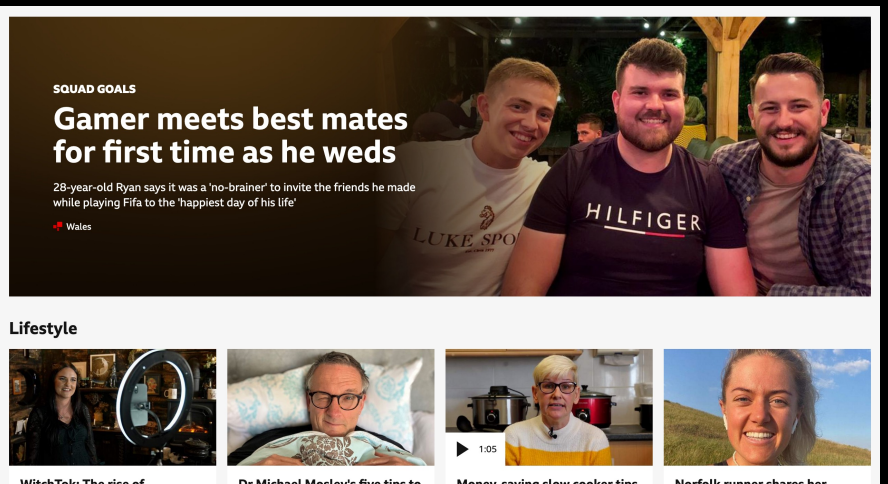
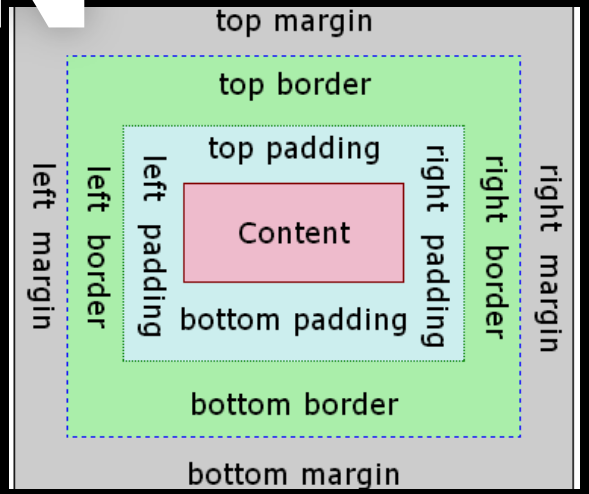
INTRINSIC WEB DESIGN

SCC.306 Guest Lecture

Lancaster University





BBC

Josh Tumath (he/him)

Senior Software Engineer

Design System

BBC

WE WANT YOU

https://www.bbc.co.uk/careers/

BBC

Coming up…

1. How does CSS layout work?
2. History of layout on the Web
3. Building pages with Intrinsic Web Design
4. Container queries and other new CSS features

How does CSS layout work?

BBC

@joshtumath

Everything is a box

Block box

Inline box

BBC

@joshtumath

Lancaster University to build solar farm near M6

6 March

Lancaster University will construct the site near the Forest of Bowland

A university is due to build a solar farm near a major motorway after it has received planning permission.

Lancaster University will construct the site on a 50-acre area it owns by the M6, near the Forest of Bowland.

The solar farm is expected to create power that could support about 3,000 homes, the Local Democracy Reporting Service says.

The plan received unanimous cross-party support from Lancaster City Council's planning committee.

Tests have indicated that drivers would not experience potential sunlight glare due to surrounding hedges and trees, the planning meeting heard.

Top Stories

LIVE Migrant centre overcrowding piles pressure on Braverman

No secrets in security breach email - Braverman

12 minutes ago

LIVE Explosions over Kyiv as Russia attacks Ukraine infrastructure

Features

Eyewitness: 'I was trying to do CPR, but they were both dead'

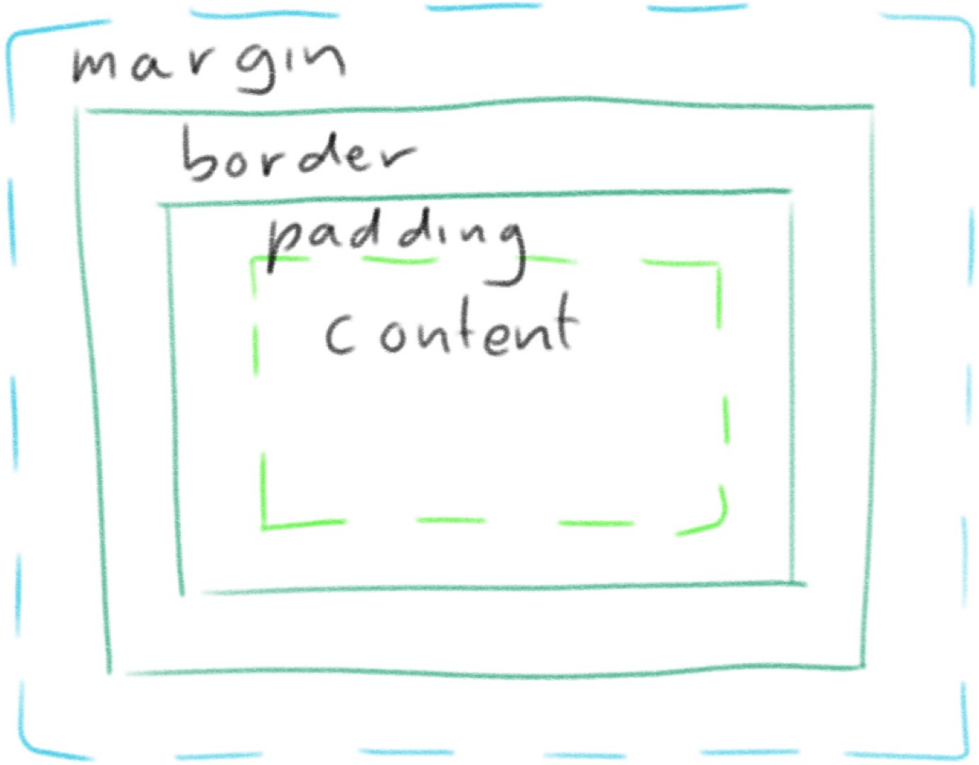
A moment in history for Brazil as Lula returns

Everything is a box

The box model

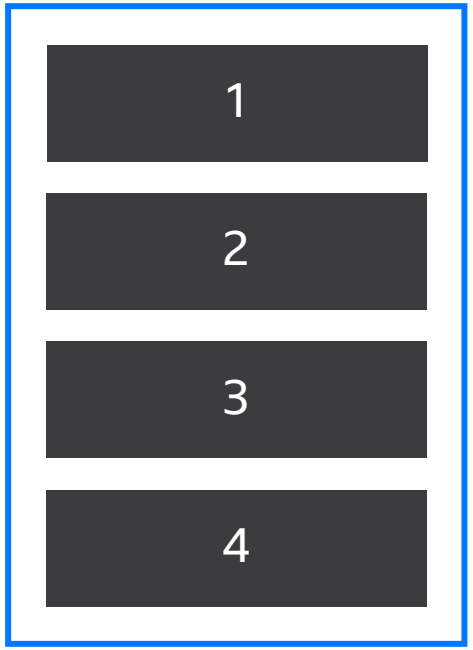
See this blog post for more info:

https://joshtumath.uk/2019/04/03/eight-css-fundamentals-no-one-teaches-us.html

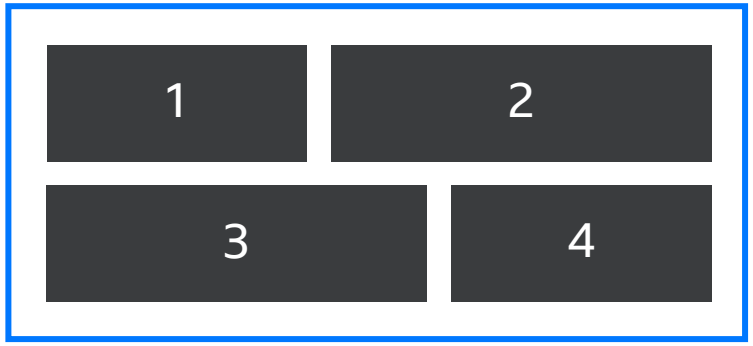


Layout methods

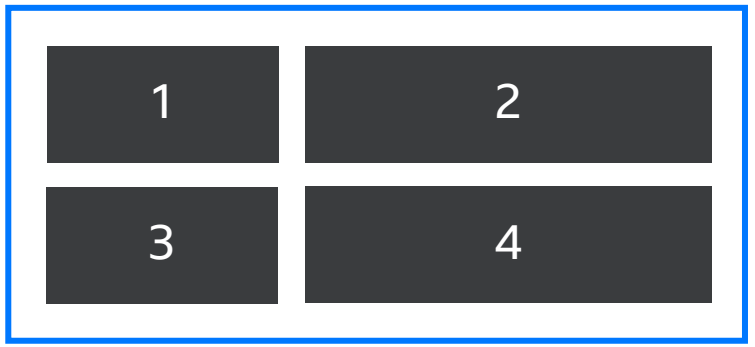
`display: block flow;`



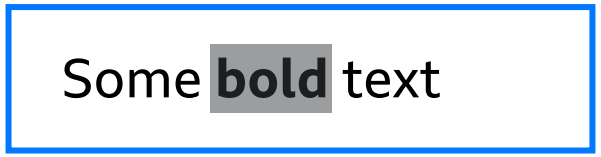
`display: block flex;`



`display: block grid;`



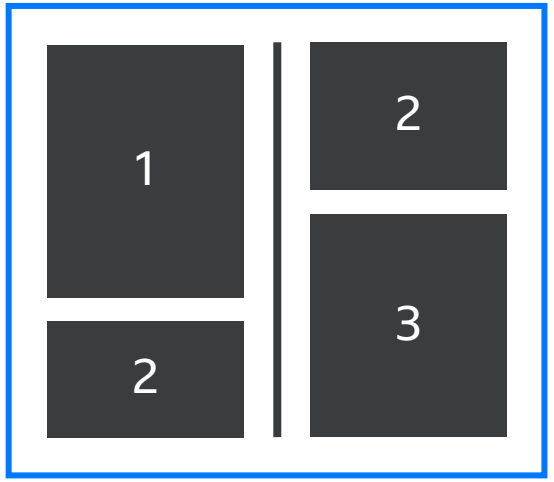
`display: inline flow;`



Other methods:

- Tables
- Ruby
- Inline blocks
- Positioning

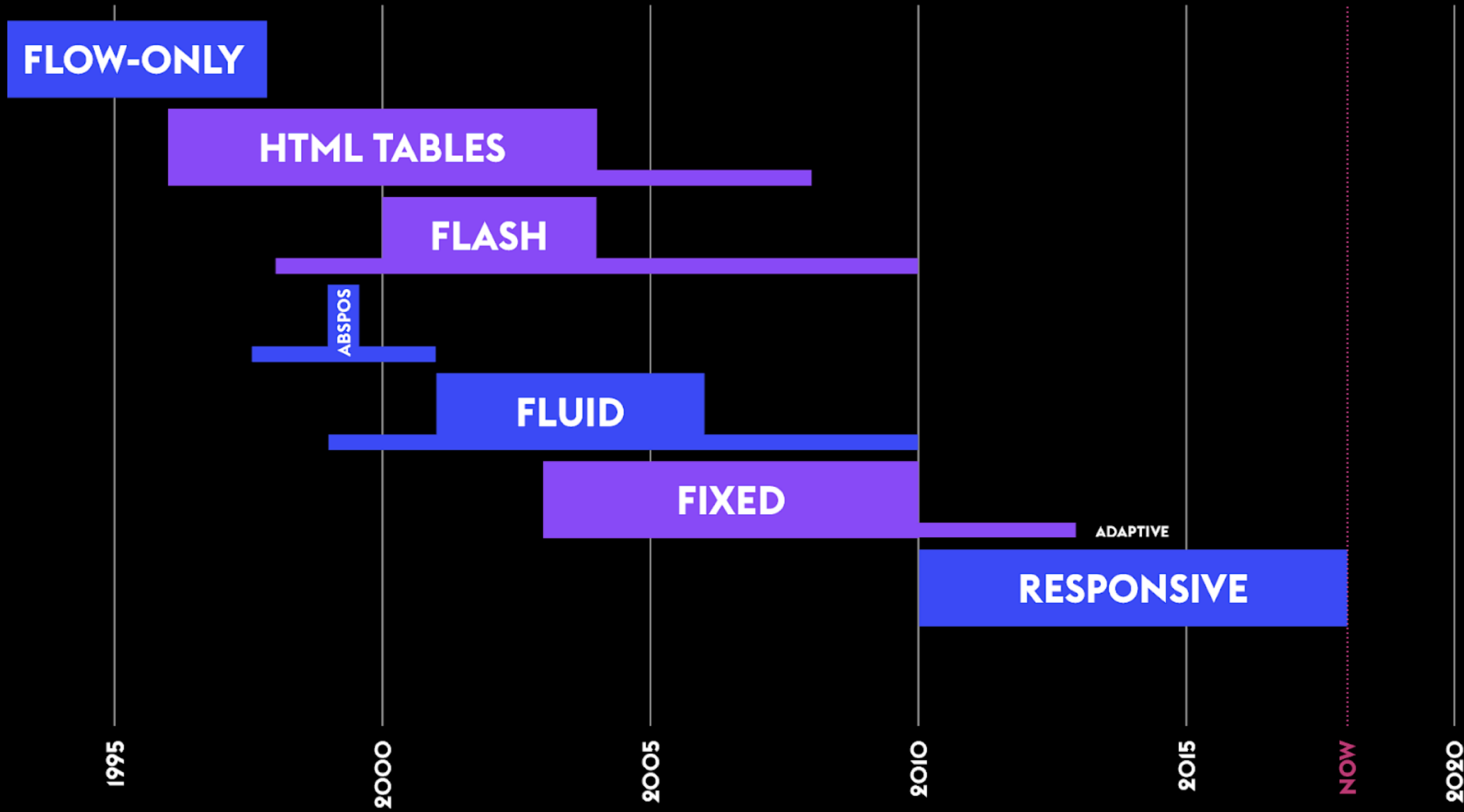
`columns: 2`



The history of layout on the Web

BBC

@joshtumath



BBC

@joshtumath

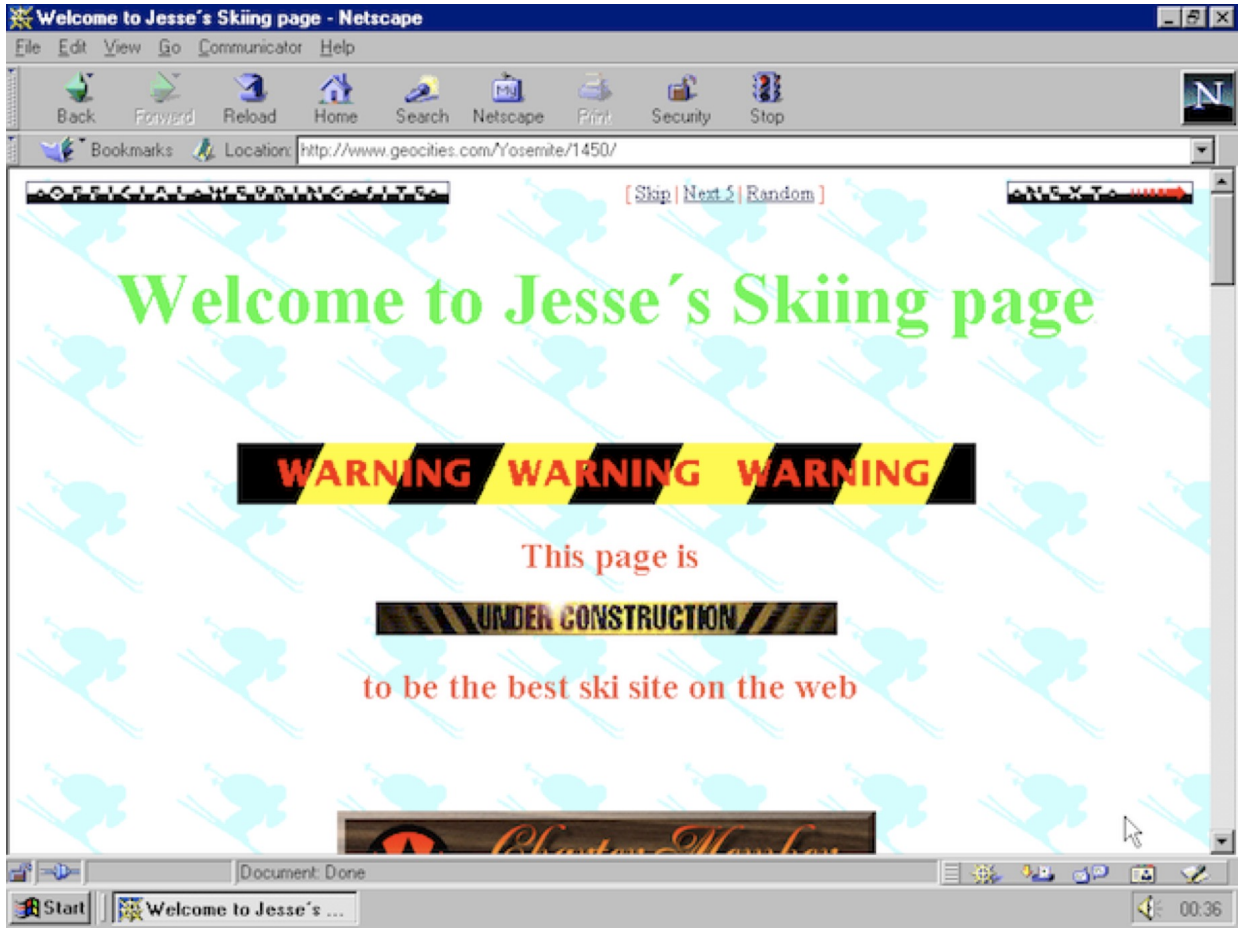
Source: Jen Simmons, ‘Everything You Know About Web Design Just Changed’

Simple flow layouts

Uses the default behaviour. Each box is laid out one box after the other.

TRIVIA

Before CSS, we'd use non-standard HTML elements and attributes like `<font>`, `<center>` and `bgcolor="blue"`.



Source: https://oneterabyteofkilobyteage.tumblr.com/

Table layouts

Abusing the <table> element, intended for showing tabular data, to lay out the website content in a grid.

DO NOT DO THIS!

- It does not work for accessibility tools
- It’s a nightmare to control and maintain

Absolute layouts

Laying out boxes using the CSS property `position: absolute`

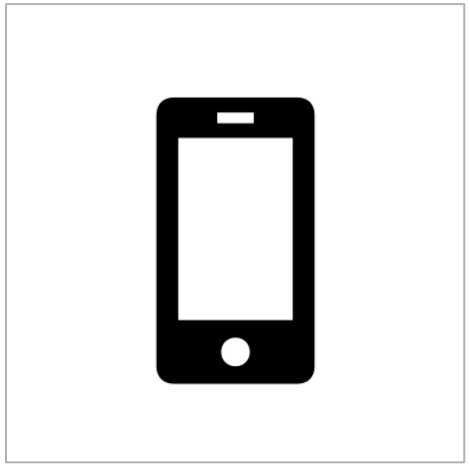
[No picture, because it was rarely done]

THE PROBLEM

- It does not work if you don’t know the exact height of the content
- Difficult to centre boxes

Separate mobile and desktop websites

Creating two websites: one for phones and one for desktops.



THE PROBLEM

- You had to make and maintain two websites!
- The Web server couldn't easily check whether the user was on a mobile.
- The mobile website might have different content to the desktop website.

Fluid and fixed layouts

Abusing the CSS property `float: left` to lay out the content side-by-side. Setting the width of boxes with pixels or percentages.

THE PROBLEM

- Fluid content looked rubbish on a widescreen.
- You can't have both fixed and fluid boxes side-by-side! The content needs to be either fluid or fixed.

BBC

@joshtumath

MAY 25, 2010

Responsive Web Design

by ETHAN MARCOTTE

Published in: CSS, Layout, User Interface Design | Discuss this article »

> *The control which designers know in the print medium, and often desire in the web medium, is simply a function of the limitation of the printed page. We should embrace the fact that the web doesn't have the same constraints, and design for this flexibility. But first, we must "accept the ebb and flow of things."*
>
> *John Allsopp, "A Dao of Web Design"*

The English architect Christopher Wren once quipped that his chosen field "aims for Eternity," and there's something appealing about that formula: Unlike the web, which often feels like aiming for next week, architecture is a discipline very much defined by its permanence. A building's foundation defines its footprint, which defines its frame, which shapes the facade. Each phase of the architectural process is more immutable, more unchanging than the last. Creative decisions quite literally shape a physical space, defining the way in which people move through its confines for decades or even centuries.

Working on the web, however, is a wholly different matter. Our work is defined by its transience, often refined or replaced within a year or two. Inconsistent window widths, screen resolutions, user preferences, and our users' installed fonts are but a few of the intangibles we negotiate when we publish our work, and over the years, we've become incredibly adept at doing so.

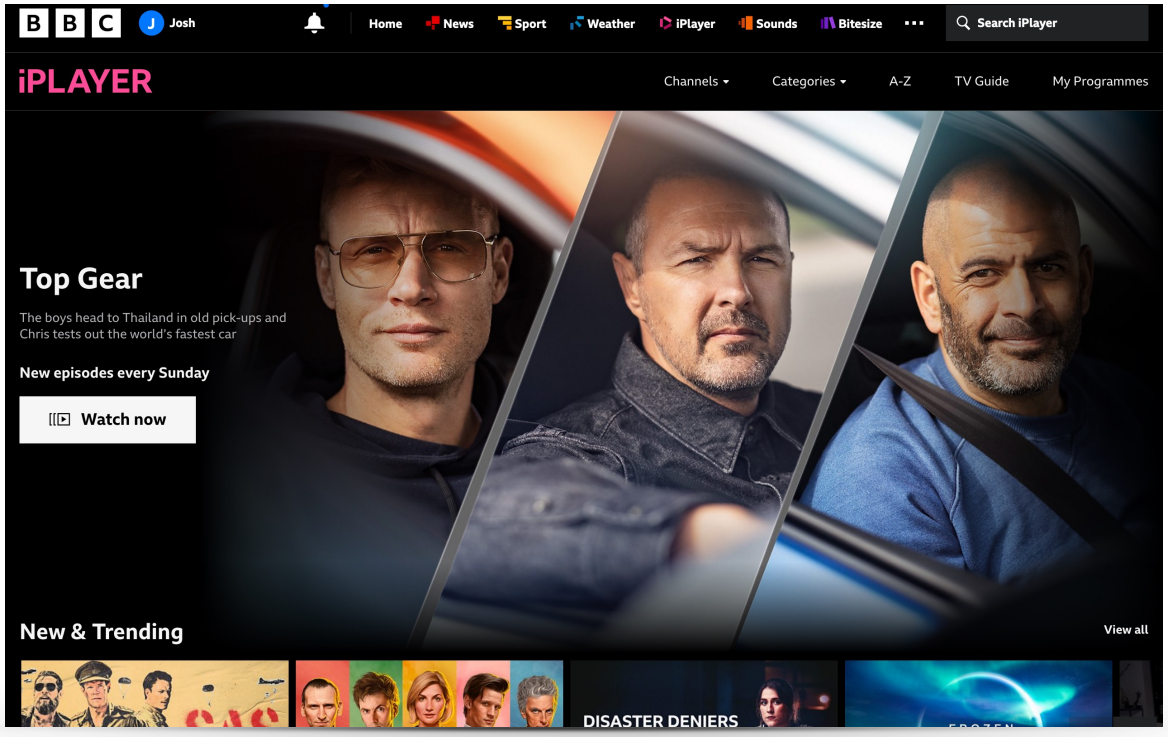
But the landscape is shifting, perhaps more quickly than we might like. Mobile browsing is expected to outpace desktop-based access within three to five years. Two of the three dominant video game consoles have web browsers (and one of them is quite excellent). We're designing for mice and keyboards, for T9 keypads, for handheld game controllers, for touch interfaces. In short, we're faced with a greater number of devices, input modes, and browsers than ever before.

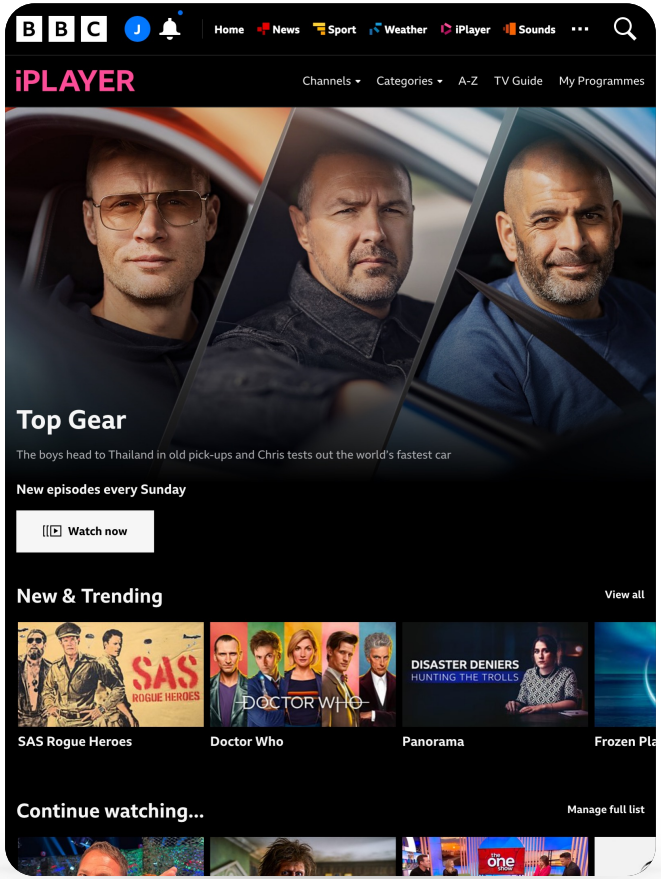
In recent years, I've been meeting with more companies that request "an iPhone website" as part of their project. It's an interesting phrase: At face value, of course, it speaks to mobile WebKit's quality as a browser, as well as a powerful business case for thinking beyond the desktop. But as designers, I think we often take comfort in such explicit requirements, as they allow us to compartmentalize the problems before us. We can quarantine the mobile experience on separate subdomains, spaces distinct and separate from "the non-iPhone website." But what's next? An iPad website? An N90 website? Can we really continue to commit to supporting each new user agent with its own bespoke experience? At some point, this starts to feel like a zero sum game. But how can we—and our designs—adapt?

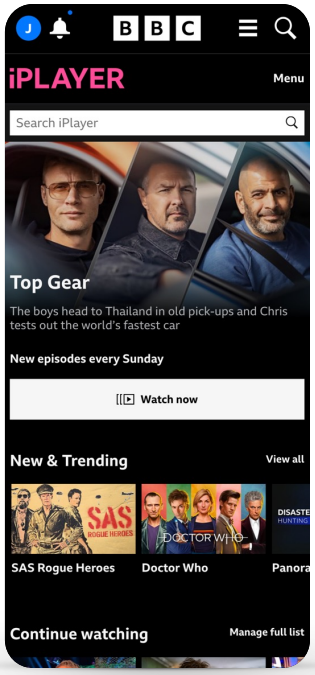
Snapshot

Learn how to use fluid grids, flexible images, and media queries to create elegant user experiences with responsive web design.

Responsive Web Design







Using CSS media queries to adapt the layout of the same content to different viewport (i.e. screen) sizes.

Responsive Web Design

We can set a Media Query to show more columns in a grid at different breakpoints.

ℹ️ *Note: CSS Grids are a very new. Back in the day, you would have used CSS Floats to replicate a grid layout.*

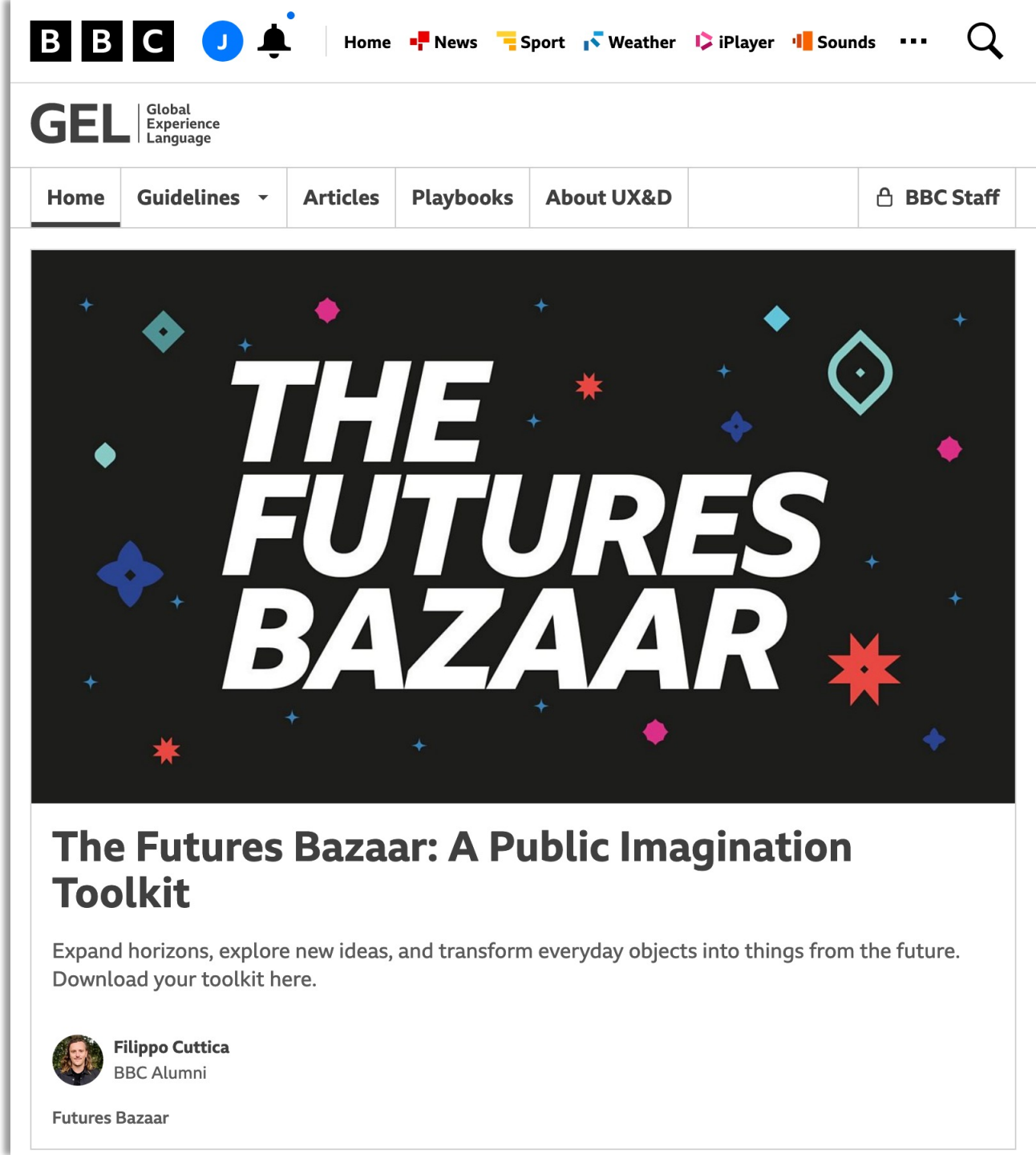
```
.grid {
  display: grid;
  gap: 1rem;
}

@media (min-width: 500px) {
  .grid {
    grid-template-columns: 1fr 1fr;
  }
}

@media (min-width: 700px) {
  .grid {
    grid-template-columns: 1fr 1fr 1fr;
  }
}

@media (min-width: 900px) {
  .grid {
    grid-template-columns: 1fr 1fr 1fr 1fr;
  }
}
```

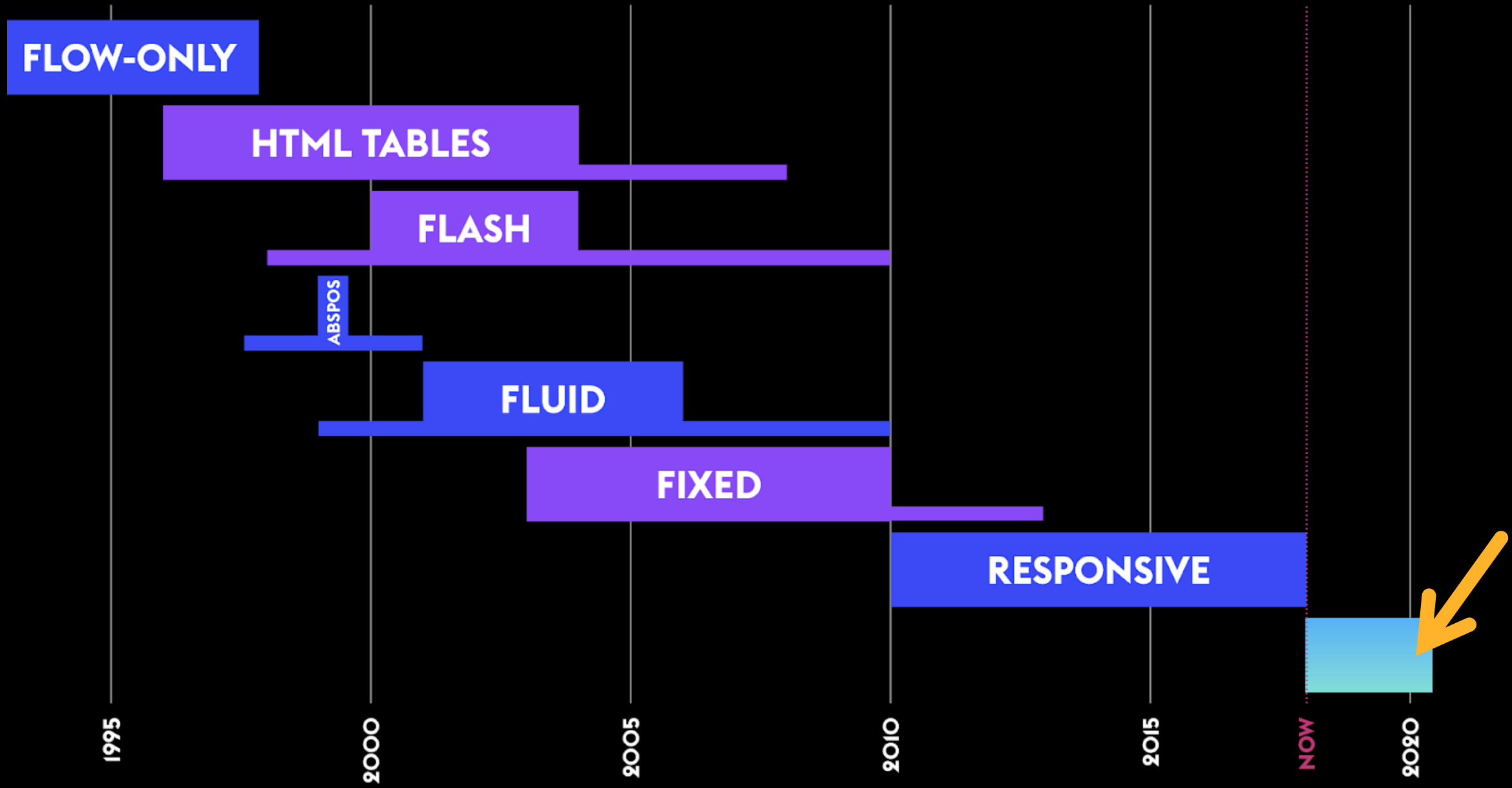
Demo: Responsive BBC GEL homepage

https://www.bbc.co.uk/gel

BBC

@joshtumath





BBC

@joshtumath

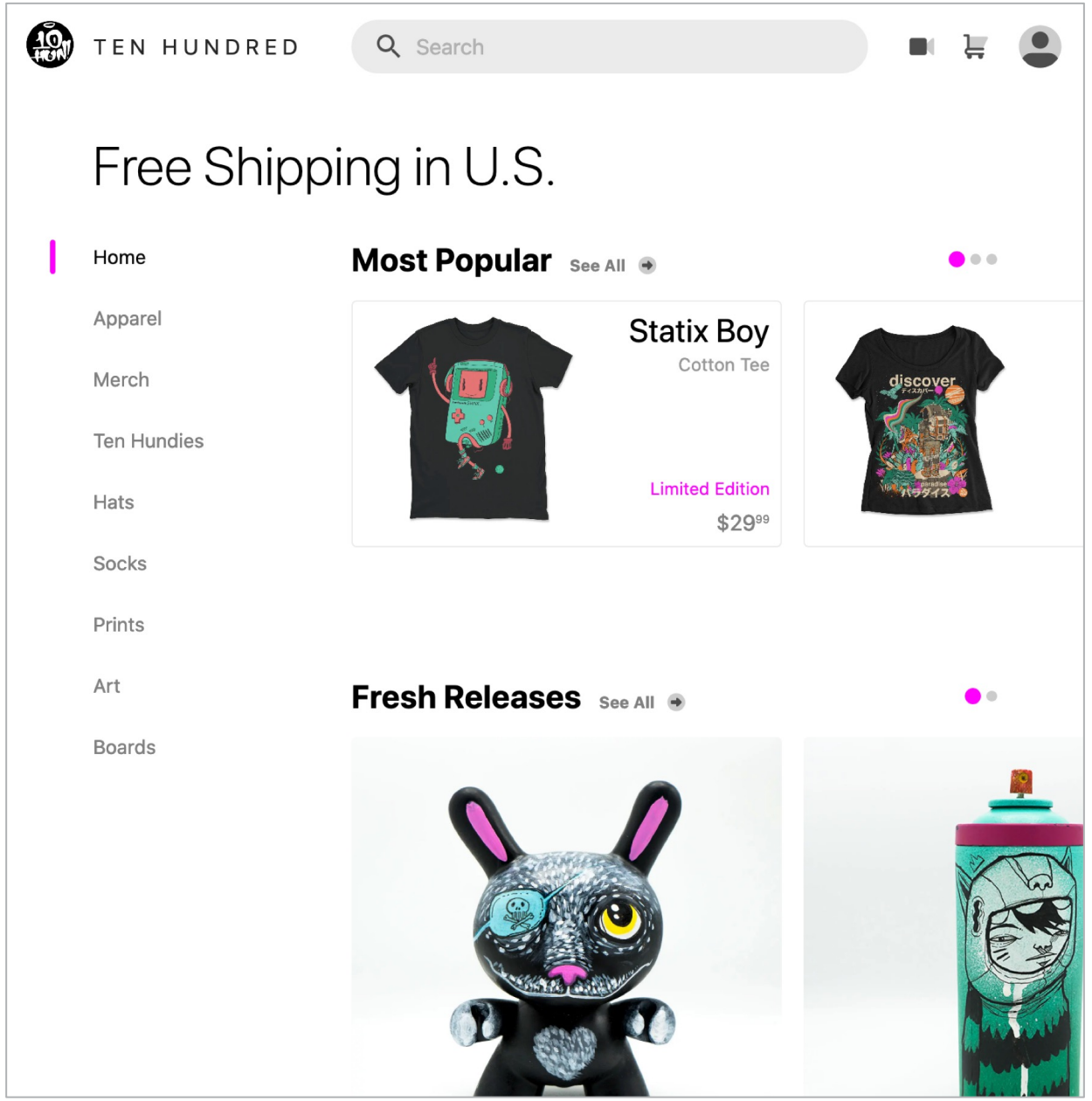
Source: Jen Simmons, ‘Everything You Know About Web Design Just Changed’

Intrinsic Web Design

Using the newest CSS features to let the content itself dictate its layout based on constraints.

A NEW ERA

- You don’t need media queries
- You don’t need to worry about where you can put your content
- Older browsers might need a fallback



Source: https://intrinsic-css.netlify.app/

Building pages with Intrinsic Web Design

BBC

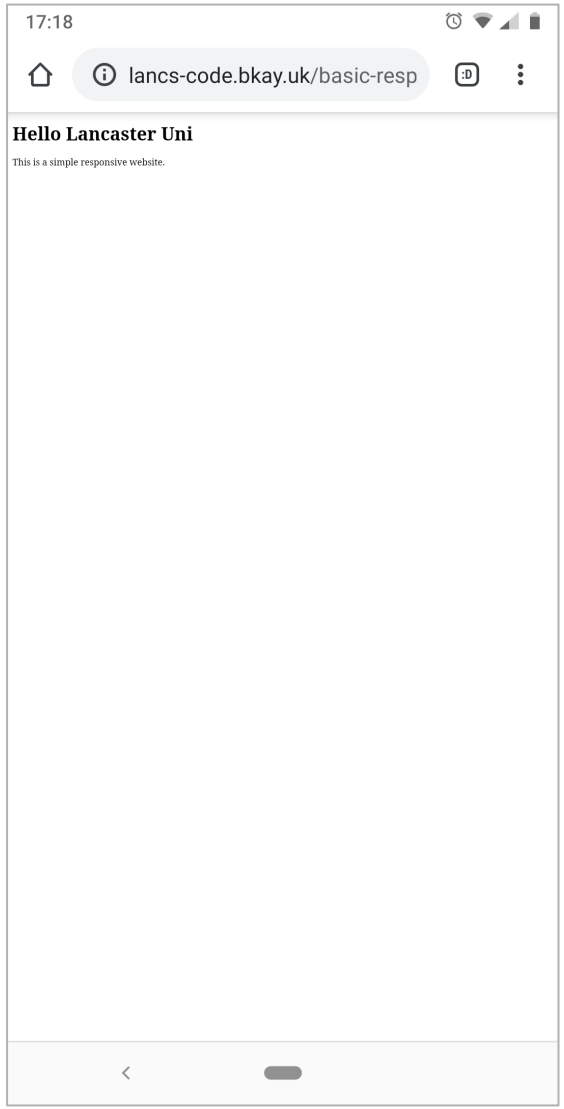
@joshtumath

How do we start?

We build our websites mobile first with progressive enhancement

Just one problem…

By default, mobile Web browsers think your Web page has been designed for desktop browsers.



The `meta` viewport tag to the rescue

Use this non-standard tag that Apple created to tell mobile Web browsers your website supports smaller viewports.

See MDN for more info:

https://developer.mozilla.org/en-US/docs/Web/HTML/Viewport_meta_tag

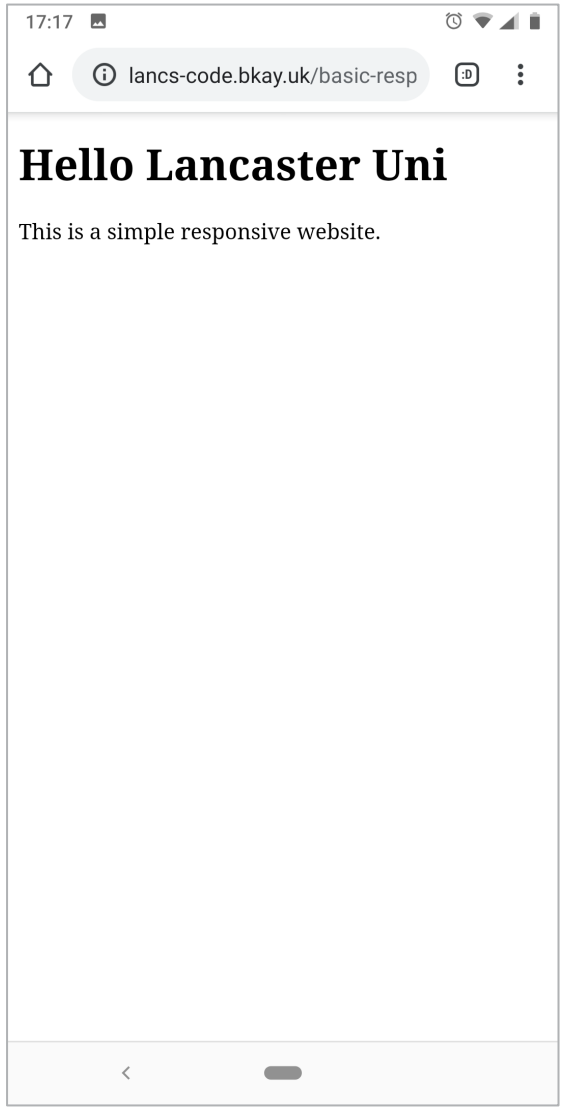
```
<meta
  name="viewport”
  content="width=device-width, initial-scale=1”>
```

The `meta` viewport tag to the rescue

Use this non-standard tag that Apple created to tell mobile Web browsers your website supports smaller viewports.

See MDN for more info:

https://developer.mozilla.org/en-US/docs/Web/HTML/Viewport_meta_tag



EVERY LAYOUT

RUDIMENTS

- Boxes
- Composition
- Units
- Global and local styling
- Modular scale
- Axioms

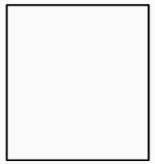
LAYOUTS

Picture index

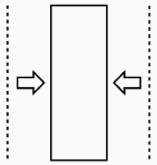
Layouts

Each of the following *layout primitives* is documented with a code generator, and includes a custom element implementation. You can use them together, in composition, to create robust and responsive layouts without the need for `@media` breakpoints.

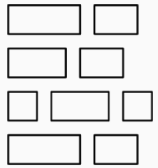
The Stack



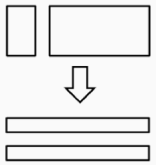
The Box



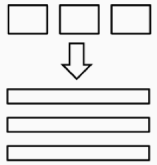
The Center



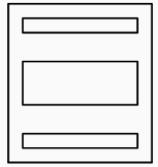
The Cluster



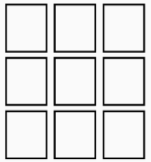
The Sidebar



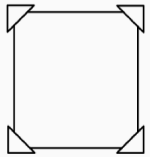
The Switcher



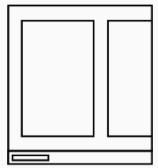
The Cover



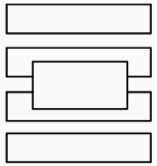
The Grid



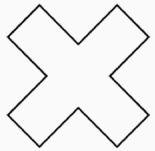
The Frame



The Reel



The Imposter



The Icon

■

Every Layout is created by
Heydon Pickering & Andy Bell
Privacy Policy - Terms and conditions

BBC

@joshtumath

“By designing to *ideal element dimensions*, and tolerating reasonable variance, you can essentially do away with `@media` breakpoints. Your component handles its own layout, intrinsically, and without the need for manual intervention.”

Heydon Pickering, Andy Bell • Every Layout

BBC

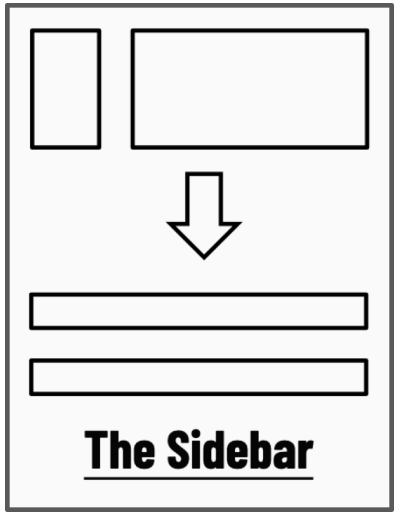
@joshtumath

“For instance, we might want to create a classic sidebar layout, wherein one of two adjacent elements has a fixed width, and the other—the `main` element, if you will—takes up the rest of the available space. This should be responsive, without `@media` breakpoints, and we should be able to set a container-based breakpoint for wrapping the elements into a vertical configuration.”

Heydon Pickering, Andy Bell • Every Layout

Making an intrinsic sidebar

Let’s have a go!




```
<div class=”with-sidebar”>
  <main class=”main-column”>
    ...
  </main>

  <aside class=”side-column”>
    ...
  </aside>
</div>
```

BBC

Making an intrinsic sidebar

Let’s have a go!

We’re using CSS flexbox.

The side column will wrap if the main column doesn’t have at least 50% room.

```
.with-sidebar {
  display: flex;
  flex-wrap: wrap;
  gap: 1rem;
}

.with-sidebar > * {
  flex: 1;
}

.main-column {
  flex-basis: 0;
  flex-grow: 999;
  min-width: 50%;
}
```

BBC

But hold on. The sidebar column doesn’t have a width.

Let your content's size bubble up

It's the content's responsibility to know its size.

We could say a paragraph in a 'rich text area' has a minimum width.

```
.rich-text p {
  min-width: 33ch;
}
```

BBC

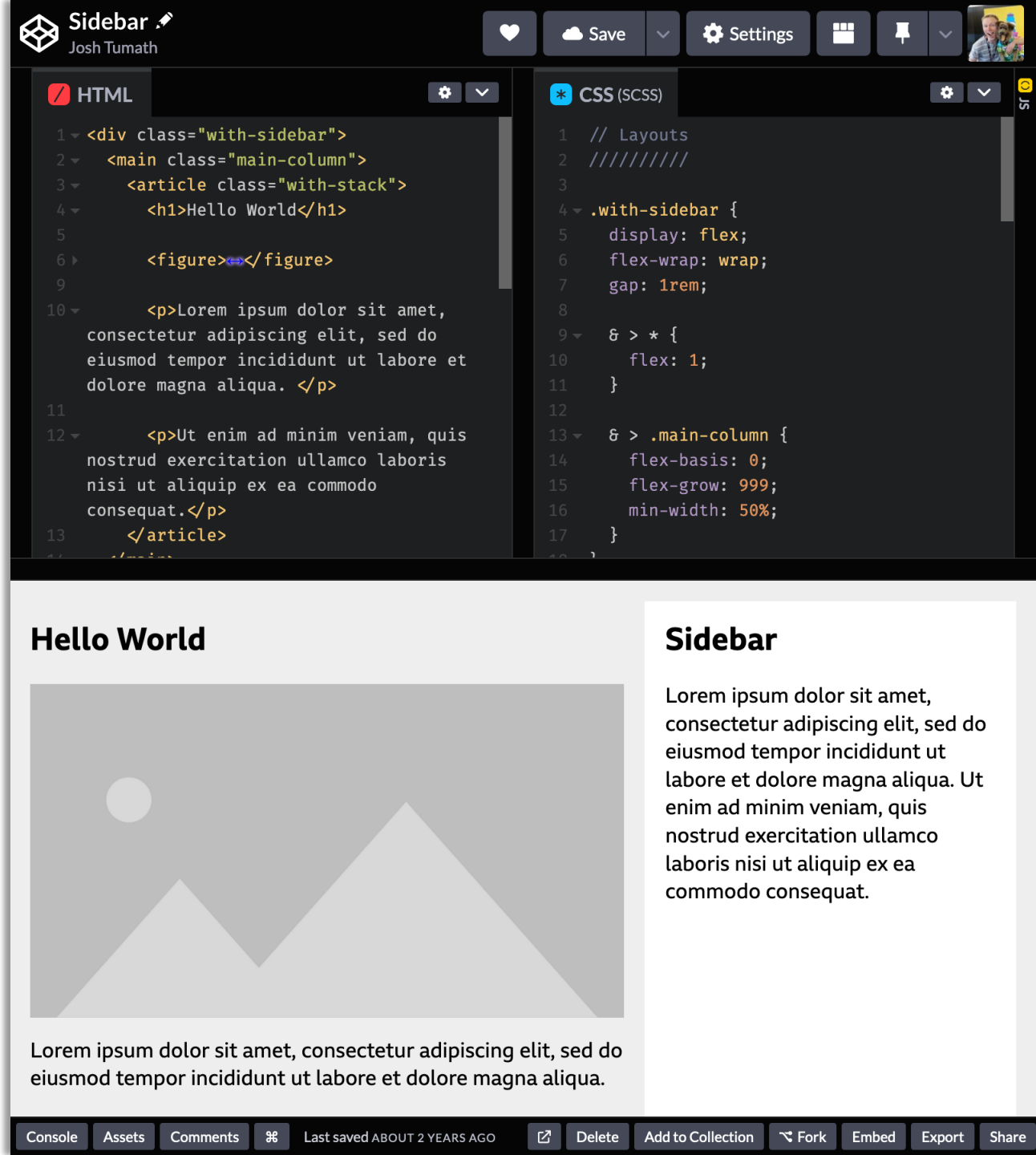
@joshtumath

Demo: Intrinsic sidebar

https://codepen.io/joshtumath/pen/XWKKYKX

BBC

@joshtumath



Making an intrinsic grid

Let's make a grid of cards. It will try to fit as many cards as it can into each row.

We tell the columns to `repeat()` forever. They will be at least `10rem` wide in width. At most, they will each take up the same 1 fraction of space.

We use the new `min()` function in case the grid is used in a box thinner than `20rem`.

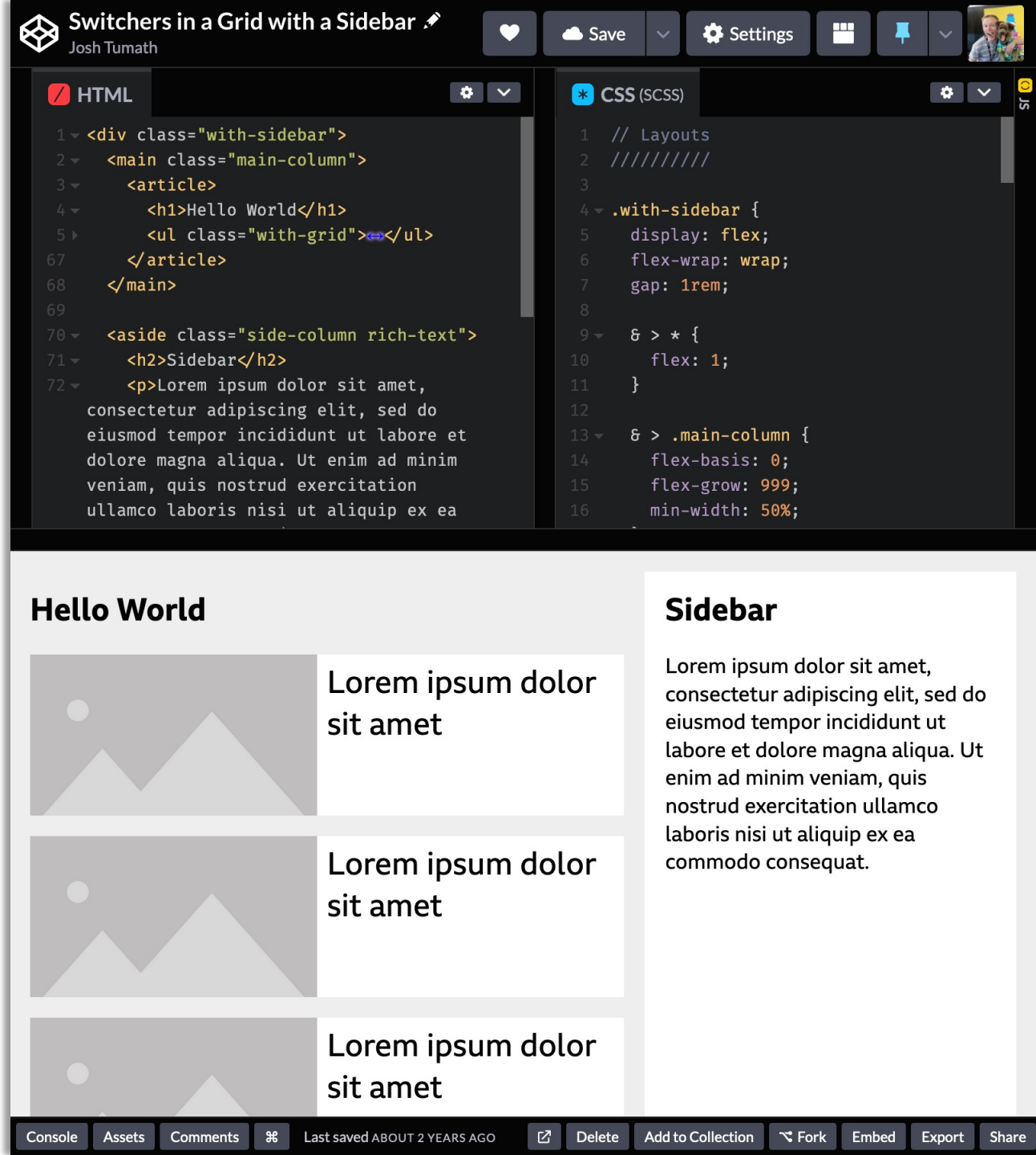
```
.grid {
  display: grid;
  grid-gap: 1rem;
  grid-template-columns: repeat(
    auto-fill,
    minmax(10rem, 1fr)
  );
}

@supports (width: min(10rem, 100%)) {
  .grid {
    grid-template-columns: repeat(
      auto-fit,
      minmax(min(10rem, 100%), 1fr)
    );
  }
}
```

Demo: Intrinsic page

https://codepen.io/joshtumath/pen/LYVOdbR

BBC

@joshtumath



Container queries and other new CSS features

BBC

@joshtumath

This is where the magic happens

Container Queries are like Media Queries, but you can query the width of a box instead

Making a Container Query

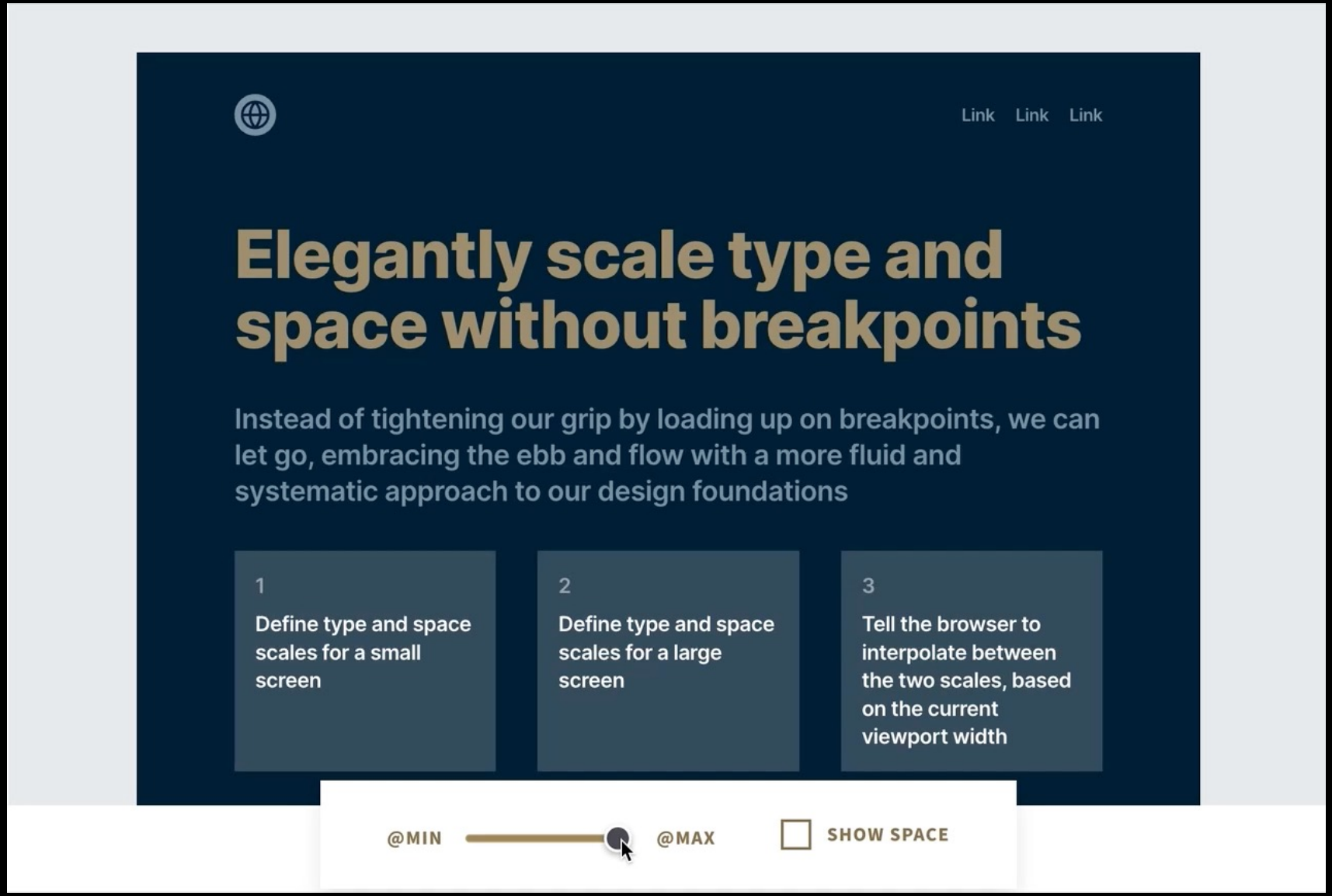
Let's use container queries to change a card from portrait to landscape.

We set the grid items to be containers. Then, anything inside them can respond to a container query. (Naming the container is optional.)

```
.grid > * {
  container-type: inline-size;
  container-name: grid-item;
}

.card {
  display: flex;
  flex-direction: column;
}

@container grid-item (inline-size > 30rem) {
  .card {
    flex-direction: row;
  }
}
```

BBC

@joshtumath



BBC

@joshtumath

Fluid typography and spacing

Let's use viewport units to smoothly tween to different font sizes and gaps.

https://utopia.fyi

```
body {
  font-size: clamp(1.00rem, calc(0.76rem +
1.22vw), 1.63rem);
}
```

BBC

Fluid typography and spacing

You can also use container units!

They'll fallback to viewport units if the element is not in a container.

```
body {
  font-size: clamp(1.00rem, calc(0.76rem +
1.22cqi), 1.63rem);
}
```

The parent selector, :has()

The :has pseudo-class lets you select something based on the parent

If the grid's last grid item is an odd number, then select the first grid item and apply this style…

```
.grid:has(> :last-child:nth-child(odd)) >
:first-child {
  grid-column: span 2;
}
```

BBC

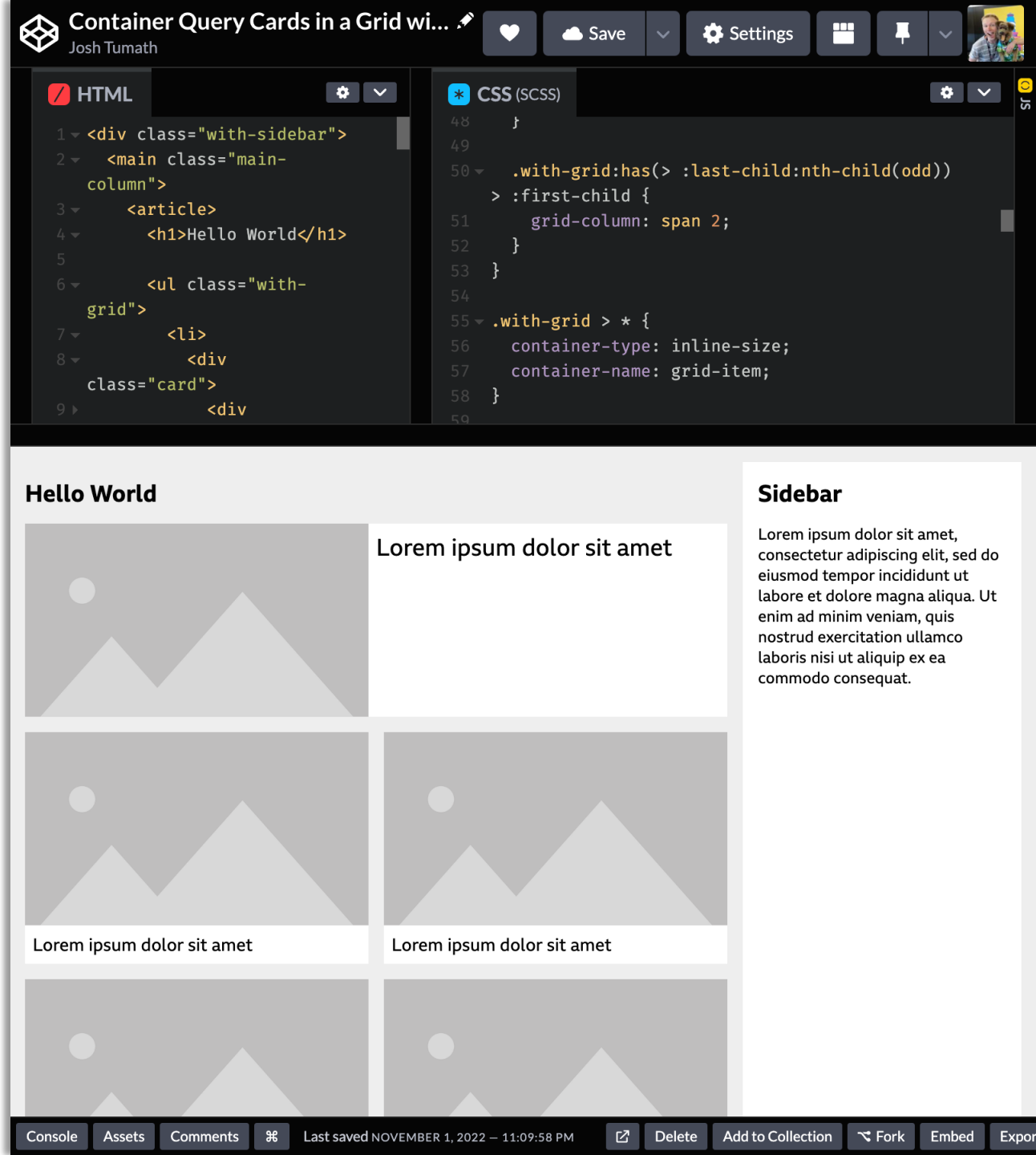
@joshtumath

Demo: Container Queries, fluid typography and :has()

https://codepen.io/joshtumath/pen/XWYXOVY

BBC

@joshtumath



View Transitions
CSS View Transitions 1

Layers
CSS Cascading and Inheritance 5

Be sure to check out what else modern CSS can do!

Subgrid
CSS Grid Layout 2

Nesting
CSS Nesting 1

Scroll-driven animations
CSS Animations 2
CSS Scroll-driven animations 1

Style queries
CSS Containment 3

BBC

Conclusion

- 📦 Everything is a box
- 🔀 The CSS `display` property changes the layout method
- 📱 A lot of recent innovation in CSS has made modern layouts possible
- 🧠 We can achieve a lot of intrinsic layouts without container queries
- 🥳 Container queries and `:has()` are here and they will change everything

Further reading

If you just can’t get enough of CSS

- **Everything You Know About Web Design Just Changed, J. Simmons**
  The original talk that started the IWD movement.
- **every-layout.dev, H. Pickering and A. Bell**
  A great online book explaining the basics of CSS and reusable Intrinsic Web Design patterns. (There is a paywall for some content.)
- **Intrinsic website example, A. Argyle**
  Open your browser’s Dev Tools and check out how it’s been made.
- **MDN, Mozilla**
  The fantastic learning resource and reference that explains all CSS features and properties.