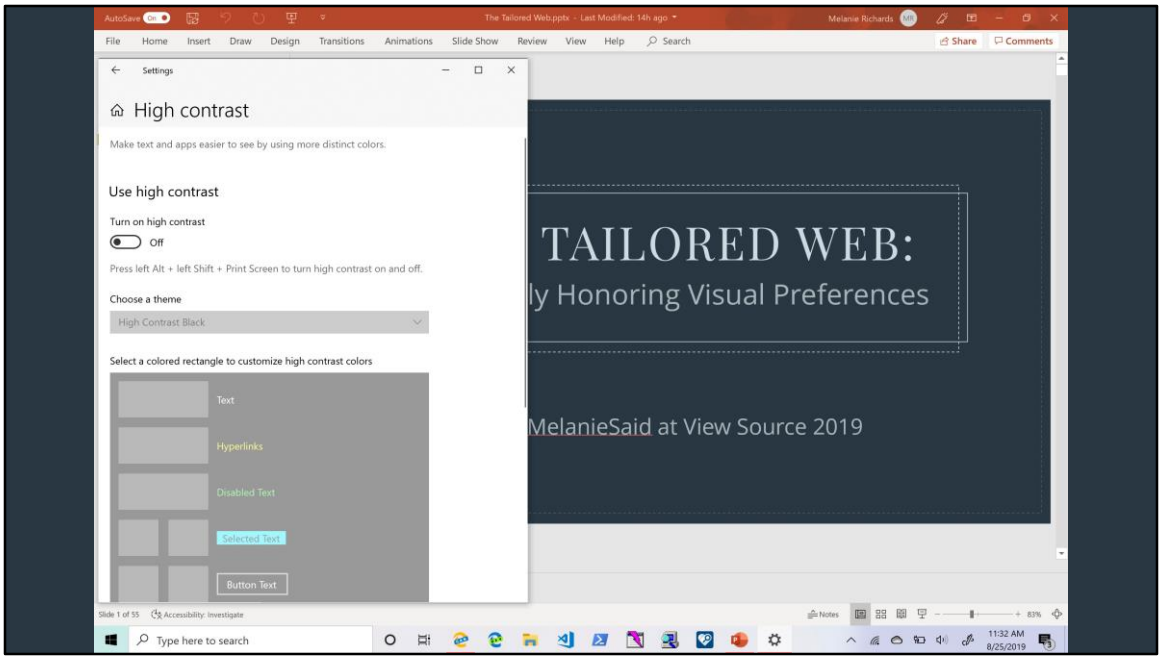
FINESSING FORCED-COLORS:

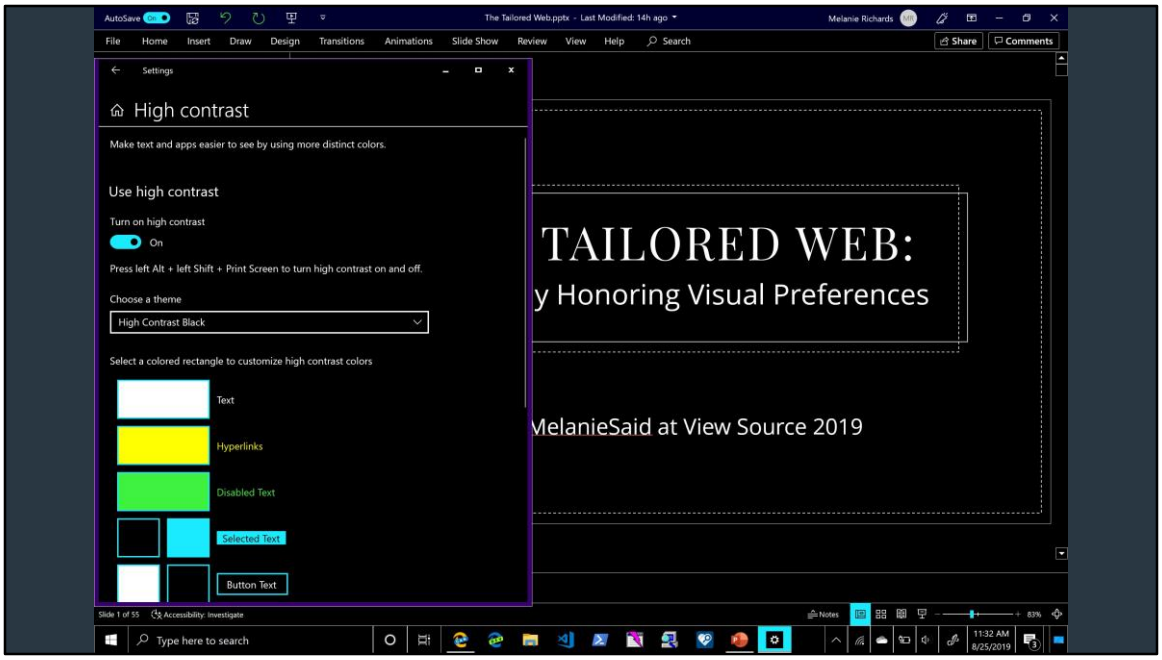
Tailoring the High Contrast Experience

@soMelanieSaid at TPAC 2019
Web Developer Meetup

Hi everyone, I'm Melanie, a member of the Microsoft Edge web platform team and participant in W3C web standards communities, including the CSS Working Group.



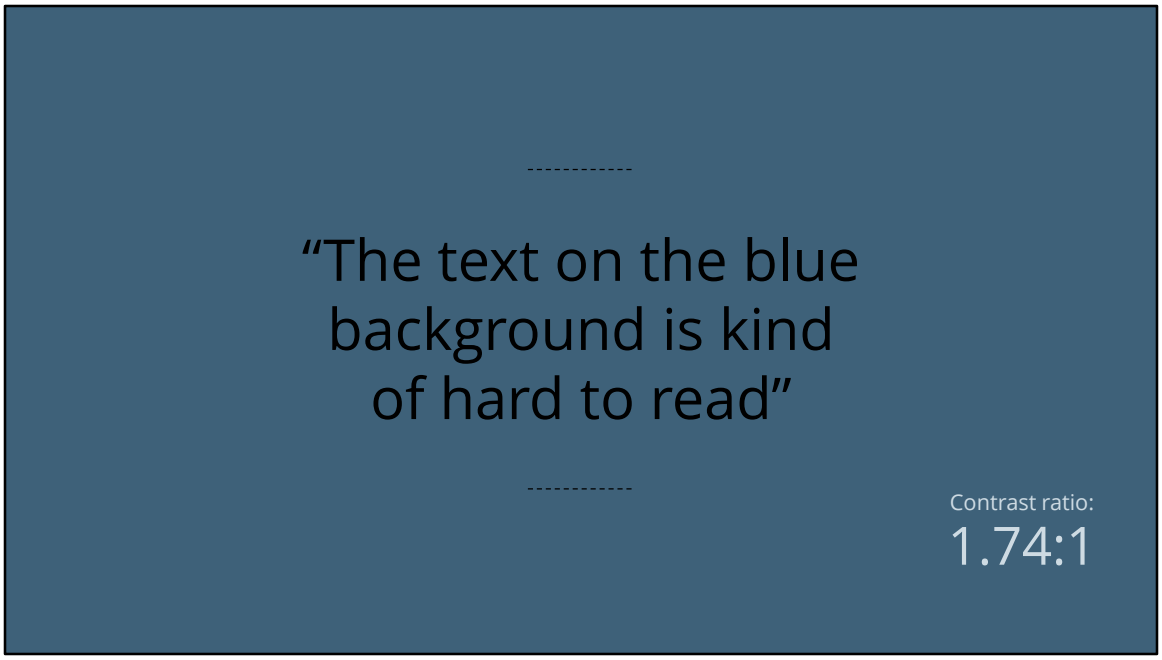
The CSS working group has recently been developing standards for integration with user system settings, such as Windows High Contrast. If you're unfamiliar, High Contrast is an accessibility feature intended to increase text legibility while reducing cognitive load. The feature works by enabling the user to select theme colors for a scoped number of semantic elements.



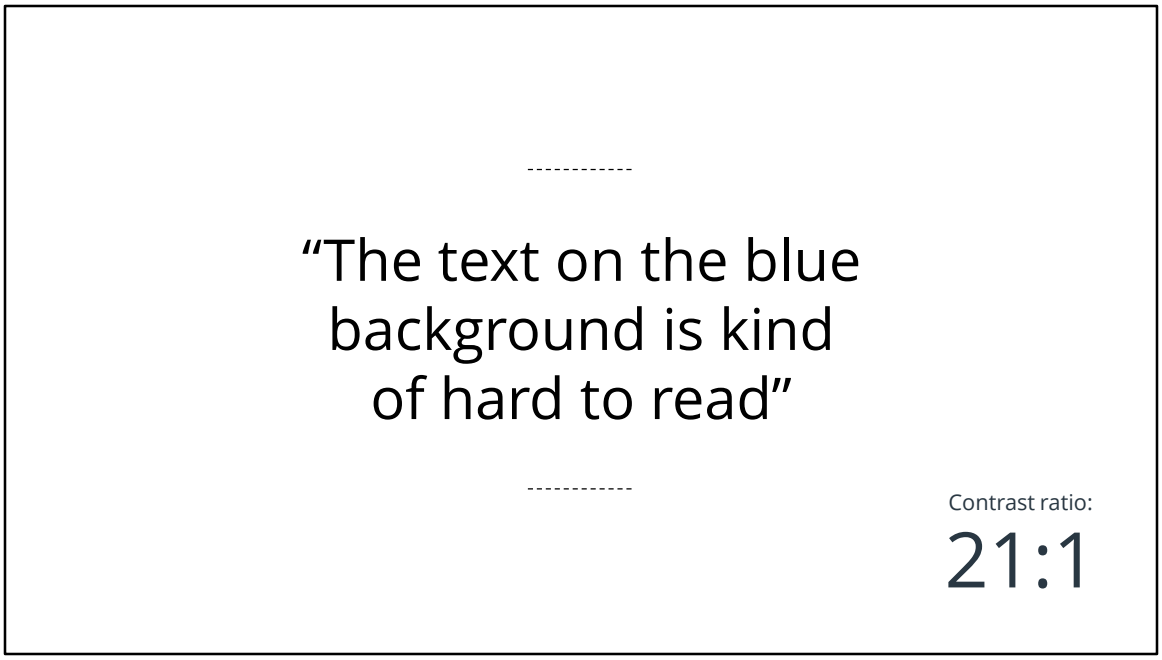
This scheme can then be applied to user interfaces and app content, reducing visual complexity while guaranteeing the user's preferred contrast level. Here's an example of the operating system and some apps rendered in a High Contrast Black theme; if you don't have a Windows machine, I'll share later on in the presentation how you can experiment with this feature.

In any case, like other apps, browsers can render their UI and even web content in the user's provided color scheme. In parallel to bringing High Contrast styling techniques to standards, I was part of a usability study for High Contrast on the web.

We had some folks who identify as having low vision go through websites with and without High Contrast enabled. One individual, while going through a site rendered in its original color scheme, remarked: "The text on the blue background is kind of hard to read."



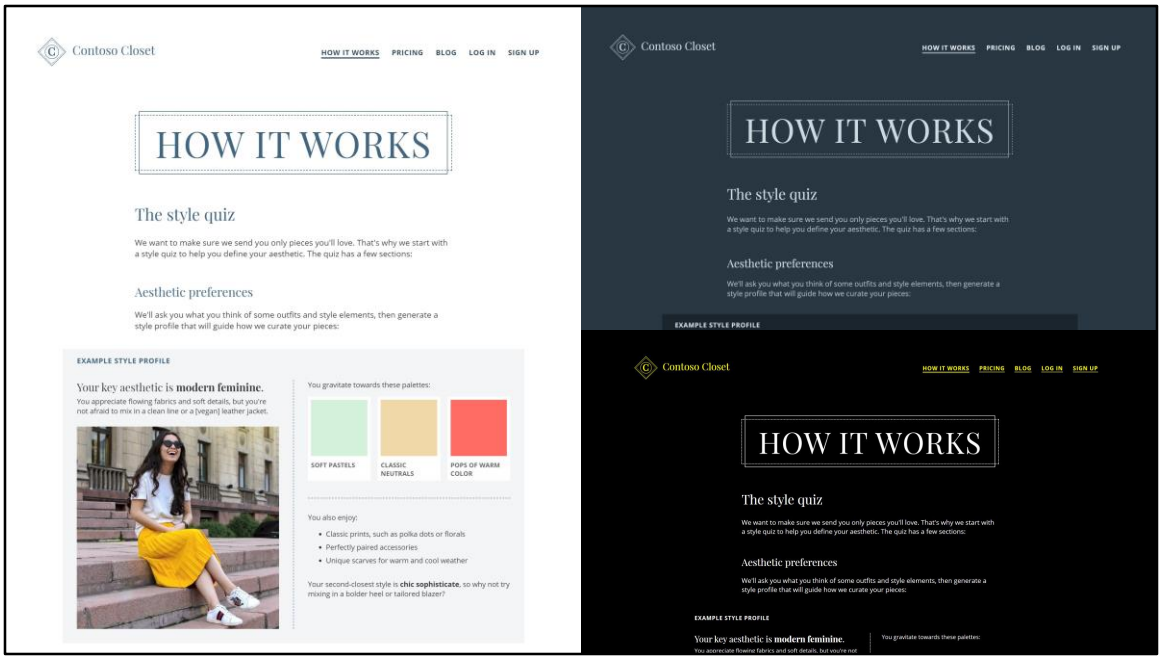
You might think that this was some type of shenanigans where the designer put dark text on a dark background, like some footer text they were trying to make more subtle.



But actually, the participant was referring to black text on a white background. You cannot have a higher foreground-background contrast ratio than this. And still, that person's eyes were picking up on the blue light, and they had trouble reading the text.

We turned on High Contrast with a dark theme, and the participant said "that's much better!" This is the same exact contrast ratio, but in a color scheme that was easier for this particular individual to read.

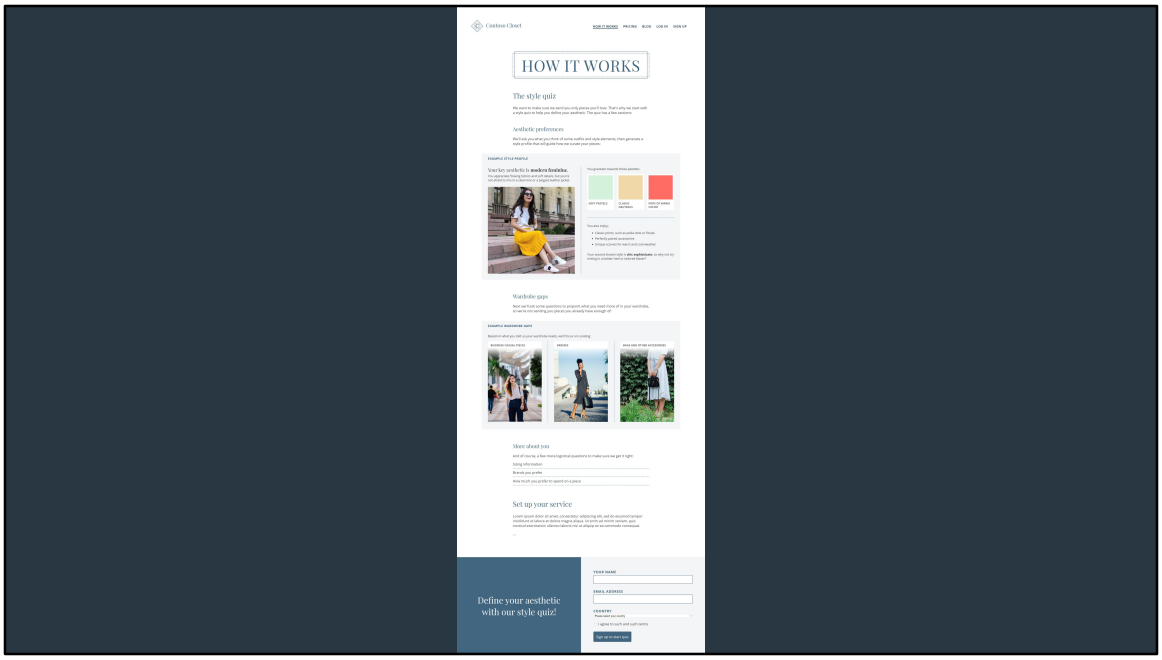
While we may try to provide reasonable defaults in our styles, color and contrast needs varying from person to person. These are highly contextual, and can change by the day or even the hour. Maybe someone had a recent injury that makes it hard to look at certain colors. Or they find that a simplified color scheme helps them focus better. Or maybe they're trying to finish a major project in the middle of the night without keeping their partner up.



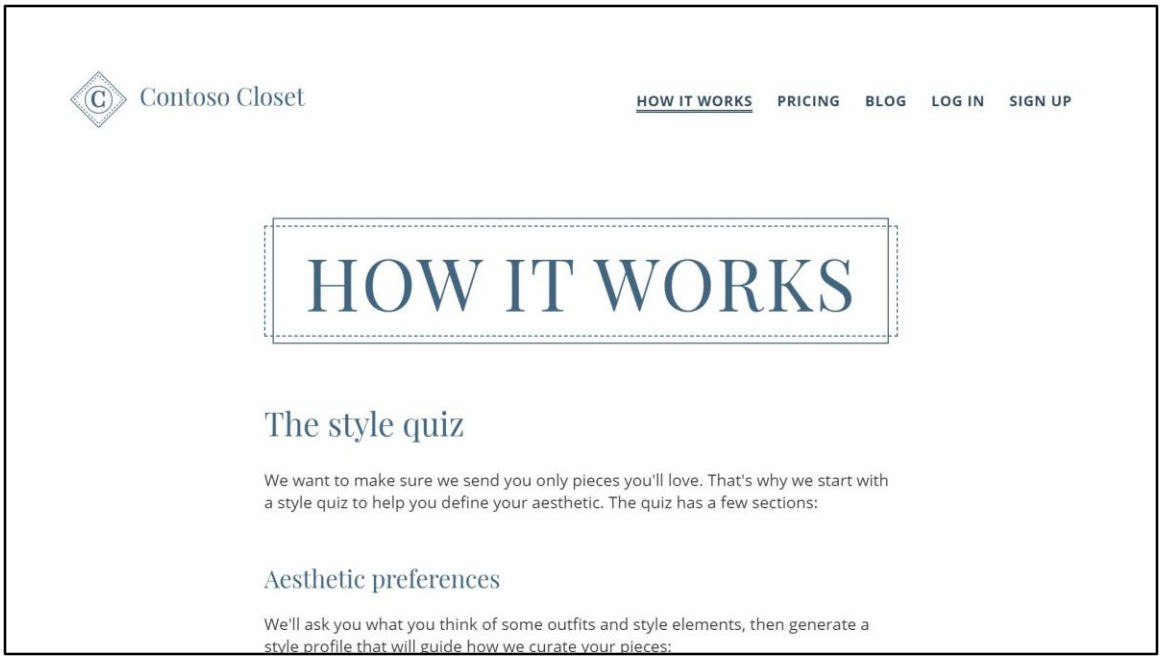
The good news is, we can take a layered approach towards inclusive color and contrast. Emerging media queries and CSS properties enable us to respond to hints as to the user's preferred color scheme, contrast preference, and the presence of forced color modes like High Contrast.

We can tailor our websites to users' visual needs.

With these new standards in hand, we can efficiently tailor our websites to users' visual needs.



Let’s suppose we’re building out a marketing page for a service that sends you clothing tailored to your individual preferences.



Zooming in a bit here, our base brand is a light theme with cool accent colors.

EXAMPLE STYLE PROFILE

Your key aesthetic is **modern feminine**.

You appreciate flowing fabrics and soft details, but you're not afraid to mix in a clean line or a [vegan] leather jacket.

You gravitate towards these palettes:

SOFT PASTELS

CLASSIC NEUTRALS

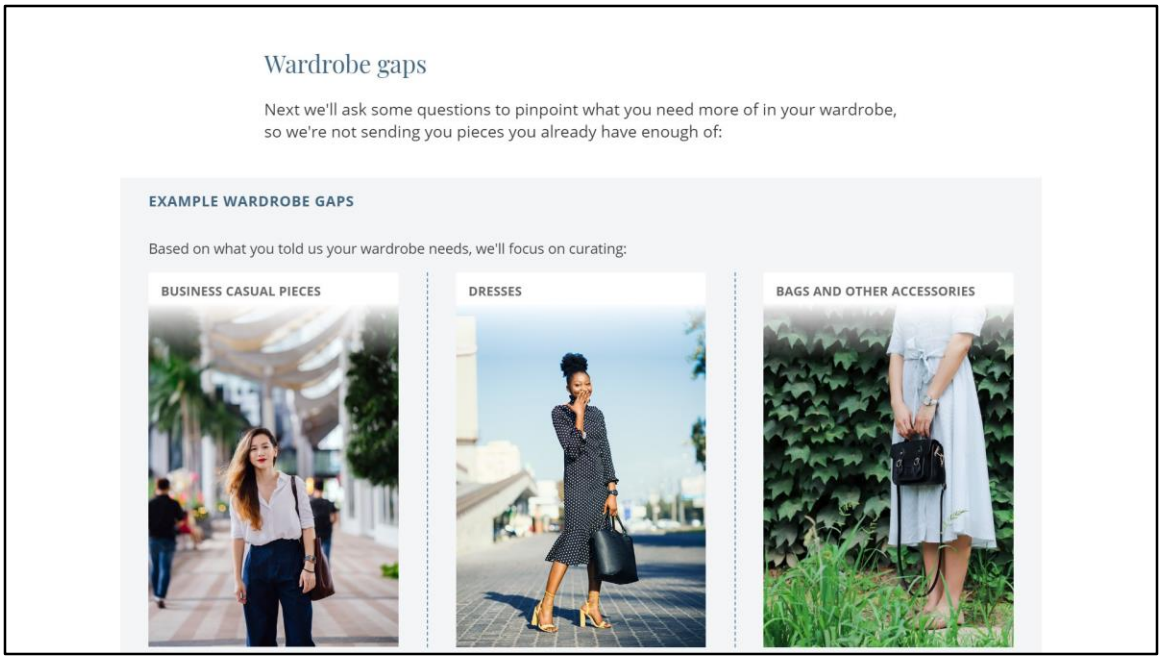
POPS OF WARM COLOR

You also enjoy:

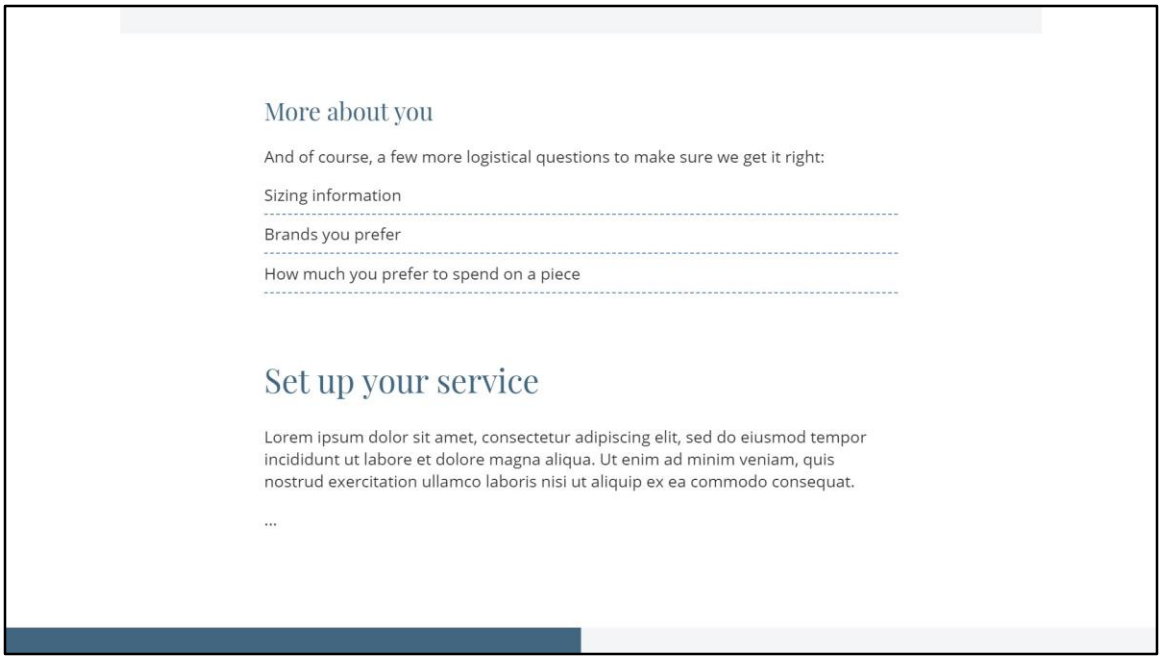
- Classic prints, such as polka dots or florals
- Perfectly paired accessories
- Unique scarves for warm and cool weather

Your second-closest style is **chic sophisticate**, so why not try mixing in a bolder heel or tailored blazer?

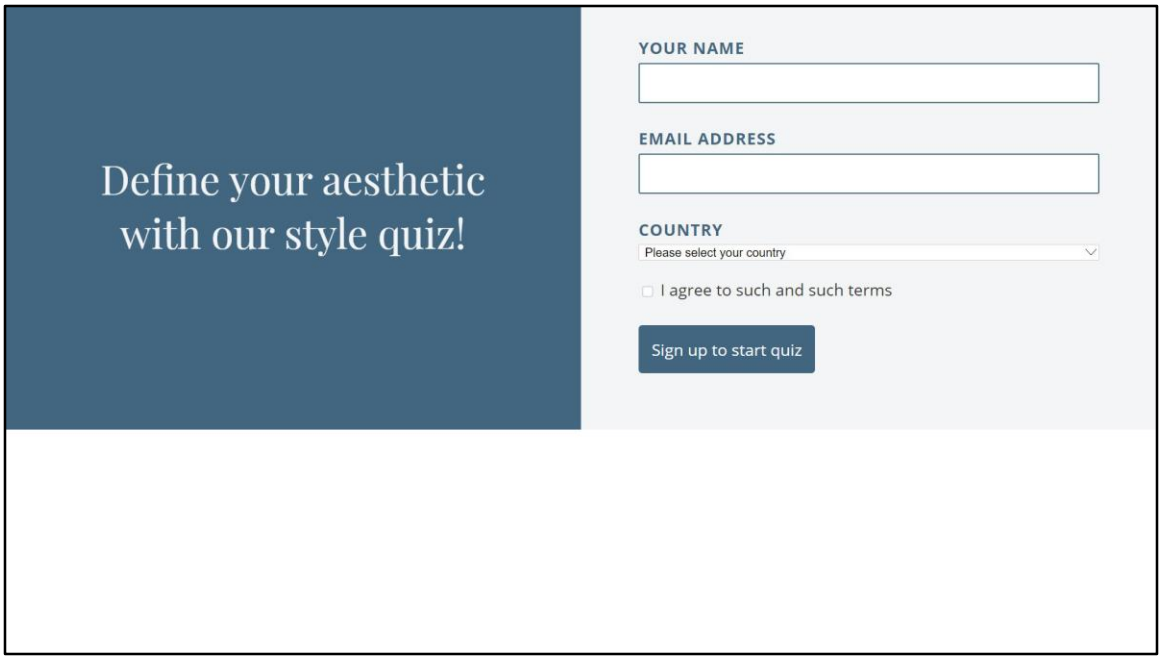
We've got some mixed content here that includes a sample color palette,



Gradients overlaid on top of photography,



As well as decorative dashed borders.

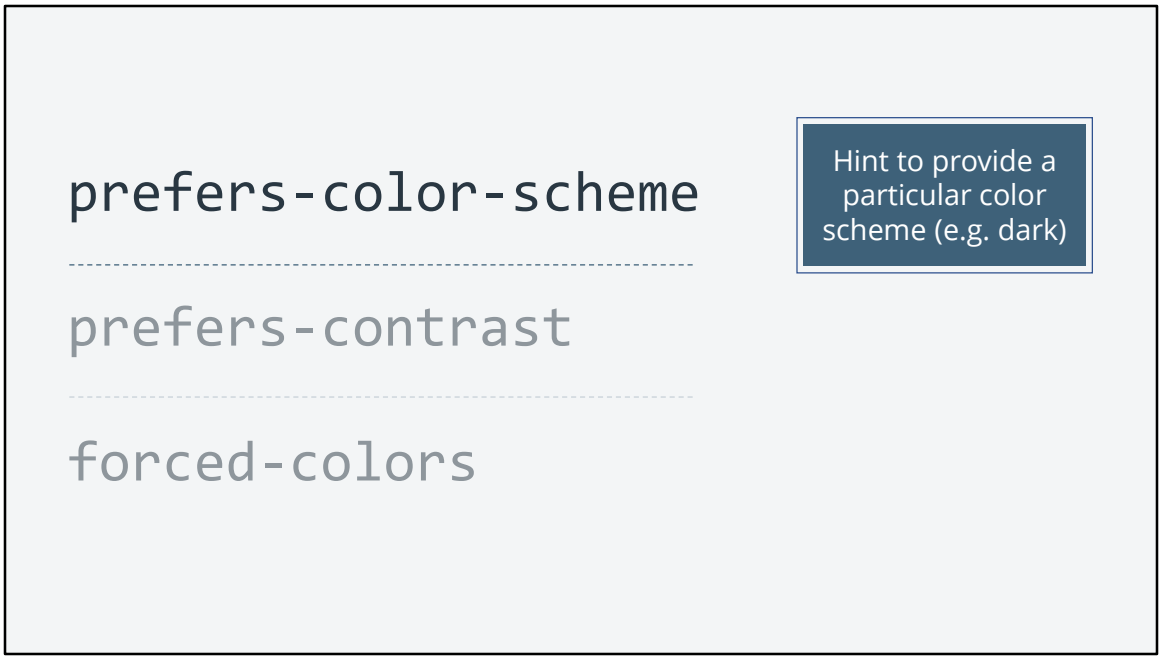


We also have some form fields at the bottom of our page.

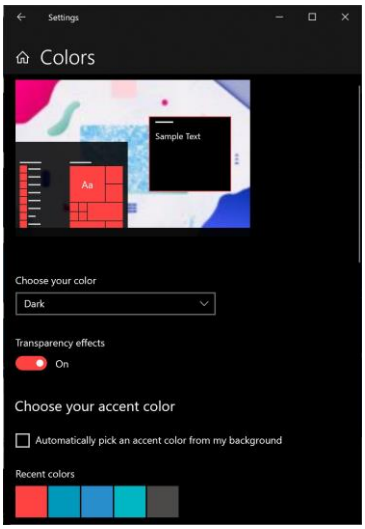
A FEW RELATED MEDIA QUERIES

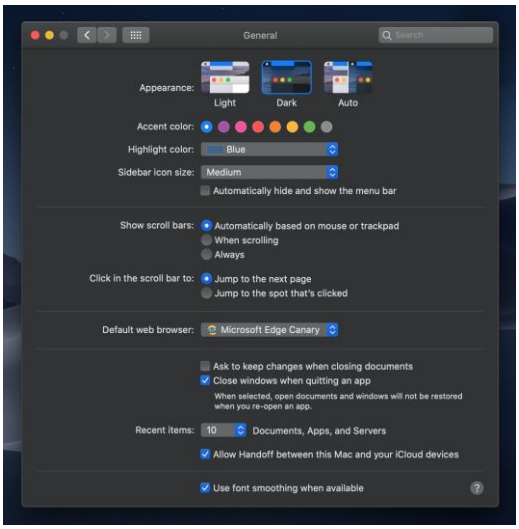
There are a few different media queries that we can use to adjust our theme for color and contrast settings.

prefers-color-scheme, prefers-contrast, and forced-colors



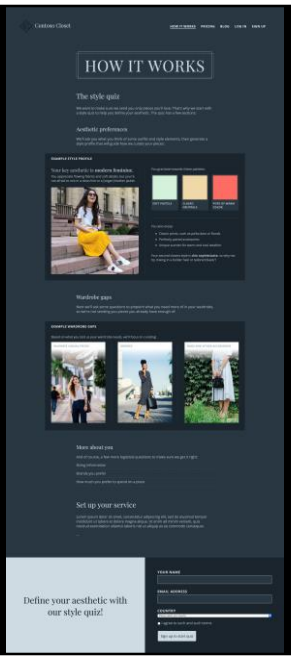
Prefers-color-scheme is a hint to provide a particular color scheme.



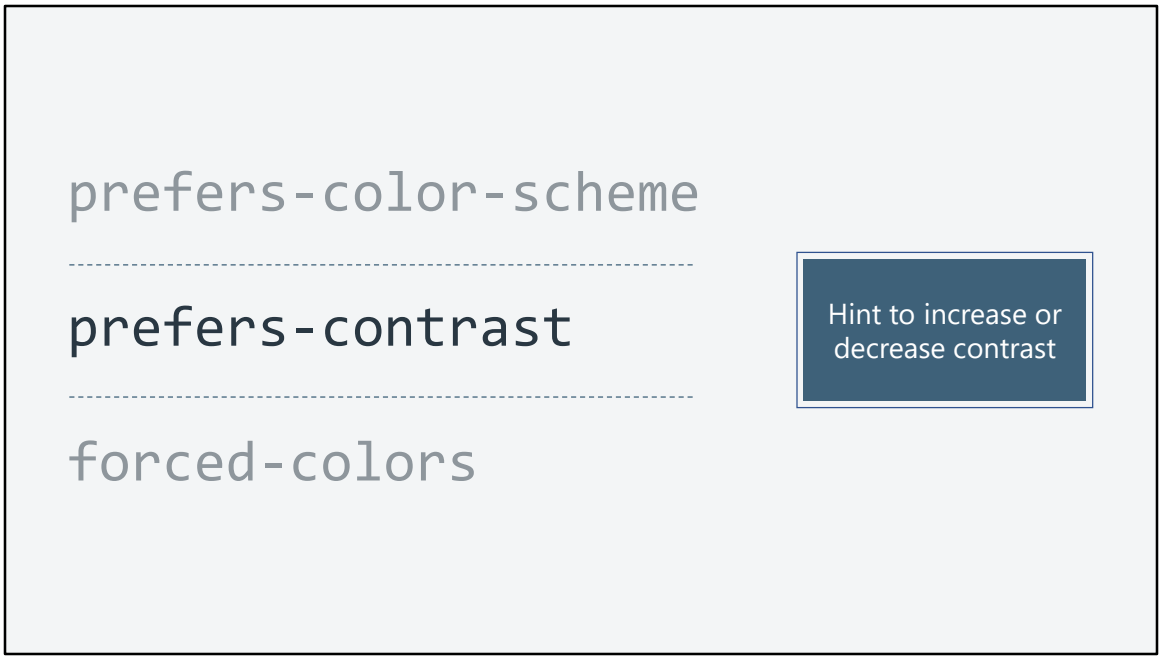


This hint can come from the browser itself or from the operating system, which can provide settings like light mode or dark mode.

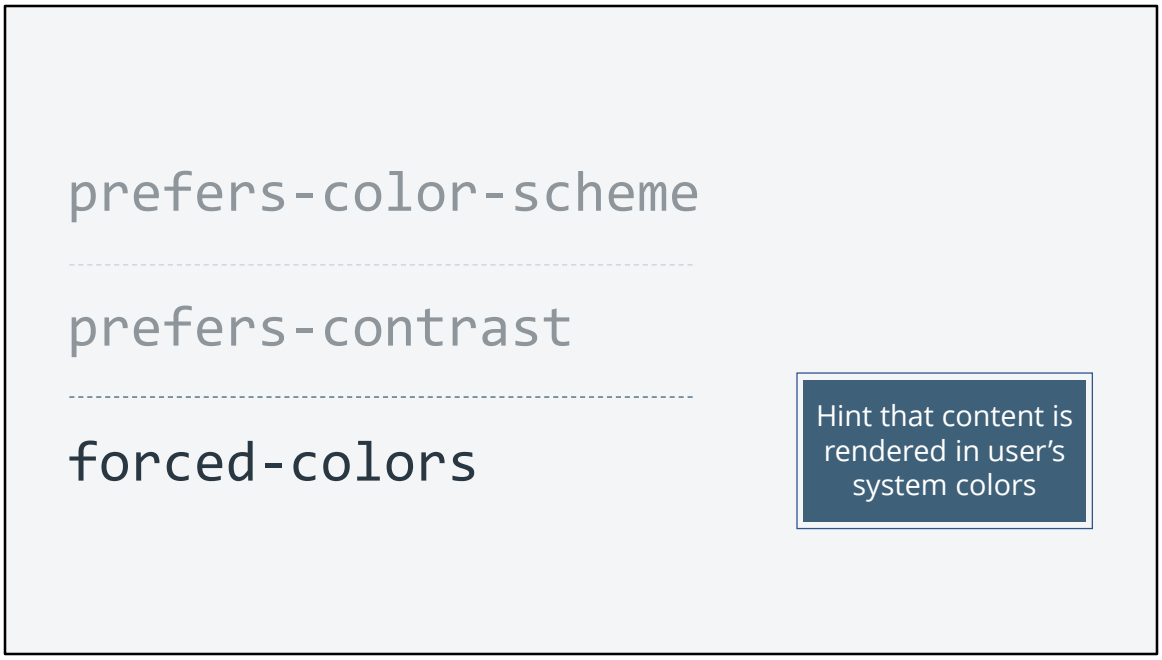
```
@media (prefers-color-scheme: dark) {
  /* “Dark mode” styles */
}
```



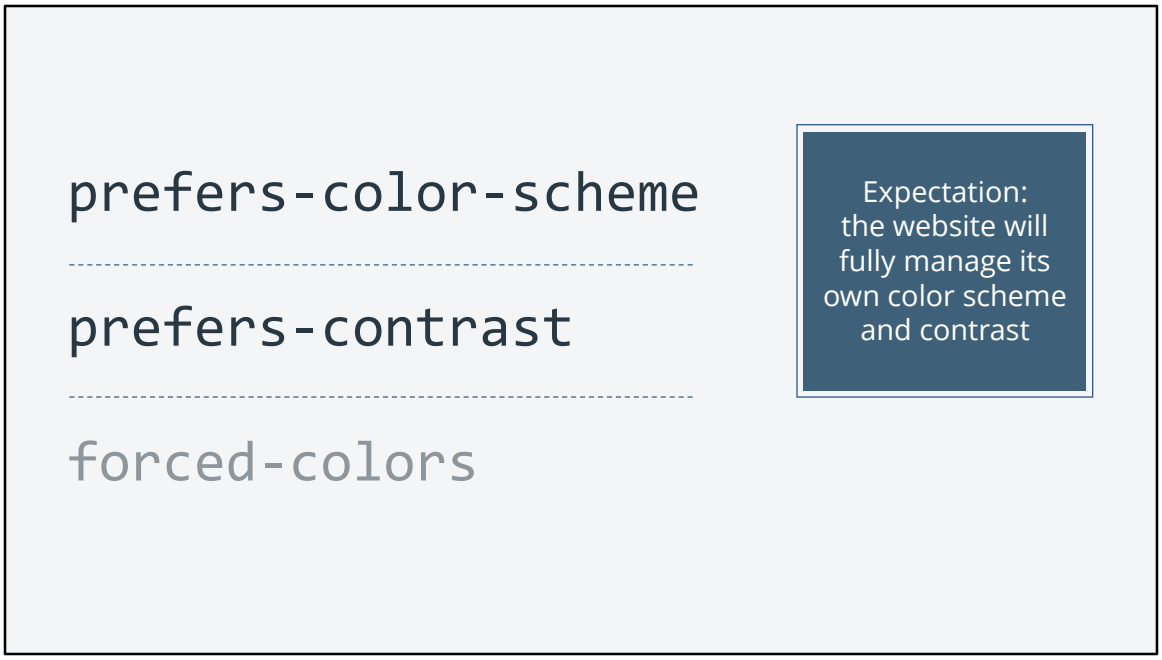
In fact, our demo page provides a dark blue scheme whenever prefers-color-scheme: dark evaluates true. A link to the full demo code will be at the end of the presentation.



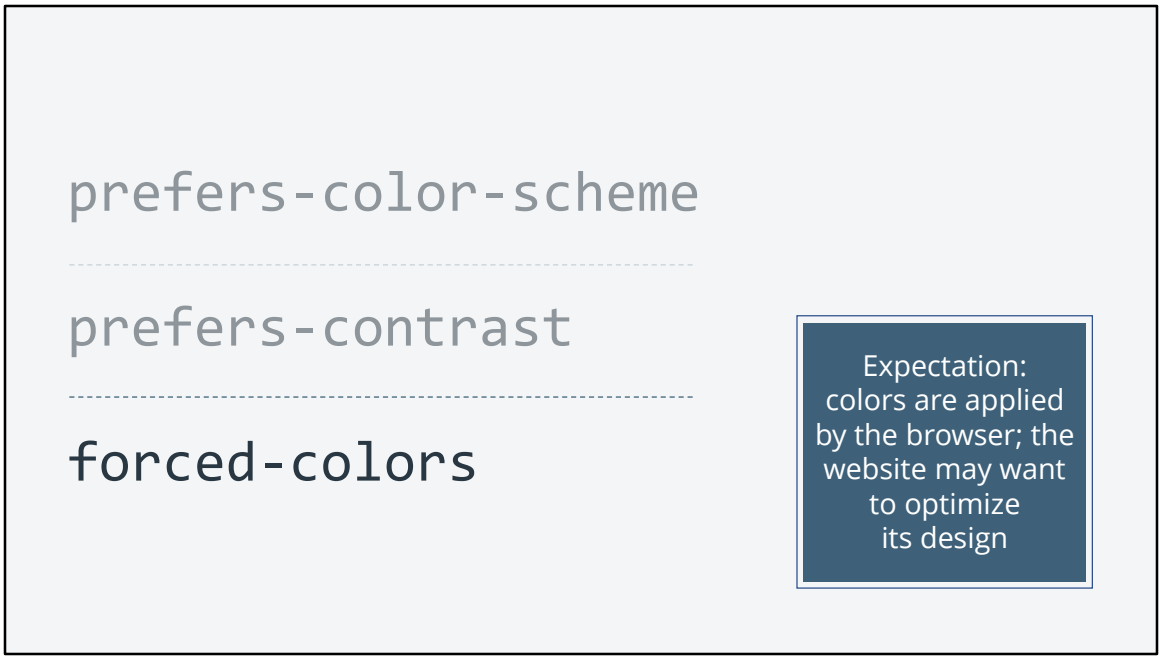
Prefers contrast is a hint to either increase or decrease contrast, which can likewise come from user settings.



Forced colors is a hint that content is rendered by the browser's rendering engine in the user's chosen colors. For example, when High Contrast mode is active.



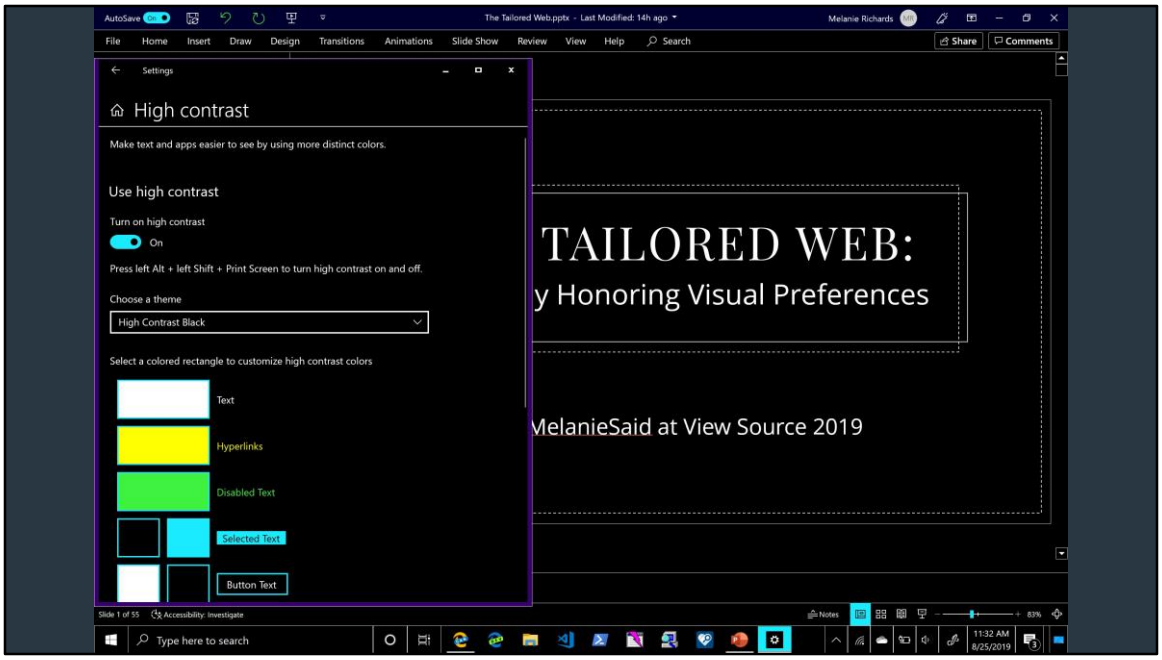
There's a couple different expectations on the web developer. For prefers-color-scheme and prefers-contrast, the expectation is that the website will fully manage its own colors and background-foreground contrast.



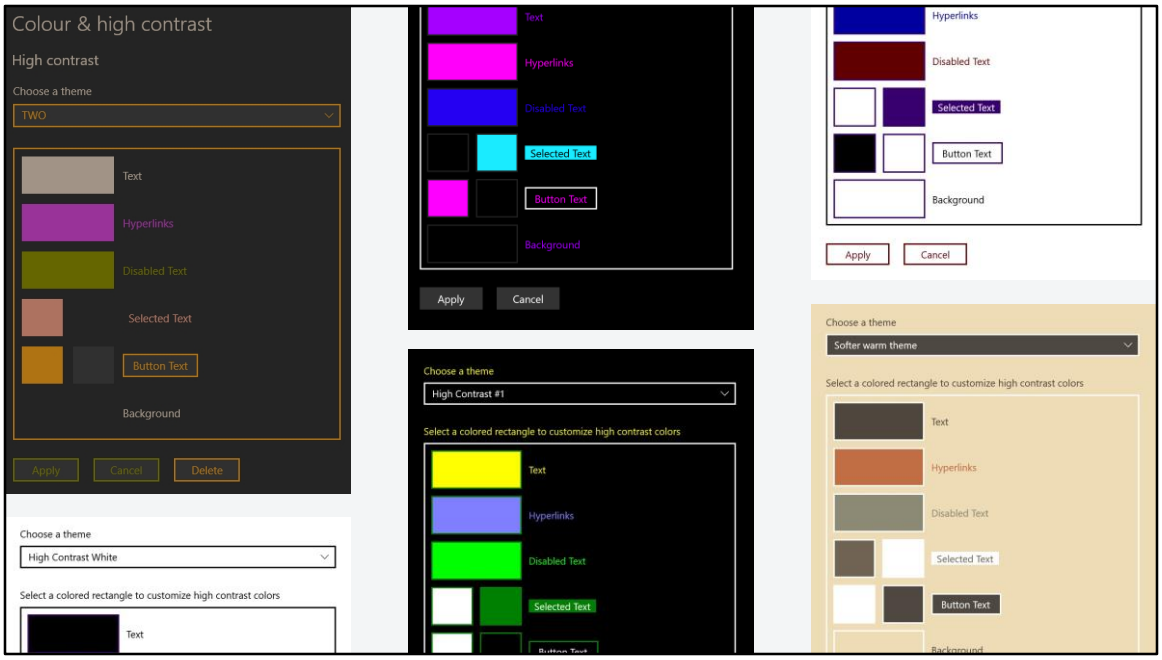
In forced color modes, these colors have already been applied, and the website may want to optimize its design.

We don’t have much time together today, so I’ll focus on how you can optimize for forced color modes. If you’d like to learn more about prefers-color-scheme and prefers-contrast, I have a link out to resources at the end of this presentation.

Alright, let’s dive into forced colors.



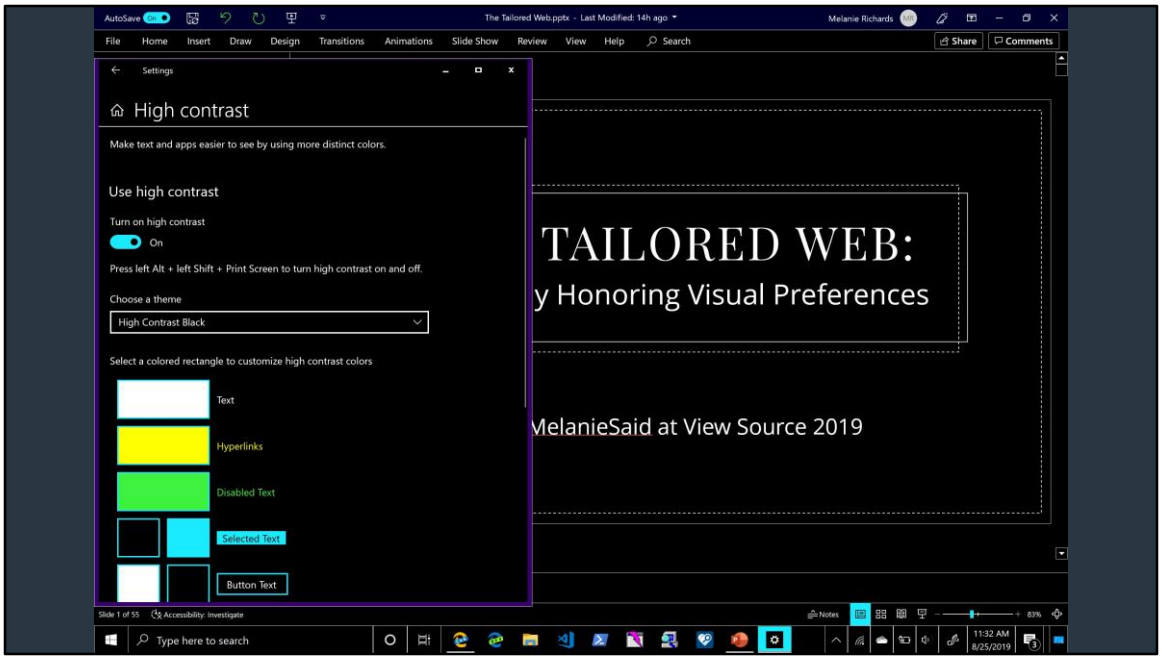
Standards around forced colors are heavily influenced by the Windows High Contrast feature shown earlier, as well as prior art used to style for this mode in IE and Microsoft Edge.



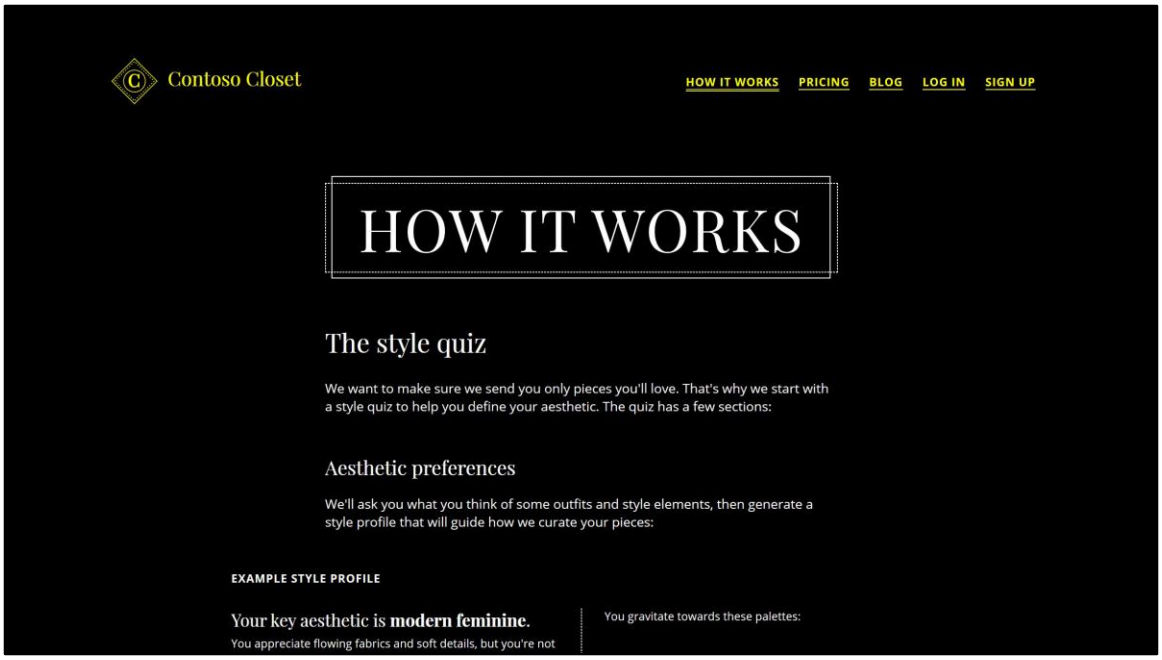
It should be noted that Windows High Contrast is somewhat of a misnomer. While the in-box themes are certainly the most popular and guarantee high contrast, users can adjust these colors to anything they want. This can be helpful if someone needs colors that are a bit more differentiated from each other, or if they actually need UI to render with SOFTER contrast. No matter their reason for using High Contrast, this is far and away one of the most popular Ease of Access features on Windows—last time we checked, used on 4% of active devices. That represents millions of people!

How browsers apply forced colors to web content

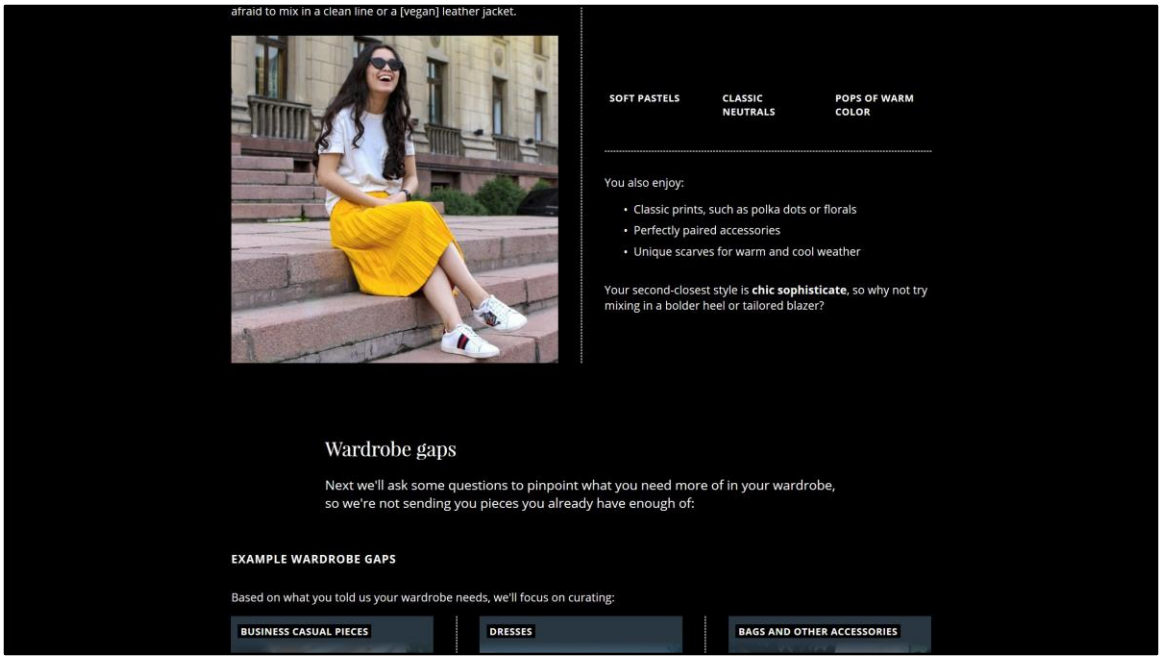
The OS setting provides the information needed to implement forced color modes, but it's up to individual apps to apply these colors in a semantic fashion to their UI. This means that the browser rendering engine is directly responsible for applying the user's colors to web content. It also means that the browser can provide its own forced color modes separately from the OS; for example, Firefox currently offers some in-browser forced color settings. Let's take a look at our page in a forced color mode.



We'll set the machine to a High Contrast Black theme…



And without doing any work on the web development end, you can see that our page automatically picked up the colors. The rendering engine has applied these semantically: regular text gets the text color, while links are rendered in their own color from the palette.



These colors are shifted throughout the page…



More about you

And of course, a few more logistical questions to make sure we get it right:

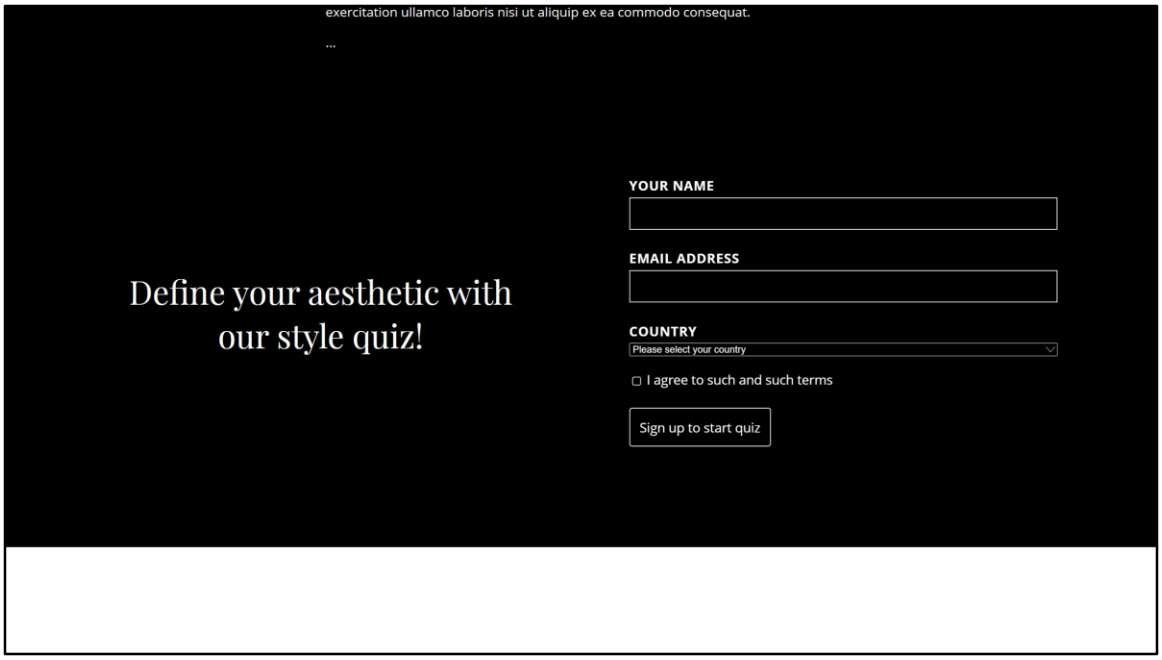
Sizing information

Brands you prefer

How much you prefer to spend on a piece

Set up your service

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud



…including native form controls.

Reverted properties

`color`
`fill`
`stroke`
`text-decoration-color`
`text-emphasis-color`
`background-color`
`border-color`
`outline-color`
`column-rule-color`
`scrollbar-color`
`-webkit-tap-highlight-color`
`box-shadow`
`text-shadow`

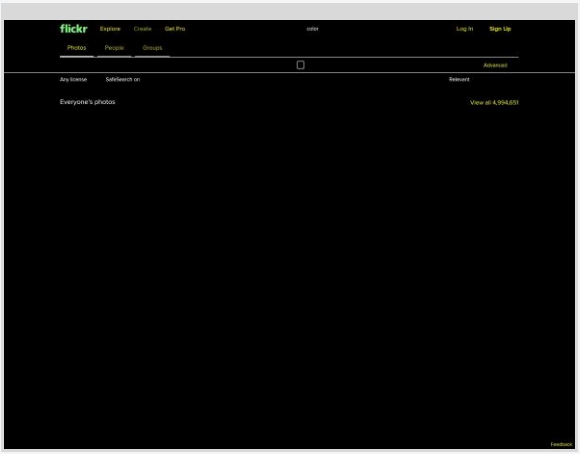
`background-image` (only on user interactive controls, excluding `buttons`)

Let's explore how this works under the hood. The rendering engine reverts a list of properties related to color and other various decorations that could interfere with legibility. Then, it supplies its own user agent stylesheet, where various elements are themed with the user's system colors. What's important to note here as web developers is that if you use a reverted property to denote state, you may need to make some adjustments when your content is rendered in a forced color mode.

Another interesting point to call out is that background-image is only reverted in user interactive controls, excluding buttons.

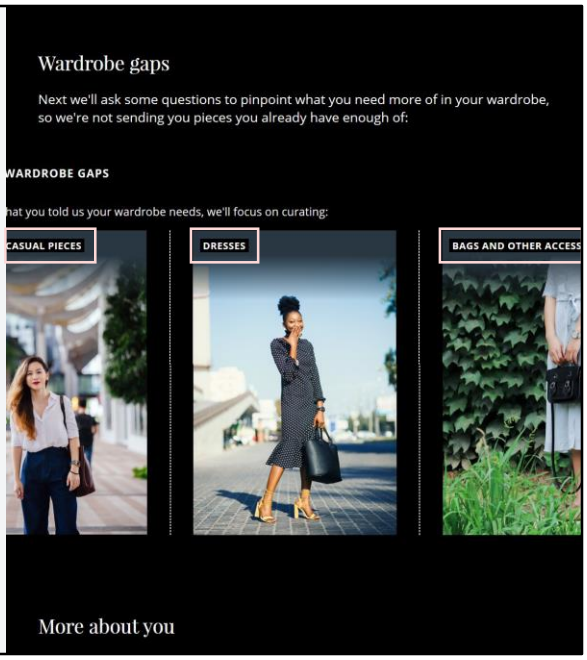
Why not globally revert `background-image`?

Many developers use this property to display “content” images



Why not globally revert background-image? In historical implementations that remove background-images, this causes issues when CSS is used to display content images. Check out this feed of photos, for example. Let the record show, there are no images visible in this screenshot.

Text backplates guarantee legibility

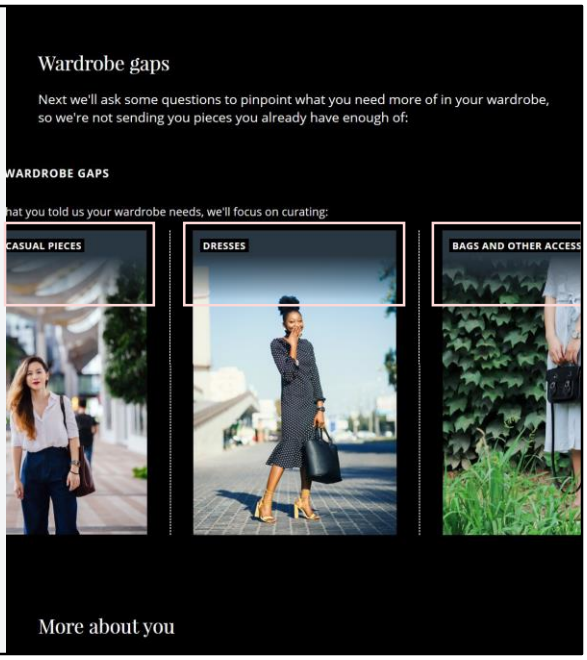


So, how do we show background images and guarantee legibility of text layered on top of these images? The solution we came up with in EdgeHTML, and which is now part of the standard, is to render text backplates behind runs of text in forced color modes. No matter what the text is layered on top of, users will be able to read it in their preferred color scheme.

Future standards may unlock the ability to tweak backplate styling; for example, adding the ability to change the border radius.

Color schemes in forced color modes

`prefers-color-scheme` evaluates appropriately based on Canvas color (page background)



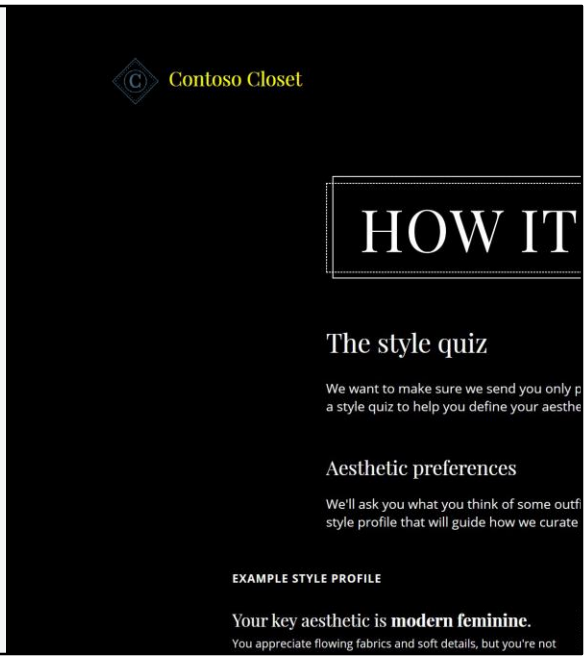
A new detail in standard implementations is that prefers-color-scheme also evaluates properly in forced color modes. The rendering engine uses the Canvas color, or page background color, to determine if the color scheme is light or dark. It then evaluates prefers-color-scheme media queries accordingly. Because our dark mode styles that I mentioned earlier shift the color of these gradients to dark blue, they appear dark blue in this dark forced color scheme as well.

Web devs can apply content-specific knowledge to tailor the forced colors experience.

Rendering engines can get most of the way there by applying reasonable defaults semantically to web content. But the people who work on individual sites know the needs of their content best. Web devs can apply content-specific knowledge to tailor the forced colors experience.

SVGs are not modified…mostly

Only SVG `<text>` and `<foreignObject>` have forced colors applied for legibility

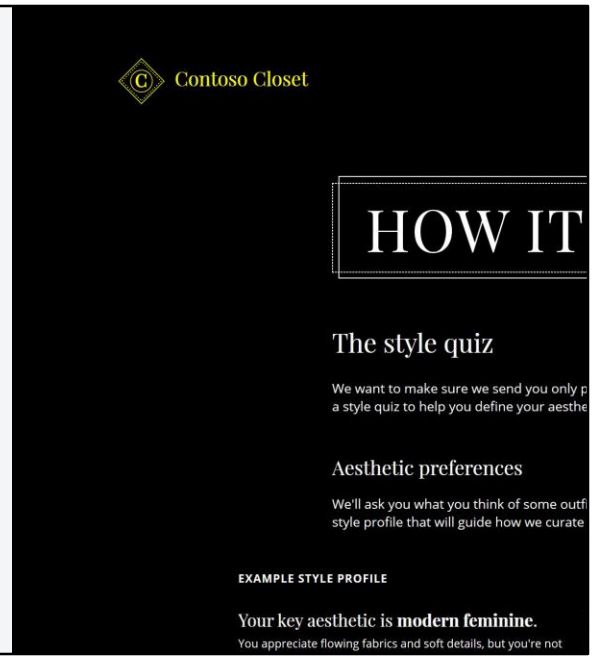


For example, our logo doesn't look too great in forced color modes. SVGs can contain complex content, and so as rendering engines we can't really guess at how these should be rendered. The only parts of SVG that have forced colors applied are <text> elements, and <foreignObject>s (which embed HTML elements within SVG).

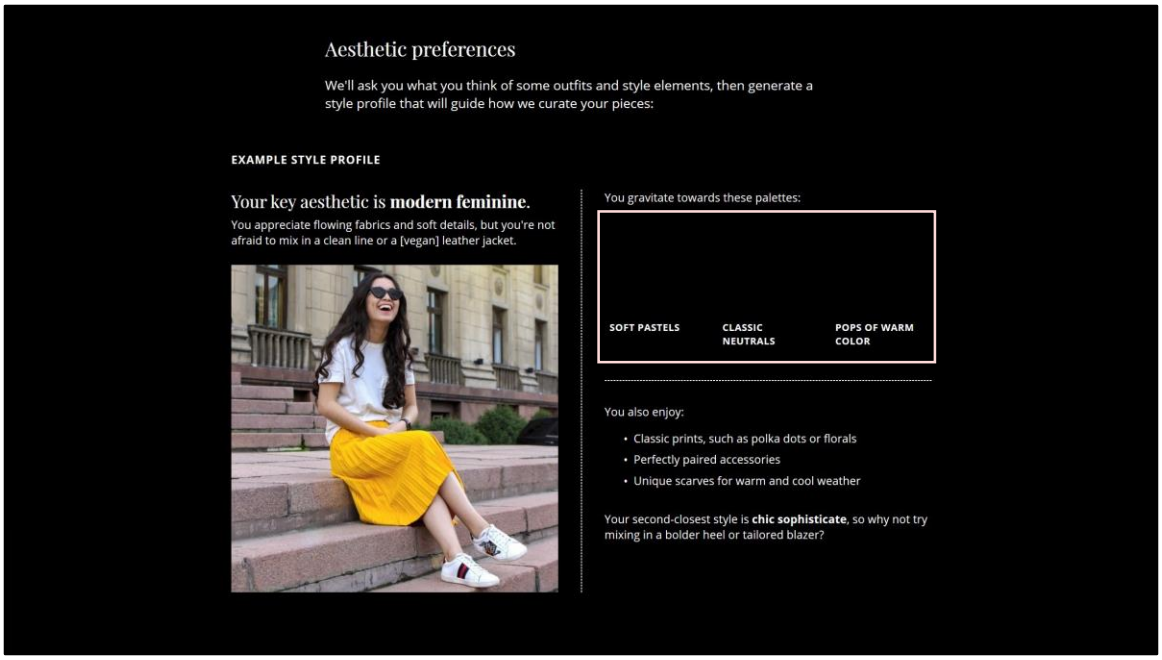
currentColor

```
.header__brand-icon * {
  stroke:
  currentColor;
}

.header__c-glyph {
  fill: currentColor;
}
```



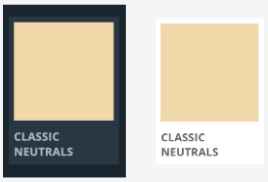
So for single-color SVG icons like our logo, we'll set the strokes and fills to currentColor. This will inherit the correct user color from the parent link! We'll do this outside of any media queries, because we always want the logo to inherit this color, whether it's in light mode, dark mode, or a forced color mode.



You may have noticed that when we flipped into a forced color mode, we lost some detail in the palette of this style profile. That's because we used background-color to supply the color swatches. We could switch to using images, or we could manage this component ourselves.

Component requirements

1. Retain the color swatch
2. Prevent swatch rendering on top of any given background color (muddy?)
3. Preserve the user's color scheme for text legibility



Let's review our requirements in forced color modes for this component.

We want to:

1. Retain the color swatch.
2. Prevent the swatch from rendering on top of any old background color the user may have chosen, in case it looks a little muddy. So we've decided we always want to render that on top of either black or white.
3. And preserve the user's color scheme for text legibility.

Prior art: media query syntax

```
@media (-ms-high-contrast: active) {}
```

Values:

`none` (IE and early MS Edge) | `active` | `black-on-white` | `white-on-black`

Let’s start with our media query. In browser versions built before these new standards, we’d query for –ms-high-contrast: active

Standard media query syntax

```
@media (-ms-high-contrast: active),
       (forced-colors: active) {}
```

Values:

```
none | active
```

This has been renamed in web standards for forced-colors, and the value we're querying for is "active". The CSS Working Group landed on this name to future-proof the media query, so that web developers can efficiently style for any OS or browser feature which applies forced colors to web content.

Prior art: opt out of forced colors

```
@media (-ms-high-contrast: active) {
  .prefs__palette {
    -ms-high-contrast-adjust: none;
  }
  …
```

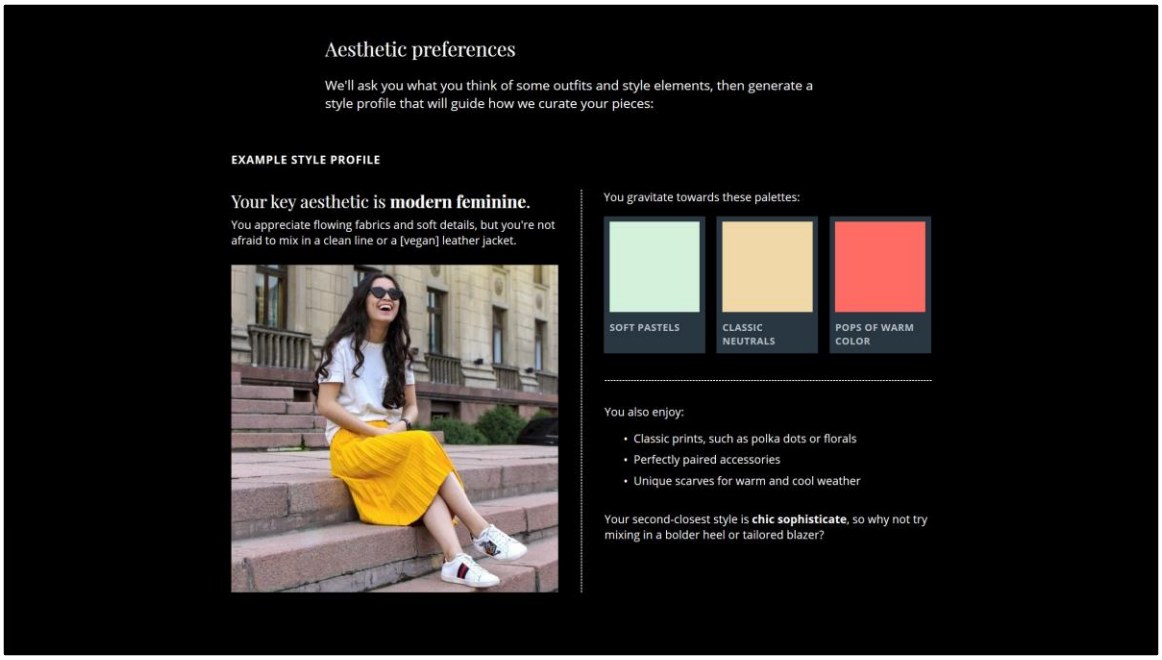
After setting up our media query, we'll want to unset the forced colors from the element. This strategy comes from the –ms-high-contrast-adjust CSS property.

Opt out of forced colors

```
@media (-ms-high-contrast: active),
(forced-colors: active) {

  .prefs__palette {
    -ms-high-contrast-adjust: none;
    forced-color-adjust: none;
  }
```

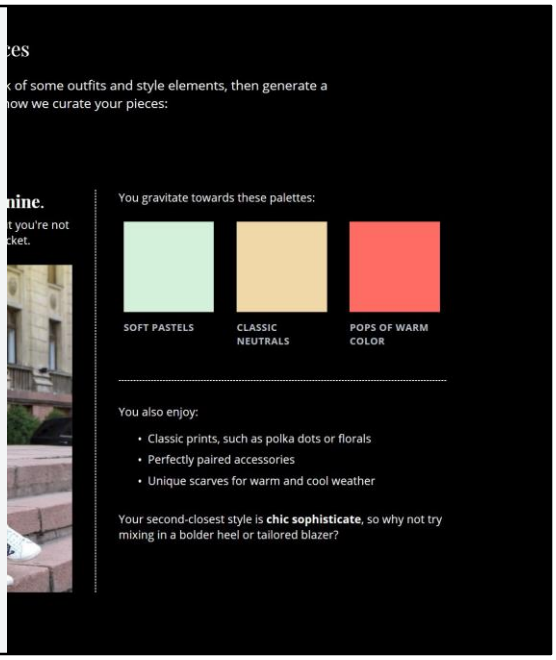
In the new standards, forced-color-adjust set to “none” will remove the colors from the component and its children, including all text backplates.



Our palette now looks as it did in dark mode.

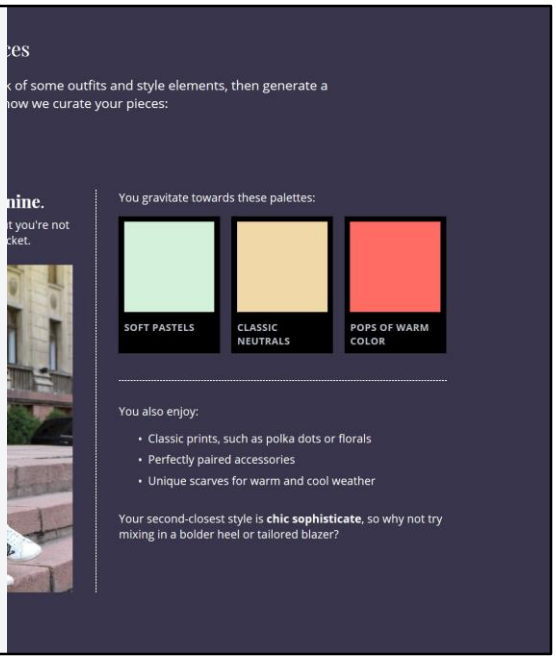
Be careful with static colors

```
.prefs__palette {
  forced-color-adjust:
  none;
  background-color: #000;
}
```



Next, we want to render the background of this component in black. We're doing this because we don't want the swatch to render on a random color, but we should NOT try and use static colors to match a user's theme. This reason why is less obvious when we have a black theme.

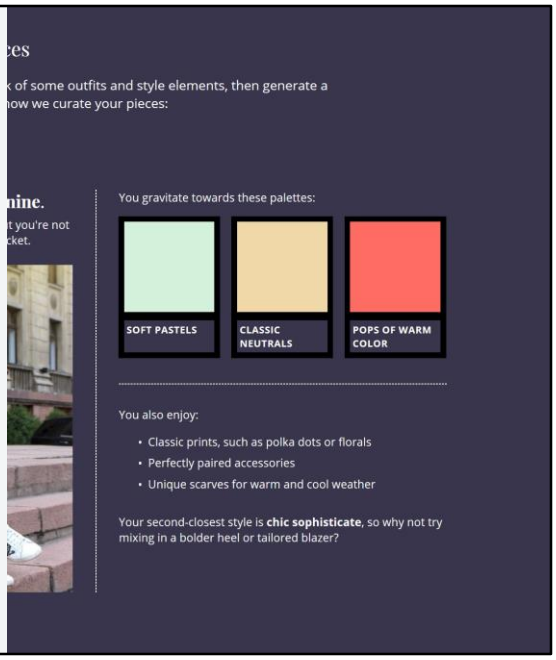
Static color values are unaware of the user scheme



But flip this into, say, a purple user color scheme, and you can see that our static color isn't aware of the user scheme. We want to make sure that the text is in the user's colors, so in order to do that…

Use appropriate system colors

```
.c-label {
  padding: .25em;
  background-color:
  Canvas;
  color: Text;
}
```



…we're going to use system colors to guarantee legibility. We'll create our own text backplate by setting the background-color to Canvas (page background) and the color to Text.

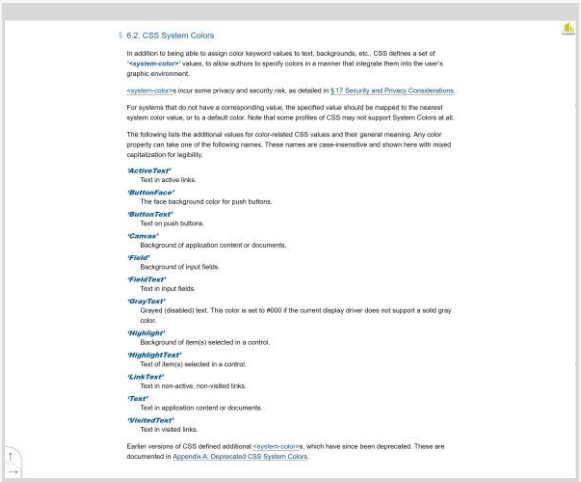
CSS system colors relevant to Windows High Contrast

High Contrast Theme Color	CSS System Color	Prior System Colors
Text	`Text`	`WindowText`
Hyperlinks	`LinkText`	`-ms-hotlight` (IE and MS Edge only)
Disabled Text	`GrayText`	
Selected Text	`HighlightText` (foreground), `Highlight` (background)	
Button Text	`ButtonText` (foreground), `ButtonFace` (background)	
Background	`Canvas`	`Window`

There are a few CSS system colors that are relevant when Windows High Contrast is active. System colors help keep styling in forced color modes manageable, so that your colors “just work” in any user color scheme. The updated set of colors introduces a LinkText color, replacing the –ms-hotlight value.

Full list of system colors on CSS Color Editor's Draft

aka.ms/
system-colors-draft



§ 6.2. CSS System Colors

In addition to being able to assign color keyword values to text, backgrounds, etc., CSS defines a set of '<system-color>' values, to allow authors to specify colors in a manner that integrate them into the user's graphic environment.

<system-color>s incur some privacy and security risk, as detailed in § 17 Security and Privacy Considerations.

For systems that do not have a corresponding value, the specified value should be mapped to the nearest system color value, or to a default color. Note that some profiles of CSS may not support System Colors at all.

The following lists the additional values for color-related CSS values and their general meaning. Any color property can take one of the following names. These names are case-insensitive and shown here with mixed capitalization for legibility.

'ActiveText'
Text in active links.

'ButtonFace'
The face background color for push buttons.

'ButtonText'
Text on push buttons.

'Canvas'
Background of application content or documents.

'Field'
Background of input fields.

'FieldText'
Text in input fields.

'GrayText'
Grayed (disabled) text. This color is set to #000 if the current display driver does not support a solid gray color.

'Highlight'
Background of item(s) selected in a control.

'HighlightText'
Text of item(s) selected in a control.

'LinkText'
Text in non-active, non-visited links.

'Text'
Text in application content or documents.

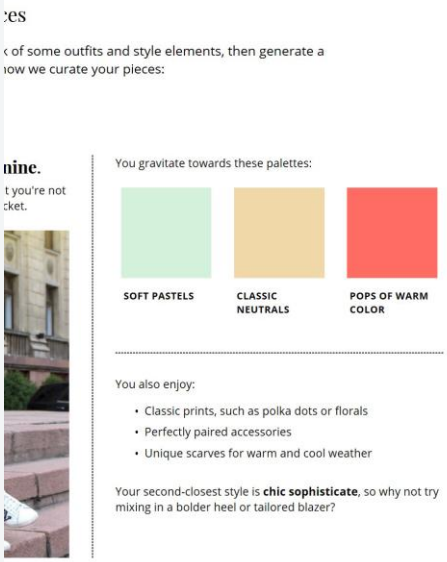
'VisitedText'
Text in visited links.

Earlier versions of CSS defined additional <system-color>s, which have since been deprecated. These are documented in Appendix A: Deprecated CSS System Colors.

You can find the full list of standardized system colors on the CSS Color draft specification. Not every operating system will be able to support every standardized system color, so it's important to test whenever you use these values.

Optimize for dark / light forced colors

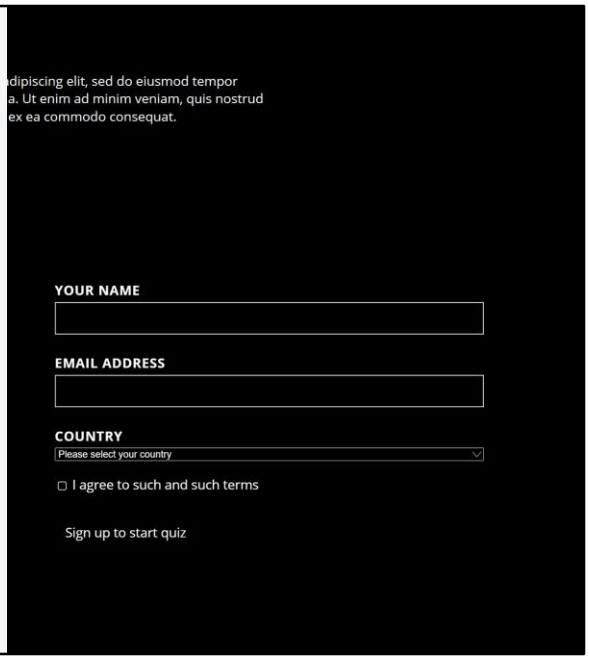
```
@media (forced-colors: active)
and (prefers-color-scheme: light){
  .prefs__palette {
    background: #fff;
  }
}
```



Going back to our component, since we want the static color to change to white in light themes, we can set those styles from within a query for (forced-colors: active) and (prefers-color-scheme: light)

Instead of removing styles…

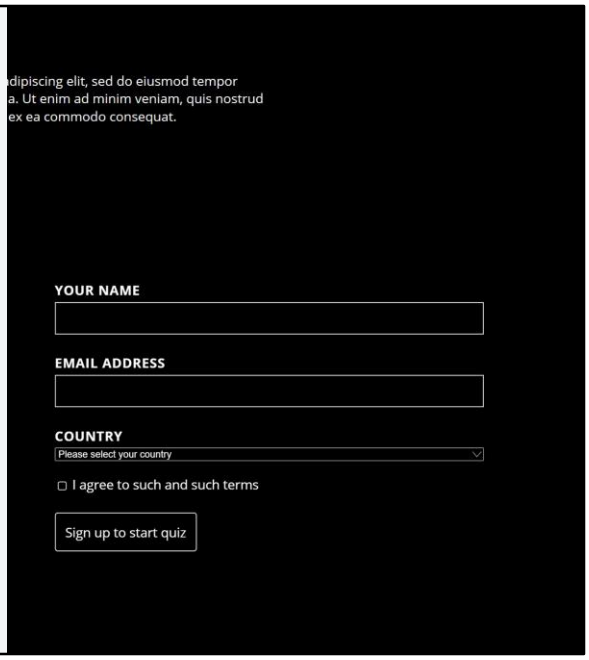
```
button {
  border: 0;
}
```



In other considerations, we should be careful about using shorthands to remove styles. For example, in this forced color scheme, the button and page background both happen to be black. So setting border: 0 means that the user can't see the bounds of this button when in this forced color mode.

…consider transparency

```
button {
  border: 1px solid
  transparent;
}
```



Change this to border: 1px solid transparent, and the border is rendered in forced color modes while invisible in the base theme.

Browser support for forced colors (desktop)

Vendor-prefixed styling

New standards

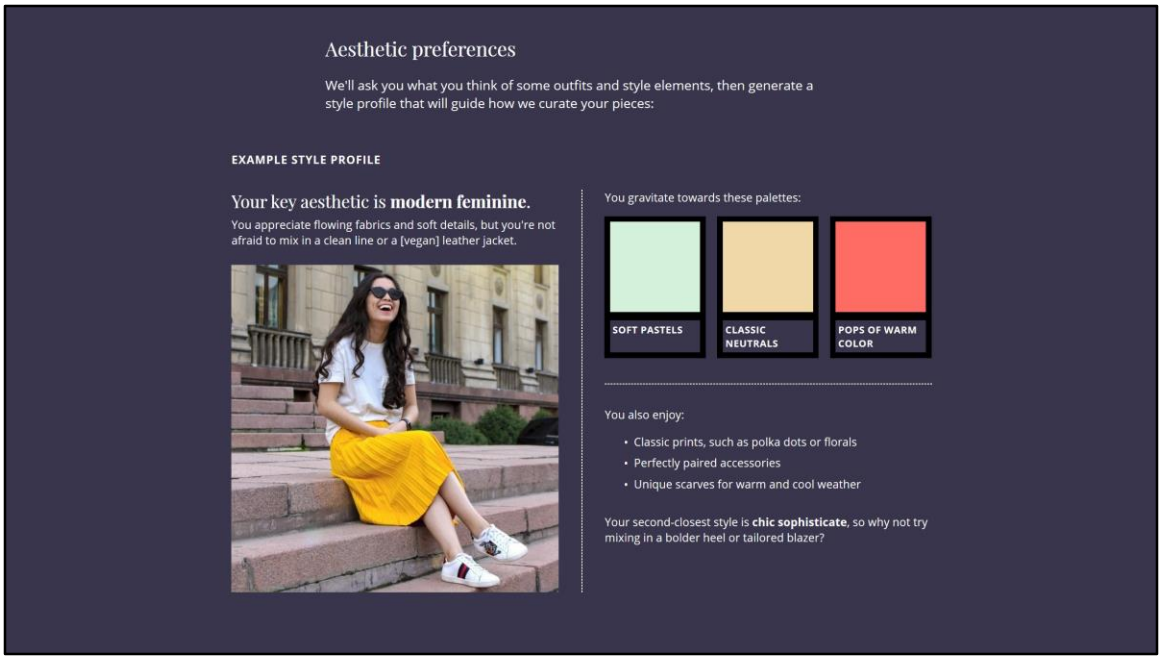
In development behind a flag in the Chromium project

These standards are pretty new, so stable implementations aren't quite ready. But forced colors are in development in the Chromium project, and should be ready to play with in preview versions of Chromium-based browsers soon.

Testing without a Windows machine?

Virtual Machines:
aka.ms/web-vms

If you don’t have a Windows machine, you can use a free Virtual Machine from aka.ms/web-vms to try out forced colors on Windows. In the future, cross-platform browsers could also explore providing dev tools to “spoof” forced color modes, so that you don’t need a particular OS or to push your whole machine into forced colors. Show of hands, who would be interested in such a feature?



I hope you do get the chance to try out your content in forced color modes. Because while browser rendering engines can get pretty far with reasonable defaults, you are the expert on your content and can work on top of the platform to deliver a truly well-tailored experience.

With these new standards in hand, let's design a more inclusive and accessible web together. Thank you so much, and you can find these slides and other resources at aka.ms/tpac-hc