Ship Lighter Sites by Friday

Table of Contents

Introduction

For this crash course we'll create a page that will work mainly as a sales page but it can easily be adopted to almost anything.

We'll be using the command-line, along with your web browser and favorite text editor, so you'll get comfortable jumping between writing copy, tweaking HTML, and testing in the browser.

One quick convention note: in the printed version of these lessons you'll see a "/" at the end of long shell commands. That slash simply indicates the command continues on the next line so it fits in this format—when you run the command yourself, you can type it on a single line without the trailing slashes.

Day 1: Creating a simple, minimalist, pure html page

Today we're going to create a simple, minimal, pure html page. Here's a quick overview of the steps.

1. Create a directory and change into it
2. Download a simple html template
3. Add a splash of style

Create a directory and change into it

From your home directory we're going to create a directory called purehtml and switch into it. Run these two commands.

```
mkdir purehtml
cd purehtml
```

Download a simple html template

We're going to download a boiler plate html file, as index.html. Run this command.

```
curl -o /
  index.html /
  https://neat.joeldare.com/blank.html
```

Now open the index.html file in your web browser to take a look.

Add a splash of style

We'll add a tiny bit of style by downloading neat.css. Neat is a minimalist css file that's just 3Kb.

```
curl -O /
  https://neat.joeldare.com/neat.css
```

Open the file in your browser again to see the new style. Notice the page works in both light and dark modes.

You can also read my article about Why I'm Writing Pure HTML & CSS if you want to dig into the mindset behind this approach.

Now that you have the base page, add your own title and a little bit of text. Tomorrow we'll add an image and a call-to-action button and we'll talk a little bit about semantics and accessibility.

Day 2: Inserting an image and a call-to-action

Today we're going to add an image, a call-to-action button, and we'll talk a little bit about semantics and accessibility. Here's a quick overview of the steps.

1. Add an image
2. Create a call-to-action button
3. Use semantic HTML tags

Add an image

Open the index.html file that you created yesterday.

Just below the body text, add an image. Here's the code:

```
<img class="bordered" src="sailboat.jpg">
```

We used a Neat CSS style. The class adds a solid border around the image.

Use the following command to download a sample image.

```
curl -O /
  https://joeldare.com/media/sailboat.jpg
```

Now view the page in your web browser.

Create a call-to-action button

Just below the text body, add a new div and a link. Here's the code:

```
<div class="center">
  <a class="button breathe" href="">
    Buy Now
  </a>
</div>
```

This creates a centered "Buy Now" button at the bottom of the page.

We used a couple more Neat CSS styles. The `center` class centers everything in the div. The `button` class makes the link (a or anchor tag) look like a button. The `breathe` class gives the button a little extra white space, allowing it to breathe.

Set the `href` to any URL you want. You could link to a third party payment page or input form, for example.

Use semantic HTML tags

Use semantic HTML tags to make your web pages

easier to read and understand. Semantic tags like `<header>`, `<main>`, `<article>`, and `<footer>` describe the structure and meaning of your content, instead of just how it looks. This helps with accessibility, search engine optimization, and makes your code easier to maintain.

I originally wrote the following code for this sample:

```
<p class="center">
  <a class="button" href="">
    Buy Now
  </a>
</p>
```

The `<p>` (paragraph) and `<a>` (anchor/link) tags can also be semantic because they describe the purpose of the content they contain, but the `<p>` tag doesn't contain a paragraph of text, so it's not really semantic.

You might also like my article about The 14kb Problem where I talk about how I craft pages that load in a single round trip.

Depending on how you feel about call-to-action buttons, you might be interested in my business journey in How I Lost Money with 25 Years of Failed Businesses.

Now that you have an image and a call to action button the page is nearly complete. Tomorrow we'll talk about

improving image performance.

Day 3: Performance quick wins for large images

Yesterday you downloaded my sample image of a sailboat. Today we're going to talk about how I optimized that image. Here's a quick overview of the steps.

1. Resize the image
2. Interlace the image
3. WebP alternative

Resize the image

You might have noticed that it downloaded pretty quickly—it's only 28K. I used ImageMagick to reduce the original size from 1.8MB to 28K. Here's the command.

```
magick /
  sailboat.png /
  -resize 800x /
  -strip /
  -quality 75 /
  sailboat.jpg
```

This takes the original image, `sailboat.png`, resizes it to 800 pixels wide and saves it as a JPG with a quality of 75%. That reduces the image quality quite a bit but also

makes it much, much smaller. Smaller means faster, cheaper, and less carbon. It's especially useful for users on slower internet connections, such as those in rural areas.

Interlace the image

Interlacing causes the image to display in several passes so a coarse version shows first and progressively sharpens as more data arrives. This will only be noticeable on very slow connections.

Add `-interlace Plane` to the command to create a progressive image. Here's the full command with all the options:

```
magick /
  sailboat.png /
  -resize 800x /
  -strip /
  -interlace Plane /
  -quality 75 /
  sailboat.jpg
```

Now you've got tiny images. Tomorrow we'll talk about adding your unique style.

Alternative image types

A user asked, "why use the JPEG format and not WebP?"

I use JPEG for the broadest compatibility and for historical reasons but it might be okay to switch to WebP.

We're probably to the point where WebP is common enough. It looks like the last browsers got support around 2020.

If you prefer WebP, the following command will work:

```
magick sailboat.png /
  -resize 800x /
  -strip /
  -quality 75 /
  sailboat.webp
```

WebP images will be smaller than their JPEG counterparts. Keep in mind, however, that WebP doesn't support progressive images. On a really slow connection, a larger progressive JPEG might still feel faster than a smaller WebP image.

If you're interested in other ways to use ImageMagick, you might like these articles:

Reducing Animated GIF File Size with ImageMagick

Rounding Image Corners with ImageMagick

Combining Multiple Images into an Animated GIF with ImageMagick

Stack Images with ImageMagick

Day 4: How to customize a page with your unique style

Today we're going to add some custom style to the page. Here's a quick overview of the steps.

1. Add a custom stylesheet
2. Center your headings
3. Change your link colors

Add a custom stylesheet

It's easy to customize pages using Neat. The best way is to create a new `custom.css` file and then add the following line to the head of your page.

```
<link rel="stylesheet"
  type="text/css"
  href="custom.css">
```

That will give you the option to customize any aspect of your page and to update `neat.css`, if you ever need to, without losing any of your personalizations.

Here are a couple examples...

Center your headings

If you wanted all your headings to be centered you could add the following to your `custom.css` file.

```
h1 {
  text-align: center;
}
```

Change your link colors

If you wanted all your links to be a different color you could add something like the following to your `custom.css` file.

```
a {
  color: red;
}
```

These are just a few simple examples. You can change anything about the design.

Now that you've added your own style, tomorrow we'll talk about starting with plain-text markdown.

Day 5: Writing the copy first with Markdown

Sometimes I find it useful to start with my copy and build my page after I've figured out what I want to say. Here's a quick overview of the steps.

1. Start with your message (copy first)
2. Outline in Markdown
3. Replace placeholders with real copy

What “copy-first” means

Copy-first design means writing the actual words (the “copy”) before creating layouts or visuals. This puts your message first, making sure the design supports the content, not the other way around. I start by outlining sections, headlines, and body text before thinking about formatting or images.

Create a simple Markdown outline

Here's an example of a simple Markdown file that I might start with:

```
# My Heading Here

Some main text goes here...
```

```
## Another Section

Even more text goes here...
```

Replace placeholders with your real copy

Then I replace the placeholder text with whatever I want to say.

I host entire sites on GitHub with nothing but Markdown. Speaking of which...

This is the last micro-lesson, but I've got one more bonus for you. Tomorrow we'll talk about how you can host a static site on GitHub for FREE.

Bonus: Deploying your static site on GitHub

Now that you have a static HTML site, you can host it free on GitHub. Here's a quick overview of the steps.

1. Create a repository
2. Initialize and push the repository
3. Enable GitHub Pages

Create a repository

Head over to GitHub and create a new public repository. Give the repository a name, then click Create.

Initialize and push the repository

In your working directory, initialize a local repository and make your first commit:

```
git init
git add .
git commit -m "first commit"
```

Add GitHub as your `origin` (replace `[url]` with the URL GitHub gave you):

```
git remote add origin [url]
```

Push your code to GitHub:

```
git push -u origin main
```

Enable GitHub Pages

Back on GitHub.com, go to **Settings → Pages**. Select the source branch as `main` and click Save.

Wait a couple of minutes and refresh. At the top, GitHub should say, "Your page is live at..." followed by a URL. Click it to see your site.

That's it!

If you want to set up a custom domain, I explain my process in Setting up a Domain.

More information

GitHub also has a good set of detailed instructions for the entire process:

Websites for You and Your Projects

Share your feedback

Now I *really* want to hear from you. What would you like to see done differently? Where did you struggle? Was anything missing, confusing, or just not clear enough? Absolutely anything that crosses your mind is fair game.

If you have ideas or suggestions, please submit them (and vote on other people's feedback). Your notes directly shape the next version of this course.

Neat CSS Feedback Page

Thank you for helping make Neat CSS, and this course, even better.