Mandy Michael

Follow 2.1K Followers About

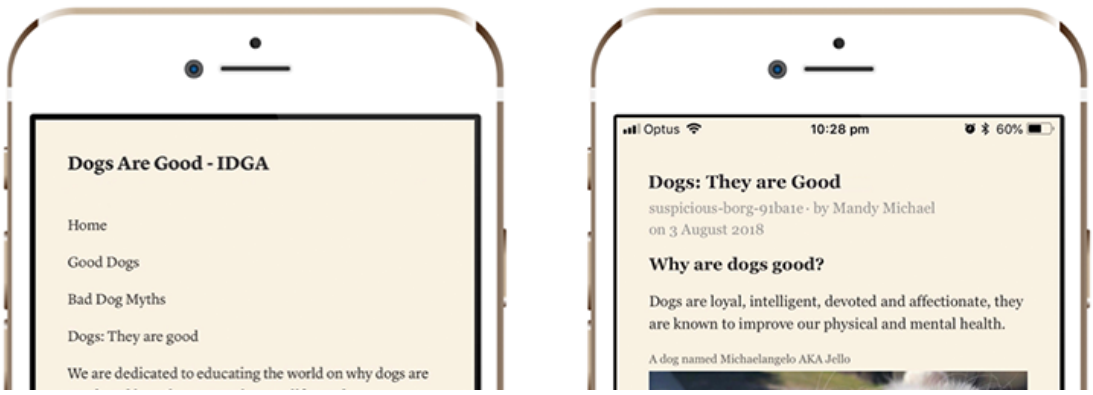
Building websites for Safari Reader Mode and other reading apps.

 Mandy Michael Aug 9, 2018 · 10 min read

Alternate reading modes offer users improved readability and options for personalisation — people can choose to use these tools for any number of reasons including better accessibility with improved contrast, or enjoying the aesthetics of a low noise environment to read in.

I went on a bit of a journey exploring how our websites are experienced across these technologies and how we can make sure that our content is interpreted the way we want, regardless of how people are consuming it.

For demo purposes let's look at what a simple HTML page looks like in Instapaper, Pocket and Safari's Reader mode and compare `<div>` only content and semantic HTML.



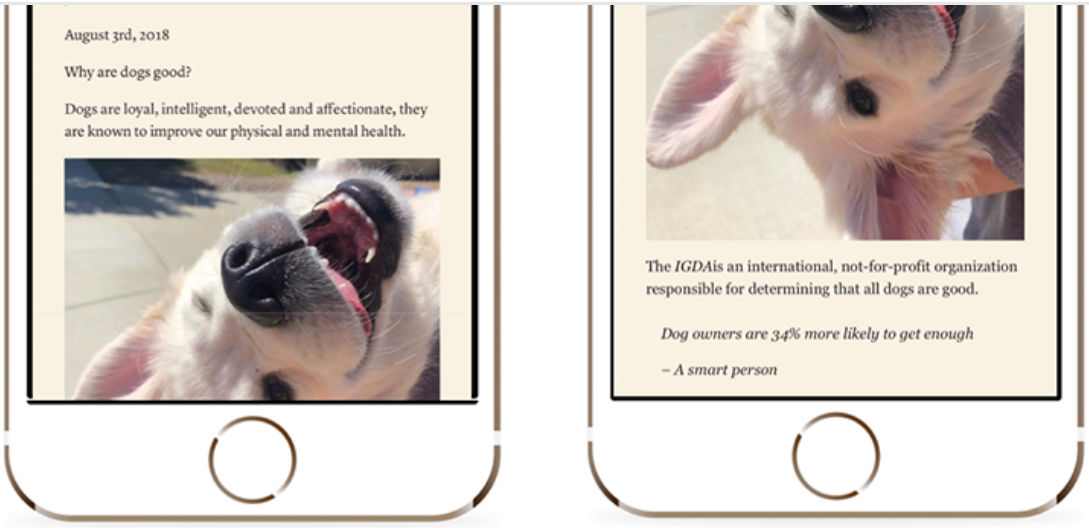


Figure 1: Instapaper example. Left Image: only, Right Image Semantic HTML

How it looks in Instapaper

First up we have Instapaper. Figure 1 demonstrates my webpage using divs and semantic html. The left image is using just the `<div>` element and the right image is using semantic HTML. There are a few things going on here.

1. **The page titles are different.** As there is no `<h1>` in the div only page the title is taken from the `<title>` element. Whereas when I use Semantic HTML is uses my documents `<h1>` to represent my page.

2. **The `<div>` version has what looks like links?** When you compare the two versions you can see there are what look like links at the top of the page. These were marked up as divs so Instapaper had no idea they were navigation items or that they were links and just put them in the page as text. Not ideal at all and completely useless.

3. **No content styling on `<div>` version**. When you compare the two examples you can see that “Why dogs are good” is styled nicely in the semantic version but not in the `<div>` version. As Instapaper didn’t know that “Why are dogs good” is actually a heading of the page it just treated it like any other text within the document. This is also the case for other elements further down the page such as quotes, lists, links etc.

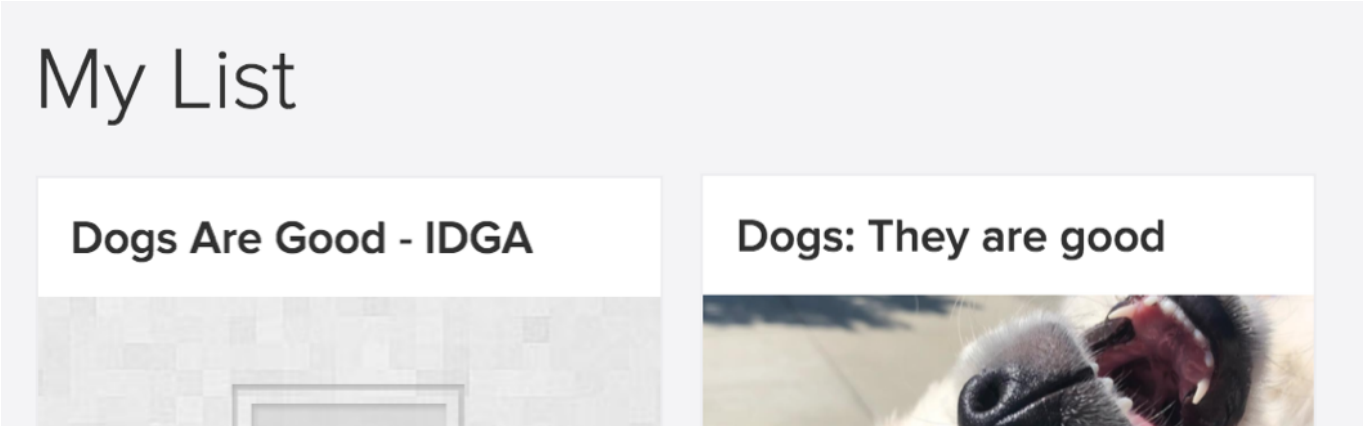
Figure 2: Top section of Instapaper. Left Image is made with <divs> and is missing byline and date information, Right Image with Semantic HTML and displays byline and date information.

In Figure 2 you can see the `<div>` version is missing information around the page url, who wrote the document and the date the document was published. Knowing who wrote the document and when is really valuable for both users and publishers when the websites original branding, or context is no longer available. Especially if, like me, you have hundreds of webpages saved to “read later”.

I will add that it took a bit of investigation for me to get this information to display correctly at the top of the page, but don’t worry i’ll explain later on in this post how to unlock this achievement.

How it looks in Pocket

When you save an article into Pocket on the Desktop web view it looks a bit like this. This displays my `<div>` version and my semantic HTML version. Guess which one is which! As an interesting untested but observed aside the `<div>` only version actually took longer to process and save than the semantic one.



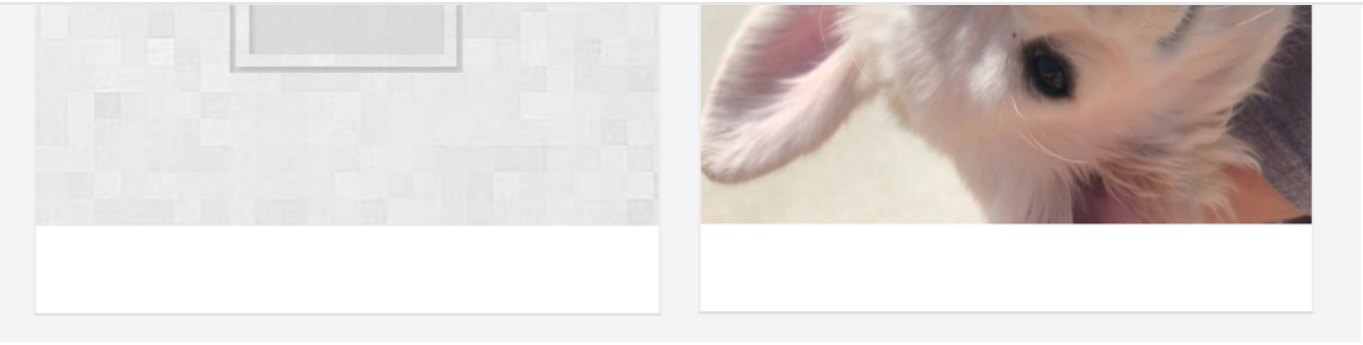


Figure 3: Pages displayed in Pocket. Left represents only page, Right represents semantic html

So there is a couple of things to note here:

1. **The titles are different.** For the same reasons as Instapaper above the page titles are different.
2. **No image on the `<div>` version.** Despite using the `<img>` element to display the image Pocket was unable to find and produce the image in the preview. Sometimes it would render random text from somewhere on the page instead of the image.
3. **`<div>` version cannot be viewed in the reading app.** Pocket was unable to reproduce the page in their reading mode for Desktop Web View and the Mobile App, I was required to link off to the original page.
   The Desktop App was able to display in reading mode, however, the following message was displayed.

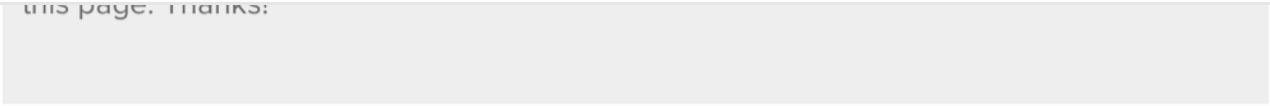


Figure 4: Warning message from Pocket suggesting I switch to webpage view as my content is not article content.

As there is no structural or semantic HTML in my document Pocket was unable to determine that this was an article and therefore provided a warning to users. This isn't really idea it makes your content look untrustworthy. Big warning boxes are never great.

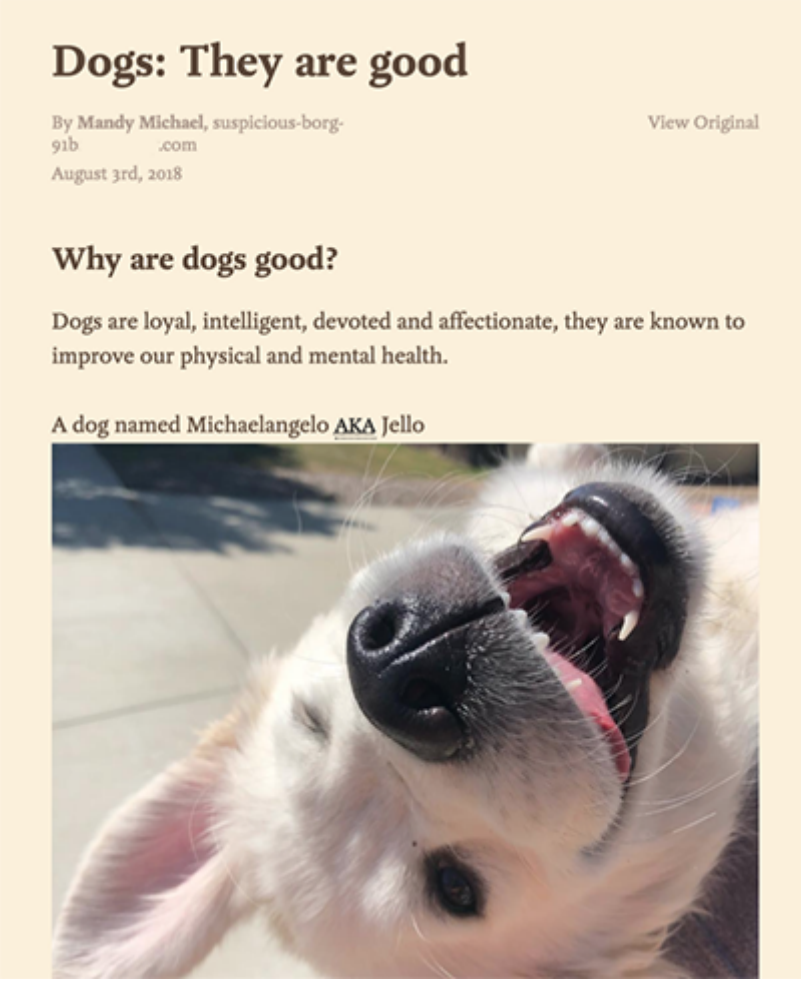
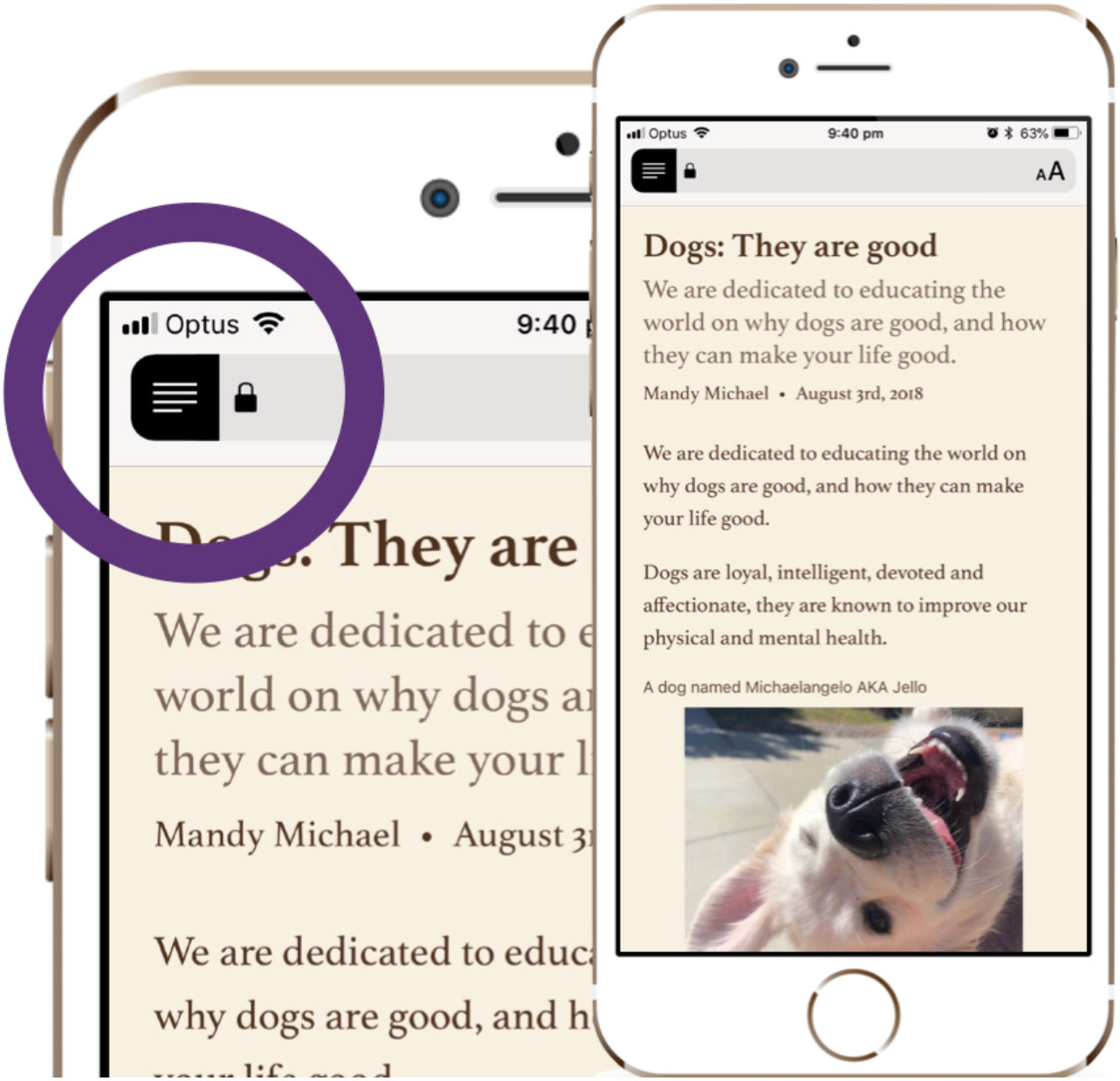


Figure 5: Article displayed in Pocket's reading mode

Figure 5 shows how my webpage article is rendered in Pocket's reading mode when using Semantic HTML.

top of the page under the headline providing that additional information and context around who wrote the content and when it was published.

How it looks in Safari Reader Mode



Last but not least of my tests was Safari's Reader Mode which is by far my favourite. You can access reading mode by clicking the little button that looks like lines of text in the corner of safari.

First thing to note is that I could not access the `<div>` only version in Safari Reader mode, the option was not available.

But what was most interesting is that my first attempt at Reader Mode resulted in some problems. I found that some of my content was being ignored, my headings, byline, dates, these were all being excluded from the page, even worse some of my actual page content, like lists and quotes were also being excluded.

I was really confused and spent a lot of time inspecting other people's websites and trying things to see what was going on. Coincidentally I happened to spot a couple of tweets about Reader mode by Ricky Mondello who kindly answered a few of my questions. You can read thread here it's pretty interesting.

These issues were a result of a number of things which are also key for making sure your content display correctly in apps like Instapaper and Pocket so before I get into how to get bylines and dates at the top of your content let's look at some simple things you need to do to make sure everything renders correctly.

What can we do?

Use Semantic HTML

It's always good to remind ourselves why semantic HTML is important and reading modes and apps are an excellent example why. These tools and apps take the structured content of a page and display it with custom styles.

> *Not everything that consumes your page is going to have access, or use, your CSS and JavaScript. These apps or tools will evaluate your site based on the context you have provided.*

For this reason using semantic HTML allows tools and apps to assign their own CSS and JavaScript to your elements relevant to their new context and environment.

A few additional key things to consider with Semantic HTML.

Use the sectioning elements

page and enables tools and apps to exclude content inside elements like `<nav>` and `<aside>`, which are not vital to convey the main purpose of the document.

The `<article>` element is often used by these tools to pull out just the article content of the page providing the best experience for users in reading apps like Instapaper and Pocket.

Create a clear structure and hierarchy

Clear structure and hierarchy allows other technologies to better understand what the most valuable content is on your site. This might include using elements like the heading elements e.g. `<h1>` `<h2>` etc. Heading elements create a clear document outline and describe the heirarchy of content. Other structural elements like `<header>` and `<footer>` can be used to identify parts of the page that do not form the main content region.

Similarly for this reason you can use the `<main>` element to identify the main content region. Where `<article>` is used for re-usable, re-distributable content for syndication etc, `<main>` represents the dominant content of the document. These can often be confused, but remember not all content is distributable or reusable, so in those case use `<main>`.

Make the most of built in functionality

Generally try to use native HTML elements if there is one, rather than making your own by adding functionality to the `<div>` element with JavaScript. As mentioned earlier your JavaScript may not always be available and tools and technology may not understand that your `<div>` with an `onClick()` event is actually a link, video element, button or other feature. Native elements are understood by other technologies so it is the safest way to ensure your meaning and context is carried across environments and apps with the highest likelihood of success.

Combine elements for better context

HTML is incredibly powerful when you combine elements, some elements are designed to work in conjunction with each other and when combined they provide additional meaning and context which may have otherwise been lost.

Sometimes, the elements themselves are not enough and we need to provide additional information or data along with our HTML, this brings me to structured data.

Use Structured Data

Google has a great document on Structured Data which I recommend you read, it's got a lot of useful information in it. For the purposes of my demos and examples let's focus on Microdata.

Microdata is used to apply metadata within our content on web pages. Other technologies like search engines, bots, browsers and apps can extract and understand Microdata and use it to provide better browsing experiences.

In order to use Microdata we need to make use of a vocabulary system, the most common is Schema.org which provides a collection of shared vocabularies about different types of content. Schema.org vocabularies can be used with our Microdata to mark up our HTML in ways that can be understood by reading apps, search engines and other technologies including the Apple Watch.

There are all types of content types on Schema.org including Person, Article, Creative Work, Publisher and so on. It's quite an interesting read so worth checking out to understand what your options are.

Using Microdata

In order to use Microdata in our HTML we'll make use of HTML attributes, specifically `<itemscope>` `<itemtype>` and `<itemprop>`.

Getting my name at the top of the various reading modes,

```
<p itemscope itemtype="http://schema.org/Person">
  By <span itemprop="author">Mandy Michael </span>
</p>
```

2. **Add the `<itemscope>` attribute** on my paragraph. This creates a new item, and anything inside that paragraph is considered part of the scope of my item.
3. **Add the `<itemtype>` attribute** on the paragraph. This is used to specify a vocabulary, most commonly, schema.org. Because i'm defining a Person i'm using the Person schema which describes my item's context as a Person.
4. Finally I need to **set up a child element**. I wrapped a span around my name, and added the `<itemprop>` attribute to my span. As i want to say that I am the author i set `author` as the value. This is part of the Person schema on schema.org and says to anything reading my document that "Mandy Michael" is the author of this document.

You can add all sorts of information to the page by using Microdata, I was also able to add a description, and the publication date to the top of Safari Reader Mode by specifying an `<itemprop>` value of *description* and *datePublished*.

```
<p itemProp="description">We are dedicated to educating the world on
why dogs are good, and how they can make your life good.</p>

<time datetime="2018-08-03" itemProp="datePublished">August 3rd,
2018</time>
```

What is important to note here is that technology cannot tell the difference between a paragraph with someones name it and a paragraph with some other random text, so using HTML attributes and making the most of Microdata helps us to make these relationships more explicit. This is the same idea behind using aria-attributes to provide more context and information around screen readers.

So this was fun, I hope you learned some things and your websites end up looking awesome in various reading modes and apps.

Most of all it's important to remember that HTML isn't just the foundation that we build our websites on it plays a vital role in making our websites function as expected across those platforms and technologies.

tech, not just now but in the future as well.

You might be interested to know that everything you just read is also relevant for making your websites not only more accessible to various assistive technologies but also to display correctly on different technology like the Apple Watch.

👍

Web Development Reading Mode Semantic Html