

**UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II
WEB TECHNOLOGIES — LECTURE 04**

CSS: LAYOUTS, RESPONSIVE DESIGN

Luigi Libero Lucio Starace, PhD

luigiliberolucio.starace@unina.it

<https://luistar.github.io>

<https://www.docenti.unina.it/luigiliberolucio.starace>



PREVIOUSLY, ON WEB TECHNOLOGIES

In the last lecture, we have learned:

- What **CSS** is and how to include it in HTML documents
- Some basic **styling properties** (color, background, font-style, ...)
- Selectors
- The Cascade algorithm
- Inheritance

CSS SIZING UNITS

CSS SIZING UNITS OVERVIEW

Some CSS properties can be used to change the **size** of elements or their box model properties:

- **width, height, font-size, margin, padding, border, ...**

When sizing elements in CSS it possible to use:

- **Absolute** lengths
- **Relative** lengths (depend on other sizes, i.e.: font or viewport sizes)

CSS: ABSOLUTE SIZING UNITS

- Absolute lengths are defined by using a number and one of the supported length units

```
div {  
    background: red;  
    width: 2.54cm; /* centimeters */  
    height: 1in; /* inches */  
    border: 2mm solid black; /* millimiters */  
    color: white;  
    font-size: 24px; /* pixels */  
}
```

```
<div>hello</div>
```

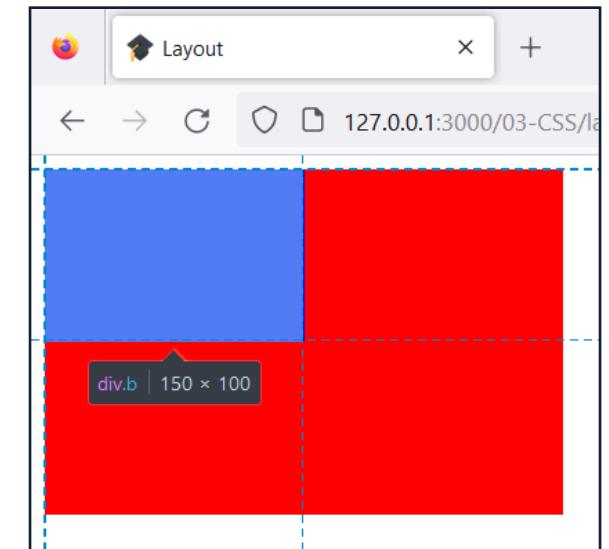


CSS: PERCENTAGE SIZING

- Percentages define sizes as relative to the parent elements
- They can be used with many properties, including:
 - width, height, font-size, margin, padding, ...

```
.a {  
    width: 300px; height: 200px;  
    background: red;  
}  
.b {  
    width: 50%; /*150px*/  
    height: 50%; /*100px*/  
    background: blue;  
}
```

```
<div class="a">  
    <div class="b"></div>  
</div>
```



CSS: RELATIVE SIZING UNITS (FONTS)

Font size-relative units include:

- em: **1em** represents the current font-size. **1.5em** is 50% larger than the current font-size. Historically, in typography, font size was derived from the width of the capital “M”, hence the name of the unit “em”.
- rem: **1rem** represents the font-size computed at the root element (default 16px)



```
p {  
    font-family: sans-serif;  
    font-size: 1.5rem;  
}  
  
span {  
    font-size: 1.5em;  
}
```

```
<p>HELLO<span>SIZES</span></p>
```



CSS: RELATIVE SIZING UNITS (VIEWPORT)

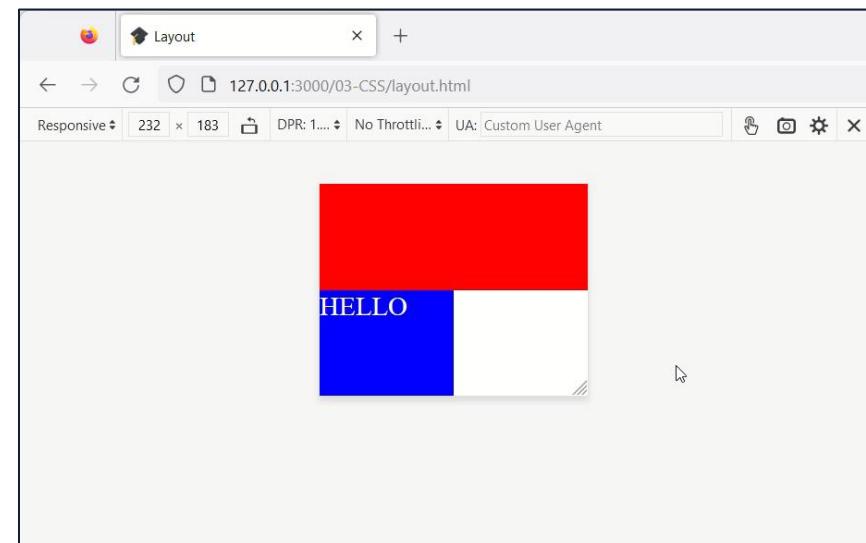
The **viewport** is the browser window.

Viewport-relative size units include:

- **vw**: **1vw** represents 1% of the width of the current viewport.
- **vh**: **1vh** represents 1% of the height of the current viewport.

```
div {height: 50vh;}  
.a {background: red;}  
.b{  
    background: blue; color: white;  
    width: 50vw; font-size: 10vw;  
}
```

```
<div class="a"></div>  
<div class="b">HELLO</div>
```

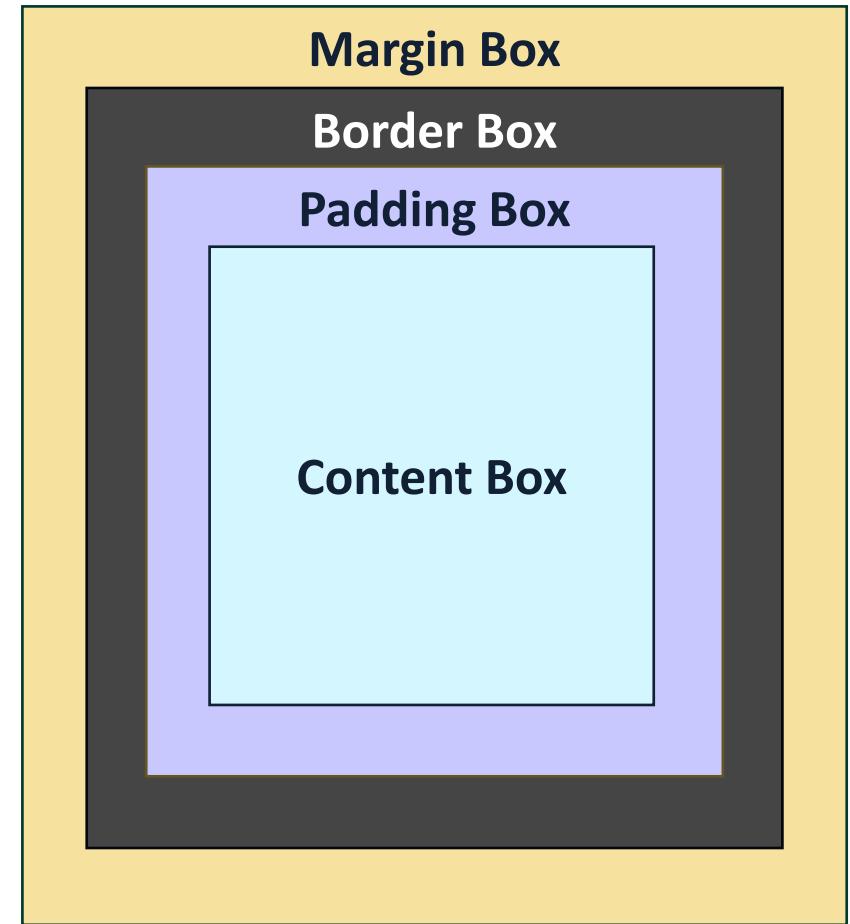


LAYOUTS



THE BOX MODEL

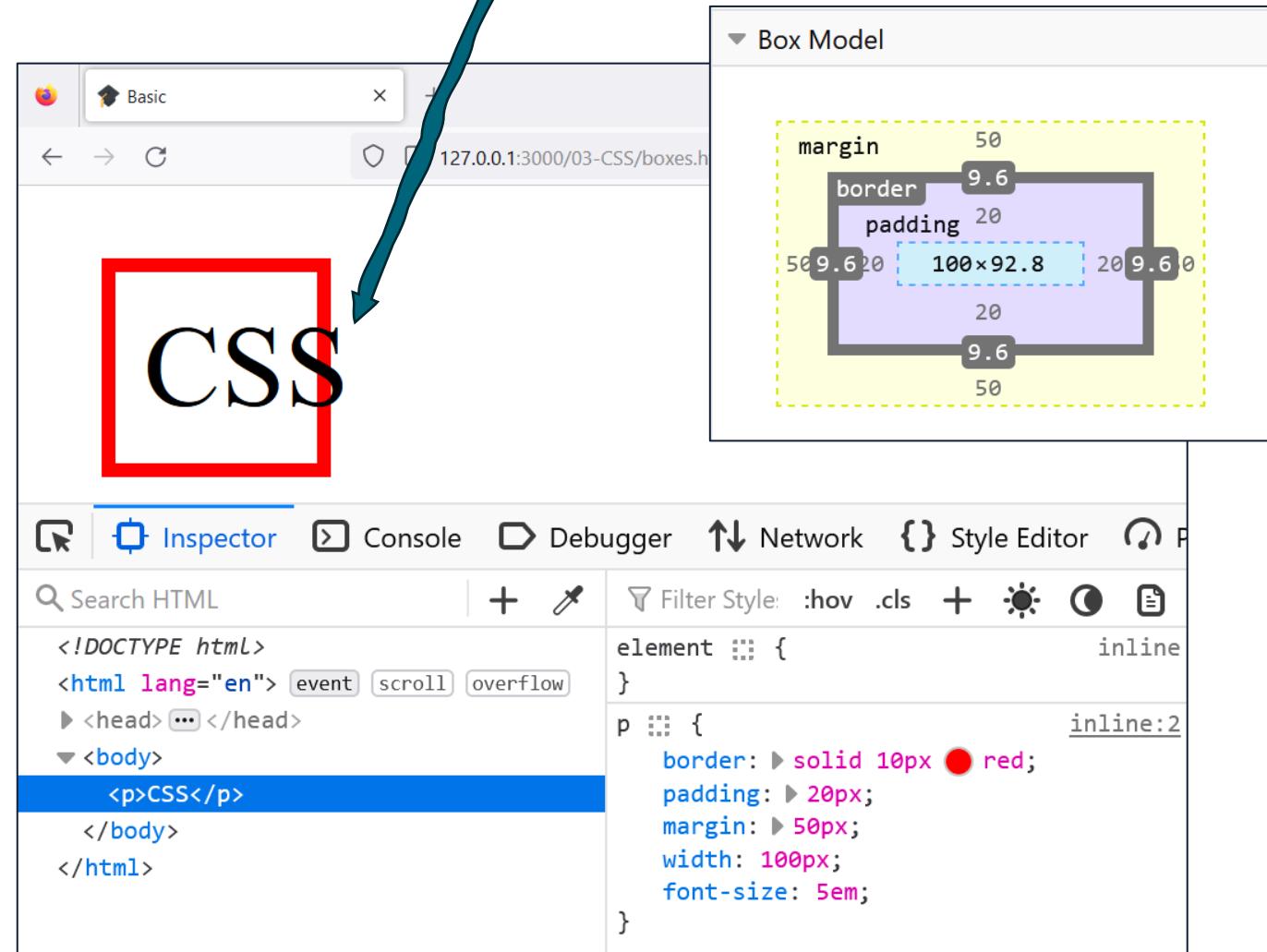
- Every HTML element is a **box**
- Boxes are made up of distinct areas
- **Content box** is where the children of the element live
- **Padding** («inner spacing») separates the content box from the border
- **Border** is the boundary of elements
- **Margin** creates space around elements



THE BOX MODEL

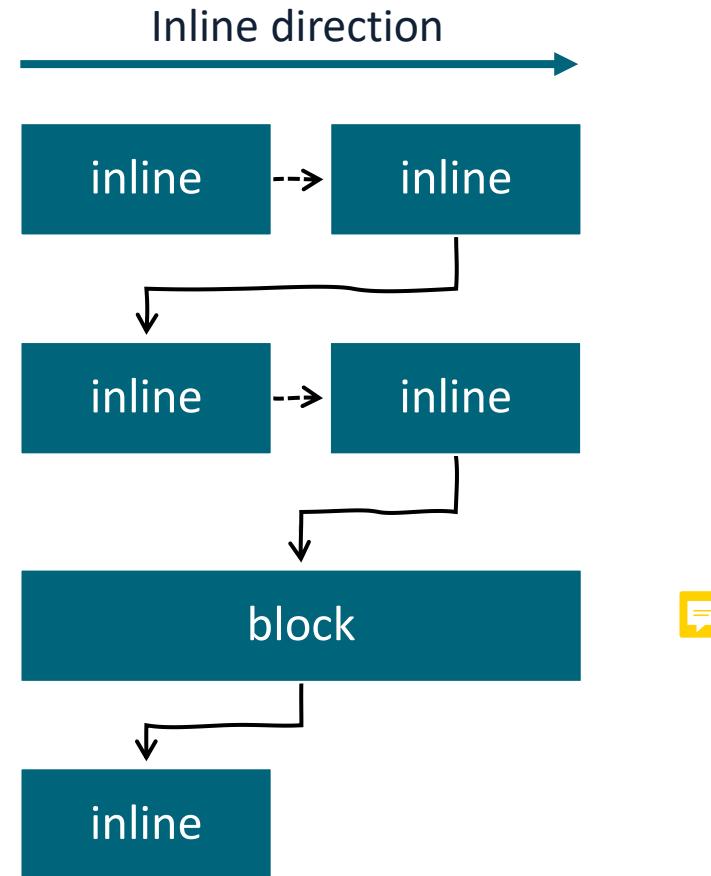
- The size of each area can be defined using CSS declarations
- The behaviour and appearance of boxes is determined by their:
 - **Layout mode**
 - **Content**
 - **Box model properties**

Content might be too big to fit in its box. The part of content that does not fit is called «**overflow**».



CSS FLOW LAYOUT

- By default, boxes are displayed using the **flow layout** (a.k.a. **normal flow**)
- **Inline** elements display in the inline direction
- **Block** elements display one after another

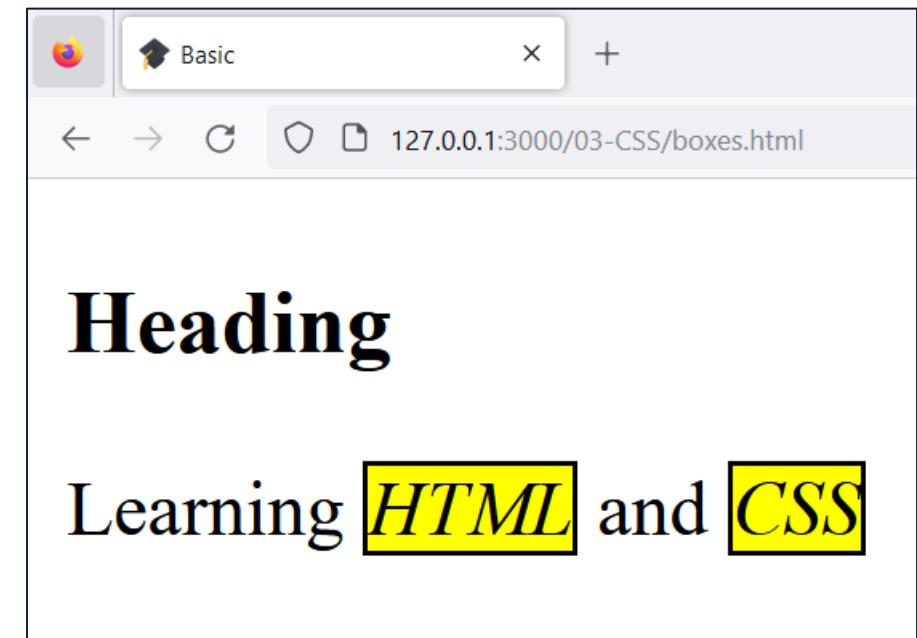


THE DISPLAY PROPERTY: INLINE

- `display: inline` positions the box next to the previous one in the inline direction, like words in a sentence
- Anchors `<a>`, `` and `` are typically displayed inline in default user agent styles

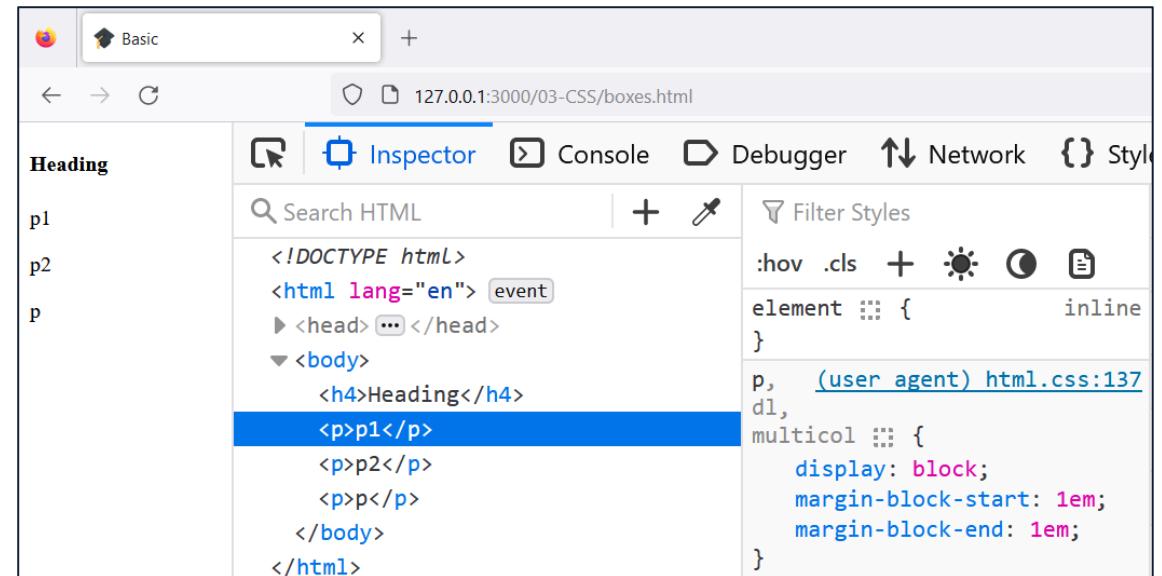
```
em {  
    display: inline; /* default */  
    background: yellow; border: 1px solid;  
    width: 50px; height: 50px; /* ignored */  
}
```

```
<h3>Heading</h3>  
<p>  
    Learning <em>HTML</em> and <em>CSS</em>  
</p>
```

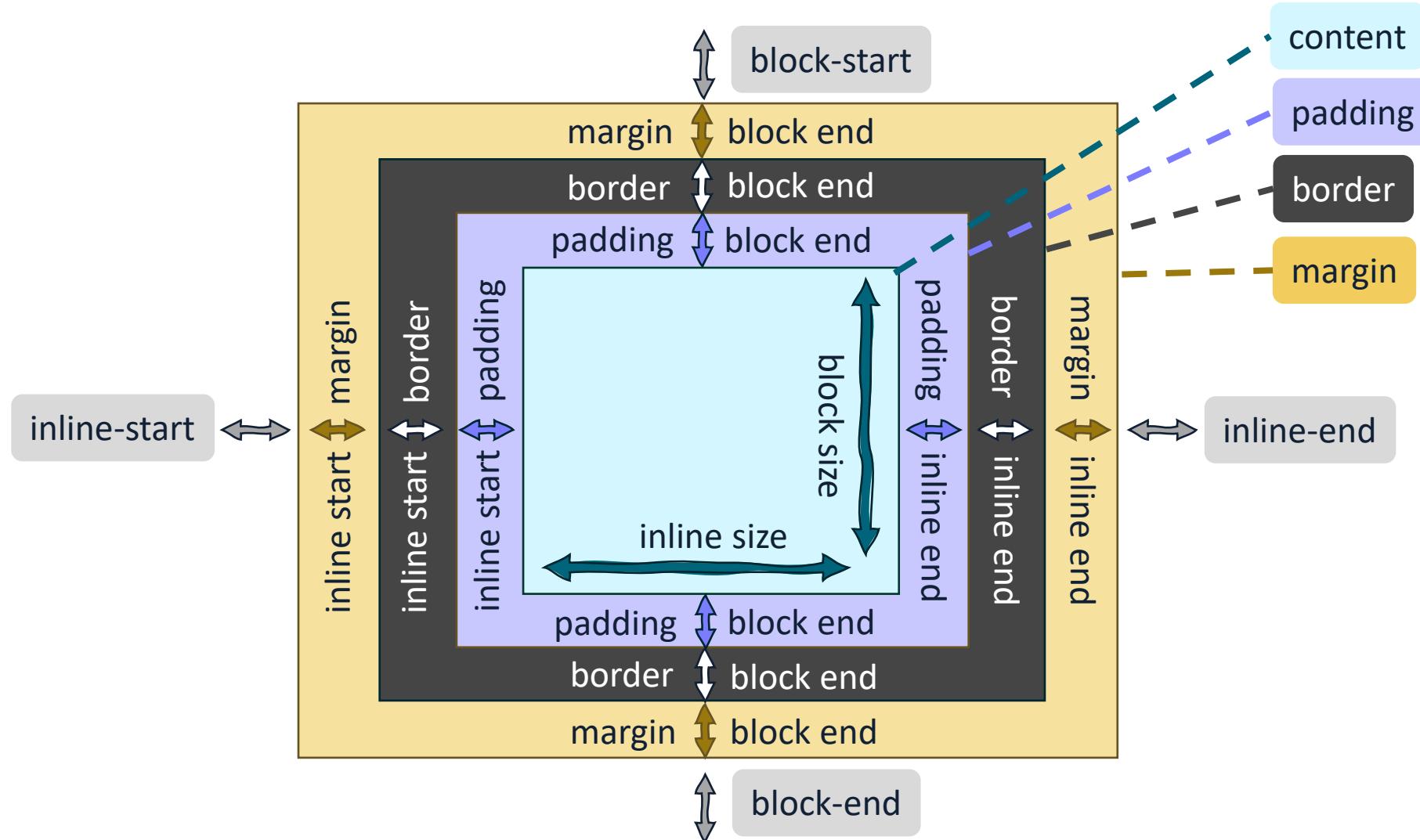


THE DISPLAY PROPERTY: BLOCK

- `display: block` positions the box on a **new, dedicated line**
- Paragraphs `<p>` and headings `<hx>` are typically displayed as blocks in default user agent styles
- Unless specified otherwise, blocks expand to the **entire size of the inline dimension**



THE BOX MODEL

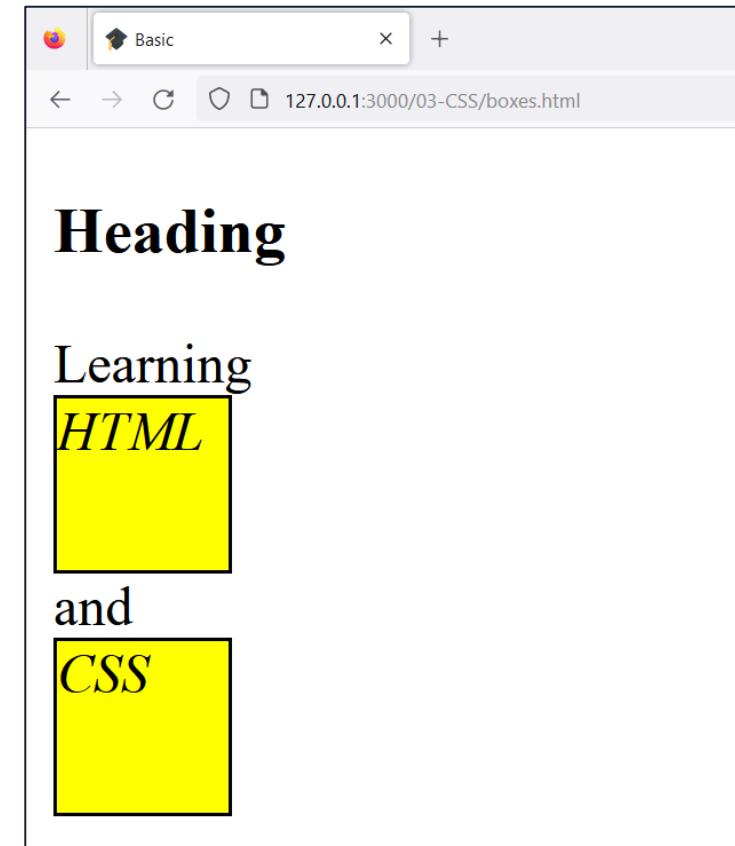


THE DISPLAY PROPERTY: INLINE VS BLOCK

- What happens if we use display **display: block** on the ****s?

```
em {  
    display: block;  
    background: yellow;  
    border: 1px solid;  
    width: 50px; height: 50px;  
}
```

```
<h3>Heading</h3>  
<p>  
    Learning <em>HTML</em> and <em>CSS</em>  
</p>
```

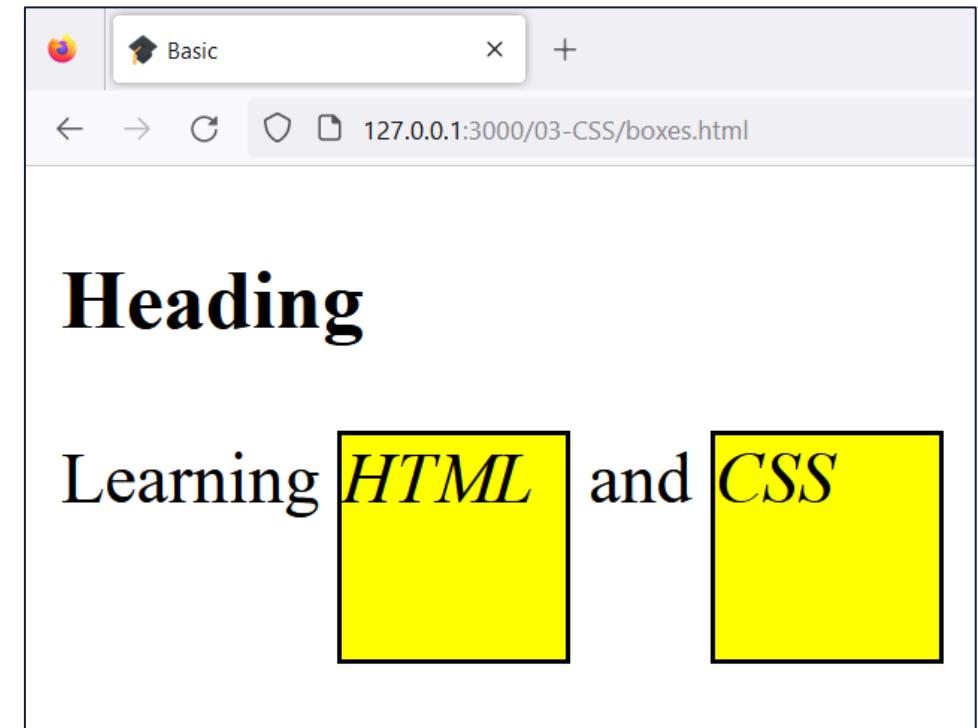


THE DISPLAY PROPERTY: INLINE-BLOCK

- `display: inline-block` is the same as `inline`, but allows also to set a width and a height

```
em {  
    display: inline-block;  
    background: yellow;  
    border: 1px solid;  
    width: 50px; height: 50px;  
}
```

```
<h3>Heading</h3>  
<p>  
    Learning <em>HTML</em> and <em>CSS</em>  
</p>
```

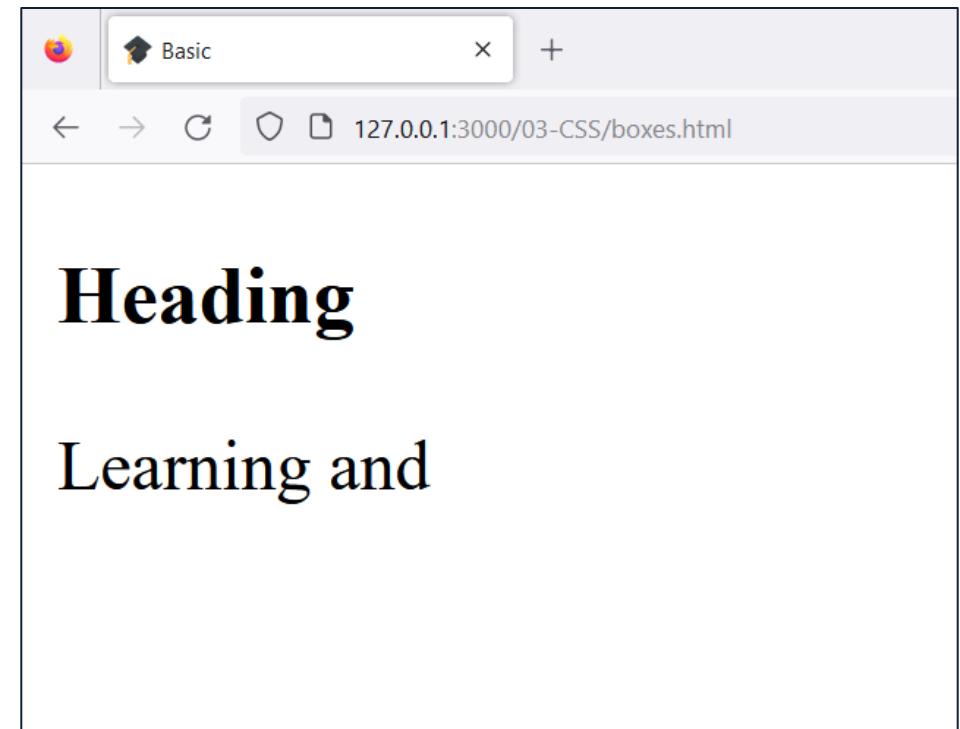


THE DISPLAY PROPERTY: NONE

- `display: none` can be used to completely **remove** the element from the visualization

```
em {  
    display: none;   
    background: yellow;  
    border: 1px solid;  
    width: 50px; height: 50px;  
}
```

```
<h3>Heading</h3>  
<p>  
    Learning <em>HTML</em> and <em>CSS</em>  
</p>
```



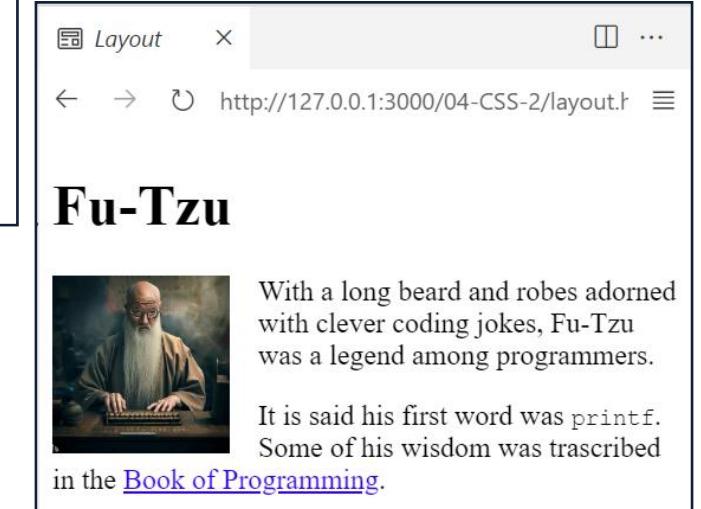
FLOATS

The **float** property can be used to make elements «float» in the given direction, and have subsequent sibling elements **wrap** around it

```
<h1>Fu-Tzu</h1>

<p>With a long beard and robes adorned with clever
coding jokes, Fu-Tzu was a legend among programmers.</p>
<p>It is said his first word was <code>printf</code>.
Some of his wisdom was transcribed in the
<a href="index.html">Book of Programming</a>.</p>
```

```
img {
  width: 20vw; min-width: 100px;
  float: left; margin-right: 1rem;
}
```



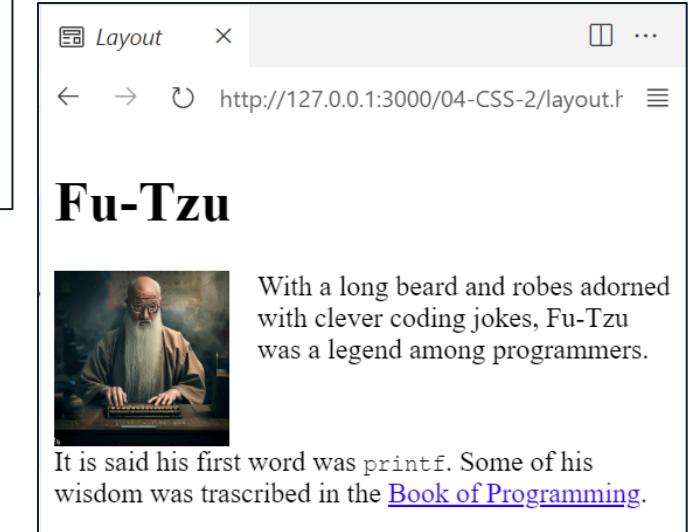
FLOATS

The **clear** property can be used to prevent a subsequent sibling to wrap around a floating element

```
<h1>Fu-Tzu</h1>

<p>With a long beard and robes adorned with clever
coding jokes, Fu-Tzu was a legend among programmers.</p>
<p>It is said his first word was <code>printf</code>.
Some of his wisdom was transcribed in the
<a href="index.html">Book of Programming</a>.</p>
```

```
img {
  width: 20vw; min-width: 100px;
  float: left; margin-right: 1rem;
}
p+p { clear: both; }
```



POSITIONING

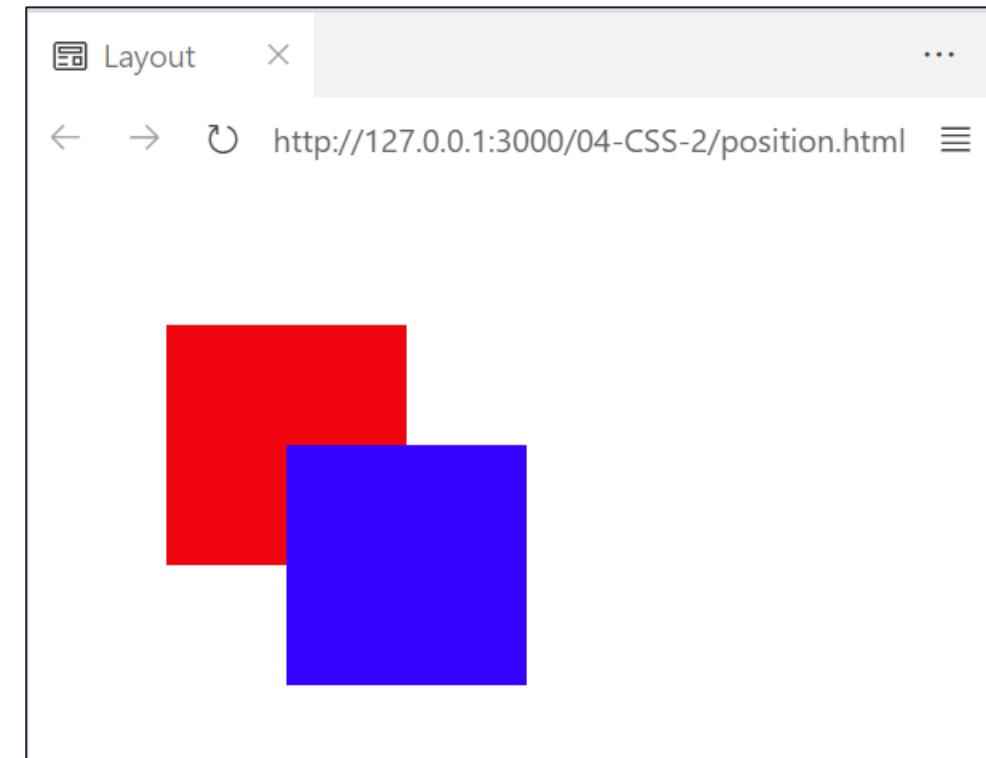
- The **position** property changes the way an element behaves in the normal flow of the document.
- The default value for the property is **static**
- Other possible values are:
 - **relative**
 - **absolute**
 - **fixed**
 - **sticky**

POSITION: RELATIVE

- Elements with **position: relative** are positioned relative to their normal position

```
div {  
  position: relative;  
  top: 50px; left: 50px;  
  background: red;  
  width: 100px; height: 100px;  
}  
div div {  
  background: blue;  
}
```

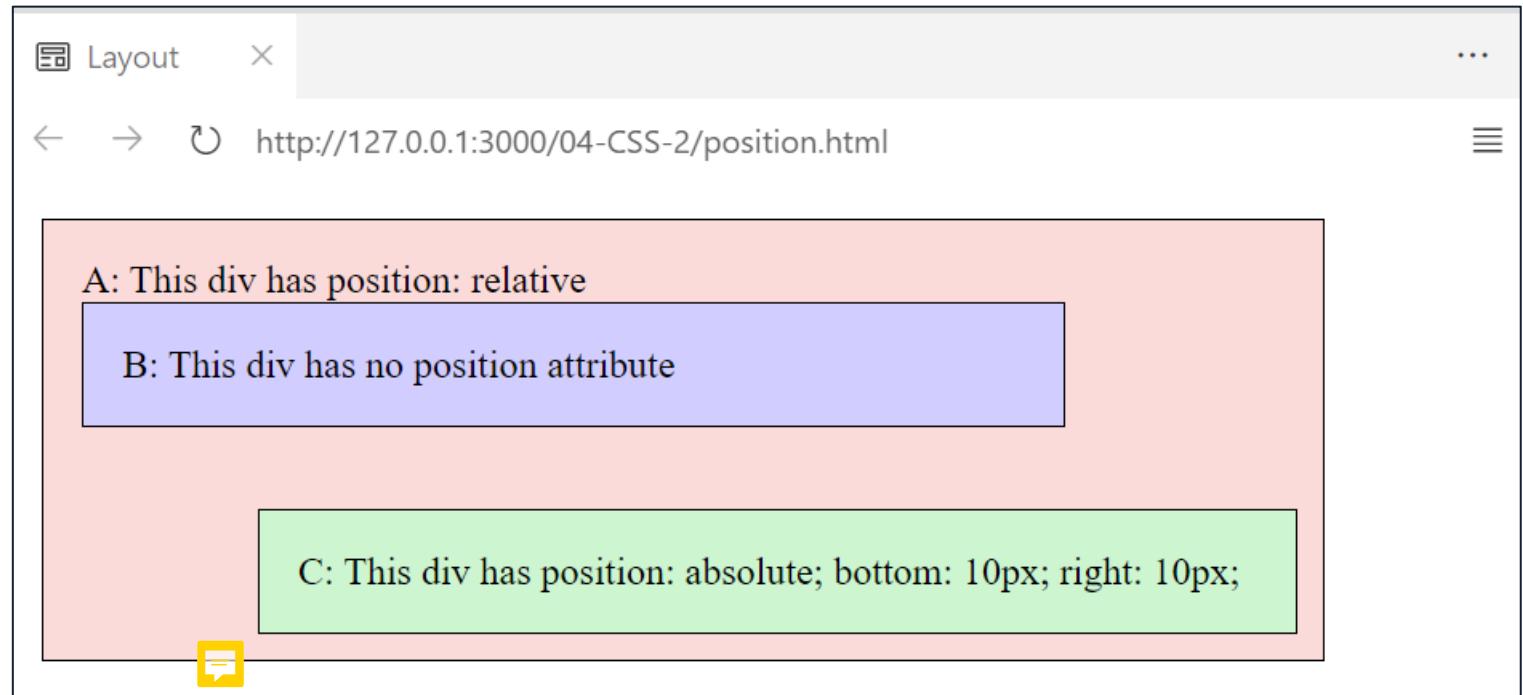
```
<div><div></div></div>
```



POSITION: ABSOLUTE

- Elements with **position: absolute** are positioned relative to their nearest relative-positioned ancestor

```
<div class="a">
  <!-- text -->
  <div class="b">
    <!-- text -->
    <div class="c">
      <!-- text -->
    </div>
  </div>
</div>
```

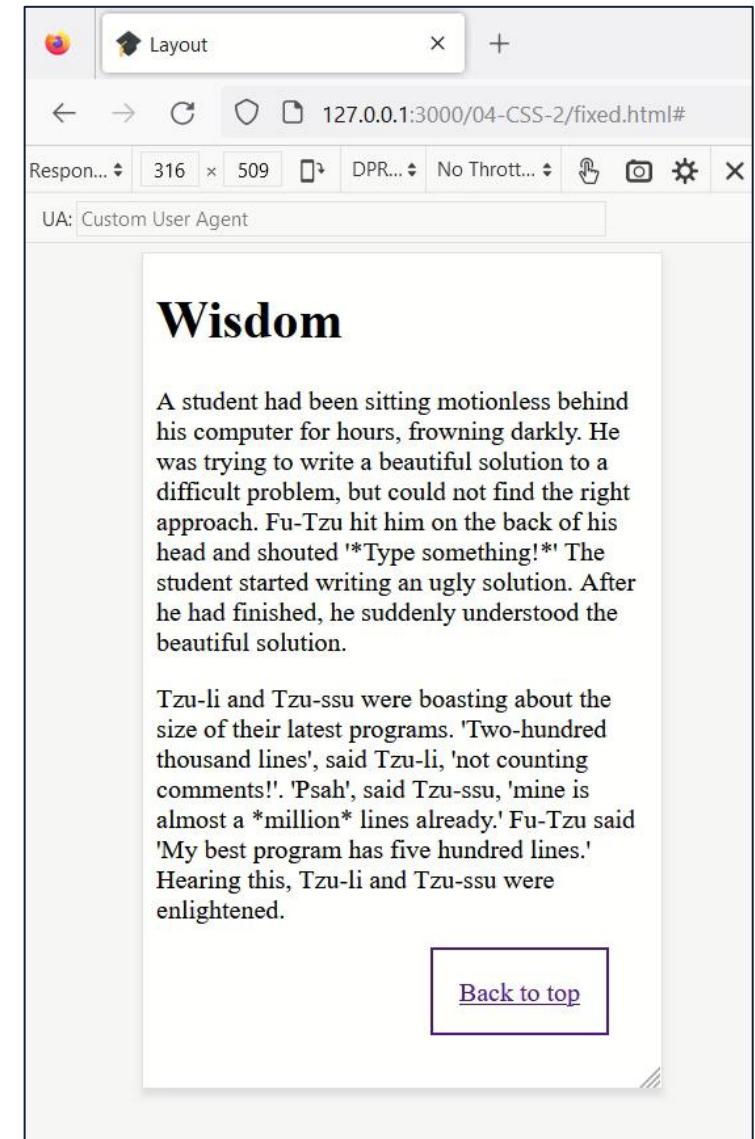


POSITION: FIXED

- Elements with **position: fixed** are positioned relative to the viewport

```
<h1>Wisdom</h1>
<p><!-- text --></p>
<p><!-- text --></p>
<a href="#">Back to top</a>
```

```
a {
  position: fixed;
  bottom: 2rem; right: 2rem;
  background-color: white;
  border: 2px solid;
  padding: 1em;
}
```



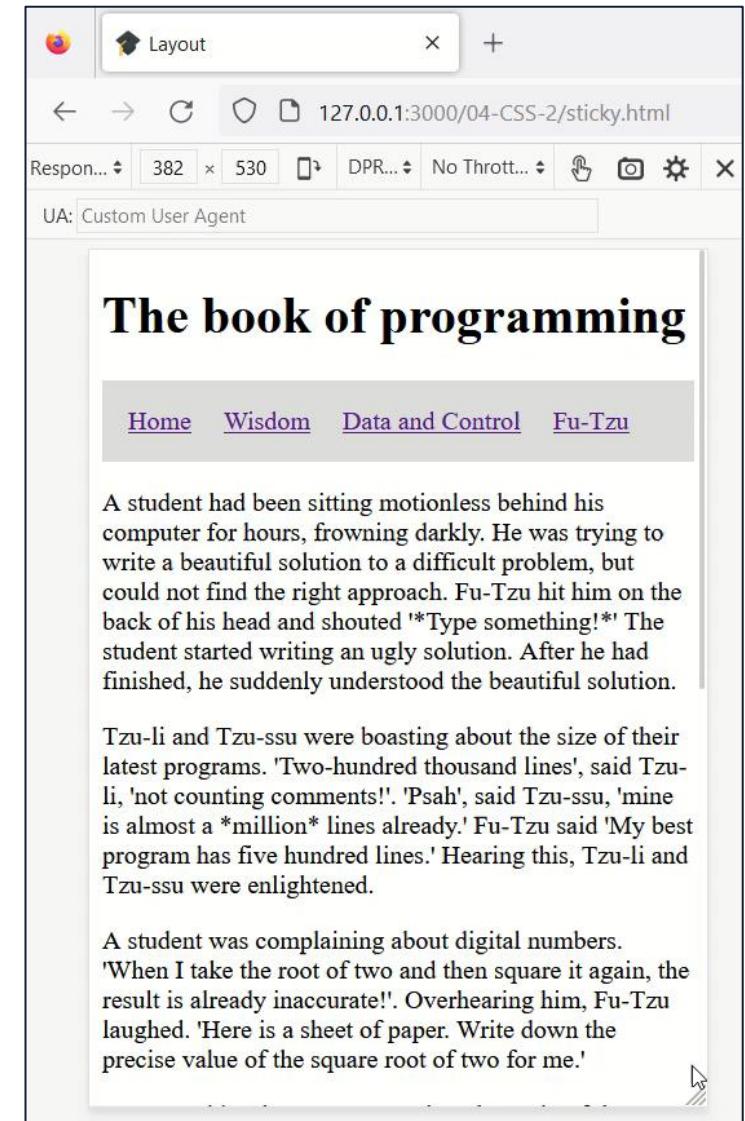
POSITION: STICKY



- Elements with **position: sticky** are positioned based on the user's scroll position
- Sort of a relative/fixed hybrid
 - Relative until it crosses a threshold
 - Fixed until it reaches the boundary of its parent

```
<h1>The book of programming</h1>
<nav><!-- links --></nav>
<p><!-- text --></p><p><!-- text --></p>
```

```
nav {
  padding: 1em; background: gainsboro;
  position: sticky; top: 0px;
}
nav a { padding-right: 1em; }
```

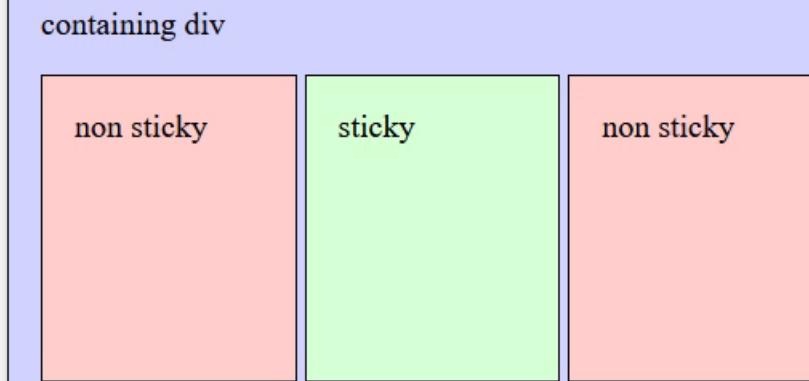


POSITION: STICKY

```
<div class="container">
  <p>containing div</p>
  <div class="item"></div>
  <div class="item sticky"></div>
  <div class="item"></div>
</div>
```

```
.sticky {
  position: sticky;
  top: 10px;
  background: rgb(212, 255, 212);
}
```

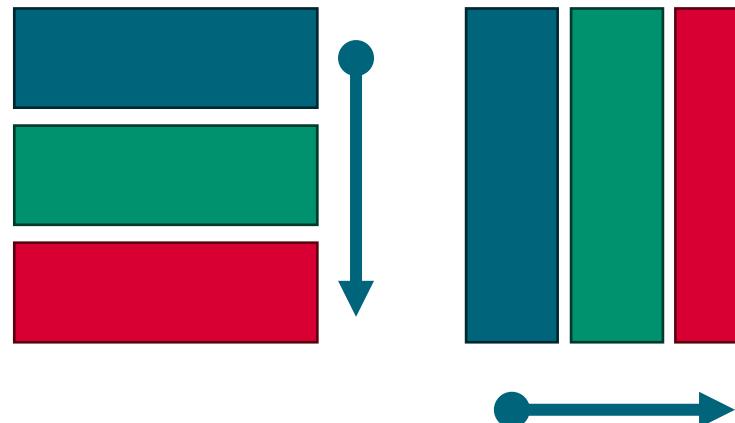
Sticky Example



MODERN CSS LAYOUTS: FLEX AND GRID

Modern CSS features two additional layout mechanisms in addition to the normal flow: **Flexbox** and **Grid**

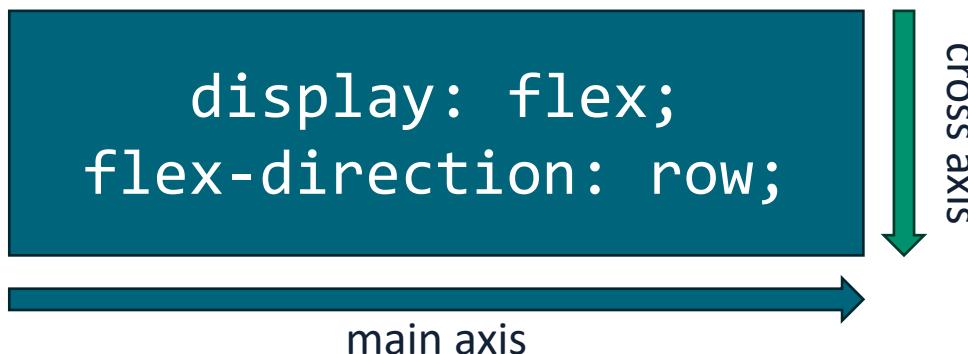
- **Flexbox** is designed for **one-dimensional layouts** (either horizontal or vertical)
- **Grid** is designed for **two-dimensional layouts**



FLEXBOX: FLEX CONTAINERS

Flex containers are declared using the **display: flex** property

- They are **block-level** elements, with **flex item** children
- Each flex container has a **main** and a **cross axis**
- The main axis is set using the **flex-direction** property (default is row)
- The cross axis is orthogonal to the main one

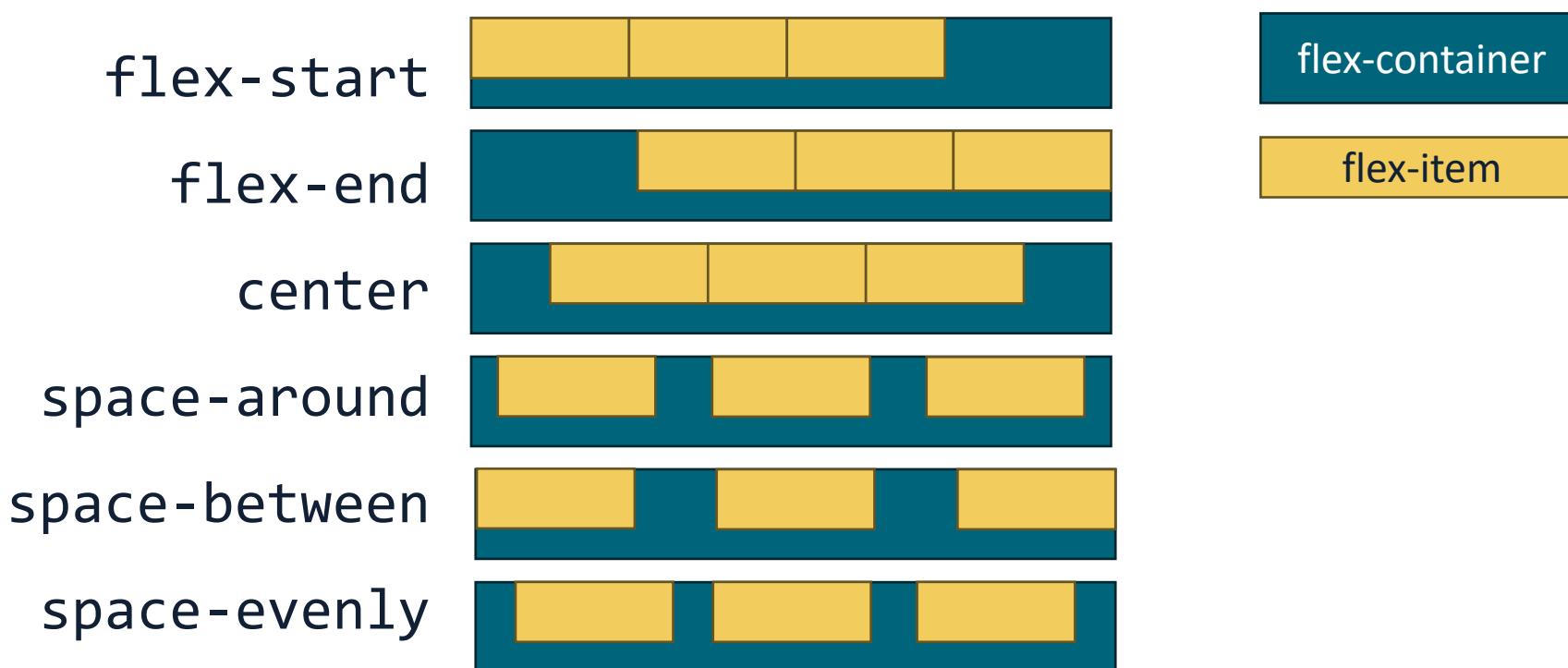


FLEXBOX: FLEX ITEMS

- The children of the flex container are converted to **flex items**
 - Support **flex** properties, to specify how they behave in the flex container
- They are placed along the main axis
- Setting **flex-grow** on a flex item makes it expand to all the available space in the main axis.
- Setting **flex-shrink** can be used to control whether the flex item can reduce its base size along the main axis to fit in the flex container.
- Overflows are possible, and can be controlled using the **flex-wrap** property on the flex container.

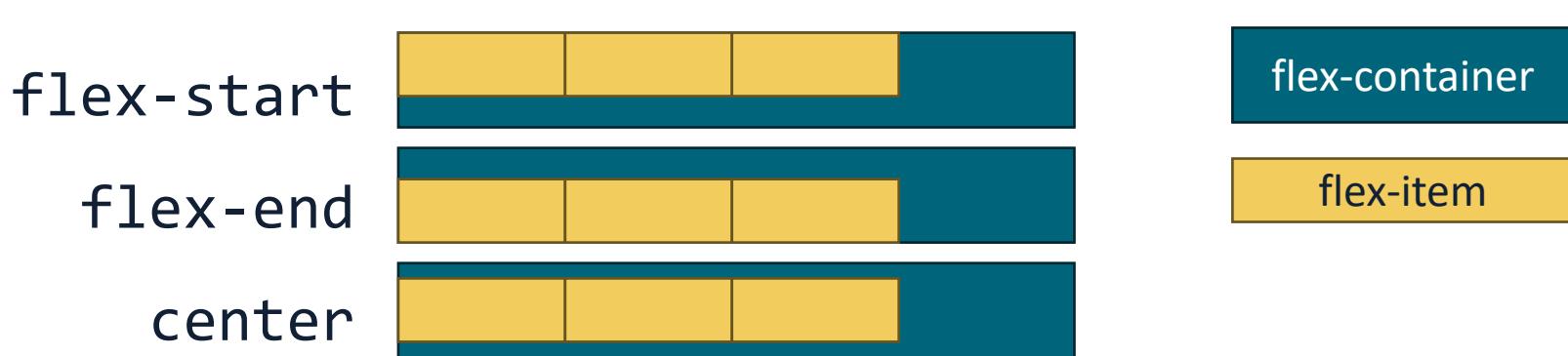
FLEXBOX: JUSTIFY–CONTENT

The **justify-content** flex container property specifies how the **free space** along the main axis is to be managed. Possible values include:



FLEXBOX: ALIGN–CONTENT

The **align-content** flex container property specifies how the **free space** along the cross axis is to be managed. Possible values include:

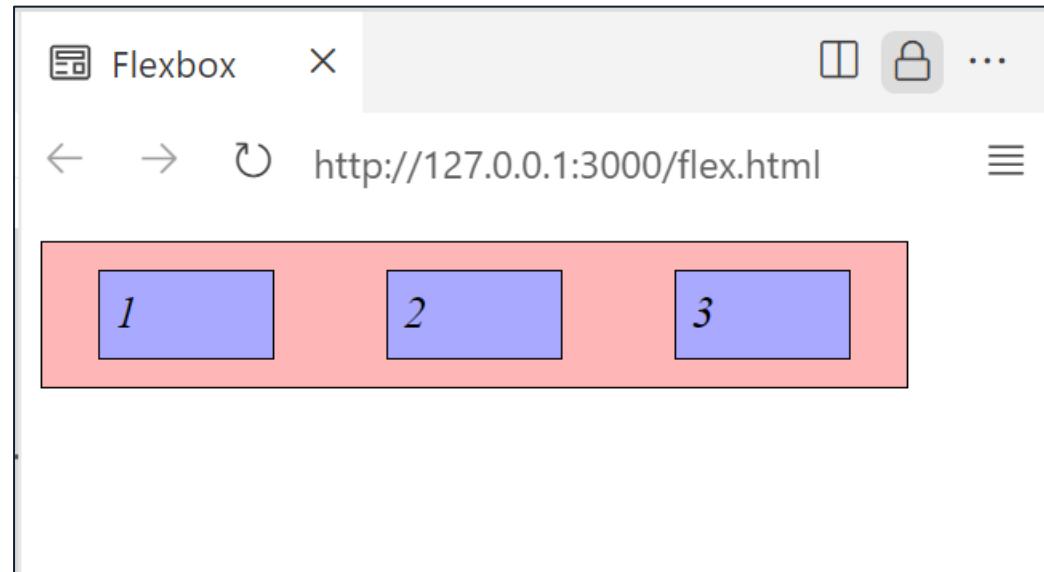


<https://flexbox.help/> is useful to visualize and practice with these properties

FLEXBOX: EXAMPLE

```
.container {  
    width: 300px;  
    height: 50px;  
    background: rgb(255, 182, 182);  
    display: flex;  
    justify-content: space-around;  
    align-items: center;  
}  
  
.item {  
    height: 20px; width: 50px;  
    padding: 5px;  
    background: rgb(169, 169, 255);  
}
```

```
<div class="container">  
    <div class="item special"><em>1</em></div>  
    <div class="item "><em>2</em></div>  
    <div class="item"><em>3</em></div>  
</div>
```



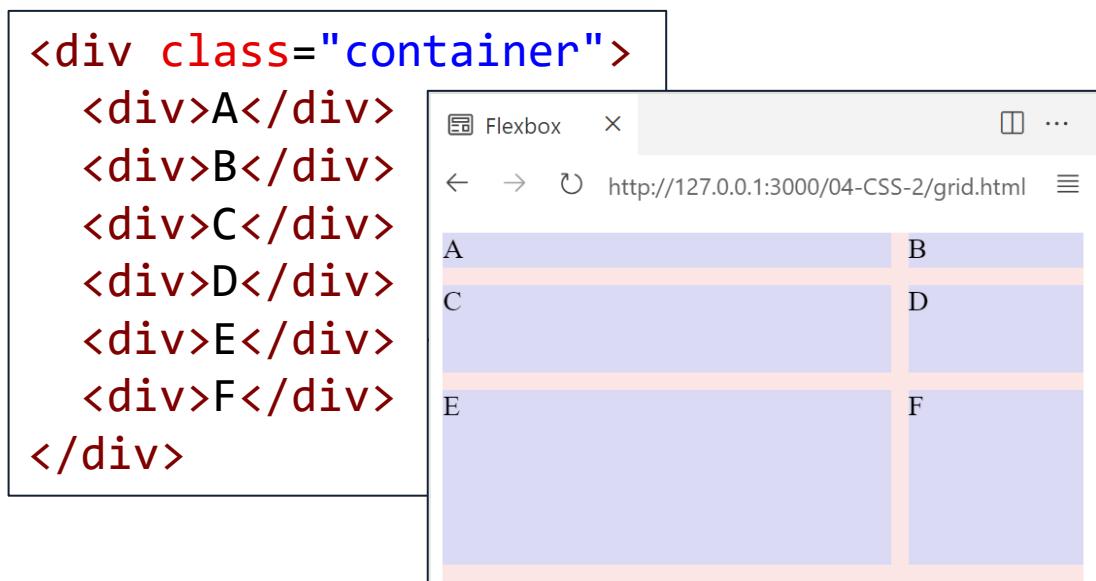
GRID

Grid is a **two-dimensional** layout, based on **rows** and **columns**

- Grid containers are declared using the **display: grid** property
 - The direct **children** of grid containers are **grid items**.
 - Containers define **number** and **size** of their rows and columns

```
.container {  
  display: grid;  
  /* two columns */  
  grid-template-columns: 1fr 100px;  
  /* three rows */  
  grid-template-rows: 20px 50px 100px;  
  gap: 10px;  
}
```

```
<div class="container">  
  <div>A</div>  
  <div>B</div>  
  <div>C</div>  
  <div>D</div>  
  <div>E</div>  
  <div>F</div>  
</div>
```



GRID: THE FR UNIT

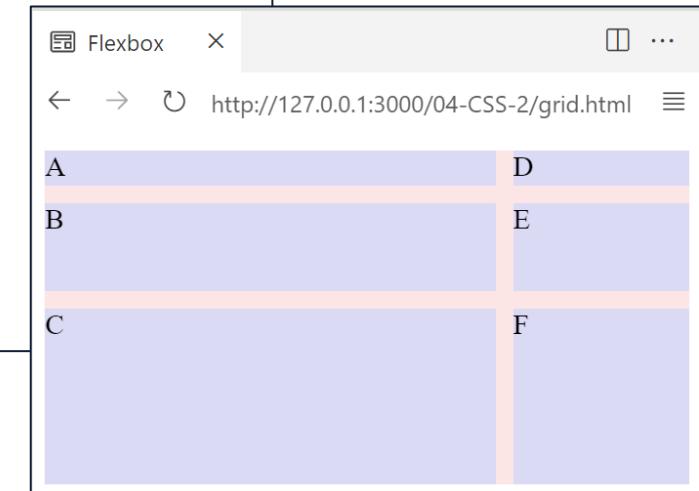
- The **fr** unit is a specialized relative unit that works only in grid layouts
- It represents a flexible length corresponding to a share of the available space
- It works similarly to the **flex** property
- For example, **grid-template-columns: 1fr 1fr 1fr;** defines three columns which all get the same share of the available space.

GRID: PLACEMENT OF GRID–ITEMS

- By default, grid items are placed along the rows
- It is possible to place items along columns using the **grid-auto-flow: column** property

```
.container {  
  display: grid;  
  /* two columns */  
  grid-template-columns: 1fr 100px;  
  /* three rows */  
  grid-template-rows: 20px 50px 100px;  
  grid-auto-flow: column;  
  gap: 10px;  
}
```

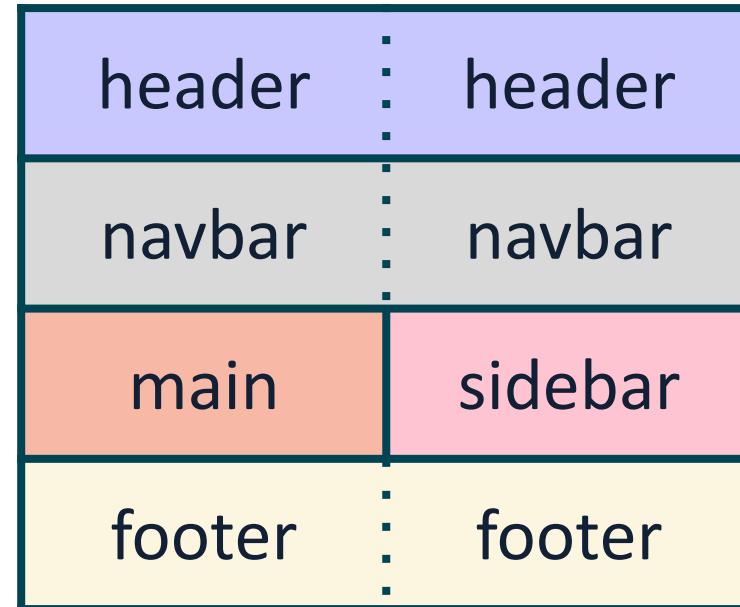
```
<div class="container">  
  <div>A</div>  
  <div>B</div>  
  <div>C</div>  
  <div>D</div>  
  <div>E</div>  
  <div>F</div>  
</div>
```



GRID: TEMPLATE AREAS

- It is also possible to assign a name to **areas** (set of cells) of the grid
- And to place grid items in a given area using the **grid-area** property

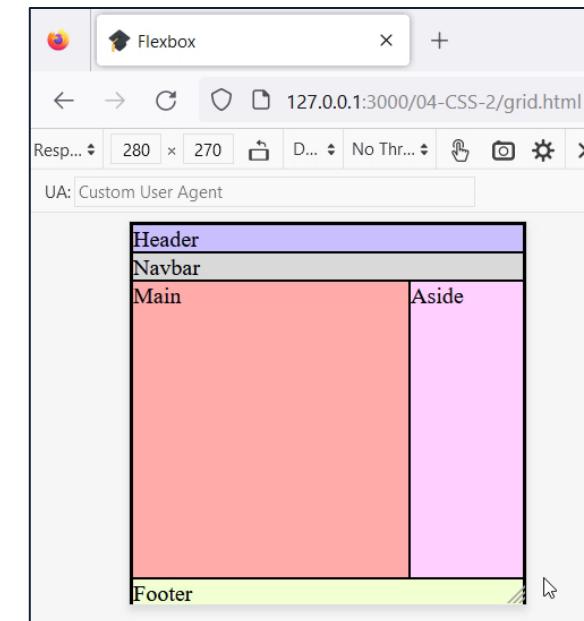
```
display: grid;  
grid-template-columns: 70vw 1fr;  
grid-template-rows: auto auto 1fr auto;  
grid-template-areas:  
  "header header"  
  "navbar navbar"  
  "main sidebar"  
  "footer footer";  
height: 100vh;  
margin: 0;
```



GRID: USING TEMPLATE AREAS

```
* {border: 1px solid black;}  
body {  
    display: grid;  
    grid-template-columns: 70vw 1fr;  
    grid-template-rows: auto auto 1fr auto;  
    grid-template-areas:  
        "header header"  
        "navbar navbar"  
        "main sidebar"  
        "footer footer";  
    height: 100vh; margin: 0;  
} /* background colors omitted */  
header { grid-area: header; }  
nav { grid-area: navbar; }  
main { grid-area: main; }  
aside { grid-area: sidebar; }  
footer { grid-area: footer; }
```

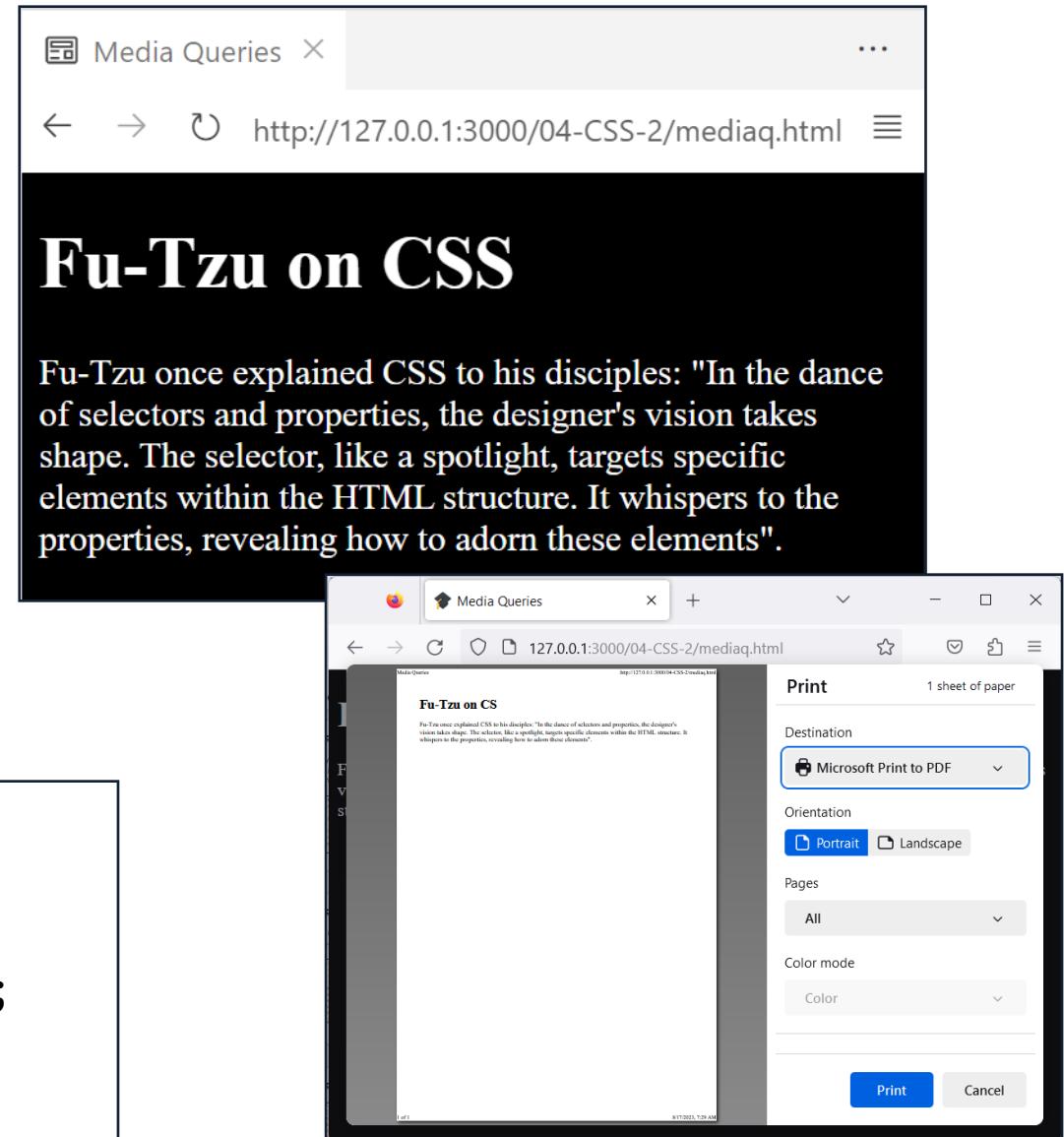
```
<body>  
    <header>Header</header>  
    <nav>Navbar</nav>  
    <main>Main</main><aside>Aside</aside>  
    <footer>Footer</footer>  
</body>
```



MEDIA QUERIES

- Media queries can apply certain CSS styles only when the device which is viewing the content has **specific characteristics**
- Media queries are initiated with the **@media** keyword

```
body {color: white; background: black;}  
@media print {  
    body {  
        color: black; background: transparent;  
    }  
}
```



MEDIA QUERIES: TYPES OF OUTPUT

Modern CSS supports three output (media) types in media queries:

- **print** intended for paged materials and documents viewed on a screen in print preview mode
- **screen** intended for devices that visualize the document on a screen
- **all** applies to all output devices

MEDIA QUERIES: MEDIA FEATURES

- Media queries can also test for specific characteristics of the user-agent, output device, or environment.
 - These are called «Media Features» ([reference here](#))
 - The full syntax of a media query is as follows

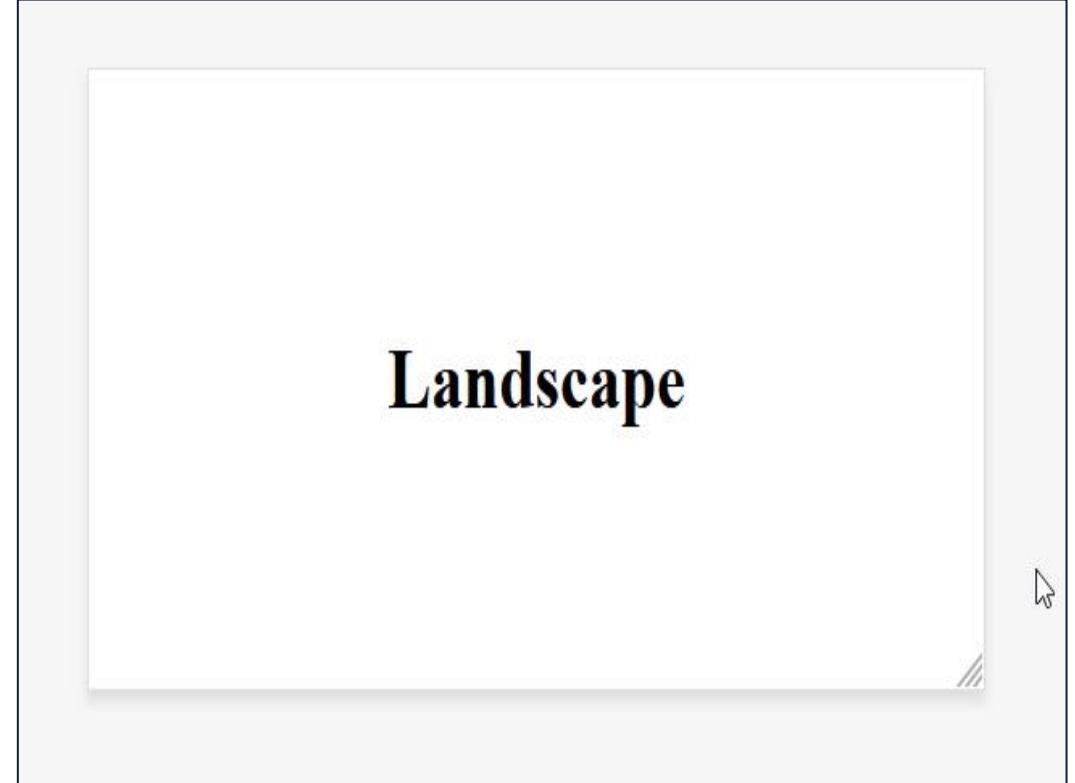
`@media media_type and (media_feature: value) and ...`

- If `media_type` is omitted, `all` is used by default
- Media features need to be enclosed in parentheses, and are optional

MEDIA QUERIES: EXAMPLES

```
@media screen and (orientation: landscape) {  
    h1::after {  
        content: "Landscape";  
    }  
}  
  
@media screen and (orientation: portrait) {  
    h1::after {  
        content: "Portrait";  
    }  
}
```

```
<body><h1></h1></body>
```



MEDIA QUERIES: EXAMPLES

```
@media (max-width: 600px) {  
    h1::after {content: "XSmall";}  
}  
  
@media (min-width: 600px) {  
    h1::after {content: "Small";}  
}  
  
@media (min-width: 768px) {  
    h1::after {content: "Medium";}  
}
```

```
@media (min-width: 992px) {  
    h1::after {content: "Large";}  
}  
  
@media (min-width: 1200px) {  
    h1::after {content: "XLarge";}  
}
```

```
<body><h1></h1></body>
```

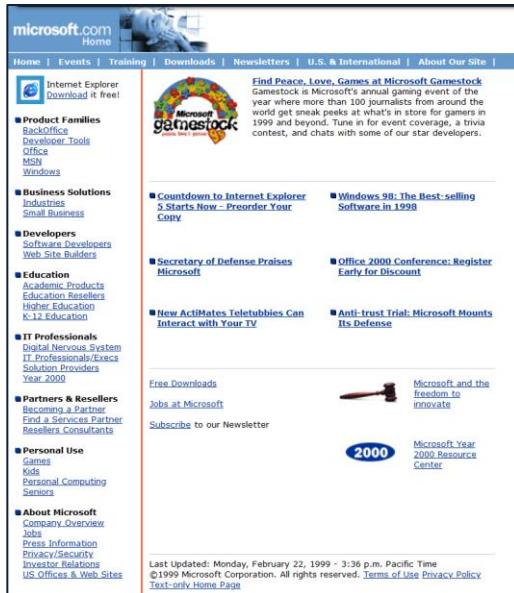


XSmall

RESPONSIVE DESIGN

FIXED-WIDTH LAYOUTS

- In the late 1990s, most monitors were 640px wide and 480px tall
- In the early days of web design, it was a safe bet to design pages with a 640px width.



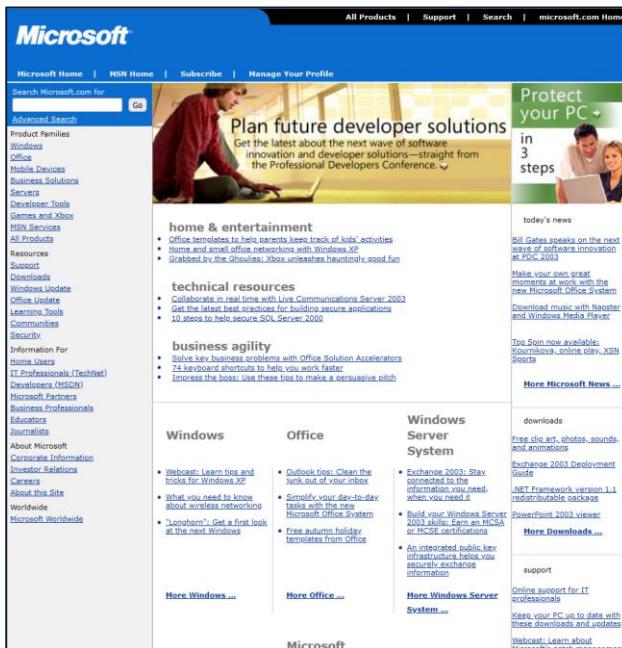
The Microsoft website,
from 1999 (640px wide)



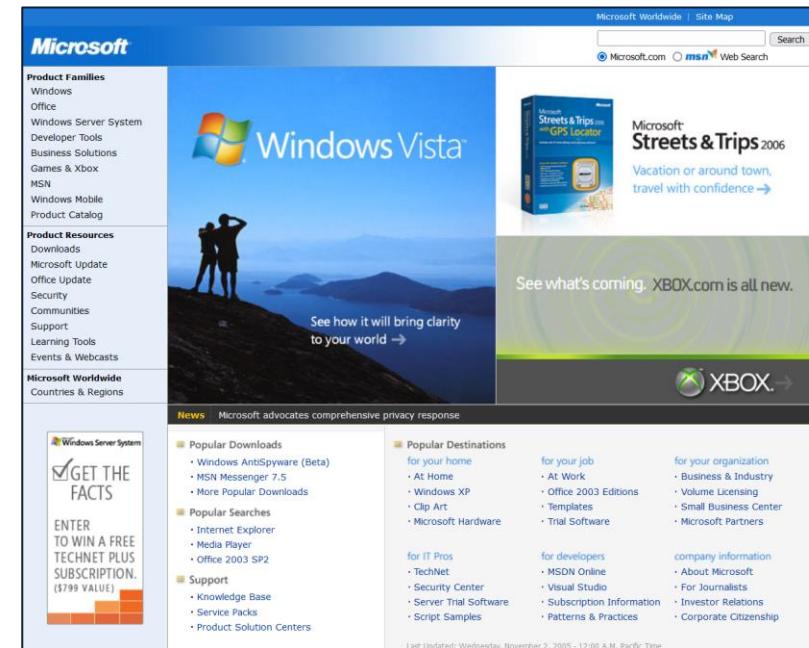
Illustration: Freepik.com

FIXED-WIDTH LAYOUTS

- Then the monitors started getting bigger...
- Most monitors became 800x600, then 1024x768, ...
- Web designers updated their fixed-width design accordingly



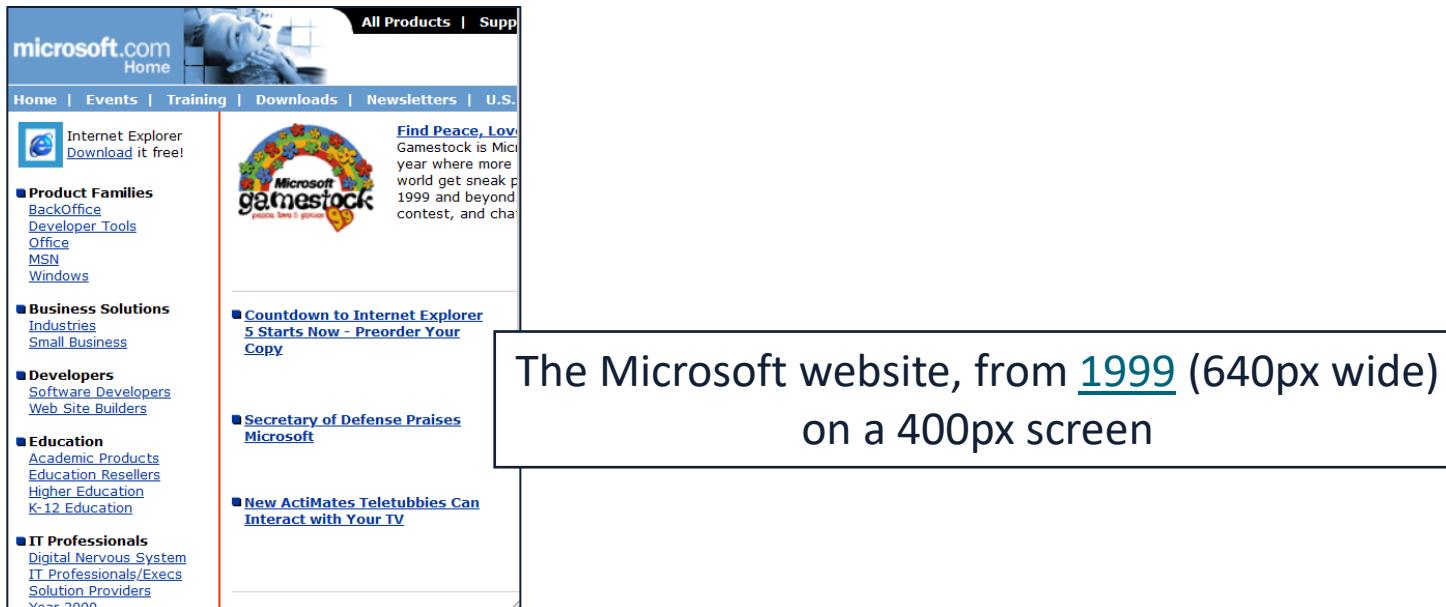
The Microsoft website, from 2003 (800px)



The Microsoft website, from 2005 (1024px)

FIXED-WIDTH LAYOUTS

- Whether it is 640, 800, or 1024 pixels, working with a single specific width is called **fixed-width** design
- Fixed-width design will look good only at the considered width.
 - Smaller screens won't be able to see the entire page without horizontal scrolling



FIXED-WIDTH LAYOUTS

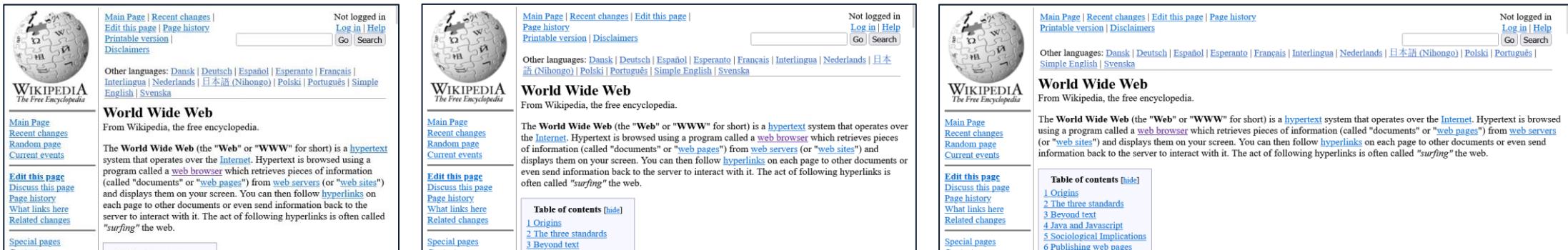
- Whether it is 640, 800, or 1024 pixels, working with a single specific width is called **fixed-width** design
- Fixed-width design will look good only at the considered width.
 - Smaller screens won't be able to see the entire page without horizontal scrolling
 - Larger screens will have a lot of (wasted) space



The Microsoft website, from 1999 (640px wide)
on a QHD 2560px screen

LIQUID (FLUID) LAYOUTS

- While most web designers used fixed-width layouts, some made their layouts **more flexible**
- There was no flexbox, grid or media queries. They did that by using percentages as column widths. The layouts are called **liquid layouts**.
- The Wikipedia website did this



The Wikipedia website from 2004, viewed in a 640px, 800px, and 1024px screens

LIQUID (FLUID) LAYOUTS

- Liquid layouts looks good across a wide range of widths...
- ... but begin to **worsen** at the **extremes**:
 - On a very wide screen, the layout will look **stretched**
 - On a very narrow one, the layout will look **squashed**



The screenshot shows the Wikipedia homepage from 2004. The page has a fluid layout. At the top, there's a navigation bar with links like "Main Page", "Recent changes", "Edit this page", and "Page history". Below the navigation, there's a large section titled "World Wide Web" with a sub-section about its history and how it works. The sidebar on the left contains links for "Main Page", "Recent changes", "Random page", "Current events", "Edit this page", "Discuss this page", "Page history", "What links here", "Related changes", "Special pages", and "Contact us". The main content area is quite wide, showing the full title and some introductory text.

[Edit this page](#)
[Discuss this page](#)
[Page history](#)
[What links here](#)
[Related changes](#)

[Special pages](#)
[Contact us](#)
[Donations](#)

World Wide Web
From Wikipedia,
the free encyclopedia

The World Wide Web (the "Web" or "WWW" for short) is a [hypertext](#) system that operates over the [Internet](#).

Hypertext is browsed using a program called a [web browser](#) which retrieves pieces of information (called "documents" or "web pages") from [web servers](#) (or "web sites") and displays them on your screen. You can then follow [hyperlinks](#) on each page to other documents or even send information back to the server to interact with it. The act of following hyperlinks is often called "surfing" the web.

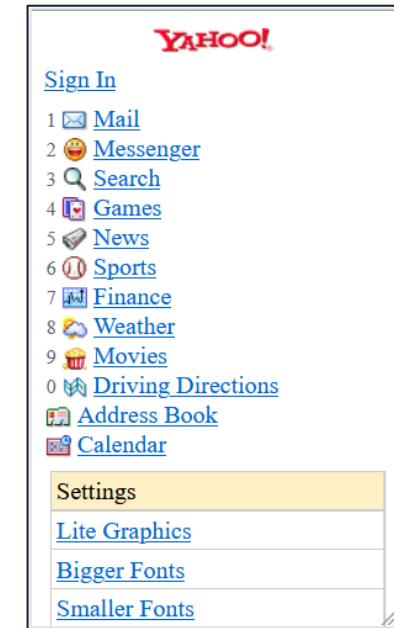
The Wikipedia website from [2004](#), viewed in a 2560px (left) and in a 250px (right) screens

SEPARATE WEBSITES

- Then, mobile devices arrived, with screens as small as 240px
 - Fixed-width and liquid layouts do not work well on those
- One option is to have a **separate website** for mobile devices



[yahoo.com, 2006](http://yahoo.com), on a 800px screen



[wap.oa.yahoo.com, 2006](http://wap.oa.yahoo.com), on a 240px screen

SEPARATE WEBSITES

- Typically leveraged user agent sniffing to redirect mobile users to the mobile version of the website, which was hosted in a subdomain.
- **Cons:**
 - UA sniffing is unreliable (might fail to redirect mobile users, and might wrongly redirect desktop users)
 - Each separate website needs to be maintained
 - Today, the distinction between mobile and non-mobile is blurred
- It would be nice to have a single website that renders differently depending on the characteristics of the viewing device...

ADAPTIVE LAYOUTS

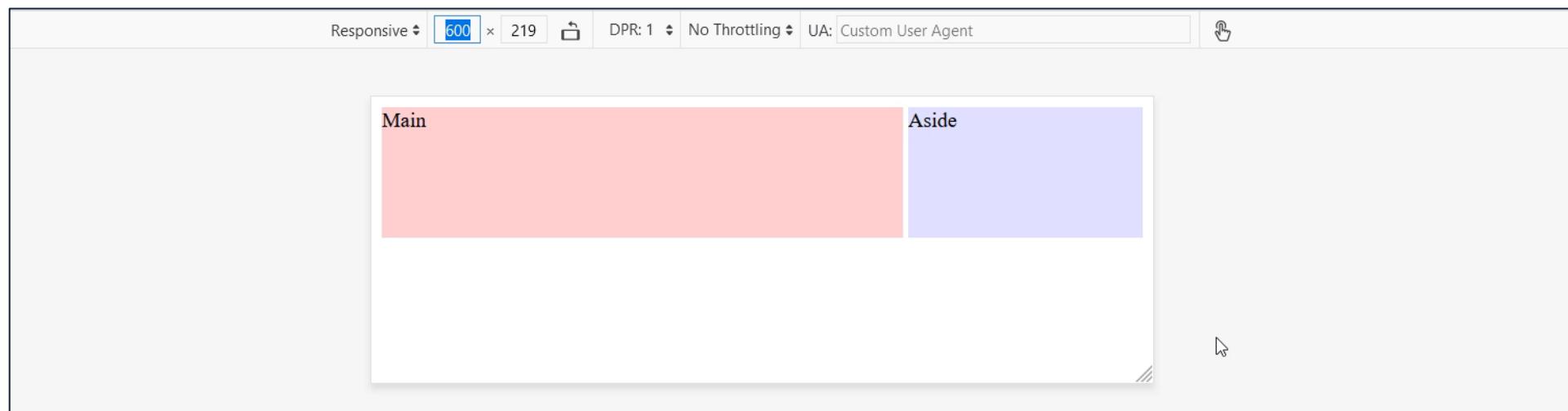
- With the introduction of CSS media queries (~2009), more flexible layouts were possible
- Initially, web designers were still most comfortable with fixed-widths layouts
- They designed websites that switched between a handful of fixed-widths designs using media queries.
- This approach is called **adaptive design**

ADAPTIVE LAYOUT: EXAMPLE

```
@media (max-width: 800px) {  
    main {width: 400px;}  
    aside {width: 180px;}  
}  
  
@media (min-width: 800px) {  
    main {width: 500px;}  
    aside {width: 280px;}  
}
```

```
main {  
    display: inline-block;  
    height: 200px;  
}  
  
aside {  
    display: inline-block;  
    height: 200px;  
}
```

```
<body>  
<main>Main</main>  
<aside>Aside</aside>  
</body>
```



ADAPTIVE LAYOUTS

- Allow to style a single website that looks good at few different sizes
- Still, the design does not look quite right for any size in between the considered ones
- Users are shown the layout that is closest to the size of their browser.
- Due to the variety of device sizes, chances are most users will experience a layout that looks less than optimal.

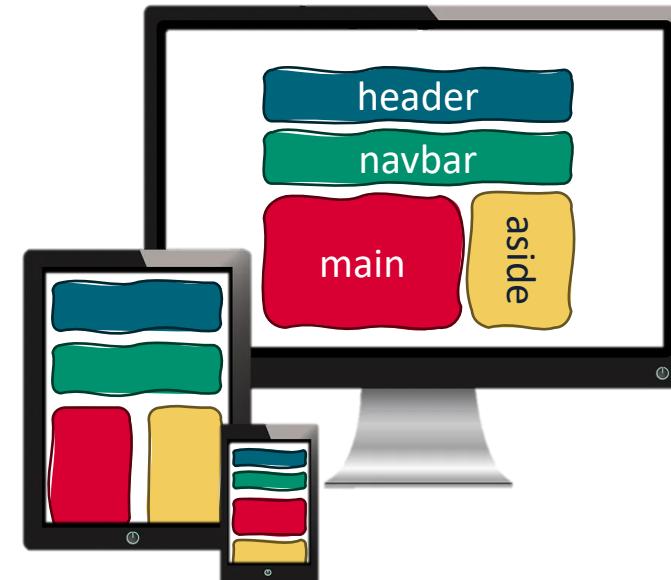
RESPONSIVE LAYOUTS

- Adaptive layouts are a mashup of media queries + fixed-width layouts
- **Responsive layouts are a mashup of media queries + liquid layouts**
- The term was coined by [Ethan Marcotte in an article](#) in 2010

Responsive design is characterized by:

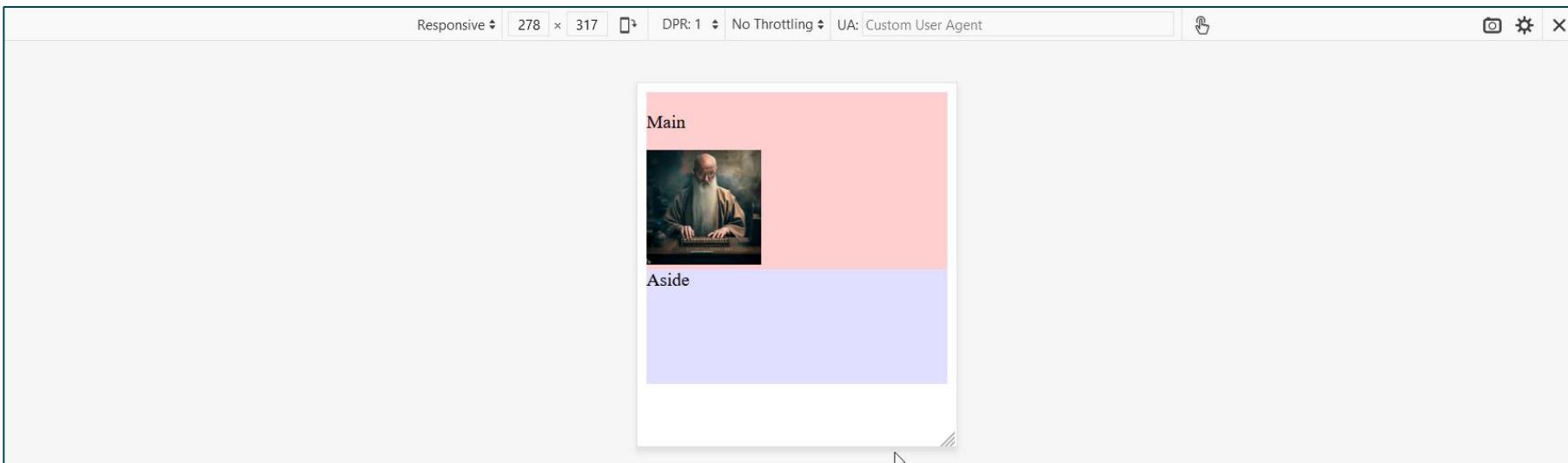
- Fluid containers
- Fluid media
- Media Queries

Layout and images of a responsive site should look good on **any** device



RESPONSIVE LAYOUT: EXAMPLE

```
body { display: flex; justify-content: center; flex-wrap: wrap; }
@media (max-width: 600px){ main {width: 100%} aside {width: 100%}}
@media (min-width: 600px){ main {width: 60%} aside {width: 40%}}
@media (min-width: 768px){ main {width: 70%} aside {width: 20%}}
main {display: inline-block;}
aside {display: inline-block; min-height: 100px;}
img {width: 20%; min-width: 100px;}
```



RESPONSIVE LAYOUT: VIEWPORT META TAG

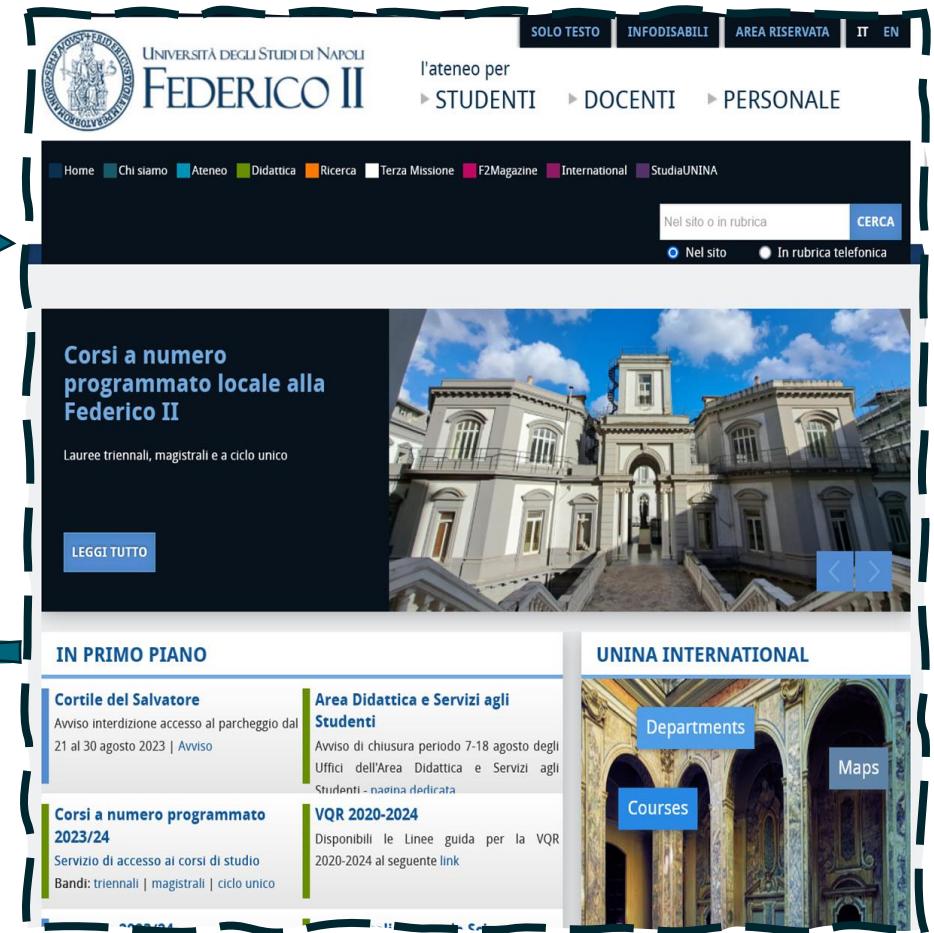
The first mobile browsers had to deal with websites that were designed for screens much wider than their own

- They assumed that websites were designed for a **980px** wide screen
- They rendered the web pages in a 980px **virtual viewport**
- Then, they **scaled** the rendered page down to fit the actual screen width

RESPONSIVE LAYOUT: VIEWPORT META TAG



1: Render in virtual viewport



The screenshot shows the homepage of the University of Naples Federico II. At the top, there is a navigation bar with links for SOLO TESTO, INFODISABILI, AREA RISERVATA, IT, and EN. Below the navigation is the university's logo and name. A main banner features a large image of a classical building. To the left of the banner, there is a section titled "Corsi a numero programmato locale alla Federico II" with a "LEGGI TUTTO" button. Below this is a "IN PRIMO PIANO" section containing news items about parking restrictions and VQR 2020-2024. To the right of the banner is a "UNINA INTERNATIONAL" section with links for Departments, Courses, and Maps.

2: Scale down to device

BlackBerry Torch 9800 (2010)
480x360 display

980px-wide Virtual Viewport

RESPONSIVE LAYOUT: VIEWPORT META TAG

- The virtual viewport mechanism allowed non-mobile-optimized websites to look better on narrow screens
- However, the mechanism doesn't work for websites that **are** optimized for mobile devices
 - The media queries that kick in at 640px will never be used at 980px!
- The **viewport** HTML **meta tag** allows to control the virtual viewport mechanism
- Mobile-optimized web pages should include in their **<head>**:

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

RESPONSIVE LAYOUT: VIEWPORT META TAG

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

The above viewport HTML meta tag specifies two rules:

- **width=device-width** tells the browser to assume that the width the website was designed for is the width of the device
- **initial-scale=1** tells the browser to do no scaling at all.

RECAP: A BRIEF HISTORY OF WEB LAYOUTS

FIXED-WIDTH LAYOUTS

- In the late 1990s, most monitors were 640px wide and 480px tall
- In the early days of web design, it was a safe bet to design pages with a 640px width.



The Microsoft website, from 1999 (640px wide)



LIQUID (FLUID) LAYOUTS

- While most web designers used fixed-width layouts, some made their layouts **more flexible**
- There was no flexbox, grid or media queries. They did that by using percentages as column widths. The layouts are called **liquid layouts**.
- The Wikipedia website did this



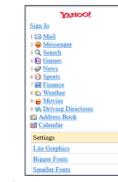
The Wikipedia website from 2004, viewed in a 640px, 800px, and 1024px screens

SEPARATE WEBSITES

- Then, mobile devices arrived, with screens as small as 240px
 - Fixed-width and liquid layouts do not work well on those
- One option is to have a **separate website** for mobile devices



yahoo.com, 2006, on a 800px screen



wap.oo.yahoo.com, 2006, on a 240px screen

50

Luigi Libero Lucio Starace, Ph.D. - University of Naples Federico II - Web Technologies Course - Lecture 04 - CSS Layouts

Luigi Libero Lucio Starace, Ph.D. - University of Naples Federico II - Web Technologies Course - Lecture 04 - CSS Layouts

ADAPTIVE LAYOUTS

- With the introduction of CSS media queries (~2009), more flexible layouts were possible
- Initially, web designers were still most comfortable with fixed-widths layouts
- They designed websites that switched between a handful of fixed-widths designs using media queries.
- This approach is called **adaptive design**



Luigi Libero Lucio Starace, Ph.D. - University of Naples Federico II - Web Technologies Course - Lecture 04 - CSS Layouts

52

RESPONSIVE LAYOUTS

- Adaptive layouts are a mashup of media queries + fixed-width layouts
- Responsive layouts are a mashup of media queries + liquid layouts**
- The term was coined by [Ethan Marcotte](#) in an article in 2010

Responsive design is characterized by:

- Fluid containers
- Fluid media
- Media Queries

Layout and images of a responsive site should look good on **any** device



Luigi Libero Lucio Starace, Ph.D. - University of Naples Federico II - Web Technologies Course - Lecture 04 - CSS Layouts

55

REFERENCES

- **Learn CSS**
web.dev
<https://web.dev/learn/css/>
Sections: 2, 8 to 11
- **Learn Responsive Design**
web.dev
<https://web.dev/learn/design/>
Sections: 1 to 3, 15
- **Game-based approaches to learning CSS Layouts**
<https://flexboxfroggy.com/> (Flexbox)
<https://cssgridgarden.com/> (Grid layouts)

