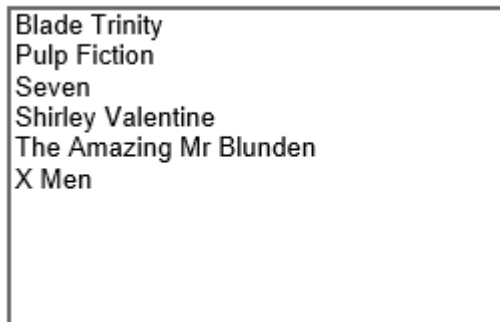


## Creating Dynamic Tables

One important feature of any application is the ability to generate lists of data.

The DVDSwap has one example of a list on the main page of the application.

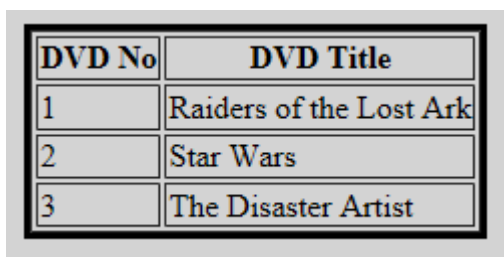


There is something to note though about this type of list. Since the application's interface is built around web forms, this list is an ASPX control. The problem is that on the majority of web sites, this list is almost never used.

If we are to create the list in HTML 5 then we need to think about doing this using HTML 5 mark-up rather than ASPX controls.

What we will do is create some dummy middle layer classes and then use those classes in the HTML 5 presentation layer displaying the data in an HTML 5 table.

Like so...



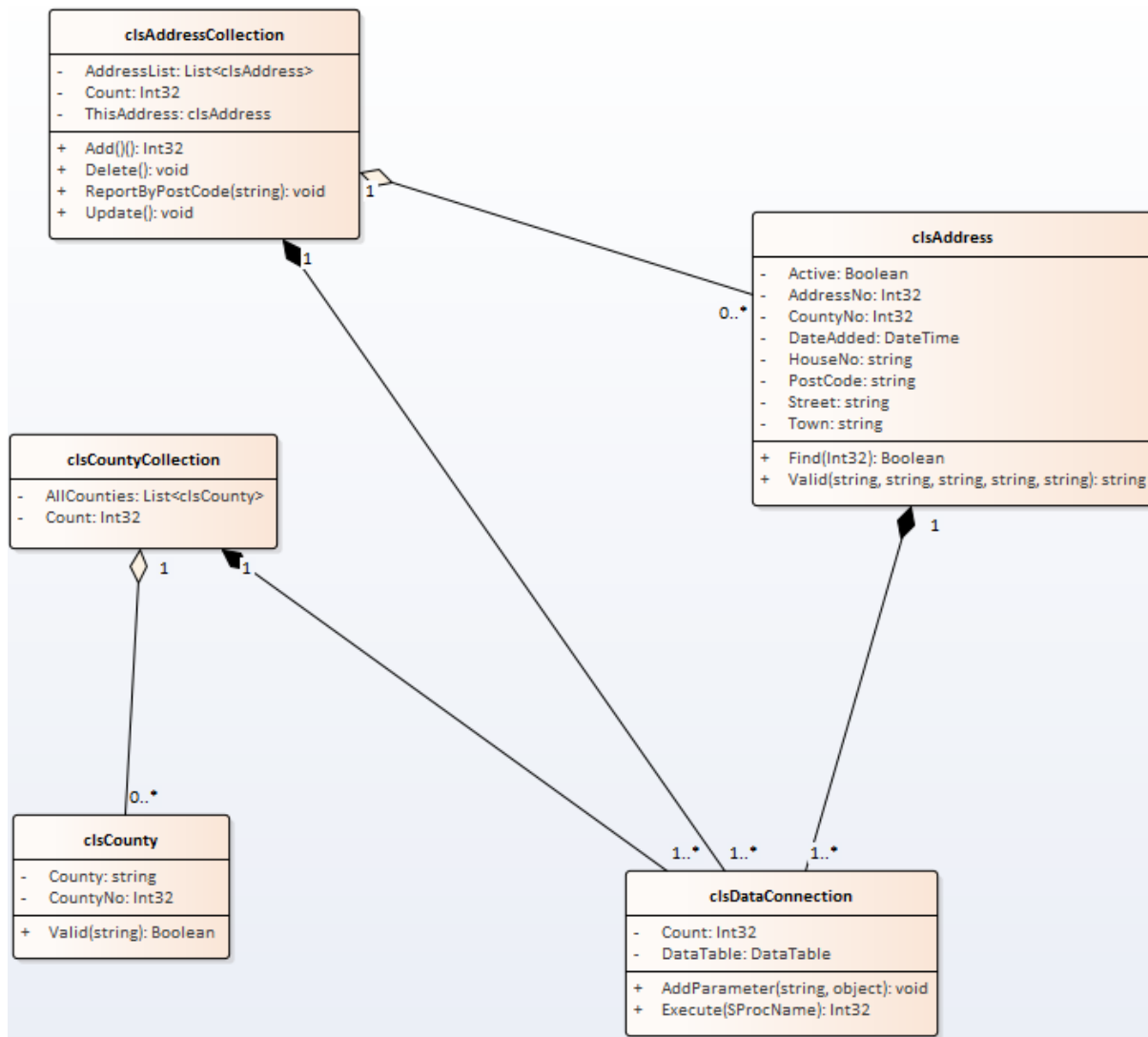
| DVD No | DVD Title               |
|--------|-------------------------|
| 1      | Raiders of the Lost Ark |
| 2      | Star Wars               |
| 3      | The Disaster Artist     |

### *Creating the Dummy Classes*

If we want to represent a list in code then we will need two classes.

1. A Collection Class
2. An Item Class

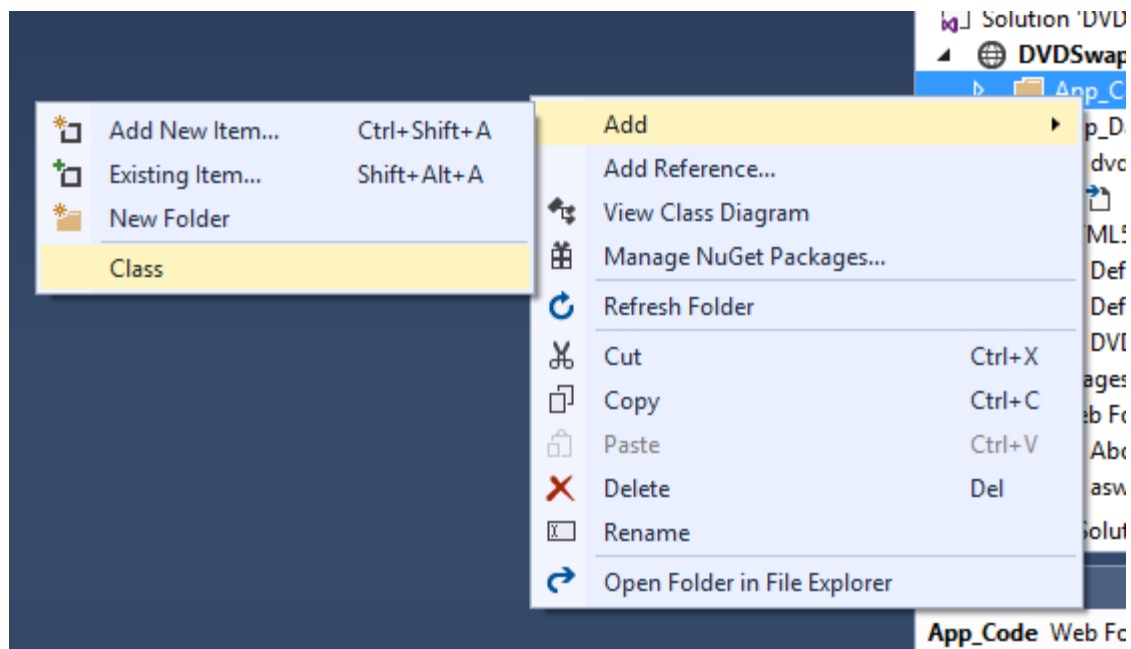
In this class diagram below we can see each, clsAddressCollection and clsAddress.



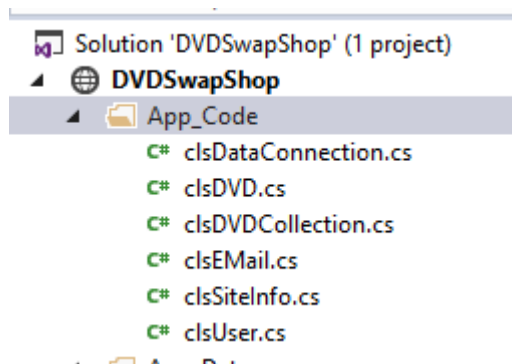
The thing is for this work we want to concentrate on the presentation layer and not spend loads of time on creating the database and the code for the middle layer.

What we will do instead is create two dummy classes that simulate the presence of a data layer even when there isn't one.

Within DVD Swap create two new classes called clsDVD and clsDVDCollection. Just to remind you right click on App\_Code and select add class.



You should end up with two classes like so...



As I said, we don't want to spend hours writing code so we shall create just two properties in clsDVD, the primary key and the DVD title like so...

```

private Int32 mDVDNo;
public Int32 DVDNo
{
    get
    {
        return mDVDNo;
    }
    set
    {
        mDVDNo = value;
    }
}

```

```

private string mTitle;
public string Title
{
    get
    {
        return mTitle;
    }
    set
    {
        mTitle = value;
    }
}

```

Close the class file and save it.

We will now edit the collection class to generate a list of DVDs.

Rather than putting in the effort of creating a data layer and looping through each record we shall do the following.

First we will create a public property for the DVDList like so...

```

public List<clsDVD> DVDList
{
    get
    {
    }
}

```

We will now generate the array list of DVDs like so...

```

public List<clsDVD> DVDList
{
    get
    {
        //create and array list of DVDs
        List<clsDVD> mDVDList = new List<clsDVD>();
        //create a single DVD
        clsDVD SomeDVD = new clsDVD();
        SomeDVD.DVDNo = 1;
        SomeDVD.Title = "Raiders of the Lost Ark";
        //add it to the array list
        mDVDList.Add(SomeDVD);
        //create another DVD
        SomeDVD = new clsDVD();
        SomeDVD.DVDNo = 2;
        SomeDVD.Title = "Star Wars";
        //add it to the array list
        mDVDList.Add(SomeDVD);
        //create another DVD
        SomeDVD = new clsDVD();
        SomeDVD.DVDNo = 3;
        SomeDVD.Title = "The Disaster Artist";
        //add it to the array list
        mDVDList.Add(SomeDVD);
        //return the populated list
        return mDVDList;
    }
}

```

Rather than looping through records in a database we are hard coding each DVD title one after the other and then returning the list. Not a very good way of generating 200 records in the list but fine for what we want to do, the presentation layer will not know the data isn't coming from a database.

The last property we need to create is the Count property like so...

```

public Int32 Count
{
    //hard coded count property
    get
    {
        return 3;
    }
}

```

And that's all the coding we plan to do for this example, we need to look at the presentation layer.

Assuming you completed last week's work, you should have a presentation layer that looks something like this...

|   |  |
|---|--|
| <p style="text-align: center;"><b>Fred's DVD Swap</b><br/><i>Swapping DVDs Since 2016</i></p> | <p>E-Mail Address<br/><input type="text"/></p> <p>Password<br/><input type="password"/></p> <p><input type="submit" value="Submit"/></p> |
| Empty content area  |  |

In this example I have left borders on Nav, Article and Header to make it easier to see where they are.

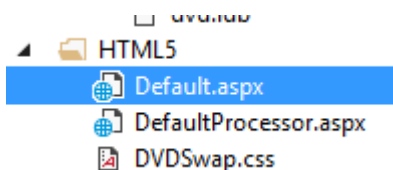
### Adding Server Side Code

The problem with an HTML page is that any code we embed is completely ignored by the server.

What we want to happen is that the server processes the page, runs any C# code in the page and then delivers the resulting page to the browser.

To achieve this we can't use a simple HTML file, we need to change the file extension to .ASPX.

Right click on the page Default.html and rename the extension to .aspx like so...



The file is still an HTML 5 file, however changing the file extension means that the server will look at the file before sending it to the browser and execute any C# code it finds.

We are also going to have to modify the header of the page such that it knows what language to use and mark up an area to run server side scripts, like so...

```

Default.aspx*  [icon] [X]
<%@ Page Language="C#" %>

<!DOCTYPE html>

<script runat="server">
</script>

<html>
<head>

```

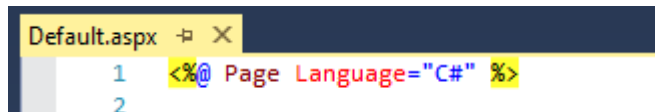
The server now knows any code it finds will be C# and also has an area where we may add C# server side scripts.

## *Adding the C# Code*

Next we want the data from the middle layer to be presented dynamically in a table.

This means that we will need to add some C# code to the page so that it displays the data for us.

This being a dynamic page we have already told it what language it is to use at the top line...



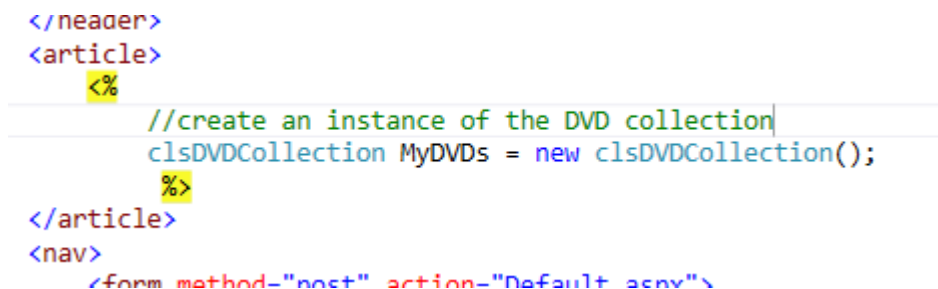
```
Default.aspx
1  <%@ Page Language='C#' %>
2
```

Notice how this line is enclosed by <% %>!

This tells the web server that this is a section of embedded server side code that needs to be processed before the page is sent to the browser.

We can use these tags to mark up C# code as we see fit.

In this example we will add some code into the article tag like so...



```
</header>
<article>
  <%
    //create an instance of the DVD collection
    clsDVDCollection MyDVDs = new clsDVDCollection();
  %>
</article>
<nav>
  <form method='post' action='Default.aspx'>
```

We have now added a section of C# code that will execute before the server sends the page to the browser.

To extend this (so that we can see something happening) modify the code like so...

```

</head>
<body>
    <%
        //create an instance of the DVD collection
        clsDVDCollection MyDVDs = new clsDVDCollection();
        //create the index variable initialised at 0
        Int32 Index = 0;
        //get the count of records
        Int32 RecordCount = MyDVDs.Count;
        //loop through each DVD
        while (Index < RecordCount)
        {
            //write the dvd no to the browser
            Response.Write(MyDVDs.DVDList[Index].Title);
            //increment the index
            Index++;
        }
    %>
</body>
</html>

```

Giving us this rather odd looking output...

|  |  |   |
|--|--|---|
| Raiders of the Lost ArkStar WarsThe Disaster Artist<br><b>Fred's DVD Swap</b><br><i>Swapping DVDs Since 2016</i> |  | E-Mail Address<br><input type="text"/><br>Password<br><input type="password"/><br><input type="button" value="Submit"/> |
|  |  |   |

The problem is that we need to organise the data such that it is easier to read. In this example we shall create an HTML table and then apply some styling to change the appearance.

With this approach we are now able to mix and match between C# and HTML, but we do need to be careful about how we structure the code.

The first question to consider is where does the table start and end? We actually want the table to start before the loop begins and ends, as it is the loops job to generate the rows and the columns of the table.



Take a look at the following code...

```
<%  
    //create an instance of the DVD collection  
    clsDVDCollection MyDVDs = new clsDVDCollection();  
    //create the index variable initialised at 0  
    Int32 Index = 0;  
    //get the count of records  
    Int32 RecordCount = MyDVDs.Count;  
    //loop through each DVD  
    %><table><%  
    while (Index < RecordCount)  
    {  
        //write the title to the browser  
        Response.Write(MyDVDs.DVDList[Index].Title);  
        //increment the index  
        Index++;  
    }  
    %></table><%  
    %>
```

In this example by using the <% %> tags we are able to switch between C# and HTML code, the <table> tags being written using HTML.

Having decided where the table starts and ends, we need to decide where each row of the table should be generated.

The loop will loop round once for each record. Since each row of the table displays one record we can assume that a single iteration equates to one record.

This means we need to generate the table rows like so...

```

<%
//create an instance of the DVD collection
clsDVDCollection MyDVDs = new clsDVDCollection();
//create the index variable initialised at 0
Int32 Index = 0;
//get the count of records
Int32 RecordCount = MyDVDs.Count;
//loop through each DVD
%>
<table>
<%
while (Index < RecordCount)
{
    %>
    <tr>
    <%
    //write the dvd title to the browser
    Response.Write(MyDVDs.DVDList[Index].Title);
    //increment the index
    Index++;
    %>
    </tr>
    <%
}
%></table><%
%>

```

The last question is where do we generate the columns for the table?

For a single iteration of the loop it needs to write several fields from the record.

The table columns need to be generated like so...

```

<%
//create an instance of the DVD collection
clsDVDCollection MyDVDs = new clsDVDCollection();
//create the index variable initialised at 0
Int32 Index = 0;
//get the count of records
Int32 RecordCount = MyDVDs.Count;
//loop through each DVD
%><table><%
while (Index < RecordCount)
{
    %><tr><%
    %><td><%
    //write the title to the browser
    Response.Write(MyDVDs.DVDList[Index].Title);
    %></td><%
    %></tr><%
    //increment the index
    Index++;
}
%></table><%
%>

```

You should see output generated in the browser like so...

|   |   |   |
|---|---|---|
| Raiders of the Lost Ark<br>Star Wars<br>The Disaster Artist | <b>Fred's DVD Swap</b><br><i>Swapping DVDs Since 2016</i> | E-Mail Address<br><input type="text"/><br>Password<br><input type="password"/><br><input type="button" value="Submit"/> |
|   |   |   |

One little trick to try at this point is to add a temporary border to the table. We won't keep this but it can be useful to help spot any errors.

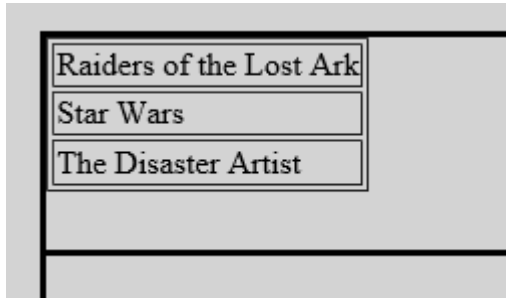
Modify the table tag like so...

```

...
Int32 Index = 0;
%>
<table border="1">
<%
//loop through each record in the cc
while (Index < RecordCount)
{
    _

```

Produces the following effect...



|                         |
|-------------------------|
| Raiders of the Lost Ark |
| Star Wars               |
| The Disaster Artist     |

Now take a look at the following code which displays the primary key of each record...

```

<%
//create an instance of the DVD collection
clsDVDCollection MyDVDs = new clsDVDCollection();
//create the index variable initialised at 0
Int32 Index = 0;
//get the count of records
Int32 RecordCount = MyDVDs.Count;
//loop through each DVD
%><table border="1"><%
while (Index < RecordCount)
{
    %><tr><%
    %><td><%
    //write the title to the browser
    Response.Write(MyDVDs.DVDList[Index].DVDNo);
    %></td><%
    %><td><%
    //write the title to the browser
    Response.Write(MyDVDs.DVDList[Index].Title);
    %></td><%
    %></tr><%
    //increment the index
    Index++;
}
%></table><%
%>

```

Like so...

|   |                         |
|---|-------------------------|
| 1 | Raiders of the Lost Ark |
| 2 | Star Wars               |
| 3 | The Disaster Artist     |

## Styling the Table

The final task is to style the table so that it appears on the page where we would like it to appear.

To do this we will need to enclose it in a Div.

If we want to format a small section of text in a sentence we would make use of a `<span>`.

In this case we want to format a larger section of text we will make use of a `<div>`

To make this work we will need to mark-up the table HTML and also include the embedded C# code...

```

<article>
  <div class="MainTable">
    <%
      //create an instance of the DVD collection
      clsDVDCollection MyDVDs = new clsDVDCollection();
      //create the index variable initialised at 0
      Int32 Index = 0;
      //get the count of records
      Int32 RecordCount = MyDVDs.Count;
      //loop through each DVD
      %><table border="1"><%
        while (Index < RecordCount)
        {
          %><tr><%
            %><td><%
              //write the title to the browser
              Response.Write(MyDVDs.DVDList[Index].DVDNo);
            %></td><%
            %><td><%
              //write the title to the browser
              Response.Write(MyDVDs.DVDList[Index].Title);
            %></td><%
          %></tr><%
          //increment the index
          Index++;
        }
      %></table><%
    %>
  </div>
</article>

```

In this case we have created a div called MainTable.

To style the `<div>` in the style sheet we would need to create a style like this...

```

.MainTable {
    border:solid;
    position:fixed;
    top:25%;
    left:15%;
}

```

### Some things to note...

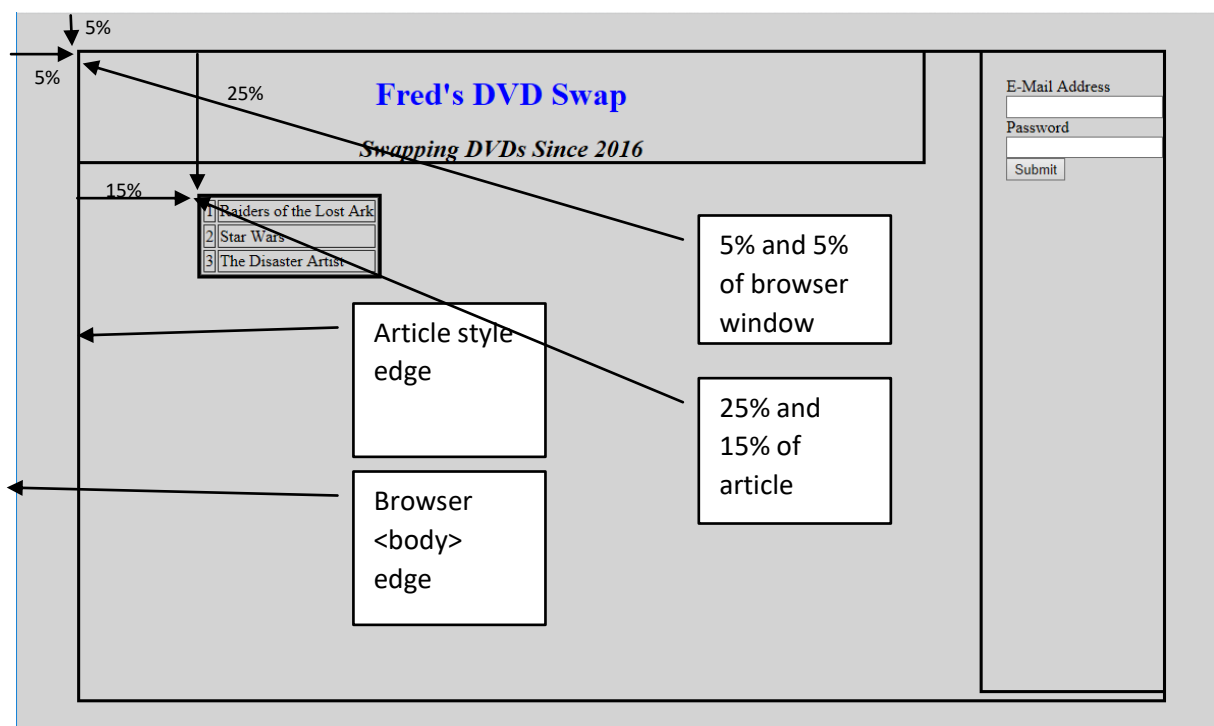
In this style we are creating a new style called .MainTable

Don't forget the dot at the front otherwise it won't work.

We are creating a fixed style with a top of 25% and a left value of 15%.

The question is 25% and 15% of what.

Take a look at the layout you will get from this and all of the other styles combined...



The new <div> is a child of the style article...

```

<article>
  <div class="MainTable">
    <%
      //create an instance of the DVD collection
      clsDVDCollection MyDVDs = new clsDVDCollection();
      //create the index variable initialised at 0
      Int32 Index = 0;
    %>
  </div>
</article>

```

This means that the top and left values are calculated with reference to the parent style.

The style “article” is a child of the body tag...

```
<body>
  <header>
    <h1>
      Fred's DVD Swap
    </h1>
    <h2>
      Swapping DVDs Since 2016
    </h2>
  </header>
  <article>
```

This means that its top and left are calculated with reference to the size of the main browser window.

### *Finally...*

Using the tags <tr> & <th> have a go at creating a header row for your table like so...

| DVD No | DVD Title               |
|--------|-------------------------|
| 1      | Raiders of the Lost Ark |
| 2      | Star Wars               |
| 3      | The Disaster Artist     |

You will need to do a little research to figure out what the tags do.