passed, and that argument is a number, instead of making an array of one element with that number value in it, it constructs an empty array with a `length` property equal to the number. This action produces the unfortunate and quirky "empty slots" behavior that's reviled about JS arrays.

`Array.of(..)` replaces `Array(..)` as the preferred function-form constructor for arrays, because `Array.of(..)` does not have that special single-number-argument case. Consider:

```
var a = Array( 3 );
a.length;                       // 3
a[0];                           // undefined

var b = Array.of( 3 );
b.length;                       // 1
b[0];                           // 3

var c = Array.of( 1, 2, 3 );
c.length;                       // 3
c;                              // [1,2,3]
```

`Array.from(..)` Static Function

An "array-like object" in JavaScript is an object that has a `length` property on it, specifically with an integer value of zero or higher.

`Arrav.from(..)` looks to see if the first argument is an iterable , and if so, it uses the iterator to produce values to "copy" into the returned array. Since real arrays have an iterator for those values, that iterator is automatically used.

But if you pass an array-like object as the first argument to `Array.from(..)`, it behaves basically the same as `slice()` (no arguments!) or `apply(..)` does, which is that it simply loops over the value, accessing numerically named properties from `0` up to whatever the value of `length` is.

Consider:

```
var arrLike = {
    length: 4,
    2: "foo"
};


Array.from( arrLike );
// [ undefined, undefined, "foo", undefined ]
```

Since positions `0`, `1`, and `3` didn't exist on `arrLike`, the result was the `undefined` value for each of those slots.

You could produce a similar outcome like this:

```
var emptySlotsArr = [];
emptySlotsArr.length = 4;
emptySlotsArr[2] = "foo";

Array.from( emptySlotsArr );
// [ undefined, undefined, "foo", undefined ]
```

Spread / Rest

ES6 introduces a new `...` operator that's typically referred to as the *spread* or *rest* operator, depending on where/how it's used. Let's take a look:

```
function foo(x,y,z) {
    console.log( x, y, z );
}

foo( ...[1,2,3] );              // 1 2 3
```

You'll typically see that usage as is shown in that previous snippet, when spreading out an array as a set of arguments to a function call. In this usage, `...` acts to give us a simpler syntactic replacement for the `apply(..)` method, which we would typically have used pre-ES6 as:

```
foo.apply( null, [1,2,3] );     // 1 2 3
```

But `...` can be used to spread out/expand a value in other contexts as well, such as inside another array declaration:

```
var a = [2,3,4];
var b = [ 1, ...a, 5 ];

console.log( b );                   // [1,2,3,4,5]
```

In this usage, `...` is basically replacing `concat(..)`, as the above behaves like `[1].con cat( a, [5] )`.

The other common usage of `...` can be seen as almost the opposite; instead of spreading a value out, the `...` *gathers* a set of values together into an array. Consider:

```
function foo(x, y, ...z) {
    console.log( x, y, z );
}

foo( 1, 2, 3, 4, 5 );           // 1 2 [3,4,5]
```

The `...z` in this snippet is essentially saying: "gather the *rest* of the arguments (if any) into an array called `z`." Since `x` was assigned `1`, and `y` was assigned `2`, the rest of the arguments `3`, `4`, and `5` were gathered into `z`.

Of course, if you don't have any named parameters, the `...` gathers all arguments:

```
function foo(...args) {
    console.log( args );
}

foo( 1, 2, 3, 4, 5);            // [1,2,3,4,5]
```

The `...args` in the `foo(..)` function declaration is usually called "rest parameters", since you're collecting the rest of the parameters. I prefer "gather", since it's more descriptive of what it does, not what it contains.

The best part about this usage is that is provides a very solid alternative to using the long-since deprecated `arguments` array — actually, it's not really an array, but an array-like object. Since `args` (or whatever you call it — a lot of people prefer `r` or `rest`) is a real array, we can get rid of lots of silly pre-ES6 tricks we jumped through to make `arguments` into something we can treat as an array.

```
// doing things the new ES6 way
function foo(...args) {
    // `args` is already a real array

    // discard first element in `args`
    args.shift();

    // pass along all of `args` as arguments
    // to `console.log(..)`
    console.log( ...args );
}

// doing things the old-school pre-ES6 way
function bar() {
    // turn `arguments` into a real array
    var args = Array.prototype.slice.call( arguments );

    // add some elements on the end
    args.push( 4, 5 );

    // filter out odd numbers
    args = args.filter( function(v){
        return v % 2 == 0;
    } );

    // pass along all of `args` as arguments
    // to `foo(..)`
    foo.apply( null, args );
}

bar( 0, 1, 2, 3 );                  // 2 4
```

The `...args` in the `foo(..)` function declaration gathers arguments, and the `...args` in the `console.log(..)` call spreads them out. That's a good illustration of the symmetric but opposite uses of the `...` operator.

Default Parameter Values

```
function foo(x = 11, y = 31) {
    console.log( x + y );
}

foo();                  // 42
foo( 5, 6 );            // 11
foo( 0, 42 );           // 42

foo( 5 );               // 36
foo( 5, undefined );    // 36 <-- `undefined` is missing
foo( 5, null );         // 5  <-- null coerces to `0`

foo( undefined, 6 );    // 17 <-- `undefined` is missing
foo( null, 6 );         // 6  <-- null coerces to `0`
```

Maps

Those with much JS experience know that objects are the primary mechanism for creating unordered key/value-pair data structures, otherwise known as maps. However, the major drawback with objects-as-maps is the inability to use a non-string value as the key.

```
var m = {};

var x = { id: 1 },
    y = { id: 2 };

m[x] = "foo";
m[y] = "bar";

m[x];                           // "bar"
m[y];                           // "bar"
```

What's going on here? The two objects `x` and `y` both stringify to `"[object Object]"`, so only that one key is being set in `m`.

Some have implemented fake maps by maintaining a parallel array of non-string keys alongside an array of the values, such as:

```
var keys = [], vals = [];

var x = { id: 1 },
    y = { id: 2 };

keys.push( x );
vals.push( "foo" );

keys.push( y );
vals.push( "bar" );

keys[0] === x;                  // true
vals[0];                        // "foo"

keys[1] === y;                  // true
vals[1];                        // "bar"
```

Symbols

For the first time in quite awhile, a new primitive type has been added to JavaScript, in ES6: the `symbol`. Unlike the other primitive types, however, symbols don't have a literal form.

Here's how you create a symbol:

```
var sym = Symbol( "some optional description" );

typeof sym;     // "symbol"
```

Some things to note:

- You cannot and should not use `new` with `Symbol(..)`. It's not a constructor, nor are you producing an object.
- The parameter passed to `Symbol(..)` is optional. If passed, it should be a string that gives a friendly description for the symbol's purpose.
- The `typeof` output is a new value (`"symbol"`) that is the primary way to identify a symbol.

The internal value of a symbol itself — referred to as its `name` — is hidden from the code and cannot be obtained. You can think of this symbol value as an automatically generated, unique (within your application) string value.

But if the value is hidden and unobtainable, what's the point of having a symbol at all?

The main point of a symbol is to create a string-like value that can't collide with any other value.

Iterators

An *iterator* is a structured pattern for pulling information from a source in one-at-a-time fashion. This pattern has been around programming for a long time. And to be sure, JS developers have been ad hoc designing and implementing iterators in JS programs since before anyone can remember, so it's not at all a new topic.

What ES6 has done is introduce an implicit standardized interface for iterators. Many of the built in data structures in JavaScript will now expose an iterator implementing this standard. And you can also construct your own iterators adhering to the same standard, for maximal interoperability.

Iterators are a way of organizing ordered, sequential, pull-based consumption of data.

For example, you may implement a utility that produces a new unique identifier each time it's requested. Or you may produce an infinite series of values that rotate through a fixed list, in round-robin fashion. Or you could attach an iterator to a database query result to pull out new rows one at a time.

Though not as common a usage of iterators to this point in JS, iterators can also be thought of as controlling behavior one step at a time. This can be illustrated quite clearly when considering generators, though you can certainly do the same without generators.

GRUNT

The JavaScript Task Runner

HackerU PRO UPDATE NEEDED

What is GRUNT?

Grunt is a javascript task runner a tool used to automatically perform frequent tasks such as:

- Minification
- Compilation
- Unit testing
- Linting

What does GRUNT do?

Performs repetitive tasks such as:

- Prefixing css rules
- Compiling sass files into css
- Minifying js/css files
- Concatenating files

Preparing a new Grunt project

A typical setup will involve adding two files to your project:

package.json: This file is used by npm to store metadata for projects published as npm modules. You will list grunt and the Grunt plugins your project needs as devDependencies in this file.

Gruntfile: This file is named `Gruntfile.js` and is used to configure or define tasks and load Grunt plugins.

The package.json file belongs in the root directory of your project, next to the Gruntfile, and should be committed with your project source. Running npm install in the same folder as a package.json file will install the correct version of each dependency listed therein.

There are a few ways to create a package.json file for your project:

- Most grunt-init templates will automatically create a project-specific package.json file.
- The npm init command will create a basic package.json file.

For example:

```
{
  "name": "my-project-name",
  "version": "0.1.0",
  "devDependencies": {
    "grunt": "~0.4.5",
    "grunt-contrib-jshint": "~0.10.0",
    "grunt-contrib-nodeunit": "~0.4.1",
    "grunt-contrib-uglify": "~0.5.0"
  }
}
```

Installing Grunt and Grunt plugins

The easiest way to add Grunt and gruntplugins to an existing `package.json` is with the command `npm install <module> --save-dev`. Not only will this install `<module>` locally, but it will automatically be added to the devDependencies section, using a tilde version range.

For example:

```
npm install grunt --save-dev
```

```
npm install grunt-contrib-jshint --save-dev
```

The Gruntfile

The Gruntfile.js file is a valid JavaScript file that belongs in the root directory of your project, next to the package.json file, and should be committed with your project source.

A Gruntfile is comprised of the following parts:

- The "wrapper" function
- Project and task configuration
- Loading Grunt plugins and tasks
- Custom tasks

An example Gruntfile

```
module.exports = function(grunt) {

  // Project configuration.
  grunt.initConfig({
    pkg: grunt.file.readJSON('package.json'),
    uglify: {
      options: {
        banner: '/*! <%= pkg.name %> <%= grunt.template.today("yyyy-mm-dd") %> */\n'
      },
      build: {
        src: 'src/<%= pkg.name %>.js',
        dest: 'build/<%= pkg.name %>.min.js'
      }
    }
  });

  // Load the plugin that provides the "uglify" task.
  grunt.loadNpmTasks('grunt-contrib-uglify');

  // Default task(s).
  grunt.registerTask('default', ['uglify']);

};
```

The Wrapper function:

```
module.exports = function(grunt) {
  // Do grunt-related things in here
};
```

Project and task configuration

```
// Project configuration.
grunt.initConfig({
  pkg: grunt.file.readJSON('package.json'),
  uglify: {
    options: {
      banner: '/*! <%= pkg.name %> <%= grunt.template.today("yyyy-mm-dd") %> */\n'
    },
    build: {
      src: 'src/<%= pkg.name %>.js',
      dest: 'build/<%= pkg.name %>.min.js'
    }
  }
});
```

Loading grunt plugins and tasks

Many commonly used tasks like concatenation, minification and linting are available as grunt plugins. As long as a plugin is specified in package.json as a dependency, and has been installed via npm install, it may be enabled inside your Gruntfile with a simple command:

```
// Load the plugin that provides the "uglify" task.
grunt.loadNpmTasks('grunt-contrib-uglify');
```

Note: the grunt --help command will list all available tasks.

Custom tasks

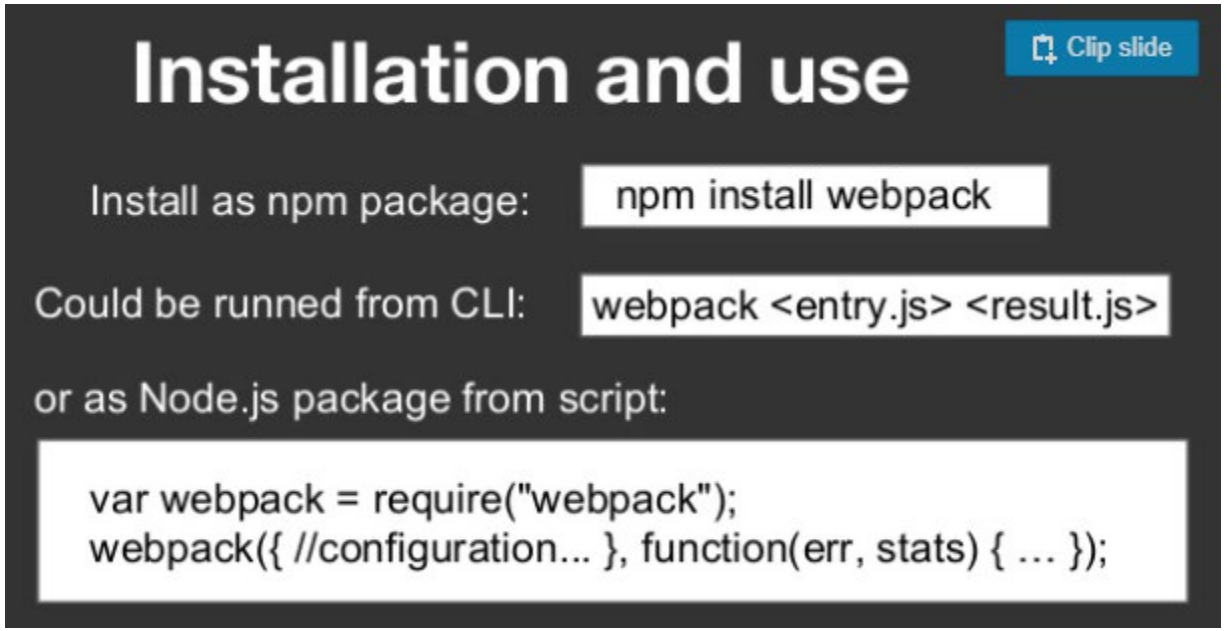
You can configure Grunt to run one or more tasks by default by defining a `default` task. In the following example, running `grunt` at the command line without specifying a task will run the `uglify` task. This is functionally the same as explicitly running `grunt uglify` or even `grunt default`. Any number of tasks (with or without arguments) may be specified in the array.

```
// Default task(s).
grunt.registerTask('default', ['uglify']);
```

If your project requires tasks not provided by a Grunt plugin, you may define custom tasks right inside the Gruntfile. For example, this Gruntfile defines a completely custom default task that doesn't even utilize task configuration:

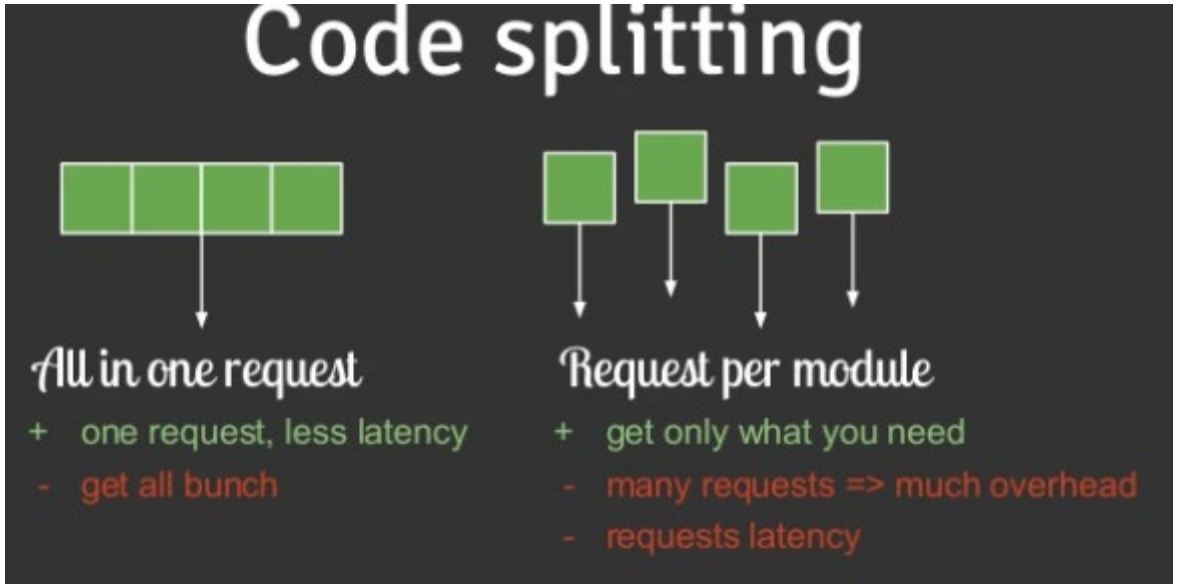
```
module.exports = function(grunt) {

  // A very basic default task.
  grunt.registerTask('default', 'Log some stuff.', function() {
    grunt.log.write('Logging some stuff...').ok();
  });

};
```

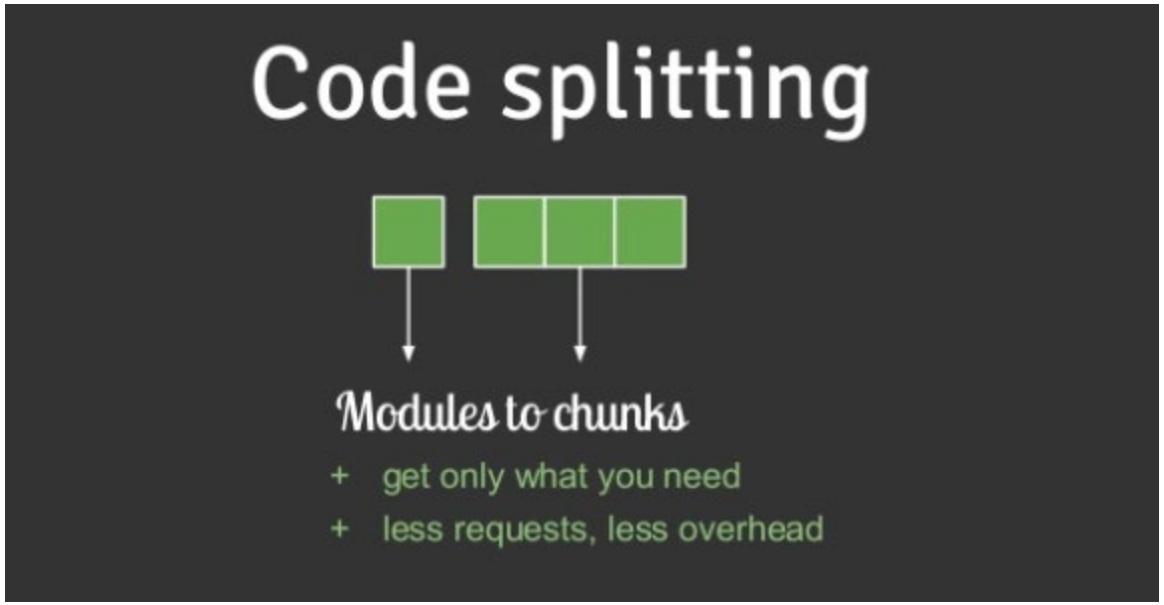

JS
Webpack
The new rock star of build tools


Installation and use
Clip slide
Install as npm package:
npm install webpack
Could be runned from CLI:
webpack <entry.js> <result.js>
or as Node.js package from script:
var webpack = require("webpack");
webpack({ //configuration... }, function(err, stats) { ... });

Webpack Config

you don't need to write pure JSON into the configuration. Use any JavaScript you want. It's just a node.js (CommonJs) module...


Code splitting
All in one request
+ one request, less latency
- get all bunch
Request per module
+ get only what you need
- many requests => much overhead
- requests latency


Code splitting
Modules to chunks
+ get only what you need
+ less requests, less overhead

But let check how webpack can help us with simple example project

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
  <link rel="stylesheet" href="module1.css">
  <link rel="stylesheet" href="module2.css">
  <link rel="stylesheet" href="module3.css">
</head>
<body>
  Hello world
  <script src="module1.js"></script>
  <script src="module2.js"></script>
  <script src="module3.js"></script>
</body>
</html>
```

webpack.config.js

```
module.exports = {
  entry: {
    app: [
      "./app/js/module1",
      "./app/js/module2"
    ]
  },
  output: {
    path: "./public/js/",
    filename: "bundle.js"
  }
};
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
  <link rel="stylesheet" href="module1.css">
  <link rel="stylesheet" href="module2.css">
  <link rel="stylesheet" href="module3.css">
</head>
<body>
  Hello world
  <script src="bundle.js"></script>
  <script src="module3.js"></script>
</body>
</html>
```

What about other files??

.css .png .jsx .woff .svg .
json .styl .eot .html .scss .
jpeg .jade ...

Loaders

Loaders allow you to preprocess files as you require() or “load” them.
Loaders are kind of like “tasks” are in other build tools

code -> loaders -> plugins -> output

Continue with styles

```
npm install style-loader css-loader sass-loader autoprefixer-loader
```

```
require("style!css!autoprefixer!sass!./mystyles.scss");
```

```
module.exports = {
  ...
  module: {
    loaders: [
      { test: /\.scss$/, loader: "style!css!autoprefixer!sass" }
    ]}
};
```

Optimize workflow with images !

```
npm install url-loader
```

Insert image in the bundle (via Base64) if it's smaller than 10kb

```
require("url?limit=10000!./file.png");
```

Modularity

module1.js

```
require("module1.scss");
console.log("module1 js code");
```

module3.js

```
require("module3.scss");
console.log("module3 js code");
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  Hello world
  <script src="bundle.js"></script>
  <script src="module3.js"></script>
</body>
</html>
```

Code splitting

webpack.config.js

```
module.exports = {
  ...
  entry: {
    app: "./app/js/module1"
  },
  ...
};
```

module1.js

```
document
  .querySelector(".btn-require-module")
  .addEventListener("click", function(){
    require.ensure(["./module3"], function(require){
      var module3 = require("./module3");
    }, "module3")
  })
```


Karma
React
Mocha
ES6
ESLint
JSCS
Webpack
loaders
60+
Riot
Polymer
Vue

Plugins

code -> loaders -> plugins -> output


node
JS

Install


https://nodejs.org/en/download/
LTS
Recommended For Most Users
Current
Latest Features
Windows Installer
Macintosh Installer
Source Code
Windows Installer (.msi)
32-bit
64-bit
Windows Binary (.exe)
32-bit
64-bit
macOS Installer (.pkg)
64-bit
macOS Binaries (.tar.gz)
64-bit
Linux Binaries (x86/x64)
32-bit
64-bit
Linux Binaries (ARM)
ARMv6
ARMv7
ARMv8
Source Code
node-v6.9.4.tar.gz
node –v (Node version)

Open Source, cross platform

Huge community

Ryan Dahl, Joyent

Developed in 2009

Published in 2011 (Linux + Windows)

Similar design to systems like Ruby's Event Machine or Python's Twisted

Who is using NodeJs?

Built on Chrome's Javascript Runtime (**V8**)

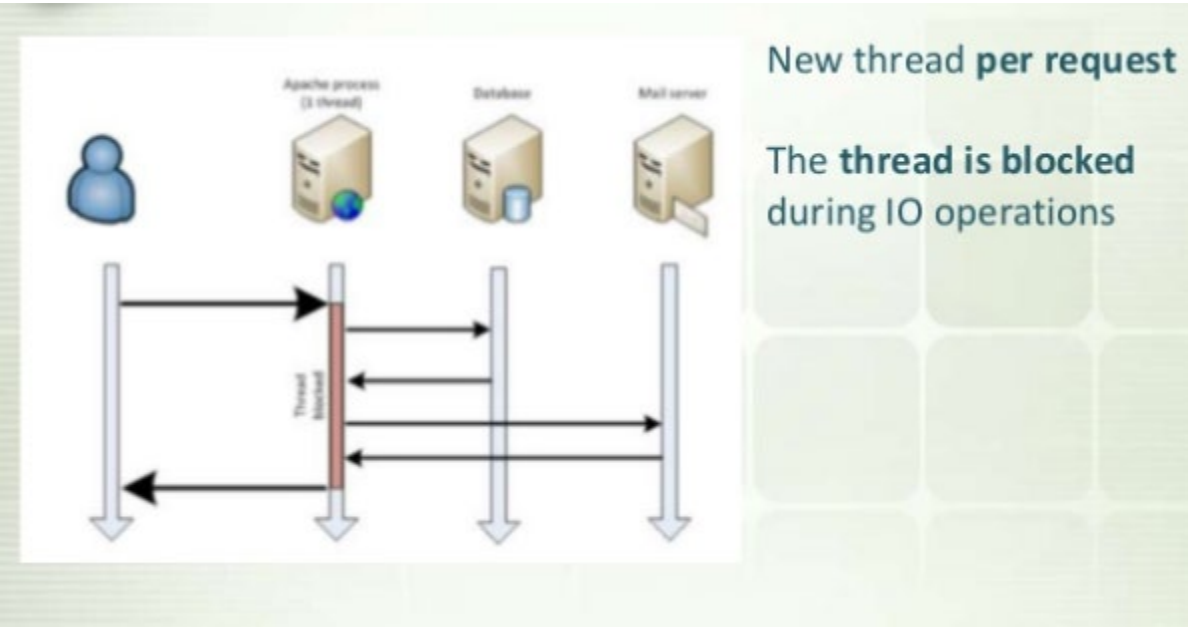
Single thread, event oriented

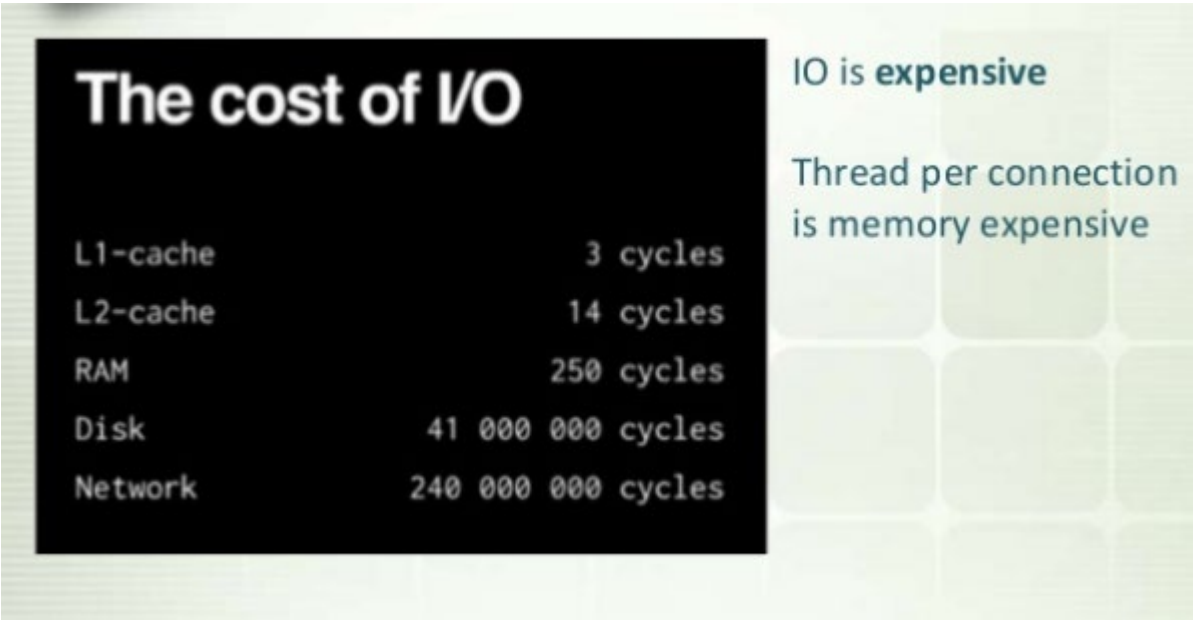
Event-Driven with Non-blocking I/O

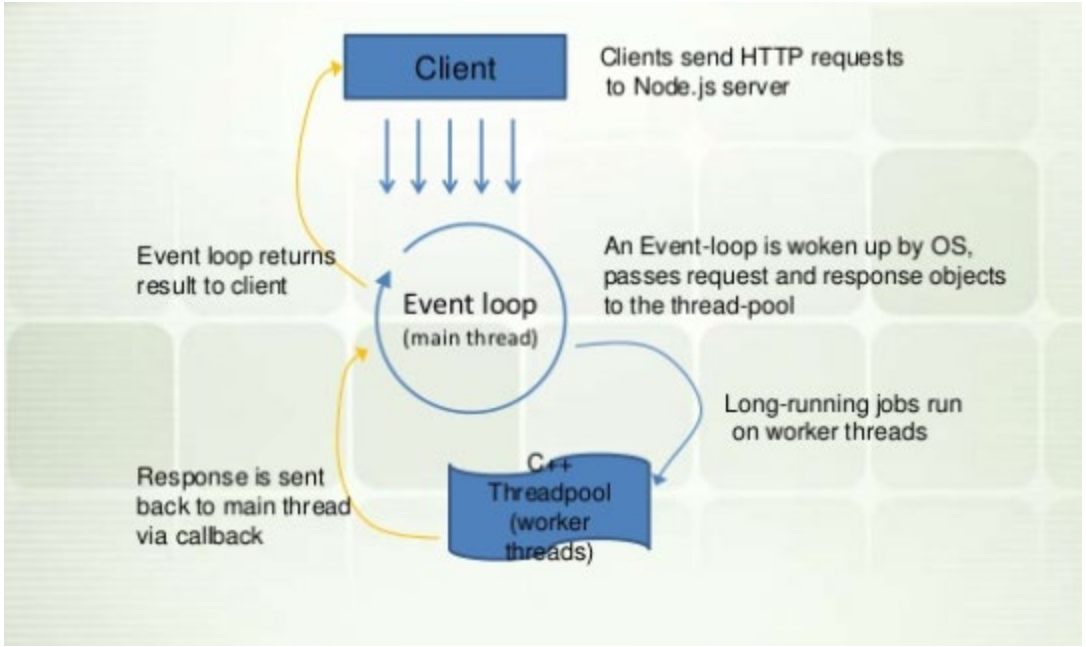
Use a Event Loop for Non-blocking I/O

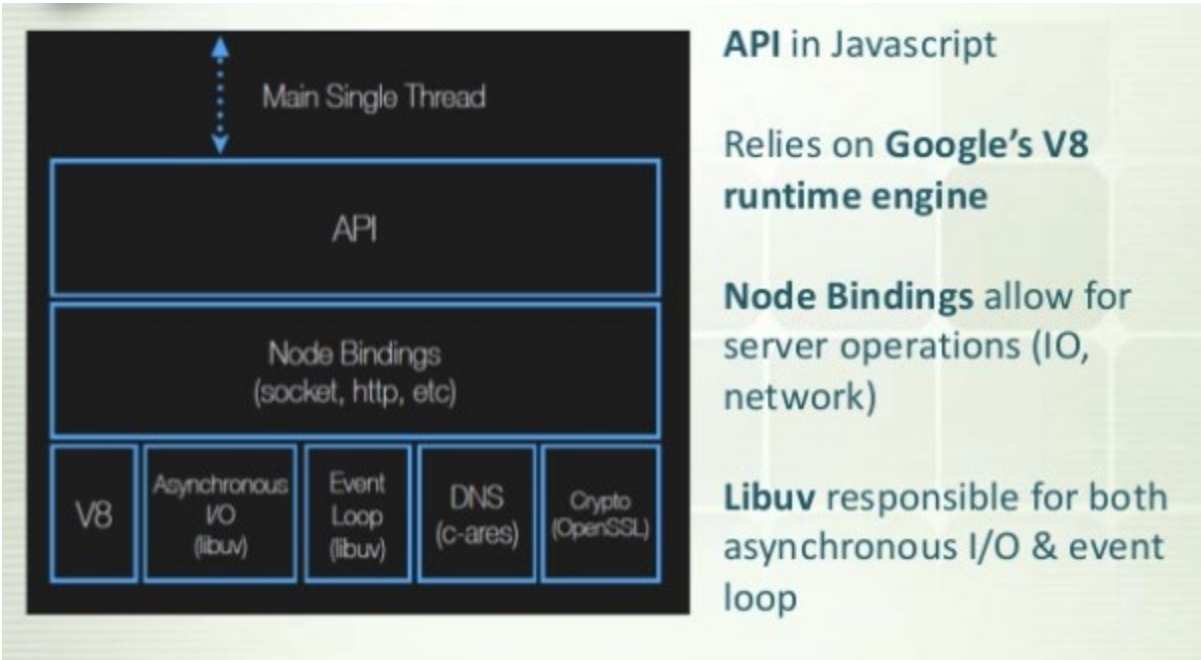
Can handle thousands of concurrent connections with minimal overhead (CPU/Memory)

40% JS and 60% C++


Apache process
(1 thread)
Database
Mail server
Thread blocked
New thread **per request**
The **thread is blocked**
during IO operations


The cost of I/O
L1-cache 3 cycles
L2-cache 14 cycles
RAM 250 cycles
Disk 41 000 000 cycles
Network 240 000 000 cycles
IO is expensive
Thread per connection is memory expensive


Client
Clients send HTTP requests to Node.js server
Event loop returns result to client
Event loop
(main thread)
An Event-loop is woken up by OS, passes request and response objects to the thread-pool
Long-running jobs run on worker threads
Response is sent back to main thread via callback
C++ Threadpool (worker threads)


Main Single Thread
API
Node Bindings
(socket, http, etc)
V8
Asynchronous I/O (libuv)
Event Loop (libuv)
DNS (c-ares)
Crypto (OpenSSL)
API in Javascript
Relies on Google's V8 runtime engine
Node Bindings allow for server operations (IO, network)
Libuv responsible for both asynchronous I/O & event loop

When to use NodeJs?

When not to use NodeJs?

Heavy CPU intensive calculations on server-side

Concurrent Task based applications

NPM

Package manager for Node

Bundled and installed automatically with the environment

Frequently Usage

- npm install --save package_name
- npm update

How Does it works?

- Read package.json
- Installs the dependencies in the local node_modules folder
- In global modes, it make a node module accessible to all

Express.js

Web application framework

Minimal and Flexible

Many Framework based on Express

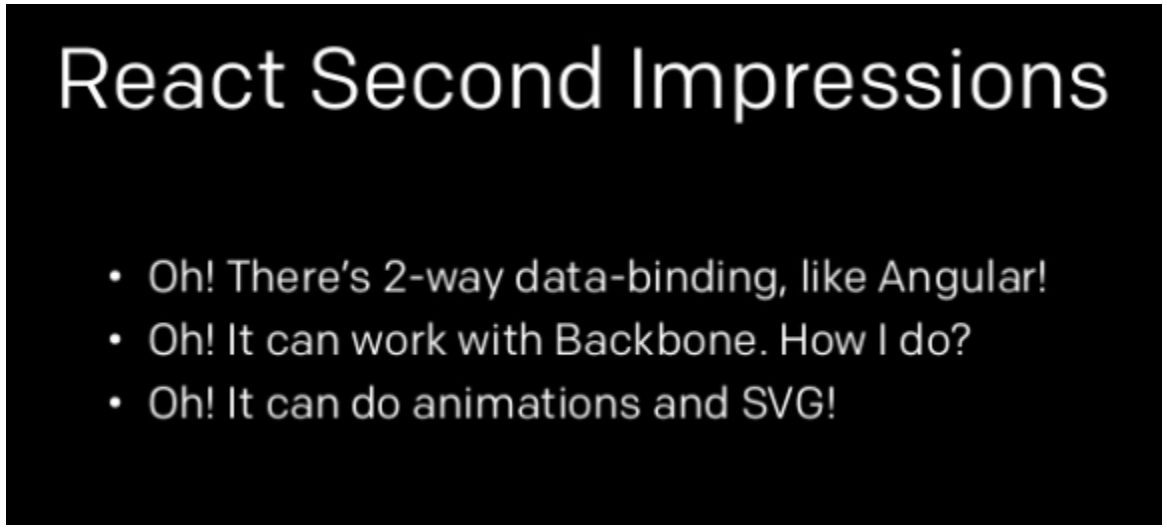
Features

- MVC
- Routing
- Session Support
- Cookie Parsing
- Middleware (Request-Response Cycle)
- View Engines (Jade, Ejs, Vash)


React

React First Impressions

```
var LikeButton = React.createClass({
  getInitialState: function() {
    return {liked: false};
  },
  handleClick: function(event) {
    this.setState({liked: !this.state.liked});
  },
  render: function() {
    var text = this.state.liked ? 'like' : 'unlike';
    return (
      <p onClick={this.handleClick}>
        You {text} this. Click to toggle.
      </p>
    );
  }
});
```

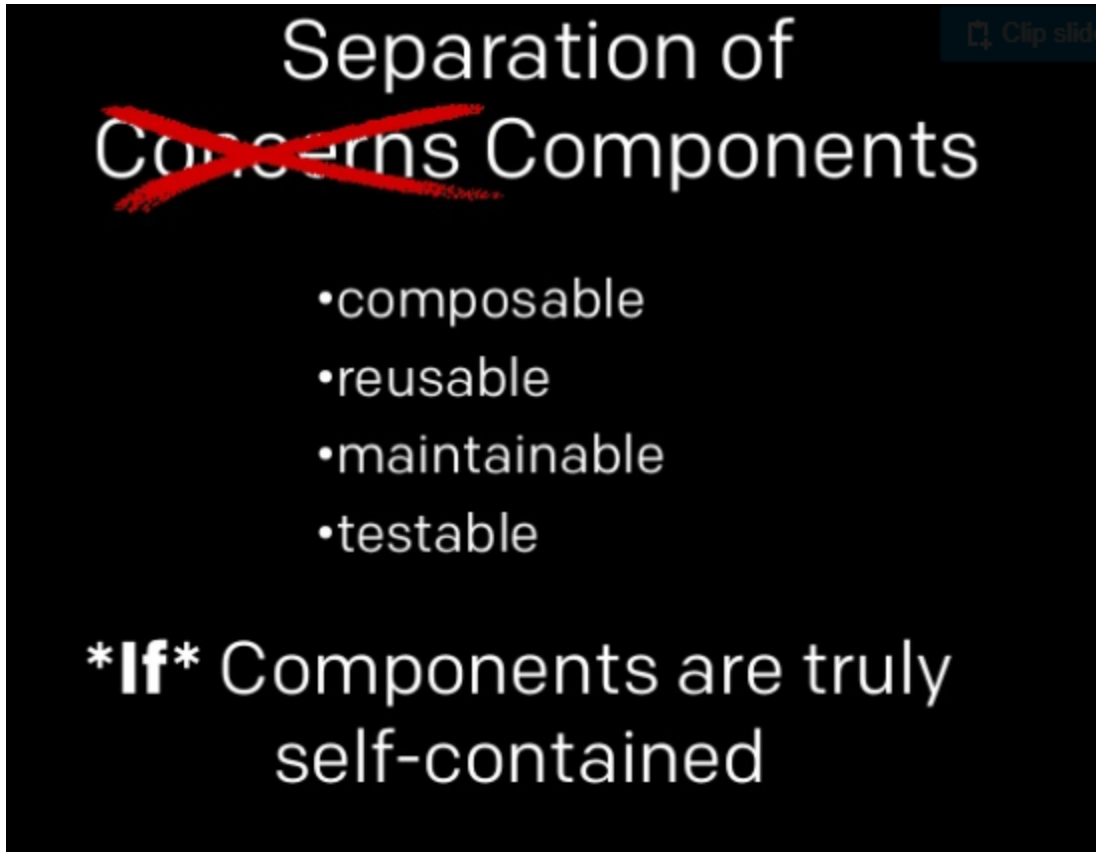

React Second Impressions
• Oh! There's 2-way data-binding, like Angular!
• Oh! It can work with Backbone. How I do?
• Oh! It can do animations and SVG!

#1
Everything is a Component

React has no...

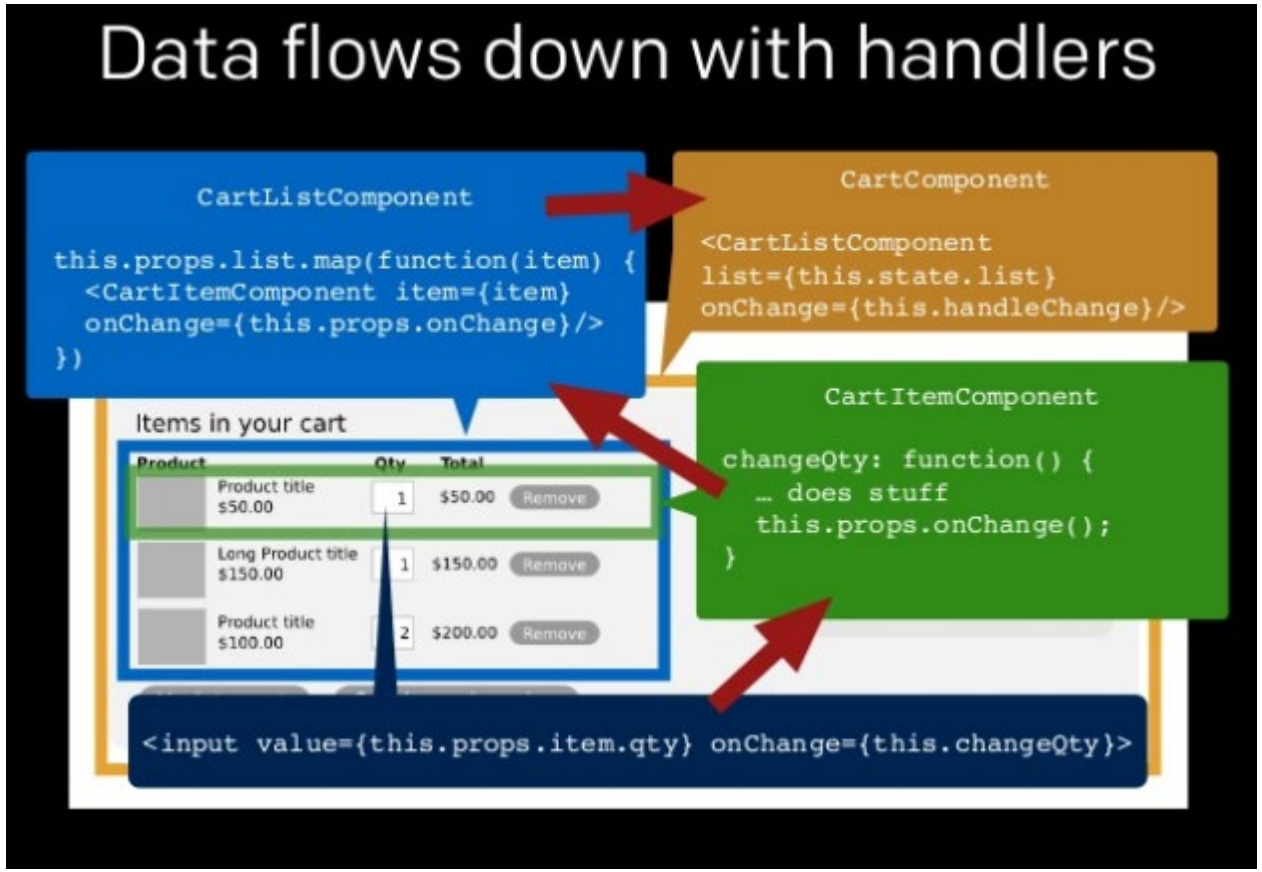
... controllers
... directives
... templates
... global event listeners
... models
... no view models

Just Components


Separation of
Components
•composable
•reusable
•maintainable
•testable
If Components are truly
self-contained

#2
Single Source of Truth

Data only flows one way



#3
Virtual DOM

Touching the DOM is evil

- It's inconsistent
- It's hard to test
- It's brittle
- It's EXPENSIVE!

Virtual DOM

- It's a pure Javascript, in-memory representation of the DOM
- render() fires whenever something changes
- React modifies the real DOM to match
- It's FAST
- It's pure
- It just works

Centrally located in Ramat-Gan, 3 minutes from the Savidor train station.

Professional and comprehensive training with the industry-leading experts.

Boutique and tailored services.

Spacious classrooms equipped with state-of-the-art technology.

Our Business Department at Your Service!

www.hackerupro.co.il | Tel: 03-5087690
info@hackerupro.co.il