CHAPTER 8 – RESPONSIVE WEB DESIGN

TOPICS TO BE COVERED

- **Introduction**

  Reasons behind it.
- **CSS FLEXBOX**
- **CSS GRID**

More info on this topic can be found in e-books provided in LMS.
Presentation slides made by Fifah S., M. Khalid.

INTRODUCTION TO RESPONSIVENESS

- By default, most browsers on small devices (tablets, smartphones) shrink a web page down to fit the screen and provide mechanisms for *zooming* and *moving* around the page.

- It works, BUT not a great experience to begin with:
  - The text too small to read.
  - Link too small to tap.
  - Zooming and moving around is distracting.

- Responsive Web Design is a strategy to provide custom layouts to devices based on the size of the *viewport* (browser window).

PAGE LAYOUT STRATEGIES

- There are several page layout approaches emerged that address the issues in various ways:

1. **Fixed layouts**
   Stay put at a specific pixel width regardless of the size of the browser window / text size.

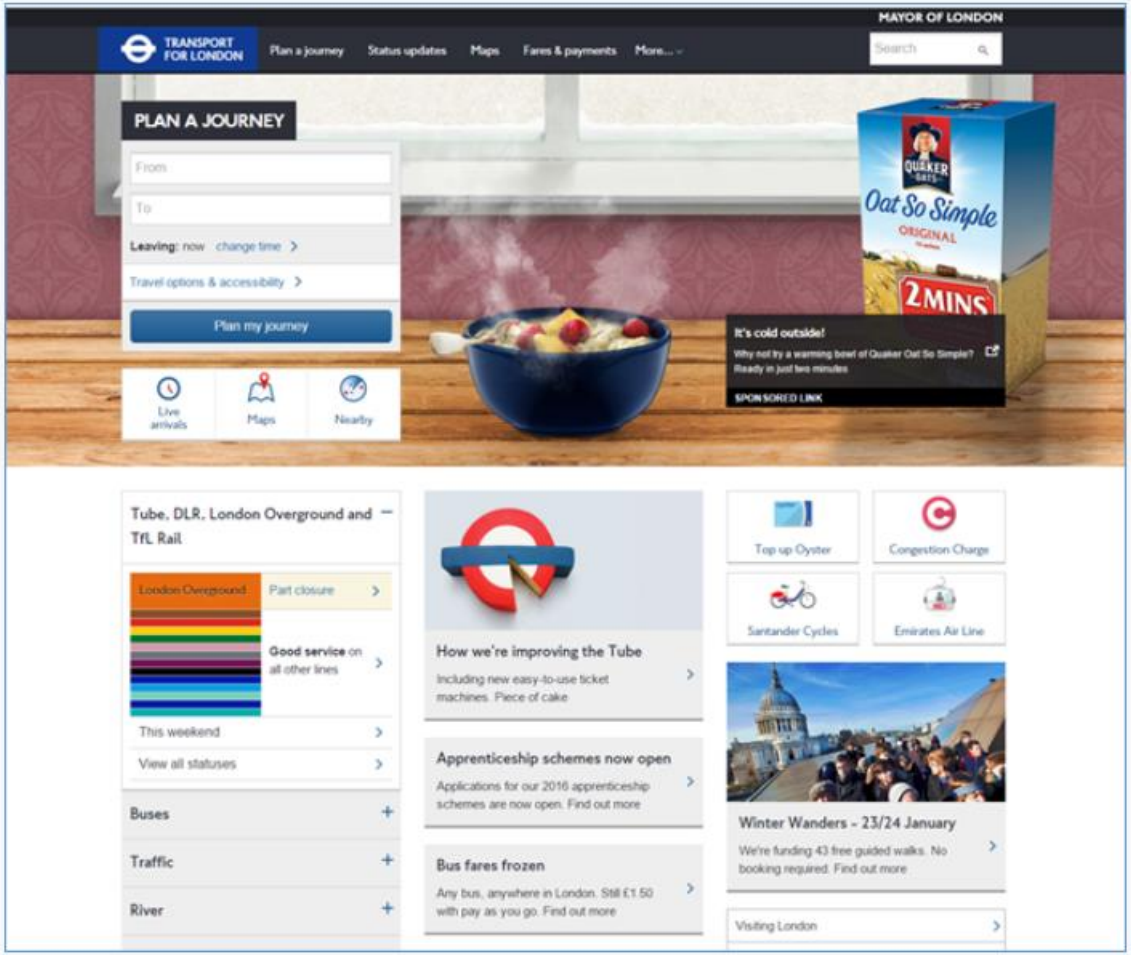
2. **Fluid layouts**
   Resize proportionally when the browser window resizes.

3. **Elastic layouts**
   Resize proportionally based on the size of the text.

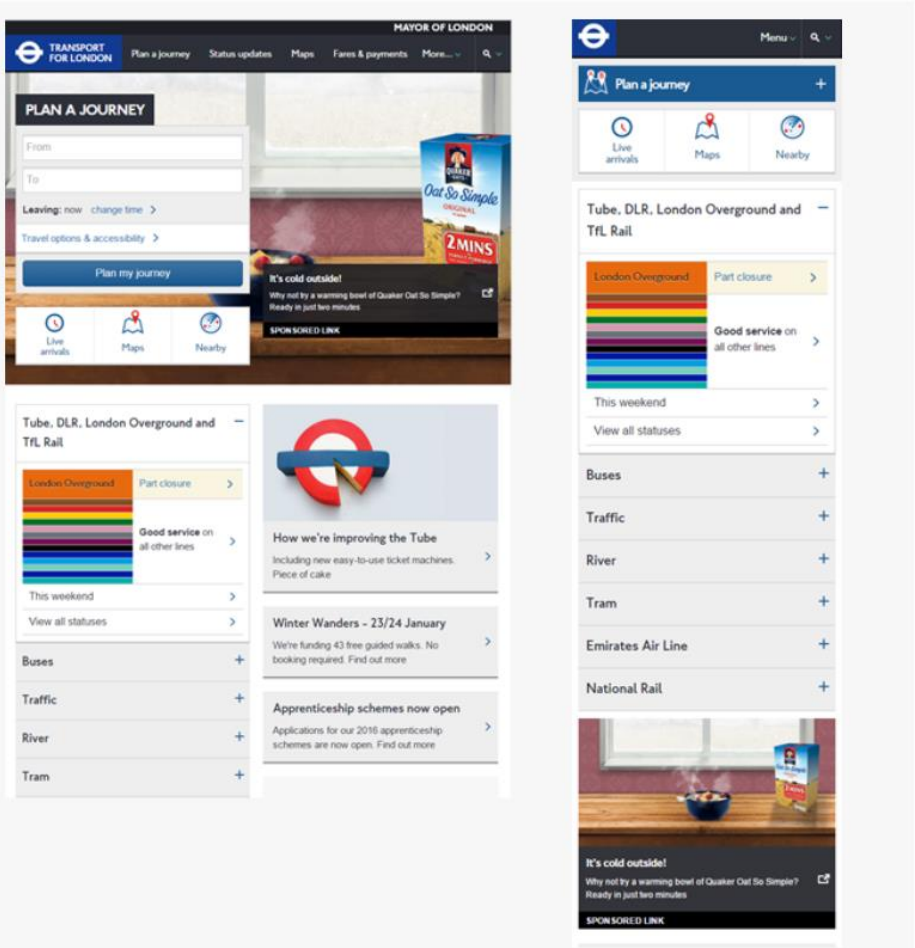
4. **Hybrid layouts**
   Combine fixed and scalable areas.

DIFFERENT TYPES OF SITE

- Not all websites are created equal. There are:
  1. **Mobile dedicated sites** are designed for mobile phones.
  2. **Web apps** are a special type of mobile-dedicated site that looks and feels like an app.
  3. **Responsive-design site** are sites design for a multitude of devices with different screen sizes; i.e. automatically adjust the layout of their content to the available screen size.
  4. **Desktop sites** are designed for the desktop and are not mobile optimized.



Transport for London (tfl.gov.uk) is a responsive site. On the desktop the content was formatted across 3 columns.



The tablet (left) and mobile (right) versions of tfl.gov.uk showed the desktop content in two columns and one column respectively.

PAGE LAYOUT TECHNIQUES

» CSS are constantly evolving to meet the requirements and needs of web designers and developers.

1. **Multicolumn layouts using floats and positions**
   Most straightforward layout improvement implemented in the browsers. There are few advantages and disadvantages using this method.

2. **CSS Flexbox**
   Provides a much simpler way to arrange element boxes in relation to one another.

3. **CSS Grid layout system**
   Establishing a grid or rows and columns for an element and then position other elements along that grid.

#1 – USING FLOATS AND POSITIONS

Two columns, fluid layout

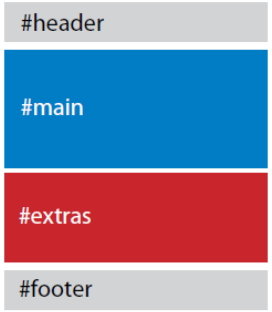
- Set widths on both column elements and float them to the left.
- Clear the footer to keep it at the bottom of the page.

The markup

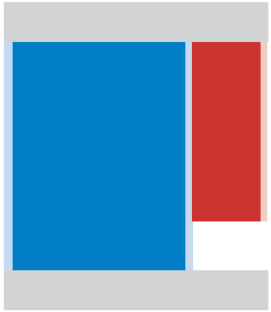
```
<div id="header">Masthead and headline</div>
<div id="main">Main article</div>
<div id="extras">List of links and news</div>
<div id="footer">Copyright information</div>
```

The styles

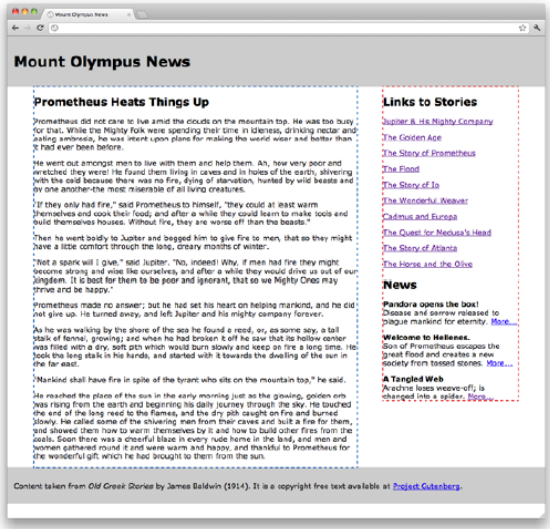
```
#main {
  float: left;
  width: 60%;
  margin: 0 5%;
}
#extras {
  float: left;
  width: 25%;
  margin: 0 5% 0 0;
}
#footer {
  clear: left;
}
```



Source order



Layout



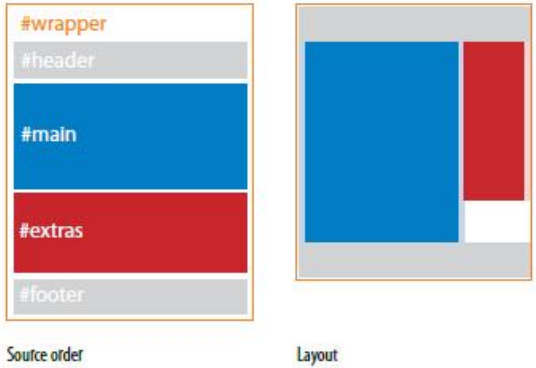
#1 – USING FLOATS AND POSITIONS

Two columns, fixed-width layout

» Wrap the content in *div* to which we can set a specific pixel width.

The styles

```
#wrapper {
  width: 960px;
}
#main {
  float: left;
  width: 650px;
  margin: 0 20px;
}
#extras {
  float: left;
  width: 250px;
  margin: 0 20px 0 0;
}
#footer {
  clear: left;
}
```



The markup

```
<div id="wrapper">
  <div id="header">Masthead and headline</div>
  <div id="main">Main article</div>
  <div id="extras">List of links and news</div>
  <div id="footer">Copyright information</div>
</div>
```

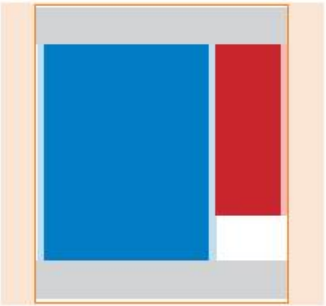
#1 – USING FLOATS AND POSITIONS

Two columns, fixed-width layout, centered

» Setting the left and right margins to auto, which will keep the whole page centered.

```
#wrapper {
  width: 960px;
  margin: 0 auto;
}
```

Source order

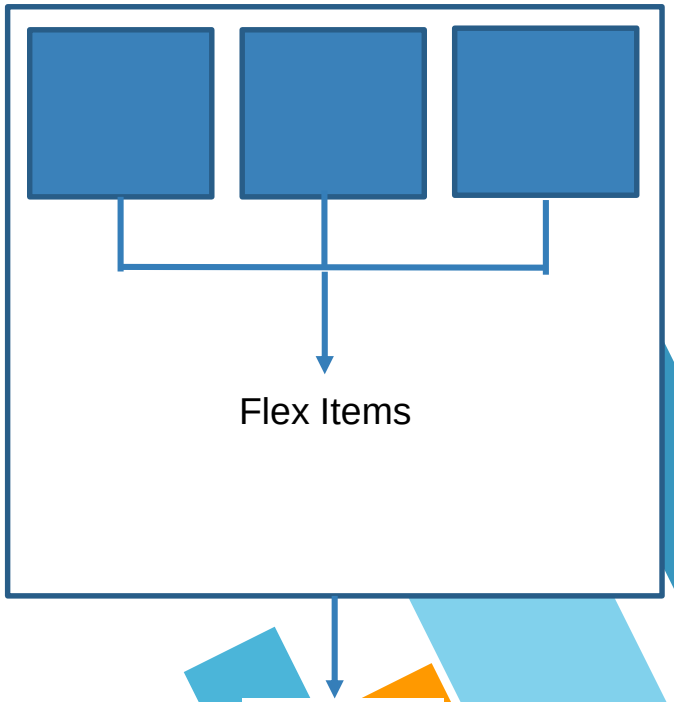


Layout

#2 – CSS FLEXBOX

- This layout controls the position, size and spacing of elements relative to their parent elements and each other.

- A container must be implemented / created first.
- Flex item is the direct descendent element within the container.
- Hence, controlling the behavior of the flex items is made easier by using flexbox properties.
- Example of behavior:
  - Grow within the container
  - Shrink relative to each other.
  - Stacking on top of each other or side by side.
  - Etc.



INTRODUCTION TO CSS FLEXBOX

```
<body>
  <div class="central"> <!--to centralise elements-->
    <div class="flex-container">
        <div class="box one"></div>
        <div class="box two"></div>
        <div class="box three"></div>
    </div>
  </div>
</body>
```

- Save the file as index.html.
- Create three divisions within the div class flex-container.
- This is to show how the behavior change relative to the parent (i.e. the flex-container).

FLEX ITEMS

```
.flex-container{
  display: flex;
}
```

- The items can be modify by creating specifying the display value inside the container to **display: flex.**
- Then, we can change each and every items relative to the container to have the same behavior.

NORMAL VS FLEX

Without **display: flex;**

display: flex;

FLEX DEFAULT BEHAVIOR

- The default behavior of flex items are stacked left to right in a row.
- P.s. using **float: left;** may have the same effect BUT the element collapse the parent element (.flex-container).

FLEX GROW PROPERTY

```
.one{
  background: purple;
  flex-grow: 1;

}

.two{
  background: orange;
  flex-grow: 2;
}

.three{
  background: lime;
  flex-grow: 3;
}
```

- Giving the same grow rate property on each flex item.
- The items grows at a given rate as specified in the **flex-grow** property value.
- The growth rate is some-like columns on a grid. We can use flex-grow property to define how many columns we want the element to be.

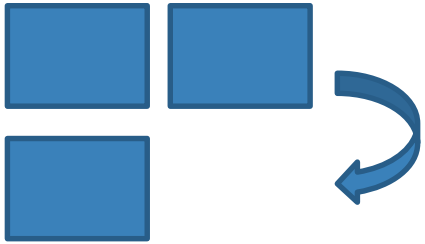
FLEX SHRINK PROPERTY

```
.box{
  height: 100px;
  width: 320px;
}

.one{
  background: purple;
  flex-shrink: 1;

}

.two{
  background: orange;
  flex-shrink: 2;
}

.three{
  background: lime;
  flex-shrink: 3;
}
```

- Opposite of flex-grow property.
- The value determine the rate at which they shrink as the browser gets smaller.
- By default, all the flex items will be shrink in proportion to one another at the same rate.
- The bigger value of flex shrink determines the more item being shrinked.

FLEX WRAP PROPERTY

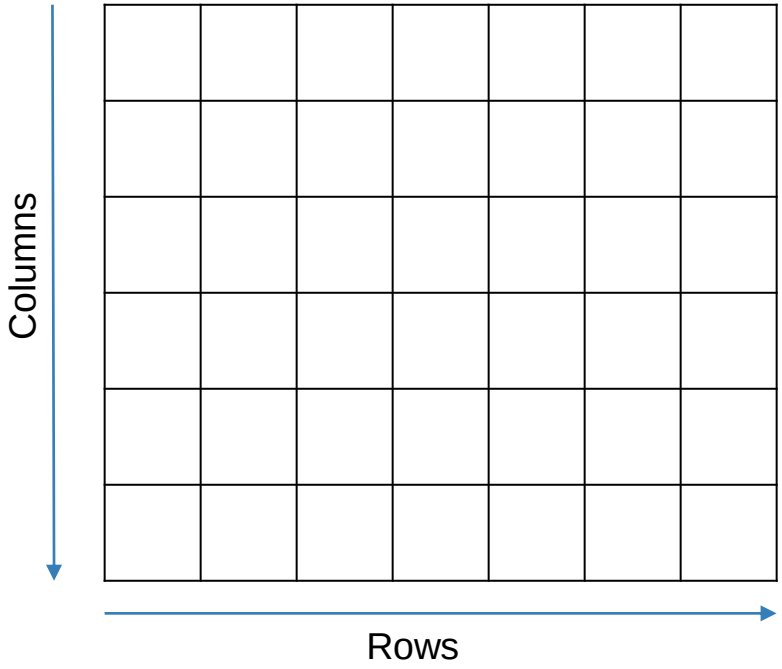
TO



- Scroll bar will be present when we reduce the width of the browser. This is due to the specified property **min-width: 200px;**
- Use this property to create a more responsive elements.
- Values of flex-wrap:
  - Wrap – element will go below.
  - wrap-reverse – element will go on top.
  - Nowrap – default behavior.

#3 – CSS GRID

- Elements can be put everywhere on the grid.
- There are no limits on the number of columns.



INTRODUCTION TO CSS GRID

```
<!DOCTYPE html>
<html>
<head>
  <title>Using CSS Grid</title>
</head>
<body>
  <div id="content">
    <div>1</div>
    <div>2</div>
    <div>3</div>
    <div>4</div>
    <div>5</div>
    <div>6</div>
    <div>7</div>
    <div>8</div>
    <div>9</div>
  </div>
</body>
</html>
```

- Save the file as index.html.
- The output will be:

GRID COLUMNS

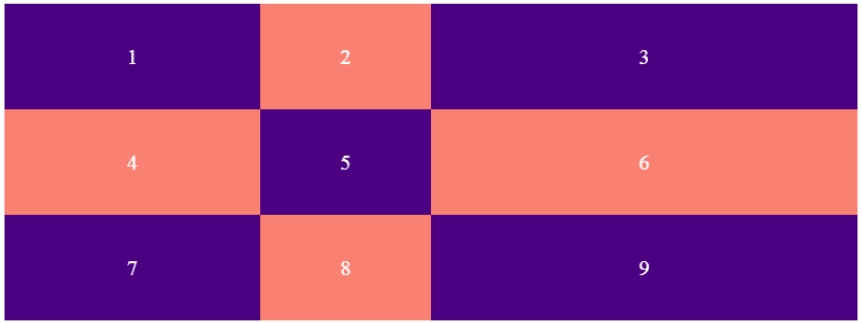
```
<head>
  <title>Using CSS Grid</title>
  <style>
    #content{
      display: grid;
      grid-template-columns: 30% 20% 50%;
      max-width: 960px;
      margin: 0 auto;
    }
  </style>
</head>
```

- **#content** acts as a grid wrapper.
- **Display: grid** is to create a grid. Since there are 9 's value, the grid will automatically be 3 X 3.
- **Grid-template-columns** used to make different sizes of columns.
- There are three different values of grid-template columns:
    1. Percentage
    2. Fractions
    3. Repeat

1. PERCENTAGE GRID

```
<head>
  <title>Using CSS Grid</title>
  <style>
    #content{
      display: grid;
      grid-template-columns: 30% 20% 50%;
      max-width: 960px;
      margin: 0 auto;
    }
  </style>
</head>
```

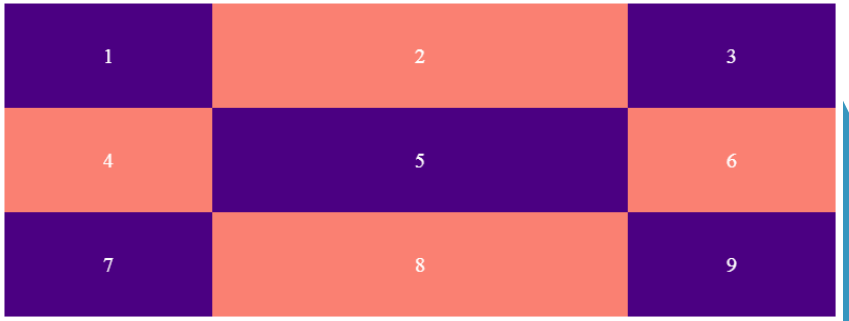
- There are 9 's value, hence it will be 3 columns by 3 rows.
- By stating 30% 20% 50%, we are setting the columns by 3:2:5 ratio.



2. FRACTIONS GRID

```
<head>
  <title>Using CSS Grid</title>
  <style>
    #content{
      display: grid;
      /*grid-template-columns: 30% 20% 50%; */
      grid-template-columns: 1fr 2fr 1fr;
      max-width: 960px;
      margin: 0 auto;
    }
  </style>
</head>
```

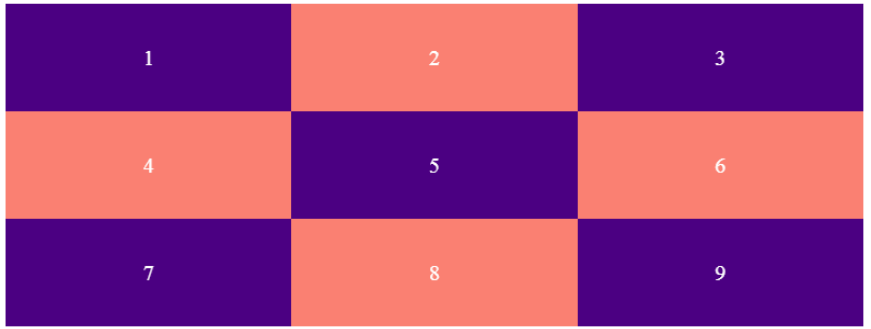
- There are 9 's value, hence it will be 3 columns by 3 rows.
- By stating 1fr 2fr 1fr we are setting the columns by 1:2:1 ratio.



3. REPEAT GRID

```
<head>
  <title>Using CSS Grid</title>
  <style>
    #content{
      display: grid;
      /*grid-template-columns: 30% 20% 50%; */
      /*grid-template-columns: 1fr 2fr 1fr;*/
      grid-template-columns: repeat(3, 1fr);
      max-width: 960px;
      margin: 0 auto;
    }
  </style>
</head>
```

- By stating repeat (3, 1fr), we are setting 3 columns with width of 1 fraction for all 3 columns.
- Repeat (4, 1fr) means we set 4 columns with 1 fraction width.



GRID ROWS

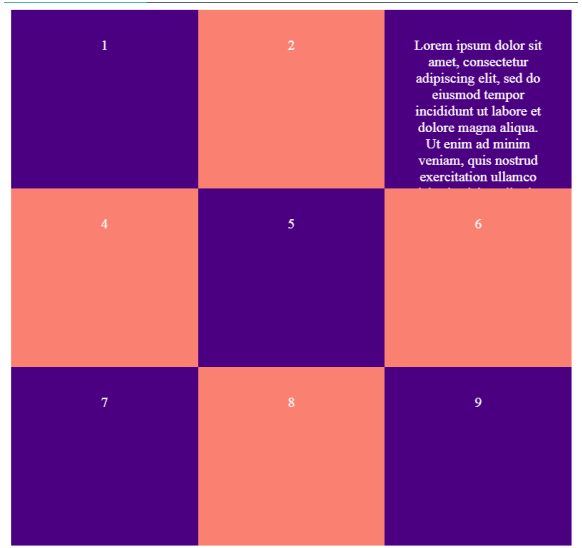
- Rows created when contents are inserted. The contents are wrapped automatically.
- If we add more content onto the grid, the row height is going to be dictated by that amount of content.

```
<div id="content">
  <div>1</div>
  <div>2</div>
  <div>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do
    eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad
    minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip
    ex ea commodo consequat. Duis aute irure dolor in reprehenderit in
    voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur
    sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt
    mollit anim id est laborum.</div>
  <div>4</div>
  <div>5</div>
  <div>6</div>
  <div>7</div>
  <div>8</div>
  <div>9</div>
</div>
```

GRID-AUTO-ROWS

- Rows created when contents are inserted. The contents are wrapped automatically.
- If we add more content onto the grid, the row height is going to be dictated by that amount of content.

```
#content{
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-auto-rows: 200px;
  max-width: 960px;
  margin: 0 auto;
}
```



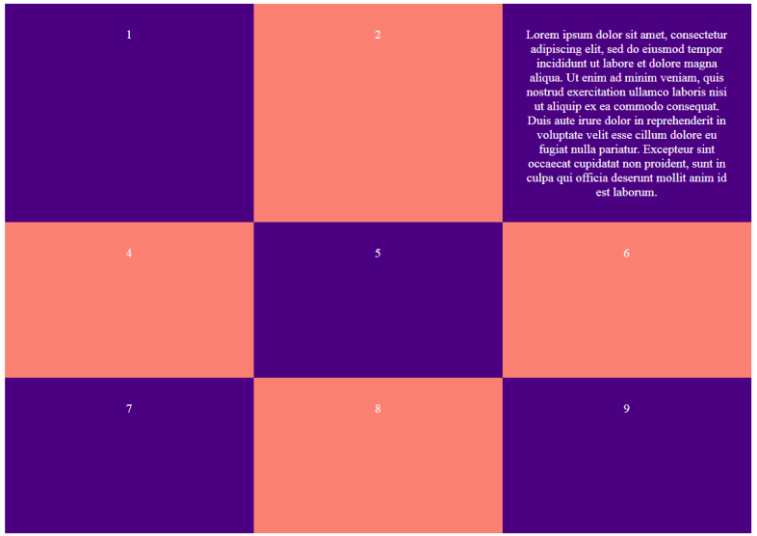
grid-auto-rows: 200px;
All rows will have 200px in height.

Cons:

- The content is getting cut off; totally disregards any content inside of the row.

GRID-AUTO-ROWS

```
#content{
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-auto-rows: minmax(200px, auto);
  max-width: 960px;
  margin: 0 auto;
}
```



grid-auto-rows: minmax(200px, auto);

- If there is no content, the height will be 200px.
- If there is content, the height will be changed to the height of the content.

NUMBER OF ROWS

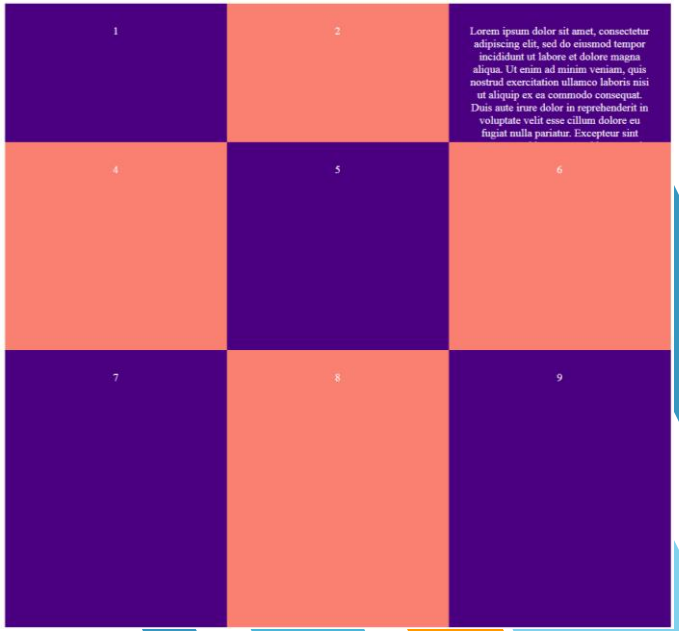
- There are TWO ways to specify the number of rows:

1. **grid-template-rows: 200px 300px 400px 200px;**

   means that the height of:

   - First row: 200px
   - Second row: 300px
   - Third row: 400px
   - Forth row: 200px

```
#content{
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-auto-rows: minmax(200px, auto);
  grid-template-rows: 200px 300px 400px 200px;
  max-width: 960px;
  margin: 0 auto;
}
```



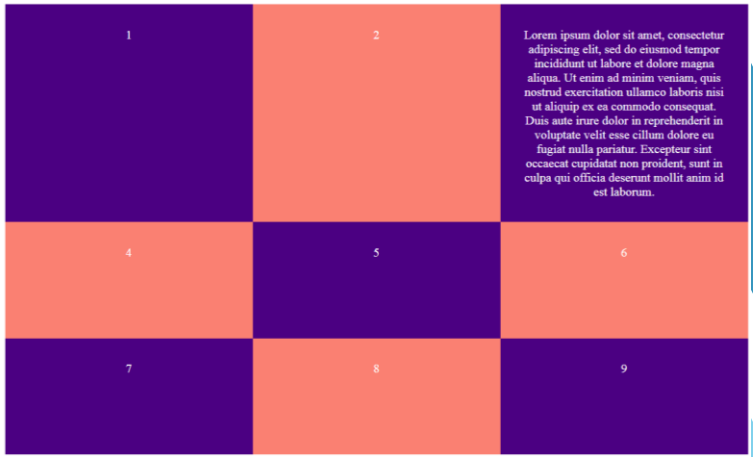
NUMBER OF ROWS

- There are TWO ways to specify the number of rows:

2. **grid-template-rows: repeat(3, minmax(200px, auto));**

   means that the grid will be having 3 rows, with minimum height of 200px and maximum height of the content.

```
#content{
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-auto-rows: minmax(200px, auto);
  grid-template-rows: repeat(3, minmax(150px, auto));
  max-width: 960px;
  margin: 0 auto;
}
```



ROW/COLUMN GAP

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

```
#content{
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-auto-rows: minmax(200px, auto);
  grid-template-rows: repeat(3, minmax(150px, auto));
  grid-row-gap: 10px;
  grid-column-gap: 10px;
  max-width: 960px;
  margin: 0 auto;
}
```

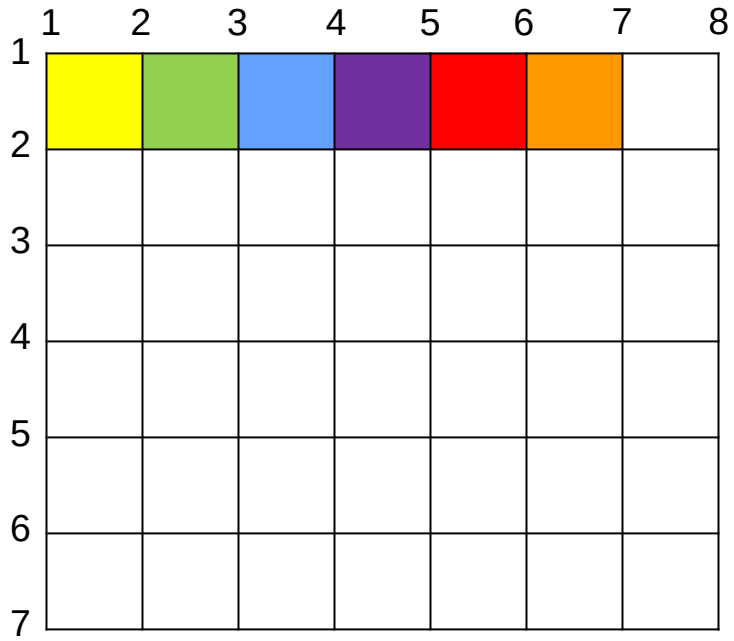
ROW/COLUMN GAP

- Grid-gap: 10px;

```
#content{
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-auto-rows: minmax(200px, auto);
  grid-template-rows: repeat(3, minmax(150px, auto));
  /*grid-row-gap: 10px;
  grid-column-gap: 10px;*/
  grid-gap: 10px;
  max-width: 960px;
  margin: 0 auto;
}
```

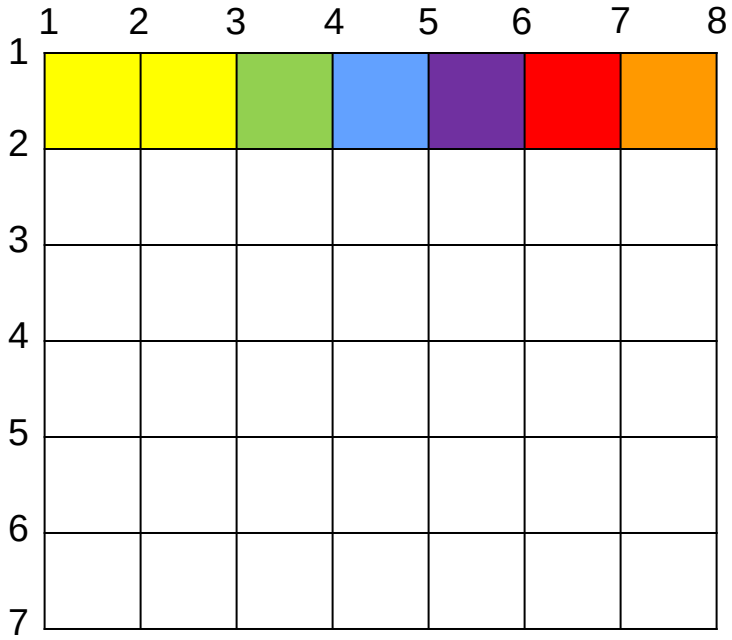
GRID LINES

- We can place elements anywhere, even though its not in the first block.



- There are 7 columns and 7+1 column lines.
- And 6 rows with 6+1 row lines.

COLUMN GRID LINES

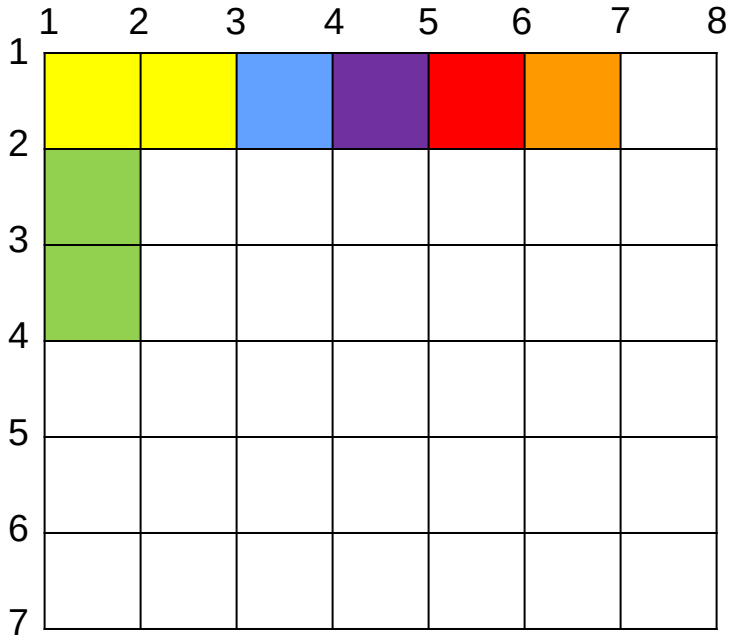



```
.one{
    grid-column-start : 1;
    grid-column-end: 3;
}
```

OR

```
.one{
    grid-column: 1/3;
}
```

ROW GRID LINES

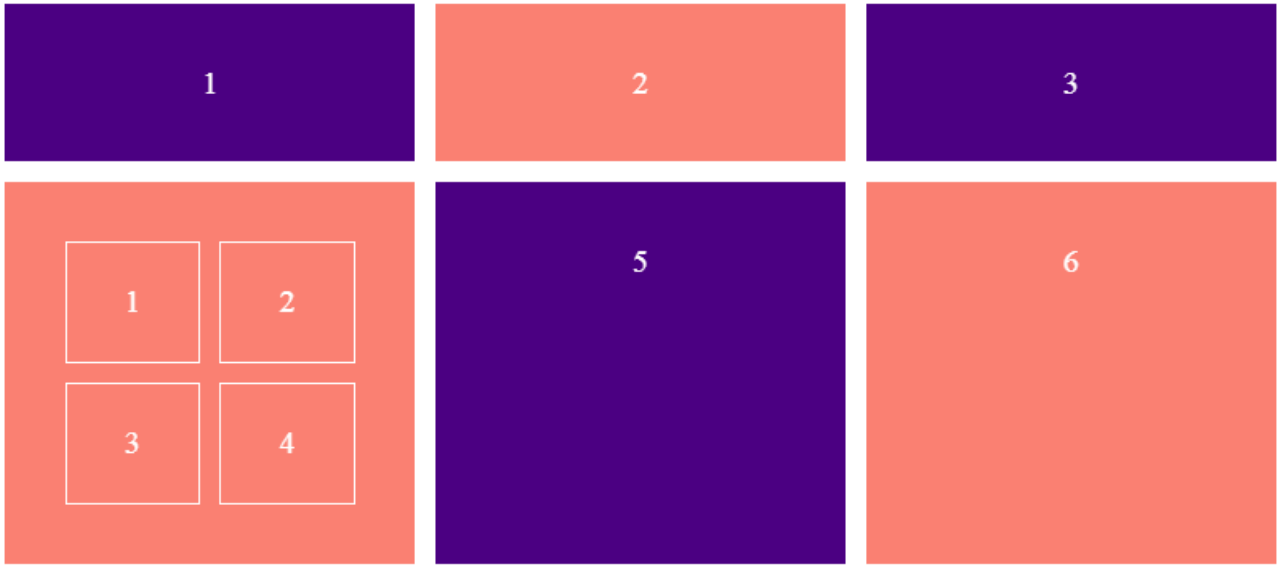



```
.two{
    grid-row-start : 2;
    grid-row-end: 4;
}
```

OR

```
.two{
    grid-row: 2/4;
}
```

NESTED GRID



NESTED GRID

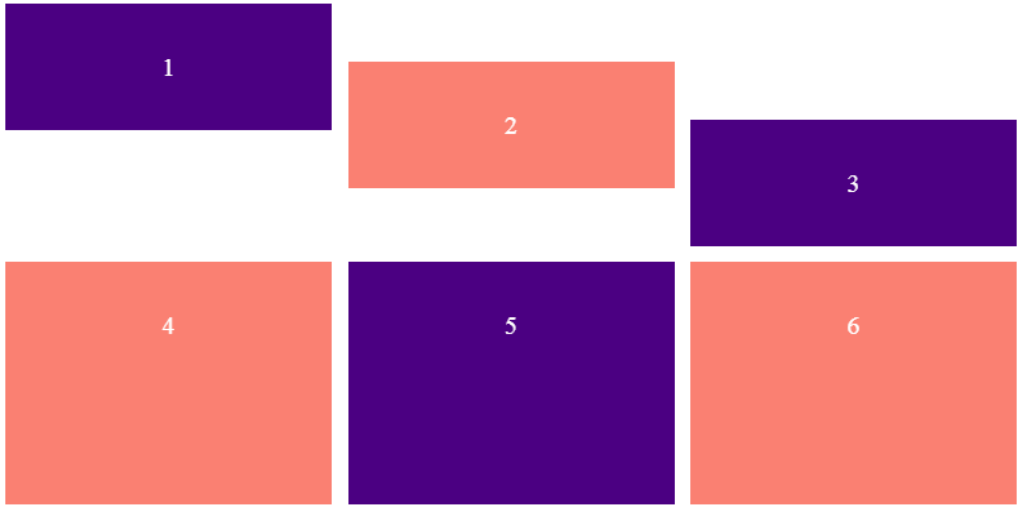
```html
<body>
  <div id="content">
    <div>1</div>
    <div>2</div>
    <div>3</div>
    <div class="nested">
      <p>1</p>
      <p>2</p>
      <p>3</p>
      <p>4</p>
    </div>
    <div>5</div>
    <div>6</div>
  </div>
</body>
```

```css
.nested{
    display: grid;
    grid-template-columns: 1fr 1fr;
    grid-gap: 10px;
}
.nested p{
    border: 1px solid #fff;
    padding: 20px;
    margin: 0;
}
```

ALIGNING + JUSTIFYING ITEMS

- Align means adjusting top and bottom.
- There are THREE values of attributes for align-self:

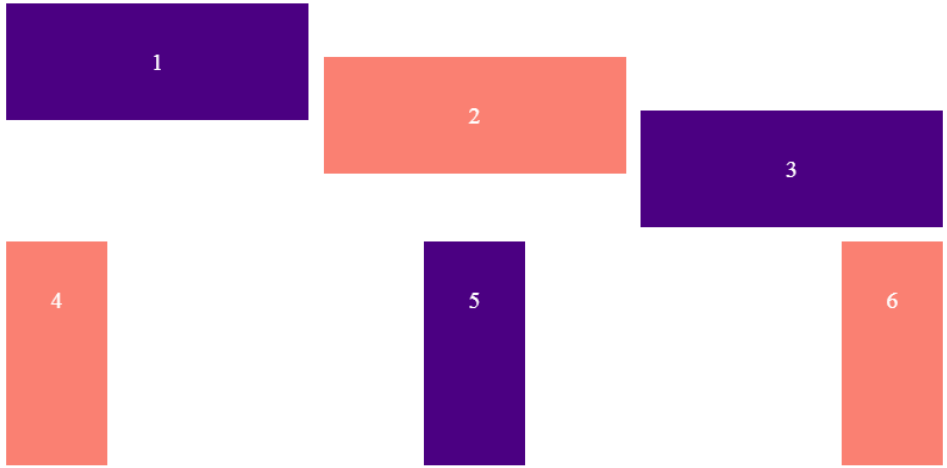
1. **align-self: start;**
2. **align-self: center;**
3. **align-self: end;**



ALIGNING + JUSTIFYING ITEMS

- Justify means adjusting left and right.
- There are THREE values of attributes for justify-self:

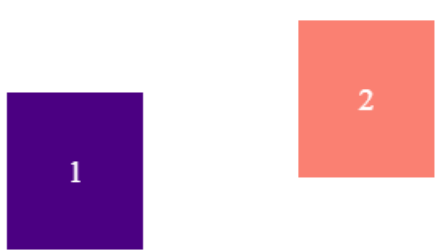
4. **justify-self: start;**
5. **justify-self: center;**
6. **justify-self: end;**



ALIGNING + JUSTIFYING ITEMS

```
.one{
    justify-self: end;
    align-self: end;
}
.two{
    justify-self: center;
    align-self: center;
}
.three{
    align-self: start;
    justify-self:start;
}
```

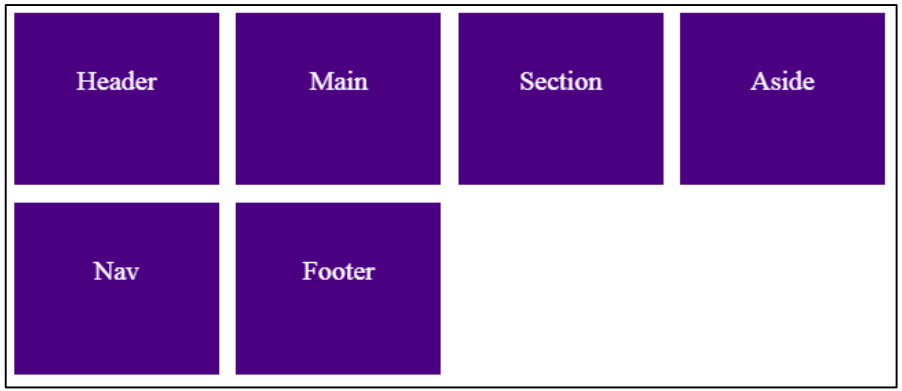
Original without align/justify-self



Adding codes

GRID-AREA

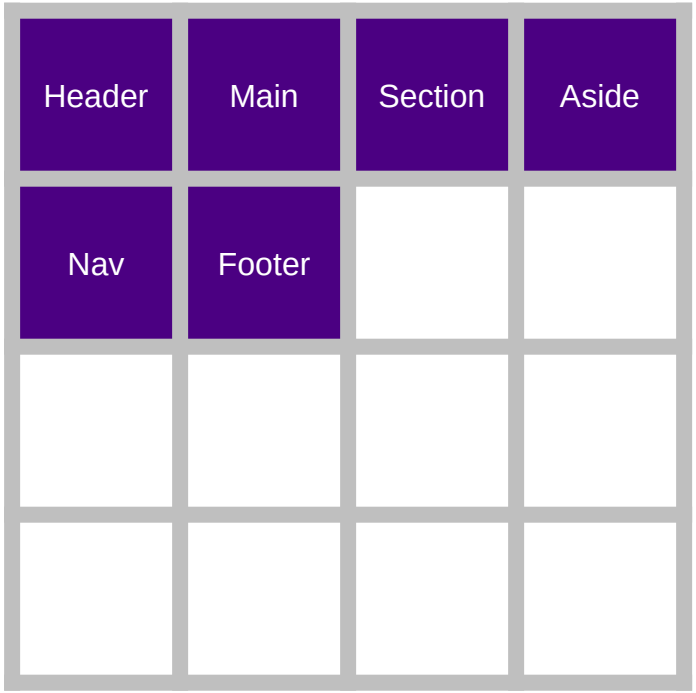
```html
<body>
  <div id="content">
    <header>Header</header>
    <main>Main</main>
    <section>Section</section>
    <aside>Aside</aside>
    <nav>Nav</nav>
    <footer>Footer</footer>
  </div>
</body>
```




```html
<style>
    body{
      color: #fff;
      font-family: 'Nunito Semibold';
      text-align: center;
    }
    #content{
      display: grid;
      grid-template-columns: repeat(4, 1fr);
      grid-auto-rows: minmax(100px, auto);
      grid-gap: 10px;
      max-width: 960px;
      margin: 0 auto;
    }

    #content > *{
      background: #4b0082;
      padding: 30px;
    }
</style>
```

GRID-AREA

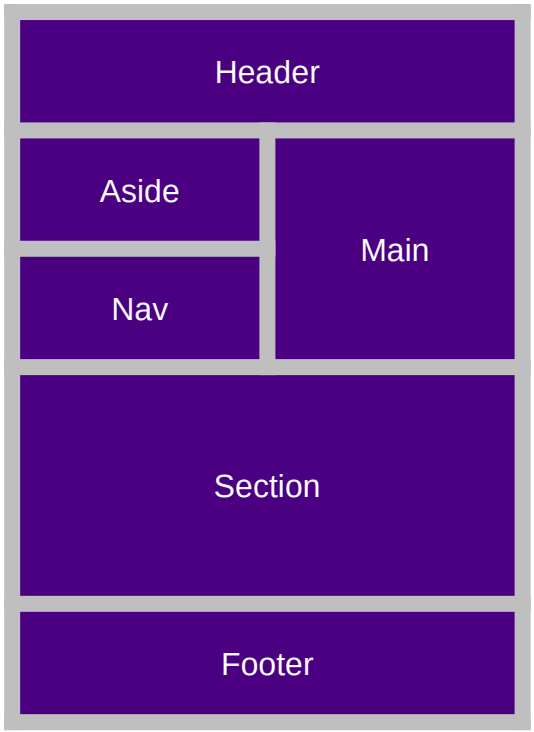


- Assign names to ALL elements in the grid by using **grid-area.**
- For example, we assign header to be named as “header”.

- **<style>**

```
...
header{
    grid-area: header;
}
main{
    grid-area: main;
}
```

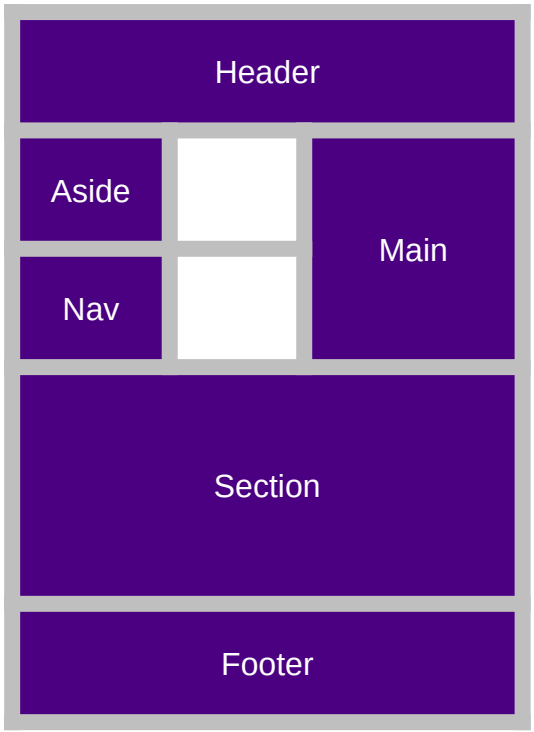
GRID-TEMPLATE-AREAS



- To re-arrange the elements in the grid into a desired template, we use **grid-template-areas.**

- ```
  #content{
    …
    grid-template-areas:
        “header header header header”
        “aside aside main main”
        “nav nav main main”
        “section section section section”
        “section section section section”
        “footer footer footer footer”
  }
  ```
  ```

GRID-TEMPLATE-AREAS



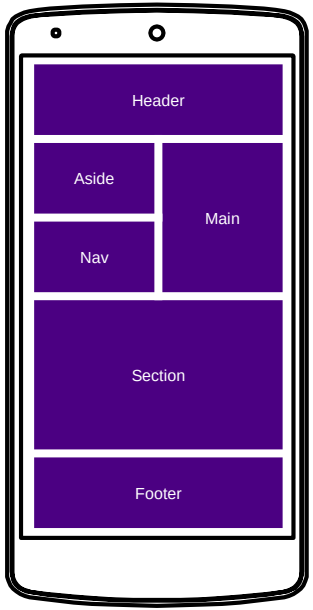
- For adding blank space, we use . (period).

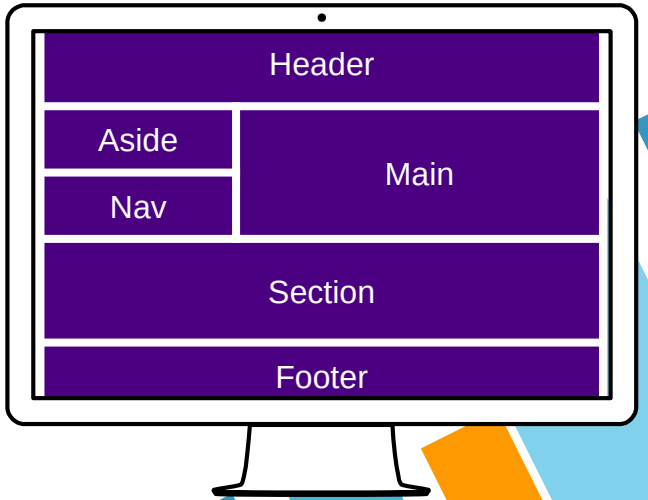
- #content{
  …
  grid-template-areas:
  “header header header header”
  “aside . main main”
  “nav . main main”
  “section section section section”
  “section section section section”
  “footer footer footer footer”
  }

RESPONSIVE LAYOUT

- There are TWO types of layout, with different screen sizes:

1. **Desktop View**
2. **Mobile View**





MEDIA QUERY MAGIC

- Media queries allow designers to deliver styles based on media type.
- The defined media types are **print, screen, handheld, braille, projection, screen, tty** and **tv.** Keyword **all** indicated that the styles apply to all media types.

- For example:

  @media *screen* and (*min-width: 480px*;){

  /* put styles for devices & browsers that pass this test inside the curly braces */

  }

  - Setting min-width to 480px to test whether the display is at least 480px wide. Hence, 768px wide would pass and get the styles but 320px would not.

MEDIA QUERY MAGIC

- 2nd example:

  @media *screen* and (*min-width: 700px*;) and (orientation: landscape;){

  /* put styles for devices & browsers that pass this test inside the curly braces */

  }

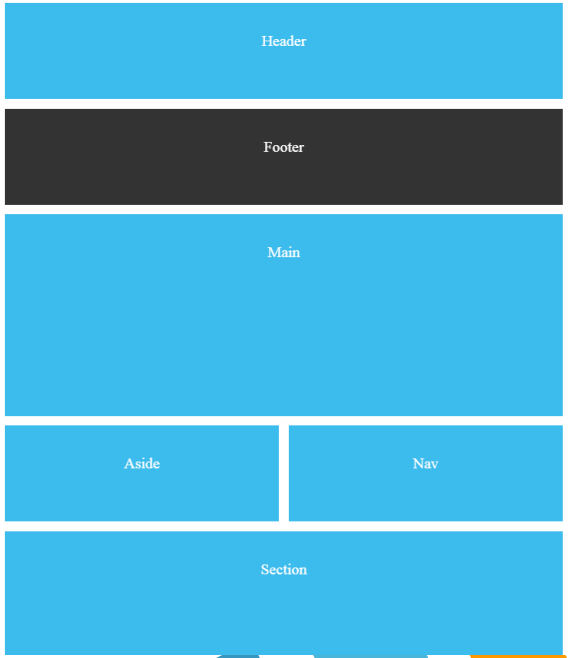
  - Check that the screen is under 700 pixels wide AND
  - Is in landscape orientation.

- The device used to browse the webpage must meet and pass all of the requirements in order to deliver the enclosed style.

RESPONSIVE LAYOUT

- Making the layout for mobile view.

```
/*mobile grid */
#content{
  display: grid;
  grid-template-columns: repeat(4, 1fr);
  grid-auto-rows: minmax(100px, auto);
  grid-gap: 10px;
  max-width: 960px;
  margin: 0 auto;
  grid-template-areas:
    "header header header header"
    "footer footer footer footer"
    "main main main main"
    "main main main main"
    "aside aside nav nav"
    "section section section section"
    "section section section section";
}
```



RESPONSIVE LAYOUT

- To make a layout responsive, we use **@media** properties.

```
/* desktop grid */
@media screen and (min-width: 760px){
  #content{
    display: grid;
    grid-template-columns: repeat(4, 1fr);
    grid-auto-rows: minmax(100px, auto);
    grid-gap: 10px;
    max-width: 960px;
    margin: 0 auto;
          grid-template-areas:
              "header header header header"
              "aside main main main"
              "nav main main main"
              "section section section section"
              "section section section section"
              "footer footer footer footer";
  }
}
```