# Demo Script: 1.1.9 HTML Containers

* [Replit Demo](https://replit.com/@mrrocampbell/1-1-9HTML-Containers#index.html)
* [Word Version of Script](https://chegg-my.sharepoint.com/:w:/p/rocampbell/EQRbRCobKDlOnJJkAfzEV5kBkFOeZOmTW15Td42SMaSS7A?e=19nSKr)
* [Loom Recording](https://www.loom.com/share/b48966ea62d6422ea5343e5fc67f6f9d)
* [Contentful lesson](https://overview.thinkful.com/preview/JS-MODULAR/version/1/introduction-to-web-development/guild-b2b-eng-module-introduction-html-css-fundamentals/modeng-html-containers)
* Jira Tickets:
  + [TFENG-2318: 1.1.9\_intro\_html\_css\_fundamentals\_html\_containers: DemoVideo: Write Script](https://chegg.atlassian.net/browse/TFENG-2318)

## Lesson Learning Objective

By the end of this lesson, you will be able to organize content on a web page using container elements.

## Script

Hey, this is Roger from Thinkful. In this video, we will look at why it is important to use semantic container within our HTML to develop code that is easy to read.

A semantic container clearly describes the type of content it will contain to both the browser and developers. An example of a semantic container is <header></header> whereas a non-semantic container is a <div></div>.

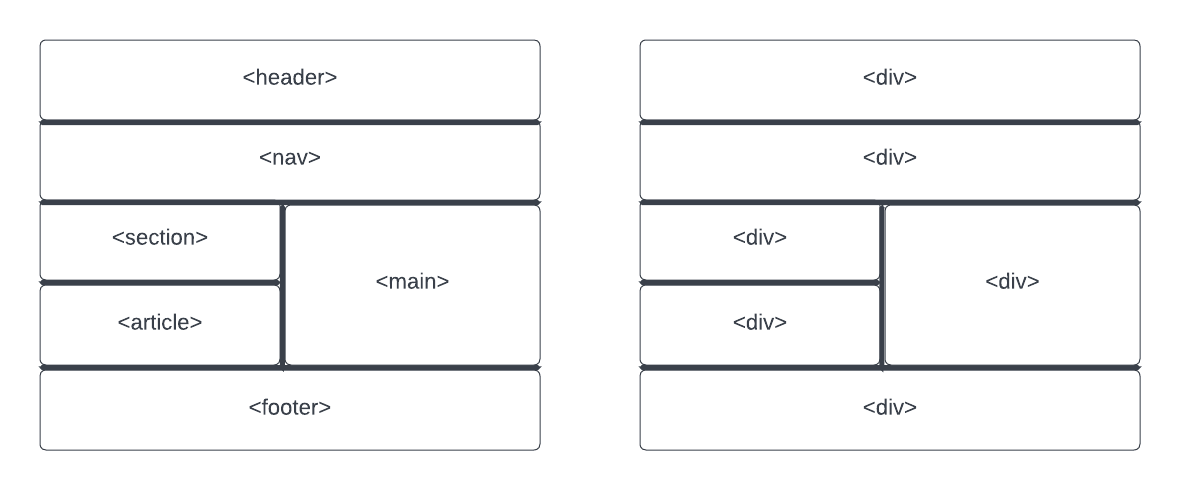
The problem with using non-semantic containers such as a <div></div> is that they do not provide any insight into what content they will hold.

Where as a semantic container such as a <header></header> tells a developer that the content within this container will should be for the header portion of the website.

Providing this kind of insight is very important because as an developer you want to write code that is easy to read, understand, and in turn reusable.

Part of achieving these goal is by writing code that others can easily read and make changes. Which can be done with semantic elements and containers.

Take a look at this diagram which displays a visual representation of a website.



Sematic Containers VS Non-Sematic Containers

On the left we see that by using sematic containers such as header, nav, section, aside, and footer we are able to clearly define what content each section of the website will contain.

In comparison to the diagram on the right which only utilizes divs, which makes it very hard to know what information is going to be provided in each container.

Even though both diagrams would display the same layout via the browser the one on the left makes it much easier to understand the layout that was intended for the website and the information that should be contained in each portion of the site.

For further reference lets take a look at the code version of this diagram.

<div id="sematic-container">  
 <h2>Sematic Containers</h2>  
 <header>Header</header>  
 <nav>Nav</nav>  
 <section>Section</section>  
 <aside>Aside</aside>  
 <main>Main</main>  
 <footer>Footer</footer>  
 </div>  
  
  
 <div id="non-sematic-container">  
 <h2>Non-Sematic Containers</h2>  
 <div class="header">Div</div>  
 <div class="nav">Div</div>  
 <div class="section">Div</div>  
 <div class="aside">Div</div>  
 <div class="main">Div</div>  
 <div class="footer">Div</div>  
 </div>

html, body {  
 height: 100%;  
 width: 100%;  
}  
  
body{  
 display: flex;  
 justify-content: space-around;  
}  
  
#sematic-container{  
 display: inline-grid;  
 grid-template-columns: repeat(2, 10em);  
 grid-template-rows: repeat(6, 5em);  
 column-gap: 5px;  
 row-gap: 5px;  
}  
  
header, nav, section, main, aside, footer {  
 text-align: center;  
 display: flex;  
 justify-content: center;  
 align-items: center;  
  
}  
  
h2{  
 grid-column: 1 / span 2;  
 grid-row: 1;   
 text-align: center;  
}  
  
header{  
 grid-column: 1 / span 2;  
 grid-row: 2;  
 border: solid blue 3px;  
}  
  
nav{  
 grid-column: 1 / span 2;  
 grid-row: 3;  
 border: solid black 3px;  
}  
  
section{  
 grid-column: 1 / span 1;  
 grid-row: 4;  
 border: solid red 3px;  
}  
  
aside{  
 grid-column: 1 / span 1;  
 grid-row: 5;  
 border: solid green 3px;  
}  
  
main{  
 grid-column: 2 / span 1;  
 grid-row: 4 / span 2;  
 border: solid rosybrown 3px;  
}  
  
footer{  
 grid-column: 1 / span 2;  
 grid-row: 6;  
 border: solid purple 3px;   
}  
  
/\* End of Sematic Styling \*/  
  
/\* Start of Non- Sematic Styling \*/  
#non-sematic-container {  
 display: inline-grid;  
 grid-template-columns: repeat(2, 10em);  
 grid-template-rows: repeat(6, 5em);  
 column-gap: 5px;  
 row-gap: 5px;  
}  
  
.header, .nav, .section, .main, .aside, .footer {  
 text-align: center;  
 display: flex;  
 justify-content: center;  
 align-items: center;  
  
}  
  
.header{  
 grid-column: 1 / span 2;  
 grid-row: 2;  
 border: solid blue 3px;  
}  
  
.nav{  
 grid-column: 1 / span 2;  
 grid-row: 3;  
 border: solid black 3px;  
}  
  
.section{  
 grid-column: 1 / span 1;  
 grid-row: 4;  
 border: solid red 3px;  
}  
  
.aside{  
 grid-column: 1 / span 1;  
 grid-row: 5;  
 border: solid green 3px;  
}  
  
.main{  
 grid-column: 2 / span 1;  
 grid-row: 4 / span 2;  
 border: solid rosybrown 3px;  
}  
  
.footer{  
 grid-column: 1 / span 2;  
 grid-row: 6;  
 border: solid purple 3px;  
}

Now before we look at this example, I want to say that there is quite a bit of code here. I highly recommend that once I explain this portion you pause the video and read through the code yourself. Another thing to note is that I have added some custom css styling to make this code display like the diagram I showed earlier. This is not something I would like you to concern yourself with at the moment, the point of this example is for you to be able to read the code and see that by utilizing sematic containers your code becomes much easier to read versus non-sematic containers.

So, to the example; as you can see the code with in the sematic-container div is much easier to read and understand what is going on rather than the code within the non-sematic-container container.

By utilizing sematic container I we are easily able to see which container is going to be used for what. It also makes selecting each container in our css easier.

Whereas with the non-sematic-container I must add a class to each div so that I am able to provide information for what is going to be stored within each one. I also have to use classes to handle selecting each container within my css.

Again do not worry about how the css is working here but pay attention to which set is easier to read.

We covered a lot in this video so at this point I want you to press pause, read through the code, and then if you’d like re-watch explanation on the code again.

Thanks for watching! I’ll see you in the next video!

## Notes for Editor

Provided any notes for post-production team