HTML Tables

A table is a structured set of data made up of rows and columns (tabular data). A table allows you to quickly and easily look up values that indicate some kind of connection between different types of data. The point of a table is that it is rigid. Information is easily interpreted by making visual associations between row and column headers. When done correctly, even blind people can interpret tabular data in an HTML table — a successful HTML table should enhance the experience of sighted and visually impaired users alike.

Be under no illusion; for tables to be effective on the web, you need to provide some styling information with CSS, as well as good solid structure with HTML.

Using tables for layout rather than CSS layout techniques is a bad idea. The main reasons are as follows:

1. **Layout tables reduce accessibility for visually impaired users**: Screenreaders, used by blind people, interpret the tags that exist in an HTML page and read out the contents to the user. Because tables are not the right tool for layout, and the markup is more complex than with CSS layout techniques, the screenreaders' output will be confusing to their users.
2. **Tables produce tag soup**: As mentioned above, table layouts generally involve more complex markup structures than proper layout techniques. This can result in the code being harder to write, maintain, and debug.
3. **Tables are not automatically responsive**: When you use proper layout containers (such as `<header>`, `<section>`, `<article>`, or `<div>`), their width defaults to 100% of their parent element. Tables on the other hand are sized according to their content by default, so extra measures are needed to get table layout styling to effectively work across a variety of devices.

Creating an HTML Table

- The content of every HTML table is enclosed by these two tags : **`<table>`** **`</table>`**
- The smallest container inside a table is a table **cell**, which is created by a **`<td>`** element ('td' stands for 'table data').

```
<td>Hi, I'm your first cell.</td>
```

If we want a row of four cells, we need to copy these tags three times. Update the contents of your table to look like so:

```
<td>Hi, I'm your first cell.</td>
<td>I'm your second cell.</td>
<td>I'm your third cell.</td>
<td>I'm your fourth cell.</td>
```

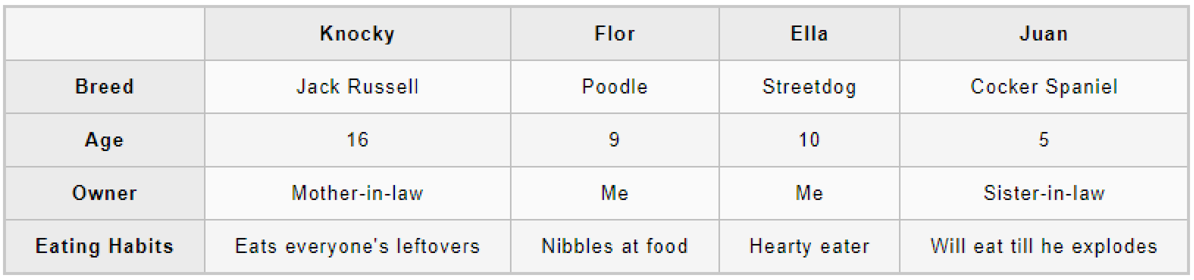
- As you will see, the cells are not placed underneath each other, rather they are automatically aligned with each other on the same row. Each `<td>` element creates a single cell and together they make up the first row. Every cell we add makes the row grow longer.
- To stop this row from growing and start placing subsequent cells on a second row, we need to use the `<tr>` element ('tr' stands for 'table row').

```
<tr>
  <td>Hi, I'm your first cell.</td>
  <td>I'm your second cell.</td>
  <td>I'm your third cell.</td>
  <td>I'm your fourth cell.</td>
</tr>
```

Adding Table Headers

Table headers are special cells that go at the start of a row or column and define the type of data that row or column contains. To recognize the table headers as headers, both visually and semantically, you can use the `<th>` element ('th' stands for 'table header'). This works in exactly the same way as a `<td>`, except that it denotes a header, not a normal cell.

```
<table>
      <tr>
        <td>&nbsp;</td>
        <th>Knocky</th>
        <th>Flor</th>
        <th>Ella</th>
        <th>Juan</th>
      </tr>
      <tr>
        <th>Breed</th>
        <td>Jack Russell</td>
        <td>Poodle</td>
        <td>Streetdog</td>
        <td>Cocker Spaniel</td>
      </tr>
      <tr>
        <th>Age</th>
        <td>16</td>
        <td>9</td>
        <td>10</td>
        <td>5</td>
      </tr>
      <tr>
        <th>Owner</th>
        <td>Mother-in-law</td>
        <td>Me</td>
        <td>Me</td>
        <td>Sister-in-law</td>
      </tr>
      <tr>
        <th>Eating Habits</th>
        <td>Eats everyone's leftovers</td>
        <td>Nibbles at food</td>
        <td>Hearty eater</td>
        <td>Will eat till he explodes</td>
      </tr>
    </table>
```



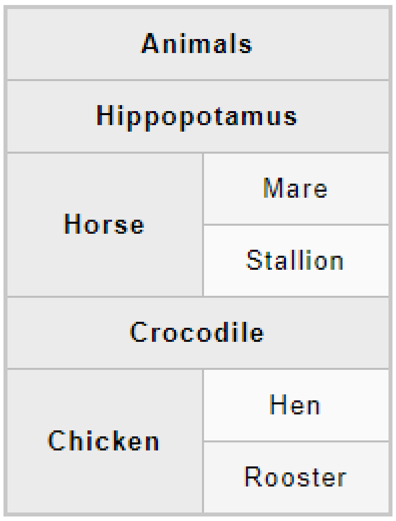
	Knocky	**Flor**	**Ella**	**Juan**
Breed	Jack Russell	Poodle	Streetdog	Cocker Spaniel
Age	16	9	10	5
Owner	Mother-in-law	Me	Me	Sister-in-law
Eating Habits	Eats everyone's leftovers	Nibbles at food	Hearty eater	Will eat till he explodes

Tables headers also have an added benefit — along with the scope attribute (which we'll learn about in the next article), they allow you to make tables more accessible by associating each header with all the data in the same row or column. Screenreaders are then able to read out a whole row or column of data at once, which is pretty useful.

Multi-span rows and columns

Sometimes we want cells to span multiple rows or columns. Table headers and cells have the `colspan` and `rowspan` attributes, which allow us to do just those things. Both accept a unitless number value, which equals the number of rows or columns you want spanned.

```
<table>
      <tr>
        <th colspan="2">Animals</th>
      </tr>
      <tr>
        <th colspan="2">Hippopotamus</th>
      </tr>
      <tr>
        <th rowspan="2">Horse</th>
        <td>Mare</td>
      </tr>
      <tr>
        <td>Stallion</td>
      </tr>
      <tr>
        <th colspan="2">Crocodile</th>
      </tr>
      <tr>
        <th rowspan="2">Chicken</th>
        <td>Hen</td>
      </tr>
      <tr>
        <td>Rooster</td>
      </tr>
    </table>
```



Animals	
Hippopotamus	
Horse	Mare
	Stallion
Crocodile	
Chicken	Hen
	Rooster

Styling Columns

HTML has a method of defining styling information for an entire column of data all in one place — the `<col>` and `<colgroup>` elements. These exist because it can be a bit annoying and inefficient having to specify styling on columns — you generally have to specify your styling information on every `<td>` or `<th>` in the column, or use a complex selector such as `:nth-child`. Styling columns like this is limited to a few properties: `border`, `background`, `width`, and `visibility`. To set other properties you'll have to either style every `<td>` or `<th>` in the column, or use a complex selector such as `:nth-child`.

Consider:

```
<table>
  <tr>
    <th>Data 1</th>
    <th style="background-color: yellow">Data 2</th>
  </tr>
  <tr>
    <td>Calcutta</td>
    <td style="background-color: yellow">Orange</td>
  </tr>
  <tr>
    <td>Robots</td>
    <td style="background-color: yellow">Jazz</td>
  </tr>
</table>
```

This isn't ideal, as we have to repeat the styling information across all three cells in the column (we'd probably have a class set on all three in a real project and specify the styling in a separate stylesheet).

Instead of doing this, we can specify the information once, on a `<col>` element. `<col>` elements are specified inside a `<colgroup>` container just below the opening `<table>` tag. We could create the same effect as we see above by specifying our table as follows:

```
<table>
  <colgroup>
    <col>
    <col style="background-color: yellow">
  </colgroup>
  <tr>
    <th>Data 1</th>
    <th>Data 2</th>
  </tr>
  <tr>
    <td>Calcutta</td>
    <td>Orange</td>
  </tr>
  <tr>
    <td>Robots</td>
    <td>Jazz</td>
  </tr>
</table>
```

Effectively we are defining two "style columns", one specifying styling information for each column. We are not styling the first column, but we still have to include a blank `<col>` element — if we didn't, the styling would just be applied to the first column.

If we wanted to apply the styling information to both columns, we could just include one `<col>` element with a span attribute on it, like this:

```
<colgroup>
  <col style="background-color: yellow" span="2">
</colgroup>
```

Just like `colspan` and `rowspan`, `span` takes a unitless number value that specifies the number of columns you want the styling to apply to.

Adding a caption to your table with <caption>

You can give your table a caption by putting it inside a `<caption>` element and nesting that inside the `<table>` element. You should put it just below the opening `<table>` tag. The caption is meant to contain a description of the table contents. This is useful for all readers wishing to get a quick idea of whether the table is useful to them as they scan the page, but particularly for blind users. Rather than have a screenreader read out the contents of many cells just to find out what the table is about, the user can rely on a caption and then decide whether or not to read the table in greater detail.

```
<table>
  <caption>Dinosaurs in the Jurassic period</caption>

  ...
</table>
```

Adding structure with <thead>, <tfoot>, and <tbody>

As your tables get a bit more complex in structure, it is useful to give them more structural definition. One clear way to do this is by using **`<thead>`**, **`<tfoot>`**, and **`<tbody>`**, which allow you to mark up a header, footer, and body section for the table.

These elements don't make the table any more accessible to screenreader users, and don't result in any visual enhancement on their own. They are however very useful for styling and layout — acting as useful hooks for adding CSS to your table.

To use them:

- The `<thead>` element must wrap the part of the table that is the header — this is usually the first row containing the column headings, but this is not necessarily always the case. If you are using `<col>`/`<colgroup>` element, the table header should come just below those.
- The `<tfoot>` element needs to wrap the part of the table that is the footer — this might be a final row with items in the previous rows summed, for example. You can include the table footer right at the bottom of the table as you'd expect, or just below the table header (the browser will still render it at the bottom of the table).
- The `<tbody>` element needs to wrap the other parts of the table content that aren't in the table header or footer. It will appear below the table header or sometimes footer, depending on how you decided to structure it.

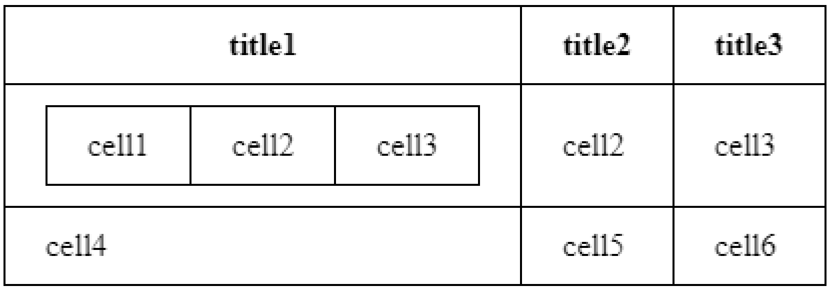
Note: `<tbody>` is always included in every table, implicitly if you don't specify it in your code. To check this, open up one of your previous examples that doesn't include `<tbody>` and look at the HTML code in your browser developer tools — you will see that the browser has added this tag for you. You might wonder why you ought to bother including it at all — you should, because it gives you more control over your table structure and styling.

Nesting Tables

It is possible to nest a table inside another one, as long as you include the complete structure, including the `<table>` element. This is generally not really advised, as it makes the markup more confusing and less accessible to screenreader users, and in many cases you might as well just insert extra cells/rows/columns into the existing table. It is however sometimes necessary, for example if you want to import content easily from other sources.

The following markup shows a simple nested table, with its subsequent output.

```
<table id="table1">
  <tr>
    <th>title1</th>
    <th>title2</th>
    <th>title3</th>
  </tr>
  <tr>
    <td id="nested">
      <table id="table2">
        <tr>
          <td>cell1</td>
          <td>cell2</td>
          <td>cell3</td>
        </tr>
      </table>
    </td>
    <td>cell2</td>
    <td>cell3</td>
  </tr>
  <tr>
    <td>cell4</td>
    <td>cell5</td>
    <td>cell6</td>
  </tr>
</table>
```



title1	title2	title3
cell1 cell2 cell3	cell2	cell3
cell4	cell5	cell6

Tables for visually impaired users

A table can be a handy tool, for giving us quick access to data and allowing us to look up different values using visual associations between the data and its column and/or row headers. But what if you cannot make those visual associations? Interpreting a table can be quite a challenge for a blind person. Nevertheless, with the proper markup we can replace visual associations by programmatic ones.

Here are the techniques by which we can make tables more accessible, especially for those who are visually impaired.

Using column and row headers

(This is already covered above)

The scope attribute

The `scope` attribute, which can be added to the `<th>` element to tell screenreaders exactly what cells the header is a header for — is it a header for the row it is in, or the column, for example?

```
<thead>
  <tr>
    <th scope="col">Purchase</th>
    <th scope="col">Location</th>
    <th scope="col">Date</th>
    <th scope="col">Evaluation</th>
    <th scope="col">Cost (€)</th>
  </tr>
</thead>
```

And each row could have a header defined like this (if we added row headers as well as column headers):

```
<tr>
  <th scope="row">Haircut</th>
  <td>Hairdresser</td>
  <td>12/09</td>
  <td>Great idea</td>
  <td>30</td>
</tr>
```

Screenreaders will recognize markup structured like this, and allow their users to read out the entire column or row at once, for example.

`scope` has two more possible values — `colgroup` and `rowgroup`. these are used for headings that sit over the top of multiple columns or rows.

The id and headers attributes

An alternative to using the `scope` attribute is to use `id` and `headers` attributes to create associations between headers and cells. The way they are used is as follows:

1. You add a unique `id` to each `<th>` element.
2. You add a `headers` attribute to each `<td>` element. Each `headers` attribute has to contain a list of the `id`s of all the `<th>` elements that act as a header for that cell, separated by spaces.

This gives your HTML table an explicit definition of the position of each cell in the table, defined by the header(s) for each column and row it is part of, kind of like a spreadsheet. For it to work well, the table really needs both column and row headers.

```
<thead>
  <tr>
    <th id="purchase">Purchase</th>
    <th id="location">Location</th>
    <th id="date">Date</th>
    <th id="evaluation">Evaluation</th>
    <th id="cost">Cost (€)</th>
  </tr>
</thead>
<tbody>
<tr>
  <th id="haircut">Haircut</th>
  <td headers="location haircut">Hairdresser</td>
  <td headers="date haircut">12/09</td>
  <td headers="evaluation haircut">Great idea</td>
  <td headers="cost haircut">30</td>
</tr>

  ...

</tbody>
```

Note: This method creates very precise associations between headers and data cells but it uses a lot more markup and does not leave any room for errors. The `scope` approach is usually enough for most tables.