Web Fundamentals

HTML and CSS

Learning objectives for this segment

Basics of an HTML page	Page head	Page body
Displaying text	Common tags	Semantics
The "right" tag	Adding attributes	Linking to pages
Styling with CSS	Class attribute	CSS selectors
Tag selectors	Class selectors	ID selectors
Location selectors	Complex selectors	CSS specificity
Multiple CSS classes	Adding images	The box model
Size units	Flexbox for Layout	Responsive Web

What will we be building through this segment

Ok, so how do we get started?

Every project should begin with at least a sketch

Mobile First

— Notice we started with the mobile view.
— Web is moving, some would say has already moved, to being mobile dominated
— Mobile first focuses our attention on the most important elements of the page
— Allows us to progressively *add* more information to the page
— Easier to add than it is to remove

Content First

Get all the content onto the page without focusing on the structure of the page

Blank page syndrome

— Great, we have a sketch and we are going to design mobile first

— We start by creating an HTML page / document

What is HTML?

— HTML (Hypertext Markup Language) is a markup language
— Used to tell your browser how to structure the web pages
— Consists of a series of elements
— Elements enclose, wrap, or mark up different parts of the content

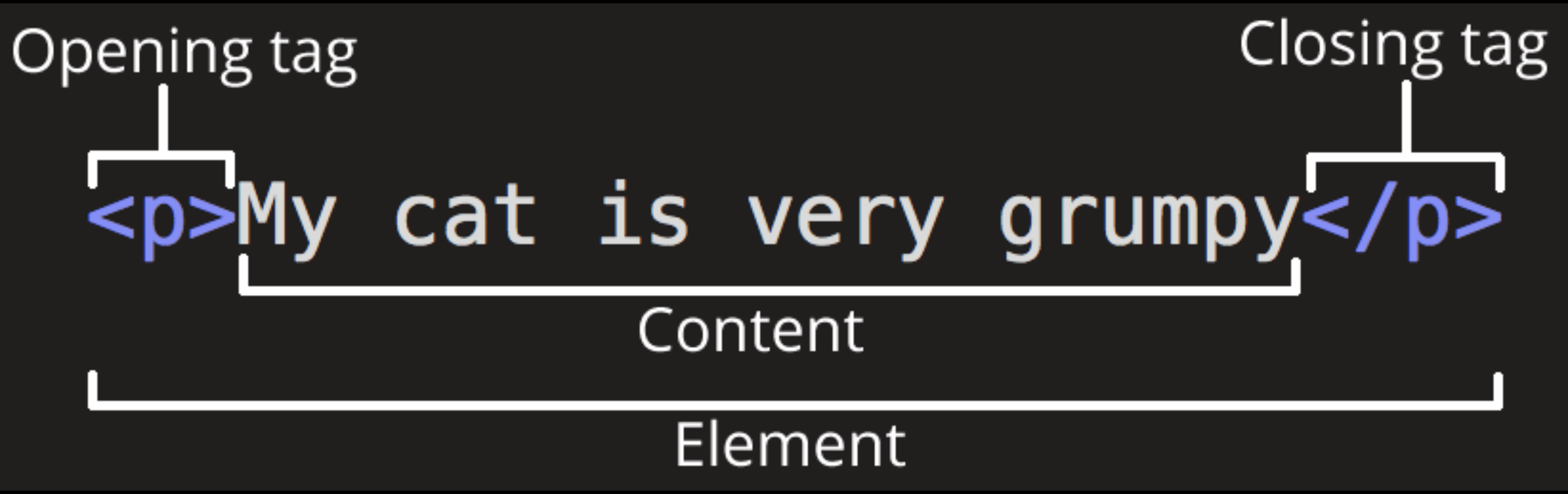
Most basic HTML is plain text

Jason

Adding meaning to text

```
<p>Jason</p>
```

What is an HTML Element?


Opening tag
Closing tag
My cat is very grumpy
Content
Element

The main parts of an element are:

1. The opening tag: the name of the element (in this case, p), wrapped in opening and closing angle brackets.
2. The closing tag: same as the opening tag, except that it includes a forward slash before the element name.
3. The content: the content of the element, which in this case is just text.

Nesting Elements

To say that Jason is our CEO, we could wrap the word "CEO" in a `<strong>` element. This indicates, both *semantically* and *visually* that the word is to be strongly emphasized

```
<p>Jason is our <strong>CEO</strong></p>
```

Careful!

You do however need to make sure that your elements are properly nested: in the example above, we opened the p element first, then the strong element, therefore we have to close the strong element first, then the p.

The following is incorrect:

```
<p>Jason is our <strong>CEO</p></strong>
```

Browser are too good

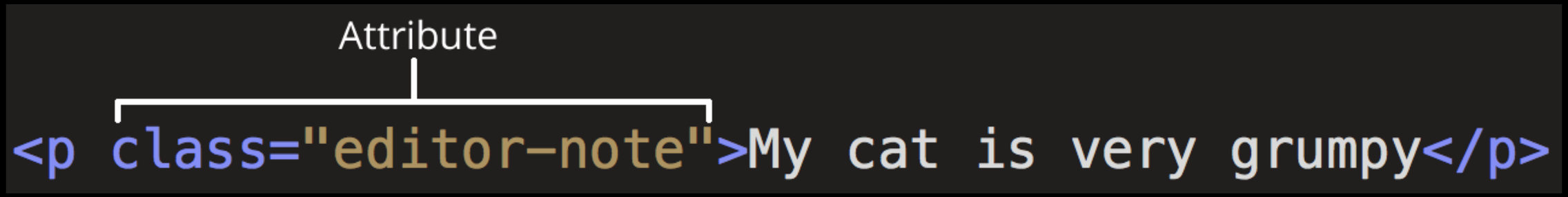
Browsers are very forgiving of invalid HTML (even if your instructors or teammates are not)

For instance the invalid code above looks like this in the browser. There is no difference in the browser of the broken code as the correct code. Only until we add more content to the page may we see the effects of our error.

Making sure you have *correct* code is very important.

HTML Attributes

Elements can also have attributes, which look like this:



Attributes

— Attributes contain extra information about the element which you don't want to appear in the actual content.

— In this case, the class attribute allows you to give the element an identifying name that can be later used to target the element with style information and other things.

Attributes Should Have

1. A space between it and the element name (or the previous attribute, if the element already has one or more attributes.)
2. The attribute name, followed by an equals sign.
3. An attribute value, with opening and closing quote marks wrapped around it.

Anatomy of an HTML document

HTML elements aren't very useful on their own. Now we'll look at how individual elements are combined to form an entire HTML page:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>About SDG</title>
  </head>
  <body>
    <p>Jason is our <strong>CEO</strong></p>
  </body>
</html>
```

Lets break this down

<!DOCTYPE html>

The doctype. In the mists of time, when HTML was young (about 1991/2), doctypes were meant to act as links to a set of rules that the HTML page had to follow to be considered good HTML, which could mean automatic error checking and other useful things.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>About SDG</title>
  </head>
  <body>
    <p>Jason is our <strong>CEO</strong></p>
  </body>
</html>
```

<!DOCTYPE html>

However, these days no one really cares about them, and they are really just a historical artifact that needs to be included for everything to work right. `<!DOCTYPE html>` is the shortest string of characters that counts as a valid doctype; that's all you really need to know.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>About SDG</title>
  </head>
  <body>
    <p>Jason is our <strong>CEO</strong></p>
  </body>
</html>
```

`<html></html>`

The `<html>` element. This element wraps all the content on the entire page, and is sometimes known as the root element.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>About SDG</title>
  </head>
  <body>
    <p>Jason is our <strong>CEO</strong></p>
  </body>
</html>
```

<head></head>

The <head> element. This element acts as a container for all the stuff you want to include on the HTML page that isn't the content you are showing to your page's viewers. This includes things like keywords and a page description that you want to appear in search results, CSS to style our content, character set declarations, and more.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>About SDG</title>
  </head>
  <body>
    <p>Jason is our <strong>CEO</strong></p>
  </body>
</html>
```

```
<meta charset="utf-8">
```

This element sets the character set your document should use to UTF-8, which includes most characters from the vast majority of human written languages. Essentially it can now handle any textual content you might put on it. There is no reason not to set this, and it can help avoid some problems later.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>About SDG</title>
  </head>
  <body>
    <p>Jason is our <strong>CEO</strong></p>
  </body>
</html>
```

<title></title>

The <title> element. This sets the title of your page, which is the title that appears in the browser tab the page is loaded in, and is used to describe the page when you bookmark/ favorite it.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>About SDG</title>
  </head>
  <body>
    <p>Jason is our <strong>CEO</strong></p>
  </body>
</html>
```

<body></body>

The <body> element. This contains all the content that you want to show to web users when they visit your page, whether that's text, images, videos, games, playable audio tracks, or whatever else.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>About SDG</title>
  </head>
  <body>
    <p>Jason is our <strong>CEO</strong></p>
  </body>
</html>
```

Whitespace

In the above examples you may have noticed that a lot of whitespace is included in the code listings — this is not necessary at all; the two following code snippets are equivalent:

```
<p>Dogs are silly.</p>
```

```
<p>Dogs          are
         silly.</p>
```

Whitespace

No matter how much whitespace you use (which can include space characters, but also line breaks), HTML reduces each one down to a single space when rendering.

Whitespace

So why use so much whitespace? The answer is readability — it is so much easier to understand what is going on in your code if you have it nicely formatted. It is up to you what style of formatting you use (how many spaces for each level of indentation, for example), but you should consider formatting it.

Whitespace

At Suncoast Developers Guild we will use a tool named `prettier` *to ensure our code is nicely formatted*

Displaying Text

One of HTML's main jobs is to give text structure and meaning (also known as semantics) so that a browser can display it correctly. Let's take a look at how HTML can be used to structure a page of text by adding headings and paragraphs, emphasizing words, creating lists, and more.

The basics: Headings and Paragraphs

Most structured text is comprised of headings and paragraphs, irrespective of whether you are reading a story, a newspaper, a college textbook, a magazine, etc.

Structured content makes the reading experience easier and more enjoyable.

Paragraphs

In HTML, each paragraph has to be wrapped in a <p> element, like so:

```
<p>I am a paragraph, oh yes I am.</p>
```

Headings

Each heading has to be wrapped in a heading element:

```
<h1>I am the title of the story.</h1>
```

Headings

There are six heading elements — `<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>`, and `<h6>`.

Each represents a different level of content; `<h1>` represents the main heading, `<h2>` subheadings, `<h3>` sub-subheadings, and so on.

For example, in a story, `<h1>` would represent the title, `<h2>`s the title of each chapter and `<h3>`s the sub-sections of each chapter, and so on.

Example

```
<h1>The Crushing Bore</h1>

<p>By Chris Mills</p>

<h2>Chapter 1: The dark night</h2>

<p>
  It was a dark night. Somewhere, an owl hooted. The rain lashed down on the ...
</p>

<h2>Chapter 2: The eternal silence</h2>

<p>
  Our protagonist could not so much as a whisper out of the shadowy figure ...
</p>

<h3>The specter speaks</h3>

<p>
  Several more hours had passed, when all of a sudden the specter sat bolt
  upright and exclaimed, "Please have mercy on my soul!"
</p>
```

Semantics

It's really up to you what exactly the elements involved represent, as long as the hierarchy makes sense. You just need to bear in mind a few best practices as you create such structures.

Some General Rules

1. Use the headings in the correct order in the hierarchy. Don't use `<h3>`s to represent subheadings, followed by `<h2>`s to represent sub-subheadings — that doesn't make sense and will lead to weird results.
2. Of the six heading levels available, you should aim to use no more than three per page, unless you feel it is necessary.

Why do we need semantics?

Semantics are relied on everywhere around us — we rely on previous experience to tell us what the function of everyday objects is; when we see something, we know what its function will be. So, for example, we expect a red traffic light to mean "stop", and a green traffic light to mean "go".

Why do we need semantics?

In a similar vein, we need to make sure we are using the correct elements, giving our content the correct meaning, function, or appearance. In this context the `<h1>` element is also a semantic element, which gives the text it wraps around the role (or meaning) of "a top level heading on your page."

```
<h1>This is a top level heading</h1>
```

How are semantics used?

By default, the browser will give it a large font size to make it look like a heading (although you could style it to look like anything you wanted using CSS). More importantly, its semantic value will be used in multiple ways, for example by search engines and screen readers.

More on semantics later

In a later section we will discuss a wide range of semantic tags to help give our pages more meaning and context.

Lists

Lists are everywhere in life — from your shopping list to the list of directions you subconsciously follow to get to your house every day, to the lists of instructions you are following in these tutorials!

Lists are everywhere on the Web too, and we have three different types to use.

Unordered Lists

Unordered lists are used to mark up lists of items for which the order of the items doesn't matter.

Let's take a shopping list as an example.

```
milk
eggs
bread
hummus
```

Unordered List Container

Every unordered list starts off with a `<ul>` element — this wraps around all the list items:

```
<ul>
milk
eggs
bread
hummus
</ul>
```

Unordered List Items

The last step is to wrap each list item in an `<li>` (list item) element:

```
<ul>
  <li>milk</li>
  <li>eggs</li>
  <li>bread</li>
  <li>hummus</li>
</ul>
```

`<ul>` and `<li>` relationship

You'll notice that when we added the `<li>` items we now *indent* those elements.

The `<li>` are *nested* inside the `<ul>` and we say they are *children* of the `<ul>` element.

To help us visually we indent the children element several spaces for every level of nesting.

The browser doesn't care about the nesting, but it helps us as writers (and as readers) of our code.

Ordered

Ordered lists are lists in which the order of the items does matter — let's take a set of directions as an example:

```
Drive to the end of the road
Turn right
Go straight across the first two roundabouts
Turn left at the third roundabout
The school is on your right, 300 meters up the road
```

Ordered list

The markup structure is the same as for unordered lists, except that you have to wrap the list items in an `<ol>` element, rather than `<ul>`

```
<ol>
  <li>Drive to the end of the road</li>
  <li>Turn right</li>
  <li>Go straight across the first two roundabouts</li>
  <li>Turn left at the third roundabout</li>
  <li>The school is on your right, 300 meters up the road</li>
</ol>
```

Nesting lists

It is perfectly ok to nest one list inside another one. You might want to have some sub-bullets sitting below a top level bullet. Let's look at this recipe example:

```
<ol>
  <li>Remove the skin from the garlic, and chop coarsely.</li>
  <li>Remove all the seeds and stalk from the pepper, and chop coarsely.</li>
  <li>Add all the ingredients into a food processor.</li>
  <li>Process all the ingredients into a paste.</li>
  <li>If you want a coarse "chunky" hummus, process it for a short time.</li>
  <li>If you want a smooth hummus, process it for a longer time.</li>
</ol>
```

Nested Lists

The last two bullets are very closely related to the one before them, it might make sense to nest them inside their own unordered list, and put that list inside the current fourth bullet.

This would look like so:

```
<ol>
  <li>Remove the skin from the garlic, and chop coarsely.</li>
  <li>Remove all the seeds and stalk from the pepper, and chop coarsely.</li>
  <li>Add all the ingredients into a food processor.</li>
  <li>
    Process all the ingredients into a paste.
    <ul>
      <li>
        If you want a coarse "chunky" hummus, process it for a short time.
      </li>
      <li>If you want a smooth hummus, process it for a longer time.</li>
    </ul>
  </li>
</ol>
```

Linking to other pages

We can create plain text, paragraphs, and lists of items now. However we can only create these on one page.

Lets add links between pages.

Links

Links (also known as Hyperlinks) are what makes the Web a Web — they allow us to link our documents to any other document (or other resource) we want to

Just about any web content can be converted to a link, so that when clicked it will make the web browser go to another web address (URL).

Anatomy of a link

A basic link is created by wrapping the content you want to turn into a link inside an <a> element, and giving it an href attribute that will contain the web address you want the link to point to.

```
<p>
  I'm creating a link to the
  <a href="https://suncoast.io/handbook">Suncoast Developers Guild Handbook</a>
  page.
</p>
```

NOTE that `href` stands for `Hypertext REFerence`

Adding supporting information with the title attribute

Another attribute you may want to add to your links is title; this is intended to contain supplementary useful information about the link, such as what kind of information the page contains, or things to be aware of. For example:

```
<p>
  I'm creating a link to the
  <a
    href="https://suncoast.io/handbook"
    title="Our curriculum and student handbook">Suncoast Developers Guild Handbook</a>
  page.
</p>
```

Block level links

As mentioned before, you can turn just about any content into a link, even block level elements. If you had an image you wanted to turn into a link, you could just put the image between tags.

```
<a href="https://suncoast.io/handbook">
  <img
    src="https://suncoast.io/brand/bouy.png"
    alt="Suncoast Developers Guild Bouy logo"
  />
</a>
```

E-mail links

It is possible to create links or buttons that, when clicked, open a new outgoing email message rather than linking to a resource or page. This is done using the `<a>` element and the `mailto:` URL scheme.

```
<p>
  You may <a href="mailto:person@suncoast.io">contact us</a> with questions.
</p>
```

Tables

These help structure *tabular* data. That is, data that is best expressed in a series of rows comprised of columns. With a little CSS we can make beautiful data representations.

NOTE: It *used* to be that developers used `<table>` for *layout* which we do not do any more since we have much better systems for layout. (more on layout later)

Table Example

Person	Age
Chris	38
Dennis	45
Sarah	29
Karen	47

Table Example

Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
Public Swim 06:30 - 10:30	Public Swim 06:30 - 09:00	Public Swim 06:30 - 09:00	Public Swim 06:30 - 11:15	Public Swim 06:30 - 09:00	Lane Swim 08:00 - 09:00	Lane Swim 08:00 - 09:00
Aquacise 10:30 - 11:15	Aqua Jog 09:15 - 10:00	Education Swimming Lessons 09:00 - 12:00	Aquacise 11:15 - 12:00	Education Swimming Lessons 09:00 - 12:00	Oldham Active Kids Swimming Lessons 09:00 - 13:00	Public Swim 09:00 - 11:00
Lane Swim 11:30 - 13:00	Parent & Baby Class 09:30 - 10:15	Lane Swim 12:00 - 13:00	Lane Swim 12:00 - 13:00	Lane Swim 12:00 - 13:00	Parent and Baby 12:00 - 12:45	Aquacise 11:00 - 11:45
Education Swimming Lessons	Public Swim 10:00 - 11:45	Public Swim 13:00 - 16:00	Education Swimming Lessons	Oldham Active Kids Swimming	Public Swim 13:00 - 17:00	Public Swim 11:45 - 13:00

How does a table work

The point of a table is that it is rigid. Information is easily interpreted by making visual associations between row and column headers.

Table Example Code

```
<table>
  <caption>
    Data about the planets of our solar system (Planetary facts taken from
    <a
      rel="noopener"
      href="http://nssdc.gsfc.nasa.gov/planetary/factsheet/"
      class="external external-icon"
      >Nasa's Planetary Fact Sheet - Metric</a
    >.
  </caption>
  <thead>
    <tr>
      <td colspan="2"> </td>
      <th scope="col">Name</th>
      <th scope="col">Mass (10<sup>24</sup>kg)</th>
      <th scope="col">Diameter (km)</th>
      <th scope="col">Notes</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <th colspan="2" rowspan="4" scope="rowgroup">Terrestial planets</th>
      <th scope="row">Mercury</th>
      <td>0.330</td>
      <td>4,879</td>
      <td>5427</td>
      <td>Closest to the Sun</td>
    </tr>
    <tr>
      <th scope="row">Venus</th>
      <td>4.87</td>
      <td>12,104</td>
      <td>5243</td>
      <td> </td>
    </tr>
  </tbody>
</table>
```

Table Example Code

Take a look at this example to see a detailed example of a table

Parts of the table

A table is comprised of three major portions: `thead`, `tfoot`, and `tbody`

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

— The `<thead>` element needs to wrap the part of the table that is the header
— The `<tfoot>` element needs to wrap the part of the table that is the footer
— The `<tbody>` element needs to wrap the other parts of the table content that aren't in the table header or footer.

Making rows

The `<thead>` and `<tbody>` should be comprised of many `<tr>` table rows.

Within the `<tr>` elements we will find a sequence of either `<th>` for *headers* or `<td>` for *data*

Beyond the basics

Now that we have covered many of the basic HTML tags, lets talk about *semantics*

Semantics relate to meaning in a language. In HTML we want to choose tags that convey to the browser, and to any future developer, the purpose of the chosen tag.

Example

For example, the `<h1>` element is a semantic element, which gives the text it wraps around the role of "a top level heading on your page."

By default, most browser's user agent stylesheet will style an `<h1>` with a large font size to make it look like a heading.

However ...

We could take any element and use styling to make it *look* like a header.

So why bother with using an `<h1>` versus an `<h2>` or a `<p>`?

Representation of structure

HTML should be coded to represent the data that will be populated and not based on its default presentation styling. Presentation (how it should look), is the sole responsibility of CSS.

Benefits

— Search engines will consider its contents as important keywords to influence the page's search rankings (see SEO)

— Screen readers can use it as a signpost to help visually impaired users navigate a page

— Finding blocks of meaningful code is significantly easier than searching though endless divs with or without semantic or namespaced classes

— Suggests to the developer the type of data that will be populated

— Semantic naming mirrors proper custom element/component naming

How to decide?

"What element(s) best describe/represent the data that I'm going to populate?"

For example:

— Is it a list of data? -- ordered, unordered?
— Is it an article with sections and an aside of related information?; does it list out definitions?
— Is it a figure or image that needs a caption?
— Should it have a header and a footer in addition to the global site-wide header and footer?

Gettings started with our `About SDG` site

In this first pass we layout just the *content* of the page.

Only place the names and the images onto the page.

Lets see what this looks like:

0 1 0 1 0 1 0

SUNCOAST DEVELOPERS GUILD


Jason

Mark

Gavin

Katherine


Live Code Time