Basic Layouts

CSS layouts have changed the way that we are able to design websites. At one time all layouts were done with the <table>. This was both slow and hard to manage.

Table free layouts have been used for almost a decade now while they are an improvement over the <table> they have flaws.

Structure

In this example we are looking the the most basic layout. Assume that this page is a standard content page that has a *header*, *navigation*, *content area* and *footer*.

```
<header></header>
<nav></nav>
<article></article>
<footer></footer>
```

The issue now is that we have no way to limit the width of the elements. One option would be with set the width on each element there are some issue that come along with that. The box model kicks in when a width is set.

Add Wrapper

The other option is to wrap all the element with an other element like a . This will allow you to define a width on it.

```
<div>
    <header></header>
    <nav></nav>
    <article></article>
    <footer></footer>
</div>
```

Adding IDs

While the structure is correct at this point we want to add an id or class to help identify this <header> from any other <header> that may be on the page.

There are some who say you should never use an id because they are too specific. I disagree with this. I feel that a well place id can help identify which element you are targeting.

```
<div id="container">
    <header id="globalHeader"></header>
    <nav id="globalNav"></nav>
    <article id="contentArea"></article>
    <footer id="gloablFooter"></footer>
</div>
```

Styles

Now that we have the structure in place we need to write the css to support the markup.

Fixed width

In this example we are setting up a page that is a fixed width of 800px.

```
<style>
    #container {
        width: 800px;
    }

    #gloablNav, #contentArea {
        float: left;
    }

    #gloablNav {
        width: 200px;
    }

    #contentArea {
        width: 600px;
    }

    #footer {
        clear: both;
    }
</style>
```

This will create a fixed two column layout. The width can be adjusted any value just note that the #globalNav and #contentArea must add up to the width of the #container.

Flex Width

There are times where we want the display to be flexible. In this example we will keep the #globalNav a fixed width of width: 200px;.

To make this work we need to keep the nav a fixed width and have the content be flexible. The issue is that once we float an element it will take up as much space as needed unless a width is set. To get around this we will change things up a bit. Rather than float both elements, we will float only the nav and offset the content by the with of that nav.

```
<style>
    #gloablNav {
        width: 200px;
        float: left;
    }

    #contentArea {
        margin-left: 200px;
    }

    #footer {
        clear: both;
    }
</style>
```

Variable Width

In the examples above we had a fixed width of the container. In one example it was 800px the other fit the width of the screen. In this example we will limit the width to be variable width of anything from 700px to 1200px.

```
<style>
    #container {
        min-width: 700px;
        max-width: 1200px;
    }

    #gloablNav {
        width: 200px;
        float: left;
    }

    #contentArea {
        margin-left: 200px;
    }

    #footer {
        clear: both;
    }
</style>
```

Center Container

The problem with centering #container on the page is there is no real property that does this. Your first thought might be text-align: center; but that will just center the text within the elements.

By setting the margin on the left and right to auto the element will be centered on the page. Because we set the left and right to auto it take the space left over after the and divide it by 2 and set it equal to each other. This will make the element center on the page.

```
<style>
#container {
    width: 800px;
    margin: 0 auto;
}
</style>
```