The Little Book of Web Development

Feng

Contents

Preface

This book is intended for newcomers and front-end developers looking to venture into back-end development, as well as back-end developers aiming to explore front-end development.

It aims to provide a broad overview of modern web development without delving too deeply into details.

You can read the book consecutively or choose individual topics that interest you.

There are numerous code examples, most of which are runnable. You are encouraged to engage with these examples hands-on, but it's also fine to skim through them.

Feng (@codemann) is a professional web developer with a passion for exploration and creation.

Preparation

This book assumes that you are using macOS. If you are using Linux or Windows, you will need to figure out how to install the required softwares by yourself, like VS Code and Docker.

Homebrew

On macOS, we use Homebrew to manage packages.

To install Homebrew itself, open Terminal (click on the Spotlight icon, type ”terminal” and press Enter), paste the following command, and press Enter:

```
/bin/bash -c \
  "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

VS Code

We will use Visual Studio Code (VS Code) as our IDE. Its support for various languages, tools, and frameworks makes it a top choice for developers of all levels.

To install VS Code, run the following command:

```
brew install --cask visual-studio-code
```

Part I

Fundamentals

Chapter 1

Web Development

Web development refers to the processes involved in building websites and web applications. It encompasses both front-end (user interface) and back-end (server-side business logic and IO) development.

Front-End Development

- **HTML (HyperText Markup Language):** The foundation of any web page, HTML provides the structure and content, defining elements such as headings, paragraphs, images, and links.
- **CSS (Cascading Style Sheets):** CSS styles HTML elements, controlling the layout, colors, fonts, and overall appearance of a website.
- **JavaScript:** This programming language adds interactivity to websites, enabling animations, user input handling, and dynamic content changes.

Back-End Development

- **Servers:** Back-end development involves setting up and managing servers. With the emergence of serverless technology, many developers no longer need to handle this aspect themselves.
- **Databases:** Databases store and manage the website's data, using tools like MySQL, PostgreSQL, and MongoDB. Back-end developers use programming languages such as Python, Ruby, or Java to interact with databases and perform data retrieval and storage.
- **Programming Languages:** Popular languages includes JavaScript, Go, Ruby, Python, PHP and Java.
- **Frameworks:** Back-end development often leverages frameworks to streamline the development process. Examples include Django and Flask for Python, Ruby on Rails for Ruby, and Spring for Java.

The Process of Web Development

1. Define goals and target audience.
2. Wireframing.
3. Visual design.
4. Frontend and backend development.
5. Test for browser compatibility, functionality, performance, and security.
6. Deployment.
7. Monitoring and maintenance.

1.1 A Brief History of Web Development

Initially, HTML was used primarily for static documents, and the web was text-only. Around 1993, graphical web browsers emerged, allowing multimedia content to be combined with text on the same page.

The Common Gateway Interface (CGI) was introduced, and server-side scripting languages like Perl and PHP gained popularity. With the launch of HTML 2.0, the `<form>` element enabled users to submit data to the backend, making the web into more dynamic.

In 1995, JavaScript was introduced, adding interactivity to web pages. A year later, CSS was created to enhance the presentation layer of web content.

By that time, hyperlinks and form submissions were the primary mechanisms for interacting with the server, often replacing the current page with another one. The advent of Ajax empowered JavaScript to perform asynchronous network operations, allowing applications to request data or HTML from the backend without refreshing the entire page. This methodology, still in use today, can be seen in libraries like pjax and htmx.

In 2006 jQuery was born, simplifying DOM manipulation, with ideas that later influenced the standard DOM API.

During the early 2000s, Flash became the primary technology for creating rich, interactive web content, including animations, games, and video playback. HTML5 was drafted as a response to Flash, enabling web applications to become more capable.

As web application be come more complex, various architectural paradigms emerged for both client-side and backend applications. On the backend, notable architectures and patterns included MVC and IoC. On the frontend, frameworks like Knockout.js (MVVM), Angular (MVVM), and Backbone (MVC) facilitated the development of large-scale, dynamic client applications.

In Year 2013, React was introduced, marking a new era in web development and leading to its current prominence.

Chapter 2

The Command Line

The command line, is a text-based interface for interacting with your computer.

Build tools, version control systems and package managers (like Gradle git and npm) are primarily driven through the command line.

While graphical interfaces simplify certain tasks, a solid command-line foundation is essential for web developers to be efficient, effective, and adaptable in their work.

Basic Commands

To open Terminal, click on the Spotlight icon (magnifying glass) in your menu bar. Type ”Terminal” and press Enter.

pwd: Where am I

Type pwd and press Enter will print current working directory.

ls: List files and directories

- To list files under working directory: `ls`
- To list hidden files: `ls -a`

cd: Change directory

- To move to the Desktop: `cd ~/Desktop`
- To move to your home directory: `cd ~`
- To move to the parent directory: `cd ..`
- To move back to the previous directory: `cd -`

mkdir: Create a new directory

- To create a directory named ”projects”: `mkdir projects`

rm: Remove a file

Be careful with rm, as it permanently deletes files.

- To remove a file: `rm myfile.txt`
- To remove a empty directory: `rmdir mydir`

- To remove a non-empty directory: `rm -rf mydir`

mv: Move or rename a file or directory

- To move a file named ”file.txt” to the ”Documents” directory: `mv file.txt ~/Documents/`
- To rename a file named ”old.txt” to ”new.txt”: `mv old.txt new.txt`

cp: Copy a file or directory

To backup a file named ”file.txt”: `cp file.txt backup.txt`

Tips and Tricks

Don't type a long command by hand, use `Ctrl + R` to search in the history.

There are also shortcuts for editing:

- `Ctrl + A`: Moves the cursor to the beginning of the line.
- `Ctrl + E`: Moves the cursor to the end of the line.
- `Ctrl + L`: Clears the screen.

Stop the current command

Use `Ctrl + C` to interrupt the current command.

Sometimes you might need `Ctrl + D` to exit.

Running Multiple Tasks

Some commands are long-running tasks, such as a web server. To run another command, you typically need to open a new terminal.

Alternatively, you can use `Ctrl + Z` to suspend the current command. After executing the new command, you can use `fg` to resume the suspended command.

If you want to run multiple long-running tasks, you can use `bg` to send the suspended command to the background. Once it's finished, use `kill` to terminate it.

```
$ bun server.ts
[1] + 55407 suspended  bun server.ts
$ bg %1
[1] + 55407 continued  bun server.ts
$ kill %1
[1] + 55407 terminated  bun server.ts
```

To start a command in the background, simply append an `&` to the command, like this:

```
bun server.ts &
```

Another more intuitive solution is to use a terminal multiplexer, such as tmux or `screen`.

Chapter 3

HTML

HTML, or HyperText Markup Language, is the fundamental language used to create and structure web pages. It provides the basic building blocks for a webpage, such as headings, paragraphs, links, images, and other content. Originally designed for organizing and presenting documents, HTML has evolved and is now extensively used in web development to create both simple and complex applications. It works in conjunction with CSS (Cascading Style Sheets) for styling and JavaScript for interactivity, enabling developers to build rich, interactive web experiences.

Your First Web Page

Open Terminal, create a folder and start up VS code:

```
mkdir html
cd html
code .
```

In VS Code Press `Command+Shift+X` to enter Extension view and install the ”Live Server” extension.

Then Press `Command+Shift+E` to open File Explorer and create a new html file `index.html`, paste the folowing code into the file:

```
<!DOCTYPE html>
<html>
<head>
  <title>My Web Page</title>
</head>
<body>
  <h1>h for header</h1>
  <p>p for paragraph</p>
</body>
</html>
```

Then right-click on your HTML file and click ”Open with Live Server”. It will start a local server and automatically open your default browser.

VS Code is highly recommand here, but there are many other options to serve static content, such as using Python's built-in HTTP server. Simply run the following command to start the server in the current directory:

```
python3 -m http.server 8000
```

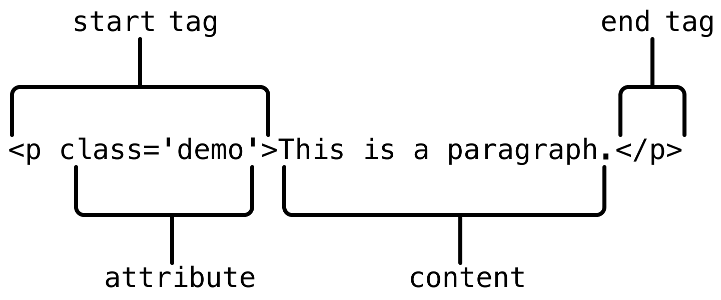
Then, visit http://localhost:8000 to view the webpage.

HTML Elments

Let's take a closer look at the code:

- `<!DOCTYPE html>`: Declares the document type as HTML5(don't bother with previous versions).
- `<html>`: The root element of an HTML document, which has a tree like structure.
- `<head>`: Contains metadata about the webpage, such as the title and stylesheets.
- `<title>`: Sets the title of the webpage, displayed in the browser's tab.
- `<body>`: Contains the visible content of the webpage.
- `<h1>`: Defines the most important heading on the page, there are also h2, h3 till h6.
- `<p>`: Defines a paragraph of text.

Basic Structure of an Element



An element consists of a start tag, content, and an end tag. The start tag specifies the element's name and contains attributes. The end tag contains a `/` before the element name. The content can be text or other elements.

Some elements are self-closing and do not require an end tag or content:

```
<img src="image.jpg" alt="A beautiful image">
```

`<div>`

The `<div>` tag, short for ”division”, does not convey any semantic meaning about the content it wraps; it is essentially a ”block-level” element used to group other elements together for styling with CSS or for scripting with JavaScript. It is highly versatile and is frequently used to create layout structures or sections within a webpage. However, for better accessibility and SEO, it's recommended to use semantic HTML tags (like `<header>`, `<footer>`, `<article>`, and `<section>`) where appropriate, as they provide more meaningful context about the content.

3.1 Accessibility

Web accessibility ensures that websites and web applications are usable by people with disabilities. It involves creating content that is accessible to everyone, regardless of their abilities.

To enhance accessibility:

1. **Use Semantic HTML**: Utilize appropriate HTML elements (e.g., `<header>`, `<nav>`, `<main>`, `<footer>`) to convey the content structure.
2. **Provide Descriptive Alt Text**: Include meaningful alt text for images to communicate their purpose to visually impaired users.
3. **Implement ARIA Attributes**: Use Accessible Rich Internet Applications (ARIA) attributes to convey additional context to assistive technologies like screen readers.

Example:

```html
<button aria-label="Play">
  <img src="play-button.png">
</button>
```

In this example, the `aria-label="Play"` provides a text label for the button, which assists screen reader users.

Other Common ARIA Attributes:

- `role="button"`: Indicates that an element should be treated as a button.
- `aria-disabled="true"`: Signals that an element is disabled.
- `aria-expanded="true"`: Indicates that an element is expanded (e.g., for a collapsible section).
- `aria-selected="true"`: Shows that an item is selected (e.g., within a list).

3.2 SVG

SVG, or Scalable Vector Graphics, is an XML-based format for creating two-dimensional vector graphics. Unlike raster images, SVGs maintain high quality at any size, making them ideal for web design and responsive layouts.

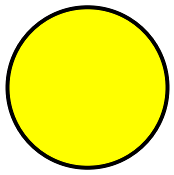
With SVG, you can create shapes, paths, and text that are easily manipulated through CSS and JavaScript, allowing for dynamic and interactive graphics. It's widely supported across modern web browsers, making it a powerful tool for developers and designers looking to enhance visual content on the web.

Syntax

While complex graphics are usually created using design tools like Illustrator and Inkscape, it's possible to write SVG by hand, and it can be embedded directly in HTML:

shape

```html
<svg width="100" height="100">
  <circle cx="50" cy="50" r="40" stroke="black" stroke-width="2" fill="yellow" />
</svg>
```

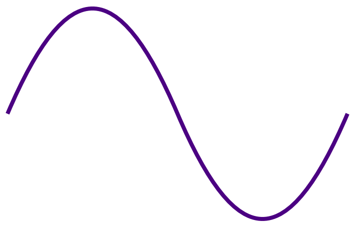


text

```html
<svg width="200" height="100">
  <rect width="200" height="100" fill="indigo" />
  <text x="100" y="75"
    font-family="Verdana"
    font-size="50"
    fill="white"
    text-anchor="middle">SVG</text>
</svg>
```

path

```
<svg width="200" height="150">
  <path d="M 10 80 C 40 10, 65 10, 95 80 S 150 150, 180 80"
        stroke="indigo" fill="transparent" stroke-width="2" />
</svg>
```



The `<path>` element is a versatile component used to create complex shapes and lines. It allows you to define a shape by specifying a series of commands and parameters in a single attribute. Here is a breakdown:

- M (move to): Moves the starting point to a specified coordinate without drawing a line.
- C (cubic Bézier curve): Draws a cubic Bézier curve from the current point to a specified point, using two control points.
- S (smooth cubic Bézier curve): Similar to C, but the first control point is inferred from the previous curve.

SVG for Application Development

SVG can be generated and manipulated programmatically, making it suitable for creating data visualizations, charts, and other dynamic graphics based on data inputs. Libraries such as D3.js and SVG.js simplify SVG manipulation, event handling, and data visualization.

Chapter 4

CSS

Cascading Style Sheets (CSS) is a language used to control the presentation and layout of HTML documents. While HTML provides the structure and content of a webpage, CSS is responsible for the visual appearance.

Inline Styles

Inline styles are defined directly on an HTML element using the style attribute. They override any external or internal CSS.

Let modify our web page:

```
<!DOCTYPE html>
<html>
<head>
    <title>My Web Page</title>
</head>
<body>
    <h1>Hello, World!</h1>
    <p style="color: white; background: blue;">This is a paragraph of text.</p>
</body>
</html>
```

Inline CSS lives in the `style` attribute in the form of `properties: values;`.

Properties determine the style attributes of the element, like color, font-size, background, etc.

Values specify the desired value for each property, such as `red`, `16px`, or `url("image.jpg")`.

Mixing styles with content looks messy? And, what if you have multiple paragraphs and want to style them consistently?

External Stylesheets

CSS can be separated from HTML. Create a separate file `styles.css` to house your styles. Link it to your HTML using the `<link>` tag:

```
<link rel="stylesheet" href="styles.css">
```

Now you need a way to specify which elements you want to style. That's where selectors come into play.

4.1 Selectors

CSS selectors are patterns used to select the elements.

Basics

Element Selector

Selects all instances of a specific HTML element. To style all paragraphs:

```
p {
  color: white;
  background: blue;
}
```

ID Selector

Selects a single element with a specific ID. Use a hash (#) before the ID name. An ID is unique, while a class is not. To avoid conflicts, IDs are rarely used.

```
#app {}
```

Class Selector

Selects elements with a specific class. Use a period `.` before the class name.

Class selectors are the most used ones.

```
.my-class {}
```

Attribute Selector

Selects elements based on their attributes. To select all text input:

```
input[type="text"] {}
```

Pseudo-class Selector

Pseudo-class selector starts with `:`.

state

`:hover`, `:focused`, `:disabled`, `:focus-within`

To select all anchors with mouse cursor on it:

```
a:hover {}
```

lang

`:lang`

```
<p lang="de"></p>
```

```
p:lang(de) {}
```

first and last

First child:

```
article *:first-child {}
```

First child and it is is a paragraph:

```
article p:first-child {}
```

First paragraph:

```
article p:first-of-type {}
```

Similary, this are `last-child` and `last-of-type`.

nth

```
p:nth-child(n) {}
p:nth-of-type(2n + 1) {}
p:nth-last-type(1) {}
```

`n`: 0 1 2 3 … `2n + 1`: 1 3 5 …

not

```
p:not(:first-child) {}
```

Pseudo Elements

Pseudo element selectors starts with double colons `::`:

```
p::first-letter {}
p::first-line {}

p::before {}
p::after {}

input::placeholder {}

dialog::backdrop {}

::selection {}
```

Combining Selectors

Selectors can also be combined in several ways:

Grouping Selectors

You can group multiple selectors that share the same styles by separating them with a comma.

```
h1, h2, h3 {
  color: green;
}
```

Escendant Selector

This selector targets elements that are nested within a specified parent element.

```
div p {
  color: blue; /* Applies to all <p> elements inside any <div> */
}
```

Child Selector

The child selector > selects elements that are direct children of a specified parent.

```
ul > li {
  list-style-type: square; /* Applies only to <li> that are direct children of <ul> */
}
```

Adjacent Sibling Selector

The adjacent sibling selector + selects an element that is immediately following another specified element.

```
h1 + p {
  margin-top: 0; /* Applies to the first <p> that comes directly after an <h1> */
}
```

Combining Class and Element Selectors

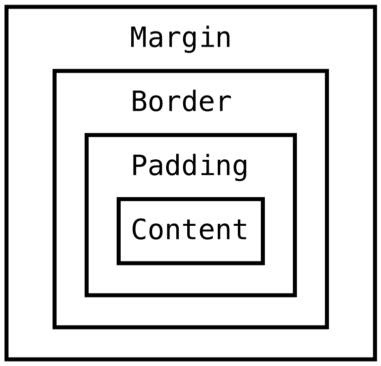
You can combine class selectors with element selectors to target specific elements with certain classes.

```
button.primary {
  background-color: blue;
}
```

4.2 The Box Model

The CSS box model describes the structure of elements. Each box consists of:

- Margin: Space outside the border that separates the element from others.
- Border: A line surrounding the padding.
- Padding: Space between the content and the border.
- Content: The actual content of the box (text, images).



```
.box {
  width: 100px;
  height: 100px;
  padding: 10px;
  border: 10px solid black;
  margin: 10px;
}
```

What does the width and height mean in above example?

box-sizing

The `box-sizing` property in CSS controls how the width and height of an element are calculated. There are two values:

`content-box`(default): Width and height only include the content, excluding padding and borders. This can lead to elements being larger than expected when adding padding or borders.

`border-box`: Width and height include content, padding, and borders. This makes layout more predictable, as you specify the total size without worrying about extra space from padding or borders.

Use the DevTools

Browsers come with built-in developer tools that help in debugging and optimizing websites. These tools can inspect HTML and CSS, analyze network requests, and test performance.

To work with the DOM or CSS, right-click an element on the page and select Inspect to jump into the Elements panel.

4.3 Shorthand Properties

CSS shorthand properties are a way to combine multiple CSS properties into a single declaration. This can make your CSS code more concise and easier to read.

Take margin as an example:

```
margin-top: 10px;
margin-right: 10px;
margin-bottom: 10px;
margin-left: 10px;
```

is the same as:

```
margin: 10px;
```

You can specify 1 to 4 values:

```
margin: [top and bottom] [right and left];
margin: [top] [right] [bottom] [left];
margin: [top] [right and left] [bottom];
```

There are many other shorthand properties available, such as padding, border, font, and background.

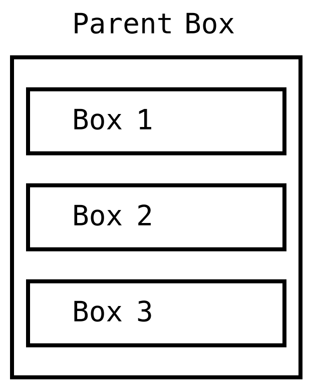
Positioning

CSS positioning allows you to control the layout of elements on the page.

- static: Default position; elements are positioned according to the normal flow.
- relative: Positioned relative to its normal position.
- absolute: Positioned relative to the nearest positioned ancestor.
- fixed: Positioned relative to the viewport; stays in place when scrolling.
- sticky: Toggles between relative and fixed, based on scroll position.

static

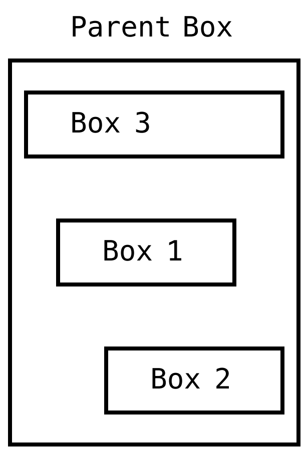
```html
<div class="parent">
  <div class="box1">Box 1</div>
  <div class="box2">Box 2</div>
  <div class="box3">Box 3</div>
</div>
```



absolute

```html
<div class="parent">
  <div class="box1">Box 1</div>
  <div class="box2">Box 2</div>
  <div class="box3">Box 3</div>
</div>
```

```css
.parent {
  position: relative;
  height: 100px;
}

.box1 {
  position: absolute;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
}

.box2 {
  position: absolute;
  bottom: 0;
  right: 0;
}
```



Parent Box is the containing element. It establishes the context for absolutely positioned elements.

Box 1 and Box 2 are positioned absolutely within the parent box.

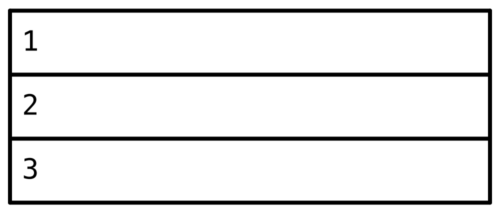
The top left corner of Box 1 is first posioned at the center of its Parent Box, then `translate` shifts it back by half of its own width and height, effectively centering it around that point.

Box 3 is the only `static` element, since `absolute` elements are taken out of the normal document flow, it goes to the top.

4.4 Flexbox

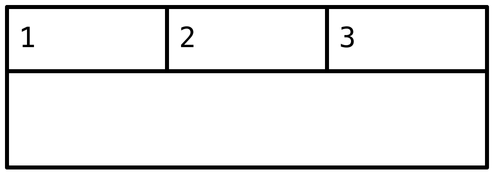
Flexbox is a layout model that allows you to design a one-dimensional layout easily.

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



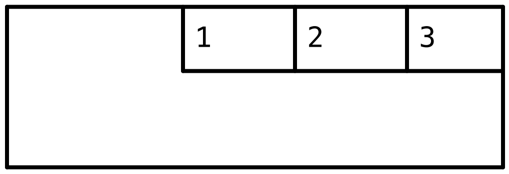
Flexbox is horizontal by default, use `flex-direction: column;` to layout vertically.

```css
.container {
  display: flex;
  height: 5em;
}
```

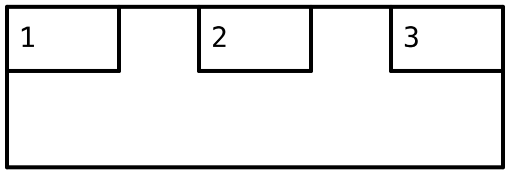


`justify-content` controls how the children is distributed on the main axis(horizontal):

```css
.container {
  display: flex;
  height: 5em;
  justify-content: flex-end;
}
```

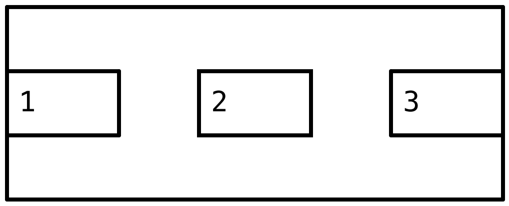


```css
.container {
  display: flex;
  height: 5em;
  justify-content: space-between;
}
```



`align-items` controls how the children is distributed on the cross axis(vertical):

```css
.container {
  display: flex;
  height: 5em;
  justify-content: space-between;
  align-items: center;
}
```



To explore the full power of flexbox, check out Flex Cheatsheet.

There is also a game to test your flexbox skills.

4.5 Grid Layout

CSS Grid Layout is a powerful two-dimensional layout system.

Like Flexbox, it consists of two parts: the container and the items.

You define a grid template on the container, then place the items onto the grid.

Defining the container

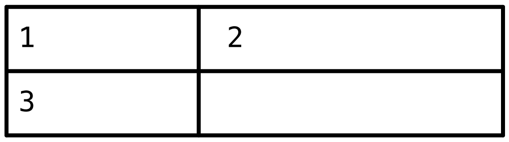
Given following HTML structure:

```html
<div class="container">
  <div class="item1">1</div>
  <div class="item2">2</div>
  <div class="item3">3</div>
</div>
```

Your can define the container's grid template with `grid-templates-rows` and `grid-template-columns`, or use `grid-template` to combine the two rules to be more concise.

```css
.container {
  display: grid;
  grid-template-columns: 1fr 2fr;
}
```

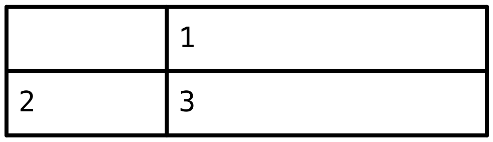
`grid-template-columns` defines the width of each column. `fr` stands for fraction, other units like `px` and `em` can also be used.



Placing the items

You can use `grid-column-start`, `grid-column-end` or `grid-column` to defined the position and size of the item in the grid.

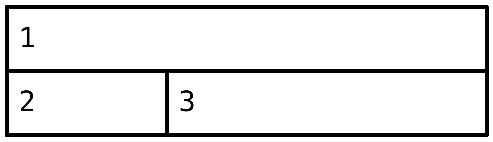
```css
.container {
  display: grid;
  grid-template-columns: 1fr 2fr;
}

.item1 {
  grid-column-start: 2;
}
```



```
.item1 {
  grid-column-end: 3;
}
```

```
.item1 {
  grid-column-end: span 2;
}
```

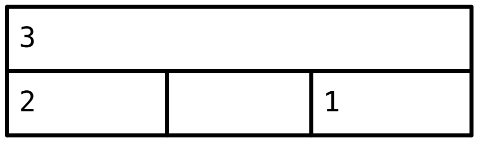
those two does the same thing:



Ascii art style grid template

`grid-template-areas` allows you to defined the template in an ascii art style.

```
.container {
  grid-template-areas:
    "a a a"
    "b . c";
}

.item1 {
  grid-area: c;
}

.item2 {
  grid-area: b;
}

.item3 {
  grid-area: c;
}
```

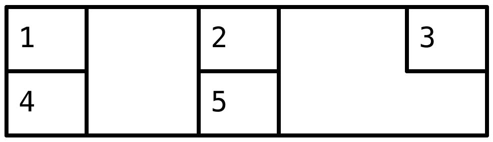


Alignment

Align columns

`justify-content` aligns columns horizontaly, similar to `justify-content` in a horizontal flexbox layout:

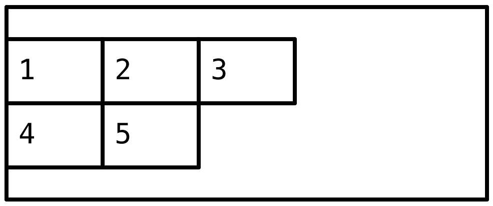
```
.container {
  display: grid;
  grid-template: repeat(2, 100px) / repeat(3, 100px);
  justify-content: space-between;
}
```



Align rows

`align-content` aligns rows verticaly, similar to `align-content` in a mutli-row horizontal flexbox layout:

```
.container {
  display: grid;
  grid-template-columns: repeat(3, 100px);
  height: 200px;
  align-content: center;
}
```



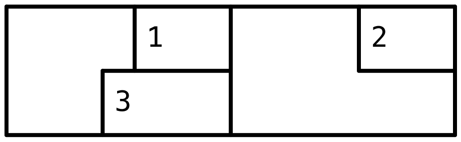
Align inside grid cells

Given a 2x2 grid:

```
.container {
  display: grid;
  grid-template: repeat(2, 1fr) / repeat(2, 1fr);
}
```

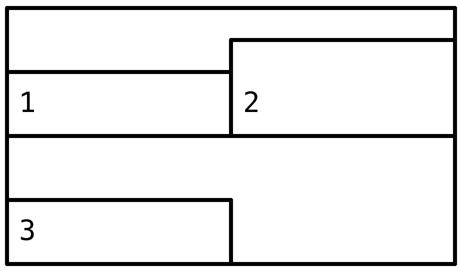
`justify-items` and `justify-self` aligns the content horizontaly, `justify-items` is applied on the container, `justify-self` on the children:

```
.item {
  justify-self: end;
}
```



Similarly `align-items` and `align-self` align the content verticaly:

```
.container {
  display: grid;
  grid-template: repeat(2, 100px) / repeat(2, 1fr);
  align-items: end;
}
```



Tips to remember thoses rules

There are a total of 6 rules (2 prefixes multiplied by 3 suffixes):

- **`justify-*`**: Used to align horizontally(main axis).
- **`align-*`**: Used to align vertically(cross axis).
- **`*-content`**: Applies to rows or columns.
- **`*-items`**: Applies to items within their cells.
- **`*-self`**: Applies to a single item within it's cell.

More

Play a game to master grid layout.

There is also a Grid Cheatsheet.

4.6 Transition

A transition allows you to change property values smoothly over a specified duration.

```css
button {
  background-color: gray;
  transition: background-color 0.3s ease-in-out;
}

button:hover {
  background-color: red;
}
```

Transform multiple property values:

```css
button {
  background-color: gray;
  transition: all 0.3s ease-in-out;
}

button:hover {
  background-color: red;
  transform: scale(1.5);
}
```

4.7 Animation

CSS animation allows you to create more complex sequences of transitions by defining keyframes. It can run continuously or a set number of times.

```css
@keyframes beat {
  0% { transform: rotate(-45deg) scale(1); }
  50% { transform: rotate(-45deg) scale(1.5); }
  100% { transform: rotate(-45deg) scale(1); }
}

.heart {
  animation: beat 1s linear infinite;
}
```

```css
.heart {
  width: 50px;
  height: 50px;
  background-color: red;
  transform: rotate(-45deg);
  margin: 50px;
}
.heart::before, .heart::after {
  content: '';
  position: absolute;
  width: 50px;
  height: 50px;
  border-radius: 50%;
  background-color: red;
}
.heart::before { left: 25px; }
.heart::after { top: -25px; }
```

4.8 Variables

CSS variables or custom properties are property names prefixed with `--`, their values can be used in other declarations using the `var()` function. CSS variables are scoped to the element they are declared on.

Define variables on the `:root` pseudo-class, so that it can be referenced globally:

```css
:root {
    --primary-color: #0000FF;
    --secondary-color: #DDDDDD;
}
```

Access variables with `var()`:

```css
body {
    background-color: var(--primary-color);
}

button {
    background-color: var(--secondary-color);
}
```

`var()` also accepts a default value, in case the variable is not defined:

```
button {
    background-color: var(--secondary-color, blue);
}
```

4.9 Responsive Design

Responsive design in CSS is a technique for ensuring that websites look and function optimally on various devices, from desktop computers to smartphones. This can be achieved by using flexbox, grid layout, relative CSS unites and CSS media queries.

CSS Media Queries

Css Media queries are conditions that can be applied to different screen sizes, orientations, and resolutions.

Use media queries to apply different styles based on screen size:

```
@media (max-width: 600px) {
  .container {
    flex-direction: column; /* Stacks items vertically on small screens */
  }
}
```

CSS Units

Use relative units like `em` or `rem` for font sizes and spacing. This allows elements to scale proportionally with the screen size.

Example:

```
h1 {
  font-size: 2em;
}
```

There are two main categories of CSS units.

Relative Units

These units are relative to the size of the parent element or the viewport. This makes them responsive and adaptable to different screen sizes. Examples of relative units include:

- `rem`: Relative to the root element's font size.
- `em`: Relative to the font size of the parent element.
- `vw`: Viewport width (1vw = 1% of the viewport width).
- `vh`: Viewport height (1vh = 1% of the viewport height).
- `vmin`: The smaller of vw and vh.
- `vmax`: The larger of vw and vh.

Absolute Units

These units are fixed and do not change based on the size of the parent element or the viewport. Examples of absolute units include:

- `px`: Pixels.
- `pt`: Points (1pt = 1/72 of an inch).
- `in`: Inches.
- `cm`: Centimeters.
- `mm`: Millimeters.

clamp

`clamp(min, preferred, max)`

The preferred value often uses a relative uinite like vw, %, em, etc. `clamp()` ensures the size stay within min and max.

```css
.container {
  width: clamp(300px, 50vw, 800px);
}
```

In this case, the width of the `.container` will be 50% of the viewport width, if that falls within the bounds (300px to 800px), other wise it will be 300px or 800px.

4.10 Real World CSS

In real-world applications usually you can't just write CSS. There are several challanges:

- Class name conflicts
- Browser compability(you can't use latest features)
- Code reuse

Besides those, there are also efforts to provide you a better than CSS language or a higher level abstraction.

CSS Modules

CSS Module solves the scoping problem. It rewrite your class name selectors to avoid conflicts:

```
import styles from './style.css';

function component() {
  return <div className={styles.myClass}></div>
}
```

PostCSS

PostCSS supports variables and mixins, transpiles future CSS syntax, inlines images, and more.

Alternative CSS languages

Less, Scss, Sass

They extend CSS with features like variables, mixins, and functions.

CSS-in-JS

Write CSS within JavaScript, leveraging JavaScript's programmatic capabilities.

Solutions require a runtime: styled-components, Emotion.

Solutions with no runtime overhead: Vanilla extract

Frameworks

Bootstrap: provide pre-built components and styles.

Tailwind: offer extensive pre-built utility classes.