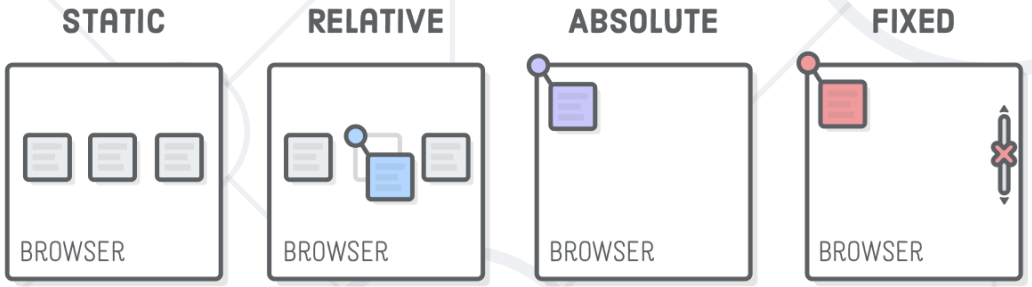
Position & Grid

CSS Positioning



SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.org

Table of Contents

1. CSS Grid
2. Position: `static`, `relative`, `absolute`, `fixed` and `sticky`
3. Positioning Properties
4. Z-index

Have a Question?

Software University

sli.do

#html-css

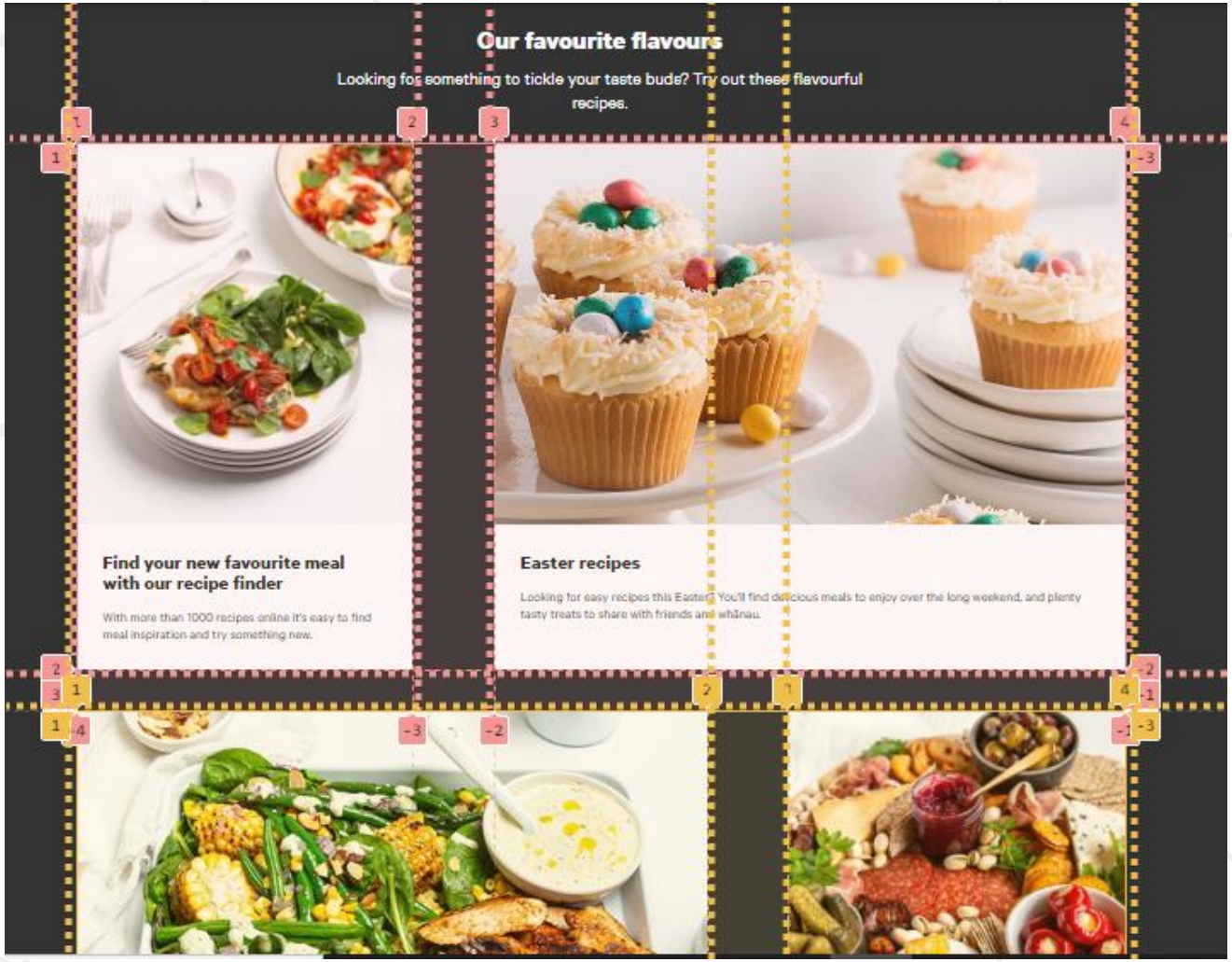
CSS Grid

Modern Layout System for the Web

CSS Grid

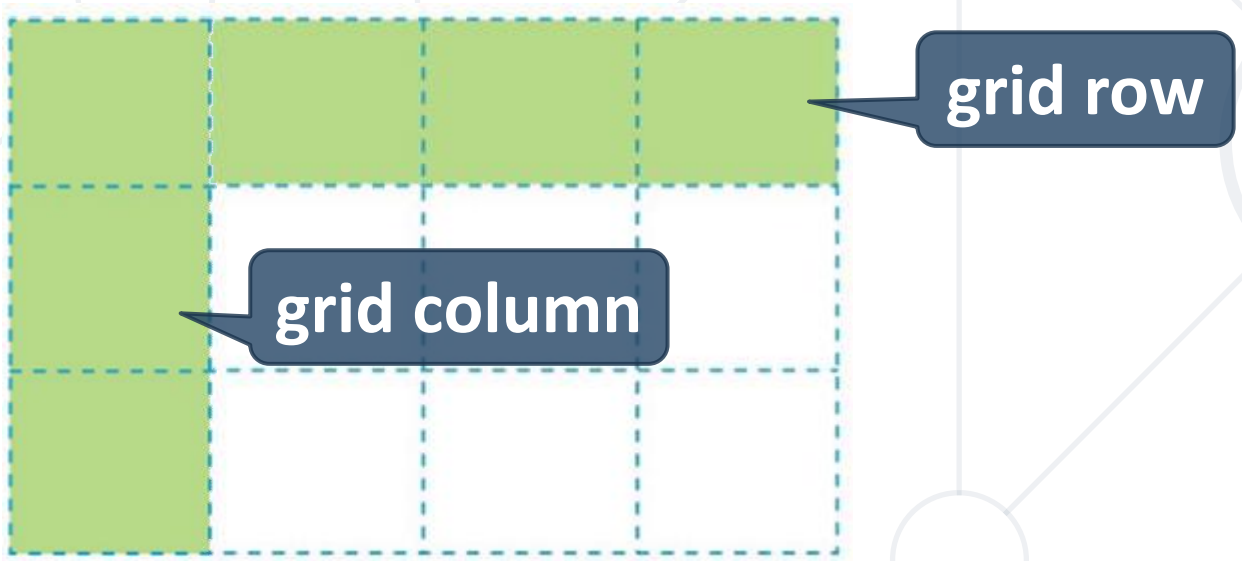
- **CSS grid** is modern CSS layout system for the Web
  - **Simple** and easy to use
  - Very **powerful**

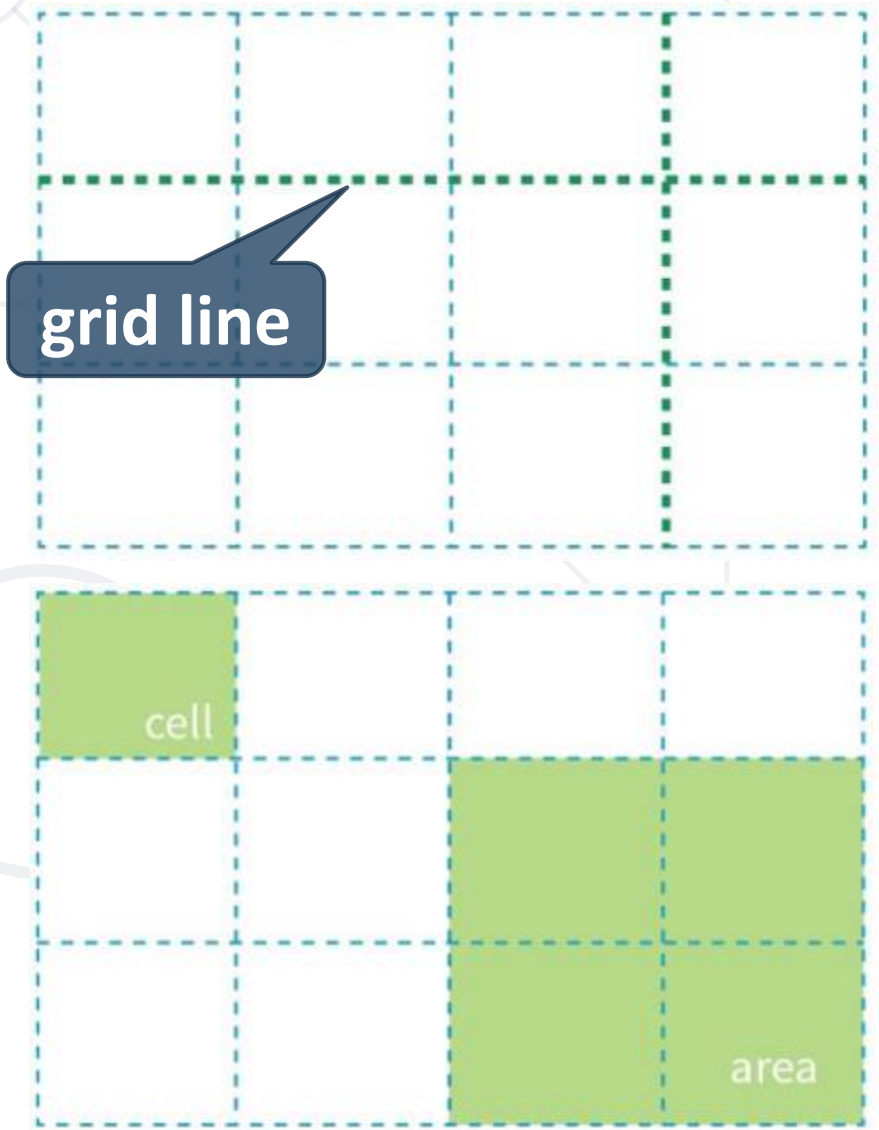
```
.grid-container {
    display: grid;
}
```



Grid Parts

- The primary parts of a grid are:
  - Grid **lines**
  - Grid **cell** and grid **areas**
  - Grid **tracks** (rows or columns)



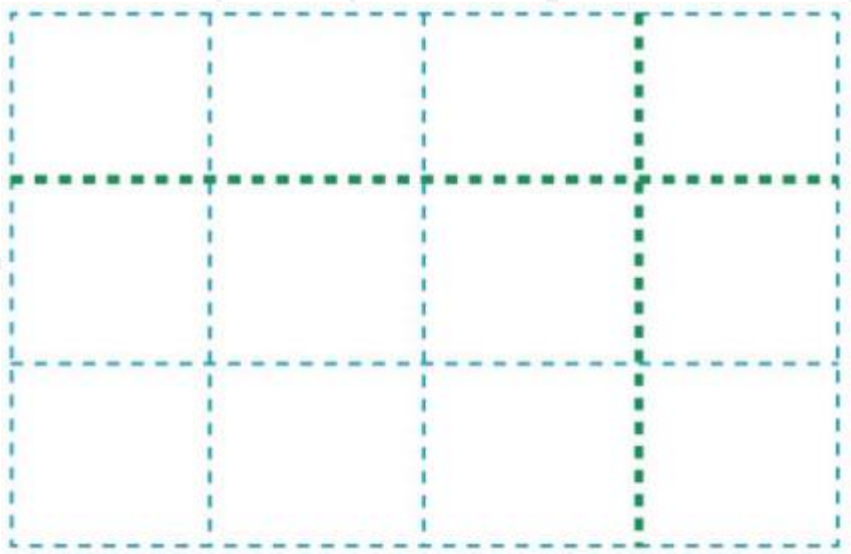


Grid Container

- An HTML element becomes a **grid container** when its display property is set to **grid** or **inline-grid**

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

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

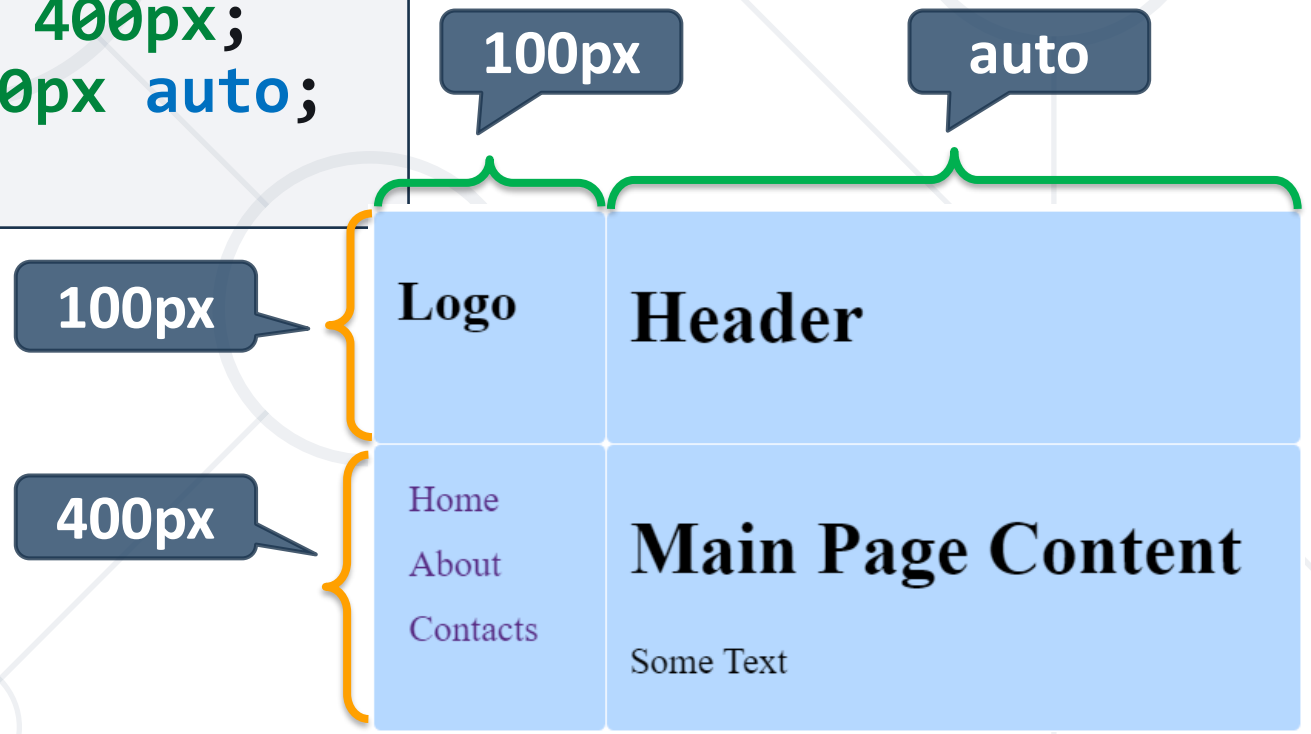


Grid Template

- The grid templates define the **look** of the grid: **count** of the **rows** and **columns** and their **size**

```
body {
   display: grid;
   grid-template-rows: 100px 400px;
   grid-template-columns: 100px auto;
}
```

- This example will create:
  - Two **rows** with **sizes**: **100px** and **400px**
  - Two **columns** with **sizes**: **100px** and **auto** size



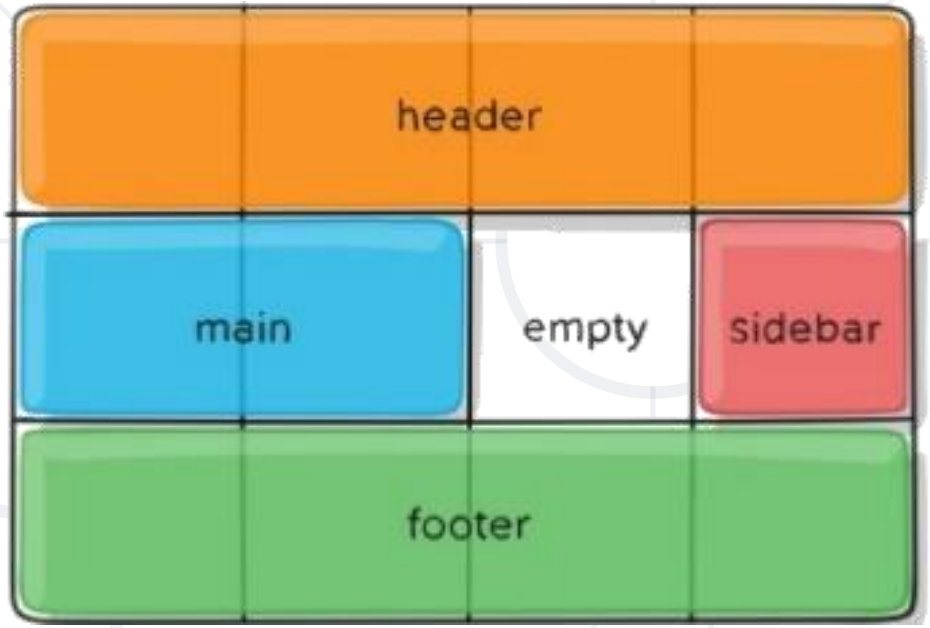
Grid Area and Grid Template Area

- **Grid area** is a rectangular area made up of one or more **adjacent grid cells**
- **Defining** grid areas in CSS:

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

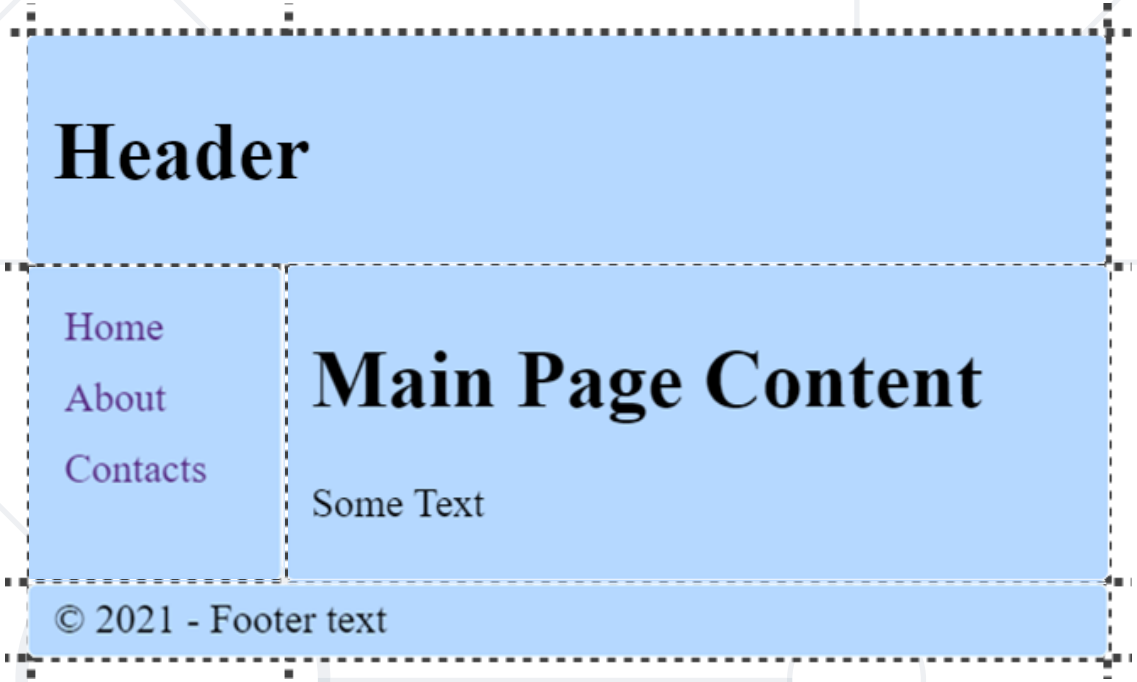
- **Referencing** the grid areas:

```
body { grid-template-areas:
 "header header header header"
 "main main empty sidebar"
 "footer footer footer footer";
```



Grid Area – Examples

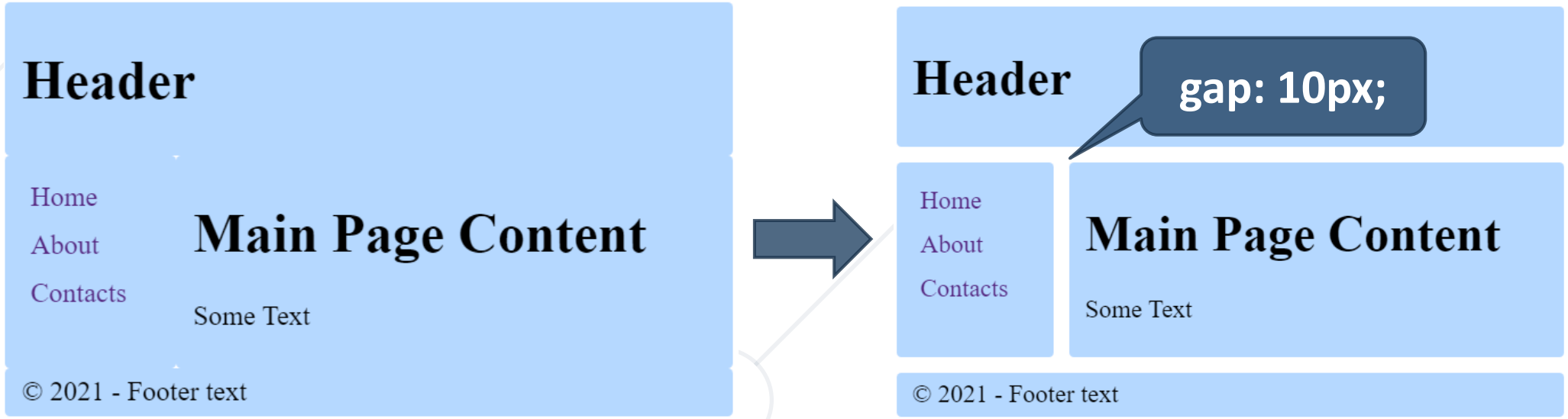
```
body { display: grid;
  grid-template-areas:
    "header header"
    "aside main"
    "footer footer";
  grid-template-columns:
    100px auto;
}

header { grid-area: header; }
aside { grid-area: aside; }
main { grid-area: main; }
footer { grid-area: footer; }
```



Gap

- **Gap** between each cell horizontally and vertically

```
body {
  display: grid;
  grid-template-columns:
    100px auto;
  gap: 10px;
  grid-template-areas:
    "header header"
    "aside main"
    "footer footer";
}
```

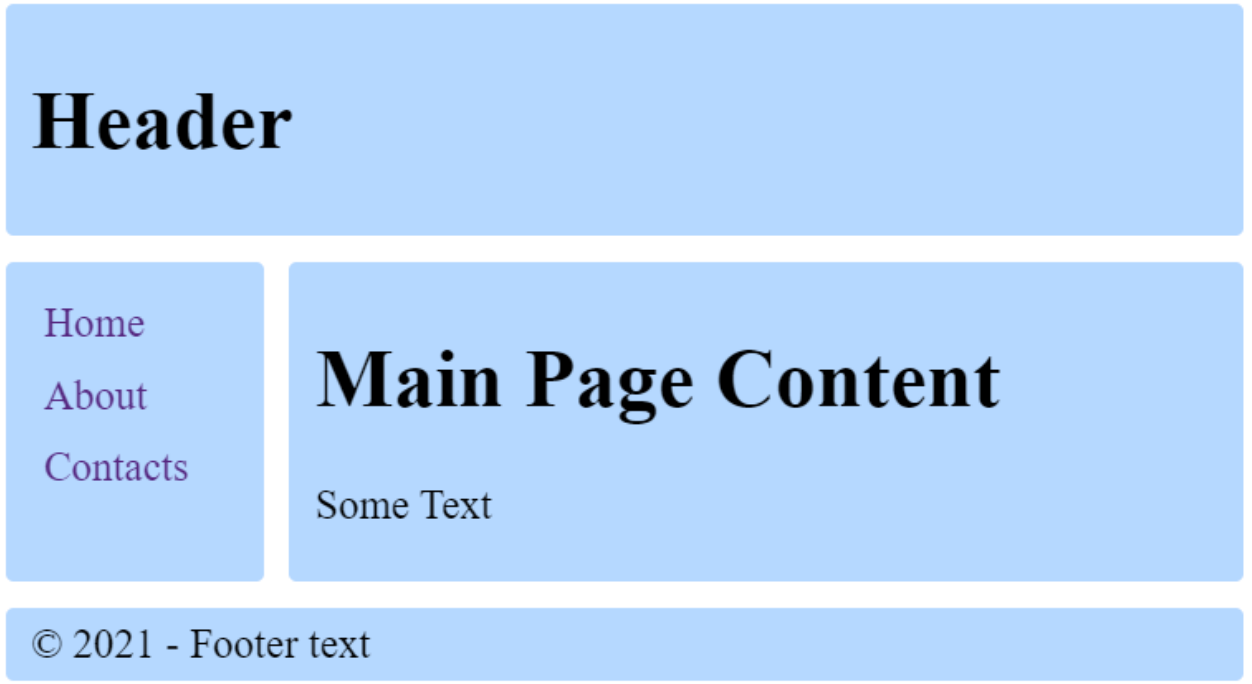


CSS Grid – Example

```
<body>
  <header><h1>Header</h1></header>
  <aside>
    <ul>
      <li>↔</li>
      <li>↔</li>
      <li>↔</li>
    </ul>
  </aside>
  <main>
    <h1>Main Page Content</h1>
    <p> Some Text
  </main>
  <footer>
    <div>© 2021 - Footer text</div>
  </footer>
</body>
```

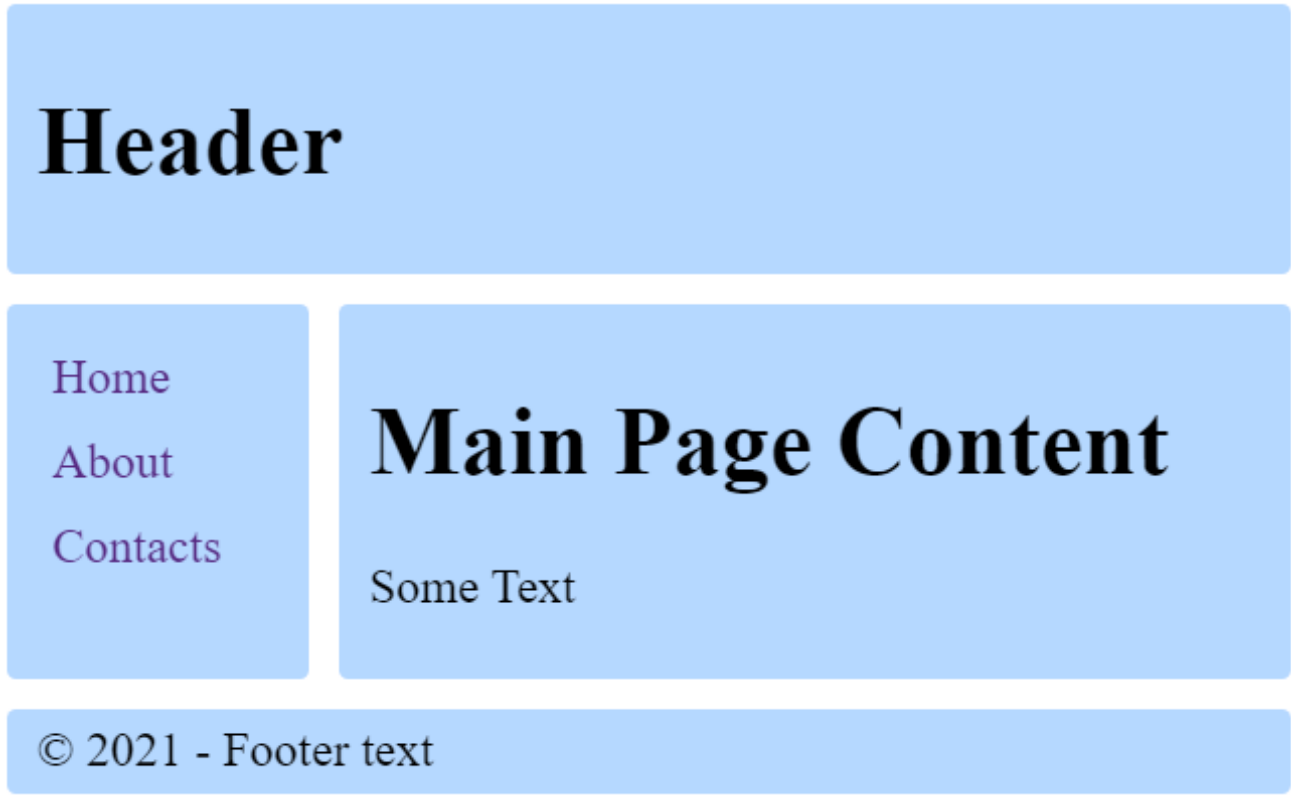
CSS Grid – Example

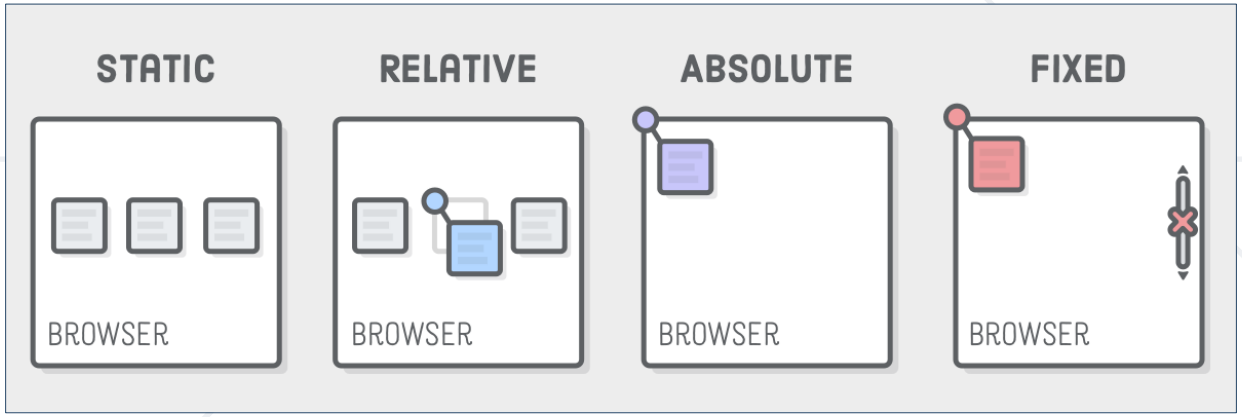
```css
body {
  display: grid;
  grid-template-areas:
    "header header"
    "aside main"
    "footer footer";
  grid-template-columns: 100px auto;
  gap: 10px;
}

header { grid-area: header; }
aside { grid-area: aside; }
footer { grid-area: footer; }
```



Demo: CSS Grid Site Layout

- https://codepen.io/snakov/pen/jOVJXVN

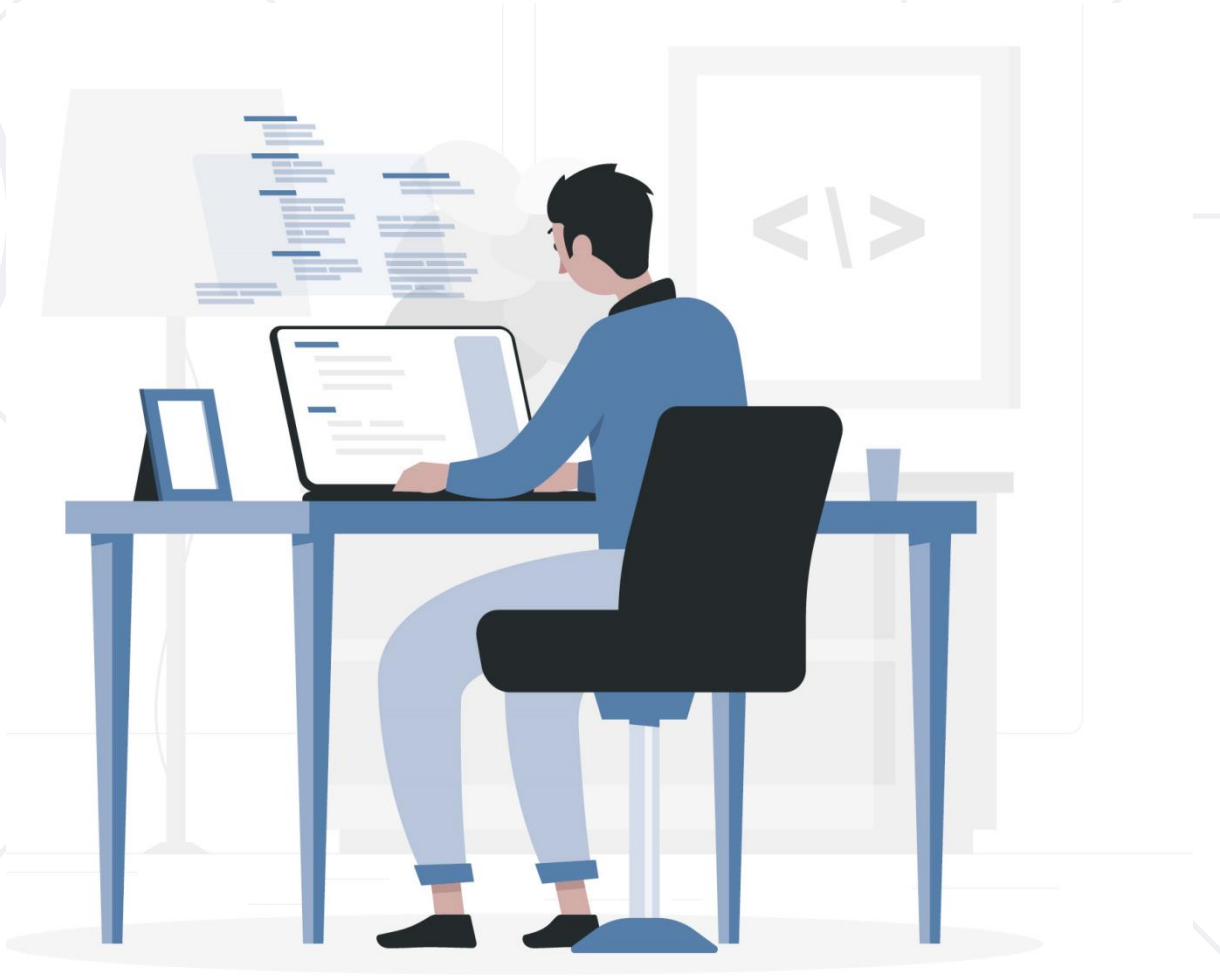




Specifies the Type of Positioning Method Used for an Element

Position

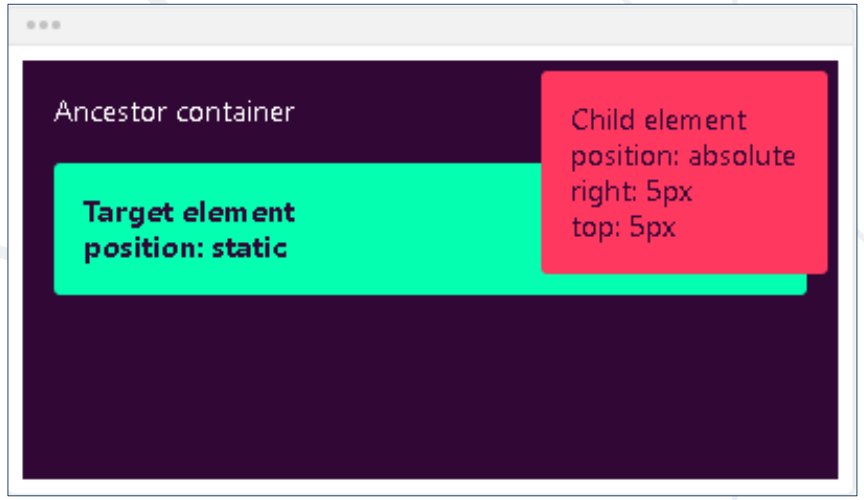
- Position properties:
  - `static`
  - `relative`
  - `absolute`
  - `fixed`
  - `sticky`



Position Static

- Static - the **default** state of every element
  - Puts the element into its **normal position** in the document layout flow
- It will **NOT** react to the following properties: **`top`**, **`bottom`**, **`left`**, **`right`**, **`z-index`**

```
div {
  position: static;
}
```

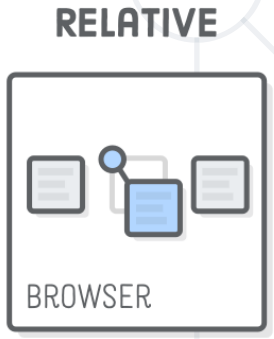


Position Relative

- It looks like static positioning, but once the positioned element has taken its place, you can then modify its final position with the **positional properties**

```
<img src="cup.jpg">
<img class="new" src="new.png">
```

```
img.new {
  position: relative;
  top: -200px;
  right: 150px;
}
```



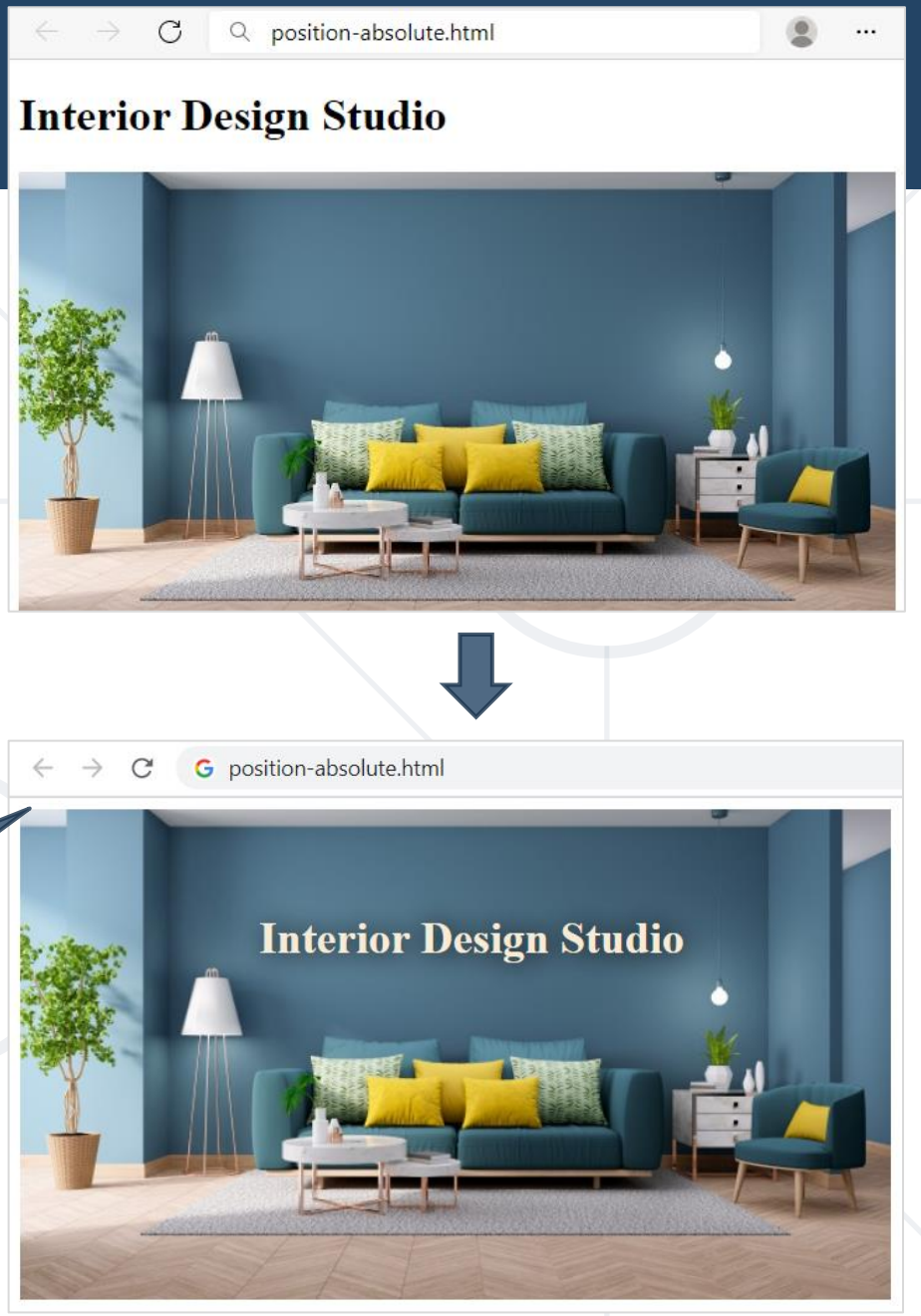
Position Absolute

- Absolute positioning → from the **upper left corner** of the parent

```
<h1>Interior Design Studio</h1>
<img src="livingroom.jpg">
```

```
h1 {
  position: absolute;
  top: 60px;
  left: 180px;
  color: antiquewhite;
  text-shadow: 1px 1px 20px black;
}
```

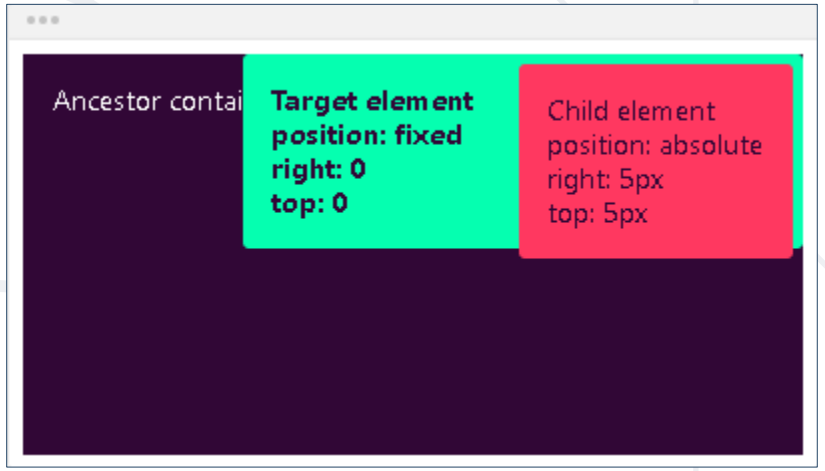
<h1> frees its original place



Position Fixed

- **Fixed** - the element will **NOT remain** in the natural flow of the page
  - **Reacts** to the positional properties
- Positions itself according to the **viewport**

```
div {
  position: fixed;
}
```



Position Sticky

- The element is positioned based on the user's **scroll position**
- A sticky element switches between **relative** and **fixed**, depending on the scroll position
- It is positioned relative until a given offset position is met in the viewport - then it "sticks" in place (like `position: fixed`)

Position Sticky – Example

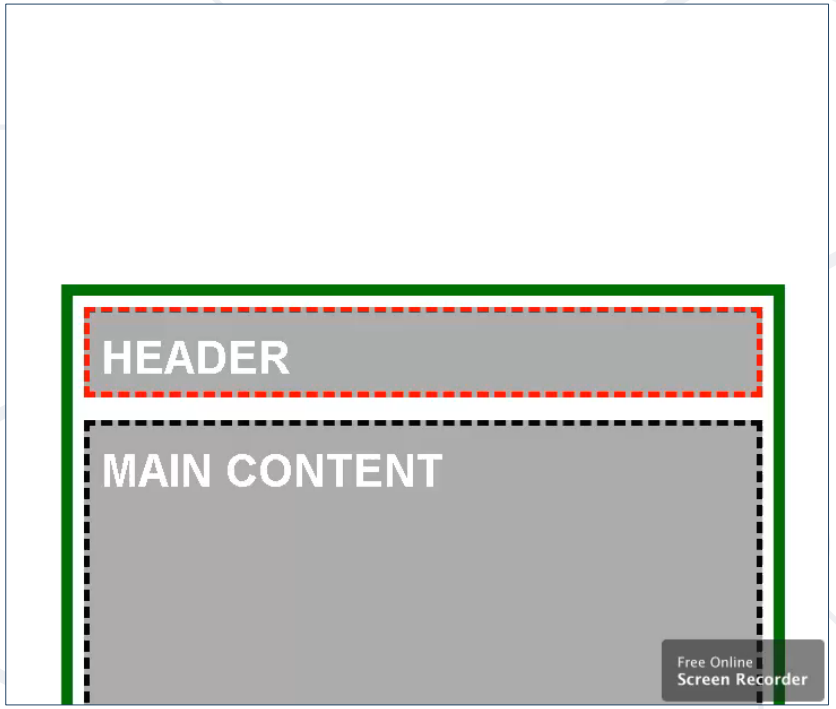
Software University

```
<main class="main-container">
    <header class="main-header">
      HEADER
    </header>
    <div class="main-content">
       MAIN CONTENT
    </div>
    <footer class="main-footer">
      FOOTER
    </footer>
</main>
```

```
.main-container {
  max-width: 600px;
  margin: 0 auto;
  border: 10px solid green;
  padding:10px;
  margin-top:40px;
}
.main-header,
.main-content,
.main-footer {
  padding: 10px;
  background: #aaa;
  border: 5px dashed #000;
  margin: 20px 0;
}
```

Position Sticky – Example

```
.main-content {
  min-height: 1000px;
}

.main-header {
  height: 50px;
  border-color: red;
  position: sticky;
  top: 0;
}
```

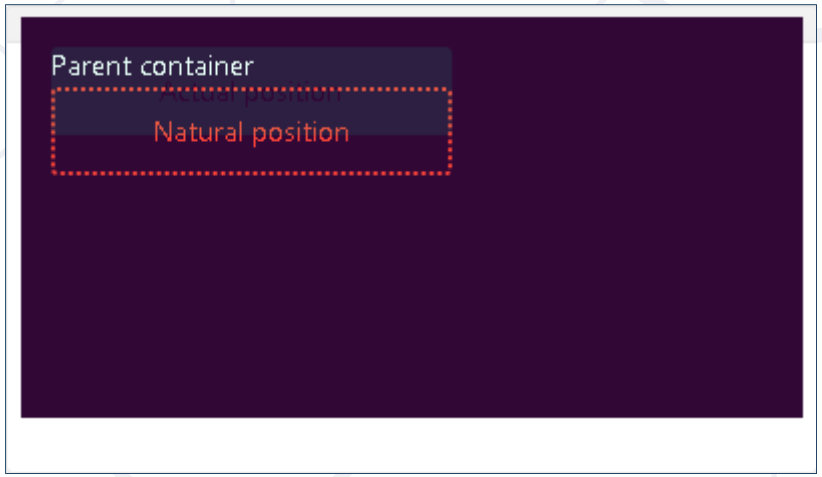


Bottom

- **Bottom** - defines the position of the element according to its bottom edge
  - **`bottom: auto;`** - the element will remain in its **natural** position

Bottom

- If the element is in position **relative**, the element will move **upwards** by the amount defined by the **bottom** value
  - `bottom: 20px;`

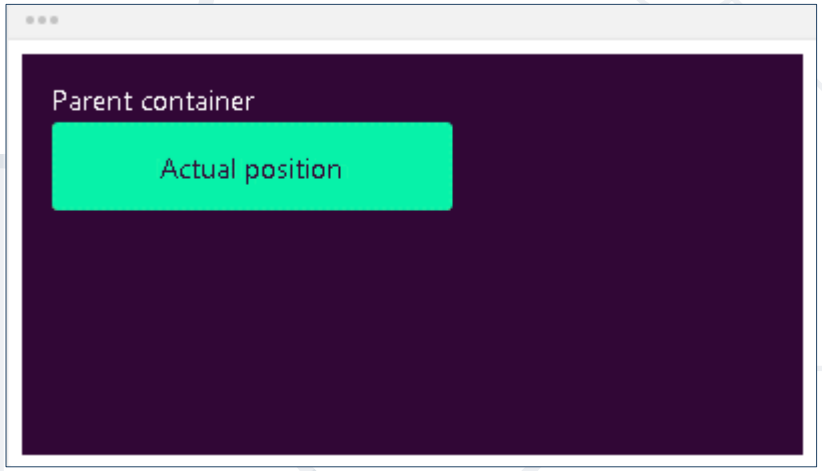


Bottom

- If the element is in position `absolute`, the element will position itself from the `bottom` of the first positioned `ancestor`
  - `bottom: 0;`

Left

- Defines the position of the element according to its left edge
  - `left: auto;` - the element will remain in its **natural** position



Left

- `left: 80px;` - if the element is in position **relative**, the element will move **left** by the amount defined by the left value

Left

- `left: -20px;` - if the element is in position **absolute**, the element will position itself from the **left** of the first positioned **ancestor**

Right

- **Right** - defines the position of the element according to its right edge
  - `right: auto;` - the element will remain in its **natural** position
  - `right: 80px;` - if the element is in position **relative**, the element will move **right** by the amount defined by the right value
  - `right: -20px;` - if the element is in position **absolute**, the element will position itself from the **right** of the first positioned **ancestor**

Top

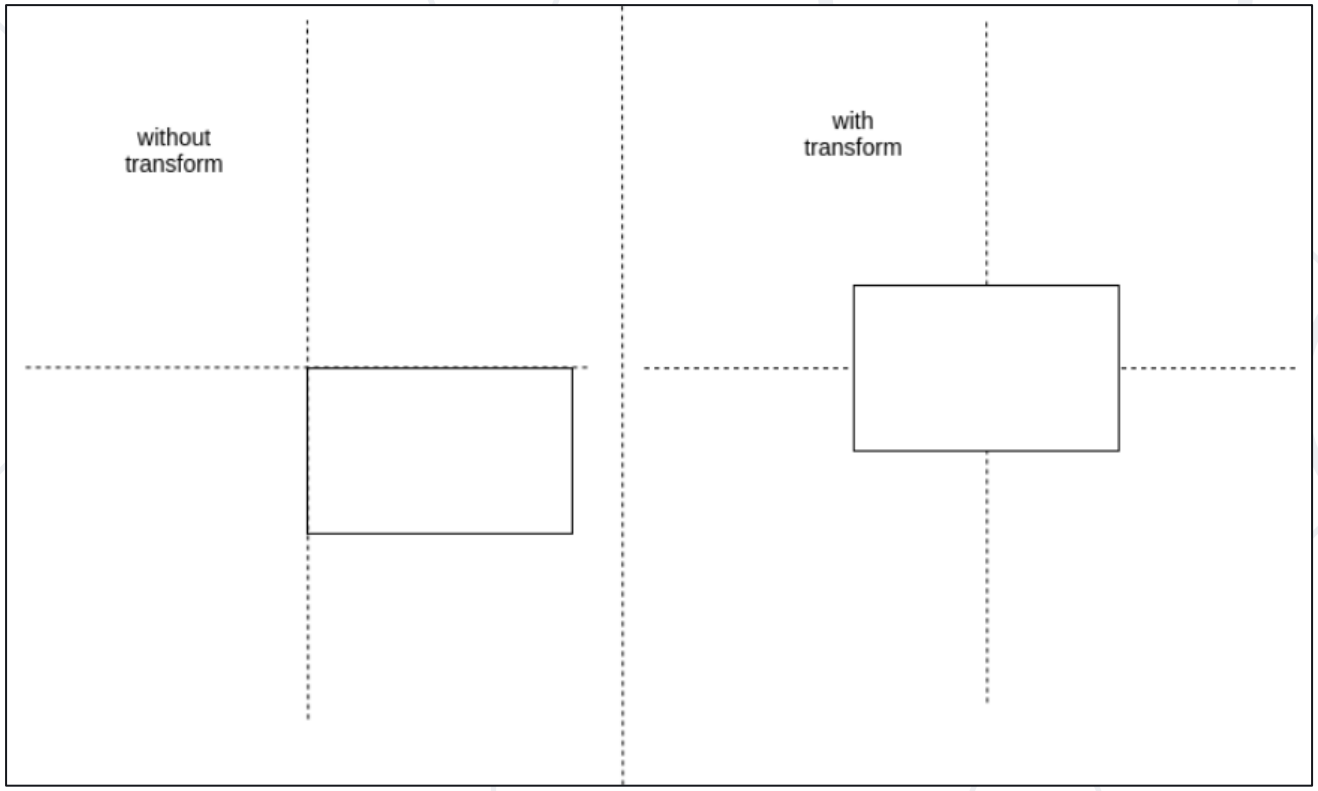
- **Top** - defines the position of the element according to its top edge
  - `top: auto;` - the element will remain in its **natural** position
  - `top: 20px;` - if the element is in position **relative**, the element will move **downwards** by the amount defined by the **top** value
  - `top: 0;` - if the element is in position **absolute**, the element will position itself from the **top** of the first positioned **ancestor**

Center

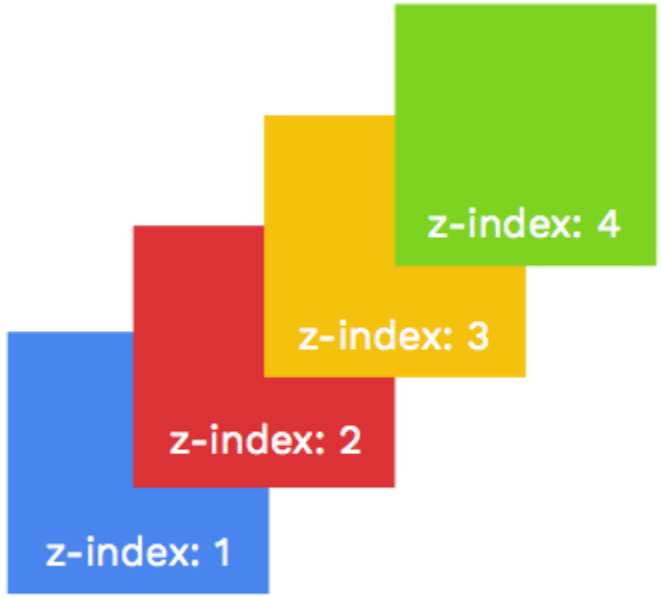
- **Center** - defines the position of the element according to center of the window
  - `position: absolute;`- it will position the element relative to its first positioned parent element
    - If it can't find one, it will be relative to the document
  - `top: 50%; left: 50%;` - the element will step out from the top left corner
  - These properties are set on the child element

Center

- `transform: translate(-50%, -50%);` - it will pull back the item with its half width and height



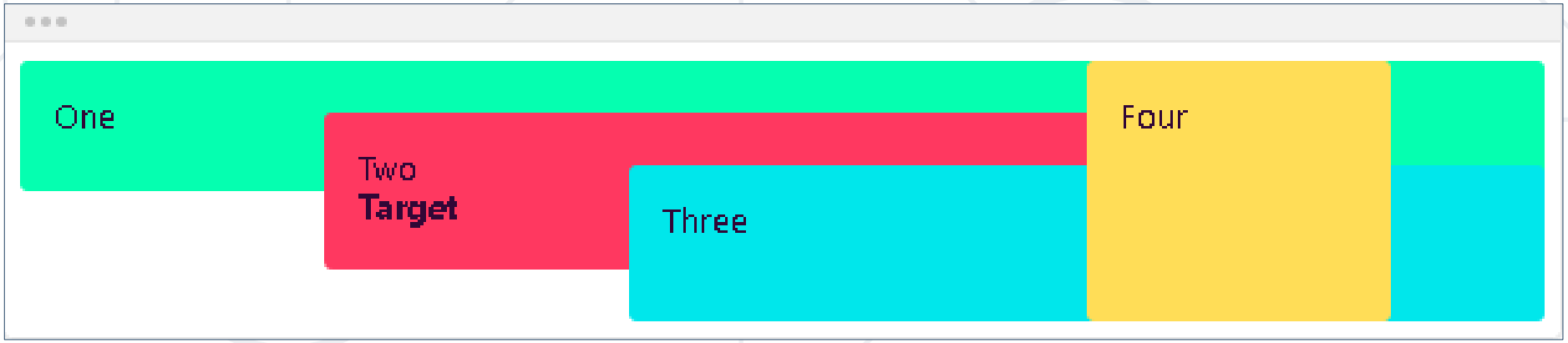

```
.parent {
position: relative;
}
.child {
position: absolute;
top: 50%;
left: 50%;
transform: translate(
-50%, -50%);
}
```



Specifies the Stack order of an Element

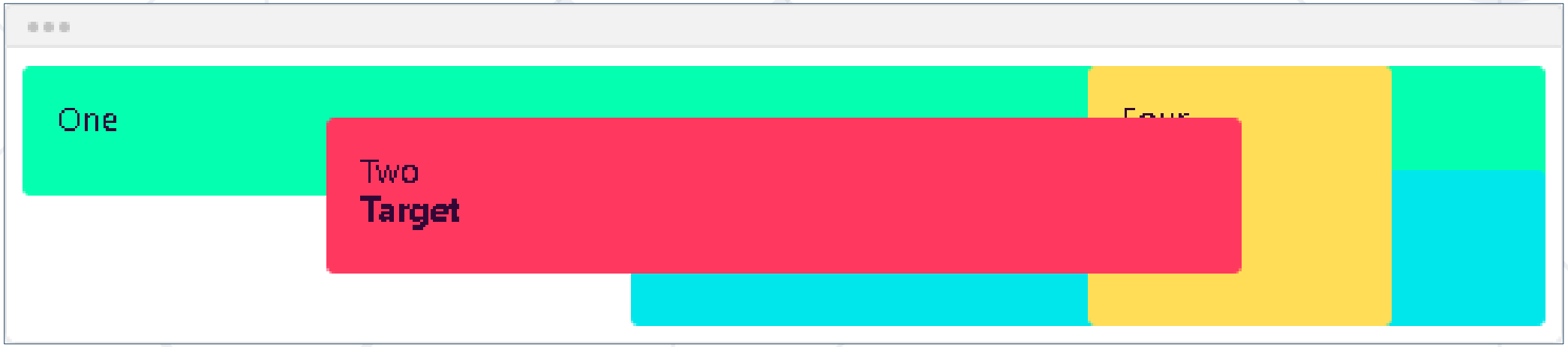
Z-index

- Defines the order of the elements on the **z-axis**. It only works on positioned elements (**anything apart from static**)
  - Default value: `z-index: auto;`
  - The order is defined by the order in the HTML code:



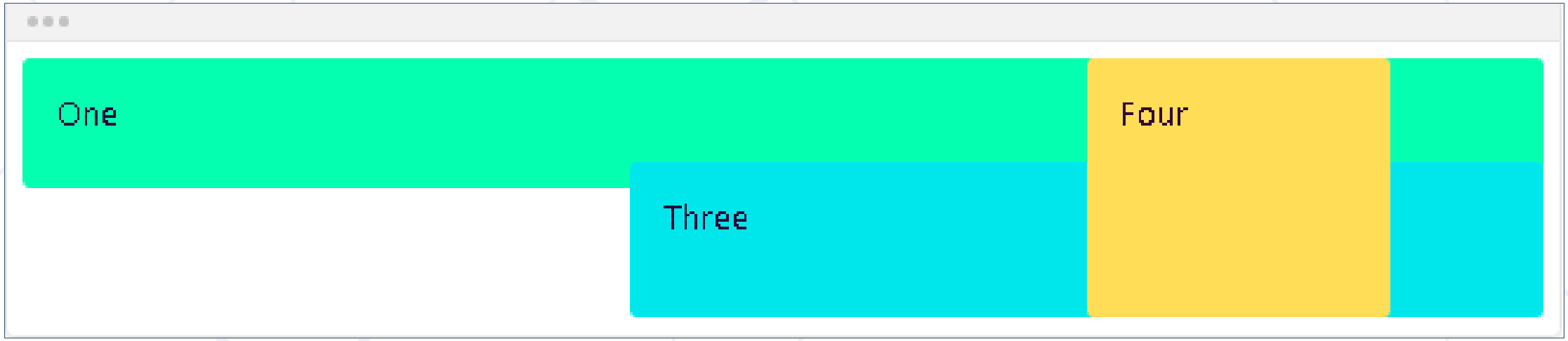
Z-index

- The z-index value is relative to the other ones
- The target element is move in **front** of its siblings
  - `z-index: 1;`



Z-index

- You can use **negative** values
- The target element is moved **behind** its **siblings**
  - `z-index: -1;`



Summary

- CSS grid
  - Lines, cell, areas, tracks
  - grid-area and grid-template-areas
- Positioning properties
- Z-Index

Questions?

SoftUni

Software University | SoftUni Creative | SoftUni Digital | SoftUni Foundation | SoftUni Kids | Finance Academy

SoftUni Diamond Partners

Software University

SUPER HOSTING .BG

Coca-Cola HBC Bulgaria

POKERSTARS
POKER | CASINO | SPORTS
a Flutter International brand

INDEAVR
Serving the high achievers

AMBITIONED

DRAFT KINGS

SOFTWARE GROUP

createX

Postbank
Решения за твоето утре

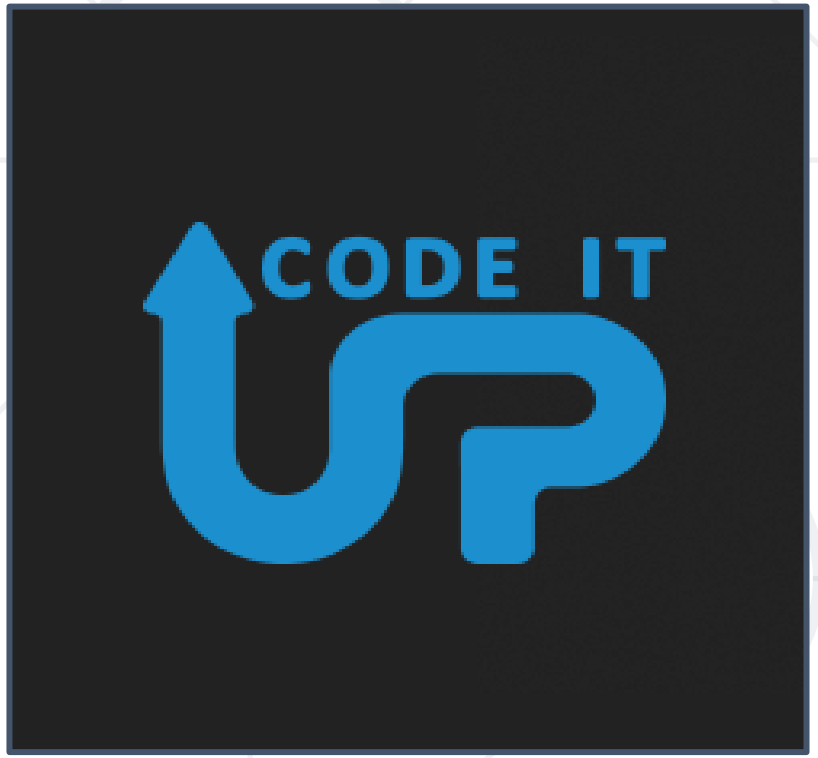
BOSCH

DXC TECHNOLOGY

SmartIT

Educational Partners

Software University



License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://about.softuni.bg/
- © Software University – https://softuni.bg

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg