CSS Flexbox

Picking the right display context and managing the layout of a webpage can be extremely inconvenient to any web developer. However, it’s part of the job and we have to face it head on.

Luckily, the flexbox is the right tool for overcoming that obstacle. That is the subject matter we’ll be dealing with during this skill. Here’s an overview:

* What is CSS Flexbox?
* The display flex.
* The CSS position.
* The CSS grid.

# Flexbox

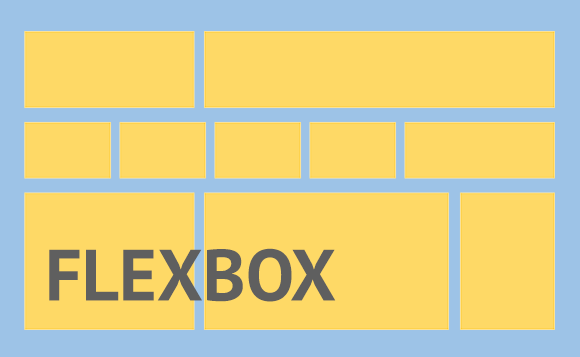
Time to meet Flex!

As we have already seen, we can display HTML elements using layout modes: Block, for sections in a web page, Inline, for text, etc.

The Flexbox will make it easier to design flexible, dynamic, and responsive layout structure without using float or positioning.

Learning Flexbox can be difficult for most people as it’s not as enjoyable as other skills we will learn. But, it challenges you to rethink your approach when it comes to layouts in CSS.

Don’t worry, I will walk you through everything that you need to know.  
So, let’s jump right in!



# What is Flexbox?

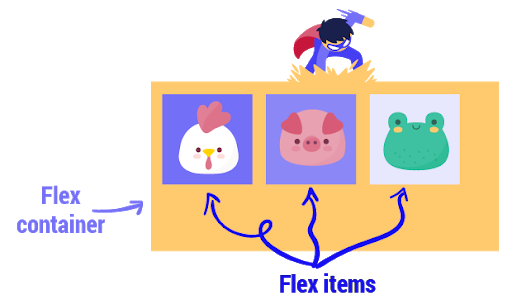
Flexbox has an overly technical definition. For the sake of simplifying this course, we’ll skip the technicalities.

You want to imagine flexbox as the layout ninja for CSS. It’s incredibly practical, dynamic and has a lot of advantages.

When you need to deal with the layout in your styles, the CSS flexbox model is likely to be your best bet.

**How does it work?**

Flexbox is kicked off by making a parent element into a flex container.



# Use display flex.

How can we access the flexbox’s super powers?

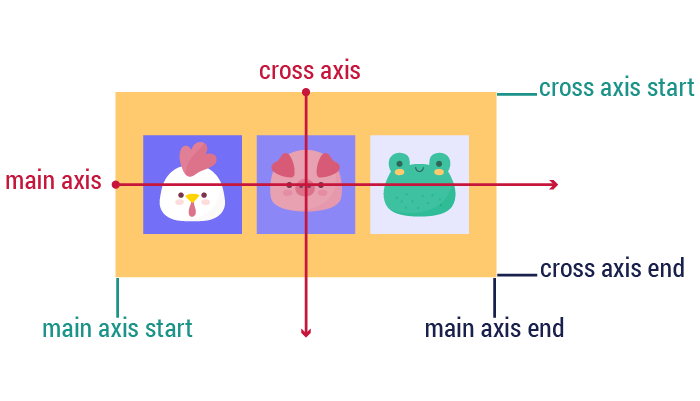
Pretty simple, let's assume the parent element in question is a div with a class name of flexy, we simply apply the following:



That one line of code does the following:  
It enables a flex context for all its direct children, so if all items need to shrink, each one will shrink equally, and if there’s no space they will be allowed to shrink horizontally until they all fit.

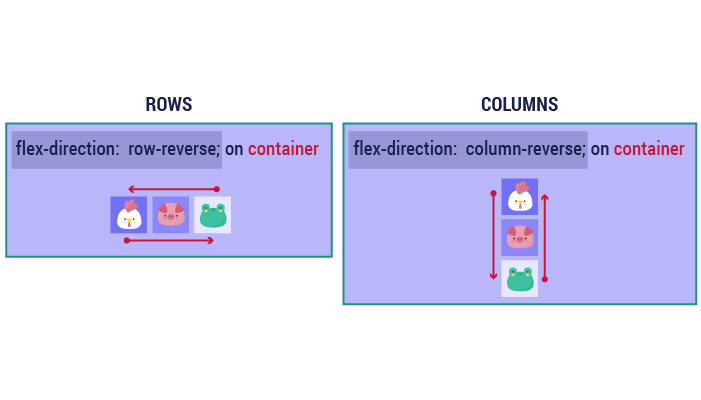
# So why do we use Flexbox and what are its super powers?

After we have made the parent a flex container, we should proceed to using it. One of the many things we can do with flexbox, is to center a child element within a parent. This centering will be along both sides, vertical and horizontal, all according to the main axis.



# Column and Row

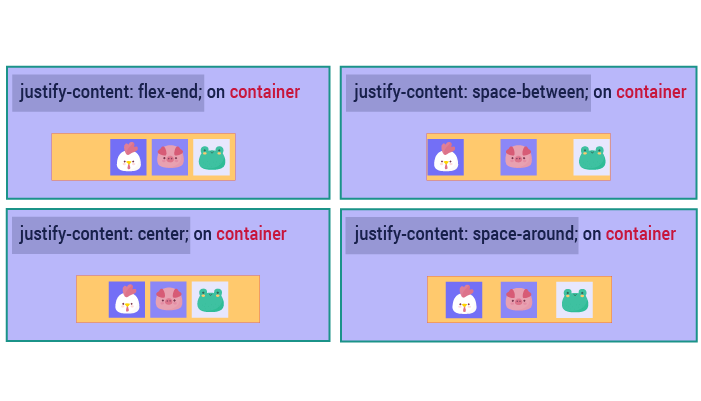
By default, flex items are laid out in their source order. Think of flex items as primarily laid out either in horizontal rows or vertical columns.,  
Therefore, if we want to have rows or columns, you apply the following rules:



# Justify content

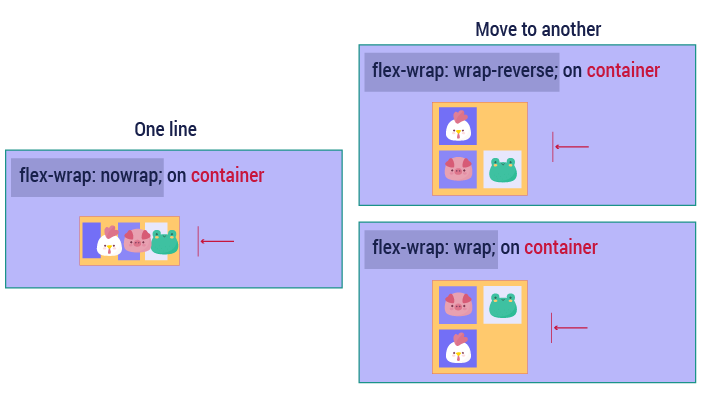
It defines how the elements are aligned along the main axis.

It helps distribute extra free space when either all the flex items on a line are inflexible, or they’re flexible but have reached their maximum size.



# Flex-wrap:

One of flex’s properties is that all items will automatically try to fit into one line.  
You can change that and allow the items to wrap as needed. So we have two choices: either we decide for them to be in one line or move to another line.



# Align-items:

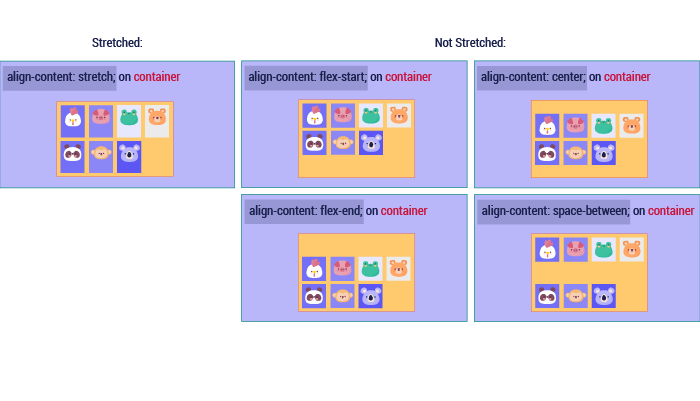
This determines the default behavior for how flex items are laid out along the cross axis on the current line. Think of it as the justify-content version for the cross axis.



# Align-content:

What if now we have multiple lines of content and we want to make it aligned?  
Piece of cake! We naturally use the align-content property.

Keep in mind, this property has no effect when there is only one line of flex items.

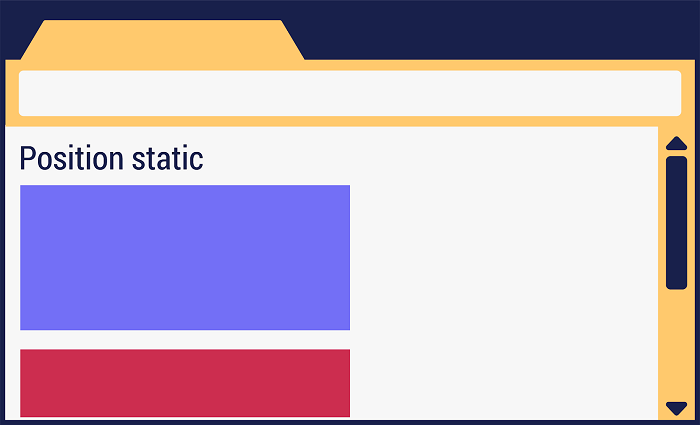


# Static position

The **Static** Position is the default positioning for all the elements on a web page.

The elements are fixed objects within the scene until you decide for them to be a part of a positioning context.

Static means elements are static according to the normal flow of the page. If one moves, another will move as well.



# Position absolute

By default, all HTML elements are in a static position. The element is not positioned in any special way. It is always positioned according to the normal flow of the page.  
In this case the parent is given position: static, and the child is given position: absolute and according to that, it will be absolute to the parent element that has a position of relative, absolute or fixed.  
If there are none, it will be absolute to the page. It means that it won’t be affected by anything around it.

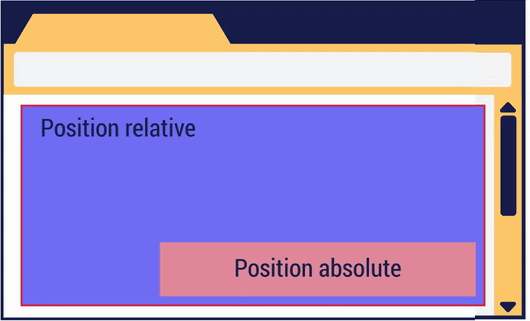


# Position relative

As we have already said, elements are statically positioned.

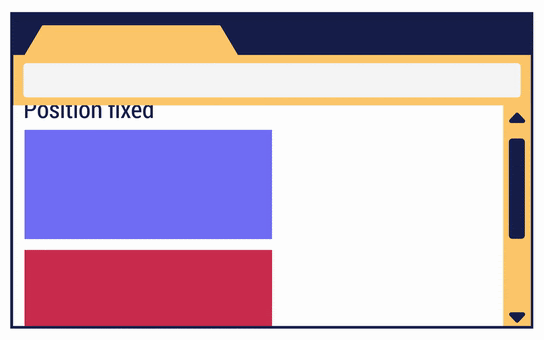
In this case, the parent is given a value of position:relative, and the child is given a value of position:absolute.

When you style an element with position: relative, it creates a positioning context for every child element within the element.  
In this way, you can go ahead and accurately position the child element with respect to any sides of the reference object.  
Relatively positioned elements behave exactly like static ones, but they serve as a local frame of reference for absolutely positioned child elements.



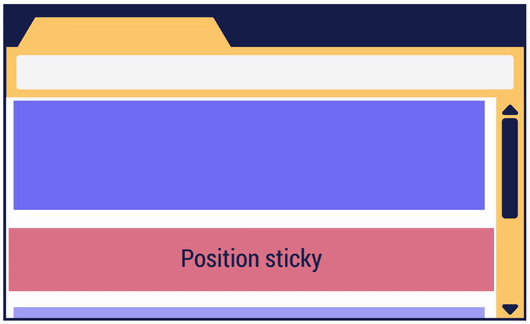
# Position fixed

If we position our element as **fixed**, it means the position will always be locked to the size of the browser's window.  
If you scroll up ,down ,left or right, the element will stay in its position.



# Position sticky

An element with position: sticky is positioned based on the user's scroll position.  
In other words, as soon as the scroll reaches the sticky element, it will stick to the screen.

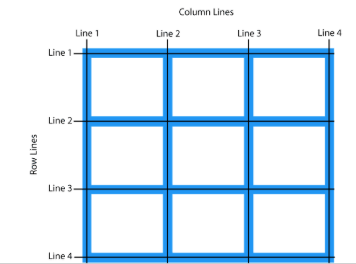


# CSS grid system

Every website or application layout you make (or have seen) is essentially boxes placed within certain defined boundaries.

In very simple terms, a grid is just horizontal and vertical lines that define the placement of other design elements.

In the context of the CSS Grid layout, a grid container is a parent that contains all items within it. The grid container defines the initial placement of the grid lines, both vertical and horizontal.



So how can we define a grid?

Just like flexbox, everything begins with a one-line **display:grid** or **display:inline-grid** for an inline version.

For example, to make a certain div a grid container,

* **The CSS code would be :**

div {

display: grid;

}

Column and Row

First we have to define columns and rows, because a grid without them is entirely pointless.

To create columns and rows within a grid container, you use these two new CSS properties: grid-template-columns and grid-template-rows.

So how do you use them? Pretty simple.

grid-template-columns determines the placement of the columns while grid-template-rows determines the placement of the rows within the grid container.

All we need is to type in the length into these properties,

* The CSS code would be :

.grid-container {

display: grid;

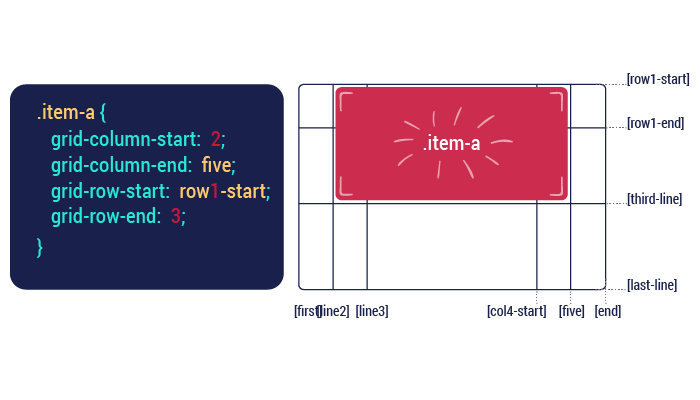
grid-template-columns: 100px 200px 300px;

grid-template-rows: 100px 200px 300px;

}

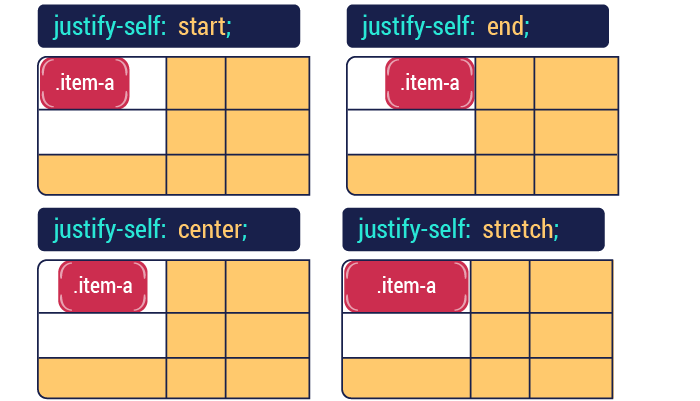
# Properties of the grid items

We have :  
grid-column-start/grid-row-start is the line where the item begins.  
grid-column-end/grid-row-end is the line where the item ends.



# Justify-self properties

Aligns a grid item inside a cell along the inline. It can have the value: center,end,start and stretch.



# What is responsive design?

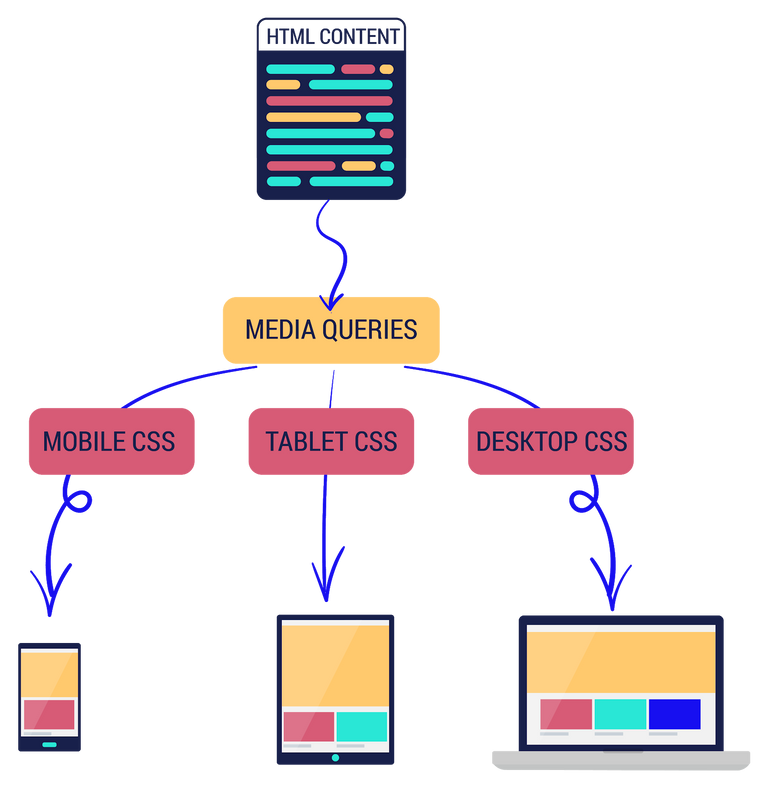
Responsive web design is an approach whereby a designer creates a web page that is highly interactive with the user. Meaning it “responds to” or resizes itself depending on the type of device it is being displayed on and depending on the user’s behaviour and preferences.

The devices could be a large desktop computer monitor, a laptop or devices with small screens such as smartphones and tablets.

Responsive web design has become an essential tool for anyone with a digital presence. With the growth of smartphones, tablets and other mobile computing devices, more people are using smaller-screens to view web pages.

# How can we make it responsive?

Responsive design is accomplished through CSS media queries. Think of media queries as a way to conditionally apply CSS rules. They tell the browser that it should ignore or apply certain rules depending on the user’s device.



What is media query?

Media queries are at the heart of responsive design. They let you target specific screen sizes and specify css rules that will be executed on that screen.

The most popular form in which media queries are used is something called the @media rule.

* The CSS code would be :

@media screen and (max-width: 300px) {

*/\* write your CSS in this code block \*/*

}

What does it really do?

So to translate the code below, it means “For a screen device with a maximum width of 300px … implement the following actions”  
Any styles within the code block will only apply to devices that match the expression, screen and (max-width: 300px) i.e. screen devices with a maximum width of 300px.

* CSS:

@media screen and (max-width: 300px) {

*/\* write your CSS in this code block \*/*

}

# Media query breakpoints

**What is a breakpoint?**  
CSS breakpoints are points where the website content responds according to the device width, allowing you to show the best possible layout to the user.

**These are the most used breakpoints:**

*/\* Extra small devices \*/*

@media (max-width: 576px) { */\* ... \*/* }

*/\* Small devices (landscape phones, 576px and up) \*/*

@media (min-width: 576px) and (max-width: 768px) { */\* ... \*/* }

*/\* Medium devices (tablets, 768px and up) \*/*

@media (min-width: 768px) and (max-width: 992px) { */\* ... \*/* }

*/\* Large devices (desktops, 992px and up) \*/*

@media (min-width: 992px) and (max-width: 1200px) { */\* ... \*/* }

*/\* Extra large devices (large desktops, 1200px and up) \*/*

@media (min-width: 1200px) { */\* ... \*/* }

# Font-Awesome

Well, we will be sharing with you some helpful and useful resources to make your web page look “awesome”.  
Time to meet Font-Awesome! It is the most popular way to add font icons to your website.  
Font Awesome icons are created using scalable vectors, so you can use high quality icons that work well on any screen size.



## **How to use it?**

First, we add the following line inside the < head > section of your HTML page:

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css">

Then, to display cool icons we simply use the material you will find in <https://fontawesome.com/>. You will find everything you will need on that website. Here’s an example:

* The HTML code would be :

<i class="fa fa-car"></i>

<i class="fa fa-car" style="font-size:48px"></i>

<i class="fa fa-car" style="font-size:60px; color:blue"></i>

* The output would be :



# font-face



Let’s get to know another useful tool, it’s called the font-face rule.

With font-face , Web developers or designers, will have the freedom to use new and interesting fonts, and throw the classic and dull fonts right out the window.

It’s very simple to use the font-face rule, first you must define a name for the font (example: Chathura Bold) Then, point it to the font file ( Src: url(../Path to your font file).

And then you simply use as a normal font in your CSS file.

@font-face {

font-family: "chathurabold";

src: url("../font/chathura\_bold\_macroman/chachathura-bold-webfont.eot");

font-weight: normal;

font-style: normal;

}

# What is bootstrap?



Remember when we spoke about responsive design? Well, time to meet Bootstrap. He will be your responsive design guide.

### **How Bootstrap will make your life easier?**

Bootstrap code is already written and is available for everyone to use. So, we’ll be treating it like a library and borrow only what we need from it.  
Everything we need is already on the Internet. We just have to be selective, thorough and pick out only what we need. This is what’s called “Snippeting”.

Snippeting is basically copying code from another source and pasting it to your code. It is ethical and non-problematic in Bootstrap, no need to worry.

### **How to use it?**

During this part, we will not get into all the classes, but we are going to mention some interesting ones. However, you can find more information on Bootstrap’s official website. Feel free to try new things.  
Like every library, to have access to its elements, you must add the following line inside the <head> section of your HTML page:

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">

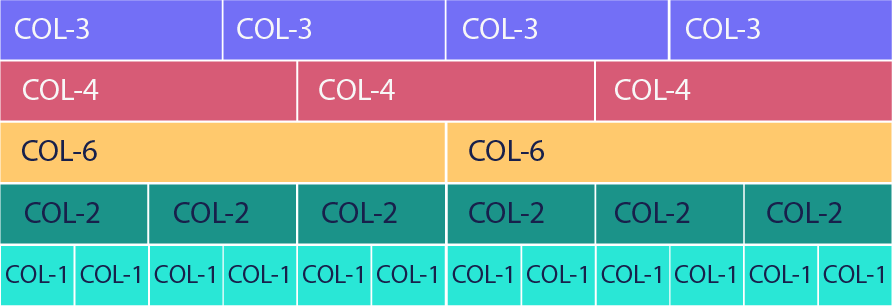
Bootstrap grid system

One of Bootstrap’s most important assets is that it makes your website responsive using its grid system.

The main idea behind Bootstrap is to divide the screen into twelve equal columns, by doing that it gives us a new unit that works independently from size of screen.

There are four types of resolutions (or classes) in Bootstrap:

* xs (for phones - screens less than 768px wide)
* sm (for tablets - screens equal to or greater than 768px wide)
* md (for small laptops - screens equal to or greater than 992px wide)
* lg (for laptops and desktops - screens equal to or greater than 1200px wide)



# How to use it?

## **First step**

To apply Bootstrap’s grid system in our web page, we need to use Bootstrap’s container and rows.  
First thing we need to do is create a div with class=”container”. It is as essential as the body tag when we’re dealing Bootstrap.

So, our Bootstrap must be inside the div with the class container. It makes a margin on the right and on the left so that our content is centred.

Added to that, If we want our page to take up all of the screen, we write container-fluid.

<body>

<div class="container">

*<!-- Some BootStrap Classes here -->*

</div>

</body>

# How to use it?

## **The second step:**

After creating the container, we can add rows.  
Rows, like the following picture demonstrates, means a line across the screen. That row will be divided into twelve equal parts, where we can put our HTML elements and that will replace the margin & padding.

* Our HTML code would be:

<div class="container-fluid">

<div class="row" style="background-color:lightblue">

First row

</div>

<hr />

<div class="row" style="background-color:#e5cf0d">

Second row

</div>

<hr />

<div class="row" style="background-color:#b30de5">

Third row

</div>

</div>

* Our output would be:



How does it works?

So, how does the grid system work?

Each row that we add can be divided into 12 equal parts called "columns", and from this moment onwards, we can say terms like: "this element should take 3 columns on a big screen and 12 columns on a small screen."

Since the row is divided into 12 columns, if an element gets 12 columns of width, it will be as wide as the whole screen. If it takes 6, then it’s as wide as half the screen. Here’s an example, col-8 will take 8 columns, and col-4 will take 4 columns.

* HTML

<div class="row">

<div class="col-8">col-8</div>

<div class="col-4">col-4</div>

</div>

* Output:

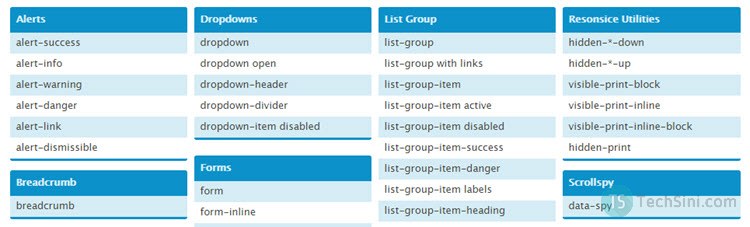


Bootstrap classes

As we have already mentioned, Bootstrap is a massive collection of reusable and versatile pieces of code which are written in CSS, HTML and JavaScript. With that said, developers have a range of components and tools at their disposal, which can be used on the website. These include, but are not limited to:

* Drop-down menus
* Navigation bars
* Progress bars
* Thumbnail images

Let’s discover some of these free tools.



Buttons

So in HTML, buttons are basic, dull and uninteresting. They also don’t have a meaning behind them. Why not improve them, make them more awesome and more advanced with Bootstrap?

* HTML code

<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>

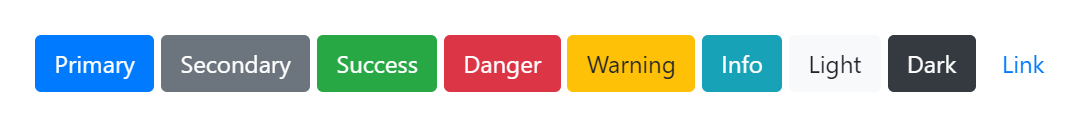
<button type="button" class="btn btn-info">Info</button>

<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>

* Output:



Alert

Alerts can be used in different situations. Let’s introduce some of them. You can imagine each scene yourself.

* HTML code

<div class="alert alert-primary" role="alert">

This is a primary alert—check it out!

</div>

<div class="alert alert-secondary" role="alert">

This is a secondary alert—check it out!

</div>

<div class="alert alert-success" role="alert">

This is a success alert—check it out!

</div>

<div class="alert alert-danger" role="alert">

This is a danger alert—check it out!

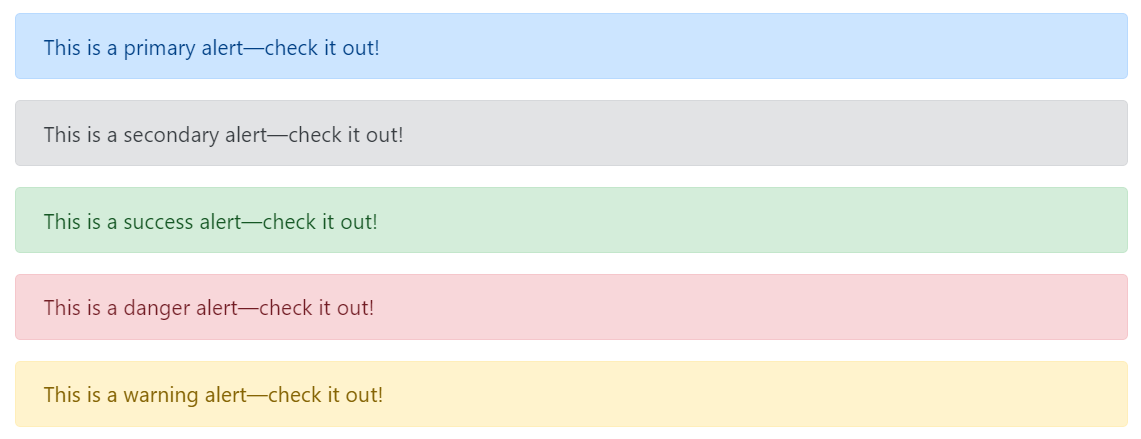
</div>

<div class="alert alert-warning" role="alert">

This is a warning alert—check it out!

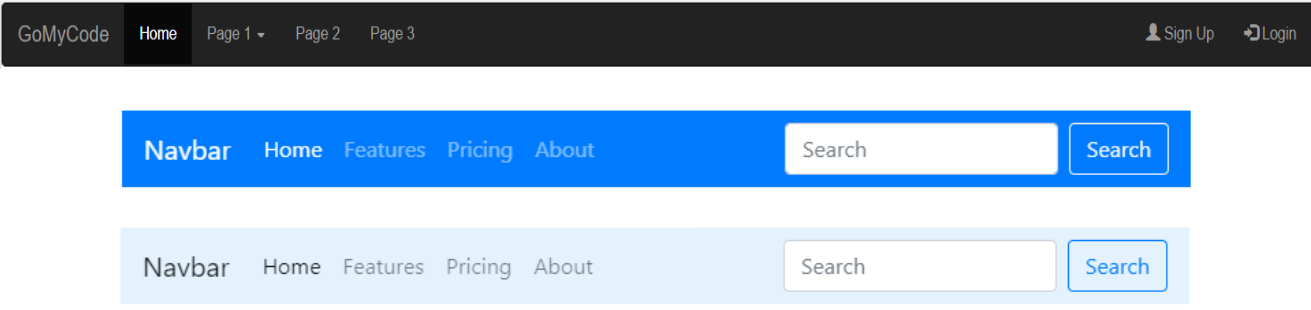
</div>

* Output:



# Navbar

Bootstrap also offers a large range of navigation bars to make things easier.  
(A navigation bar is a navigation header placed at the top of the page.)  
Here’s an example: (You will find the needed code in the Bootstrap’s official website)



Progress bar

Another useful feature is progress bars. They can be used for creating your resume page.  
For example:

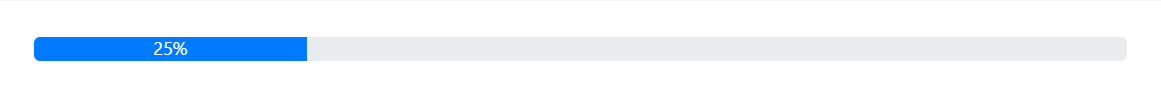
* HTML

<div class="progress">

<div class="progress-bar" role="progressbar" style="width: 25%;" aria-valuenow="25" aria-valuemin="0" aria-valuemax="100">25%</div>

</div>

* Output:



CSS Layout RECAP

Quick reminder that you’re doing great for making it this far!

By now, we have learned :

* How to create a web page using HTML, how to style it using CSS.
* How to use external useful tools like Bootstrap, Font-Awesome...Etc.
* How to make our web pages responsive, intuitive and highly user-friendly.

However, we can’t help but notice that something is missing. The web page is lacking certain functions like a search box or a comment section.

Right now, the body, with all its accessories, looks more like an immovable and emotionless mannequin in a storefront than a real human being. That’s where JavaScript comes in.

Let’s learn more about Javascript in the next Super skill.