YOUTH CAREER HUB

A software Training Institute in Kolkata.

A Unit of IIESR (Indian Institute Of Education and Social Reforms), Registered Under T.A 1882 of Govt of
W.B and Niti Aayog Govt of India. Dum Dum Junction, Kolkata - 700055

The Complete CSS Notes

for All type of Universities Semester Exam

YOUTH CAREER HUB

Email: contact@youthcareerhub.com | **Website:** www.youthcareerhub.com |

Phone: +91 9874296992

Index

8	**CSS Transition and Animation** • Transition properties • Creating simple CSS transitions • Keyframe animations • Animation properties	38-40
9	**CSS Media queries** • Introduction to responsive web design. • Using media queries to target different devices and screen sizes • Breakpoints and responsive design patterns. • Mobile-first vs desk top first approach.	41-43

Introduction to CSS

What is CSS and its role in web development ?

CSS stands for Cascading Style Sheets. It is stylesheet language used to describe the presentation and formatting of HTML documents. In web development, HTML is responsible for defining the structure and content of a web page, while CSS is responsible for the layout, colors, fonts, and other visual aspects. By separating the content from the presentation CSS allows web developers to create consistent and visually appealing designs across multiple web pages.

Inline CSS, internal CSS and external CSS

Inline CSS:

Inline CSS is applied directly to individual. HTML elements using the "style" attribute.

For example:

<p> <p style="color: blue; font-size: 16px;"> This is a paragraph with inline CSS. </p>

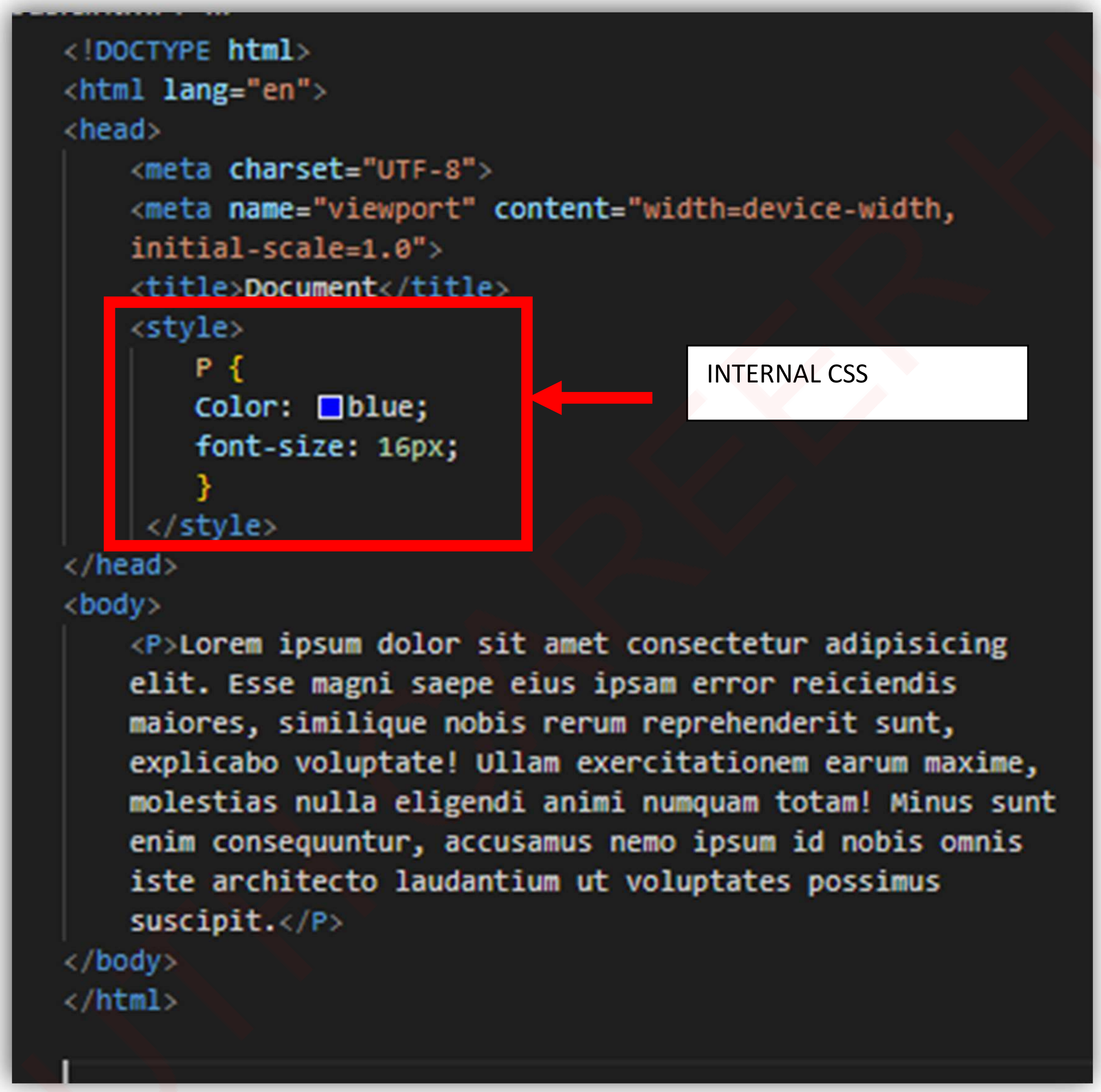
Inline CSS is generally discouraged because it mixes content with presentation, making it harder to maintain and reuse styles across the entire website.

Internal CSS

Internal CSS is included within the HTML document using the “style” element in the “head” section. It allows us to define styles for specific page elements.

For Example,

While internal CSS separates the style from the content, it may still lead to maintainability issues. as the styles are tied to the specific page. It harder to maintain and reuse styles across the entire website.

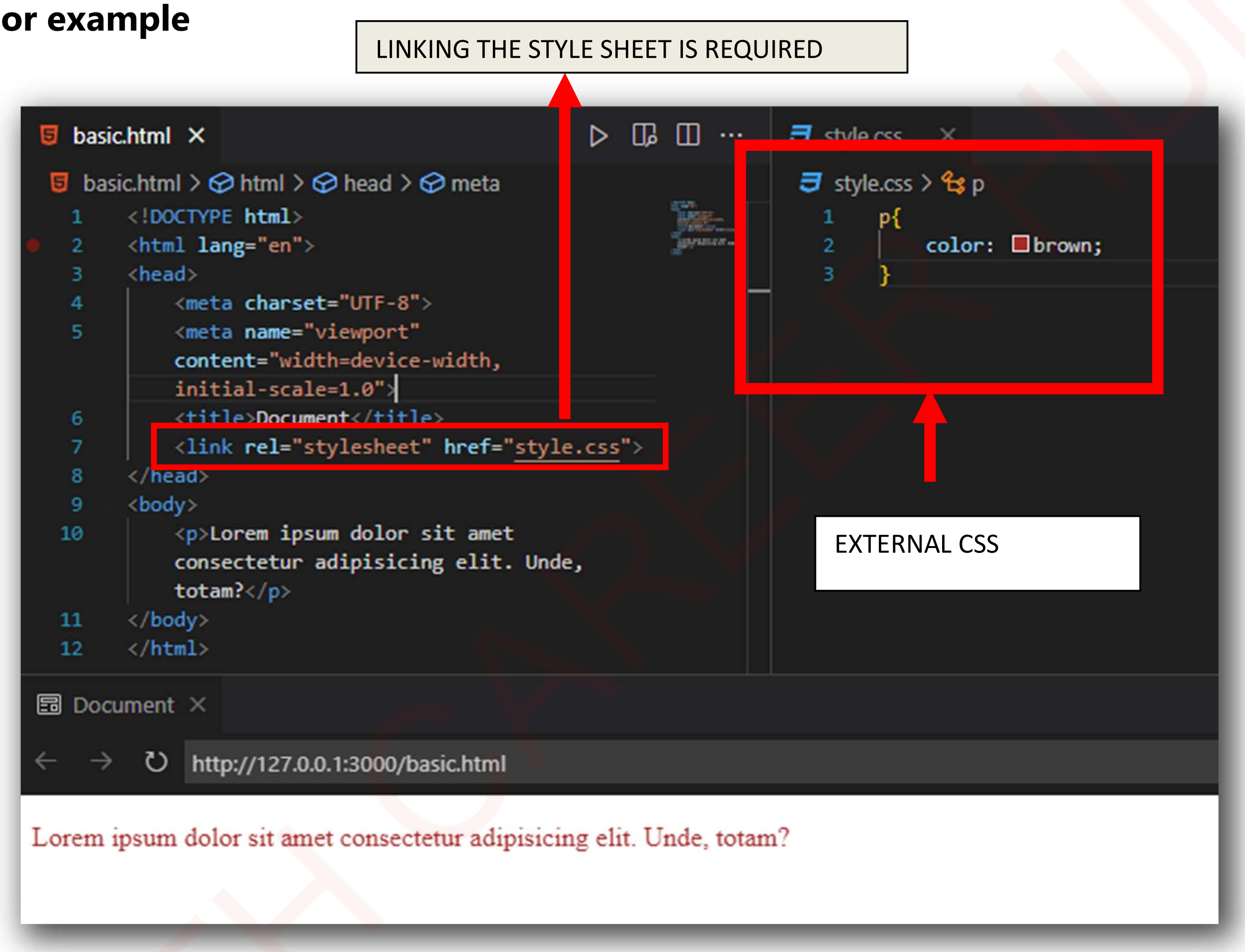


```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
    initial-scale=1.0">
    <title>Document</title>
    <style>
        P {
        Color: blue;
        font-size: 16px;
        }
     </style>
</head>
<body>
    <P>Lorem ipsum dolor sit amet consectetur adipisicing
    elit. Esse magni saepe eius ipsam error reiciendis
    maiores, similique nobis rerum reprehenderit sunt,
    explicabo voluptate! Ullam exercitationem earum maxime,
    molestias nulla eligendi animi numquam totam! Minus sunt
    enim consequuntur, accusamus nemo ipsum id nobis omnis
    iste architecto laudantium ut voluptates possimus
    suscipit.</P>
</body>
</html>
```

External CSS:

External CSS is the recommended way to apply styles to web pages. It involves creating a separate CSS file with .css extension and linking it to the HTML document the "link" element.

For example



LINKING THE STYLE SHEET IS REQUIRED

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport"
    content="width=device-width,
    initial-scale=1.0">
    <title>Document</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <p>Lorem ipsum dolor sit amet
    consectetur adipisicing elit. Unde,
    totam?</p>
</body>
</html>
```

```
p{
    color: brown;
}
```

EXTERNAL CSS

http://127.0.0.1:3000/basic.html

Lorem ipsum dolor sit amet consectetur adipisicing elit. Unde, totam?

External CSS allows for better organisation and reusability of styles across multiple pages, promoting a "consistent design throughout

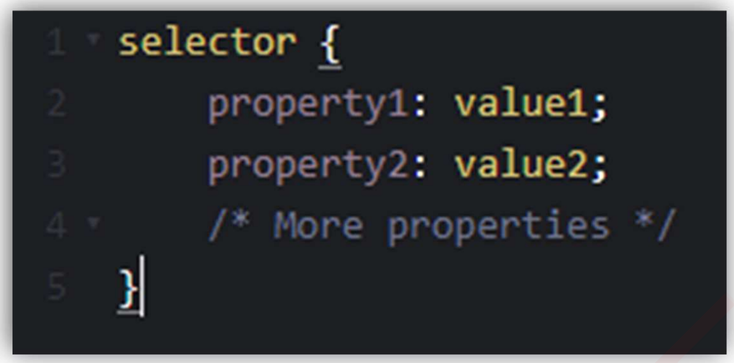
the entire website....

CSS Syntax and basic rules

Syntax: CSS rules are comprised of a selector and a declaration block.

Selector: Selects the HTML element(s) you want to style.

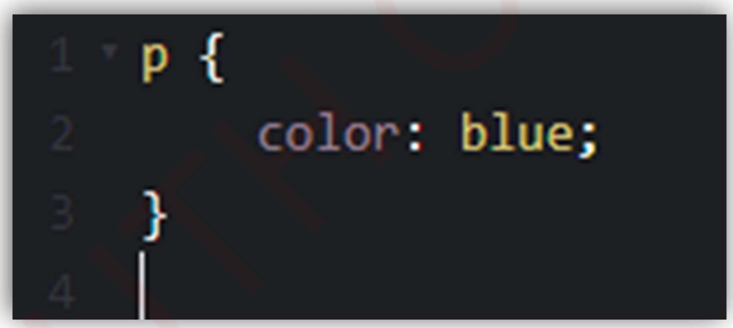
Declaration Block: Contains one or more declarations separated by semicolons.



```
selector {
    property1: value1;
    property2: value2;
    /* More properties */
}
```

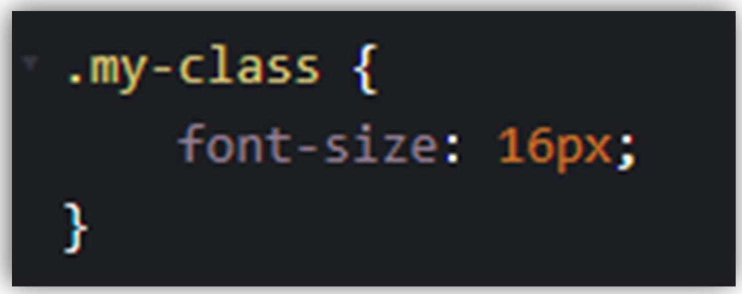
Selectors: Selectors can target elements based on their type, class, ID, attributes, or relationship with other elements.

Element Type Selector: Targets all elements of a specific type.



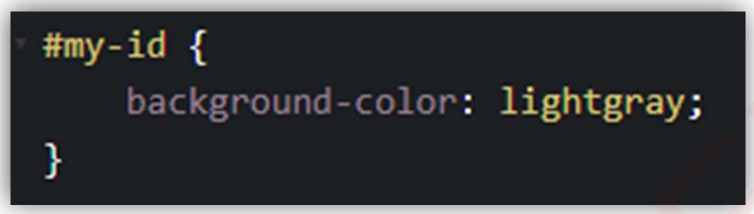
```
p {
    color: blue;
}
```

Class Selector: Targets elements with a specific class.



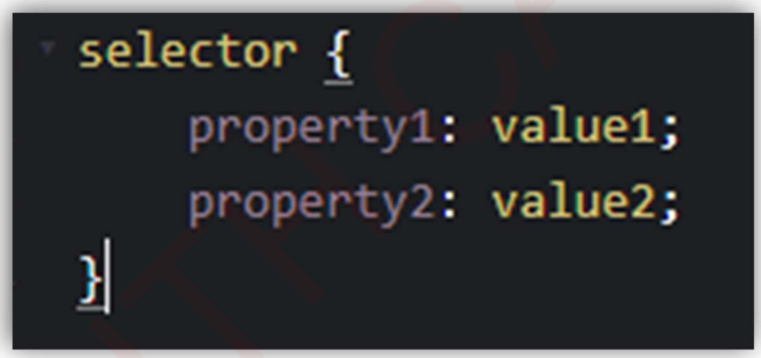
```
.my-class {
    font-size: 16px;
}
```

ID Selector: Targets a specific element with a unique ID.



```
#my-id {
    background-color: lightgray;
}
```

Properties and Values: Properties define the aspect of the element you want to style, and values specify how you want to style it.



```
selector {
    property1: value1;
    property2: value2;
}
```

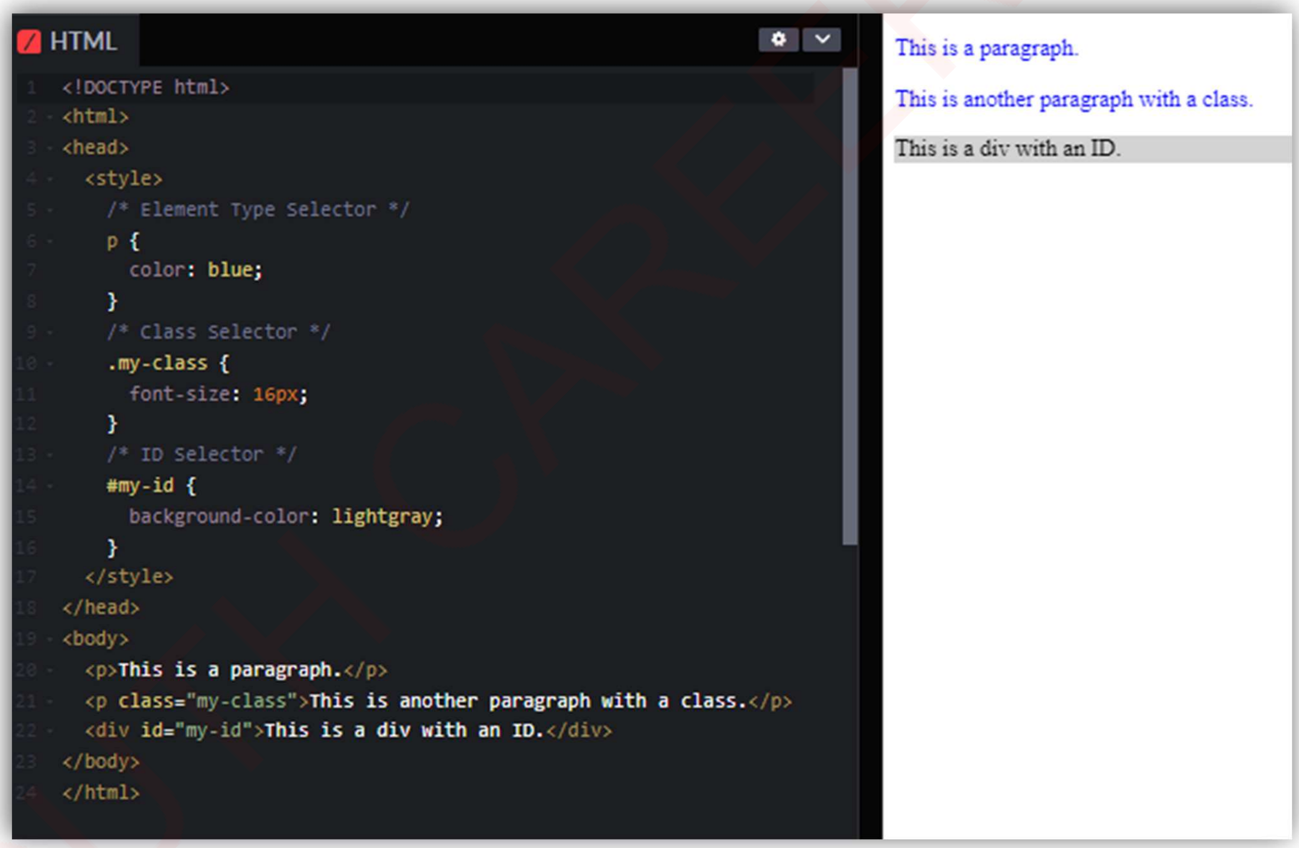
Comments: CSS supports comments, which are ignored by browsers and can be used to document your code.

/* This is a comment */

Basic Rules:

1. Use semicolons to separate property-value pairs.
2. Use colons to separate properties from values.
3. Use curly braces {} to enclose the declaration block.
4. Whitespace (spaces, tabs, newlines) is generally ignored but can improve readability.
5. Use lowercase for property names and values (although CSS is case-insensitive).

Here's an example of CSS rules applied to a HTML structure:'



```
<!DOCTYPE html>
<html>
<head>
  <style>
    /* Element Type Selector */
    p {
      color: blue;
    }
    /* Class Selector */
    .my-class {
      font-size: 16px;
    }
    /* ID Selector */
    #my-id {
      background-color: lightgray;
    }
  </style>
</head>
<body>
  <p>This is a paragraph.</p>
  <p class="my-class">This is another paragraph with a class.</p>
  <div id="my-id">This is a div with an ID.</div>
</body>
</html>
```

This is a paragraph.

This is another paragraph with a class.

This is a div with an ID.

CSS SELECTORS

What is CSS Selectors? What are the types? Explain all of the type

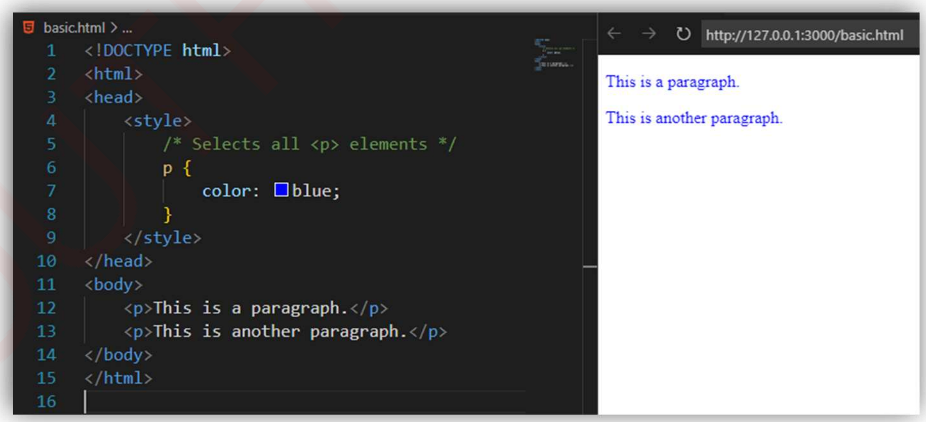
CSS selectors are used to select the content you want to style. Selectors are the part of CSS rule set. CSS selectors select HTML elements according to its id, class, type, attribute etc.

There are several different types of selectors in CSS.

- CSS Element Selector
- CSS Id Selector
- CSS Class Selector
- CSS Universal Selector
- CSS Group Selector

CSS Element Selector:

- Selects all elements of a specific type.
- Syntax: element { styles }



```
<!DOCTYPE html>
<html>
<head>
    <style>
        /* Selects all <p> elements */
        p {
            color: blue;
        }
    </style>
</head>
<body>
    <p>This is a paragraph.</p>
    <p>This is another paragraph.</p>
</body>
</html>
```

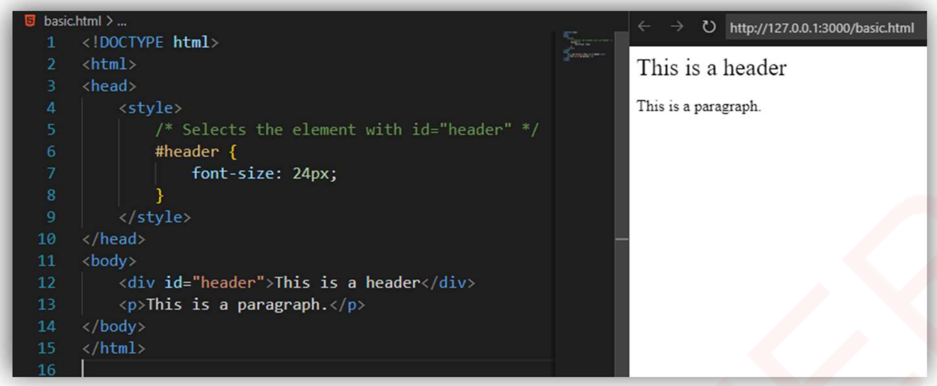
This is a paragraph.

This is another paragraph.

CSS ID Selector:

- Selects a single element with a specific ID attribute.
- Syntax: #id { styles }

Example:



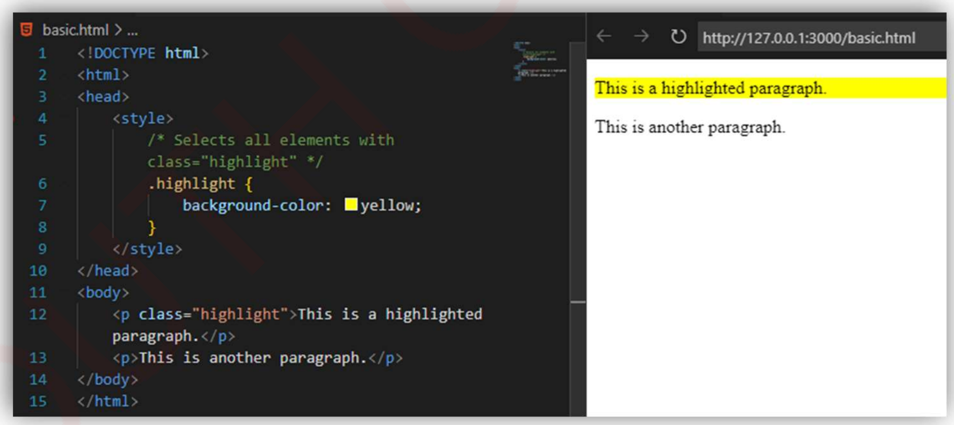
```html
<!DOCTYPE html>
<html>
<head>
    <style>
        /* Selects the element with id="header" */
        #header {
            font-size: 24px;
        }
    </style>
</head>
<body>
    <div id="header">This is a header</div>
    <p>This is a paragraph.</p>
</body>
</html>
```

```
This is a header
This is a paragraph.
```

CSS Class Selector:

- Selects elements with a specific class attribute.
- Syntax: .class { styles }

Example:

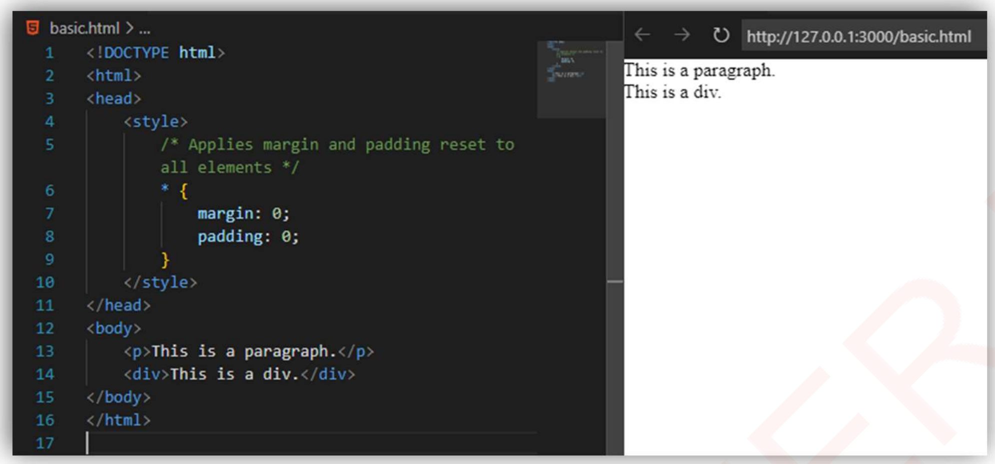


```html
<!DOCTYPE html>
<html>
<head>
    <style>
        /* Selects all elements with class="highlight" */
        .highlight {
            background-color: yellow;
        }
    </style>
</head>
<body>
    <p class="highlight">This is a highlighted paragraph.</p>
    <p>This is another paragraph.</p>
</body>
</html>
```

```
This is a highlighted paragraph.
This is another paragraph.
```

CSS Universal Selector:

- Selects all elements on the page.
- Syntax: * { styles }



basic.html > ...

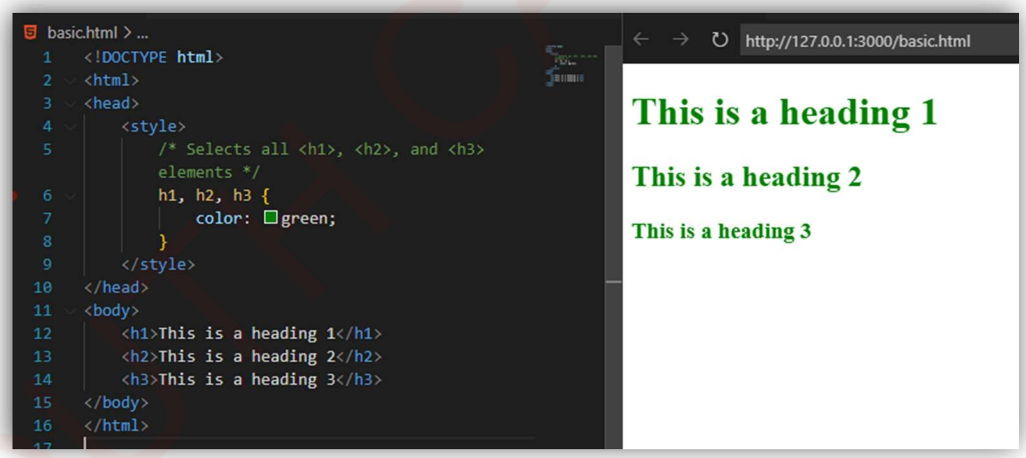
```html
<!DOCTYPE html>
<html>
<head>
    <style>
        /* Applies margin and padding reset to all elements */
        * {
            margin: 0;
            padding: 0;
        }
    </style>
</head>
<body>
    <p>This is a paragraph.</p>
    <div>This is a div.</div>
</body>
</html>
```

http://127.0.0.1:3000/basic.html

```
This is a paragraph.
This is a div.
```

CSS Group Selector:

- Selects multiple selectors and applies the same styles to all of them.
- Syntax: selector1, selector2, selector3 { styles }



basic.html > ...

```html
<!DOCTYPE html>
<html>
<head>
    <style>
        /* Selects all <h1>, <h2>, and <h3> elements */
        h1, h2, h3 {
            color: green;
        }
    </style>
</head>
<body>
    <h1>This is a heading 1</h1>
    <h2>This is a heading 2</h2>
    <h3>This is a heading 3</h3>
</body>
</html>
```

http://127.0.0.1:3000/basic.html

```
This is a heading 1
This is a heading 2
This is a heading 3
```

Pseudo class & elements

Pseudo-classes and pseudo-elements are special keywords in CSS that specify a state or a part of an element to style. They are used to target elements based on information that is not contained in the HTML markup, such as user interaction or the position of an element in the document tree.

Pseudo-classes:

- Pseudo-classes are used to define the special state of an element.
- Syntax: selector:pseudo-class { styles }

Examples:

:hover: Applies styles when the mouse is over the element.

```
a:hover {

color: red;

}
```

:first-child: Selects the first child of an element.

```
li:first-child {

font-weight: bold;

}
```

:nth-child(): Selects elements based on their position in a group of siblings.

```
tr:nth-child(even) {

background-color: lightgray;

}
```

Pseudo-elements:

- Pseudo-elements are used to style specific parts of an element.
- Syntax: selector::pseudo-element { styles }

Examples:

::before and ::after: Inserts content before and after the content of an element.

- p::before {content: "Before ";}
- p::after {content: " After";}

::first-line and ::first-letter: Styles the first line or first letter of an element.

- p::first-line {font-weight: bold;}
- p::first-letter {font-size: 200%;}

::selection: Styles the portion of an element that is selected by the user.

- ::selection {background-color: yellow;}

Difference between pseudo class and elements

Aspect	Pseudo-classes	Pseudo-elements
Definition	Used to define the special state of an element.	Used to style specific parts of an element.
Syntax	selector:pseudo-class { styles }	selector::pseudo-element { styles }
Examples	:hover, :first-child, :nth-child()	::before, ::after, ::first-line, ::first-letter, ::selection
Purpose	Target elements based on user interaction or position in the document.	Style specific parts of an element's content or structure.
Interaction	Responds to user actions like hovering, clicking, etc.	Styles specific parts of an element without adding extra markup.
Content	Does not add or change content; modifies existing elements.	Can insert content before or after an element's content.
Browser Support	Widely supported across modern browsers.	Widely supported across modern browsers.

CSS BOX MODEL

What is the CSS box model?

The CSS box model describes the rectangular boxes that are generated for elements in the document tree. These boxes consist of content, padding, border, and margin:

- **Content:** The actual content of the box, where text and images appear.
- **Padding:** The transparent area around the content, inside the border.
- **Border:** A border surrounding the padding (if any) and content.
- **Margin:** The transparent area outside the border, between the border and other elements.

Each of these components can be styled individually to create the desired layout and appearance for elements on a webpage. Understanding the box model is crucial for designing layouts and positioning elements correctly on a webpage using CSS.

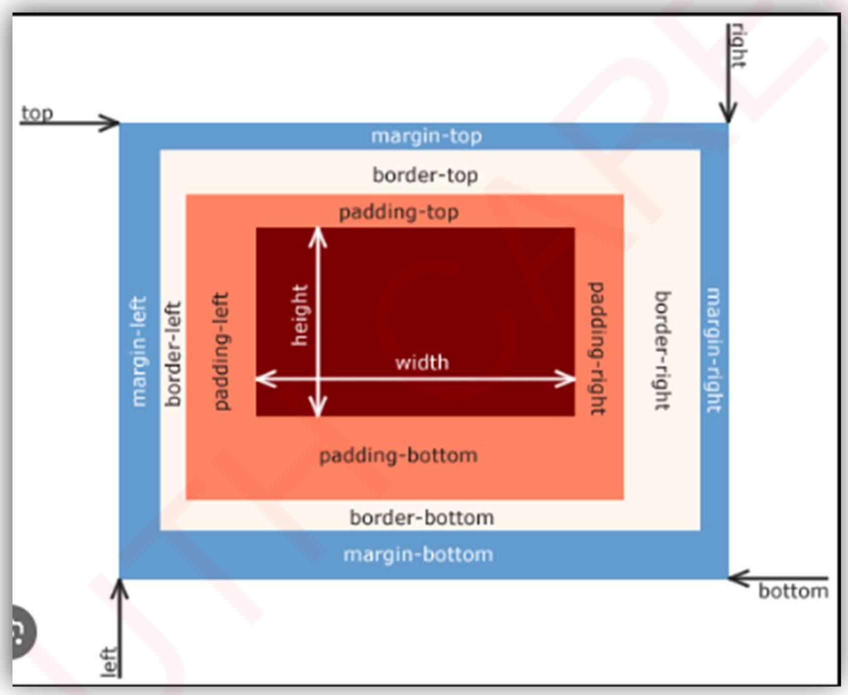
Explain the component of Box model in CSS?

The box model is a fundamental concept in CSS that defines how web browsers render HTML elements. It's a mental model that breaks down an element on a webpage into four main components: content, padding, border, and margin. Each component contributes to the overall size and position of the element.

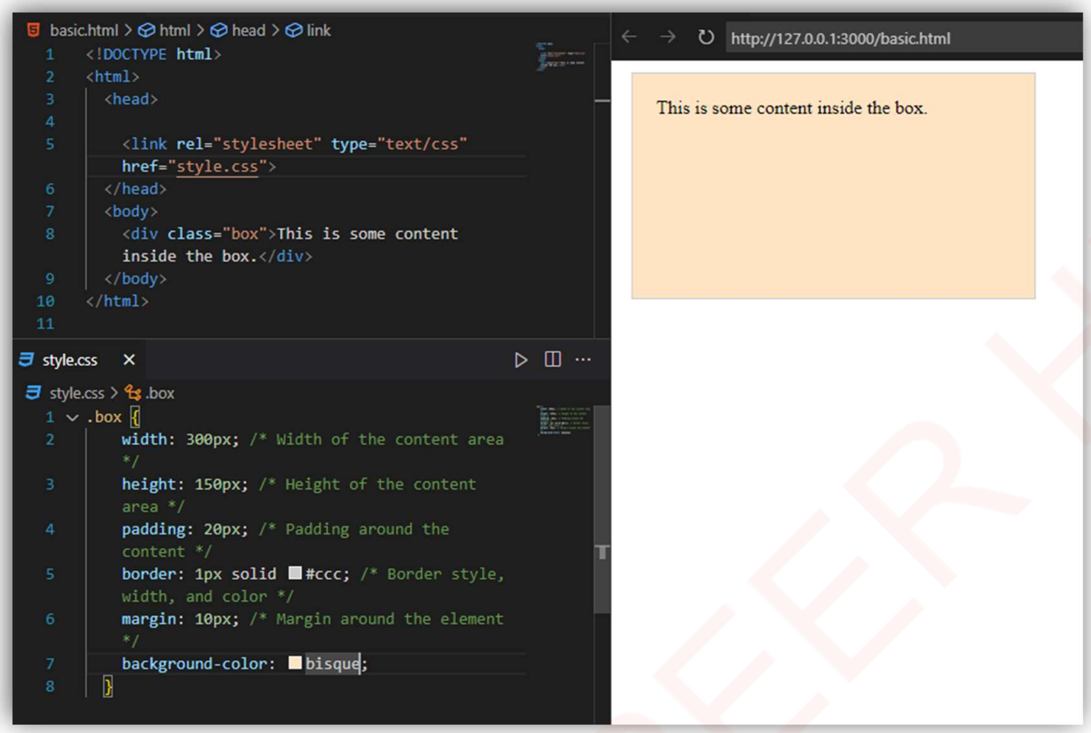
Here's a breakdown of the components with a diagram and code example:

Components

- **Content:** This is the core area where the actual content of the element resides, like text, images, or buttons.
- **Padding:** This is the transparent area around the content that creates extra space between the content and the border. You can control the padding using the padding property in CSS.
- **Border:** This is the decorative line surrounding the padding and content. You can control the border's style, width, and color using the border properties in CSS.
- **Margin:** This is the transparent area surrounding the border that creates space between the element and other elements on the page. You can control the margin using the margin property in CSS.



Code Example:


basic.html > html > head > link
style.css
style.css > .box
http://127.0.0.1:3000/basic.html

```html
<!DOCTYPE html>
<html>
  <head>

    <link rel="stylesheet" type="text/css"
    href="style.css">
  </head>
  <body>
    <div class="box">This is some content
    inside the box.</div>
  </body>
</html>

```

```css
.box {
    width: 300px; /* Width of the content area
    */
    height: 150px; /* Height of the content
    area */
    padding: 20px; /* Padding around the
    content */
    border: 1px solid #ccc; /* Border style,
    width, and color */
    margin: 10px; /* Margin around the element
    */
    background-color: bisque;
  }
```

```
This is some content inside the box.
```

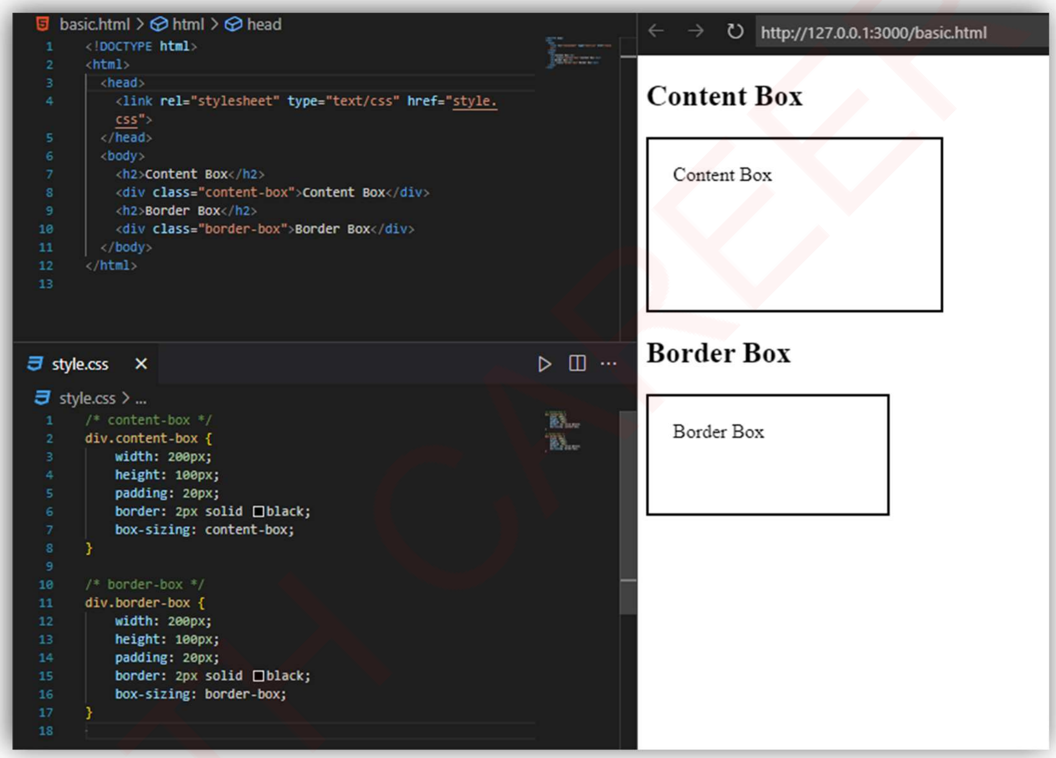
What is the difference between box-sizing: content-box; and box-sizing: border-box;?

The box-sizing property in CSS controls how the width and height of an element are calculated. There are two main values for the box-sizing property: content-box and border-box.

content-box: This is the default value. When you set box-sizing: content-box;, the width and height of the element are calculated based on the content area only. This means that padding and border are added to the width and height values, resulting in the total size of the box being larger than the specified width and height.

border-box: When you set box-sizing: border-box;, the width and height of the element are calculated including padding and border. This means that padding and border are included in the specified width and height values, so the content area will be smaller to accommodate them. This can be useful for creating elements with fixed sizes that include padding and border, without having to calculate the total size manually.

Here's a visual representation of the difference between content-box and border-box:



```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" type="text/css" href="style.
    css">
  </head>
  <body>
    <h2>Content Box</h2>
    <div class="content-box">Content Box</div>
    <h2>Border Box</h2>
    <div class="border-box">Border Box</div>
  </body>
</html>
```

```
/* content-box */
div.content-box {
    width: 200px;
    height: 100px;
    padding: 20px;
    border: 2px solid black;
    box-sizing: content-box;
}

/* border-box */
div.border-box {
    width: 200px;
    height: 100px;
    padding: 20px;
    border: 2px solid black;
    box-sizing: border-box;
}
```

Content Box

Content Box

Border Box

Border Box

How do margins collapse in the box model?

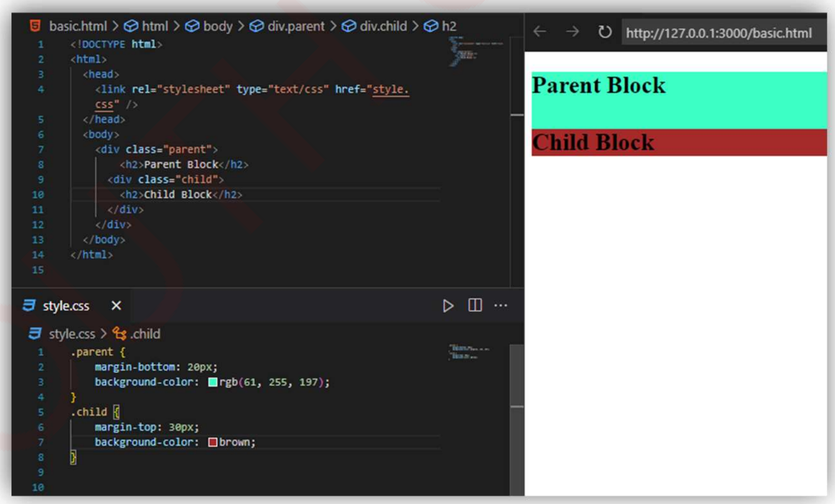
Margin collapsing in the CSS box model is a process where the vertical margins between two adjacent block-level elements are combined (collapsed) into a single margin. The final margin size is determined by taking the larger of the two margins and discarding the smaller one. This behavior can affect the spacing between elements, particularly in cases where margins are used for spacing or alignment.

Here's a brief explanation of margin collapsing with a simple example:

Theory:

- **Adjacent Margins:** When two block-level elements are adjacent to each other (with no content, padding, or border between them), their margins collapse.
- **Collapsed Margin Size:** The final margin size is determined by taking the larger of the two margins and discarding the smaller one.
- **Parent-Child Margins:** Margins between a parent element and its first/last child can also collapse under certain conditions.

Example:



basic.html > html > body > div.parent > div.child > h2

```html
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" type="text/css" href="style.css" />
  </head>
  <body>
    <div class="parent">
        <h2>Parent Block</h2>
      <div class="child">
        <h2>Child Block</h2>
      </div>
    </div>
  </body>
</html>
```

style.css > .child

```css
.parent {
    margin-bottom: 20px;
    background-color: rgb(61, 255, 197);
}
.child {
    margin-top: 30px;
    background-color: brown;
}
```

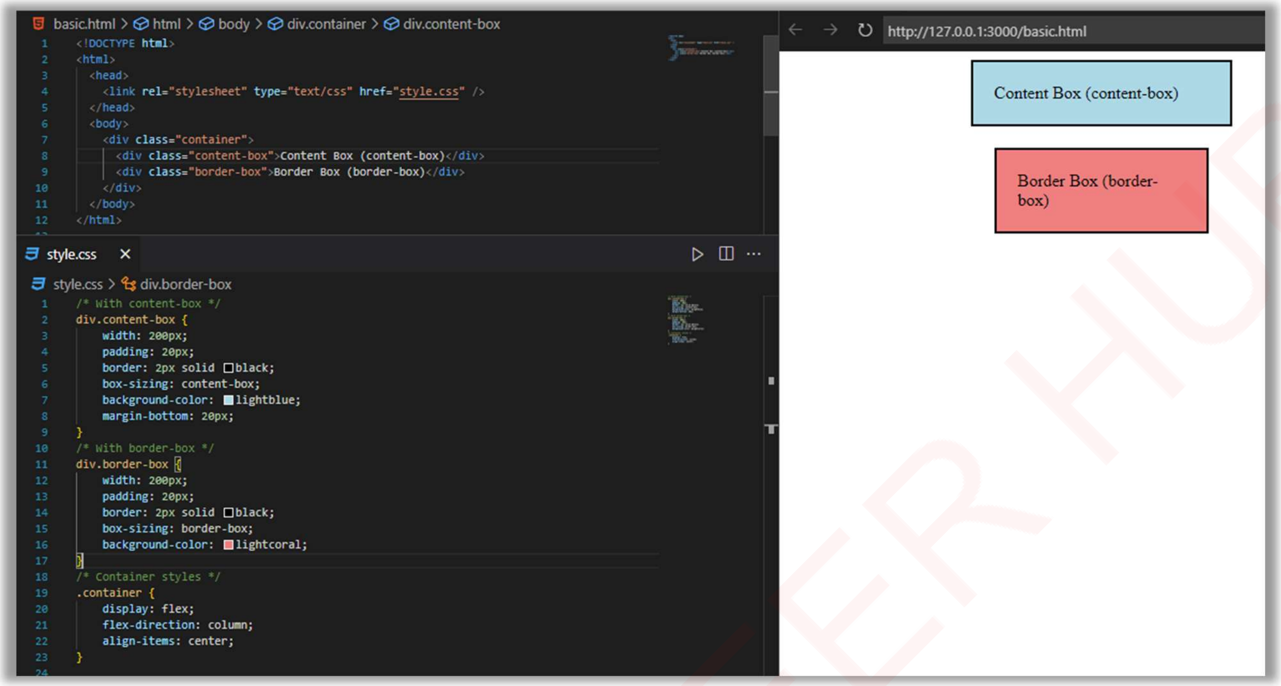
Explain Box-Sizing in CSS?

The box-sizing property in CSS is used to control how the total width and height of an element are calculated. It allows you to specify whether an element's width and height should include its padding and border, or if they should be calculated based on the content box alone.

There are two main values for the box-sizing property:

- **content-box:** This is the default value. When box-sizing: content-box; is applied to an element, its width and height are calculated based on the content box only. This means that any padding and border added to the element are added to the width and height specified in the CSS, making the total size of the element larger than the specified width and height.

- **border-box**: When box-sizing: border-box; is applied to an element, its width and height are calculated including padding and border. This means that the padding and border are included in the specified width and height values, so the content box will be smaller to accommodate them. This can be useful for creating elements with fixed sizes that include padding and border, without having to calculate the total size manually.



basic.html > html > body > div.container > div.content-box
style.css
style.css > div.border-box
http://127.0.0.1:3000/basic.html
Content Box (content-box)
Border Box (border-box)

```html
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" type="text/css" href="style.css" />
  </head>
  <body>
    <div class="container">
      <div class="content-box">Content Box (content-box)</div>
      <div class="border-box">Border Box (border-box)</div>
    </div>
  </body>
</html>
```

```css
/* With content-box */
div.content-box {
    width: 200px;
    padding: 20px;
    border: 2px solid black;
    box-sizing: content-box;
    background-color: lightblue;
    margin-bottom: 20px;
}
/* With border-box */
div.border-box {
    width: 200px;
    padding: 20px;
    border: 2px solid black;
    box-sizing: border-box;
    background-color: lightcoral;
}
/* Container styles */
.container {
    display: flex;
    flex-direction: column;
    align-items: center;
}
```

CSS LAYOUTS

What are the different types of CSS layouts?

There are several types of CSS layouts used to structure and position elements on a web page. Some of the most common types include:

1. **Static Layout:** Elements are positioned as they appear in the HTML document flow. This is the default behavior.
2. **Fluid or Liquid Layout:** Elements are sized in relative units (e.g., percentages) so they can adjust based on the size of the viewport. This allows the layout to be more flexible and responsive.
3. **Fixed Layout:** Elements are set to a fixed width, typically in pixels. The layout does not change based on the size of the viewport, which can lead to issues on different screen sizes.

4. **Adaptive Layout:** Similar to responsive design, adaptive layouts adjust based on the size of the viewport. However, instead of using fluid units, adaptive layouts use specific breakpoints to change the layout at predefined widths.

5. **Responsive Layout:** This approach uses a combination of fluid units and media queries to create a layout that adapts to different screen sizes and devices.

6. **Grid Layout:** CSS Grid Layout is a powerful layout system that allows you to create complex grid-based layouts with ease. It provides more control over the placement of elements compared to traditional methods.

7. **Flexbox Layout:** CSS Flexible Box Layout, or Flexbox, is a one-dimensional layout model that allows you to create flexible and efficient layouts. It's particularly useful for aligning items within a container along a single axis.

These are just a few examples of CSS layouts, and in practice, web developers often use a combination of these techniques to create effective and responsive designs.

CSS FLEXBOX

What is Flexbox?

Flexbox is short for "Flexible Box Layout". It's a CSS layout model that simplifies creating complex layouts. It provides a flexible way to align elements and distribute space within a container element.

The Flexbox layout model is bidirectional. This means you can either arrange your elements in rows, columns, or both. More on that later.

What are the benefits of using Flexbox?

- Before Flexbox, it was hard to create complex layouts and responsive web pages. You needed a combination of CSS floats and position properties. This required many workarounds and hacks.
- But with Flexbox, you can now do the following with less difficulty and fewer lines of code:
- Align and center elements using properties like justify-content and align-items.
- Develop responsive layouts without writing lots of media queries.
- Reorder elements without changing the HTML structure
- Create same-height columns without any extra HTML elements or background images.

Explain the difference between Flexbox and CSS Grid Layout.

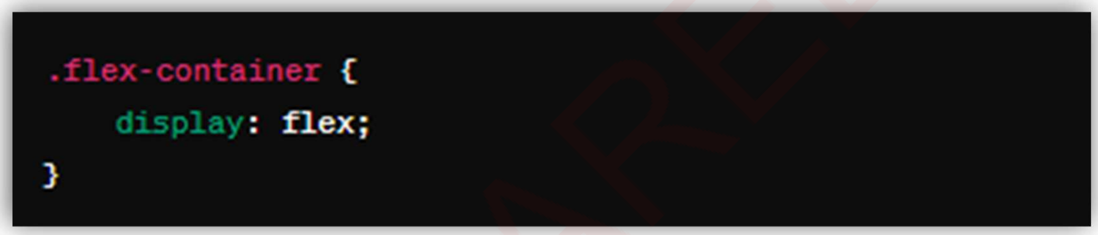
Feature	Flexbox	CSS Grid Layout
Layout Type	One-dimensional layout (either row or column)	Two-dimensional layout (rows and columns)
Main Axis and Cross Axis	Defines main axis and cross axis	Defines rows and columns
Child Element Placement	Controls how items are placed in a single dimension	Controls how items are placed in two dimensions
Alignment	Aligns items along the main and cross axes	Aligns items within rows and columns
Ordering	Can change order of items using the order property	Can reorder items in both rows and columns
Nested Layouts	Suitable for simple layouts and small-scale designs	Suitable for complex layouts and larger-scale designs
Browser Support	Widely supported across modern browsers	Widely supported across modern browsers
Use Cases	Best for laying out items in a single direction	Best for creating complex, grid-based layouts

How do you create a Flex container and define Flex items?

To create a Flex container and define Flex items, you need to follow these steps:

Create a Flex Container:

- Use the display property with a value of flex or inline-flex to create a Flex container.
- For example, to create a Flex container with a div element:

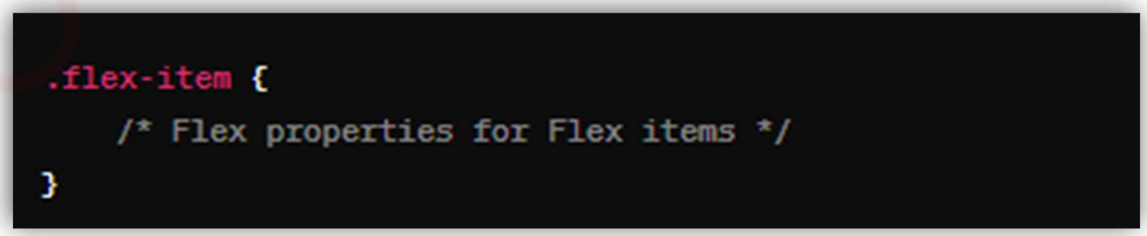


```
.flex-container {
    display: flex;
}
```

Define Flex Items:

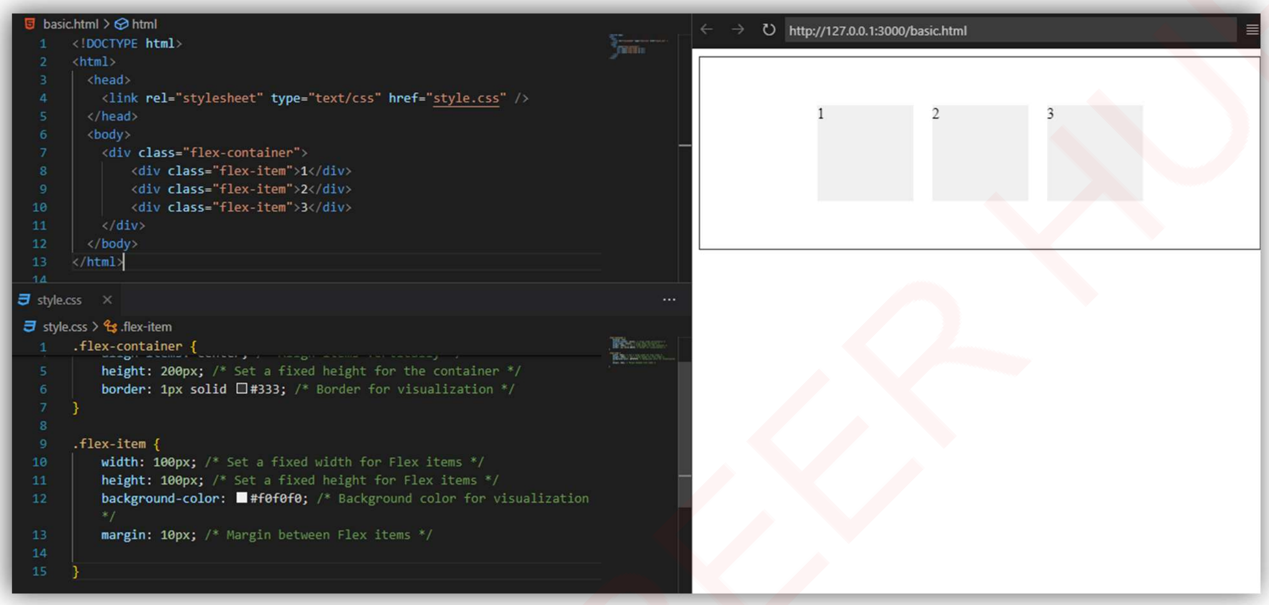
- Direct child elements of a Flex container are automatically considered Flex items.
- You can apply Flex properties to Flex items to control their behavior within the Flex container.

For example, to define Flex items within the Flex container:



```
.flex-item {
    /* Flex properties for Flex items */
}
```

Here's a complete example demonstrating how to create a Flex container and define Flex items:



```html
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" type="text/css" href="style.css" />
  </head>
  <body>
    <div class="flex-container">
        <div class="flex-item">1</div>
        <div class="flex-item">2</div>
        <div class="flex-item">3</div>
    </div>
  </body>
</html>
```

```css
.flex-container {
    height: 200px; /* Set a fixed height for the container */
    border: 1px solid #333; /* Border for visualization */
}

.flex-item {
    width: 100px; /* Set a fixed width for Flex items */
    height: 100px; /* Set a fixed height for Flex items */
    background-color: #f0f0f0; /* Background color for visualization
    */
    margin: 10px; /* Margin between Flex items */

}
```

What are the main properties for the Flex container?

The main properties for the Flex container in Flexbox are:

- **display:** Specifies the type of box for the element (e.g., flex or inline-flex).
    - **Syntax:** display: flex | inline-flex;
- **flex-direction:** Specifies the direction of the main axis.
    - **Syntax:** flex-direction: row | row-reverse | column | column-reverse;
- **flex-wrap:** Specifies whether flex items are forced into a single line or can be wrapped onto multiple lines.
    - Syntax: flex-wrap: nowrap | wrap | wrap-reverse;
- **flex-flow:** A shorthand for flex-direction and flex-wrap properties.
    - **Syntax:** flex-flow: <flex-direction> || <flex-wrap>;

- **justify-content:** Aligns flex items along the main axis.
    - **Syntax:** justify-content: flex-start | flex-end | center | space-between | space-around | space-evenly;
- **align-items:** Aligns flex items along the cross axis.
    - **Syntax:** align-items: flex-start | flex-end | center | baseline | stretch;
- **align-content:** Aligns flex lines along the cross axis when there is extra space in the container.
    - Syntax: align-content: flex-start | flex-end | center | space-between | space-around | stretch;

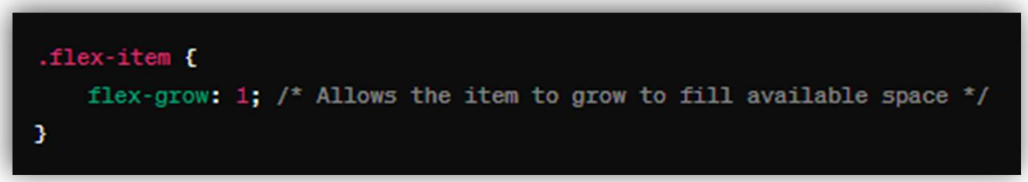
Explain the flex-grow, flex-shrink, and flex-basis properties.

The flex-grow, flex-shrink, and flex-basis properties are used to control how Flex items grow, shrink, and initially size themselves within a Flex container. These properties are part of the Flexible Box Layout module, also known as Flexbox.

flex-grow:

- The flex-grow property specifies the ability for a Flex item to grow if necessary. It defines the proportion of the available space that the Flex item should take up along the main axis.
- Default value: 0

Example:

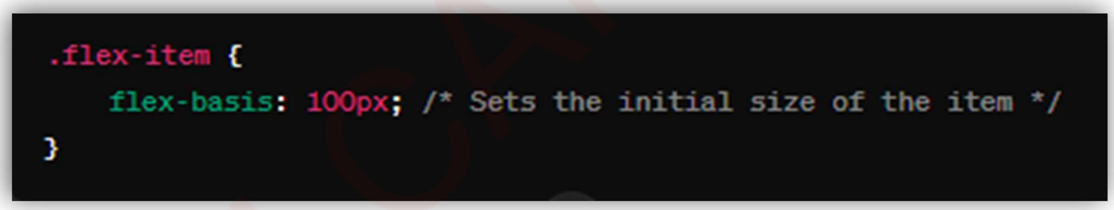


```
.flex-item {
    flex-grow: 1; /* Allows the item to grow to fill available space */
}
```

flex-shrink:

- The flex-shrink property specifies the ability for a Flex item to shrink if necessary. It defines the proportion by which the Flex item should shrink relative to the other Flex items when there is not enough space along the main axis.
- Default value: 1

Example:



```
.flex-item {
    flex-basis: 100px; /* Sets the initial size of the item */
}
```

In summary, flex-grow controls how much an item can grow relative to other items, flex-shrink controls how much it can shrink, and flex-basis controls its initial size before growth or shrinkage. These properties give you fine-grained control over the sizing behavior of Flex items within a Flex container.

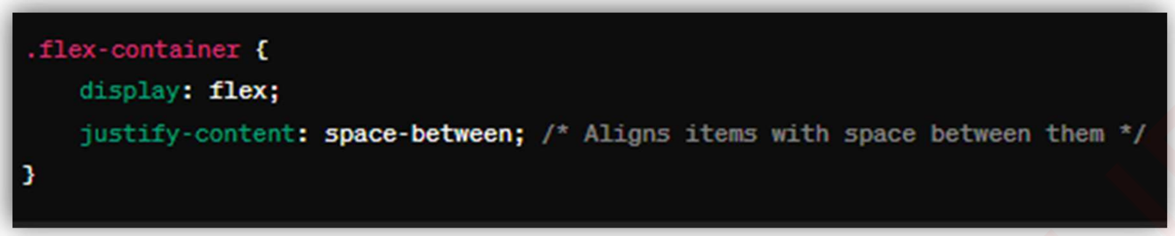
What is the purpose of the justify-content property?

The justify-content property in CSS Flexbox is used to align Flex items along the main axis of the Flex container. It helps control the distribution of space between and around Flex items when they do not take up all the available space along the main axis.

The justify-content property accepts several values, each of which determines how Flex items are aligned:

- **flex-start:** Aligns items to the start of the Flex container. This is the default value.
- **flex-end:** Aligns items to the end of the Flex container.
- **center:** Aligns items at the center of the Flex container.
- **space-between:** Distributes items evenly along the main axis. The first item is aligned to the start of the container, and the last item is aligned to the end of the container, with equal space between items.
- **space-around:** Distributes items evenly along the main axis with equal space around them. This means there is half the space before the first item and after the last item compared to the space between items.
- **space-evenly:** Distributes items evenly along the main axis with equal space around them, including before the first item and after the last item.

Here's an example of how justify-content can be used:



```
.flex-container {
    display: flex;
    justify-content: space-between; /* Aligns items with space between them */
}
```

How do you create a responsive layout using Flexbox?

To create a responsive layout using Flexbox, you can use media queries along with Flexbox properties to adjust the layout based on the screen size. Here's a general approach:

Set Up the Flex Container:

- Create a Flex container using display: flex;.
- Define the desired layout using Flexbox properties like flex-direction, justify-content, and align-items.

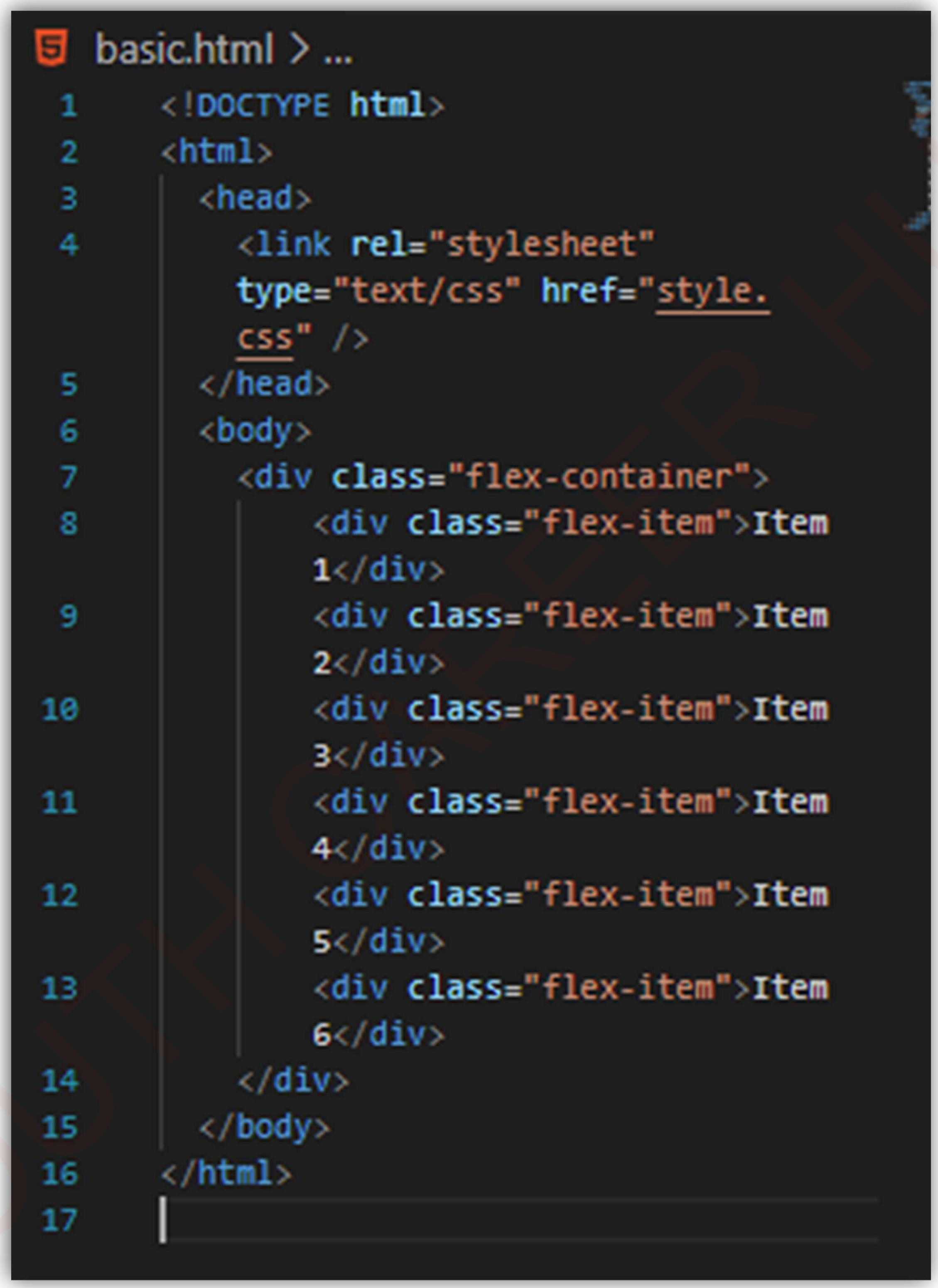
Use Flex Properties:

- Use Flex properties like flex-grow, flex-shrink, and flex-basis to control how Flex items grow, shrink, and size themselves within the container.

Add Media Queries:

- Use media queries to apply different styles based on the screen size.
- Adjust the Flexbox properties or layout as needed for different screen sizes.

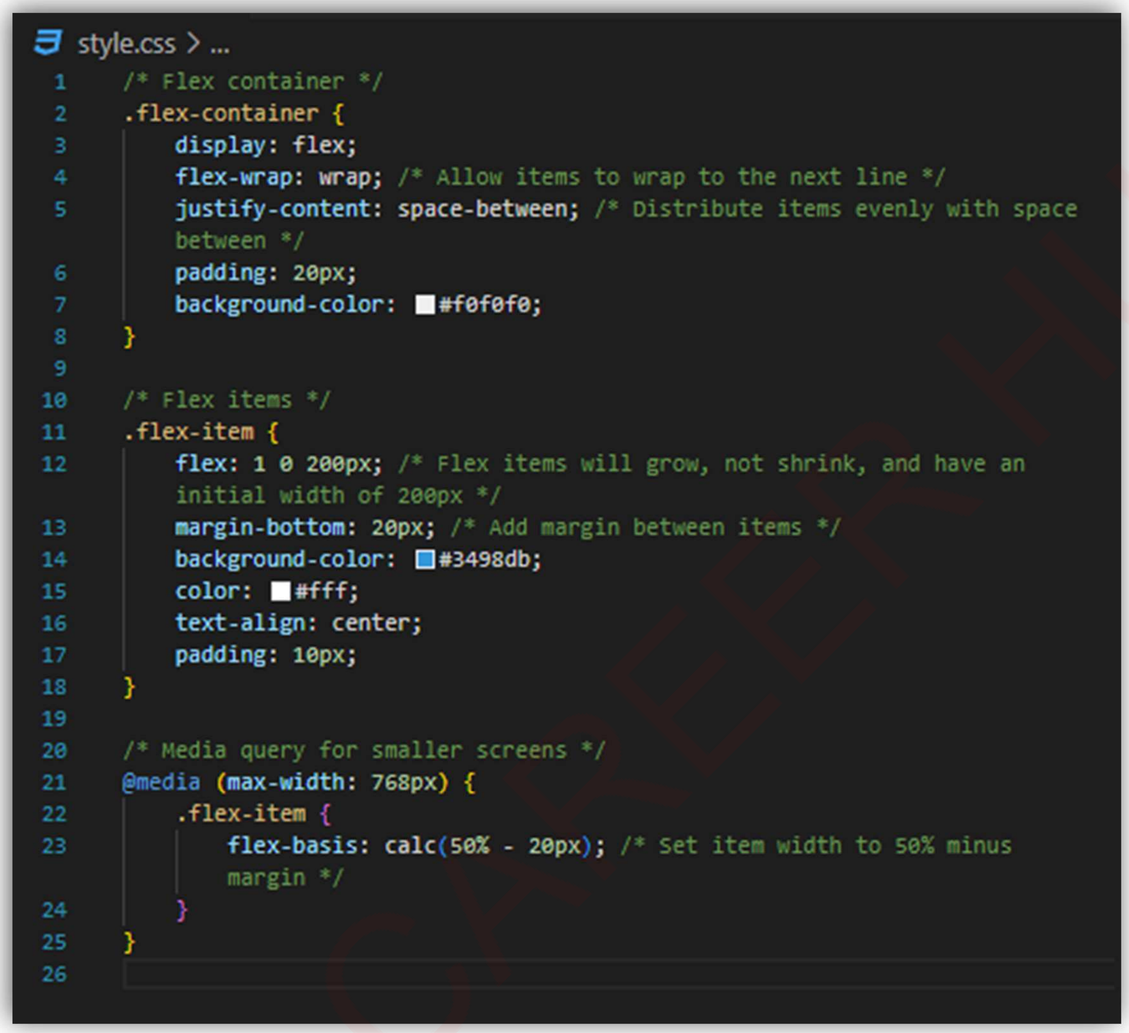
Here's a basic example of a responsive layout using Flexbox:



basic.html > ...

```html
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" type="text/css" href="style.css" />
  </head>
  <body>
    <div class="flex-container">
        <div class="flex-item">Item 1</div>
        <div class="flex-item">Item 2</div>
        <div class="flex-item">Item 3</div>
        <div class="flex-item">Item 4</div>
        <div class="flex-item">Item 5</div>
        <div class="flex-item">Item 6</div>
    </div>
  </body>
</html>

```



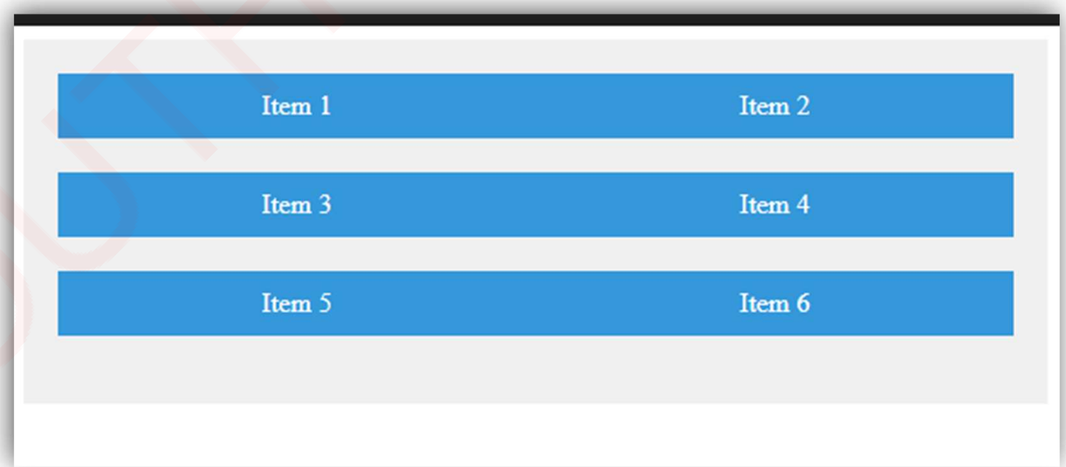
style.css > ...

```css
/* Flex container */
.flex-container {
    display: flex;
    flex-wrap: wrap; /* Allow items to wrap to the next line */
    justify-content: space-between; /* Distribute items evenly with space between */
    padding: 20px;
    background-color: #f0f0f0;
}

/* Flex items */
.flex-item {
    flex: 1 0 200px; /* Flex items will grow, not shrink, and have an initial width of 200px */
    margin-bottom: 20px; /* Add margin between items */
    background-color: #3498db;
    color: #fff;
    text-align: center;
    padding: 10px;
}

/* Media query for smaller screens */
@media (max-width: 768px) {
    .flex-item {
        flex-basis: calc(50% - 20px); /* Set item width to 50% minus margin */
    }
}

```



CSS GRID LAYOUT

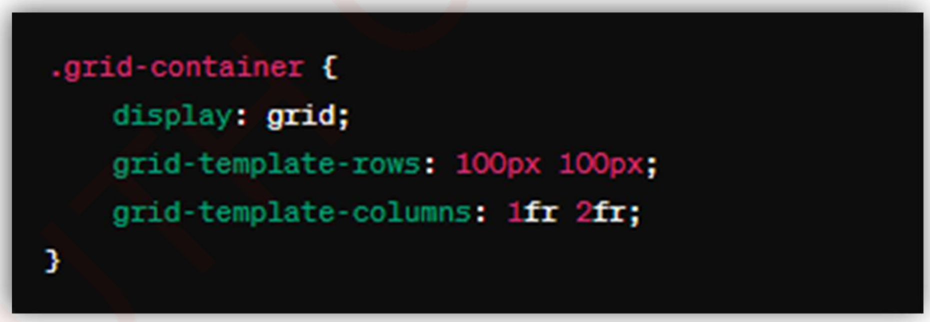
How do you create a Grid container and define Grid items?

To create a Grid container and define Grid items, you can follow these steps:

Create a Grid Container:

- Use the display property with a value of grid to create a Grid container.
- Optionally, use the grid-template-rows and grid-template-columns properties to define the size of the grid rows and columns.

For example:

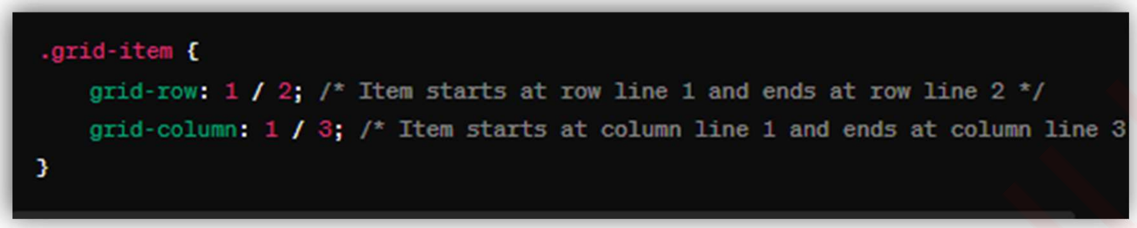


```
.grid-container {
    display: grid;
    grid-template-rows: 100px 100px;
    grid-template-columns: 1fr 2fr;
}
```

Define Grid Items:

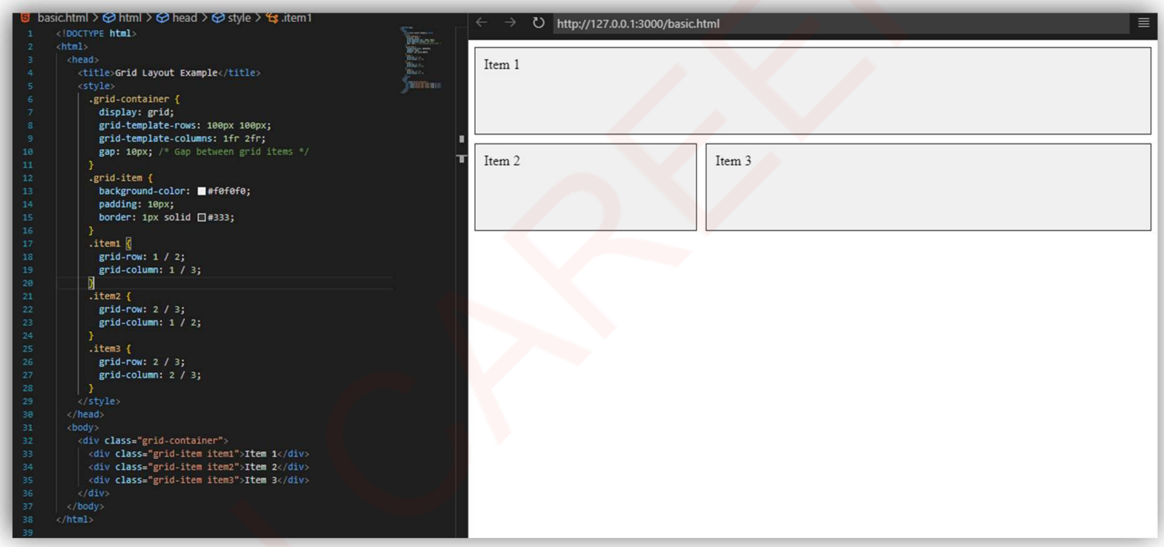
- Direct child elements of a Grid container are automatically considered Grid items.
- Use the grid-row and grid-column properties to specify where the Grid item should be placed in the grid.

For example:



```
.grid-item {
    grid-row: 1 / 2; /* Item starts at row line 1 and ends at row line 2 */
    grid-column: 1 / 3; /* Item starts at column line 1 and ends at column line 3
}
```

Here's a complete example demonstrating how to create a Grid container and define Grid items:


basic.html > html > head > style > .item1
http://127.0.0.1:3000/basic.html
Item 1
Item 2
Item 3

```
<!DOCTYPE html>
<html>
  <head>
    <title>Grid Layout Example</title>
    <style>
      .grid-container {
        display: grid;
        grid-template-rows: 100px 100px;
        grid-template-columns: 1fr 2fr;
        gap: 10px; /* Gap between grid items */
      }
      .grid-item {
        background-color: #f0f0f0;
        padding: 10px;
        border: 1px solid #333;
      }
      .item1 {
        grid-row: 1 / 2;
        grid-column: 1 / 3;
      }
      .item2 {
        grid-row: 2 / 3;
        grid-column: 1 / 2;
      }
      .item3 {
        grid-row: 2 / 3;
        grid-column: 2 / 3;
      }
    </style>
  </head>
  <body>
    <div class="grid-container">
      <div class="grid-item item1">Item 1</div>
      <div class="grid-item item2">Item 2</div>
      <div class="grid-item item3">Item 3</div>
    </div>
  </body>
</html>
```

What are the main properties for the Grid container?

The main properties for the Grid container in CSS Grid Layout are:

- **display:** Specifies the type of box for the element (e.g., grid or inline-grid).
  - Syntax: display: grid | inline-grid;
- **grid-template-rows**: Specifies the size of the grid rows.
  - Syntax: grid-template-rows: value1 value2 ...;
- **grid-template-columns:** Specifies the size of the grid columns.
  - Syntax: grid-template-columns: value1 value2 ...;
- **grid-template-areas:** Specifies named grid areas.
  - Syntax: grid-template-areas: "area1 area2" "area3 area4";
- **grid-template:** A shorthand for grid-template-rows, grid-template-columns, and grid-template-areas.
  - Syntax: grid-template: rows / columns;
- **grid-gap or gap:** Specifies the size of the gap between grid rows and columns.
  - Syntax: grid-gap: value; or gap: value;
- **justify-items:** Aligns grid items along the inline (row) axis.
  - Syntax: justify-items: start | end | center | stretch;
- **align-items:** Aligns grid items along the block (column) axis.
  - Syntax: align-items: start | end | center | stretch;
- **justify-content:** Aligns the grid along the inline (row) axis
  - Syntax: justify-content: start | end | center | stretch | space-around | space-between | space-evenly;
- **align-content:** Aligns the grid along the block (column) axis.
  - Syntax: align-content: start | end | center | stretch | space-around | space-between | space-evenly;
- **grid-auto-rows:** Specifies the size of any rows not explicitly sized.

    - Syntax: grid-auto-rows: value;
  - **grid-auto-columns:** Specifies the size of any columns not explicitly sized.
    - Syntax: grid-auto-columns: value;
  - **grid-auto-flow:** Specifies how auto-placed items are inserted in the grid.
    - Syntax: grid-auto-flow: row | column | row dense | column dense;

How do you create rows and columns in a Grid layout?

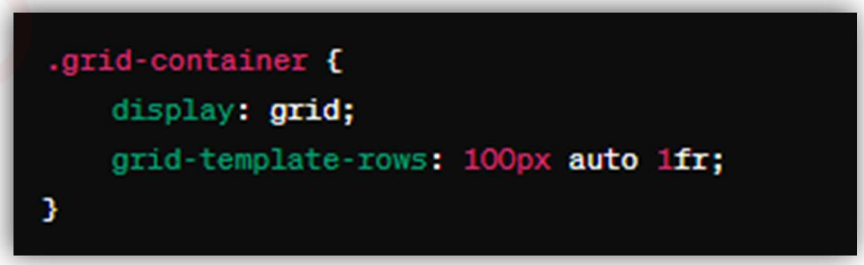
In CSS Grid Layout, you can create rows and columns using the grid-template-rows and grid-template-columns properties. These properties allow you to define the size and structure of the grid rows and columns respectively.

Here's how you can create rows and columns in a Grid layout:

Define Rows:

- Use the grid-template-rows property to define the size of each row.
- You can specify the size of each row individually or use keywords like auto, 1fr (fractional unit), or fixed sizes like 100px.

For example, to create three rows with different sizes:

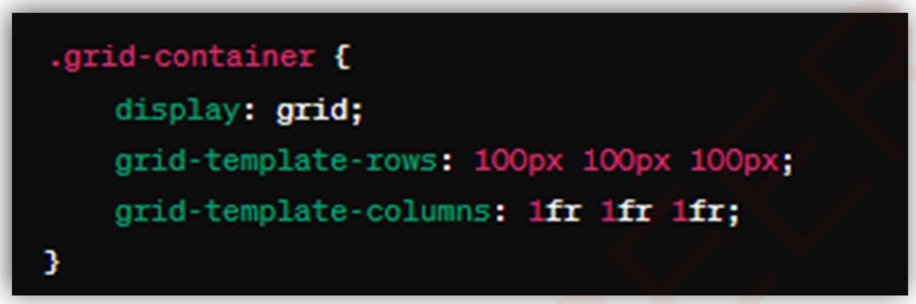


```
.grid-container {
    display: grid;
    grid-template-rows: 100px auto 1fr;
}
```

Define Columns:

- Use the grid-template-columns property to define the size of each column.
- You can specify the size of each column individually or use keywords and units like auto, 1fr, or fixed sizes like 200px.

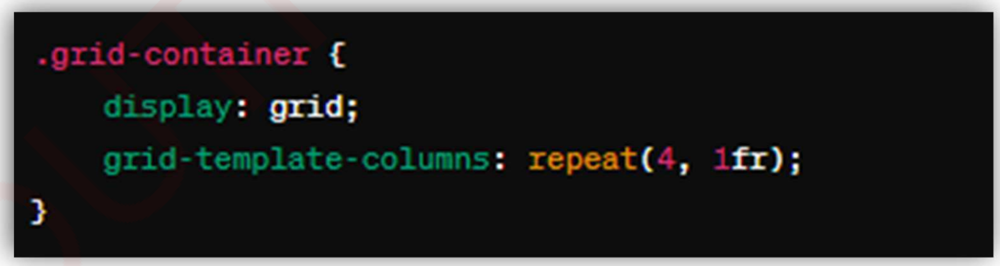
For example, to create two columns with different sizes:



```
.grid-container {
    display: grid;
    grid-template-rows: 100px 100px 100px;
    grid-template-columns: 1fr 1fr 1fr;
}
```

Using Repeat Function:

- You can use the repeat function to simplify the definition of multiple rows or columns with the same size.
- For example, to create a grid with 4 columns of equal width:



```
.grid-container {
    display: grid;
    grid-template-columns: repeat(4, 1fr);
}
```

What are the advantages of using CSS Grid for layout design?

CSS Grid Layout offers several advantages for layout design compared to traditional methods like floats and positioning:

- **Easy to create complex layouts:** CSS Grid allows you to create complex layouts with rows and columns easily, making it simpler to achieve both basic and intricate designs.

- **Responsive by default:** CSS Grid makes it straightforward to create responsive designs without relying heavily on media queries. You can use features like flexible units (fr), auto sizing, and grid areas to create layouts that adapt to different screen sizes.

- **Two-dimensional layout control:** Unlike Flexbox, which is mainly for one-dimensional layouts, CSS Grid provides a two-dimensional layout system. This means you can control both rows and columns, allowing for more versatile and precise layouts.

- **Grid gap:** CSS Grid includes a built-in way to create space between grid items using the grid-gap property. This eliminates the need for using margins or padding to create spacing between elements.

- **Grid alignment:** CSS Grid provides powerful alignment capabilities, allowing you to align grid items along both the row and column axes, as well as the ability to align entire grids within their containers.

- **Named grid lines and areas:** CSS Grid allows you to name your grid lines and areas, which can make your layout code more readable and easier to maintain, especially for larger layouts.

- **Support in modern browsers:** CSS Grid is supported in all major modern browsers, making it a viable option for creating layouts without worrying about browser compatibility issues.

Overall, CSS Grid offers a more flexible, powerful, and intuitive way to create layouts compared to traditional methods, making it a valuable tool for modern web design.

CSS Transitions and Animation

CSS Transitions and animations provides powerfull tools for adding dynamic and visulaly appealing efgffects to web elements. Transition allows smooth changes when an element changes its state while animations provide more complex and controled motion effects.

Tranisition Properties:

Property:

The CSS property we want to transition, such as color, opacity, width, etc.

Duration: The time it takes for the tranistion to complete. Specified in seconds or miliseconds.

Timing function:

Describes how the transition progresses over time. It defines the acceleration and decelaration of tranition. Common values include 'ease', 'linear', 'ease-in', 'ease-out' and 'ease-in-out'.

Creating simple CSS tranistions:

CSS transition allow us to smoothly change property values over time, craeteing visual effects like fadingm sliding or color changes. We define the starting and ending property values and specify the duration and timing function.

For example

```
.box{
Width: 100px;
Height: 100px;
Background-color: blue;
Transition: width 0.5s ease-in-out;
}
.box:hover{
Width:150px;
}
```

Keyframe animations:

Keyframe animations after more advanced and customixable effects by allowing us to define multiple intermediate stages of an animation.

Keyframes are defines with the @keyframe, where we specify various percentages of the animations progress and the properties at each stage.

For example:

```
@Keyframes slide{
0% {left: 0;}
50%{left: 50%;}
100%{left: 100%}
}
```

Explain animation properties in css

CSS provides several properties for creating animations. These properties allow you to animate the values of CSS properties over time, creating effects like movement, fading, and transformation. Here are some of the key animation properties in CSS:

- animation-name: Specifies the name of the keyframe animation you want to apply.

    - Syntax: animation-name: keyframename;
- animation-duration: Specifies how long the animation should take to complete one cycle.
    - Syntax: animation-duration: time;
- animation-timing-function: Specifies the speed curve of the animation.
    - Syntax: animation-timing-function: ease | linear | ease-in | ease-out | ease-in-out | cubic-bezier(n,n,n,n);
- animation-delay: Specifies a delay before the animation starts.
    - Syntax: animation-delay: time;
- animation-iteration-count: Specifies the number of times an animation should run.
    - Syntax: animation-iteration-count: number | infinite;
- animation-direction: Specifies whether the animation should play in reverse on alternate cycles.
    - Syntax: animation-direction: normal | reverse | alternate | alternate-reverse;
- animation-fill-mode: Specifies how the animation should apply styles before and after it is executed.
    - Syntax: animation-fill-mode: none | forwards | backwards | both;
- animation-play-state: Specifies whether the animation is running or paused.
    - Syntax: animation-play-state: running | paused;

CSS MEDIA QUERIES

What is a media query in CSS?

A media query in CSS is a technique used to apply styles based on the characteristics of the device or browser viewing the page. Media queries allow you to create responsive designs that adapt to different screen sizes, resolutions, and other device features.

Media queries consist of a media type (such as screen, print, speech, etc.) and one or more expressions that check for certain conditions. These conditions can include things like the width and height of the viewport, the device orientation (landscape or portrait), the resolution of the device, and more.

Here's a basic example of a media query that applies styles only when the viewport width is 600 pixels or less:

```
@media (max-width: 600px) {
/* Styles to apply when viewport width is 600px or less */
body {
font-size: 14px;
}
}
```

In this example, the @media rule checks if the maximum width of the viewport is 600 pixels (max-width: 600px). If this condition is true, the styles inside the media query block will be applied. Otherwise, they will be ignored.

Media queries are commonly used in responsive web design to create layouts that adjust to different screen sizes, ensuring that the content looks good and is easy to read on various devices.

How do you target specific devices, such as smartphones or tablets, using media queries?

Smartphones:

Target smartphones by using the max-width media query and specifying a maximum width that corresponds to smartphone viewport sizes. Common smartphone viewport widths range from around 320px to 480px.

Example:

```
@media (max-width: 480px) {

/* Styles for smartphones */

}
```

Tablets:

Target tablets by using the min-width and max-width media queries to specify a range of widths that correspond to tablet viewport sizes. Common tablet viewport widths range from around 768px to 1024px.

Example:

```
@media (min-width: 481px) and (max-width: 1024px) {

/* Styles for tablets */

}
```

Specific devices:

You can also target specific devices by combining media query expressions with specific device characteristics. For example, to target iPhones, you can use the device-width and device-height media features.

Example:

```
@media (device-width: 375px) and (device-height: 812px) and (-webkit-device-pixel-ratio: 3) {
```

```
/* Styles for iPhone X */

}
```

Orientation:

You can also target specific device orientations, such as landscape or portrait, using the orientation media feature.

Example:

```
@media (orientation: landscape) {

/* Styles for landscape orientation */

}
```

By using these media queries, you can apply different styles based on the characteristics of the device, allowing you to create responsive designs that adapt to different screen sizes and orientations.