**Day 25**





**“Web Development + Security”**

**More on CSS Selectors:**

**CSS Cheat Sheet:**

| **Selector Type** | **Syntax** | **Example** | **Meaning / What It Selects** |
| --- | --- | --- | --- |
| **Universal Selector** | \* | \* { margin: 0; } | Selects **all elements** |
| **Element Selector** | tag | p { color: blue; } | Selects **all <p>** elements |
| **Class Selector** | .classname | .intro { color: green; } | Selects all elements with class="intro" |
| **ID Selector** | #idname | #main { background: yellow; } | Selects element with id="main" |
| **Group Selector** | A, B | h1, h2 { color: red; } | Applies same style to **multiple elements** |
| **Descendant Selector** | A B | div p { color: blue; } | Selects <p> **inside** <div> |
| **Child Selector** | A > B | div > p { color: orange; } | Selects **direct child** <p> of <div> |
| **Adjacent Sibling** | A + B | h1 + p { color: purple; } | Selects <p> **immediately after** <h1> |
| **General Sibling** | A ~ B | h1 ~ p { color: gray; } | Selects **all <p> after <h1>** (siblings) |
| **Attribute Selector** | [attr] | [title] { color: pink; } | Elements **having** title attribute |
|  | [attr=value] | [type="text"] | Elements with attribute **equal to** given value |
|  | [attr^=value] | [class^="btn"] | Attribute **starts with** value |
|  | [attr$=value] | [src$=".jpg"] | Attribute **ends with** value |
|  | [attr\*=value] | [class\*="nav"] | Attribute **contains** value |
| **Pseudo-class** | :pseudo-class | a:hover { color: red; } | Defines a **state** of an element |
| **Common Pseudo-classes** | :hover, :focus, :active, :nth-child(), :first-child, :last-child | — | Used for interactivity and element positions |
| **Pseudo-element** | ::before, ::after | p::before { content:"👉"; } | Adds **content before or after** an element |
| **Negation Selector** | :not(selector) | p:not(.special) | Selects <p> **except** those with .special |

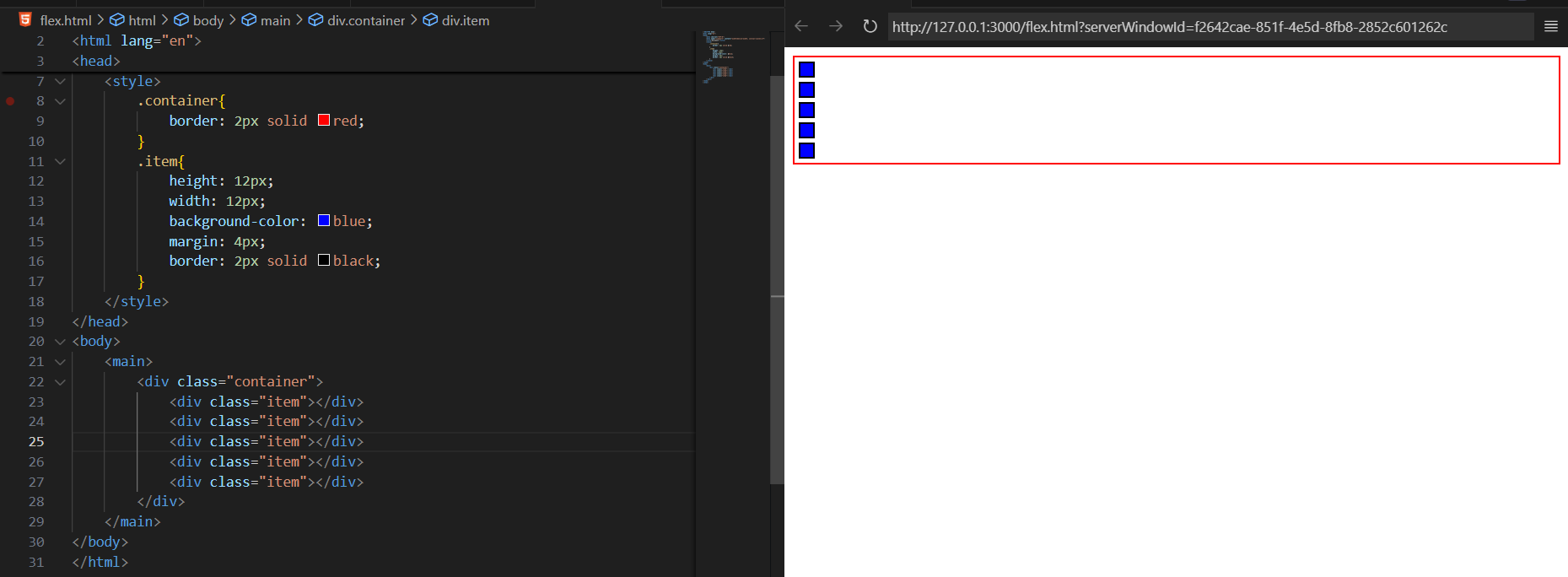
**CSS Flexbox:**

**What is Flex box?**

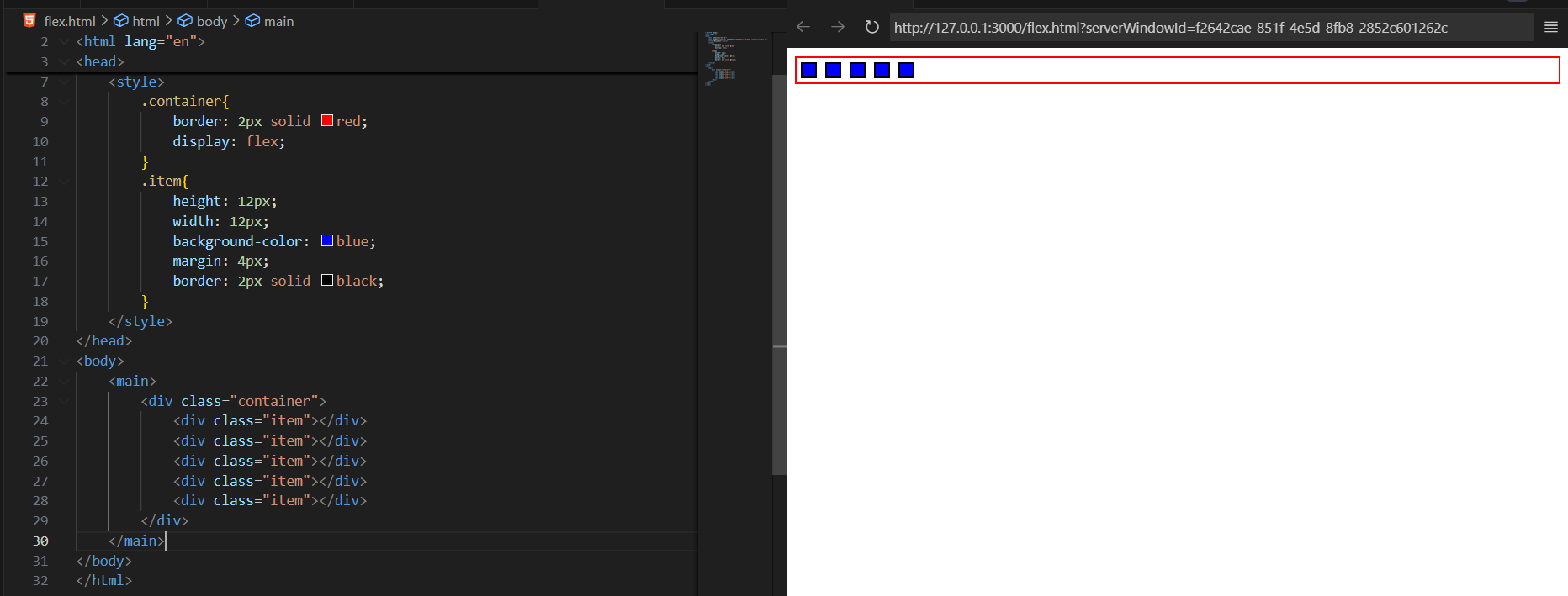
Flexbox (Flexible Box) is a CSS layout module that allows you to align and distribute space among items in a container, even when their size is unknown or dynamic. Think of it as a magic tool to control layouts easily, without floats or complicated positioning.

To understand this, we will first see how without it things work?

A basic example of <div>s: clearly, we can see that all the <div> get one above other, like in a column.

****

What if we add display:flex in the .container? We will see that all the <div> come in one row, as shown below:



**Main Flex Properties**

**A. Container Properties**

1. **flex-direction** → Sets the main axis (row or column)

.container {

display: flex;

flex-direction: row; /\* row (default), column, row-reverse, column-reverse \*/

}

1. **justify-content** → Align items along the **main axis**

.container {

justify-content: flex-start; /\* flex-start, flex-end, center, space-between, space-around, space-evenly \*/

}

1. **align-items** → Align items along the **cross axis** (perpendicular to main axis)

.container {

align-items: stretch; /\* flex-start, flex-end, center, baseline, stretch \*/

}

1. **flex-wrap** → Allow items to wrap to next line

.container {

flex-wrap: wrap; /\* nowrap (default), wrap, wrap-reverse \*/

}

1. **gap** → Space between items

.container {

gap: 10px; /\* sets spacing between flex items \*/

}

**B. Item Properties**

1. **flex-grow** → Make an item grow to fill available space

.item {

flex-grow: 1; /\* default 0 \*/

}

1. **flex-shrink** → Make an item shrink if necessary

.item {

flex-shrink: 1; /\* default 1 \*/

}

1. **flex-basis** → Initial size of item before growing/shrinking

.item {

flex-basis: 200px;

}

1. **align-self** → Override align-items for a single item

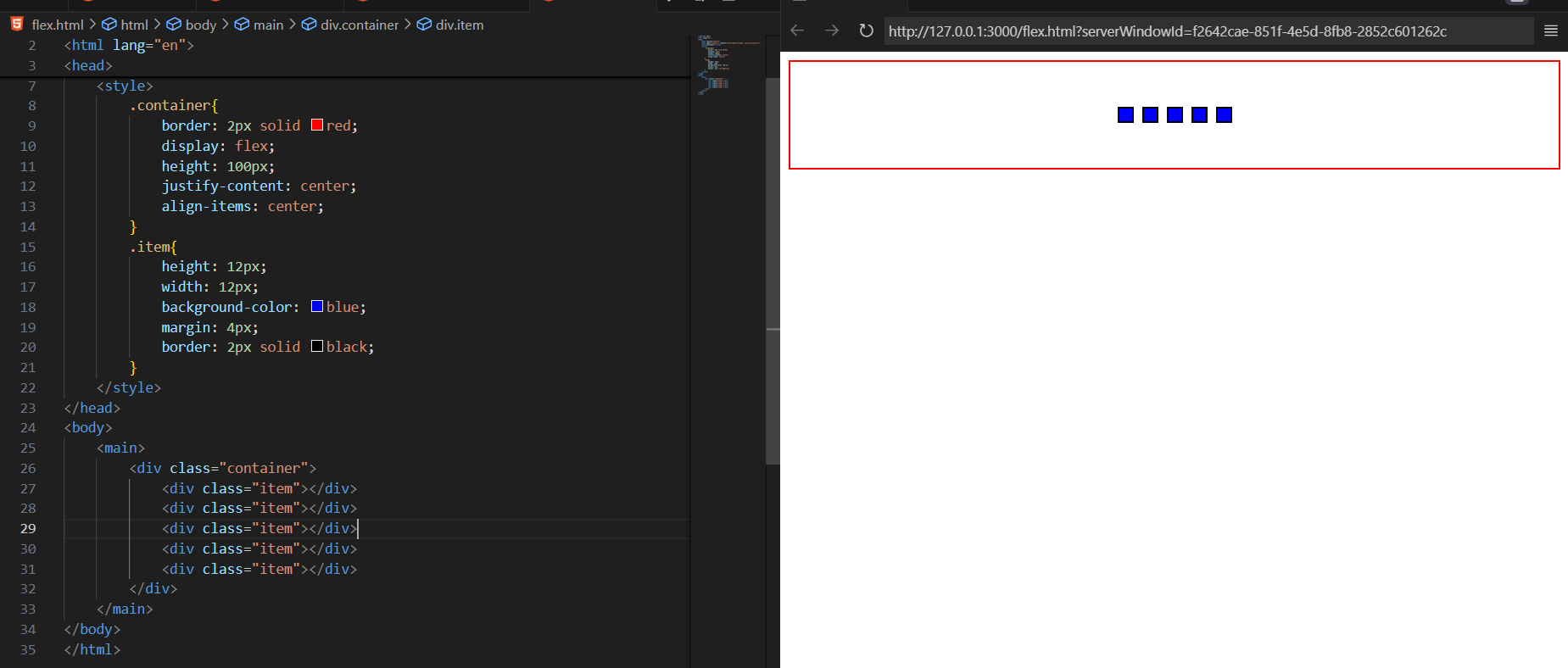
.item {

align-self: center; /\* auto, flex-start, flex-end, center, baseline, stretch \*/

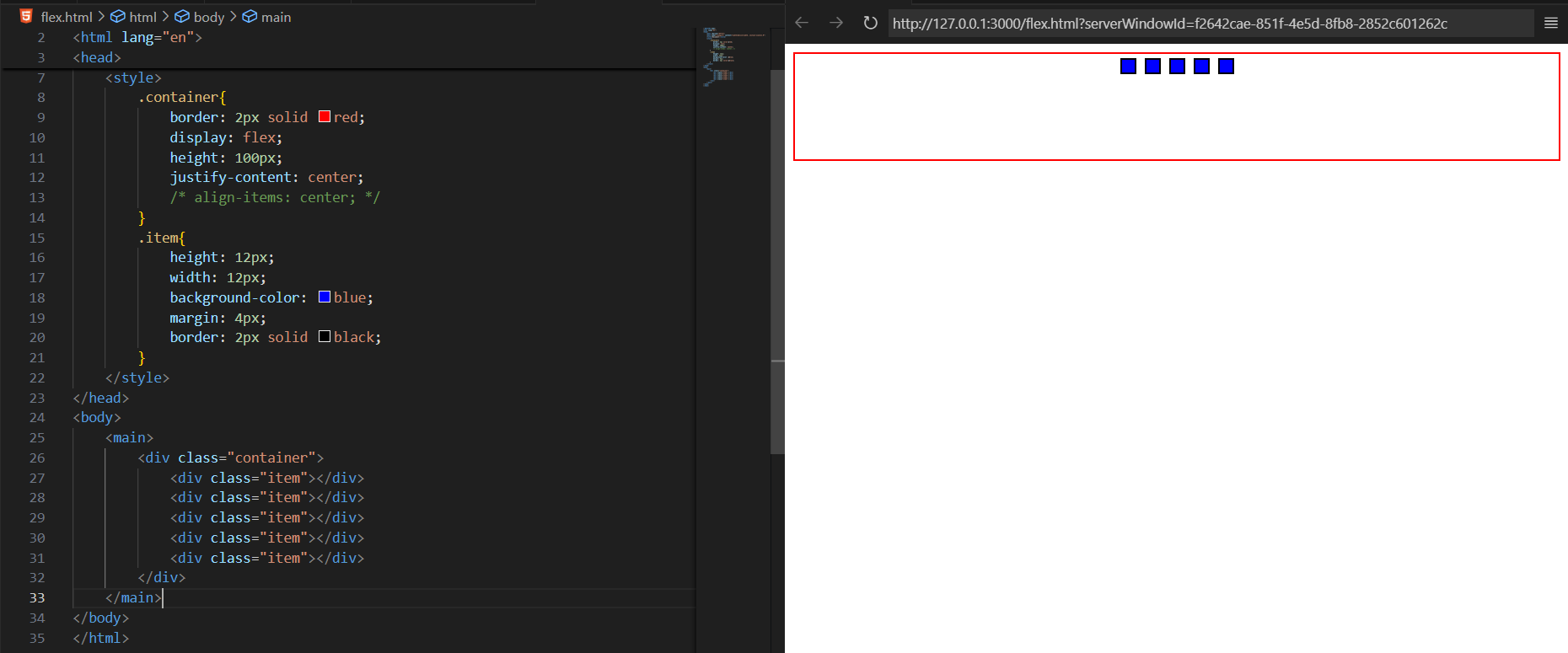
}

Now, what to do in order to make the elements in centre?

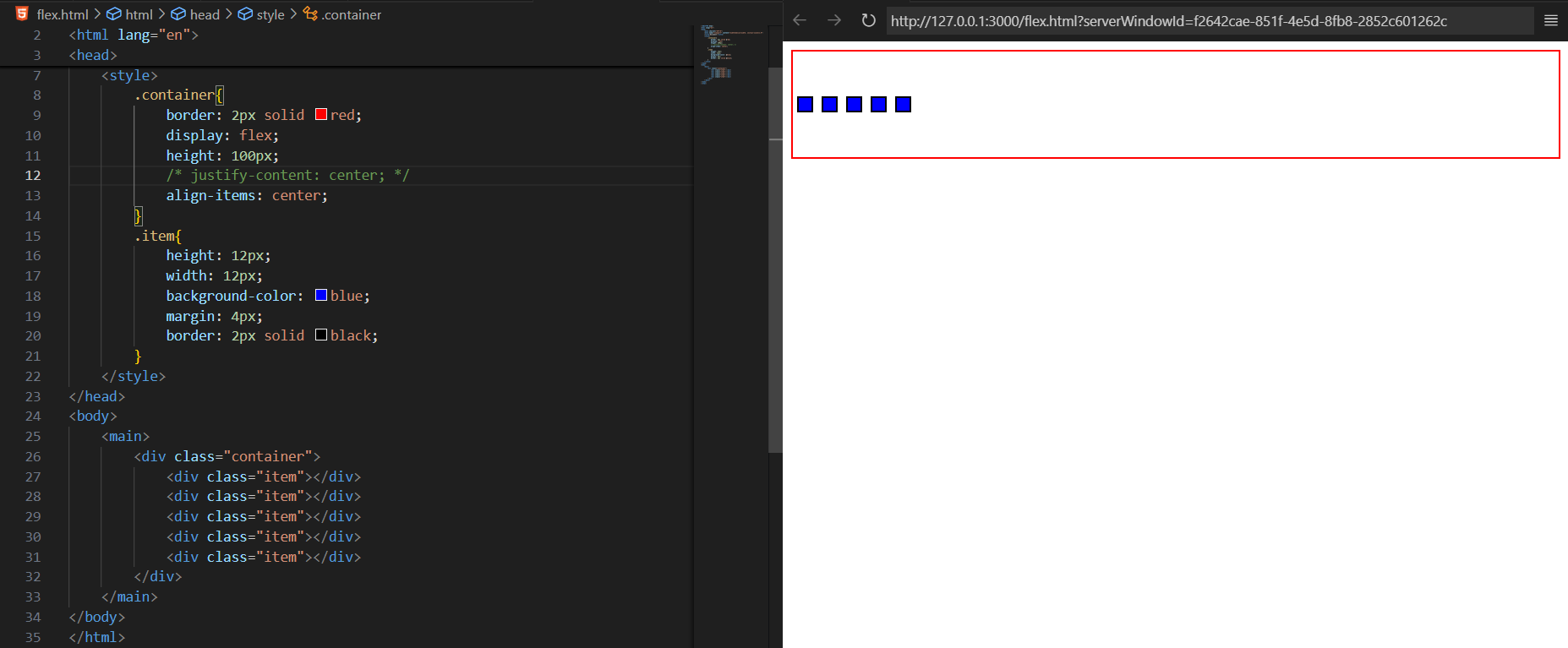
We will use the property, justify-content: center; and align-items: center; as shown below:



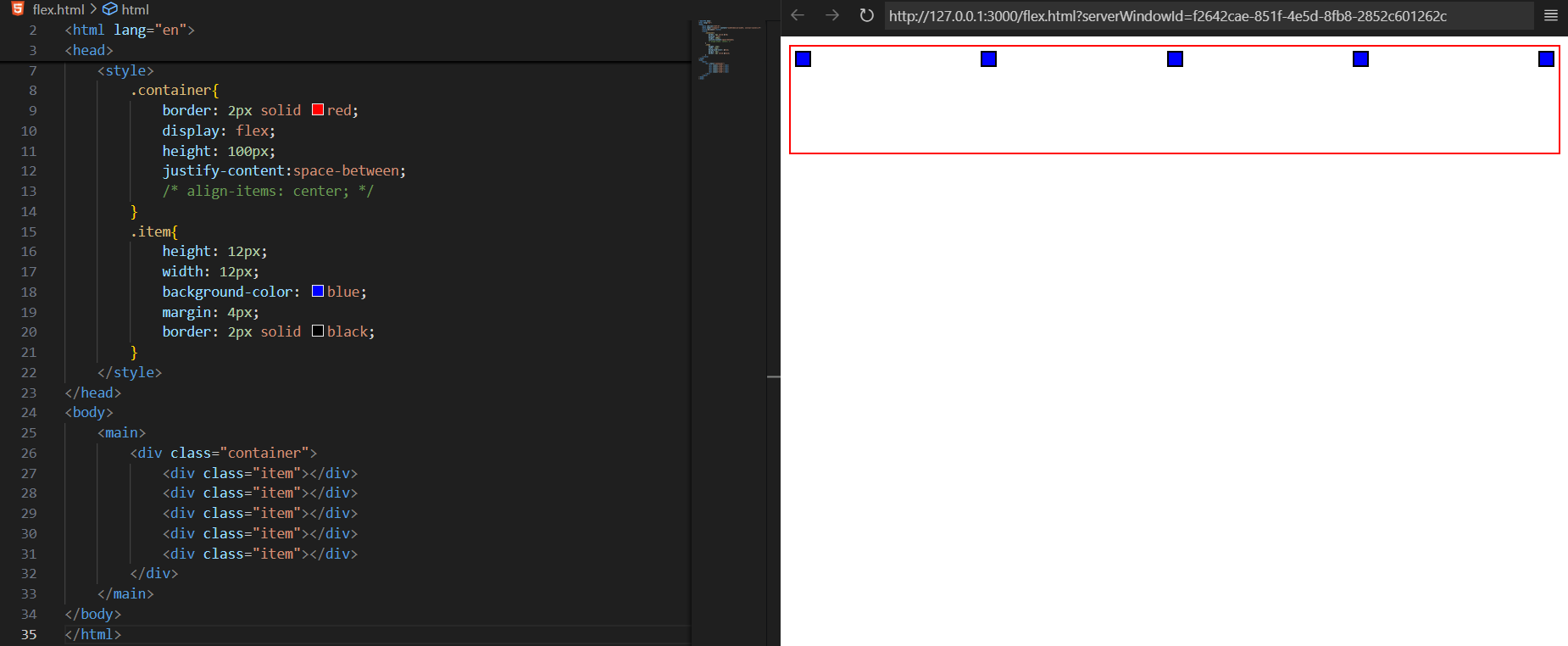
In case align-item center is not there: then horizontally only it will be in centre not vertically.



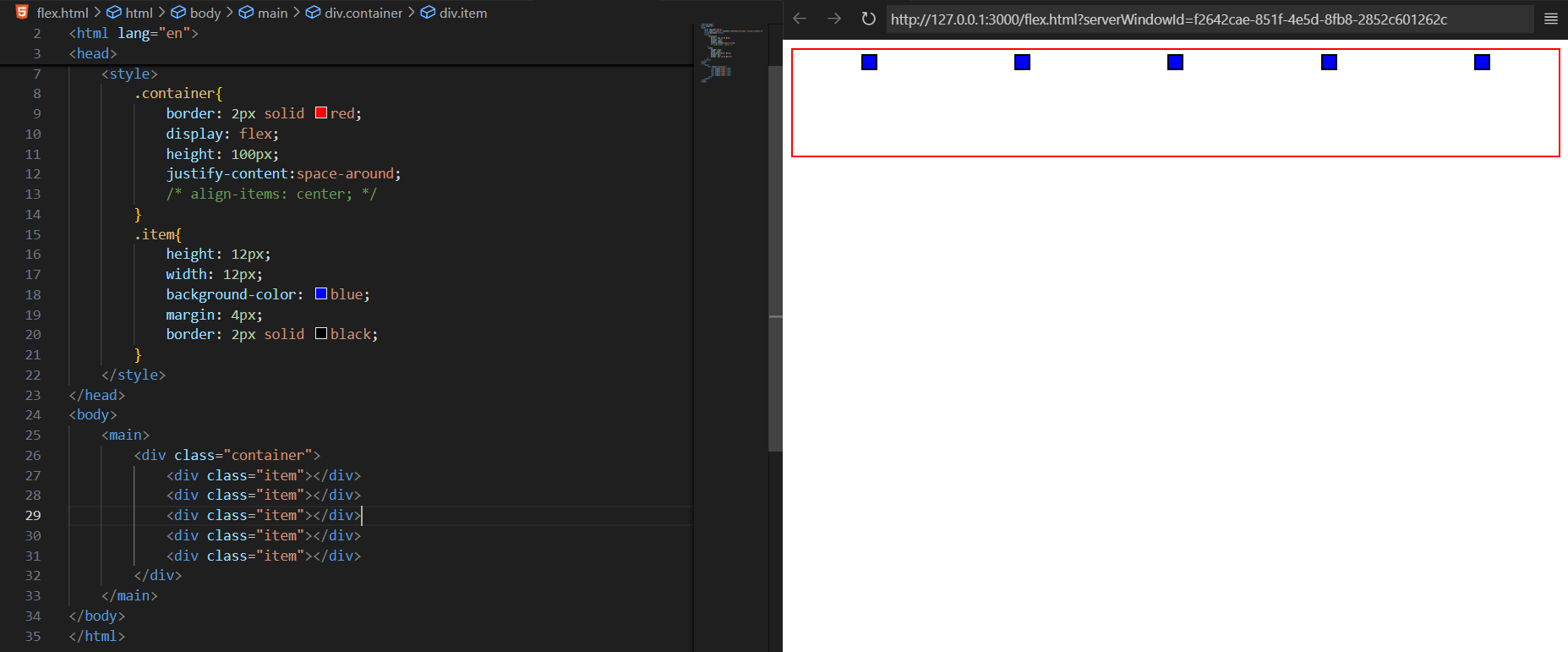
In case justify-content: center is not there: then vertically nly it will be in centre not horizontally.



We can also do justify-content: space-between: as shown below



We can also do justify-content: space-around: as shown below



**So, do you know what actually happens when we apply this flex display property?**

Basically, It Make a container a flex container using:

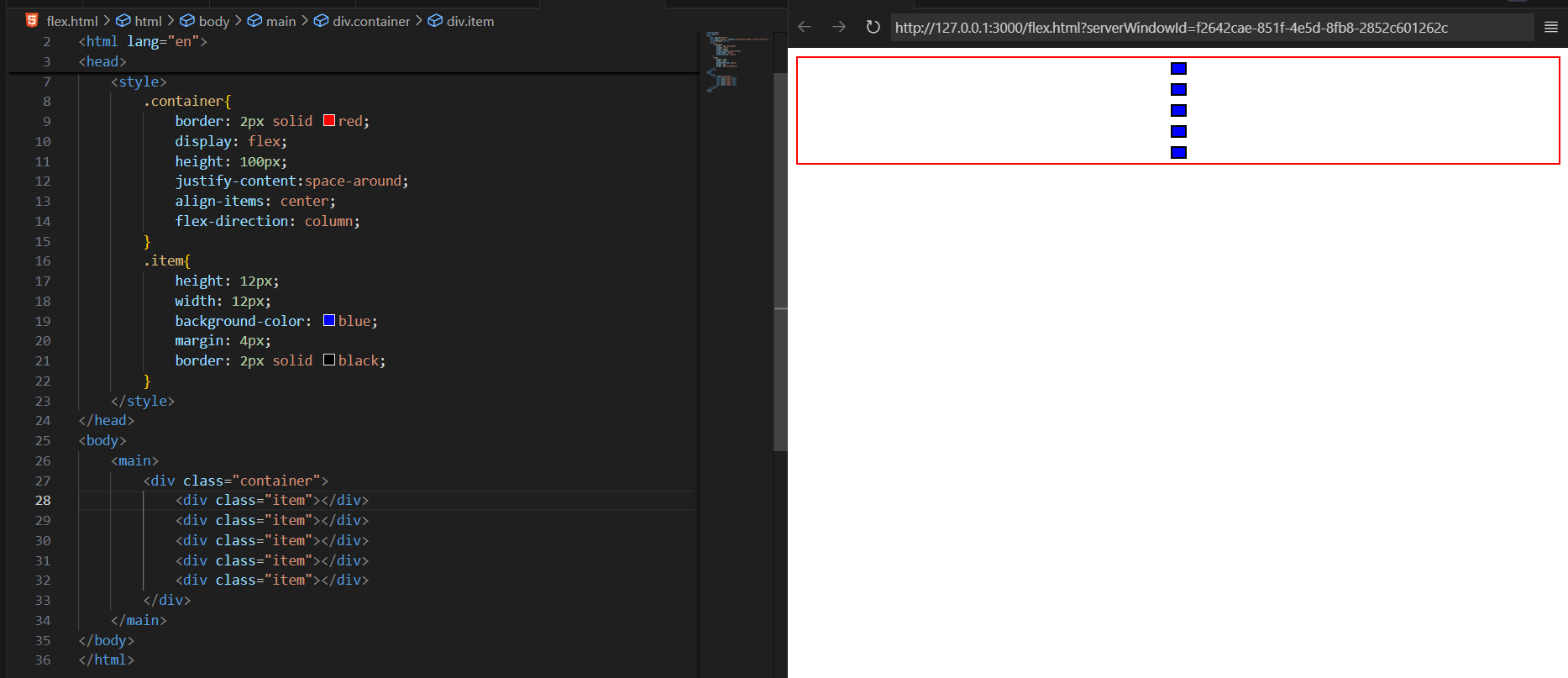
.container {

display: flex;

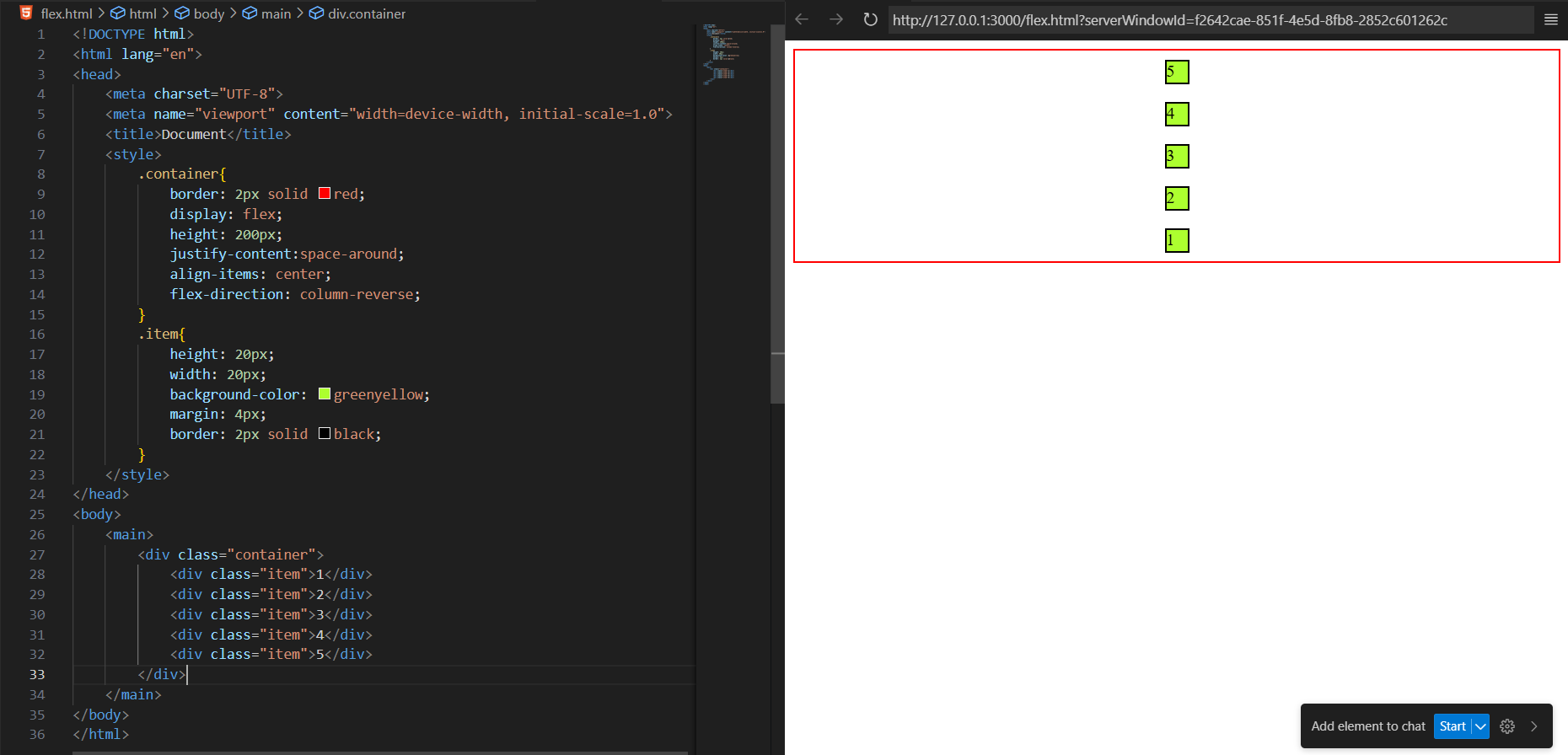
}

All direct children of .container automatically become flex items.

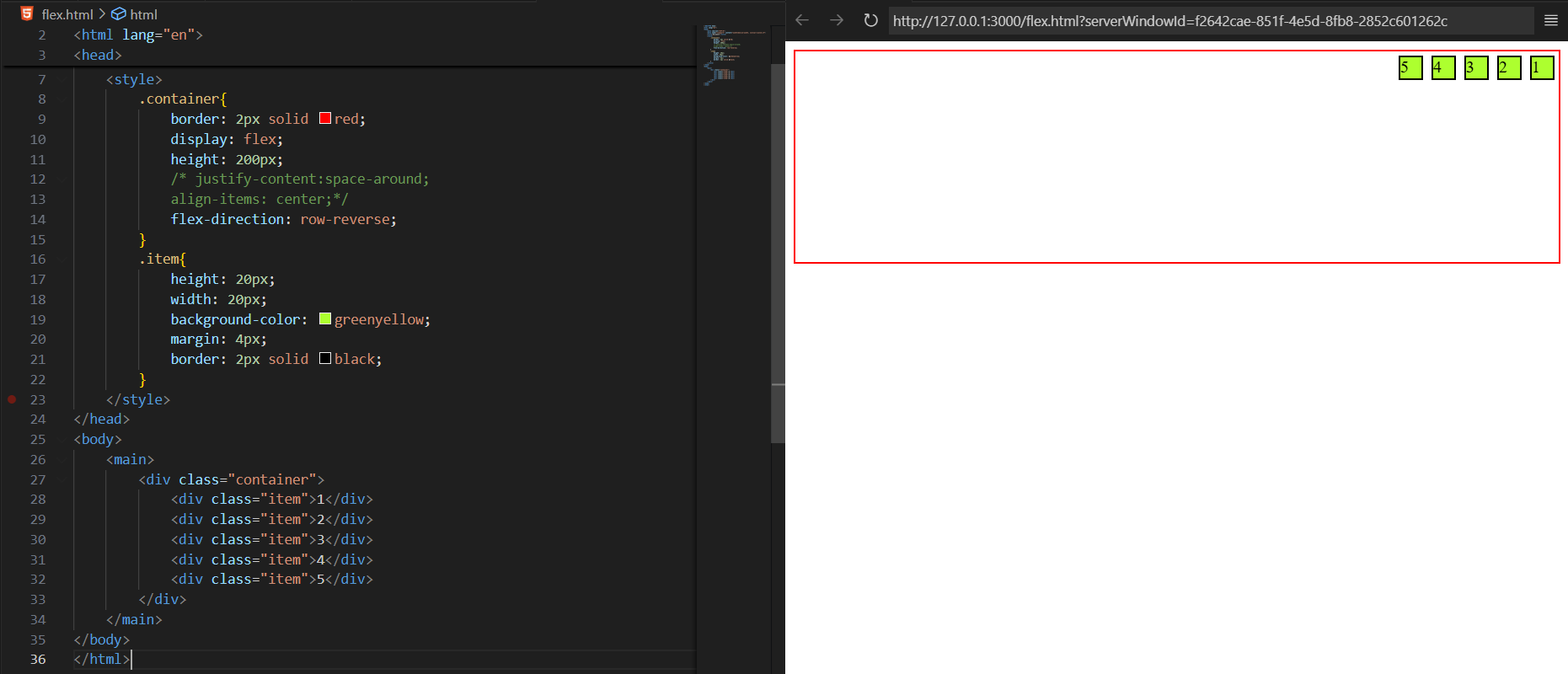
Now, let’s understand the flex-direction:column, basically it will change its direction to column:



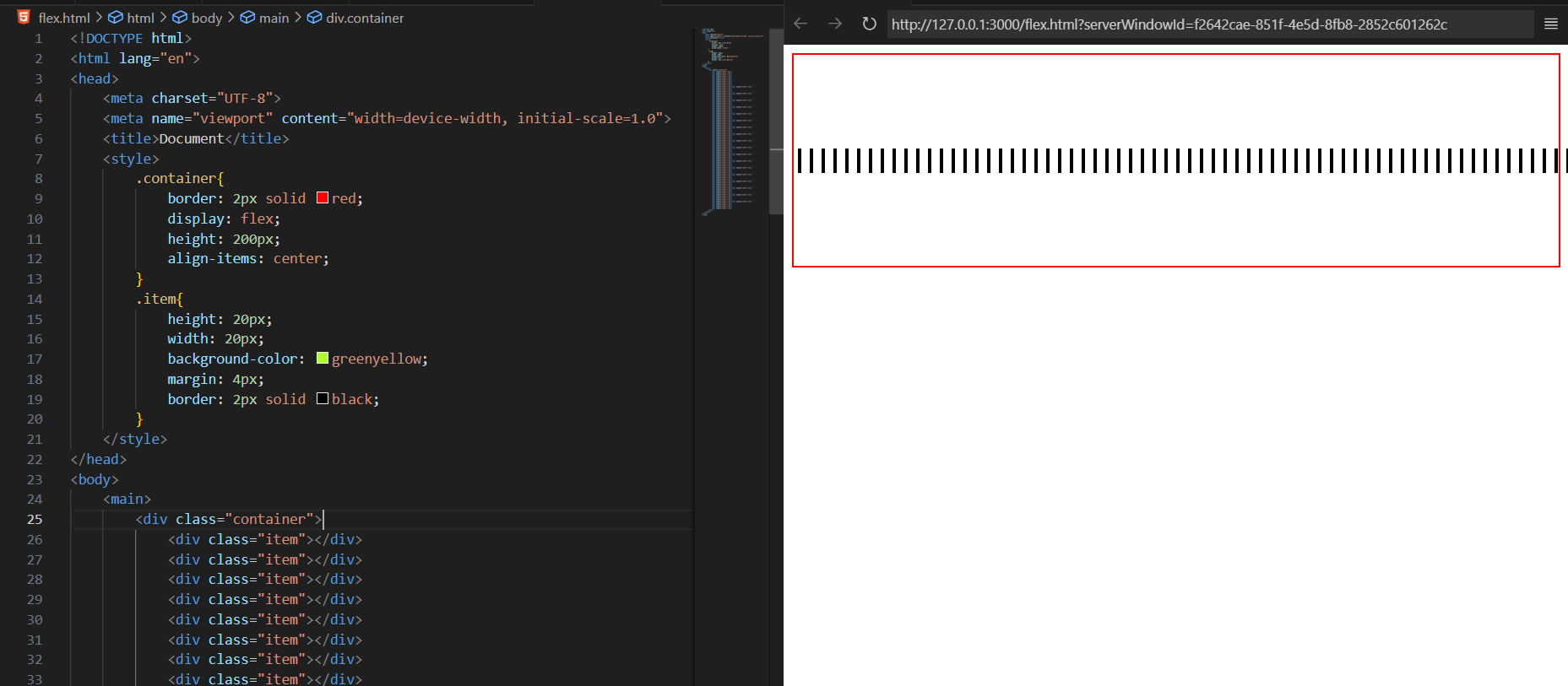
Example: use of flex-direction:column-reverse

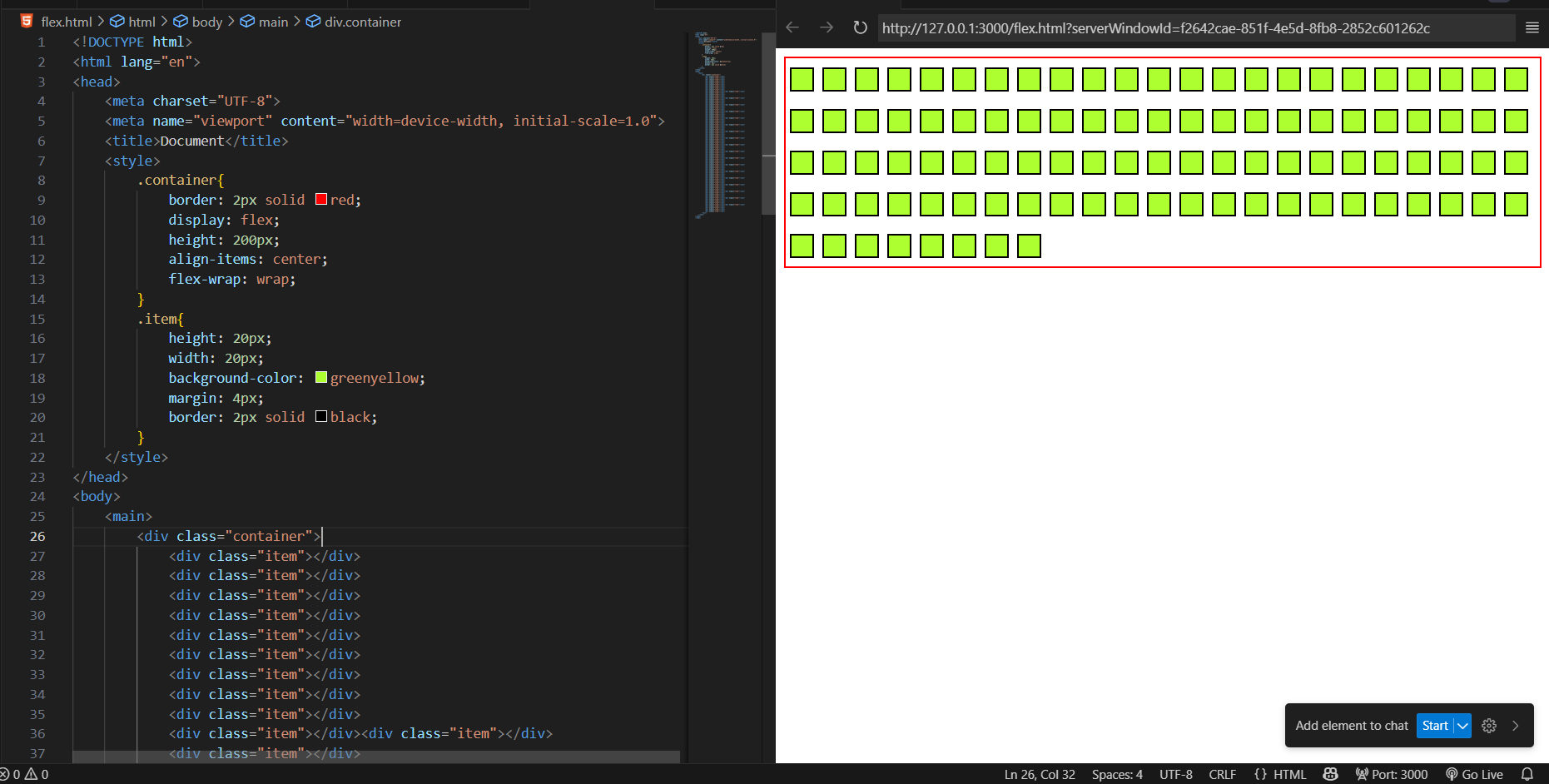


Example: use case of flex-direction:row-reverse



Now, suppose we have increased the number of <div> too much, then it will overflow. As shown below:

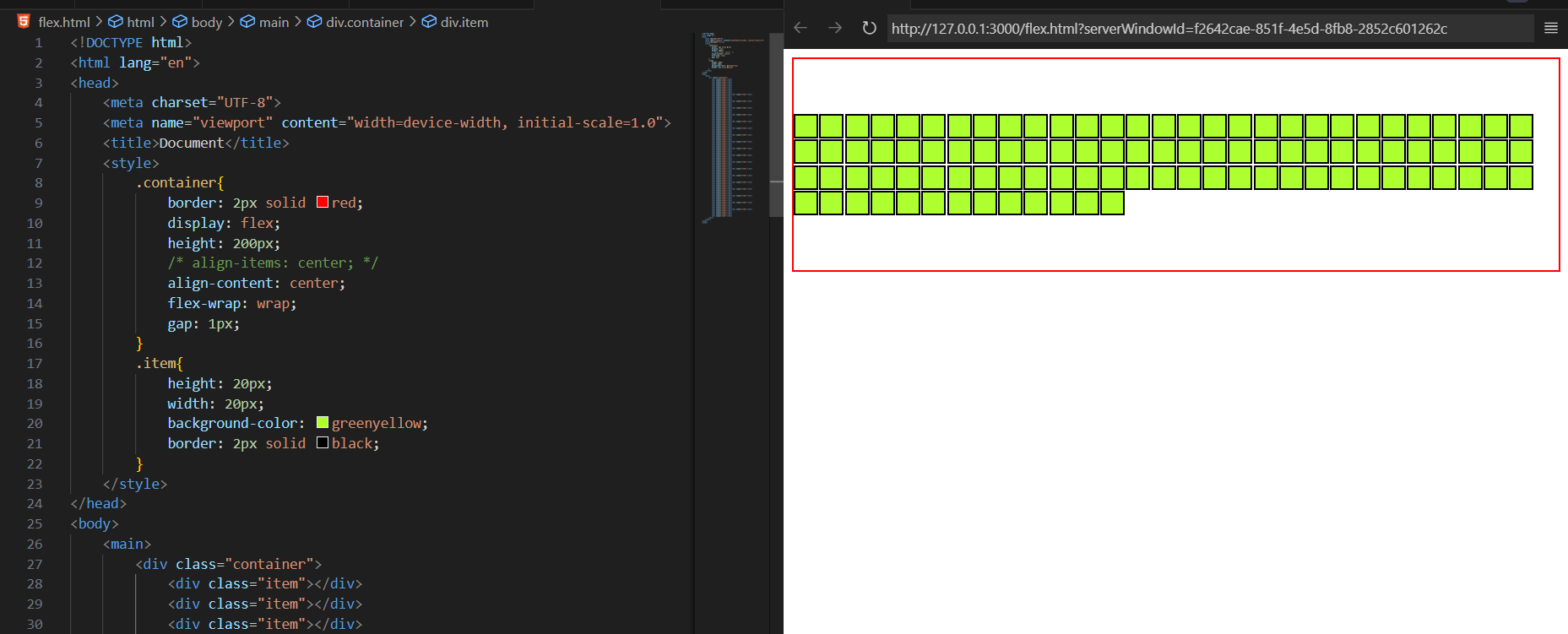


To fix, this we will use the property, flex-wrap:wrap

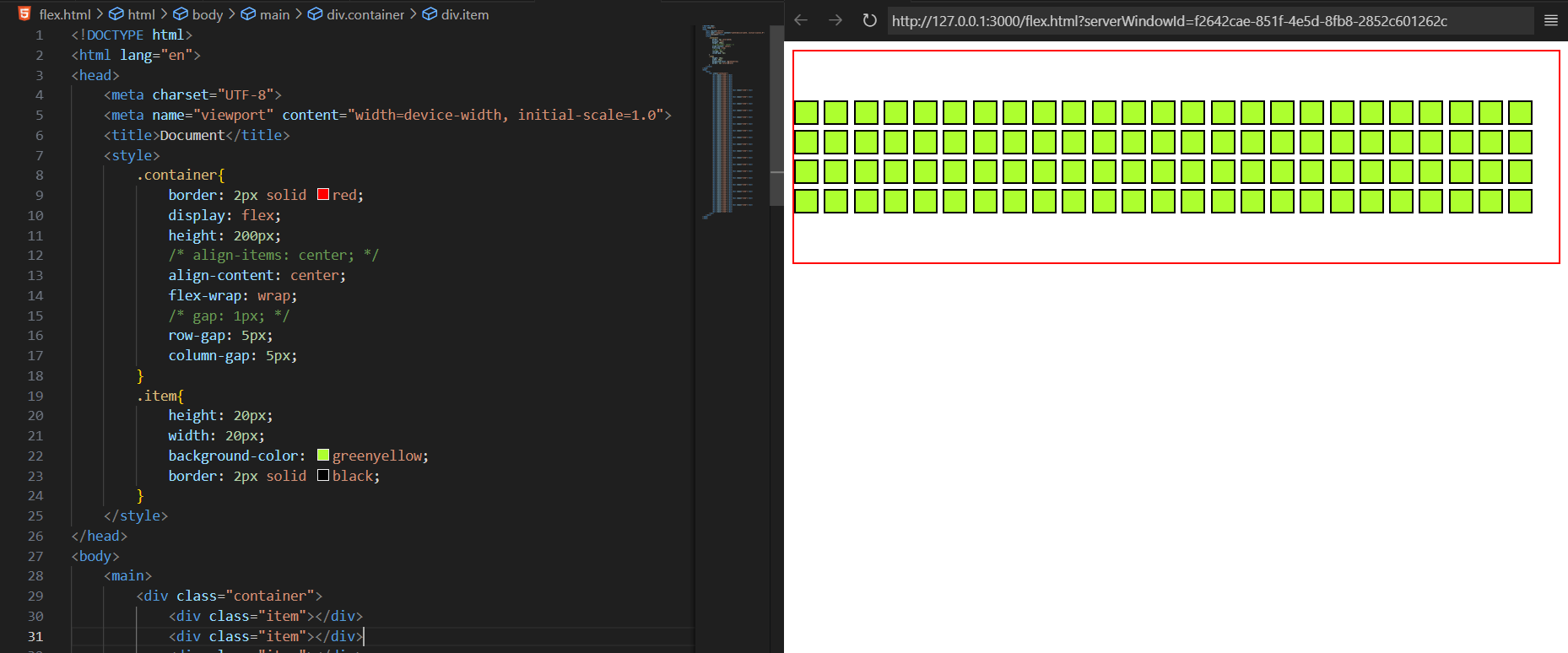
Example: use case of align-content:center. Basically, we use it when the items be in more than one line because of flex-wrap.



Now, adding the gap property in the container:



We can even, separately tell the row and column gap:



--The End--