

# Fixing Text Overflow

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Flexbox Navbar</title>
    <style>
        *{
            margin: 0;
            padding: 0;
        }

        nav{
            display: flex;

            /* justify-content: space-evenly; */
            /* The above line is good but you would not have
            precise control over margin, paadding or gap */

            justify-content: center;
            /* This line doesn't add any custom space so you can
            have more control over the three parameters here */
            gap: 0.5%;
        }

        nav ul{
            list-style: none;
            display: flex;
            gap: 20px;
        }

        nav ul a{
            text-decoration: none;
            font-family: 'Segoe UI', Tahoma, Geneva, Verdana,
            sans-serif;
            font-size: 16px;
            font-weight: 400;
            line-height: 14px ;
            color: rgb(34, 34, 34);
        }
    </style>

```

```
}
```

```
.options{  
    display: flex;  
    align-items: center;  
    justify-content: center;  
    /* no need to set height explicitly as flex-box  
    automatically matches the tallest (if we don't want then we  
    can set the height explicitly too) */
```

```
    margin: 5px ;  
    margin-left: 19%;
```

```
}
```

```
.options li{  
    padding: 2px;  
  
    cursor: pointer;  
    /*This actually changes the cursor to hand whenever  
    it enters in this div (Explored from harry's solution) */
```

```
    display: flex;  
    align-items: center;  
    justify-items: center;  
}
```

```
.other-icons{  
    display: flex;  
    align-items: center ;  
    justify-content: center;
```

```
}
```

```
.other-icons img{  
    height: 45px;  
}
```

```
nav img[src="icon.png"]{
```

```

        height: 92px;
        margin-left: 1vw;
    }

nav img[src="sub-icon.png"]{
    height: 12px;
}

/* Only Nav-Bar Code Above (From Previous Exercise) (
I Have just made a few improvements)*/
/* Now the above Navbar was made using flexbox,
whereas the below body would be made using grid */

main{
    display: grid;
    border: 2px solid black;
    grid-template-columns: 301.978px 301.978px
    301.978px;
    grid-template-rows: repeat(4, 201.903px);
    margin: 60px 308px;
}

.box{
    border: 1px solid black;
    display: grid;
    grid-template-areas: "title title title"
    "main main main"
    "footer footer footer";
    justify-items: center;
    align-items: center;
}

.box .title{
    grid-area: title;
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana,
    sans-serif;
    font-size: 18px;
    font-weight: 700;
    line-height: 26px;
}

```

```
        margin-top: 15px;
    }
.box .button{
    grid-area: main;
    background-color: #5CB85C;
    color: white;
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana,
    sans-serif;
    text-align: center;
    font-size: 12px;
    font-weight: 700;
    line-height: 24px;
    padding: 4px 16px;
    margin: 6px;
    border-radius: 6px;

}

.box .box-footer{
    grid-area: footer;
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana,
    sans-serif;
    text-align: center;
    font-size: 12px;
    font-weight: 700;
    line-height: 24px;

}

.box .box-footer{
    text-overflow: wrap;
}
```

</style>

</head>

<body>

<header>

<nav>



```
<div class="options">
    <ul>
        <li><a href="#">Products</a>
            
        </li>
        <li><a href="#">For Business</a>
            
        </li>

        <li><a href="#">Resource</a>
            
        </li>

        <li><a href="#">Academic Licensing</a>
            
        </li>

        <li><a href="#">Our Company</a>
            
        </li>
    </ul>
</div>

<div class="other-icons">
    
    
</div>
</nav>
</header>
```

```
<main>
    <div class="box">
        <div class="title">
```

```

        English
    </div>

    <div class="button">Download 64-bit</div>

    <div class="box-footer">
        <span class="font1">SHA256</span>
        <span
            class="font2">F710BF78A1948AA9DD9F7090192DA527097C
            05B9C483F455A05E93292D91AE27</span>
    </div>
</div>
</main>

</body>
</html>

```

my font2 is exceeding my grid...how can prevent overflow?

You're facing an **overflow issue** because the long `SHA256` string in `.font2` is not breaking or wrapping within the grid cell. To fix this and **keep it inside the grid box**, you can apply the following CSS properties:

### Add these styles to `.font2`:

css

```

.font2 {
    word-break: break-all;
    overflow-wrap: break-word;
    display: block; /* Optional, ensures it behaves like a block element */
    text-align: center; /* Optional, if you want it centered */
}

```

### Explanation:

- `word-break: break-all;` allows the browser to break words at any point.
- `overflow-wrap: break-word;` ensures long words/wide strings wrap inside the container.

- `display: block;` makes it stack properly if it's inside inline containers.

## Bonus (if still needed):

If it still exceeds the container for some reason, make sure the parent container (`.box-footer`) also doesn't allow overflow:

css

```
.box-footer {  
    overflow: hidden;  
    text-overflow: ellipsis; /* Optional: shows ... if you want to trim it */  
    white-space: normal; /* Allows wrapping */  
}
```

Would you like me to generate a cleaner visual layout for the grid with multiple boxes as well?

```
.box .box-footer{  
    grid-area: footer;  
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana,  
    sans-serif;  
    text-align: center;  
    font-size: 12px;  
    font-weight: 700;  
    line-height: 24px;  
    overflow: hidden;  
    white-space: normal;  
    width: 905.934;  
}
```

it doesn't work why?

Great question, Vaibhav. The issue is with this line in your CSS:

css

```
width: 905.934;
```

## ✖ What's wrong?

You've provided a number **without specifying a unit**, which makes it **invalid CSS**.

In CSS, all lengths **must include a unit** like `px`, `em`, `%`, `rem`, etc.

## ✓ Fix:

Add `px` (or the appropriate unit) to the value:

css

```
width: 905.934px;
```

## 🔍 Additional Notes:

- Since you're using **CSS Grid**, you usually don't need to manually set the width of elements inside a grid unless you want to override its behavior.
- If you do want the `.box-footer` to exactly match the full width of your `.box` (which is `301.978px` from your `grid-template-columns`), **don't hardcode it as `905.934px`**, because that's 3 columns wide. The individual `.box` should already handle its width.

## Recommended Minimal Fix:

Just let it take the full width of its container and allow wrapping:

css

```
.box .box-footer {  
    grid-area: footer;  
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;  
    text-align: center;  
    font-size: 12px;  
    font-weight: 700;  
    line-height: 24px;  
    word-break: break-all;  
    overflow-wrap: break-word;  
}
```

Let me know if you want to trim the SHA256 or display it as a copyable textbox instead.

how can we wrap items of rid?

In CSS Grid, **items do not automatically wrap** like in Flexbox. If you want items to wrap to the next row, you need to **use `grid-auto-flow: row`** and **let the container define the number of columns based on available space** using `auto-fill` or `auto-fit`.

## ✓ Example: Wrapping items in a responsive grid

css

```
main {  
  display: grid;  
  grid-template-columns: repeat(auto-fill, minmax(300px, 1fr));  
  gap: 20px;  
  padding: 20px;  
}
```

### 🔍 Explanation:

- `auto-fill` or `auto-fit` fills the row with as many columns as can fit.
- `minmax(300px, 1fr)` sets a **minimum width of 300px**, and allows it to expand.
- `1fr` distributes remaining space equally.
- This setup allows items to **wrap** when the screen is smaller.

### ✨ Alternative (Fixed Width):

If you want a fixed box width and want them to wrap, just don't hardcode `grid-template-columns`:

css

```
main {  
  display: grid;  
  grid-template-columns: repeat(auto-fill, 301.978px); /* your fixed width */  
  gap: 20px;  
  justify-content: center;  
}
```

This way, boxes will wrap automatically once they don't fit in the row.

Would you like me to refactor your current `<main>` grid with multiple `.box` items and wrapping?

if instead of auto-fill I right 1 then will I get 1 box?

Great question, Vaibhav!

## ✓ Short Answer:

Yes, if you write `grid-template-columns: repeat(1, 1fr);`, you'll get **1 column**, which means **only 1 box per row** — all items will stack **vertically** (one below the other).

## 🔍 Example:

css

```
main {  
  display: grid;  
  grid-template-columns: repeat(1, 1fr);  
  gap: 20px;  
}
```

This layout will:

- Create 1 column that takes full width (`1fr`)
- Wrap every new item to the **next row**
- Result in a **vertical list of boxes**

## ✓ When to use:

Use this when:

- You want a **mobile-first** layout
- You're making a **responsive design** and want to override layout using media queries

Let me know if you want to make it responsive — e.g., 1 column on small screens, 3 on larger ones.

I want to make it responsive as when the width is shrunked  
then my code aligns every box in a vertical manner  
<!DOCTYPE html>

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Flexbox Navbar</title>
    <style>
        *{
            margin: 0;
            padding: 0;
        }

        nav{
            display: flex;

            /* justify-content: space-evenly; */
            /* The above line is good but you would not have
            precise control over margin, paadding or gap */

            justify-content: center;
            /* This line doesn't add any custom space so you can
            have more control over the three parameters here */
            gap: 0.5%;
        }

        nav ul{
            list-style: none;
            display: flex;
            gap: 20px;
        }

        nav ul a{
            text-decoration: none;
            font-family: 'Segoe UI', Tahoma, Geneva, Verdana,
            sans-serif;
            font-size: 16px;
            font-weight: 400;
            line-height: 14px ;
            color: #4b4b4b

        }
    </style>

```

```
.options{
    display: flex;
    align-items: center;
    justify-content: center;
    /* no need to set height explicitly as flex-box
    automatically matches the tallest (if we don't want then we
    can set the height explicitly too) */

    margin: 5px ;
    margin-left: 19%;

}

.options li{
    padding: 2px;

    cursor: pointer;
    /*This actually changes the cursor to hand whenever
    it enters in this div (Explored from harry's solution) */

    display: flex;
    align-items: center;
    justify-items: center;
}

.other-icons{
    display: flex;
    align-items: center ;
    justify-content: center;

}

.other-icons img{
    height: 45px;
}

nav img[src="icon.png"]){
    height: 92px;
    margin-left: 1vw;
}
```

```

nav img[src="sub-icon.png"]{
    height: 12px;
}

/* Only Nav-Bar Code Above (From Previous Exercise) (
I Have just made a few improvements)*/
/* Now the above Navbar was made using flexbox,
whereas the below body would be made using grid */

main{
    display: grid;
    grid-template-columns: 301.978px 301.978px
    301.978px;
    grid-template-rows: repeat(4, 201.903px);
    margin: 60px 308px;
    gap: 25px;
}

.box{
    display: grid;
    grid-template-areas: "title title title"
        "main main main"
        "footer footer footer";
    justify-items: center;
    align-items: center;
    background-color: #F9F9F9;
}

.box .title{
    grid-area: title;
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana,
    sans-serif;
    font-size: 18px;
    font-weight: 700;
    line-height: 26px;
    margin-top: 15px;
}

.box .button a{

```

```
text-decoration: none;
color: white;
font-family: 'Segoe UI', Tahoma, Geneva, Verdana,
sans-serif;
text-align: center;
font-size: 12px;
font-weight: 700;
line-height: 24px;
}

.box .button{
grid-area: main;
padding: 4px 16px;
background-color: #5CB85C;
margin: 6px;
border-radius: 6px;
transform: translateY(26px);
/* I was gonna use margin-up or bottem here, but
but translate is amuch better option since it doesn't disturb
any other element */

}

.box .box-footer .font2{
grid-area: footer;
text-align: center;
font-family: 'Segoe UI', Tahoma, Geneva, Verdana,
sans-serif;
font-size: 8px;
line-height: 16px;
white-space: normal;
word-break: break-all;

}

.box .box-footer .font1{
grid-area: footer;
text-align: center;
font-family: 'Segoe UI', Tahoma, Geneva, Verdana,
sans-serif;
font-size: 10px;
line-height: 16px;
font-weight: 450;
white-space: normal;
word-break: break-all;
```

```
}
```

```
.box .box-footer{  
    margin: 8px 30px;  
    width: 240px;  
}
```

```
</style>  
</head>  
<body>  
    <header>  
        <nav>  
  
              
  
            <div class="options">  
                <ul>  
                    <li><a href="#">Products</a>  
                          
                    </li>  
                    <li><a href="#">For Business</a>  
                          
                    </li>  
  
                    <li><a href="#">Resource</a>  
                          
                    </li>  
  
                    <li><a href="#">Academic Licensing</a>  
                          
                    </li>  
  
                    <li><a href="#">Our Company</a>
```

```
        
```

```
    </li>
```

```
  </ul>
```

```
</div>
```

```
<div class="other-icons">
```

```
    
```

```
    
```

```
  </div>
```

```
</nav>
```

```
</header>
```

```
<main>
```

```
<div class="box">
```

```
    <div class="title">English</div>
```

```
    <div class="button"><a
```

```
        href="ue_english_64.exe">Download 64-bit</a></div>
```

```
    <div class="box-footer">
```

```
        <p class="font1">SHA256</p>
```

```
        <p
```

```
            class="font2">F710BF78A1948AA9DD9F7090192DA527097C  
            05B9C483F455A05E93292D91AE27</p>
```

```
    </div>
```

```
</div>
```

```
<div class="box">
```

```
    <div class="title">English</div>
```

```
    <div class="button"><a
```

```
        href="ue_english_64.exe">Download 64-bit</a></div>
```

```
    <div class="box-footer">
```

```
        <p class="font1">SHA256</p>
```

```
        <p
```

```
            class="font2">F710BF78A1948AA9DD9F7090192DA527097C  
            05B9C483F455A05E93292D91AE27</p>
```

```
    </div>
```

```
</div>
```

```
<div class="box">
    <div class="title">English</div>
    <div class="button"><a
        href="ue_english_64.exe">Download 64-bit</a></div>
        <div class="box-footer">
            <p class="font1">SHA256</p>
            <p
                class="font2">F710BF78A1948AA9DD9F7090192DA527097C
                05B9C483F455A05E93292D91AE27</p>
        </div>
    </div>
```

```
<div class="box">
    <div class="title">English</div>
    <div class="button"><a
        href="ue_english_64.exe">Download 64-bit</a></div>
        <div class="box-footer">
            <p class="font1">SHA256</p>
            <p
                class="font2">F710BF78A1948AA9DD9F7090192DA527097C
                05B9C483F455A05E93292D91AE27</p>
        </div>
    </div>
```

```
<div class="box">
    <div class="title">English</div>
    <div class="button"><a
        href="ue_english_64.exe">Download 64-bit</a></div>
        <div class="box-footer">
            <p class="font1">SHA256</p>
            <p
                class="font2">F710BF78A1948AA9DD9F7090192DA527097C
                05B9C483F455A05E93292D91AE27</p>
        </div>
    </div>
```

```
<div class="box">
    <div class="title">English</div>
    <div class="button"><a href="ue_english_64.exe">Download 64-bit</a></div>
    <div class="box-footer">
        <p class="font1">SHA256</p>
        <p
            class="font2">F710BF78A1948AA9DD9F7090192DA527097C
            05B9C483F455A05E93292D91AE27</p>
    </div>
</div>
```

```
<div class="box">
    <div class="title">English</div>
    <div class="button"><a href="ue_english_64.exe">Download 64-bit</a></div>
    <div class="box-footer">
        <p class="font1">SHA256</p>
        <p
            class="font2">F710BF78A1948AA9DD9F7090192DA527097C
            05B9C483F455A05E93292D91AE27</p>
    </div>
</div>
```

```
<div class="box">
    <div class="title">English</div>
    <div class="button"><a href="ue_english_64.exe">Download 64-bit</a></div>
    <div class="box-footer">
        <p class="font1">SHA256</p>
        <p
            class="font2">F710BF78A1948AA9DD9F7090192DA527097C
            05B9C483F455A05E93292D91AE27</p>
    </div>
</div>
```

```
<div class="box">
    <div class="title">English</div>
    <div class="button"><a
```

```
href="ue_english_64.exe">Download 64-bit</a></div>
<div class="box-footer">
<p class="font1">SHA256</p>
<p
class="font2">F710BF78A1948AA9DD9F7090192DA527097C
05B9C483F455A05E93292D91AE27</p>
</div>
</div>

<div class="box">
<div class="title">English</div>
<div class="button"><a
href="ue_english_64.exe">Download 64-bit</a></div>
<div class="box-footer">
<p class="font1">SHA256</p>
<p
class="font2">F710BF78A1948AA9DD9F7090192DA527097C
05B9C483F455A05E93292D91AE27</p>
</div>
</div>

</main>

</body>
</html>
```

this is my original code

Thanks for sharing your full code, Vaibhav! You're doing a great job with the layout and commenting. To make your layout **responsive**, so the boxes wrap neatly into fewer columns (and eventually just 1 column) on smaller screens, you need to:

---

## ✓ 1. Replace the fixed-width grid:

You're currently using:

css

```
grid-template-columns: 301.978px 301.978px 301.978px;
```

This is **not responsive** because it's hardcoded.

⟳ Change it to:

css

```
grid-template-columns: repeat(auto-fit, minmax(280px, 1fr));
```

## ✓ 2. Remove hard margin that causes horizontal overflow:

You have:

css

```
margin: 60px 308px;
```

This adds huge space on both sides and breaks responsiveness.

⟳ Replace it with:

css

```
margin: 40px auto;  
padding: 0 20px;
```

## ✓ Final updated main CSS:

css

```
main {  
  display: grid;  
  grid-template-columns: repeat(auto-fit, minmax(280px, 1fr));  
  gap: 25px;
```

```
margin: 40px auto;  
padding: 0 20px;  
max-width: 1000px; /* Optional: controls max width on large screens */  
}
```

## ✓ Result:

- On large screens → multiple columns
- On tablets → 2 columns
- On phones → 1 column

Would you like me to also help make your navbar responsive (e.g., stacking into a hamburger menu on mobile)?

so I need not to use media queries?

Great question, Vaibhav.

You **can** make your layout responsive *without* media queries if you're using **CSS Grid or Flexbox** smartly with **flexible units** like percentages (`%`), fractions (`fr`), or `auto-fit/minmax` —which is exactly what I showed in the improved version.

However, **media queries** are still useful when you want *precise control* over:

- Font sizes on different devices.
- Switching from grid to flex (or vice versa).
- Rearranging elements completely on mobile (e.g., collapsing a navbar into a hamburger menu).
- Applying different paddings/margins.

## In summary:

Approach	Use When
Flexible units (like <code>fr</code> , <code>%</code> , <code>minmax()</code> )	You want basic responsiveness (boxes wrap, adjust sizes).

Approach	Use When
Media queries	You want full control or make drastic changes on different screen sizes.

So, you don't **need media queries** for your case if:

- You're only aligning boxes vertically when the screen is small.
- You use `auto-fit` or `auto-fill` with `minmax()` inside `grid-template-columns`.

But you **can add media queries** to polish the design further (like reducing padding or font size on small screens).

Would you like help adding optional media queries for polish or is your current result good enough for now?

no I want to keep this 3 boxes per row thing but want to make it responsive for mobile devices which shows 1 box

Got it, Vaibhav! You want:

- **3 boxes per row on desktops,**
- **1 box per row on mobile devices.**

This is a perfect use case for **media queries**.

Here's how you do it with minimal changes:

## ✓ Final HTML + CSS Code:

html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>Responsive Grid</title>
  <style>
    .grid-container {
      display: grid;
```

```

grid-template-columns: repeat(3, 1fr);
gap: 20px;
padding: 20px;
}

.box {
background-color: lightblue;
padding: 40px;
text-align: center;
font-size: 20px;
border-radius: 10px;
}

/* Media query for mobile devices */
@media (max-width: 768px) {
.grid-container {
grid-template-columns: 1fr;
}
}

</style>
</head>
<body>
<div class="grid-container">
<div class="box">Box 1</div>
<div class="box">Box 2</div>
<div class="box">Box 3</div>
</div>
</body>
</html>

```

## 🧠 Explanation:

- `repeat(3, 1fr)` → 3 boxes per row by default.
- `@media (max-width: 768px)` → For screens smaller than 768px (mobiles/tablets), switch to `1fr` → 1 box per row.

This keeps your layout clean and readable across all screen sizes.

Want to add animations or effects too, or keep it minimal?

I mean I want to do it without media queries

Ah, thanks for the clarification, Vaibhav!

Yes — you can make it responsive (3 boxes on desktop, 1 on mobile) *without media queries* by using CSS Grid's `auto-fit` or `auto-fill` with `minmax()`.

## ✓ Clean Solution: No Media Queries Needed

html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>Responsive Grid Without Media Queries</title>
  <style>
    .grid-container {
      display: grid;
      grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));
      gap: 20px;
      padding: 20px;
    }

    .box {
      background-color: lightblue;
      padding: 40px;
      text-align: center;
      font-size: 20px;
      border-radius: 10px;
    }
  </style>
</head>
<body>
  <div class="grid-container">
    <div class="box">Box 1</div>
    <div class="box">Box 2</div>
    <div class="box">Box 3</div>
  </div>
```

```
</body>  
</html>
```

## 🧠 Why This Works:

- `repeat(auto-fit, minmax(250px, 1fr))` means:
  - Fit as many boxes as will fit, each **minimum 250px wide**,
  - If the screen is too small (like on mobile), it automatically shifts to 1 per row.

No `@media` needed — the grid is smart and fluid. 🙌

Want me to explain `auto-fit` vs `auto-fill` too?

I just wanna know what exactly is auto-fit and auto-fill is?

Great question, Vaibhav. Let's break it down super simply:

## 🎯 `auto-fit` vs `auto-fill` in CSS Grid

Both are used with the `repeat()` function like this:

css

```
grid-template-columns: repeat(auto-fit or auto-fill, minmax(min, max));
```

And they both **try to fit as many columns as possible** into the container — but they differ in **how they handle empty space**.

### ◆ `auto-fill`

- **Fills the row with as many columns as possible**, even if they're **empty**.
- It reserves space even if there aren't enough items.

🧠 Think of it as:

| "Fill up the row with as many grid slots as you can, even if some are empty."

---

- ◆ **auto-fit**

- Also tries to fit as many columns as possible...
- **But collapses the empty ones** — so grid items stretch to fill available space.

 Think of it as:

| "Fit items into the grid and stretch them to use the space."

---

## Example to See the Difference

Let's say your container is 1000px wide and you're using:

```
css
```

```
grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));
```

- If 3 items fit: It shows 3 columns.
- If only 1 fits on small screens: It shows 1 per row.
- **auto-fit makes them stretch** to fill the row.

With **auto-fit**, if fewer items are present, **you may see empty grid cells** (space reserved even though no box is there).

---

## Which One Should You Use?

- **Use auto-fit** 90% of the time — it's smarter for responsive layouts where stretching and collapsing is desired.
- **Use auto-fill** when you want the grid to maintain its structure even if some columns are empty (like in forms or placeholders).

Would you like me to show you a live example where you can toggle between the two and see the effect?

wu did yo give a differnt value in minmax, should max value be 301.978px since it was the original width?

Great observation, Vaibhav — and I see what you're thinking. But no, the **max value in `minmax()` doesn't need to match the original box width**. Here's why:

## 🔍 `minmax(min, max)` — What It Actually Means

This function sets a range for each column's width:

css

```
minmax(250px, 1fr)
```

- `250px` = the **minimum width** a column can shrink to.
- `1fr` = the **maximum width** it can grow to (a fraction of remaining space).

## 🤔 Why not use `minmax(250px, 301.978px)`?

Because:

- That would **lock** the max width to 301.978px, **preventing flexibility**.
- In responsive design, we **want boxes to adapt** — grow and shrink depending on screen size.
- Using `1fr` lets the grid **dynamically distribute extra space** between columns — it's way more responsive.

So unless you have a fixed design requirement, it's **better to use**:

css

```
minmax(250px, 1fr)
```

## TL;DR

- `1fr` is flexible — it makes the layout **fluid and responsive**.
- You don't need to (and shouldn't) lock it to `301.978px` unless you absolutely want fixed sizes.

Do you want me to explain what `fr` units actually are and how they distribute space?

but then it won't show the original size when I open in laptop, will it?

You're right to consider that! If you use `minmax(250px, 1fr)` instead of the original `301.978px`, the width will be **flexible** and adapt to the screen size, but it may **shrink smaller than 301.978px** when the screen gets narrow.

To **preserve the original size on larger screens** (like laptops) and allow for responsiveness on smaller screens (like mobiles), you should balance the flexibility with the original size.

Here's a more tailored approach:

### Solution: Mix `minmax()` and Preserve the Original Size

css

```
grid-template-columns: repeat(auto-fit, minmax(301.978px, 1fr));
```

- `301.978px`: The original width, which will be the **ideal width** for larger screens like laptops.
- `1fr`: This makes the columns **expand** and **shrink** when there's extra space available or when the screen size reduces.
- `auto-fit`: Automatically adjusts to the container width by filling the available space.

### What happens now:

- On **larger screens** (laptop or desktop), the columns will try to stay at `301.978px` wide (as per the original width you wanted).
- On **smaller screens** (like mobile), the width of the boxes will **shrink** but will never be smaller than `250px` (or whatever minimum you choose).

## Result:

- **Laptop/Desktop:** You'll get **3 boxes** per row with the original width ( `301.978px` ).
- **Mobile/Small Screens:** The boxes will still be responsive and adjust to **1 box per row**, but won't get smaller than `250px` (or the minimum you set).

Let me know if this solves your problem!

- ↳ On \*\*smaller screens\*\* (like mobile), the width of the boxes will \*\*shrink\*\* but will never be smaller than \*\*`250px`\*\* (or whatever minimum you choose).

how can you say that, as you can see that min is `301.978px`  
so how can this become `250px` ?

You're absolutely right to point this out! My previous explanation was a bit misleading, and I appreciate your attention to detail.

In the solution where we use `minmax(301.978px, 1fr)`, the **minimum** value is the **original size** (`301.978px`), not `250px` or any other value. The idea here is that the **minimum width** for the boxes will always be **`301.978px`** (the original size). So the boxes won't shrink smaller than that, no matter the screen size.

Here's how it actually works:

### Breakdown of `minmax(301.978px, 1fr)`:

- **`301.978px`** is the **minimum width** — meaning that each box will never be smaller than this value.
- **`1fr`** is the **maximum width** — the boxes will stretch to fill any available space, but the width will always be at least **`301.978px`**.

### Key points:

- **On larger screens:** The boxes will try to stay at **`301.978px`** wide (your original width).
- **On smaller screens:** If there's not enough space for three boxes side by side, the layout will adjust to fit fewer boxes per row (like 2 or 1 box per row), but the width of each box will **not shrink below `301.978px`**.

So, if the screen is small enough (like on mobile), the number of boxes per row will reduce, but each box will still try to maintain the **minimum width of 301.978px**.

If you want them to shrink below 301.978px on smaller screens (say for responsiveness), you'd need to **lower the minimum value** like `minmax(250px, 1fr)`. Otherwise, the boxes will **remain fixed at 301.978px**.

Let me know if that clears up the confusion!

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Flexbox Navbar Copy</title>
    <style>
        *{
            margin: 0;
            padding: 0;
        }

        nav{
            display: flex;

            /* justify-content: space-evenly; */
            /* The above line is good but you would not have
            precise control over margin, paadding or gap */

            justify-content: center;
            /* This line doesn't add any custom space so you can
            have more control over the three parameters here */
            gap: 0.5%;
        }

        nav ul{
            list-style: none;
            display: flex;
            gap: 20px;
        }

        nav ul a{
            text-decoration: none;
            font-family: 'Segoe UI', Tahoma, Geneva, Verdana,
        }
    </style>

```

```
sans-serif;  
    font-size: 16px;  
    font-weight: 400;  
    line-height: 14px ;  
    color: #4b4b4b  
  
}  
  
.options{  
    display: flex;  
    align-items: center;  
    justify-content: center;  
/* no need to set height explicitly as flex-box  
automatically matches the tallest (if we don't want then we  
can set the height explicitly too) */  
  
margin: 5px ;  
margin-left: 19%;  
  
}  
  
.options li{  
    padding: 2px;  
  
cursor: pointer;  
/*This actually changes the cursor to hand whenever  
it enters in this div (Explored from harry's solution) */  
  
display: flex;  
align-items: center;  
justify-items: center;  
}  
  
.other-icons{  
    display: flex;  
    align-items: center ;  
    justify-content: center;  
  
}
```

```
.other-icons img{  
    height: 45px;  
}
```

```
nav img[src="icon.png"]{  
    height: 92px;  
    margin-left: 1vw;  
}
```

```
nav img[src="sub-icon.png"]{  
    height: 12px;  
}
```

```
/* Only Nav-Bar Code Above (From Previous Exercise) (  
I Have just made a few improvements)*/  
/* Now the above Navbar was made using flexbox,  
whereas the below body would be made using grid */
```

```
main{  
    display: grid;  
    width: 60vw;  
    /* original  
    grid-template-columns: repeat(3, 1fr);  
    */  
    grid-template-columns: repeat(auto-fit, 1fr);  
    grid-template-rows: repeat(4, 201.903px);  
    margin: 60px auto;  
    gap: 25px;
```

```
}
```

```
.box{  
    display: grid;  
    grid-template-areas: "title title title"  
                        "main main main"  
                        "footer footer footer";  
    justify-items: center;  
    align-items: center;  
    background-color: #F9F9F9;  
}
```

```
.box .title{
    grid-area: title;
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana,
    sans-serif;
    font-size: 18px;
    font-weight: 700;
    line-height: 26px;
    margin-top: 15px;
}

.box .button a{
    text-decoration: none;
    color: white;
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana,
    sans-serif;
    text-align: center;
    font-size: 12px;
    font-weight: 700;
    line-height: 24px;
}
.box .button{
    grid-area: main;
    padding: 4px 16px;
    background-color: #5CB85C;
    margin: 6px;
    border-radius: 6px;
    transform: translateY(26px);
    /* I was gonna use margin-up or bottem here, but
    but translate is amuch better option since it doesn't disturb
    any other element */
}

.box .box-footer .font2{
    grid-area: footer;
    text-align: center;
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana,
    sans-serif;
    font-size: 8px;
    line-height: 16px;
    white-space: normal;
    word-break: break-all;
```

```
}

.box .box-footer .font1{
    grid-area: footer;
    text-align: center;
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana,
    sans-serif;
    font-size: 10px;
    line-height: 16px;
    font-weight: 450;
    white-space: normal;
    word-break: break-all;

}

.box .box-footer{
    margin: 8px 30px;
    width: 240px;
}

</style>
</head>
<body>
<header>
<nav>

    

    <div class="options">
        <ul>
            <li><a href="#">Products</a>
                
            </li>
            <li><a href="#">For Business</a>
                
            </li>
        </ul>
    </div>

```

```
<li><a href="#">Resource</a>
    
</li>

<li><a href="#">Academic Licensing</a>
    
</li>

<li><a href="#">Our Company</a>
    
</li>
</ul>
</div>

<div class="other-icons">
    
    
</div>
</nav>
</header>

<main>

<div class="box">
    <div class="title">English</div>
    <div class="button"><a
href="ue_english_64.exe">Download 64-bit</a></div>
    <div class="box-footer">
        <p class="font1">SHA256</p>
        <p
class="font2">F710BF78A1948AA9DD9F7090192DA527097C
05B9C483F455A05E93292D91AE27</p>
    </div>
</div>
```

```
<div class="box">
    <div class="title">Hindi</div>
    <div class="button"><a href="ue_english_64.exe">Download 64-bit</a></div>
    <div class="box-footer">
        <p class="font1">SHA256</p>
        <p
class="font2">F710BF78A1948AA9DD9F7090192DA527097C
05B9C483F455A05E93292D91AE27</p>
    </div>
</div>
```

```
<div class="box">
    <div class="title">Japanese</div>
    <div class="button"><a href="ue_english_64.exe">Download 64-bit</a></div>
    <div class="box-footer">
        <p class="font1">SHA256</p>
        <p
class="font2">F710BF78A1948AA9DD9F7090192DA527097C
05B9C483F455A05E93292D91AE27</p>
    </div>
</div>
```

```
<div class="box">
    <div class="title">English</div>
    <div class="button"><a href="ue_english_64.exe">Download 64-bit</a></div>
    <div class="box-footer">
        <p class="font1">SHA256</p>
        <p
class="font2">F710BF78A1948AA9DD9F7090192DA527097C
05B9C483F455A05E93292D91AE27</p>
    </div>
</div>
```

```
<div class="box">
```

```
<div class="title">English</div>
<div class="button"><a href="ue_english_64.exe">Download 64-bit</a></div>
<div class="box-footer">
    <p class="font1">SHA256</p>
    <p
        class="font2">F710BF78A1948AA9DD9F7090192DA527097C
        05B9C483F455A05E93292D91AE27</p>
    </div>
</div>
```

```
<div class="box">
    <div class="title">English</div>
    <div class="button"><a href="ue_english_64.exe">Download 64-bit</a></div>
    <div class="box-footer">
        <p class="font1">SHA256</p>
        <p
            class="font2">F710BF78A1948AA9DD9F7090192DA527097C
            05B9C483F455A05E93292D91AE27</p>
        </div>
    </div>
```

```
<div class="box">
    <div class="title">English</div>
    <div class="button"><a href="ue_english_64.exe">Download 64-bit</a></div>
    <div class="box-footer">
        <p class="font1">SHA256</p>
        <p
            class="font2">F710BF78A1948AA9DD9F7090192DA527097C
            05B9C483F455A05E93292D91AE27</p>
        </div>
    </div>
```

```
<div class="box">
    <div class="title">English</div>
    <div class="button"><a href="ue_english_64.exe">Download 64-bit</a></div>
```

```
<div class="box-footer">
    <p class="font1">SHA256</p>
    <p
        class="font2">F710BF78A1948AA9DD9F7090192DA527097C
        05B9C483F455A05E93292D91AE27</p>
    </div>
</div>

<div class="box">
    <div class="title">English</div>
    <div class="button"><a
        href="ue_english_64.exe">Download 64-bit</a></div>
    <div class="box-footer">
        <p class="font1">SHA256</p>
        <p
            class="font2">F710BF78A1948AA9DD9F7090192DA527097C
            05B9C483F455A05E93292D91AE27</p>
        </div>
    </div>

    <div class="box">
        <div class="title">English</div>
        <div class="button"><a
            href="ue_english_64.exe">Download 64-bit</a></div>
        <div class="box-footer">
            <p class="font1">SHA256</p>
            <p
                class="font2">F710BF78A1948AA9DD9F7090192DA527097C
                05B9C483F455A05E93292D91AE27</p>
            </div>
        </div>

        </main>
```

```
</body>  
</html>
```

I did this it works for when I shrink the width however, it doesn't return to its original size in laptop, and instead it continues displaying 1 row and the width of the box stretches

You're very close, Vaibhav. The issue comes down to **how you're using `auto-fit` and `1fr` together in your grid**, which causes the boxes to **stretch across the full row** when there's not enough content to trigger wrapping. That's why you're seeing one big stretched box instead of multiple items per row when on a larger screen again.

## 🔍 What's the issue?

In this line:

```
css
```

```
grid-template-columns: repeat(auto-fit, 1fr);
```

- `1fr` means “take all available space equally.”
- `auto-fit` tries to fit as many of those `1fr` columns as it can.
- When the container gets wider, `auto-fit` keeps one `1fr`, which stretches across the full width.

So once it switches to one column due to lack of space, it **never knows when to go back to multiple columns**—because there's no fixed minimum width set for a column!

## ✓ Solution: Use `minmax` with `auto-fit`

Replace this line:

```
css
```

```
grid-template-columns: repeat(auto-fit, 1fr);
```

with:

```
css
```

```
grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));
```

## ✓ Why this works:

- `minmax(250px, 1fr)` tells the browser: "Each column should be **at least 250px wide**, but can grow up to `1fr` (equal space)."
- `auto-fit` will now correctly wrap boxes when the screen shrinks **and** bring them back when there's enough space.

## ✓ Updated part of your CSS:

css

```
main{  
  display: grid;  
  width: 60vw;  
  grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));  
  grid-template-rows: repeat(4, 201.903px);  
  margin: 60px auto;  
  gap: 25px;  
}
```

Now your layout will:

- Show multiple columns on laptop/desktop,
- Stack into fewer columns (even one) on small screens,
- Go **back** to the multi-column layout when the window widens again.

Would you like me to explain the difference between `auto-fit` and `auto-fill` too?

```
grid-template-columns: repeat(auto-fit, minmax(1fr,  
3fr));
```

I actually tried using this but it doesn't work too, Idk why

Ah, I see what you're trying to do — you're expecting the boxes to automatically adjust and revert to the original grid layout when the screen size is back to normal. The issue is with this line you tried:

css

```
grid-template-columns: repeat(auto-fit, minmax(1fr, 3fr));
```

This **doesn't work as intended** because `minmax(1fr, 3fr)` is invalid — `fr` units **cannot** be used inside `minmax()` for both minimum and maximum in a way that makes sense to the browser in this context.

---

## ✓ Correct Way to Make Responsive Grid That Reverts Nicely:

Instead, use this:

css

```
grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));
```

This means:

- Minimum width of a box: `250px`
  - Maximum: `1fr` (equally distributed space)
  - `auto-fit` makes sure that when space allows, boxes are packed into rows automatically.
- 

## Your Fixed CSS Snippet:

Replace your existing `grid-template-columns` line with:

css

```
grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));
```

Also, make sure your `main` has a limited `width` and centered properly (which you already did):

css

```
main {  
  display: grid;  
  width: 60vw;  
  margin: 60px auto;  
  gap: 25px;  
  grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));  
}
```

This ensures:

- Multiple columns on big screens (e.g., laptops)
- Collapsing into fewer columns on smaller screens (like tablets or phones)
- And it **goes back** to the original layout when the screen widens again — unlike what you were seeing.

---

Would you like a quick live demo code or CodePen link to visualize it?

why did you take 250px and not 301.98

Great follow-up, Vaibhav!

You're right to think about being precise — especially since your original layout had three columns in a `60vw` container, each `301.98px` wide (as you calculated from `60vw / 3 ≈ 301.98px`).

---

## So why `250px` and not `301.98px` ?

Because `250px` is a **flexible minimum width** that still ensures:

- **3 columns fit** on wider screens (like desktops and laptops).
- **2 or 1 column** naturally wrap when screen width is reduced.
- The grid remains responsive and fluid.

If you use `minmax(301.98px, 1fr)` instead, **the layout becomes too rigid** — it won't wrap into more columns unless there's *a lot* of space, reducing responsiveness on tablets or medium screens.

## Trade-off Explanation:

`minmax(301.98px, 1fr)`

`minmax(250px, 1fr)`

Exact match to original size      Slightly smaller, more flexible

May cause overflow or scroll      Easily wraps down to 2 or 1 column

Less responsive      More responsive

### Pro Tip:

The `250px` is a **design decision** — not a strict requirement. You can absolutely set it to `301.98px` if you want the layout to stick closer to the desktop look:

css

```
grid-template-columns: repeat(auto-fit, minmax(301.98px, 1fr));
```

But this will make it **less adaptive** on narrower screens — that's why most designers choose a slightly smaller and rounded number like `250px`.

Do you want me to show both side-by-side in a CodePen so you can see the difference clearly?

it works, but can you tell what exact it does? whitih each instant

Absolutely! Let's break down this line step by step:

css

```
grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));
```

This is part of **CSS Grid**, and it defines how the columns behave in a responsive layout.

## Full Breakdown (Instant by Instant):

### 1. `grid-template-columns`

- Tells the grid **how many columns** to create and **what width** each column should be.
- 

### 2. `repeat(...)`

- Used to **repeat a column pattern**.
  - Instead of writing: `250px 250px 250px`, you just say:  
`repeat(X, Y) → repeat Y X times.`
- 

### 3. `auto-fit`

- Automatically **fits as many columns as possible** into the row, based on the available width.
- As screen size increases, more columns will appear.
- As screen size decreases, columns **wrap to the next row**.

#### Example:

- On wide screen: 3 or more columns.
  - On tablet: 2 columns.
  - On mobile: 1 column.
- 

### 4. `minmax(250px, 1fr)`

This defines the **minimum and maximum width** of each column.

- `250px` → The column should be **at least 250px wide**.
- `1fr` → The column can **expand up to 1 fraction of the remaining space** in the row.

It's saying:

"Each column should be **at least 250px**, but if there's extra space, let it grow to fill it."

## 📱 Real-Life Behavior (Examples)

Screen Width	What Happens
>1000px (desktop)	3+ columns appear, each at least 250px, spreading across
700-900px (tablet)	Only 2 columns fit in a row, others move to next row
<500px (mobile)	Only 1 column fits; all others stack

## ⚡ Final Visual:

css

```
.container {  
  display: grid;  
  grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));  
  gap: 1rem; /* optional spacing between grid items */  
}
```

This gives you a **fluid, responsive layout** where items:

- Never get squished below 250px
- Wrap neatly on smaller screens
- Expand smoothly on large screens

Would you like a quick live demo or a visual grid sketch?

so what would happen if screen size below 250px are they reduced?

Great question, Vaibhav!

If the screen size goes **below 250px**, here's **exactly what happens**:

---

## Behavior When Screen < 250px

The `minmax(250px, 1fr)` says:

"Don't shrink this column below 250px."

So if your screen is **less than 250px wide**:

- **The column will not shrink** below 250px.
  - **Horizontal scrolling** will appear on the page.
  - The grid item will overflow the screen (unless you have `overflow: hidden`, which would clip it).
- 

## Why?

Because the `min` value in `minmax(min, max)` is a **hard minimum**.

- It forces the column to **always be at least 250px wide**.
  - So even on a tiny device, the column won't squash below that.
- 

## If You Want True Responsiveness (No Scroll):

If you want columns to go smaller than 250px on very small screens, you could change the rule to something like:

css

```
grid-template-columns: repeat(auto-fit, minmax(100px, 1fr));
```

or even:

css

```
grid-template-columns: repeat(auto-fit, minmax(0, 1fr));
```

But the trade-off is that items might become too narrow and squashed, especially text or images.

Would you like a version that adjusts cleanly even on very small screens (like old phones)?

my box-footer text is overflowing and not obeying things

```
.box .box-footer{  
    margin: 8px 30px;  
    width: 240px;  
}
```

is it because of these hardcoded values?

Yes, Vaibhav — the **hardcoded values** are very likely the reason your `.box-footer` text is overflowing.

Let's break down what's happening:

## 🔍 What's going wrong?

css

```
.box .box-footer {  
    margin: 8px 30px;  
    width: 240px;  
}
```

- `margin: 30px` on both sides = **60px total horizontal space**.
- `width: 240px` + margins = total box width becomes **300px**.
- If the parent container (like a grid item) is smaller than 300px, overflow happens.

- Also, `width: 240px` means it **can't shrink** to fit small screens — it's locked.
- 

## ✓ How to fix it

Instead of hardcoding width, let the footer **adapt**:

css

```
.box .box-footer {  
  margin: 8px 16px;  
  max-width: 100%;  
  width: auto;  
  word-wrap: break-word;  
  overflow-wrap: break-word;  
}
```

### Explanation:

- `max-width: 100%` ensures it doesn't exceed the parent.
  - `width: auto` lets it shrink/grow naturally.
  - `overflow-wrap: break-word` makes long text break to the next line.
- 

Would you like help rewriting the full `.box` layout to be more responsive?

width : auto takes who's reference?

Great question, Vaibhav.

## ✓ `width: auto` — Whose reference does it take?

`width: auto` means:

"Take up as much horizontal space as **your parent container allows**, based on your content and other constraints like margin, padding, and max-width."

So:

- If `.box-footer` is inside `.box`, and `.box` is inside a grid item —  
👉 Then `auto` width will be **based on the width of that grid item**.

It's **not based on the screen size directly**, but on **whatever block-level parent** it's sitting inside.

---

## ⚠ Important Notes:

- If parent has `display: flex` or `grid`, the `auto` width can behave slightly differently.
  - `max-width: 100%` ensures it never exceeds the parent's width, which is great for responsiveness.
  - If there's long unbroken text (like URLs), also add `word-break: break-word`.
- 

Do you want me to review your `.box` structure to suggest the best responsive layout?

```
.box .box-footer{  
    margin: 8px auto;  
    width: auto;  
    word-wrap: break-word;  
    overflow-wrap: break-word;;  
}
```

I tried making margin auto from left and right but its not centred why?

Good catch, Vaibhav.

---

## ❗ Why `margin: 8px auto` is *not* centering your `.box-footer`:

Even though you wrote:

```
css
```

```
margin: 8px auto;
```

```
width: auto;
```

...it won't center unless your `.box-footer` has a **fixed width less than its parent's width**.

## 💡 Reason:

- `width: auto` → makes `.box-footer` stretch to fill the full width of its parent.
- So `auto` margins have **no space left to "center"** it — it's already full width.

## ✓ How to fix it:

Option 1: Give it a max-width and center it

css

```
.box .box-footer {  
  margin: 8px auto;  
  width: 100%;  
  max-width: 240px;  
  word-wrap: break-word;  
  overflow-wrap: break-word;  
}
```

This way:

- It doesn't exceed 240px.
- But **centers itself** because now `margin: auto` works — it has free space to divide left and right.

Want me to check the `.box` parent styles too for full alignment?

I didn't understand why did it happen at the first place?

Great question, Vaibhav — let me break it down simply.

## What `margin: 8px auto` is supposed to do:

- It centers an element **horizontally**.
  - But it only works if the element **doesn't take the full width of its parent**.
  - Because: `auto` margin divides **remaining space** between left and right.
- 

## What happened in your case:

You used this:

css

```
.box .box-footer {  
    margin: 8px auto;  
    width: auto;  
}
```

But `width: auto` means:

"Take up all available horizontal space from the parent (minus padding/margins)."

So:

- Your `.box-footer` became as wide as its parent.
  - No space left to *split* between left and right.
  - So `margin: auto` had **nothing to balance** → nothing centered.
- 

## Simple analogy:

Imagine you're placing a box inside a shelf.

- If the box **fills the whole shelf**, you can't center it — it's already everywhere.
  - But if the box is **smaller than the shelf**, you can center it by equal space on both sides.
- 

Let me know if you want a quick visual example!

oh so auto in dimension try to extend to the max width that is of the parent?

Exactly, Vaibhav! 🔥

## ✓ When you use `width: auto`:

It means:

"Make this element as wide as its parent allows, minus padding, borders, and margins."

So unless the parent has **constraints** (like `max-width` or `padding`), `width: auto` will **stretch the element fully across** the parent.

## 🧠 Why this matters:

- If you want **centering**, the element **must not** take full width.
- That's why fixed widths like `width: 240px`, or flexible widths like `width: fit-content`, help in centering with `margin: auto`.

Want help choosing between `auto`, `fit-content`, `%`, or `max-content` for your specific layout?

`overflow-wrap: break-word;`  
what does it do exactly if I have word-wrap?

Both `overflow-wrap: break-word` and `word-wrap: break-word` perform the same function, and in most cases, you only need one of them. The reason both exist is for compatibility and different browser support.

## Here's what they do:

- `word-wrap: break-word;` (older syntax):

This tells the browser to break words at arbitrary points if necessary to avoid overflow. It's essentially forcing a break if a word is too long to fit on a line.

- `overflow-wrap: break-word;` (newer standard):

This is the newer name for `word-wrap`, but they both do the same thing. It ensures that long words or unbreakable strings (like long URLs) break into the next line if they overflow the container.

## Summary:

If you already have `word-wrap: break-word;`, `overflow-wrap: break-word;` won't add any extra functionality. They are interchangeable, but it's a good practice to use `overflow-wrap` as it's the updated standard.

## When to use:

- If you're working with long words or URLs and you want to prevent them from causing horizontal scrolling, use `overflow-wrap: break-word;` (or `word-wrap` as a fallback for older browsers).

↳ "Take up all available horizontal space from the parent (minus padding/margins)."

oh so is that why when we do auto in margin/padding then it tries to take the maximum out of the left space but since we do left-right simultaneously so it gets divided into 2?

Yes, exactly! When you use `margin: auto` on the left and right, it tells the browser to divide the available horizontal space equally between the left and right margins. This is how centering works with `auto` margins:

## Here's a breakdown:

1. `margin: auto;` on the left and right tells the browser to calculate the available horizontal space in the parent container.

### 2. How it works:

- The browser will take the total available space (the width of the parent container minus any padding or other content).
- Then, it divides that space equally between the left and right margins, effectively centering the element.

## Example:

If your parent container is 500px wide, and your element is 200px wide:

- **Total space available for margins** = 500px (parent width) - 200px (element width) = 300px.
- The left and right margins will each get 150px (300px / 2).
- This centers the element within the parent container.

## So, to summarize:

- **auto margin**: The browser divides the remaining space between left and right margins equally, centering the element.

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
  initial-scale=1.0">
  <title>Flexbox Navbar Copy</title>
  <style>
    *{
      margin: 0;
      padding: 0;
    }

    nav{
      display: flex;

      /* justify-content: space-evenly; */
      /* The above line is good but you would not have
      precise control over margin, paadding or gap */

      justify-content: center;
      /* This line doesn't add any custom space so you can
      have more control over the three parameters here */
      gap: 0.5%;
    }

    nav ul{
      list-style: none;
      display: flex;
      gap: 20px;
    }
  </style>
</head>
<body>
  <nav>
    <ul>
      <li>Home</li>
      <li>About</li>
      <li>Services</li>
      <li>Contact</li>
    </ul>
  </nav>
</body>
</html>

```

```
nav ul a{  
    text-decoration: none;  
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana,  
    sans-serif;  
    font-size: 16px;  
    font-weight: 400;  
    line-height: 14px ;  
    color: #4b4b4b  
  
}
```

```
.options{  
    display: flex;  
    align-items: center;  
    justify-content: center;  
    /* no need to set height explicitly as flex-box  
    automatically matches the tallest (if we don't want then we  
    can set the height explicitly too) */
```

```
margin: 5px ;  
margin-left: 19%;
```

```
}
```

```
.options li{  
    padding: 2px;  
  
    cursor: pointer;  
    /*This actually changes the cursor to hand whenever  
    it enters in this div (Explored from harry's solution) */
```

```
display: flex;  
align-items: center;  
justify-items: center;  
}
```

```
.other-icons{  
    display: flex;  
    align-items: center ;  
    justify-content: center;
```

```
}
```

```
.other-icons img{  
    height: 45px;  
}
```

```
nav img[src="icon.png"]{  
    height: 92px;  
    margin-left: 1vw;  
}
```

```
nav img[src="sub-icon.png"]{  
    height: 12px;  
}
```

/\* Only Nav-Bar Code Above (From Previous Exercise) (  
I Have just made a few improvements)\*/

/\* Now the above Navbar was made using flexbox,  
whereas the below body would be made using grid \*/

```
main{  
    display: grid;  
    width: 60vw;  
    /* original  
    grid-template-columns: repeat(3, 1fr);  
    */  
    grid-template-columns: repeat(auto-fit,  
minmax(250px, 1fr));  
    grid-template-rows: repeat(4, 201.903px);  
    margin: 60px auto;  
    gap: 25px;
```

```
}
```

```
.box{  
    display: grid;  
    grid-template-areas: "title title title"  
                        "main main main"  
                        "footer footer footer";
```

```
justify-items: center;
align-items: center;
background-color: #F9F9F9;
}

.box .title{
grid-area: title;
font-family: 'Segoe UI', Tahoma, Geneva, Verdana,
sans-serif;
font-size: 18px;
font-weight: 700;
line-height: 26px;
margin-top: 15px;
}

.box .button a{
text-decoration: none;
color: white;
font-family: 'Segoe UI', Tahoma, Geneva, Verdana,
sans-serif;
text-align: center;
font-size: 12px;
font-weight: 700;
line-height: 24px;
}

.box .button{
grid-area: main;
padding: 4px 16px;
background-color: #5CB85C;
margin: 6px;
border-radius: 6px;
transform: translateY(26px);
/* I was gonna use margin-up or bottem here, but
but translate is amuch better option since it doesn't disturb
any other element */

}

.box .box-footer .font2{
grid-area: footer;
text-align: center;
font-family: 'Segoe UI', Tahoma, Geneva, Verdana,
sans-serif;
```

```
    font-size: 8px;  
    line-height: 16px;  
    white-space: normal;  
    word-break: break-all;  
  
}  
.box .box-footer .font1{  
    grid-area: footer;  
    text-align: center;  
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana,  
    sans-serif;  
    font-size: 10px;  
    line-height: 16px;  
    font-weight: 450;  
    white-space: normal;  
    word-break: break-all;  
  
}  
  
.box .box-footer{  
    margin: 8px auto;  
    width: 80%;  
    word-wrap: break-word;  
    overflow-wrap: break-word;;  
}
```

```
</style>  
</head>  
<body>  
    <header>  
        <nav>  
  
              
  
            <div class="options">  
                <ul>  
                    <li><a href="#">Products</a>  
                        
    </li>
    <li><a href="#">For Business</a>
        
    </li>

    <li><a href="#">Resource</a>
        
    </li>

    <li><a href="#">Academic Licensing</a>
        
    </li>

    <li><a href="#">Our Company</a>
        
    </li>
</ul>
</div>

<div class="other-icons">
    
    
</div>
</nav>
</header>
```

```
<main>

<div class="box">
    <div class="title">English</div>
    <div class="button"><a
href="ue_english_64.exe">Download 64-bit</a></div>
    <div class="box-footer">
```

```
<p class="font1">SHA256</p>
<p
class="font2">F710BF78A1948AA9DD9F7090192DA527097C
05B9C483F455A05E93292D91AE27</p>
</div>
</div>
```

```
<div class="box">
<div class="title">Hindi</div>
<div class="button"><a
href="ue_english_64.exe">Download 64-bit</a></div>
<div class="box-footer">
<p class="font1">SHA256</p>
<p
class="font2">F710BF78A1948AA9DD9F7090192DA527097C
05B9C483F455A05E93292D91AE27</p>
</div>
</div>
```

```
<div class="box">
<div class="title">Japenese</div>
<div class="button"><a
href="ue_english_64.exe">Download 64-bit</a></div>
<div class="box-footer">
<p class="font1">SHA256</p>
<p
class="font2">F710BF78A1948AA9DD9F7090192DA527097C
05B9C483F455A05E93292D91AE27</p>
</div>
</div>
```

```
<div class="box">
<div class="title">English</div>
<div class="button"><a
href="ue_english_64.exe">Download 64-bit</a></div>
<div class="box-footer">
<p class="font1">SHA256</p>
<p
class="font2">F710BF78A1948AA9DD9F7090192DA527097C
```

```
05B9C483F455A05E93292D91AE27</p>
    </div>
</div>
```

```
<div class="box">
    <div class="title">English</div>
    <div class="button"><a
href="ue_english_64.exe">Download 64-bit</a></div>
    <div class="box-footer">
        <p class="font1">SHA256</p>
        <p
class="font2">F710BF78A1948AA9DD9F7090192DA527097C
05B9C483F455A05E93292D91AE27</p>
    </div>
</div>
```

```
<div class="box">
    <div class="title">English</div>
    <div class="button"><a
href="ue_english_64.exe">Download 64-bit</a></div>
    <div class="box-footer">
        <p class="font1">SHA256</p>
        <p
class="font2">F710BF78A1948AA9DD9F7090192DA527097C
05B9C483F455A05E93292D91AE27</p>
    </div>
</div>
```

```
<div class="box">
    <div class="title">English</div>
    <div class="button"><a
href="ue_english_64.exe">Download 64-bit</a></div>
    <div class="box-footer">
        <p class="font1">SHA256</p>
        <p
class="font2">F710BF78A1948AA9DD9F7090192DA527097C
05B9C483F455A05E93292D91AE27</p>
    </div>
</div>
```

```
<div class="box">
    <div class="title">English</div>
    <div class="button"><a href="ue_english_64.exe">Download 64-bit</a></div>
    <div class="box-footer">
        <p class="font1">SHA256</p>
        <p
            class="font2">F710BF78A1948AA9DD9F7090192DA527097C
            05B9C483F455A05E93292D91AE27</p>
    </div>
</div>
```

```
<div class="box">
    <div class="title">English</div>
    <div class="button"><a href="ue_english_64.exe">Download 64-bit</a></div>
    <div class="box-footer">
        <p class="font1">SHA256</p>
        <p
            class="font2">F710BF78A1948AA9DD9F7090192DA527097C
            05B9C483F455A05E93292D91AE27</p>
    </div>
</div>
```

```
<div class="box">
    <div class="title">English</div>
    <div class="button"><a href="ue_english_64.exe">Download 64-bit</a></div>
    <div class="box-footer">
        <p class="font1">SHA256</p>
        <p
            class="font2">F710BF78A1948AA9DD9F7090192DA527097C
            05B9C483F455A05E93292D91AE27</p>
    </div>
</div>
```

```
</main>
```

```
</body>  
</html>
```

even I did what you told me to do in box-footer, it still is shifted towards left why? and not centred?

It seems like the `box-footer` is still not centered because the parent `.box` might not be behaving as expected, or the `width: 80%` in `.box-footer` isn't behaving well with the `margin: 8px auto`.

Here's what's likely happening:

1. **Parent container `.box` layout:** Since the `.box` uses `display: grid`, the `box-footer` element is being placed according to the grid layout. The `box-footer`'s `width: 80%` is working, but it may not fully align because of the grid layout, especially if it's not aligned to the center.
2. **`width: 80%` on `.box-footer`:** This makes the `.box-footer` take up 80% of the width of its parent, which can cause it not to be exactly centered within the `.box`. The `margin: 8px auto` will horizontally center it, but because `width: 80%` is set, the centering may not work if the parent has padding, extra content, or other constraints affecting it.

## Try these adjustments:

1. **Use `display: flex` for `.box`:** Instead of using `grid` for `.box`, try using `flex` to center its content properly.
2. **Fix `.box-footer` alignment:** If you want to keep using `grid` for `.box`, then you should ensure that the `.box-footer` itself is centered. Also, you may want to consider using `auto` for the width of `.box-footer` to allow it to naturally size to its content.

Here's a revised version of your code with these suggestions:

css

```

.box {
  display: flex; /* Use flexbox instead of grid for centering */
  flex-direction: column; /* Ensure items are stacked vertically */
  justify-content: center;
  align-items: center; /* Center everything inside */
  background-color: #F9F9F9;
}

.box .box-footer {
  width: auto; /* Let it auto-size to content */
  margin: 8px auto; /* Center the footer */
  word-wrap: break-word;
  overflow-wrap: break-word;
  text-align: center; /* Ensure text inside is also centered */
}

```

## Explanation:

- Flexbox on `.box`**: Using `display: flex` will center the `.box-footer` and its contents properly.
- `width: auto` on `.box-footer`**: This allows the footer to adjust its width based on its content and will help center it within the `.box`.
- `text-align: center`**: Centers the text inside `.box-footer`, ensuring the content is also aligned.

Try this and check if it centers as expected!

hey can I use translate by any chance, will it make it responsive?

Yes, you can use `translate` to center the `.box-footer`, and it can still work responsively, though with some caveats.

## How `translate` works:

- The `transform: translateX(-50%)` (for horizontal centering) will shift the element to the left by 50% of its width, making it effectively centered.
- This works well with elements that have a defined `position`, like `absolute`, `relative`, or `fixed`.

## Using `translate` for responsive centering:

Here's how you can modify your `.box-footer` using `translate`:

1. Set the element to `position: relative` (or `absolute` depending on the layout).
2. Apply `transform: translateX(-50%)` to center it horizontally, which makes it responsive.

### Example:

css

```
.box .box-footer {  
    position: relative; /* Or absolute depending on your layout */  
    margin: 8px 0; /* Optional: for vertical margin */  
    width: 80%; /* Or any other width */  
    transform: translateX(-50%); /* Shift it left by 50% of its own width */  
    left: 50%; /* Move it 50% from the left of its container */  
    word-wrap: break-word;  
    overflow-wrap: break-word;  
    text-align: center; /* Center text inside the footer */  
}
```

### Explanation:

- `position: relative;` (or `absolute` if necessary) allows the `transform` property to work.
- `left: 50%;` moves the footer to the center of the parent container.
- `transform: translateX(-50%);` shifts the footer to the left by 50% of its width, perfectly centering it.

### Will it work responsively?

Yes! The `transform: translateX(-50%)` approach is responsive because it works based on the element's width. As the screen size changes, the element's width will adjust (for example, if you use percentage widths or other responsive techniques), and `translateX(-50%)` will ensure the element stays centered.

### Conclusion:

Using `translate` with `position` and `left` is an effective way to center elements and it remains responsive. It's especially useful when you don't want to rely solely on `margin`:

`auto` or `flexbox / grid`, and it works well in dynamic layouts.

btw earlier you wrote `width:100%`  
and then you did `margin auto`

but you mentioned `auto` for margin takes the leftover space  
that's why I kept the width 80%

Yes, you're correct to think about how `width: 100%` and `margin: auto` work together! Let me explain both scenarios more clearly and why I suggested different values.

### When using `width: 100%` and `margin: auto`:

- `width: 100%` makes the element stretch to occupy the full width of its parent container.
- `margin: auto` (when set on both left and right) will divide the remaining space equally on both sides, centering the element. However, because the element is already 100% wide, there's no space left to divide, so it doesn't visibly "center."

This means `width: 100%` combined with `margin: auto` is often not the best for centering, because it doesn't leave any space for the margin to act upon. The element will just take up the entire available width.

### When using `width: 80%` and `margin: auto`:

- `width: 80%` makes the element take up 80% of the width of its parent container, leaving 20% of space on both sides (10% on each side).
- `margin: auto` then divides this leftover space equally on both sides, centering the element.

This is why setting `width: 80%` and using `margin: auto` works better for centering, because it leaves some "extra space" on the left and right to be divided equally by the `auto` margins, making the element centered.

### What happens with `transform: translateX(-50%)`:

When using `translateX(-50%)` (with `position: relative`), you're shifting the element to the left by 50% of its own width, which effectively centers it. This method doesn't require margins and works well for centering without worrying about the available space.

## Conclusion:

- `width: 100%` with `margin: auto` won't give you the expected centering effect because it fills the whole container width.
- `width: 80%` with `margin: auto` works because it leaves space for the margins to divide and center the element.
- `translateX(-50%)` is a more direct way of centering and works regardless of the width, making it a flexible and responsive option.

In your case, `width: 80%` with `margin: auto` is fine for centering, but if you're looking for more flexibility and precision, you can also use `translateX` with `position` and `left`.

```
.box .box-footer{

    /* auto when put for dimensions like width and
       height "Take up all available horizontal space from the
       parent (minus padding/margins)."

    So if a parent has not padding then the child with
    auto take the full space of that parent

    As, for margin and padding shorthand auto takes
    the leftover space and divides it equally between the two
    sides (either vertically or horizontally depending if we use
    vertical or horizontal padding/margin, which actually
    centres the element)

    */
    /* that's why I wrote the below two lines to centre
       the box-footer within the the box */
    margin: 8px auto;
    width: 80%;

    /* for some reason maybe because of grid the above
       two lines were written to center the footer but it didn't
       happen and then translate saved everything(its actually
       very useful) */

    transform: translateX(18%);

    /* Either of below two line is sufficient to do word-
       wraping inside div which prevents overflow */
    word-wrap: break-word;
    overflow-wrap: break-word;
    /* They both work the same, its just the former one
       is older syntax and the latter one is newer syntax (so some
       browser which still support the older syntax may use the
```

```
former one although most of them will use the latter one)  
*/  
}
```

I hope I wrote write, do letme know I missed somethinf

Your explanation is mostly spot-on, but there are a few things I would clarify and suggest improving to make your code even more efficient and clear:

## About centering with `margin: auto`:

Your understanding of `margin: auto` is correct: it divides the remaining space equally on both sides of the element. However, when you use `width: 80%`, it ensures that the `box-footer` element will take up 80% of the parent's width, and `margin: auto` will center it within that parent. This is generally a reliable way to center elements.

## The `transform: translateX(18%)` part:

This is useful for shifting the element by a percentage of its width. However, this value ( `18%` ) may not always be ideal because it depends on the layout and size of the parent container. Using `translateX` is more of a manual fix. If centering is the main goal, it's better to rely on `margin: auto` and `width` or use `flexbox` / `grid` properties for centering, but if you need to tweak the positioning further, `translateX` can come in handy.

## `word-wrap: break-word` and `overflow-wrap: break-word`:

- `word-wrap: break-word` is the older property, while `overflow-wrap: break-word` is the newer standard.
- Most modern browsers support the newer syntax ( `overflow-wrap` ), but for legacy support, it's safe to include both. However, you don't necessarily need both, as `overflow-wrap: break-word` is generally enough to prevent overflow and break long words.

## Suggested Improvements:

css

```
.box .box-footer {  
  margin: 8px auto;  
  width: 80%;  
  /* To center the box-footer within the box, transform may not be needed unless */
```

```

you want to adjust further */
/* transform: translateX(18%); (Only use if necessary) */

/* Word wrapping */
overflow-wrap: break-word; /* Newer, preferred syntax */
word-wrap: break-word; /* Older syntax for compatibility */
}

```

## Key takeaways:

- `margin: auto` and `width: 80%` will center your footer properly.
- `transform: translateX(18%)` is not needed for centering, but if you have a specific design requirement, it can shift the element by a certain percentage of its own width.
- `overflow-wrap` is sufficient for handling word breaks and preventing overflow, so you don't need both properties unless you need extra compatibility.

Let me know if this clears things up or if you need further help!

↳ \* \*\*\*`margin: auto`\*\* and \*\*`width: 80%`\*\* will center your footer properly.

but it didn't why?

If `margin: auto` and `width: 80%` did not center your footer, there could be a few reasons why this happened. Let's go over some potential issues and solutions:

## 1. Parent Element's Width:

- `margin: auto` centers the element horizontally within its **parent container**. If the parent element (in this case, `.box`) does not have a defined width or takes up 100% of the screen width, `margin: auto` may not work as expected.

**Solution:** Make sure the parent container (`.box`) has enough width (or is set to 100% width). You can explicitly define the width for the `.box` or rely on the grid layout to take care of it.

## 2. Grid Layout:

- Since you're using **CSS Grid** for the layout of the `.box`, it might affect how margins are applied. When you use a grid container, the child elements may align based on grid layout rules, and margin behavior might be different compared to using Flexbox or block layout.

**Solution:** Check if `.box` is properly utilizing grid properties. You might want to ensure `.box` has sufficient width and is properly aligned. You could also try using `display: flex` or `text-align: center` for better centering, or adjust the `grid-template-areas` to see if this affects the footer centering.

### 3. Content Overflow or Padding:

- If the `.box-footer` has padding, large text, or content that exceeds the available width, it can affect the centering, especially if you're using `width: 80%` and there is not enough space inside the parent container.

**Solution:** Ensure there's no content overflow or that the padding inside `.box-footer` does not push it outside its container. Check if the content is too wide and adjust accordingly.

### 4. `transform: translateX(18%)`:

- If you are still using `transform: translateX(18%)` (as you mentioned earlier), this can override the centering effect from `margin: auto`. This is because `transform` physically moves the element by a set percentage, which could break the centering.

**Solution:** Try removing `transform: translateX(18%)` temporarily and see if it improves the centering. The combination of `margin: auto` and `width: 80%` should be sufficient for centering the element in most cases.

---

## Suggested Debugging Steps:

1. Ensure the parent container (`.box`) has proper width and grid setup.
2. Try using `display: flex` inside `.box-footer` to control alignment:

css

```
.box .box-footer {  
    display: flex;  
    justify-content: center; /* Centers the content */  
    align-items: center;  
    margin: 8px auto;  
    width: 80%;  
}
```

3. Temporarily remove the `transform: translateX(18%)` to check if it resolves the issue.
4. Inspect the page with developer tools to see if any unexpected padding or overflow is causing misalignment.

Try these adjustments, and if the issue persists, let me know the result of each step!

```
/* remember that if width:auto or width: 100% then it will  
not work as the parent doesn't have any padding so using  
these two properties is practically the same a sauto will try  
to occupy as much space as possible inside the parent (and  
here max-space is actually 100% only(due to no padding))  
and for 100% space auto for padding/margin doesn't work  
as then there would be no leftover space to divide so the  
element won't be visually centred */  
margin: 8px auto;  
width: 80%;  
/* for some reason m
```

I also added this

It seems like you're explaining a very important concept here: when using `margin: auto` for centering, the parent element needs to have some "remaining space" to distribute the margin. If the parent takes up the full available width (like with `width: 100%`) and has no padding, then there will be no leftover space to distribute the margins equally on both sides, which prevents centering.

## Here's a breakdown of your explanation:

### 1. When `width: auto` or `width: 100%` is applied:

- The child element will try to take up as much space as possible inside the parent, and if the parent itself is already taking up 100% of the available width, the child will also take up 100%.
- Without any padding in the parent, there will be no "leftover space" to apply the auto margins, so it won't be visually centered.

### 2. To fix this, you are correctly setting `width: 80%`:

- By limiting the child element's width to 80%, you allow the parent to have space on either side of the child element. This leftover space is then split equally by `margin: auto`, which achieves centering.

## The key idea:

- `width: 100%` or `width: auto` on a child element does not work well with `margin: auto` for centering unless the parent has some available space (like padding). That's why you're setting `width: 80%` to create space for the margins to take effect.

## Additional Tip:

- If your parent container (`.box`) doesn't have any padding, you could consider adding some internal padding (for example, `padding: 0 10px;`) to ensure the child element has some breathing room on either side.

In summary, your approach is correct: `margin: auto` works when there is leftover space in the parent container, and setting the child's width to 80% ensures there is space for this to happen. The added explanation is spot on!

```
nav{  
    display: flex;  
  
    /* justify-content: space-evenly; */  
    /* The above line is good but you would not have  
       precise control over margin, paadding or gap */  
  
    justify-content: center;  
    /* This line doesn't add any custom space so you can  
       have more control over the three parameters here */  
    gap: 0.5%;  
}  
  
nav ul{  
    list-style: none;  
    display: flex;  
    gap: 20px;  
}  
nav ul a{  
    text-decoration: none;  
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana,  
    sans-serif;  
    font-size: 16px;  
    font-weight: 400;
```

```
    line-height: 14px ;  
    color: #4b4b4b
```

```
}
```

```
.options{  
    display: flex;  
    align-items: center;  
    justify-content: center;  
    /* no need to set height explicitly as flex-box  
    automatically matches the tallest (if we don't want then we  
    can set the height explicitly too) */
```

```
    margin: 5px ;  
    margin-left: 19%;
```

```
}
```

```
.options li{  
    padding: 2px;  
  
    cursor: pointer;  
    /*This actually changes the cursor to hand whenever  
    it enters in this div (Explored from harry's solution) */
```

```
    display: flex;  
    align-items: center;  
    justify-items: center;  
}
```

```
.other-icons{  
    display: flex;  
    align-items: center ;  
    justify-content: center;
```

```
}
```

```
.other-icons img{  
    height: 45px;  
}
```

```
nav img[src="icon.png"]{  
    height: 92px;  
    margin-left: 1vw;  
}  
  
nav img[src="sub-icon.png"]{  
    height: 12px;  
}
```

how can I make this nav bar responsive too?

To make the navbar responsive, you need to consider how it behaves on smaller screens. Typically, a responsive navbar involves making sure the navigation links (and other elements) adjust based on the screen size. Here's how you can modify your CSS to achieve responsiveness:

## Key Adjustments:

- 1. Use of Media Queries:** You can use media queries to adjust the navbar's layout on different screen sizes.
- 2. Flexible Layout:** Your current flexbox approach is good for a flexible layout, but you may need to adjust it based on screen width.
- 3. Stacking Links on Small Screens:** You might want to stack the navigation items vertically on smaller screens.
- 4. Hamburger Menu:** Often, a hamburger menu is used for smaller screens, but that would require a bit of JavaScript to toggle the menu.

## Example Implementation:

css

```
/* Base styles for desktop and large screens */  
nav {  
    display: flex;  
    justify-content: center;  
    gap: 0.5%;  
}  
  
nav ul {  
    list-style: none;
```

```
display: flex;
gap: 20px;
}

nav ul a {
    text-decoration: none;
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    font-size: 16px;
    font-weight: 400;
    line-height: 14px;
    color: #4b4b4b;
}

.options {
    display: flex;
    align-items: center;
    justify-content: center;
    margin: 5px;
    margin-left: 19%;
}

.options li {
    padding: 2px;
    cursor: pointer;
    display: flex;
    align-items: center;
    justify-items: center;
}

.other-icons {
    display: flex;
    align-items: center;
    justify-content: center;
}

.other-icons img {
    height: 45px;
}

nav img[src="icon.png"] {
    height: 92px;
    margin-left: 1vw;
}
```

```

}

nav img[src="sub-icon.png"] {
    height: 12px;
}

/* Add media queries for responsiveness */
@media (max-width: 768px) {
    /* Stack nav links vertically on smaller screens */
    nav ul {
        flex-direction: column;
        gap: 10px; /* Adjust the gap between the links */
        align-items: center; /* Center links */
    }

    /* Reduce margin for the options */
    .options {
        margin-left: 0;
    }

    .options li {
        padding: 8px; /* Increase padding for better touch interaction */
    }

    /* Make images smaller */
    nav img[src="icon.png"] {
        height: 50px;
        margin-left: 0;
    }

    nav img[src="sub-icon.png"] {
        height: 10px;
    }

    /* Adjust other styles as necessary for small screens */
}
}

@media (max-width: 480px) {
    /* Further reduce font size and spacing on very small screens */
    nav ul a {
        font-size: 14px;
        padding: 8px;
}

```

```

    }

    .options li {
      padding: 10px;
    }

    /* Optionally add a hamburger menu for very small screens */
    /* Implement JavaScript to toggle visibility of the menu */

}

```

## Key Changes:

- Media Queries (`@media`):** These allow you to define different styles based on the screen width. For example, for screens with a width of `768px` or less, the layout will change to a column layout for the navbar.
- Stacking Links:** `flex-direction: column` in the `nav ul` makes the links stack vertically instead of horizontally on smaller screens.
- Adjusting Margins and Padding:** Reducing margins and padding helps fit the elements properly on smaller screens, making the navbar look more compact.
- Font Size and Image Sizes:** For smaller screens, font sizes and image heights are reduced to ensure everything fits nicely.

## Optional (Hamburger Menu):

If you want to implement a hamburger menu for very small screens (like phones), you can hide the links and show a hamburger icon, which when clicked, toggles the visibility of the navbar links. This would require a bit of JavaScript to handle the toggle action.

Let me know if you'd like to implement that, and I can guide you through the process!

In some designs for for width design, my footer tags over the button in the boxes at the bottom but the above boxes are perfectly fine?

It sounds like the footer element in your design is not being contained properly within the layout, causing it to overlap with the button in the boxes. This issue typically happens when elements are not given enough space or the layout isn't adjusting as expected on different screen sizes or based on the available space.

## Possible Causes and Solutions:

## 1. Footer Height and Overflow:

If the footer is too large or there isn't enough space for it, it can overflow into the content area. You can prevent this by ensuring the footer has enough space, and its content doesn't push over other elements.

### Solution:

- Ensure the footer has `clear: both;` or `overflow: hidden;` to make sure it clears the other content.
- Add a minimum height to the footer or a flexible layout that prevents it from overlapping.

```
css
```

```
.footer {  
    clear: both; /* Makes sure it doesn't overlap content */  
    overflow: hidden; /* Prevents overflowing content */  
}
```

## 2. Absolute or Fixed Positioning:

If you have used `position: absolute` or `position: fixed` for the footer, it might be taken out of the normal document flow and overlap with other content.

### Solution:

- Ensure the footer is positioned relative to its parent and not fixed or absolute unless necessary.

```
css
```

```
.footer {  
    position: relative;  
    width: 100%; /* Ensures it occupies the full width */  
}
```

## 3. Flexbox Layout or Grid Issues:

If you are using Flexbox or Grid layouts, ensure that the footer is within the flow of the layout and isn't positioned outside of the containing box.

### Solution:

- Check if you're using `flex` or `grid` on the parent container and ensure that the footer is within the correct flex or grid cell.

- Make sure there's no `position: absolute` or `position: fixed` on the footer inside a flex/grid container, unless required.

#### 4. Container Sizing:

The container holding the footer and other elements might not be properly sized. If the container doesn't stretch enough to fit the footer, it can overlap with the buttons or other elements below it.

##### Solution:

- Set a `min-height` on the container, so the footer stays at the bottom.
- Alternatively, using `flex-grow` or `flex-shrink` in the parent container can help ensure that space is appropriately distributed.

css

```
.container {
  display: flex;
  flex-direction: column;
  min-height: 100vh; /* Ensures the container takes the full height of the
viewport */
}

.footer {
  margin-top: auto; /* Pushes the footer to the bottom if the content doesn't
fill the space */
}
```

#### 5. Box Model Considerations (Margins and Padding):

Sometimes, margins and padding can push content out of the expected area. If you have large margins on the footer or the content above it, they might not be accounted for properly.

##### Solution:

- Check the margin or padding values on the footer and the button elements.
- If you want the footer to sit at the bottom without overlapping, ensure proper spacing and margin collapse management.

css

```
.footer {  
    margin-top: 20px; /* Adds space between content and footer */  
}
```

## 6. Responsive Layout Issues:

On smaller screens, the layout might not adjust properly, causing the footer to overlap. You may need to add specific styles for smaller screens to ensure the footer remains properly positioned.

### Solution:

- Use media queries to adjust the layout and ensure the footer fits properly on small screens.

css

```
@media (max-width: 768px) {  
    .footer {  
        padding: 10px; /* Adjust padding for smaller screens */  
    }  
}
```

## Example to Fix Overlap:

css

```
/* Container holding content and footer */  
.container {  
    display: flex;  
    flex-direction: column;  
    min-height: 100vh; /* Ensures it takes up full height */  
}  
  
/* Content section (above the footer) */  
.content {  
    flex-grow: 1; /* Makes sure it takes all available space */  
}  
  
/* Footer */  
.footer {  
    clear: both;  
    overflow: hidden;
```

```
margin-top: 20px;  
padding: 20px;  
background-color: #f1f1f1;  
width: 100%;  
}  
  
.button-container {  
position: relative; /* Ensure buttons are inside the container */  
}
```

This layout ensures that the footer will be pushed down to the bottom of the page, and the content will take up the remaining space.

## Recap:

- Ensure that the footer isn't using absolute or fixed positioning unless necessary.
- Use `flex` or `grid` to manage layout and make sure the footer is placed properly at the bottom.
- Ensure proper margins, padding, and overflow handling to avoid content overlap.
- Add media queries to handle small screen adjustments.

Let me know if you need more specific help on any of these!

the thing is that for the boxes in the startin everything is fine but as I move oewn the boxs are getting compressed a bit that causes vertical overflow?

It sounds like you're dealing with **vertical overflow** when the boxes are compressed as you scroll down the page. This issue is typically caused by containers or elements not being properly sized to accommodate their content, especially in a flexible layout.

Here are some things you can check and adjustments you can make to resolve the issue:

### 1. Set Proper Height for Containers:

If your boxes (or containers) are being compressed, they may not have enough height to accommodate their content. Try setting a `min-height` or adjusting their height to ensure they don't shrink too much.

css

```
.box {  
    min-height: 200px; /* Adjust as needed */  
    padding: 20px;      /* Add padding for better spacing */  
    margin-bottom: 10px; /* Add space between boxes */  
}
```

## 2. Use `flex` or `grid` for Proper Layout:

If you're using `flex` or `grid`, you may want to ensure that the layout behaves correctly when the content grows or shrinks. For a column layout, ensure that the container for your boxes can expand and hold the boxes properly.

css

```
.container {  
    display: flex;  
    flex-direction: column;  
    gap: 20px; /* Ensures space between boxes */  
    flex-wrap: wrap;  
    justify-content: flex-start; /* Aligns the content at the top */  
}  
  
.box {  
    flex-grow: 1; /* Allows the box to grow */  
    min-height: 200px; /* Prevents compression */  
    overflow: hidden; /* Prevents overflow inside the box */  
}
```

## 3. Avoid Fixed Heights (if possible):

If you have set fixed heights for the boxes, try removing them or use `min-height` instead to prevent them from shrinking too much and causing overflow.

css

```
.box {  
    min-height: 200px; /* Avoid fixed height */  
    height: auto; /* Let the box expand as needed */  
}
```

## 4. Ensure No Negative Margins:

Sometimes negative margins can cause the boxes to overlap or be pushed out of the container, which might be a cause for vertical overflow. Make sure you're not unintentionally using negative margins.

css

```
/* Ensure there's no unintentional negative margin pushing boxes outside */
.box {
    margin: 10px 0; /* Proper spacing between boxes */
}
```

## 5. Overflow Handling:

If the content inside the boxes is too large, causing overflow, you can control the overflow behavior using `overflow-y` or `overflow` to prevent vertical scrolling or content from spilling out.

css

```
.box {
    overflow-y: auto; /* Allows vertical scrolling within the box */
    max-height: 500px; /* Prevents the box from growing too large */
}
```

## 6. Check for Parent Container Overflow:

Sometimes, the parent container can cause overflow. If the parent doesn't have enough height or space to accommodate the boxes, it might cause them to overflow as you scroll.

For example:

css

```
.parent-container {
    overflow: hidden; /* Hides overflow to prevent scroll */
    padding: 20px;
}
```

Alternatively, use `overflow-y: auto;` if you want to allow scrolling in the parent container.

css

```
.parent-container {  
    overflow-y: auto; /* Allows scrolling if content overflows */  
    height: 100vh; /* Makes sure the parent takes the full viewport height */  
}
```

## 7. Media Queries for Responsive Layout:

If the issue only occurs on certain screen sizes (like mobile), ensure you're using media queries to adjust the layout for smaller screens. You can adjust box size or layout properties dynamically to ensure there's no overflow.

css

```
@media (max-width: 768px) {  
    .box {  
        min-height: 150px; /* Adjust box height for smaller screens */  
    }  
}
```

## 8. Prevent Fixed Footer from Overlapping Content:

If you have a footer and boxes are being compressed by it, make sure the footer doesn't overlap the content. Add margin or padding to the bottom of the last box to prevent this.

css

```
.footer {  
    margin-top: 20px;  
    padding: 20px;  
}
```

## Example Fix for Vertical Overflow:

css

```
.container {  
    display: flex;  
    flex-direction: column;  
    gap: 20px; /* Ensures space between boxes */  
    flex-wrap: wrap;  
    justify-content: flex-start;
```

```

}

.box {
  min-height: 200px; /* Ensures box has sufficient height */
  flex-grow: 1; /* Allows box to expand if needed */
  overflow-y: auto; /* Allows scrolling inside the box if content overflows */
  margin-bottom: 10px; /* Adds space between boxes */
}

/* For the footer or parent container */
.parent-container {
  overflow-y: auto; /* Allows scrolling if content exceeds */
  height: 100vh; /* Ensures it takes the full viewport height */
}

.footer {
  padding: 20px;
  margin-top: auto; /* Pushes the footer to the bottom */
}

```

## Conclusion:

By ensuring your boxes have enough space (using `min-height`), are not over-compressed, and managing overflow with `overflow-y`, `flex` layout properties, and `media queries`, you should be able to fix the vertical overflow issue. Let me know if you need further assistance!

what feels strange is that why it didn't work with all the boxes and only happen to boxes below in mobile phones?

If the issue with overflow only happens with certain boxes (like the ones lower down) on mobile phones, it likely has to do with **layout, container sizing, or content scaling** that behaves differently across screen sizes.

Here are a few possible reasons why this is happening and why it might be affecting only the lower boxes on mobile devices:

### 1. Box Sizing and Flex Layout

If you're using `flex` or `grid`, there might be a situation where the content in the lower boxes is growing more than the available space on smaller screens. Since the top boxes might have enough space, they aren't affected, but the lower boxes could overflow because:

- On smaller screens, the container is shrinking, and there's less available space to fit all the boxes.
- The `flex` layout might be causing the last boxes to overflow if their content is too large and the parent container is not resizing properly.

## Solution:

Try to use `flex-grow`, `min-height`, and `max-height` to control how the boxes grow/shrink in a responsive manner.

css

```
.container {
  display: flex;
  flex-wrap: wrap;
  justify-content: flex-start;
  gap: 20px; /* Space between boxes */
}

.box {
  flex: 1 1 200px; /* Flex-grow: 1, Flex-shrink: 1, Basis: 200px */
  min-height: 200px;
  max-height: 300px; /* Prevents box from growing too large */
  overflow-y: auto; /* Allows scrolling if content overflows */
}
```

## 2. Viewport Height (vh) Issues on Mobile

Mobile browsers treat `vh` units (viewport height) differently, especially when the address bar is visible or when the user scrolls. This might cause your boxes to not fit the screen properly.

- If you used `height: 100vh` for any of your containers or boxes, it can be problematic because mobile browsers reduce the available height when the user scrolls, causing unexpected overflow.

## Solution:

Use JavaScript to set the height dynamically for mobile screens or adjust the layout so it adapts better to these changes:

css

```

@media (max-width: 768px) {
    .box {
        min-height: auto; /* Adjust for smaller screens */
    }
}

@media (max-width: 768px) {
    .container {
        height: calc(100vh - 50px); /* Adjust height for smaller screens */
    }
}

```

Or use JavaScript to dynamically calculate the height:

```

javascript

if (window.innerWidth <= 768) {
    document.querySelector('.container').style.height = window.innerHeight + 'px';
}

```

### 3. Padding and Margin Issues on Smaller Screens

Margins or paddings on smaller screens might cause the boxes to not fit properly. The vertical overflow could happen if the total height of the boxes (including margin and padding) exceeds the available space on mobile.

#### Solution:

Reduce the margin/padding on mobile screens using **media queries** to make sure the content fits within the available space.

```

css

@media (max-width: 768px) {
    .box {
        margin: 5px;
        padding: 10px; /* Adjust padding for mobile */
    }
}

```

### 4. Content Overflowing Inside the Box

If the content inside the boxes (like text, images, etc.) is too large, it could push the box beyond its height, especially if you're using fixed heights or not setting `overflow` properly.

## Solution:

Ensure that the boxes have `overflow-y: auto` or `overflow: hidden` to prevent content from spilling out.

css

```
.box {  
  overflow-y: auto; /* Ensures content scrolls if it overflows */  
  height: 200px; /* Fixed height or max-height */  
}
```

## 5. Flexbox Wrapping and Column Layout

If the boxes are in a flex container with a column layout, the available space on mobile screens may not be enough for all the boxes, causing overflow in the lower boxes while the top ones remain fine.

## Solution:

Use `flex-wrap: wrap` to ensure the boxes wrap properly on smaller screens.

css

```
.container {  
  display: flex;  
  flex-wrap: wrap; /* Ensures the boxes wrap to a new line on smaller screens */  
  justify-content: flex-start;  
}  
  
.box {  
  flex-basis: 100%; /* Forces each box to take full width on smaller screens */  
  padding: 10px;  
}
```

## 6. Media Queries Not Properly Handling Small Screens

Sometimes, media queries for mobile screens may not be set properly, or they may only target certain screen widths. This can result in the lower boxes not adapting correctly to the screen size, especially if you have a lot of content in them.

## Solution:

Ensure your media queries are covering the entire mobile screen range and adjusting the layout properly for all screen sizes.

css

```
@media (max-width: 768px) {  
    .container {  
        flex-direction: column; /* Ensure flex items stack on mobile */  
    }  
  
    .box {  
        width: 100%; /* Make boxes full width on mobile */  
        margin: 10px 0;  
    }  
}
```

## Conclusion:

To fix this issue, focus on the following:

1. Ensure proper layout with `flex-wrap`, `flex-grow`, and `min-height`.
2. Avoid using `vh` for height if it's causing issues on mobile.
3. Use media queries to adjust margins, padding, and box layout for smaller screens.
4. Ensure content inside boxes doesn't overflow by using `overflow-y: auto`.

By doing this, the lower boxes should behave more consistently on mobile screens without overflowing. Let me know if you need further help!

can't i use auto fit in template columns too?

Yes, you can use `auto-fit` in `grid-template-columns` to make your layout more responsive. This is a great approach if you want to automatically adjust the number of columns based on the available space.

Here's how it works:

`auto-fit` in `grid-template-columns`:

- `auto-fit` automatically adjusts the number of columns to fit the available space.

- It will expand the columns to fill the space, but if there's extra space, it will keep the column widths flexible.

## Example:

If you want to create a grid where the number of columns automatically adjusts based on the available width, you can use `auto-fit` along with `minmax()`. This allows you to set a minimum and maximum column width.

css

```
.container {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));
  gap: 20px; /* Adjust space between items */
}
```

## Explanation:

- `repeat(auto-fit, minmax(200px, 1fr))`:
  - `auto-fit`: This tells the grid to fit as many columns as possible based on the container's width.
  - `minmax(200px, 1fr)`: This defines the minimum and maximum width of each column. Each column will be at least `200px` wide but can expand to `1fr` (1 fractional unit) if there's available space.

## How `auto-fit` works:

- As the screen width changes, the grid will automatically adjust the number of columns to fit the space.
- On smaller screens, the columns will shrink to fit, and on larger screens, they will expand.
- This is especially useful for creating responsive designs that adapt to different screen sizes.

## Example with a grid of boxes:

html

```
<div class="container">
  <div class="box">Box 1</div>
  <div class="box">Box 2</div>
  <div class="box">Box 3</div>
  <div class="box">Box 4</div>
  <div class="box">Box 5</div>
  <div class="box">Box 6</div>
</div>
```

css

```
.container {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));
  gap: 20px; /* Space between grid items */
}

.box {
  background: lightblue;
  padding: 20px;
  border: 1px solid #ccc;
}
```

In this example:

- The grid will display as many columns as possible, with each column having a minimum width of `200px`.
- The number of columns will adjust automatically based on the screen size (i.e., the available space).

## When to Use `auto-fit`:

- Use `auto-fit` when you want the layout to be flexible and responsive, adjusting the number of columns as needed based on the screen width.
- It's especially useful when you don't know the exact number of items in your grid, or when you want the grid items to scale smoothly across different screen sizes.

## `auto-fill` vs `auto-fit`:

- `auto-fit`: It will fill the grid with as many columns as possible, even if some columns are empty.

- `auto-fit`: It behaves similarly but will collapse the columns if there's insufficient content to fill them.

For most cases where you're building responsive grids, `auto-fit` is the preferred option, as it makes the layout more adaptive.

u sais min width says thst 200px will be the min width and it will strat shrinkinf after it but what does it mean by max-width and what happen if its exceeded?

Great question! Let's break it down:

### `minmax(min, max)` in CSS Grid:

- `minmax(min, max)` is used to set a range for the size of grid items (or columns in the case of `grid-template-columns` ).
- `min` defines the **minimum size** the grid item should be.
- `max` defines the **maximum size** the grid item can grow to.

### How does it work?

- The **minimum value** (`min`) ensures that the grid item doesn't shrink below a certain width.
- The **maximum value** (`max`) limits how much the grid item can grow.

### Example:

css

```
grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));
```

Here:

- `200px` is the minimum width the grid column can be. If the container gets smaller than this, the column will stop shrinking.
- `1fr` is the maximum width the column can grow to. It represents a flexible unit that takes up the remaining available space in the grid. As long as there's space, the column will expand to fill it.

### What happens if the `max` value is exceeded?

If the grid container has more space than the `max` value (i.e., the available width exceeds the `max` value you set), the grid item will **stop growing** beyond the `max` value. In the case of `1fr`, it won't stretch beyond the available space divided by the number of columns. The grid item will remain within the maximum limit you set.

Let's illustrate with an example:

#### Example 1: `minmax(200px, 300px)`

css

```
grid-template-columns: repeat(auto-fit, minmax(200px, 300px));
```

- **Minimum width:** `200px`
- **Maximum width:** `300px`

In this case:

- If the container is **smaller than `200px`**, the column will **shrink to `200px`**.
- If the container is **larger than `300px`**, the column will stop growing at `300px`.
- Between `200px` and `300px`, the column can expand or contract based on the available space.

#### Example 2: `minmax(200px, 1fr)`

css

```
grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));
```

- **Minimum width:** `200px`
- **Maximum width:** `1fr` (the column can grow but is limited by the space available)
- If the container has enough space, the column will grow to take up the available space (but no more than the space allocated to it).
- If the container is narrow, it will shrink down to `200px` (minimum width).

## What happens if it's exceeded:

If the **maximum value** is exceeded:

- The grid item will **stop growing**. It will stay at the maximum value even if there's more space.

- For example, if the container is 1000px wide, but the column's maximum width is 300px, the column will remain at 300px, and the remaining space will be divided among the other columns (if any).

## In summary:

- `min` : The grid item will not shrink smaller than the minimum size (e.g., `200px` ).
- `max` : The grid item will not grow larger than the maximum size (e.g., `300px` ), even if the container has more space.
- If the available space is larger than the `max` value, the grid item will stay at its `max` value and won't stretch beyond it.

so when will coluns start multiplying?

Columns in CSS Grid will start "multiplying" or adding more columns when there is enough space in the container to accommodate additional columns based on the `auto-fit` or `auto-fill` behavior combined with the `minmax()` function or any other set column size.

## Here's how this works:

### 1. `auto-fit` vs `auto-fill`:

- `auto-fit` : It attempts to fit as many columns as possible into the available space, but if there's extra space, it will collapse any empty columns (if they're not explicitly sized).
- `auto-fill` : It will create as many columns as it can, even if some columns are empty, and fill the extra space by making the columns as wide as possible. The difference is that `auto-fill` keeps the extra empty columns.

### 2. When do columns start multiplying?

The columns will start multiplying when the available space in the grid container allows for more columns than the set number. This is based on the size of each column, determined by the `minmax()` or any other size value like `fr` (fractional units).

## Example 1: Using `auto-fit` with `minmax()`

css

```
grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));
```

- `auto-fit` will add columns to the grid, each at least `200px` wide, but it can expand each column up to `1fr` (taking up any leftover space).
- If the container width is large enough, more columns will be added.
- As the container shrinks, the columns will either stay at `200px` or shrink until they fit (but won't go smaller than `200px` due to `minmax(200px, 1fr)` ).
- When the container is wide enough, it will fit as many `1fr` columns as the container allows, creating multiple columns that grow and shrink based on the available space.

### Scenario:

- **Container width: 1200px**
  - With `minmax(200px, 1fr)` and `auto-fit`, it will fit as many `1fr` columns as possible within the 1200px.
  - You could fit **6 columns** (if each column takes up `200px` minimum width).
- **Container width: 800px**
  - The columns will shrink but stay at least `200px` wide. If there's not enough space for 6 columns, the number will drop to fit the available space.

### Example 2: Using `auto-fill` with `minmax()`

css

```
grid-template-columns: repeat(auto-fill, minmax(200px, 1fr));
```

- `auto-fill` behaves similarly but keeps adding empty columns if there's more space, even if they are empty.
- If you have extra space, it will keep adding columns (even empty ones) to fill up the space.
- This can lead to **more columns than you need**, but it ensures that the columns occupy the full width of the container.

### Scenario:

- **Container width: 1200px**
  - The grid will try to fit as many columns as possible, but if there's excess space, empty columns will be added.
- **Container width: 800px**

- The grid will add fewer columns, but some might be smaller (no less than 200px).

## Key Points:

- **Columns start multiplying** (i.e., adding more columns) when the container is wide enough to fit more columns based on your column sizing settings.
- `auto-fit` and `auto-fill` help manage the number of columns dynamically based on the available space.
- The actual number of columns that appear will depend on:
  - The **width of the container**.
  - The **minimum size** you set (e.g., `minmax(200px, 1fr)`).
  - The **flexibility** of the columns, determined by the `1fr` (fractional unit) or other sizing methods.

In summary, columns will keep multiplying (i.e., creating new columns) as long as there is enough space in the container. When there's no more space, the columns stop multiplying, and their size adjusts according to the `minmax` range or the set width.

oh so that means that even if have 2 columns at a particular screen size those two will be definitely of 1fr width since auto-fit multiplies on the basis of max width?

Not quite! Let me clarify:

## What Happens with `auto-fit` and `1fr`:

When you use `auto-fit` in combination with `1fr` (fractional units) in **CSS Grid**, it tries to fit as many columns as possible based on the available container width. But there's a subtle difference between the `minmax()` function and the **fractional unit** (`fr`). Let me explain how this works in different scenarios:

## Example:

css

```
grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));
```

In this case:

- `minmax(200px, 1fr)` : This means that each column should be at least 200px wide (the minimum), but it can expand up to `1fr`, which is a fraction of the available space.

## Scenario 1: Container width is large enough to fit multiple columns

- Available space: 1200px (for example)
- Since you have `auto-fit`, it will try to fit as many columns as it can. Each column can grow up to `1fr` (which takes equal space).
  - Columns will each take up `1fr`, meaning they will expand to share the available space equally.
  - If the space can fit, say, 6 columns (each being at least `200px` but can expand up to `1fr`), each column will take 1/6th of the container's total width.

## Scenario 2: Container width is just enough for 2 columns

- Available space: 400px (for example)
- The grid will fit as many columns as it can. Since each column has a **minimum width of 200px**, it will try to fit **2 columns**.
  - In this case, both columns will have the width of `200px` (since it can't expand beyond the minimum width in this case).
  - They will not expand to `1fr` in this case because there isn't enough space for them to expand further.

## Scenario 3: Container width is too small to fit the minimum column size

- Available space: 300px (for example)
- Since the minimum width for each column is `200px`, the grid will fit **only 1 column** and it will take up `200px` of the available space.
  - The remaining space will either be empty, or if you apply any additional styling like `justify-content` or `align-items`, it could be used for alignment.

## Key Takeaways:

1. `auto-fit` will try to fill the available space with as many columns as possible based on the available width.
  - If the available width can accommodate more columns, it will fit more columns, each of which could take up a fraction (`1fr`) of the available space.
  - However, the **minimum width** (e.g., `200px`) is always respected — no column will shrink below that width.
2. If you only have 2 columns, and the container is wide enough for them to take the available space, each column will expand to take an equal share of the available space

(since they are `1fr`). But they **will not expand beyond** the container width or shrink below the `200px` minimum (as set in the `minmax()`).

### 3. When the container is narrower:

- If the container is narrow enough that two columns don't fit with the minimum width (`200px`), the number of columns will drop, and they will take up their **minimum width of `200px`**.
- This ensures columns don't get too small but also prevents overflow.

In short, `auto-fit` **doesn't multiply columns based on the maximum width** (the `1fr` part). It multiplies columns based on the available space, ensuring each column is at least the minimum width you set (e.g., `200px`). If there's extra space, it divides that space equally among the columns (each getting `1fr`).

So that means that the container or out of it works according to the range between min width and max width and this range can be anything and if the length of the screen allows the boxes to be divided equally in a number that lies between this range of 200 pixels and one apart then it will divide those boxes in such a way right?

Exactly, Vaibhav — you've got the core idea spot on! Let me just refine and reinforce your understanding with precise terms:

### ✓ You're Right:

The **container** (or grid parent) will try to **fit as many columns as possible** such that each column's width **lies within the range defined by `minmax(min-width, max-width)`**.

For example:

css

```
grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));
```

Here's what happens:

## How It Works — Step by Step:

### 1. Minimum Size ( 200px ):

- A column can **never be narrower** than `200px`.
- If the container is too small to fit multiple columns with at least `200px` width, it will **reduce the number of columns**.

### 2. Maximum Size ( 1fr ):

- If extra space is available (after the minimum widths are fulfilled), columns will **stretch to share** that space **evenly**, up to the `1fr` max.

### 3. Number of Columns is Determined By:

- How many times the container's width can **accommodate a column of at least 200px**.
- So yes — it checks the screen/container width, and if it can fit, say, 3 columns with  $\geq 200\text{px}$  width, then:
  - You get 3 columns, and they stretch equally to take the full width.

### 4. Responsive Behavior:

- On wider screens: more columns (each at least `200px`, then expanded evenly).
- On narrower screens: fewer columns, but each still keeps at least `200px`.

---

## Visual Example:

If screen/container width is:

- **1200px** → `auto-fit` gives: `200px 200px 200px 200px 200px 200px` (6 columns), then they **expand** to fill extra space using `1fr`.
- **650px** → only **3 columns** of at least `200px`, each takes  $\sim 33\%$  of space.
- **350px** → only **1 column**, takes up 100% of width (as 200px is the min).

---

## Final Summary:

The grid **respects the min width** strictly, and the **max width (like 1fr)** is only used to **distribute extra space** when it's available. So yes, the layout responds smartly based on the range between min and max, and adjusts how many columns to fit while keeping each within that range.

Would you like a quick visual mockup to make it clearer?

does this sound good?

```
grid-template-rows: repeat(auto-fit, minmax(205px, 1fr));
```

Yes, Vaibhav — that line of code is **syntactically correct** and **works well** for making rows responsive:

CSS

```
grid-template-rows: repeat(auto-fit, minmax(205px, 1fr));
```

However, it's **unusual** to use `auto-fit` with **rows**. Here's why:

## What You Should Know:

- `auto-fit` is **commonly used with columns** (`grid-template-columns`), because **horizontal space** in layouts is more fluid and responsive.
- With `grid-template-rows`, you **usually don't** need `repeat(auto-fit, ...)` because:
  - Rows grow **automatically** to fit content.
  - Rows don't "wrap" like columns — the grid grows downward naturally.

## When This Is Okay:

You can use `grid-template-rows: repeat(auto-fit, minmax(205px, 1fr));` if:

- You have a **fixed-height container**, and you want **multiple rows** inside it to **auto-fill vertically** (like a calendar or grid dashboard).
- You want each row to be **at least 205px tall**, and stretch to share space evenly.

## But Avoid If:

- Your container's height is **not fixed**, or content inside rows varies in height — then it might cause layout issues.

---

## Safer Alternative (for most use cases):

Let rows grow naturally:

css

```
grid-auto-rows: minmax(205px, auto);
```

This tells each row:

- **Minimum height** = 205px
- **Grow naturally** based on content

---

## Final Verdict:

Yes, your syntax is valid — just make sure the **use case** needs evenly-distributed, auto-fitted rows. Otherwise, consider `grid-auto-rows` for a more natural layout.

Would you like me to review a specific layout you're working on with this?

grid-template-rows: minmax(205px, auto); I did this?

Perfect, Vaibhav — using this:

css

```
grid-template-rows: minmax(205px, auto);
```

is **technically valid**, but it's slightly **misleading** in its intent. Here's why:

---

## What This Does:

You're telling the **entire row structure** to be:

- **Minimum height:** 205px
- **Maximum height:** as much as needed by content (`auto`)

BUT — this **applies to only one row**, not multiple rows, unless you're explicitly repeating rows.

---

## If You Want Multiple Rows to Behave Like That:

Use this instead:

css

```
grid-auto-rows: minmax(205px, auto);
```

Why?

- `grid-template-rows` defines **explicit rows** (like first row, second row...)
  - `grid-auto-rows` tells the grid **how to size rows that come automatically** as content fills the grid
- 

## Quick Comparison:

Property	Behavior
<code>grid-template-rows: minmax(205px, auto)</code>	Only the <b>first row</b> gets this setting
<code>grid-auto-rows: minmax(205px, auto)</code>	All <b>implicitly added rows</b> follow this height logic

---

## Final Suggestion:

If you're working with **dynamic content** and want **all rows** to be at least 205px, use:

css

```
grid-auto-rows: minmax(205px, auto);
```

Would you like to share your current layout so I can suggest the cleanest setup for your rows and columns together?

so why didn't we use autofit?

Great question, Vaibhav!

We didn't use `auto-fit` in:

css

```
grid-template-rows: minmax(205px, auto);
```

because `auto-fit` only works with columns — not with rows in `grid-template-rows`.

## ✓ Why `auto-fit` doesn't work with rows:

- `auto-fit` and `auto-fill` are used with columns, in `grid-template-columns`, like this:

css

```
grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));
```

This automatically fits as many columns as possible based on the container width.

- But with `rows`, CSS Grid doesn't dynamically "fit" rows the same way, because:
  - Rows grow vertically as needed (since page height isn't usually a fixed limit)
  - CSS assumes unlimited vertical space, so it doesn't need to "fit" things tightly into it

## 🔍 So What Do We Use for Rows?

To size rows smartly, you use:

css

```
grid-auto-rows: minmax(205px, auto);
```

This tells the grid:

“For every new row you create automatically (as items overflow), make it at least 205px tall, but allow it to grow with content.”

## 🧠 Summary:

Concept	Columns	Rows
Auto layout?	Use <code>auto-fit</code> or <code>auto-fill</code>	Not needed (vertical grows freely)
Repeating sizes?	<code>grid-template-columns</code>	<code>grid-auto-rows</code>
Fit as many as possible?	<code>repeat(auto-fit, ...)</code>	Not applicable

Would you like a visual example showing both columns and rows with these settings?

damn it works

Haha yesss, Vaibhav! 😎

That's the power of understanding **how grid behaves with rows vs columns** — once it clicks, layouts become so much smoother to control.

Now you can:

- Use `auto-fit` for dynamic responsive **columns**
- Use `grid-auto-rows: minmax(..., ...)` for flexible **row heights**

Let me know if you want help refining your layout further or adding media queries to make it even more responsive. Want to try stacking boxes in a single column on mobile next?

↳ `grid-auto-rows: minmax(205px, auto);`

but I didn't get it...what if I did repeat(9,  
`minmax(205px, auto))`;

Great follow-up! Let's break it down so it's super clear:

## ❓ What happens when you do:

css

```
grid-template-rows: repeat(9, minmax(205px, auto));
```

You're telling the browser:

→ "I want exactly **9 rows**, each with a minimum height of **205px**, but allow them to grow taller (**auto**) if needed."

So:

- If the content in a row is **less than 205px**, it will still be **205px tall**.
- If the content needs **more space**, the row will **grow taller** to fit it (because of **auto** as the max).
- You will **always** have exactly 9 rows — even if you don't have enough content to fill them.
- This works best when your content has a **fixed or expected row count**.

## ✓ But what you did earlier:

css

```
grid-auto-rows: minmax(205px, auto);
```

This means:

- "Whenever a new row is needed (i.e., as more content is added), make that row at **least 205px tall**, and let it grow taller if needed."
- You don't need to specify a fixed number of rows — it **automatically creates new rows as needed**.

This is **more dynamic** and suited for cases when you:

- Don't know how many rows there'll be.
- Want the layout to **respond to content size**.

## 🚩 Summary

Property	Behavior
<code>grid-template-rows: repeat(9, minmax(205px, auto))</code>	Exactly 9 rows, flexible height per row
<code>grid-auto-rows: minmax(205px, auto)</code>	Auto-generate rows based on content, each flexible

Would you like a visual diagram to make this easier to visualize?

does auto rows cover all the rows ?

Great question, Vaibhav!

✓ Yes, `grid-auto-rows` covers only the rows that are not explicitly defined using `grid-template-rows`.

🔍 Here's how it works:

1. If you define rows manually, like:

css

```
grid-template-rows: 100px 200px;
```

→ The first two rows will be exactly 100px and 200px tall.

2. If more content overflows into new rows, like a 3rd or 4th row...

→ Then `grid-auto-rows` comes into play and defines the height for those extra rows:

css

```
grid-auto-rows: minmax(150px, auto);
```

## 🎯 So, to summarize:

- `grid-template-rows` → controls the **initial, explicitly declared rows**.
- `grid-auto-rows` → controls **any extra rows** that are **implicitly created by content**.

💡 If you want all rows (even the initial ones) to follow the same dynamic behavior:

Use **only** `grid-auto-rows` (and skip `grid-template-rows`), or define all rows using:

css

```
grid-template-rows: repeat(auto-fill, minmax(...));
```

Would you like a quick codepen-style example to see it live?

can you summarize this please as I wanna write proper comment about usig this

Of course! Here's a clean, concise summary you can use as a comment in your CSS:

css

```
/*
'grid-auto-rows' sets the height for rows that are automatically created
when content overflows the defined grid. It does NOT affect rows explicitly
defined using 'grid-template-rows'.
```

*Use 'grid-auto-rows' when you want extra rows to follow a consistent height pattern without manually specifying each one.*

*To apply the same rule to all rows (defined and auto-generated), use 'grid-template-rows' with repeat() and minmax().*

```
*/
```

Would you like a similar summary for `grid-template-columns` and `auto-fit` as well?

also add my speacial case of 9rows to it and mention how it works too

Absolutely! Here's an updated summary that includes your special case of 9 rows:

css

```
/*
'grid-auto-rows' sets the height for rows that are automatically created
when content exceeds the defined grid layout. It only applies to rows
not explicitly defined using 'grid-template-rows'.
```

*In your special case:*

*You defined exactly 9 rows using 'grid-template-rows', so any content that fits within these 9 rows will follow that height rule. If the content exceeds 9 rows (e.g., more items are added), then 'grid-auto-rows' will apply to any extra rows created beyond those 9.*

*To give consistent height to **\*\*all\*\*** rows (both defined and overflowed), you should use a combination like this:*

```
grid-template-rows: repeat(9, minmax(205px, auto));
grid-auto-rows: minmax(205px, auto);
```

*This ensures that both the first 9 rows and any auto-generated rows follow the same height behavior – minimum 205px and expand if needed.*

*\*/*

Would you like help converting this into a more beginner-friendly or one-liner version for quick reference too?

also specify that auto fit and auto fill only works with columns and not rows and grid-auto-rows is only for rows and there is not such grid-auto-columns (I hope I right with the latter one)

You're absolutely correct! Here's the updated summary with the requested additions:

css

```
/*
'grid-auto-rows' sets the height for rows that are automatically created
when content exceeds the defined grid layout. It only applies to rows
not explicitly defined using 'grid-template-rows'.
```

*In your special case:*

*You defined exactly 9 rows using 'grid-template-rows', so any content*

that fits within these 9 rows will follow that height rule. If the content exceeds 9 rows (e.g., more items are added), then 'grid-auto-rows' will apply to any extra rows created beyond those 9.

To give consistent height to \*\*all\*\* rows (both defined and overflowed), you should use a combination like this:

```
grid-template-rows: repeat(9, minmax(205px, auto));  
grid-auto-rows: minmax(205px, auto);
```

This ensures that both the first 9 rows and any auto-generated rows follow the same height behavior – minimum 205px and expand if needed.

**Note:**

- \*\*\*'auto-fit'\*\*\* and \*\*\*'auto-fill'\*\*\* only work with \*\*columns\*\*, not rows.
- \*\*\*'grid-auto-rows'\*\*\* is only for controlling row heights.
- There is no \*\*\*'grid-auto-columns'\*\*\* property to control automatic column widths.

don't includ double starts as it might cuase proble  
comments in css

Here's the updated comment without the double asterisks:

```
css
```

```
/*  
'grid-auto-rows' sets the height for rows that are automatically created  
when content exceeds the defined grid layout. It only applies to rows  
not explicitly defined using 'grid-template-rows'.  
*/
```

In your special case:

You defined exactly 9 rows using 'grid-template-rows', so any content that fits within these 9 rows will follow that height rule. If the content exceeds 9 rows (e.g., more items are added), then 'grid-auto-rows' will apply to any extra rows created beyond those 9.

To give consistent height to \*\*all\*\* rows (both defined and overflowed), you should use a combination like this:

```
grid-template-rows: repeat(9, minmax(205px, auto));  
grid-auto-rows: minmax(205px, auto);
```

This ensures that both the first 9 rows and any auto-generated rows follow the same height behavior – minimum 205px and expand if needed.

**Note:**

- `auto-fit` and `auto-fill` only work with `columns`, not rows.
- `grid-auto-rows` is only for controlling row heights.
- There is no `grid-auto-columns` property to control automatic column widths.

This should work fine now!

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>Flexbox Navbar Copy</title>
  <style>
    *{
      margin: 0;
      padding: 0;
    }

    nav{
      display: flex;

      /* justify-content: space-evenly; */
      /* The above line is good but you would not have
precise control over margin, paadding or gap */

      justify-content: center;
      /* This line doesn't add any custom space so you can
have more control over the three parameters here */
      gap: 0.5%;
    }

    nav ul{
      list-style: none;
      display: flex;
      gap: 20px;
    }
  </style>
</head>
<body>
  <nav>
    <ul>
      <li>Home</li>
      <li>About</li>
      <li>Services</li>
      <li>Contact</li>
    </ul>
  </nav>
</body>
</html>
```

```
nav ul a{  
    text-decoration: none;  
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana,  
    sans-serif;  
    font-size: 16px;  
    font-weight: 400;  
    line-height: 14px ;  
    color: #4b4b4b  
  
}
```

```
.options{  
    display: flex;  
    align-items: center;  
    justify-content: center;  
    /* no need to set height explicitly as flex-box  
    automatically matches the tallest (if we don't want then we  
    can set the height explicitly too) */
```

```
margin: 5px ;  
margin-left: 19%;
```

```
}
```

```
.options li{  
    padding: 2px;  
  
    cursor: pointer;  
    /*This actually changes the cursor to hand whenever  
    it enters in this div (Explored from harry's solution) */
```

```
display: flex;  
align-items: center;  
justify-items: center;  
}
```

```
.other-icons{  
    display: flex;  
    align-items: center ;  
    justify-content: center;
```

```
}
```

```
.other-icons img{  
    height: 45px;  
}
```

```
nav img[src="icon.png"]{  
    height: 92px;  
    margin-left: 1vw;  
}
```

```
nav img[src="sub-icon.png"]{  
    height: 12px;  
}
```

```
/* Only Nav-Bar Code Above (From Previous Exercise) (  
I Have just made a few improvements)*/  
/* Now the above Navbar was made using flexbox,  
whereas the below body would be made using grid */
```

```
main{  
    display: grid;  
    width: 60vw;  
    /* original  
    grid-template-columns: repeat(3, 1fr);  
    */  
    grid-template-columns: repeat(auto-fit,  
minmax(250px, 1fr));  
    grid-auto-rows: minmax(205px, auto);  
    /*  
    'grid-auto-rows' sets the height for rows that are  
    automatically created  
    when content exceeds the defined grid layout. It only  
    applies to rows  
    not explicitly defined using 'grid-template-rows'.  
    */
```

In your special case:  
You defined exactly 9 rows using 'grid-template-rows', so any content

that fits within these 9 rows will follow that height rule. If the content exceeds 9 rows (e.g., more items are added), then 'grid-auto-rows' will apply to any extra rows created beyond those 9.

To give consistent height to **\*\*all\*\*** rows (both defined and overflowed), you should use a combination like this:

```
grid-template-rows: repeat(9, minmax(205px, auto));  
grid-auto-rows: minmax(205px, auto);
```

This ensures that both the first 9 rows and any auto-generated rows follow the same height behavior — minimum 205px and expand if needed.

Note:

- **\*\*'auto-fit'\*\*** and **\*\*'auto-fill'\*\*** only work with **\*\*columns\*\***, not rows.
  - **\*\*'grid-auto-rows'\*\*** is only for controlling row heights.
  - There is no **\*\*'grid-auto-columns'\*\*** property to control automatic column widths.

\*/

```
margin: 60px auto;  
gap: 25px;
```

```
}
```

```
.box{  
display: grid;  
grid-template-areas: "title title title"  
"main main main"  
"footer footer footer";  
justify-items: center;  
align-items: center;  
background-color: #F9F9F9;  
}
```

```
.box .title{
```

```
grid-area: title;
font-family: 'Segoe UI', Tahoma, Geneva, Verdana,
sans-serif;
font-size: 18px;
font-weight: 700;
line-height: 26px;
margin-top: 15px;
}

.box .button a{
text-decoration: none;
color: white;
font-family: 'Segoe UI', Tahoma, Geneva, Verdana,
sans-serif;
text-align: center;
font-size: 12px;
font-weight: 700;
line-height: 24px;
}

.box .button{
grid-area: main;
padding: 4px 16px;
background-color: #5CB85C;
margin: 6px;
border-radius: 6px;
transform: translateY(70%);
/* I was gonna use margin-up or bottem here, but
but translate is amuch better option since it doesn't disturb
any other element */

}

.box .box-footer .font2{
grid-area: footer;
text-align: center;
font-family: 'Segoe UI', Tahoma, Geneva, Verdana,
sans-serif;
font-size: 8px;
line-height: 16px;
white-space: normal;
word-break: break-all;

}
```

```
.box .box-footer .font1{  
    grid-area: footer;  
    text-align: center;  
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana,  
    sans-serif;  
    font-size: 10px;  
    line-height: 16px;  
    font-weight: 450;  
    white-space: normal;  
    word-break: break-all;  
  
}
```

```
.box .box-footer{
```

/\* auto when put for dimensions like width and height "Take up all available horizontal space from the parent (minus padding/margins)."

So if a parent has not padding then the child with auto take the full space of that parent

As, for margin and padding shorthand auto takes the leftover space and divides it equally between the two sides (either vertically or horizontally depending if we use vertical or horizontal padding/margin, which actually centres the element)

```
*/
```

/\* that's why I wrote the below two lines to centre the box-footer within the the box \*/

/\* remember that if width:auto or width: 100% then it will not work as the parent doesn't have any padding so using these two properties is practically the same a sauto will try to occupy as much space as possible inside the parent (and here max-space is actually 100% only(due to no padding)) and for 100% space auto for padding/margin doesn't work as then there would be no leftover space to divide so the element won't be visually centred \*/

```
margin: 8px auto;  
width: 80%;
```

/\* for some reason maybe because of grid the above two lines were written to center the footer but it didn't happen and then translate saved everything(its actually very useful) \*/

```
        transform: translateX(18%);  
  
        /* margin: auto and width: 80% will usually center  
        your footer properly.
```

transform: translateX(18%) is not needed for centering, but if you have a specific design requirement, it can shift the element by a certain percentage of its own width.(its more of a manual fix actually that worked) \*/

```
/* Either of below two line is sufficient to do word-  
wraping inside div which prevents overflow */
```

```
word-wrap: break-word;
```

```
overflow-wrap: break-word;
```

/\* They both work the same, its just the former one is older syntax and the latter one is newer syntax (so some browser which still support the older syntax may use the former one although most of them will use the latter one)  
\*/

```
}
```

```
</style>  
</head>  
<body>  
    <header>  
        <nav>  
  
              
  
            <div class="options">  
                <ul>  
                    <li><a href="#">Products</a>  
                          
                    </li>  
                    <li><a href="#">For Business</a>
```

```
        
    </li>

    <li><a href="#">Resource</a>
        
    </li>

    <li><a href="#">Academic Licensing</a>
        
    </li>

    <li><a href="#">Our Company</a>
        
    </li>
</ul>
</div>

<div class="other-icons">
    
    
</div>
</nav>
</header>
```

```
<main>

<div class="box">
    <div class="title">English</div>
    <div class="button"><a
href="ue_english_64.exe">Download 64-bit</a></div>
    <div class="box-footer">
        <p class="font1">SHA256</p>
        <p
class="font2">F710BF78A1948AA9DD9F7090192DA527097C
```

```
05B9C483F455A05E93292D91AE27</p>
    </div>
</div>
```

```
<div class="box">
    <div class="title">Hindi</div>
    <div class="button"><a
href="ue_english_64.exe">Download 64-bit</a></div>
    <div class="box-footer">
        <p class="font1">SHA256</p>
        <p
class="font2">F710BF78A1948AA9DD9F7090192DA527097C
05B9C483F455A05E93292D91AE27</p>
    </div>
</div>
```

```
<div class="box">
    <div class="title">Japanese</div>
    <div class="button"><a
href="ue_english_64.exe">Download 64-bit</a></div>
    <div class="box-footer">
        <p class="font1">SHA256</p>
        <p
class="font2">F710BF78A1948AA9DD9F7090192DA527097C
05B9C483F455A05E93292D91AE27</p>
    </div>
</div>
```

```
<div class="box">
    <div class="title">English</div>
    <div class="button"><a
href="ue_english_64.exe">Download 64-bit</a></div>
    <div class="box-footer">
        <p class="font1">SHA256</p>
        <p
class="font2">F710BF78A1948AA9DD9F7090192DA527097C
05B9C483F455A05E93292D91AE27</p>
    </div>
</div>
```

```
<div class="box">
    <div class="title">English</div>
    <div class="button"><a href="ue_english_64.exe">Download 64-bit</a></div>
    <div class="box-footer">
        <p class="font1">SHA256</p>
        <p
            class="font2">F710BF78A1948AA9DD9F7090192DA527097C
            05B9C483F455A05E93292D91AE27</p>
    </div>
</div>
```

```
<div class="box">
    <div class="title">English</div>
    <div class="button"><a href="ue_english_64.exe">Download 64-bit</a></div>
    <div class="box-footer">
        <p class="font1">SHA256</p>
        <p
            class="font2">F710BF78A1948AA9DD9F7090192DA527097C
            05B9C483F455A05E93292D91AE27</p>
    </div>
</div>
```

```
<div class="box">
    <div class="title">English</div>
    <div class="button"><a href="ue_english_64.exe">Download 64-bit</a></div>
    <div class="box-footer">
        <p class="font1">SHA256</p>
        <p
            class="font2">F710BF78A1948AA9DD9F7090192DA527097C
            05B9C483F455A05E93292D91AE27</p>
    </div>
</div>
```

```
<div class="box">
```

```
<div class="title">English</div>
<div class="button"><a href="ue_english_64.exe">Download 64-bit</a></div>
<div class="box-footer">
    <p class="font1">SHA256</p>
    <p
        class="font2">F710BF78A1948AA9DD9F7090192DA527097C
        05B9C483F455A05E93292D91AE27</p>
    </div>
</div>

<div class="box">
    <div class="title">English</div>
    <div class="button"><a href="ue_english_64.exe">Download 64-bit</a></div>
    <div class="box-footer">
        <p class="font1">SHA256</p>
        <p
            class="font2">F710BF78A1948AA9DD9F7090192DA527097C
            05B9C483F455A05E93292D91AE27</p>
        </div>
    </div>

    <div class="box">
        <div class="title">English</div>
        <div class="button"><a href="ue_english_64.exe">Download 64-bit</a></div>
        <div class="box-footer">
            <p class="font1">SHA256</p>
            <p
                class="font2">F710BF78A1948AA9DD9F7090192DA527097C
                05B9C483F455A05E93292D91AE27</p>
            </div>
        </div>

        </main>
```

```
</body>
</html>
```

now tell how can I make this nav-bar responsive (its soley made from flexbox)

To make your flexbox-based navbar responsive, you can use media queries to adjust the layout and styling depending on the screen size. Here's how you can approach it:

## 1. Adjust the navbar items layout:

- On larger screens, the items in the navbar should remain in a row.
- On smaller screens (e.g., mobile), the navbar should stack the items vertically.

## 2. Use media queries to control layout:

We will use `flex-direction`, `justify-content`, and `align-items` to change the layout and control the alignment.

Here's how you can modify the existing CSS to make the navbar responsive:

css

```
/* Default navbar for larger screens */
nav {
    display: flex;
    justify-content: center;
    gap: 0.5%;
    flex-wrap: wrap; /* Allow navbar items to wrap on small screens */
}

nav ul {
    list-style: none;
    display: flex;
    gap: 20px;
    flex-wrap: wrap;
}

nav ul a {
    text-decoration: none;
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
```

```
    font-size: 16px;
    font-weight: 400;
    line-height: 14px;
    color: #4b4b4b;
}

.options {
    display: flex;
    align-items: center;
    justify-content: center;
    margin: 5px;
    margin-left: 19%;
}

.options li {
    padding: 2px;
    cursor: pointer;
    display: flex;
    align-items: center;
    justify-items: center;
}

.other-icons {
    display: flex;
    align-items: center;
    justify-content: center;
}

.other-icons img {
    height: 45px;
}

nav img[src="icon.png"] {
    height: 92px;
    margin-left: 1vw;
}

nav img[src="sub-icon.png"] {
    height: 12px;
}

/* Responsive Design */
```

```

@media screen and (max-width: 768px) {
    nav {
        flex-direction: column; /* Stack items vertically */
        gap: 20px;
    }

    .options {
        margin-left: 0; /* Adjust margin */
    }

    .options ul {
        flex-direction: column; /* Stack links vertically */
        gap: 10px;
    }

    .other-icons {
        margin-top: 10px; /* Add space between icons and links */
    }

    .other-icons img {
        height: 35px; /* Reduce icon size on smaller screens */
    }

    /* Make the main logo smaller */
    nav img[src="icon.png"] {
        height: 70px; /* Resize logo for small screens */
    }
}

```

## Key Changes:

- flex-wrap: wrap;** : This allows the navbar items to wrap onto new lines if the screen width is too small.
- Media Query for max-width: 768px** : This applies styles when the screen width is 768px or smaller (a common breakpoint for tablets and mobile devices).
- flex-direction: column;** in **navbar**: This stacks the navbar items vertically on smaller screens.
- Reduced icon sizes and spacing**: Smaller icons for mobile screens ensure everything fits well.

5. **Updated margins and paddings:** Adjustments to align and space items appropriately on smaller screens.

## Testing:

- On larger screens (desktops and tablets), the navbar will remain in a horizontal row layout.
- On smaller screens (phones), the navbar will stack the items vertically and adjust the logo and icon sizes for better display.

This should ensure your navbar looks good on both large and small screens. You can also adjust the `max-width` in the media query if you want to target different screen sizes.

```
@media screen and (max-width: 768px) {  
    nav {  
        flex-direction: column; /* Stack items vertically */  
        gap: 20px;  
        align-items: center;  
        justify-content: center;  
    }  
  
    .options {  
        margin-left: 0; /* Adjust margin */  
    }  
  
    .options ul {  
        flex-direction: column; /* Stack links vertically */  
        gap: 10px;  
    }  
  
    .other-icons {  
        margin-top: 10px; /* Add space between icons and links */  
    }  
  
    .other-icons img {  
        height: 35px; /* Reduce icon size on smaller screens */  
    }  
  
    /* Make the main logo smaller */  
    nav img[src="icon.png"] {  
        width: 217px; /* Resize logo for small screens */  
    }
```

```
}
```

why here even if margin to the left is 0 its nit attached to edge of scrreene therre is still some space left why?

```
.options {  
    margin-left: 0; /* Adjust margin */  
}
```

The issue you're facing might be due to padding or margin applied to the body, html, or parent elements, which are causing the `nav` to not stretch all the way to the left edge of the screen. Even though you set `margin-left: 0` on the `.options` class, there might still be some default margins or paddings applied by the browser on the body or html elements.

To fix this, you can ensure that both the `html` and `body` elements have no margin or padding. Here's how you can do it:

css

```
/* Reset margin and padding for the whole page */  
html, body {  
    margin: 0;  
    padding: 0;  
}
```

Adding this CSS rule at the top of your stylesheet will remove any default margin or padding from the root elements, ensuring that elements like `.options` align perfectly with the edge of the screen.

This should solve the issue where there's still some space left on the left side even after setting `margin-left: 0;`.