CSS Grid + Responsive Web Notes

Responsive Web Design

Responsive Web Design is an approach to making websites that can display properly on different devices with different screen sizes, such as desktops, laptops, tablets, and smartphones.

CSS gives the tools to write different style rules, then apply them depending on the device which is displaying the page/website.

Mantra for Responsive Design: Flow of writing styles for the devices

- First: Mobile view (Highest number of users.)
- Second: Desktop view (Second highest number of users)
- Third: Tablet view

This rule can change for the Desktop First apps. For example Microsoft Azure, etc.

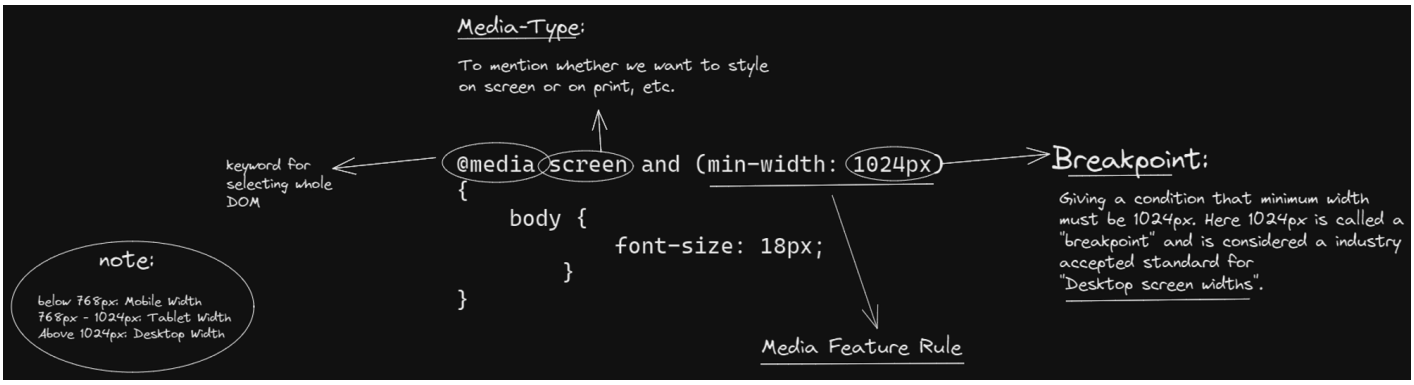
For those apps design desktop first and then mobile and tablet.

01: How does this work?

We achieve responsive design using media queries.

First, design using a mobile-first approach, and then using media queries we can make designs for desktop and tablet views.

Media Query Basics



It consists of:

- media-type - It tells the browser what kind of media this code is for. (e.g. print, screen, etc.) Mentioning media type is optional.
- media-feature-rule - A rule that must be passed for the contained CSS to get applied.
    - For example: `min-width: 768px`
- CSS rules inside the query will be applied if the rule passes and the media type is correct.

Example:

```
/* default styles for mobile-first approach*/
p {
  font-size: 8px;
}

/* Styles for desktop */
@media (min-width: 1024px) {
  p {
    font-size: 16px;
  }
}

/* Styles for tablet screen */
@media (min-width: 768px) {
  p {
    font-size: 20px;
  }
}
```

COPY

Whichever the default styles are written, gets applied to all screen sizes. But when a media query is defined, the browser applies the rules inside the queries only at specific screen intervals. Default style properties will get over-written with properties inside the media query.

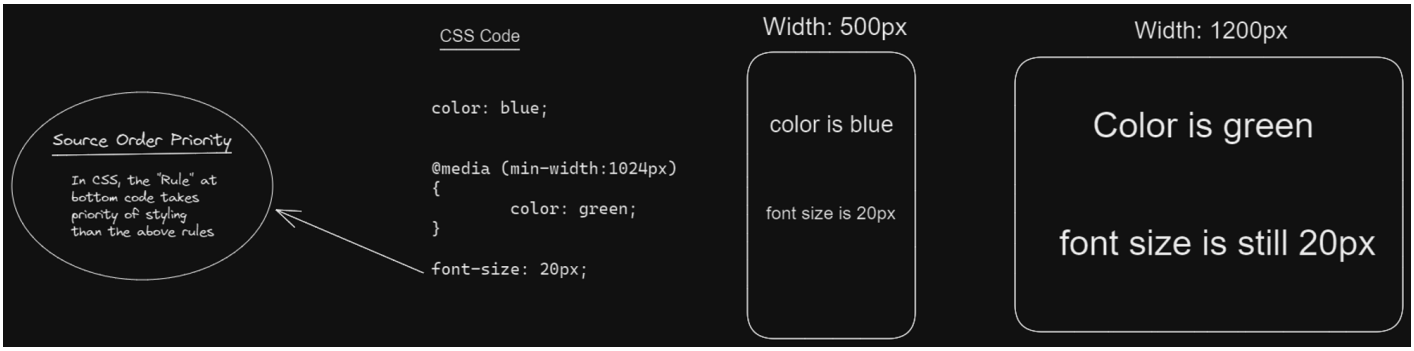
Breakpoint: A point where the screen changes its visibility/layout. Here, we can say 768px and 1024px are breakpoints.

Common breakpoints are:

- screen width >= 1024px- for desktops
- 1024px > screen width >= 768px - for tablets

- 768px > screen width - for mobiles

Example of Media Query



Note: Do not populate media queries. It’s a good practice to use fewer media queries.

02: Responsive Typography

Responsive typography adjusts the font size, spacing, and line height of text on the websites to make sure it is easily readable on different screen sizes and devices.

1. It’s important to change the font size when the screen size varies.
2. The units should be relative. Mainly rem.
3. Below is a table containing standard units, to keep it handy while building projects and in machine coding rounds.

Units

Breakpoint	h1 (Title)	h2 (Subtitle)	p (Paragraph)
Mobile	2rem	1.5rem	1rem
Tablet	2.5rem	1.75rem	1.125rem
Laptop	3rem	2rem	1.25rem

Note: Make a few sets of your own basic designs like cards, buttons, colors, and fonts. It will be useful.

Grids

CSS Grid helps to easily build complex web designs.

It works by turning an HTML element into a container (grid container) with rows and columns to place children elements wherever within the grid.

Grids are strict. Hence, use it when you know the exact number of components to be used.

Using grids we can make 2D layouts while Flex is used to make 1D layouts.

Grids are useful when you are making a layout for your website or app.

Rule of Thumb: Layout the whole page using Grid and make individual components using Flex.

Syntax:

```
display: grid;

/* to make columns */
grid-template-columns: 200px 1fr auto;
grid-column-gap: 14px;

/* to make rows */
grid-template-rows: 1fr 1fr 1fr;
grid-row-gap: 10px;
```

COPY

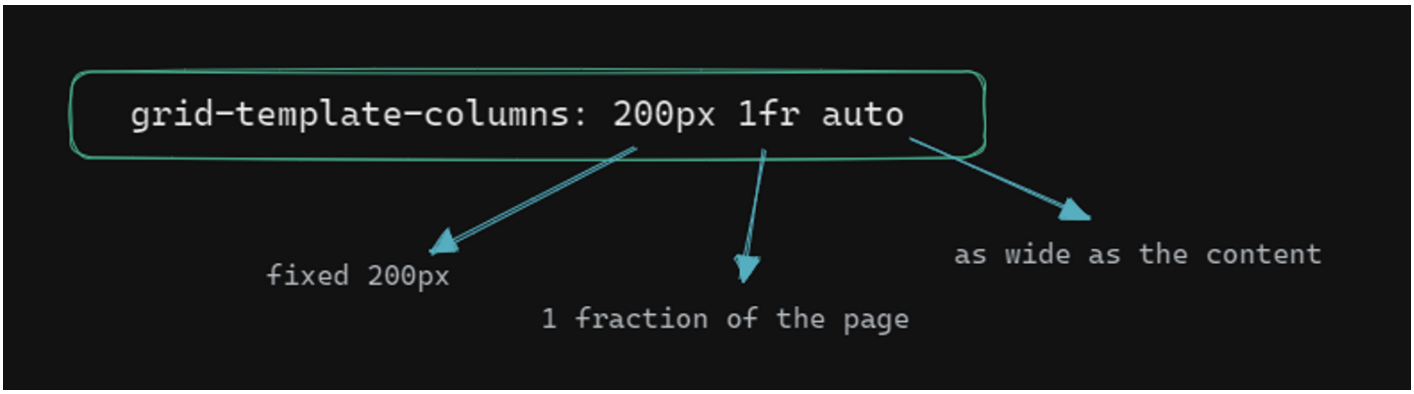
Small intro to all these properties:

- `display: grid` - Gives the ability to use all other CSS properties associated with CSS Grid.
- `grid-template-columns` - To add some columns to the grid. The number of parameters given to this property indicates the no. of columns in the grid and the value of each parameter indicates the width of each column.
- `grid-column-gap` - If we want a gap between columns, we'll use this.
- `grid-template-rows` - Sets the number of rows. Use this property in the same way grid-template-columns.
- `grid-row-gap` - Sets gap between the rows.

CSS Grid Units:

`grid-template-rows` / `grid-template-columns` can take units in 3 forms:-

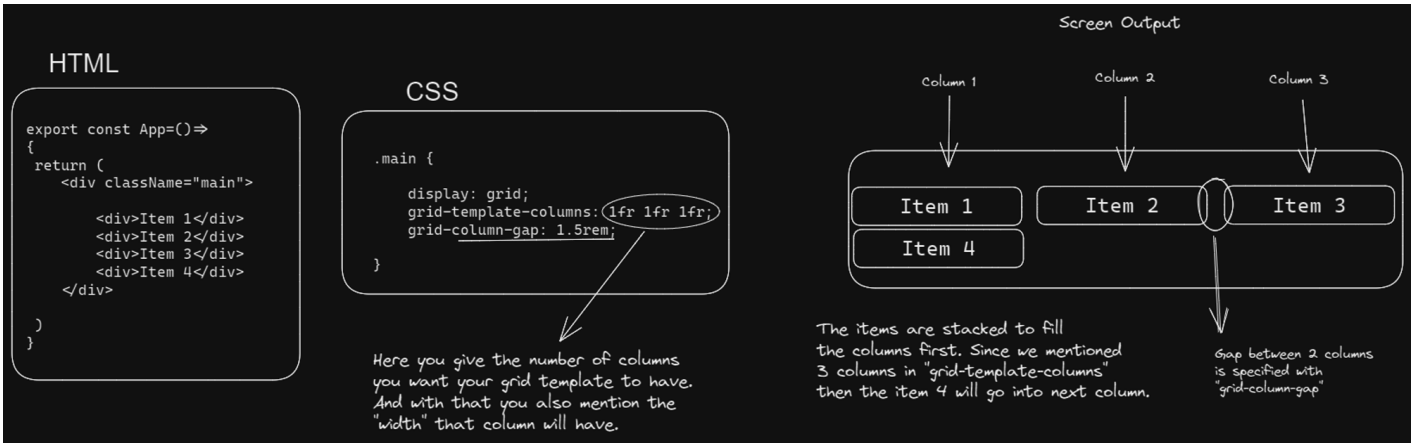
- Pixels
- Fraction
- Auto

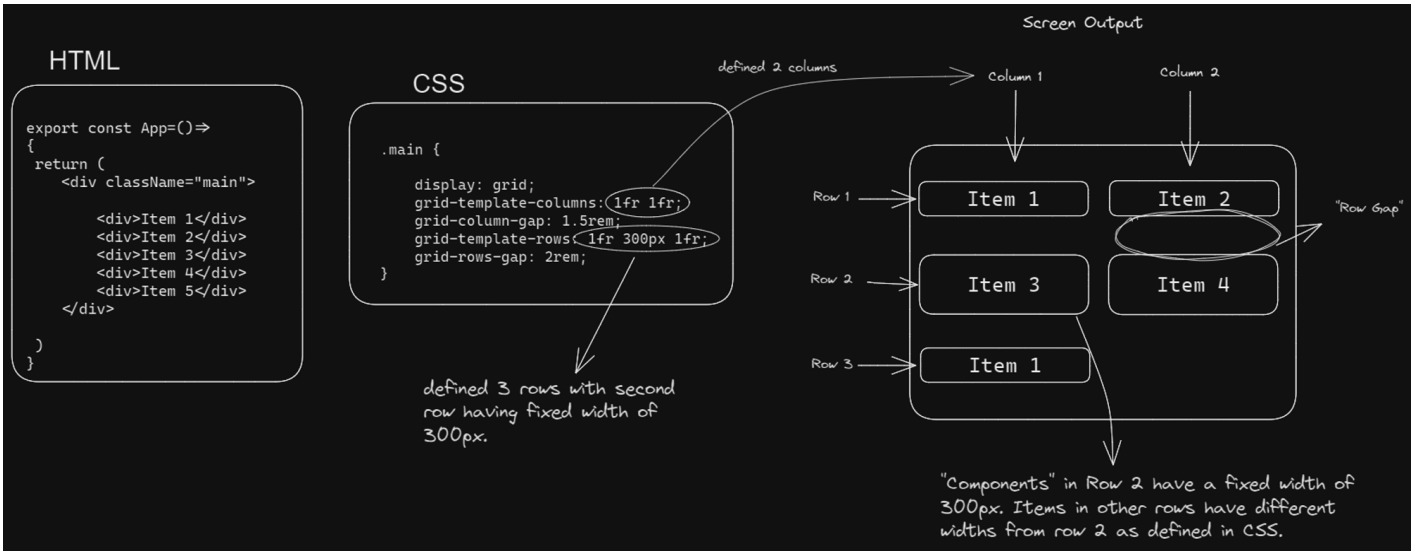


What is fr unit?

- The fr unit represents a fraction of the available space in the grid container.
- 1fr is 100% of the available space.
- 2fr is 50% of the available space.
- 3fr is 33% of available space and so on.

Example of Grid layout





The repeat() notation

New syntax introduced:

```
grid-template-columns: repeat(3, 1fr)
```

- `repeat()` is a notation that you can use with the grid-template-columns and grid-template-rows properties to make your rules more concise and easier to understand when creating a large number of columns or rows.
- `repeat()` repeats a value for a certain number of times.

Creating rows/columns without using repeat()

```
.container {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
  grid-template-rows: 200px 200px 200px;
}
```

COPY

Creating rows/columns using repeat()

```
.container {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-template-rows: repeat(3, 200px);
}
```

COPY

Grid Template Areas

Using grid, we can make the layout of the whole page. Following are the properties used to make it.

Syntax:

```
.navbar {
  grid-area: nav;
}
.main {
  grid-area: main;
}
.content {
  grid-area: aside;
}
.footer {
  grid-area: footer;
}

.page-container {
  grid-template-areas:
    'nav nav nav'
    'main main aside'
    'footer footer footer';
}
```

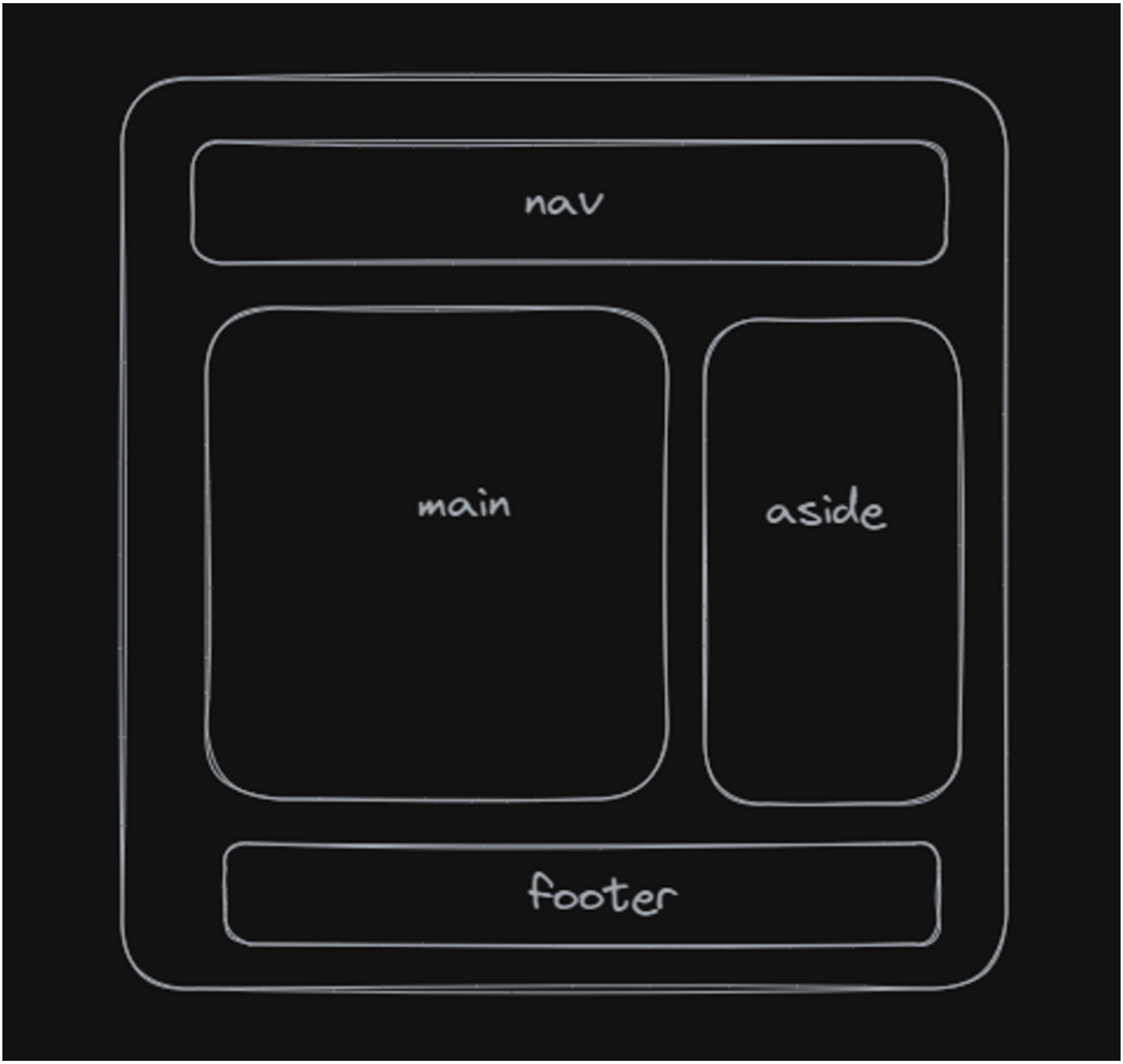
COPY

Small intro to these properties:

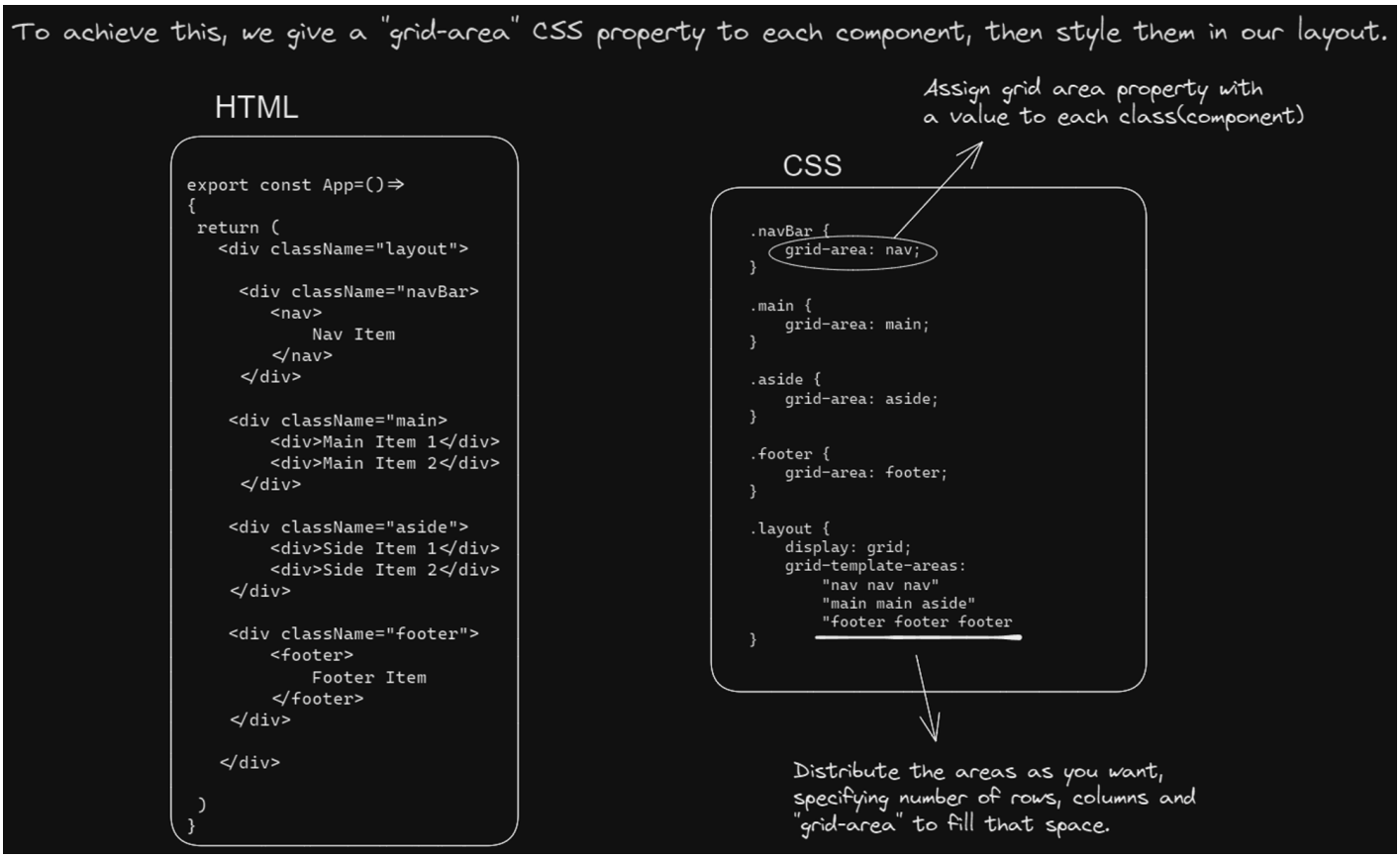
- `grid-areas` - Let the grid know that this particular property or class goes into the area given in its value.
- `grid-template-areas` - Used to group areas together and make a layout out of them.

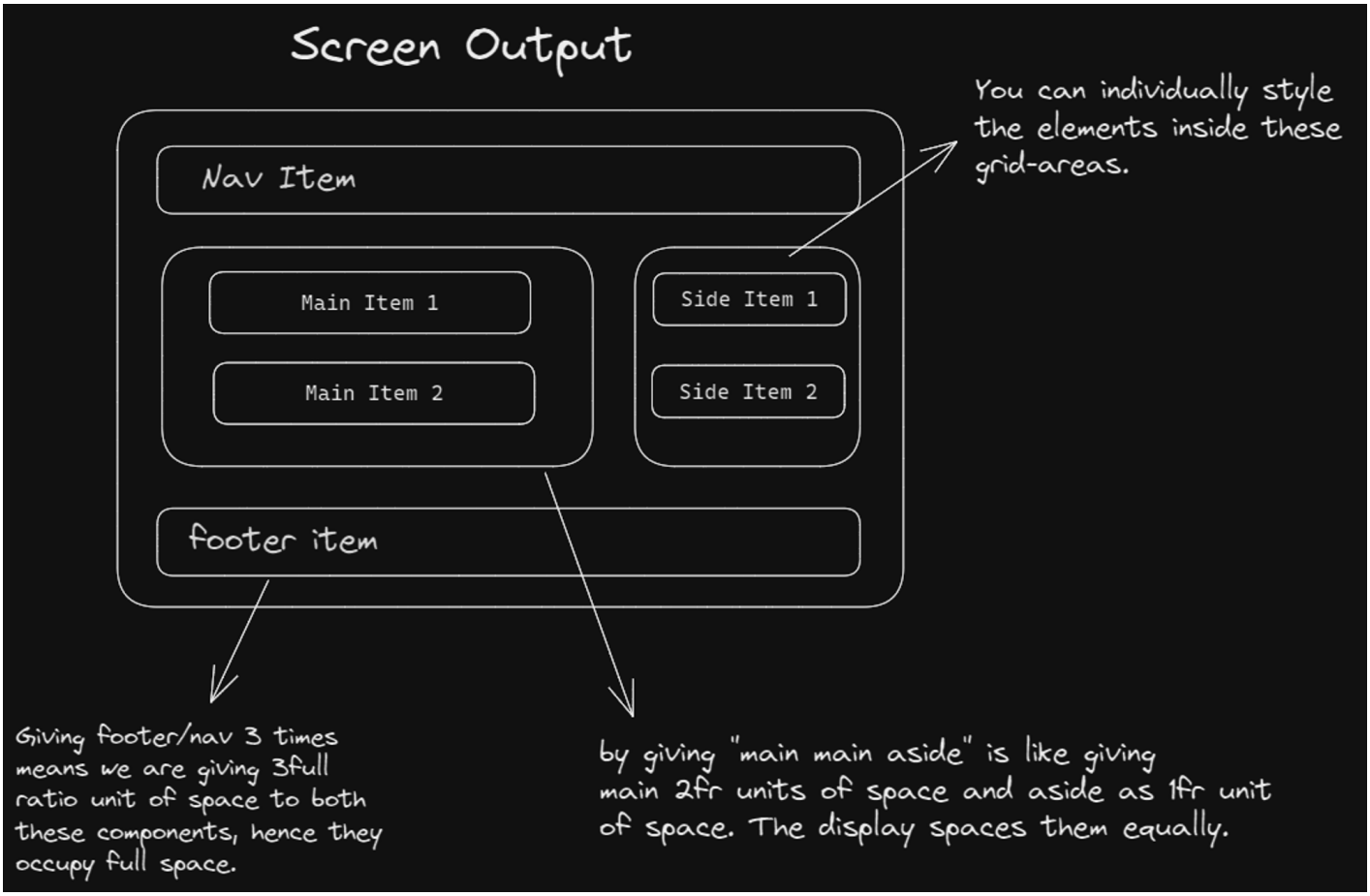
Here in the above syntax, it merges the top three cells together into an area named `nav` and the bottom three cells into `footer`. And it makes two areas in the middle row `main` and `aside`.

Visually, it will look like this:



Example of Grid Template Areas





Note:

1. The ****value of grid-area and grid-template-areas should match.
2. Always put the `grid-template-areas` into the outermost parent container.