1. CSS Grid basics

- `display: grid` makes all its direct children grid items
- `grid-template-columns: 200px 300px 50px` creates three columns (horizontal axis)
- `grid-template-rows: 100px 500px` creates two rows (vertical axis)
- `grid-gap: 20px` behaves as margin and creates space (20px) between grid items
- `grid-template-columns: repeat(5, 100px)` will create 5 columns of 100px wide
- You can use any measuring unit you want (em, rem, px, %)

2. Explicit/implicit grid

- A **grid track** is the space between two grid lines (row or a column).
- Columns or row you create yourself by defining `grid-template-columns` belong to the **explicit grid**
- Columns or rows that are created by the browser itself are belong to the **implicit grid**. For example: when you have 10 items, and define 2 columns, the browser will create 5 rows. These row tracks belong to the implicit grid. The columns will belong to the explicit grid.
- `grid-auto-rows: 100px` to give all the implicit grid rows a height of 100px. `grid-auto-columns: 100px` does the same for columns. When defining multiple values in `grid-auto-x`, it'll give height to its counterpart in the DOM. So `grid-auto-rows: 100px 200px` will give a height of 100px to the first row on the

implicit grid, and 200px on the second row of the implicit grid and give an auto height to all the items after that. (**Note: this is bugged in Firefox)**

- `grid-auto-flow: column` makes sure that any implicit grid items will be columns
- `grid-auto-flow: row` makes sure that any implicit grid items will be rows **(Note: this is the default behaviour)**

3. Sizing tracks

- Using percentages in CSS Grid will not account for `grid-gap`.
- Use the `fr` (fracture) measure unit (fractional unit) to achieve fluid layouts. These take up all the free space in a track. These work similar as `flex-grow` and `flex-shrink`.
- `grid-template-columns: 200px 1fr` makes sure that that the first column is 200px wide and the second column will take up the rest of the horizontal space.
- The `auto` keyword will automatically adjust the width of the column or the height of the row to the biggest element in it.

4. Sizing grid items

- `grid-column: span 2` can be used on a grid item (direct child of a `display: grid`). This will make sure it will span the length of two columns (just like the `colspan` attribute in tables)
- `grid-column: span` actually makes your column wider and pushes the next item more to the right. It doesn't overlap or remove items.
- `grid-row: span` works the same, but for rows.

- When you exceed the explicit grid's tracks (`grid-template-columns: 200px 300px` — *two columns*) with for example `grid-column: span 8`. The browser will create five more implicit column tracks **and widen your grid**.

5. Placing grid items

- `grid-column` is short-hand for `grid-column-start` and `grid-column-end`
- You can place grid items on track values. These track values are the lines of your explicit grid. When you define 5 columns, the first track value (1) will be left of the first column. The second track value (2) will be right of the first column. The third track value (3) will be left of the second column. And so forth..
- `grid-column-start: 2` will make sure your item gets placed on the second track value. If you add `grid-column-end: 5` it will span your item from track value 2 to 5 (so it will be three columns wide).
- `grid-column: 2 / 5` is short-hand for this pattern
- `grid-column: 2 / span 5` will start at track 2 and **span the item 5 columns**
- `grid-column-end: -1` will stretch the grid item out until the right side of the grid (making it as wide as possible)
- All of this goes for `grid-row` as well, but for rows

6. Auto-fit and auto-fill

- `grid-template-columns: repeat(`**auto-fill**`, 150px)` will figure out on its own how many 150px wide column tracks it can create on the current grid. Even when you don't have enough items

to fill the tracks, it will still create the explicit tracks to fill out the grid. *That means you can move around items on the whole horizontal axis of the grid.

- `grid-template-columns: repeat(`**auto-fit**`, 150px)` doesn't add any more tracks when you don't have enough items. So when the grid is 900px wide in total, and you have only 4 items, it will create 4 tracks of 150px and won't create any extra columns on the explicit grid. So you won't be able to move around items further the the fourth track.
- Same goes for rows.

7. Minmax

- `minmax` provide a function for a minimum and maximum size
- `grid-template-columns: repeat(auto-fill, `**minmax**`(150px, 1fr)` will make sure that each track will be a minimum of 150px wide and and a maximum of 1 fraction wide (100%).
- In conjunction with **auto-fill** or **auto-fit** it will create implicit rows when the grid gets resized to a *small* screen. So when you have min-width of 150px and 4 items, but your grid is only 450px wide, it will automatically place the fourth item on the second row (because it don't fit boiii)
- Because **auto-fit** doesn't create extra tracks (check **section 6**), it'll fill out the current tracks (max 1fr) to the width of the grid. You'll have a responsive column grid when using this.

8. Grid template areas

- `grid-template-areas` names the areas of your explicit grid

- If you want to define template areas within a grid of three columns and two rows, you literally type out your grid and give the areas names, like:

```
grid-template-columns: 150px 1fr 100px;
grid-template-rows: 150px 150px;
grid-template-areas:
  "sidebar content sidebar-ad"
  "footer footer footer";
```

- Use `grid-area: sidebar` on a grid item to place it on the sidebar template area (so the top left spot in grid in this case)
- If you place a grid item on the `grid-area: footer` spot, it will span the whole width of the grid, because the browser will connect areas with the same name, if they have the same name.
- It's (of course) possible to redefine the whole grid in media queries.

9. Naming track lines

- Every track line (the lines on the start and end of your track) can be named
- Defining track names is done with names enclosed in brackets, where you define your tracks (`grid-template-columns` and `grid-template-rows`).

```
grid-template-columns:
  [site-left sidebar-left] 150px [sidebar-right content-left]
  1fr [content-right site-right];
```

- The brackets encased names (bold) are the names of track lines. You can have multiple names per track line. The tracks itself (columns in this case) are defined like normal
- Use these names like you would track line numbers:

- `grid-column: site-left / site-right;` will span a grid item across the two columns

10. Block fitting

- `grid-auto-flow: dense` checks if an item can fit on a track. If it can't it will place it on the next line. It will then go to the next item and check if that one does fit.

11. Grid alignment and centering

- You can align the items themselves within the tracks (columns and rows)
- Use `justify-items` for horizontal alignment
- `justify-items: start` aligns you item to the left of the track its in. `center` and `end` will align them to the center and the right
- Use `align-items` for vertical alignment
- `align-items: center` will center the item on the vertical axis. `start` and `end` will place the items on the top and the bottom
- `place-items: center` is short-hand for `justify-items: center` and `align-items: center`
- If you have extra space within your grid (the container is bigger than the tracks combined), you can align all the tracks together
- `justify-content` aligns all the tracks within the grid container. Possible values behave the same way that Flexbox does
- Use `center` to center, `start` to align all the tracks to the left, `end` to align them to the right, `space-between` to divide the

space between tracks except outer margins, `space-around` for evenly divided space left and right of each track

- `align-content` does the same thing but for the vertical axis

12. Re-ordering grid items

- Use the `order` property to reorder items within your grid (place them on different tracks)
- Default value for `order` is 0, so if you order one, you'll probably have to add the `order` property to more items
- `order` messes up your accessibility (selection and taborder)