CSS Grid Model

1. **Grid Layout:**
    a. The CSS Grid Layout Module offers a grid-based layout system, with rows and columns, making it easier to design web pages without having to use floats and positioning.

2. **Core Concepts of Grid:**
    a. Grid container
    b. Grid Items

3. **Grid Container → display**
    a. An HTML element becomes a grid container when its *display:grid* property is set to grid.
    b. **Syntax**:

```
.container {
        display: grid;
}
```

4. **Grid Columns:**
    a. The vertical lines of grid items are called columns.

5. **Grid Rows:**
    a. The horizontal lines of grid items are called rows.

6. **Grid Gaps:**
    a. The spaces between each column/row are called gaps.
    b. **Syntax**:

```
row-gap : 10px;
column-gap: 10px;
```

    c. The gap property is a shorthand property for the *row-gap* and the *column-gap* properties.
    d. **Syntax**:

```
gap:10px;
```

7. **Grid Line:**
    a. Column lines ⇒ The lines between columns are called column lines.
    b. Row Lines ⇒ The lines between rows are called row lines.

8. **Grid Container Properties (Parent Properties):**
    a. grid-template-columns
    b. grid-template-rows
    c. justify-content
    d. align-content

1. **grid-template-columns property:**
    a. The *grid-template-columns* property defines the number of columns in your grid layout, and it can define the width of each column.
    b. Lets consider I want to create a 3 columns :

c. **Syntax ⇒**

```
grid-template-columns: auto auto auto;
                                100px 200px 300px;
                                1fr 2fr 1fr;
                                10% 20% 30%;
                                repeat(3, 100px);
                                100px 200px auto 300px;
```

2. grid-template-rows property:

a. The *grid-template-rows* property defines the height of each row.

b. Lets consider I want to create a 3 rows :

c. **Syntax ⇒**

```
grid-template-rows: auto auto auto;
                            100px 200px 300px;
                                1fr 2fr 1fr;
                                  10% 20% 30%;
                                      repeat(3, 100px);
                                100px 200px auto 300px;
```

3. justify-content Property:

a. The *justify-content* property is used to align the whole grid inside the container.

b. **Syntax ⇒**

```
justify-content: start|center|end|space-evenly|space-aro
und|space-between;
```

4. align-content Property:

a. The *align-content* property is used to vertically align the whole grid inside the container.

b. **Syntax** ⇒

```
align-content: start|center|end|space-evenly|space-around|space-between;
```

9. Grid Items Properties (Child Properties):

a. grid-column

b. grid-row

1. **grid-column property**:

a. The *grid-column* property defines which column(s) to place an item.

b. You define where the item will start, and where the item will end.

c. It is nothing but the merging of columns in horizontal direction.

d. **Syntax** ⇒

```
grid-column: 1 / 3 ;(1st way)
grid-column: 1/span 2;(2nd way)

Here 1 is the grid-starting column.
and  3 is the grid-ending column.
```

e. **NOTE** ⇒ *grid-column* is the shorthand property for the *grid-column-start* and *grid-column-end* .

f. You can separate both the properties instead of the shorthand property grid-column.

2. **grid-row property**:
    a. The *grid-row* property defines on which row(s) to place an item.
    b. You define where the item will start, and where the item will end.
    c. It is nothing but the merging of rows in vertical direction.
    d. **NOTE** ⇒ *Always remember one thing we have to first define the column number in which you are merging the rows.*
    e. **Syntax**:

```
grid-column: 2; (here that means I want to merge rows in
column 2)
grid-row: 1 / 3 ;(1st way)
grid-row: 1/span 2;(2nd way)

Here 1 is the grid-starting row.
and  3 is the grid-ending row.
```

    f. **NOTE** ⇒ *grid-row* is the shorthand property for the *grid-row-start* and *grid-row-end*.
    g. You can separate both the properties instead of the shorthand property grid-row.

10. **grid-template-areas property (Parent Property)**:
    a. Named grid items can be referred to by the *grid-template-areas* property of the grid container.
    b. **Syntax**:

```
grid-template-areas: "header header header"
                     "aside section article"; (as a
example)
```

11. **grid-area property (Child Property)**
    a. The *grid-area* property can also be used to assign names to grid items.
    b. **Syntax**:

```
grid-area: name_of_the_grid;
```

    c. **Example ⇒**

```
#item1 {
        grid-area: header;
    }
    #item2 {
        grid-area: aside;
    }
    #item3 {
        grid-area:article;
    }
```

12. **Shorthand Property → grid-area Property**
    a. The *grid-area* property can be used as a shorthand property for the *grid-row-start* , *grid-column-start* , *grid-row-end* and the *grid-column-end* properties.
    b. **Syntax**:

```
grid-area: grid-row-start / grid-column-start / grid-row-en
d / grid-column-start;
```

    c. **Example ⇒**

```
grid-area: 1 / 2 / 3 / 4;
    OR
grid-area: 1 / 2 / span 2 / span 3;
```

13. **justify-items: (Parent Property)**
    a. Specifies the *horizontal alignment* of all grid items inside their own grid cells along the inline axis (from left-to-right).
    b. **Syntax and its values**:
    justify-items: start /* Align items to the start (left) */
    justify-items: end /* Align items to the end (right) */
    justify-items: center /* Center items horizontally */
    justify-items: stretch /* Stretch items to fill the cell width */
    c. **Default Value**: stretch

14. **align-items: (Parent Property)**
    a. Specifies the *vertical alignment* of all grid items inside their own grid cells along the block axis (top–bottom).
    b. **Syntax and its Values**:
    align-items: start /* Align items to the top */
    align-items: end /* Align items to the bottom */
    align-items: center /* Center items vertically */
    align-items: stretch /* Stretch items to fill the cell height
    c. **Default Value**: stretch

15. **justify-self: (Child Property)**
    a. Specifies the *horizontal alignment* of a *single grid item* inside its own grid cell, overriding *justify-items*.
    b. **Syntax and its values**:
    justify-self: start /* Align to the start (left) */
    justify-self: end /* Align to the end (right) */
    justify-self: center /* Center horizontally */
    justify-self: stretch /* Stretch to fill the cell width */

c. **Default Value**: stretch

16. **align-self: (Child Property)**
    a. Specifies the *vertical alignment* of a *single grid item* inside its own grid cell, overriding *align-items*.
    b. **Syntax and its Values**:

```
align-self: start   /* Align to the top */
align-self: end     /* Align to the bottom */
align-self: center  /* Center vertically */
align-self: stretch /* Stretch to fill the cell height */
```

    c. **Default Value**: stretch

Add-On Concept in Grid:

★ **Subgrid in CSS Grid Model?**

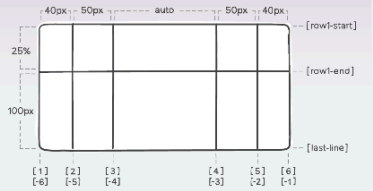
- In CSS Grid Layout, a subgrid is a feature that allows a grid item's children to align with the grid tracks (rows/columns) defined by its parent grid.
- Normally, when you create a grid container inside another grid item, that inner grid does not inherit the track definitions of the parent. Instead, it creates its own *independent grid*.
- But with *subgrid*, the child grid can reuse (inherit) the parent grid's tracks, making layouts more consistent and easier to maintain.
- **For Example**:

```css
.parent {
        display: grid;
        grid-template-columns: 1fr 2fr 1fr;
        grid-template-rows: auto auto;
      }

.child {
       display: grid;
       grid-template-columns: subgrid;
    /* Inherits from parent */
       grid-template-rows: subgrid;
    /* Inherits from parent */
    }
```

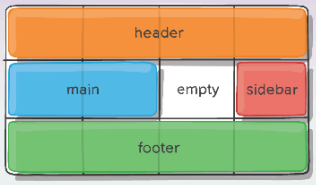
CSS Grid

grid-template-columns grid-template-rows



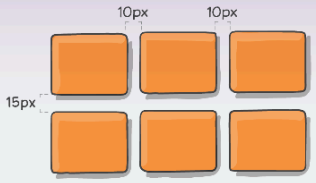

```
.container {
  grid-template-columns: <value> | <name>;
  grid-template-rows: <value> | <name>;
}
```

grid-template-areas



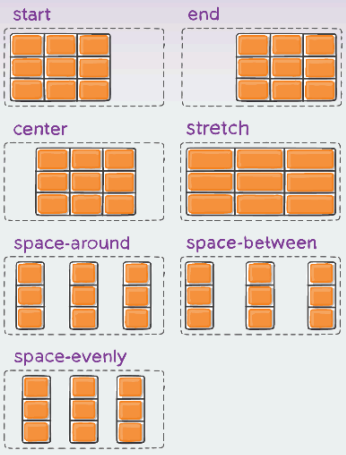

```
.container {
  grid-template-areas: "<name> | . | none";
}
```

column-gap, row-gap, gap




```
.container {
  column-gap: <value>;
  row-gap: <value>;
  gap: <row-gap> <column-gap>;
}
```

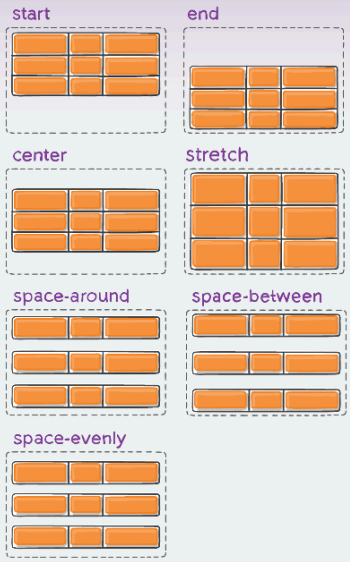
justify-content




```
.container {
  justify-content: start | end | center |
  stretch | space-around | space-between |
  space-evenly;
}
```

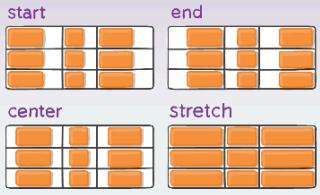
```
.container {
  display: grid; /* or inline-grid */
}
```

align-content



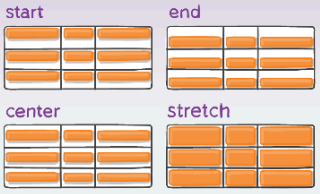

```
.container {
  align-content: start | end | center |
  stretch | space-around | space-between |
  space-evenly;
}
```

justify-items



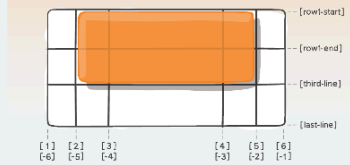

```
.container {
  justify-items: start | end |
  center | stretch;
}
```

align-items



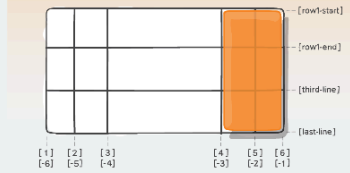

```
.container {
  align-items: start | end |
  center | stretch;
}
```

grid-column, grid-row



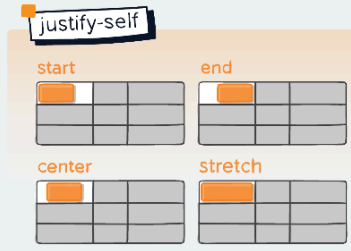

```
.item {
  grid-column: <start-line> / <end-line> |
  <start-line> / span <value>;
  grid-row: <start-line> / <end-line> |
  <start-line> / span <value>;
}
```

grid-area



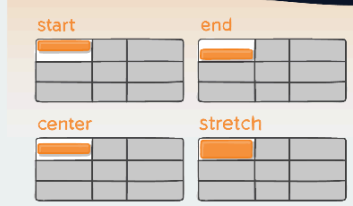

```
.item {
  grid-area: <row-start> / <column-start>
  / <row-end> / <column-end> | <name>;
}
```

justify-self




```
.item {
  justify-self: start | end | center |
  stretch;
}
```

align-self




```
.item {
  align-self: start | end | center |
  stretch;
}
```