CSS for Paged Media (Pt 3 Paged Media)

Objective: Add CSS for paged media utilizing an alternate (supporting) user agent

Starter files: You will *not* use the Canned Fruit site in this part. New starter files are provided in Moodle. You will continue using Canned Fruit in Part 4.

Introduction

CSS-for-print has been around forever. You know – creating styles that apply only to print to do things like hiding navbars and non-essential images, removing all background colors, setting all text to black, setting a serif font, … There are plenty of tutorials and guidelines on the Internet for that.

The focus for this assignment is *paged media.* Specifically, electronic paged media. While regular old CSS-for-print means the printed pages are *paged* – our focus today is to generate an eBook in PDF format. This means it should be formatted like a physical book with page numbers, table of contents with page numbers, controlled page breaks, and usable cross-references.

Many of the techniques you learn today can easily be adapted to regular CSS-for-print. In fact, today is essentially CSS-for-print plus a bunch of other cool stuff.

Many of the features we need to add to our code is accomplished with the @page rule. Unfortunately, no modern browser fully supports @page so we won't be able to use a normal browser to test our pages. We'll start with a simple sandbox example to test this and then we'll download a user agent that not only supports @page, but will also convert our HTML/CSS/JS to a PDF eBook.

Part 3a: Emulate Printing in The Viewport

1. Open **print-sandbox.html** in your editor and again in your browser. (Please use Google Chrome because we'll need a developer tool from Chrome in a minute.)

2. Create the following CSS file and save it as indicated in the HTML code line 7:

```
1  h1 {
2    border-bottom: 10px solid #333;
3  }
4
5  h2 {
6    border-bottom: 3px solid #333;
7  }
```

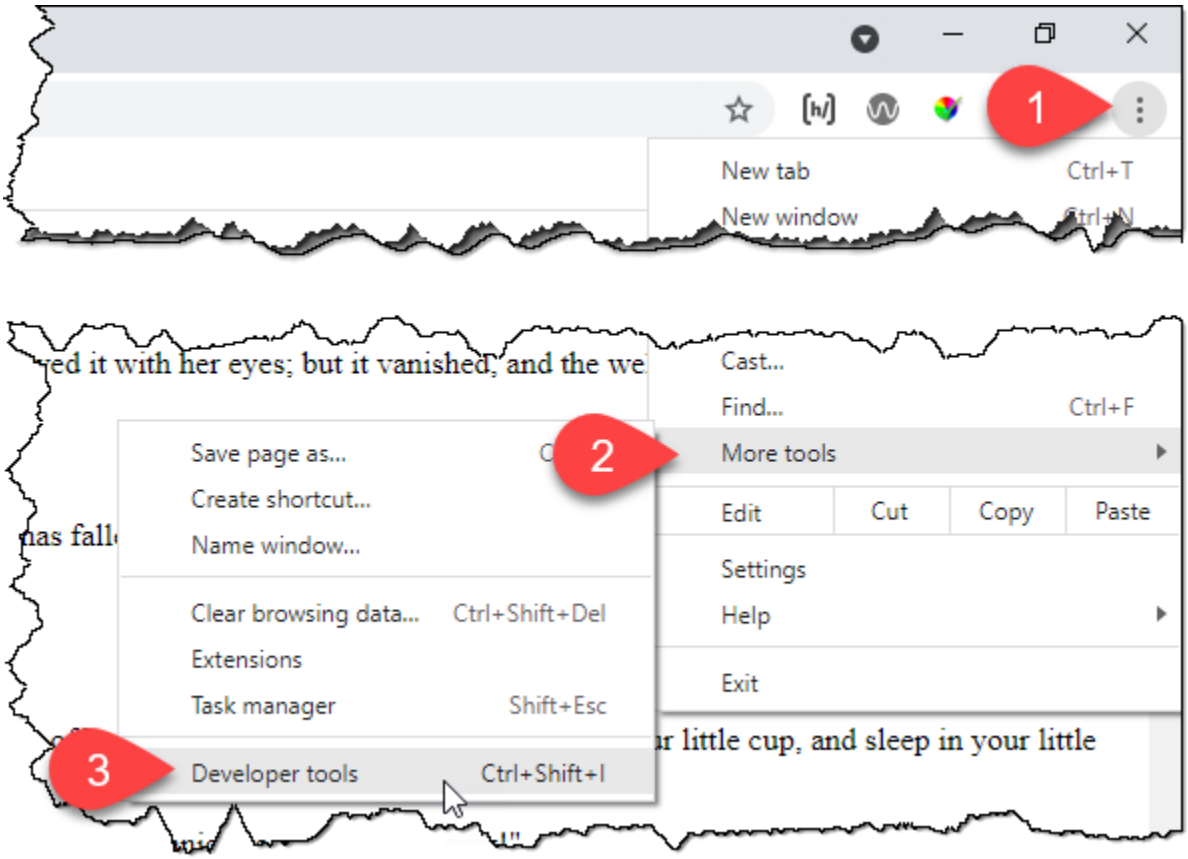
3. Test the HTML file in your browser.

   The headings fail to display their bottom border because our CSS only gets applied when the page is printed.

4. Tap [Ctrl]+[p] to bring up the Print Preview window and notice the preview shows the headings with their expected underline.

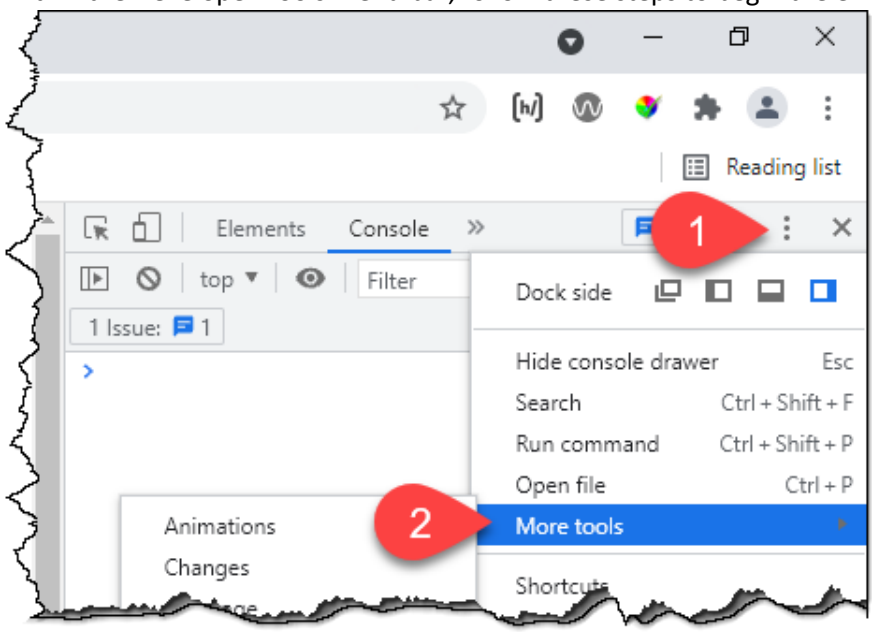
5. Click [Cancel] to close the print preview window.

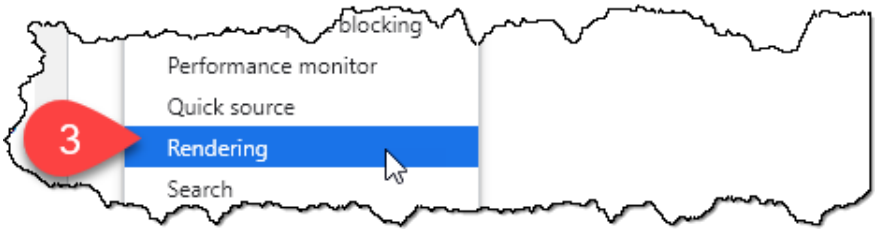
6. It’s inefficient to have to bring up the print preview dialog every time you want to test your page. Instead, use Google Chrome’s Developer Tool feature to emulate printing. First, display the Developer Tools:



1) Click the **Options** menu.
2) Click **More Tools**.
3) Click **Developer Tools**.

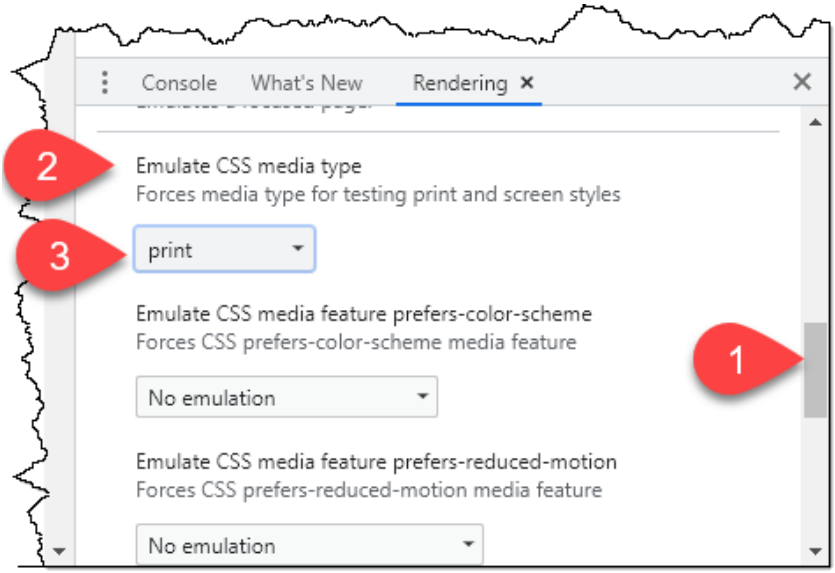
7. Within the Developer Tools menu bar, follow these steps to begin the emulation process:





1) Click the **Options** menu *in the Developer Tools menu bar*.
2) Click **More Tools**.
3) Click **Rendering**.

8. Follow these steps to emulate print CSS in the viewport:



1) At the bottom of the Developer Tools sidebar, **scroll down** and…
2) …locate the **Emulate CSS media type** section.
3) Set the option to **Print**.

The viewport now displays the print CSS and we can see the borders below the headings.

Part 3b: Discover Why Emulation Won't Work for Us

Normally, this print emulation works great. But it fails as soon as we try to work with paged media. That's because the viewport doesn't have pages. It scrolls infinitely.

Let's force each fairy tale to start printing on a new page.

1. Edit the CSS to force a page break before each H2:

```
5   h2 {
6     border-bottom: 3px solid #333;
7     page-break-before: always;
8   }
```

2. Save and preview. As you scroll down, there are no pages! The viewport just scrolls down one lone page. The only way to see individual pages is to go back to the print preview.

3. **Click anywhere inside the viewport to shift focus away from the Developer Tools sidebar** and then tap [Ctrl]+[p] to display the print preview dialog.

4. Scroll through the print preview and notice each H2 prints at the top of a new page.

   While you can use a combination of CSS print emulation and print preview to test your pages, there are some things that just don't work at all. That's coming in the next section.

Part 3c: The @page Rule

The @page rule is used to style paged media. Some features work in a regular, but most don't.

1. Add a new CSS rule to specify the page size as 6-inches wide by 4-inches tall. This isn't a standard size for books, but it should be obvious that something happened when we preview it.

```
10  @page {
11    size: 6in 4in;
12  }
```

2. Preview in the browser and notice that because the viewport isn't paged, we don't see the new page size. We'll have to use print preview.

3. Tap [Ctrl]+[p] to open the print preview dialog and notice the page size has changed. There's a huge margin around the text that should be reduced – which we'll attack next. At least we've confirmed the browser does support @page. Kind of. Close the print preview dialog,

4. Edit the CSS rule to tame the huge margins:

```
10  @page {
11      margin: 1cm;
12      size: 6in 4in;
13  }
```

5. Save, refresh the browser, and display the print preview dialog window. The margin declaration didn't work. The same huge margins are still there. Browsers support some @page features (like *size*) but not all (like *margin*). We'll need to use a different user agent.

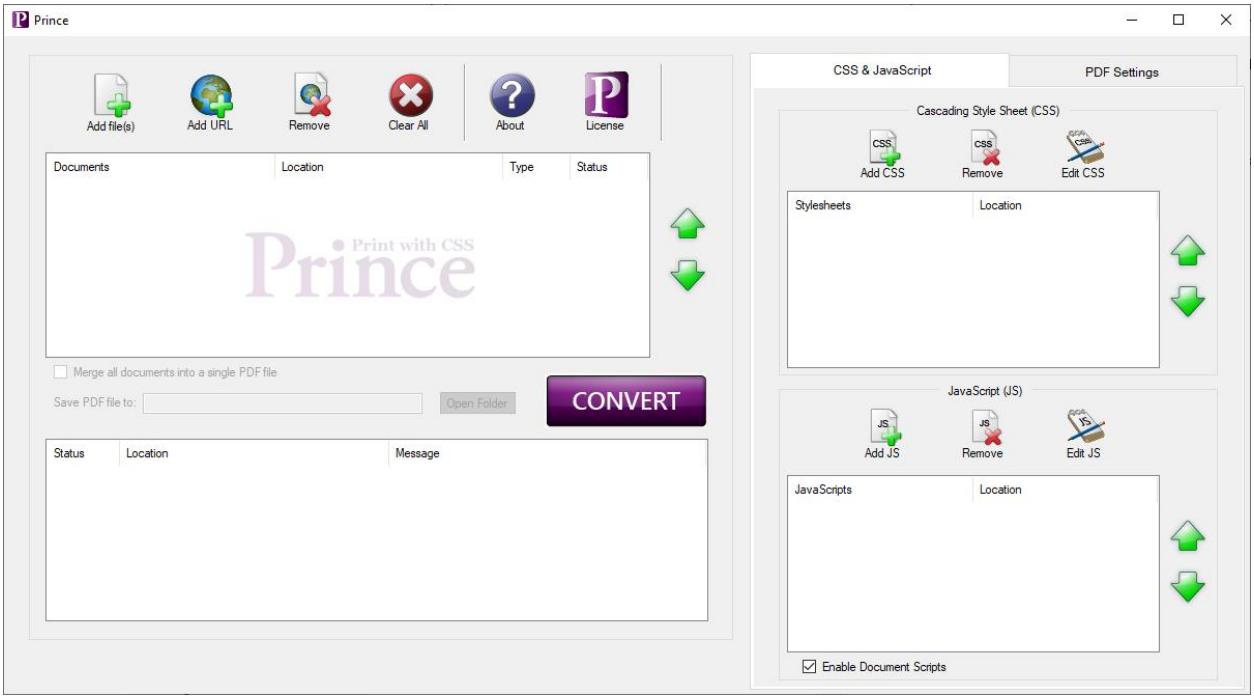
Part 3d: Prince

Prince is an alternate user agent that converts HTML/CSS/JS to PDF. You can use it free for personal use, but need to pay for a commercial license if you're going to use it to make money. There's no viewport in Prince. You need to use it to export your code to a PDF and then open the PDF. A little tedious, but not too bad once you get into the groove of the workflow.

Download and Install Prince

1. Head to https://www.princexml.com/ and figure out how to download and install it.

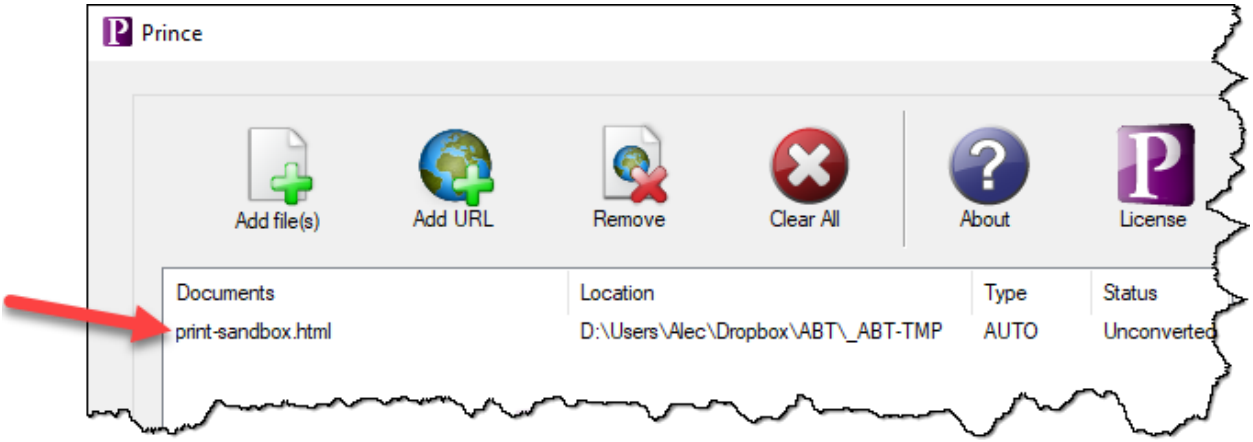
2. Once it's installed, launch it and pin it to your Taskbar (or Dock) for easy access. The default window looks like this:



3. Heads-up – we'll use only the box in the top-left area to load our HTML file. You don't need to use the CSS area (top-right) or JS area (bottom-right) if your files are properly linked. In fact, using those areas can lead to unexpected and unwanted results!

Convert a File

4. Either use the **Add Files** button at the top-right to browse to and load your HTML file, or just drag the **print-sandbox.html** file into the top-left box. I prefer to drag. When done, it should look like this (of course, your Location string will be different):



5. Click the big purple **Convert** button. There will be a brief flash and it's done. Be sure to examine the lower-left box for errors. There shouldn't be any!

6. The PDF was created in the same folder where the HTML file exists, so navigate to that folder and locate the PDF.

7. Open the PDF in any app that can open it. You'll see the page size (6x4) was supported and the margin (1cm) was supported. You'll also see an icon in the top-right corner of the first page that the free version of Prince inserted. It you used Adobe Acrobat to view the PDF, you can delete that icon as it's just a PDF comment.

8. Here are the general workflow steps:
    a. Write your HTML/CSS/JS code.
    b. Load the HTML file into Prince and Convert it.
    c. Test the generated PDF.
    d. Quit the PDF viewing app and delete the PDF.
    e. Edit your code to make changes based on what you just previewed.
    f. Click Convert (no need to reload the HTML file).
    g. Repeat Steps C-F until you're satisfied with the PDF.

Summary

This was a gentle introduction to CSS for paged media. Actually – we hardly did anything with @page. That will come in the next part of this assignment series. Get ready because that will require a lot of reading, research, and self-study! Be sure you are comfortable using Prince before then.