10 simple steps to master CSS Flexbox:

What is Flexbox?

CSS Flexbox, short for Flexible Box Layout, is a powerful and efficient layout module that allows developers to design complex and responsive web layouts more easily and predictably. It provides a method for aligning and distributing space among items in a container, even when their sizes are unknown or dynamic.

It is one of the most useful tool in CSS styling and provides flexibility of use with a faster layout creation, saves development time and increases creativity.

Importance Of CSS Flexbox

- Simplifies Complex Layouts: Easily manage the distribution of space and alignment of items.
- Responsive Design: Automatically adjusts items to fit different screen sizes and orientations.
- Improved Flexibility: Allows for dynamic changes in the layout without using floats or positioning.
- Vertical and Horizontal Centering: Effortlessly center items within a container.

Key Concepts Of Flexbox

The Key concepts of the CSS Flexbox are the flex container and the flex items

Flex Container: An element that has **display: flex** or **display: inline-flex**. It serves as the parent element that holds the flex items.

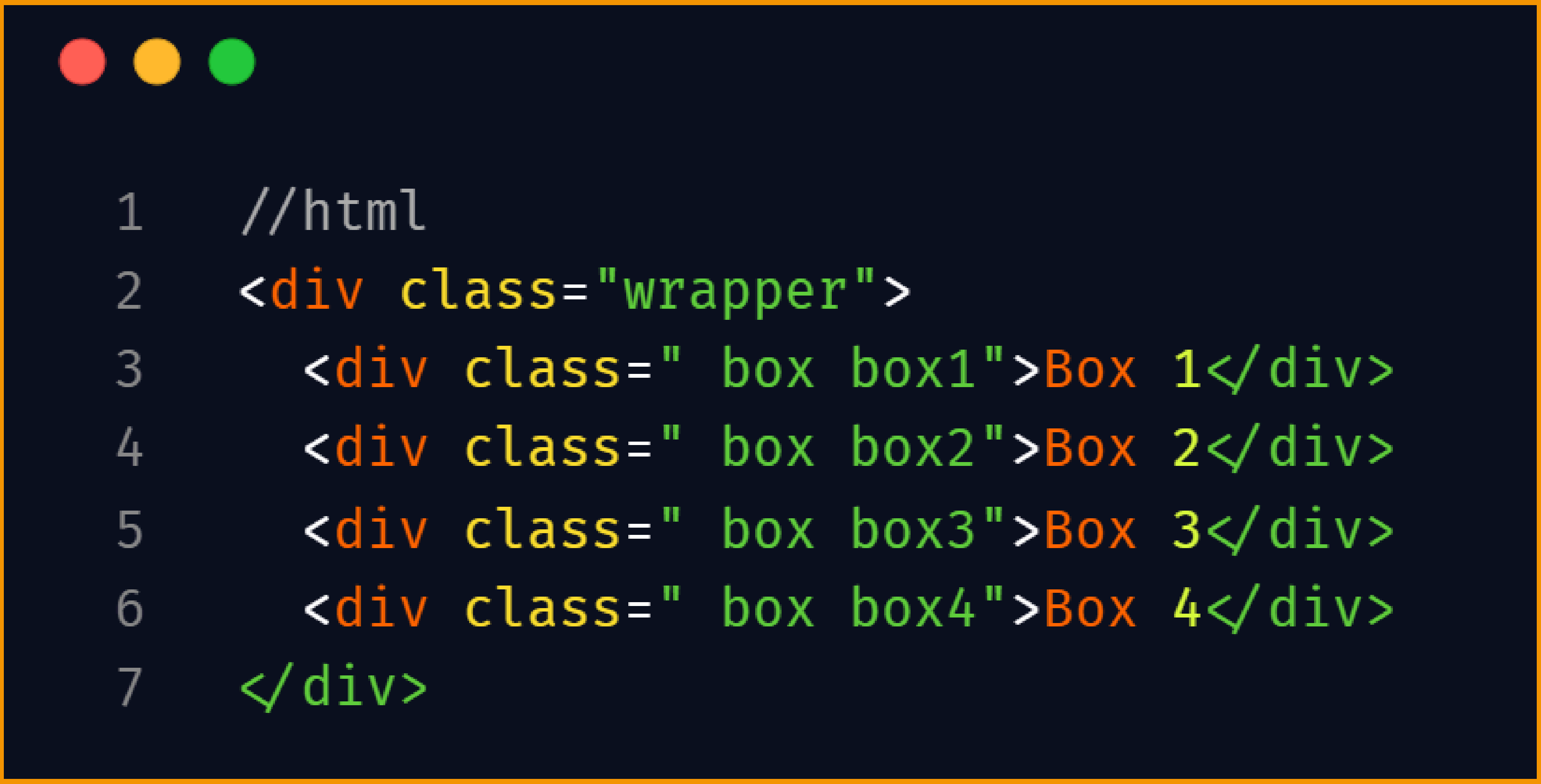
Flex Items: The direct children of a flex container. These items are laid out along a flex line, which can be either horizontal or vertical depending on the container’s flex direction.

Now let’s learn about the 10 steps to understanding CSS Flexbox:

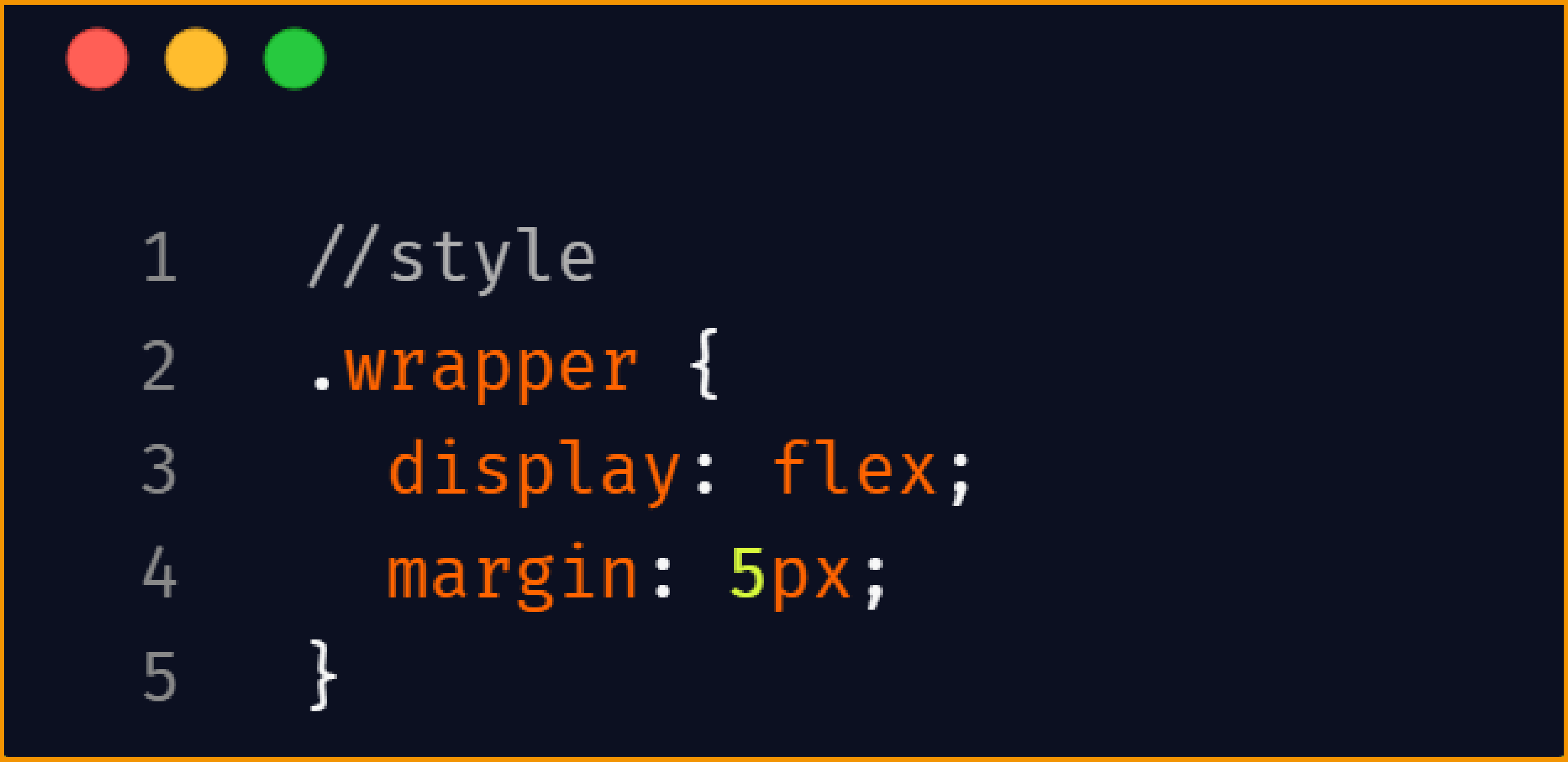
Step 1: Define a Flex container

This sets the stage for flexible layouts.

For example, consider these:

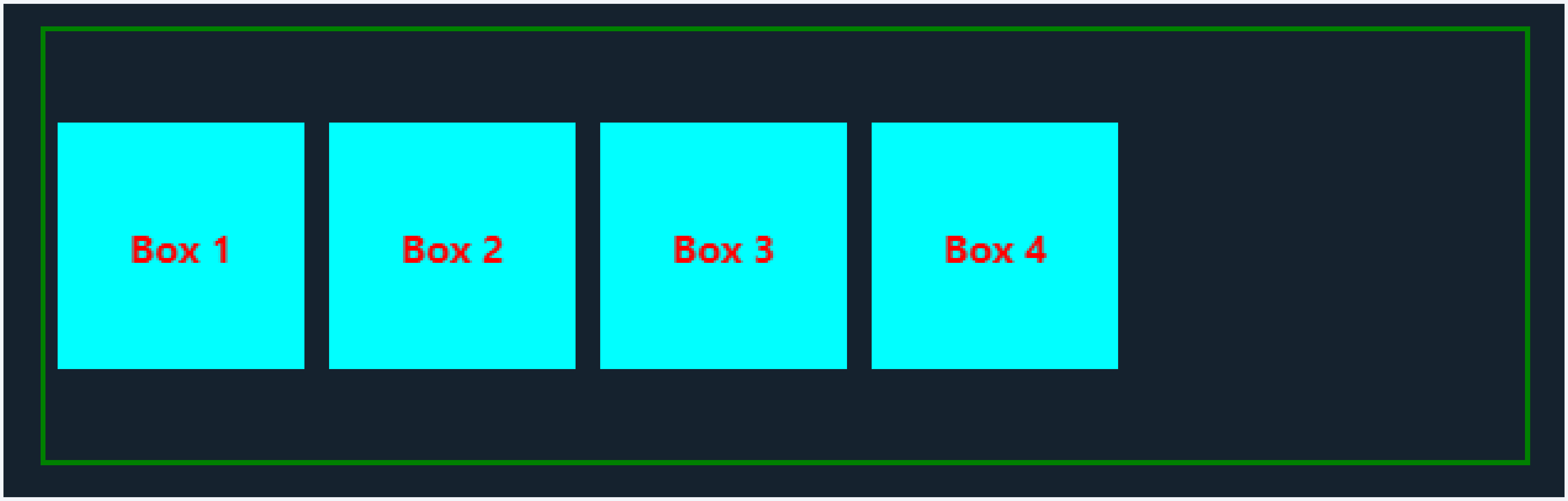


```
//html
<div class="wrapper">
  <div class=" box box1">Box 1</div>
  <div class=" box box2">Box 2</div>
  <div class=" box box3">Box 3</div>
  <div class=" box box4">Box 4</div>
</div>
```



```
//style
.wrapper {
  display: flex;
  margin: 5px;
}
```

Result:



The **display: flex;** property makes the container a flex container. All direct children of the container become flex items.

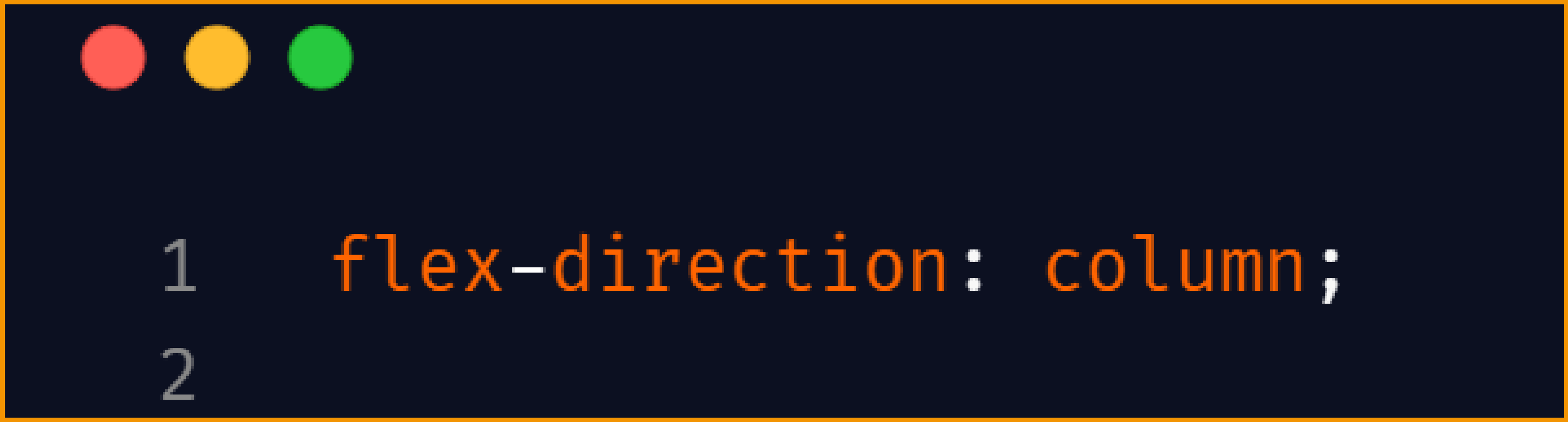
The **margin** property is used to set the space between the flexed items for proper visuality. We will use another property to achieve this as we learn along.

Step 2: Set the Flex Direction

This controls the direction of your flex items. The default direction for flexed items is row. this direction can be set to fit the content depending on the screen or viewport width.

The **flex direction** property is for column is usually set like this:

For example:

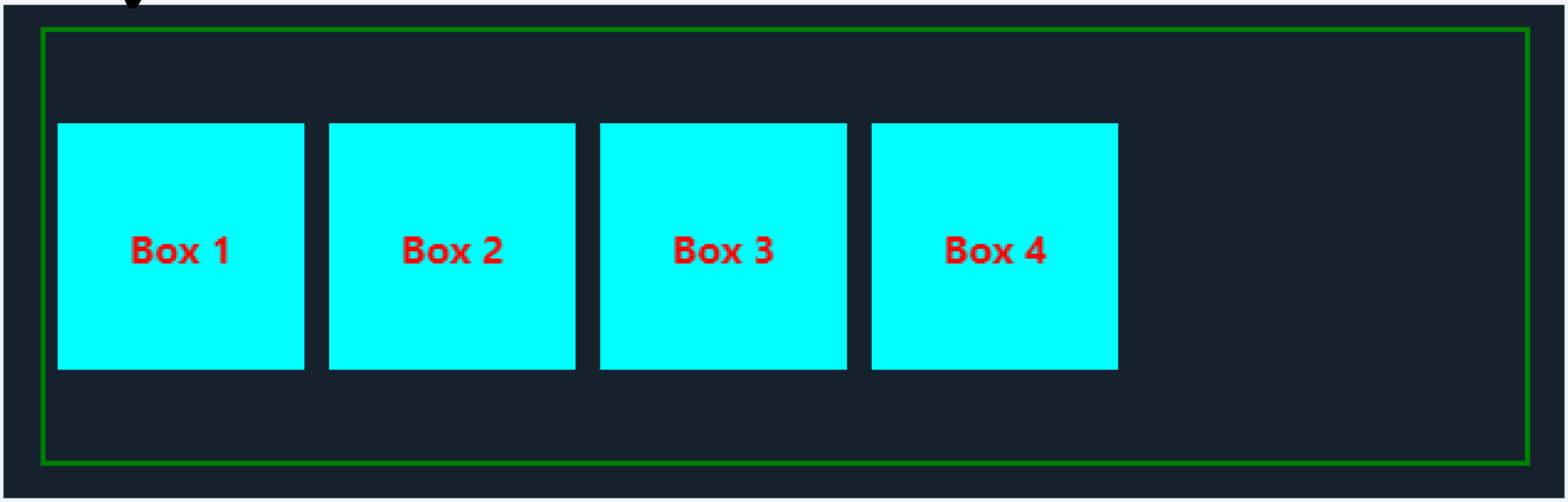


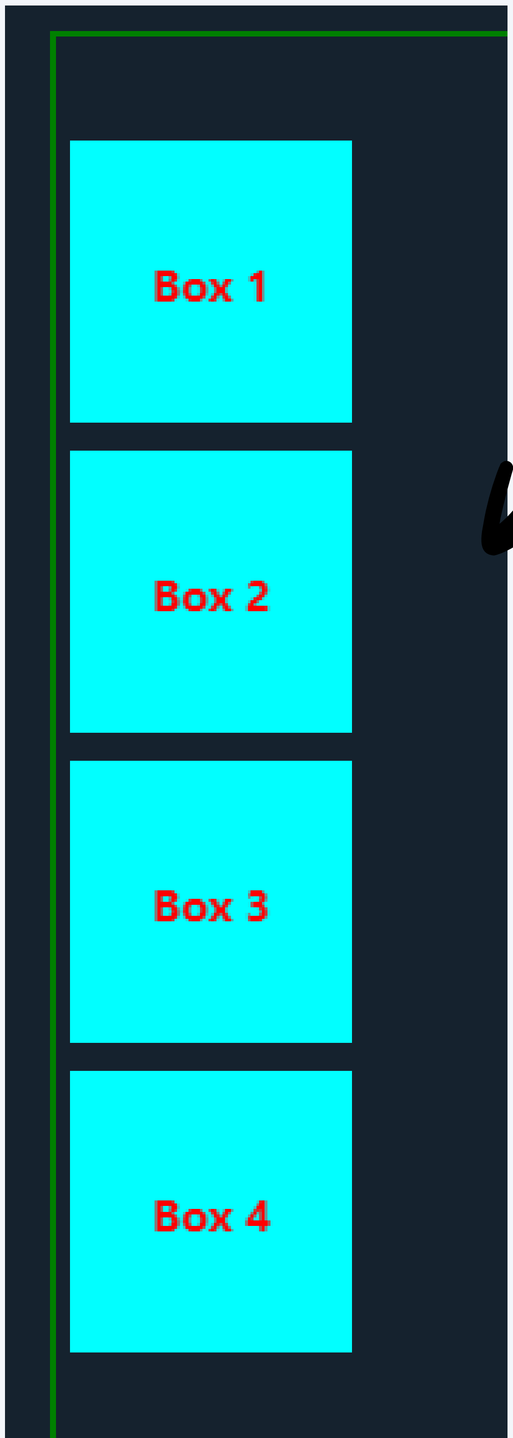
```
flex-direction: column;

```

Result:

Default orientation (**row**). Items are aligned horizontally.



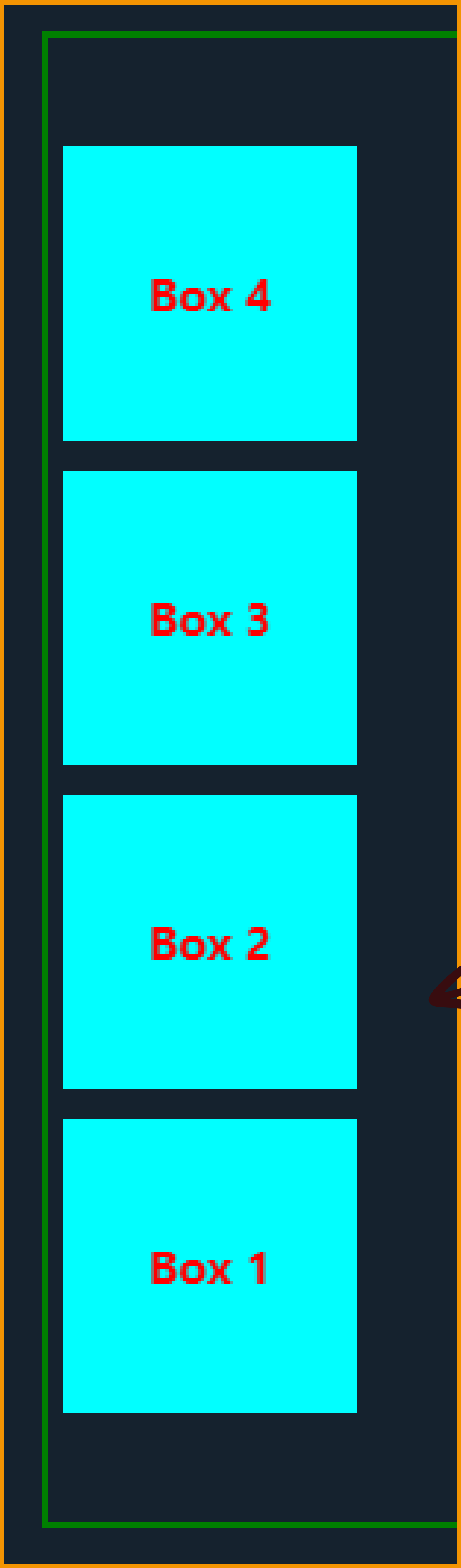


The **flex-direction** is set to **column** to align the flex items on the vertical axis.

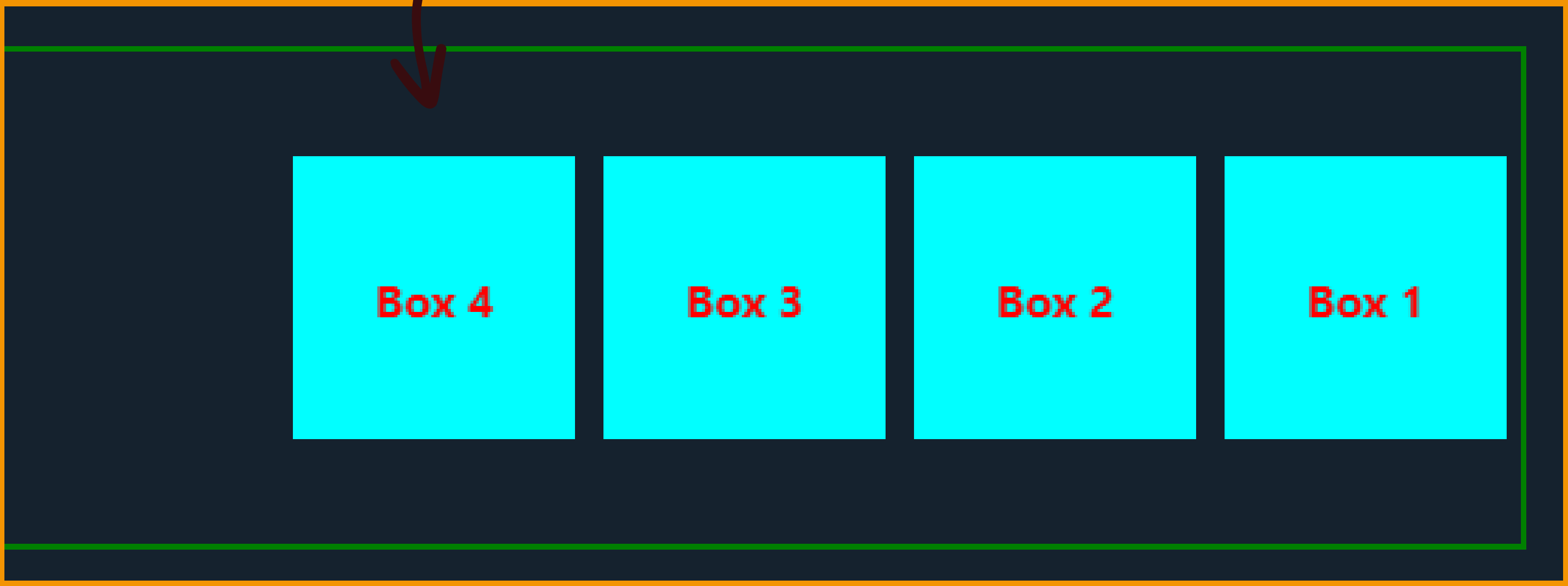
This is especially useful for mobile view layouts.

Other values of flex-direction are:

- row-reverse: reverse row from right to left.
- column-reverse: reverse column from bottom to top.



row-reverse (Box 4 - Box 1



column-reverse (Box 4 - Box 1

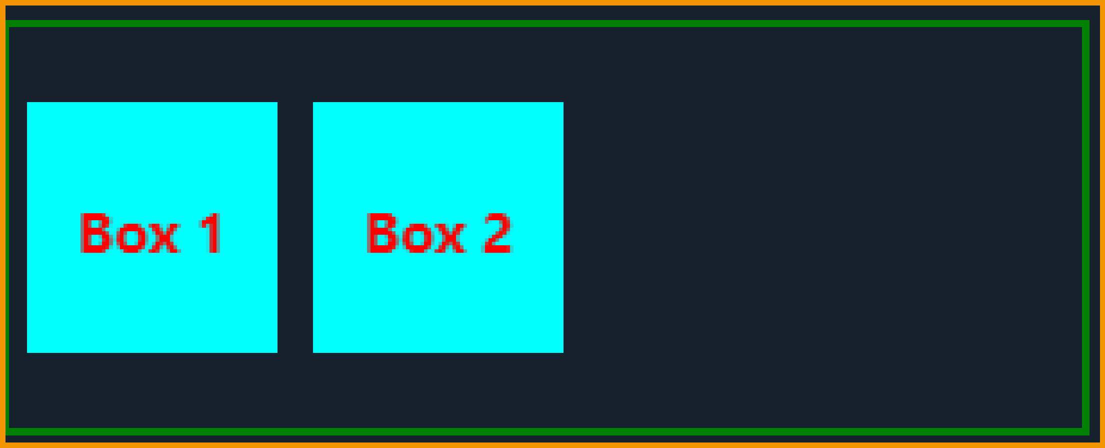
Step 3: Align Items Along the Main Axis

This helps distribute space between items horizontally. The different values for **justify-content** property are:

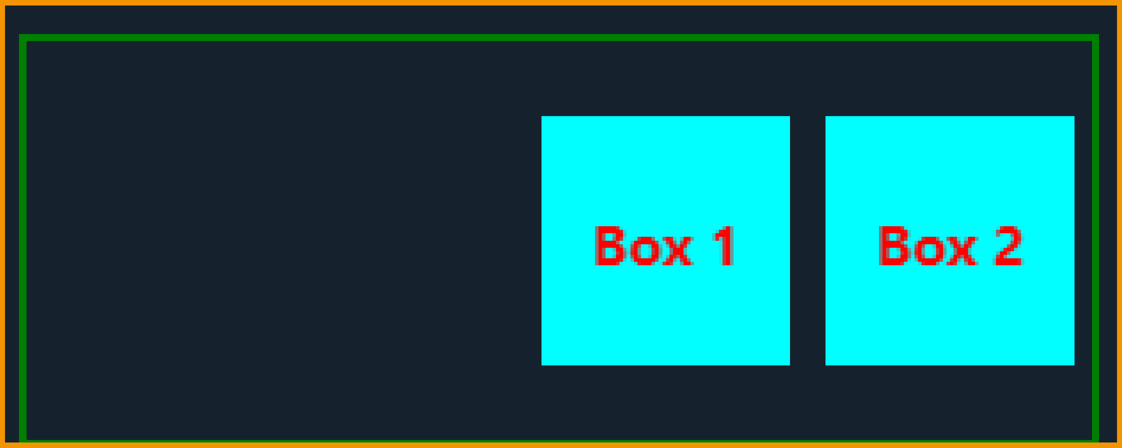
- **flex-start (default):** items are packed toward the start of the flex-direction.
- **flex-end:** items are packed toward the end.
- **center:** items are centered along the line.
- **space-between:** items are evenly distributed, with the first item at the start and the last item at the end.
- **space-around:** items are evenly distributed with equal space around them.
- **space-evenly:** items are distributed so that the spacing between any two items (and the space to the edges) is equal.

With the syntax: **justify-content: {value},** we have:

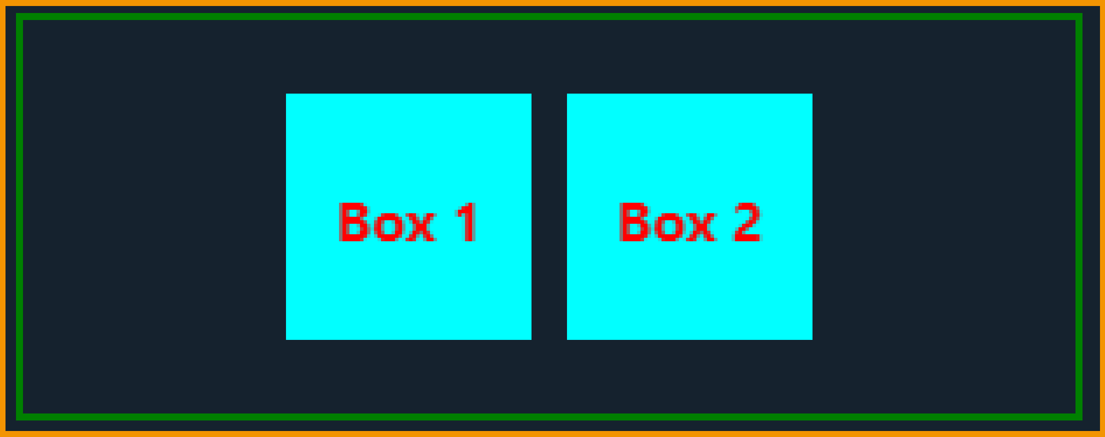
value: flex-start



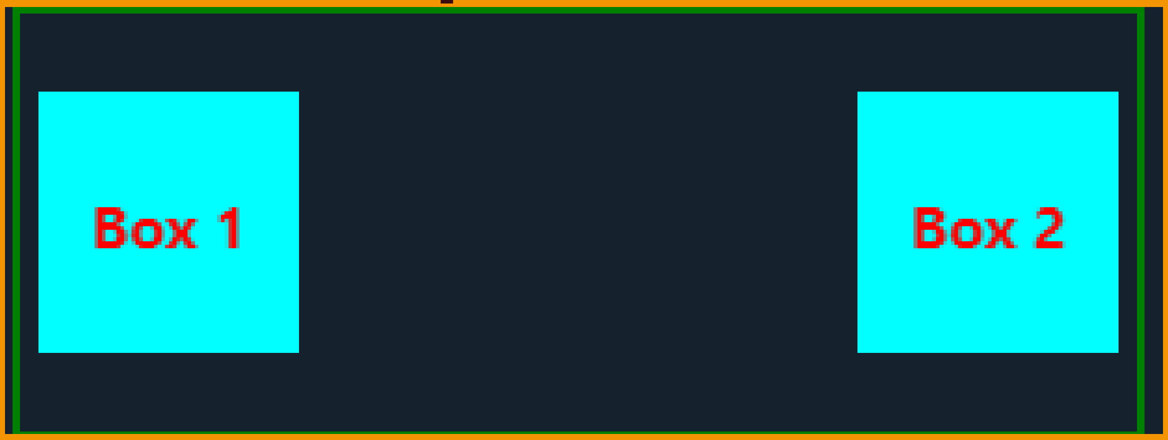
value: flex-end



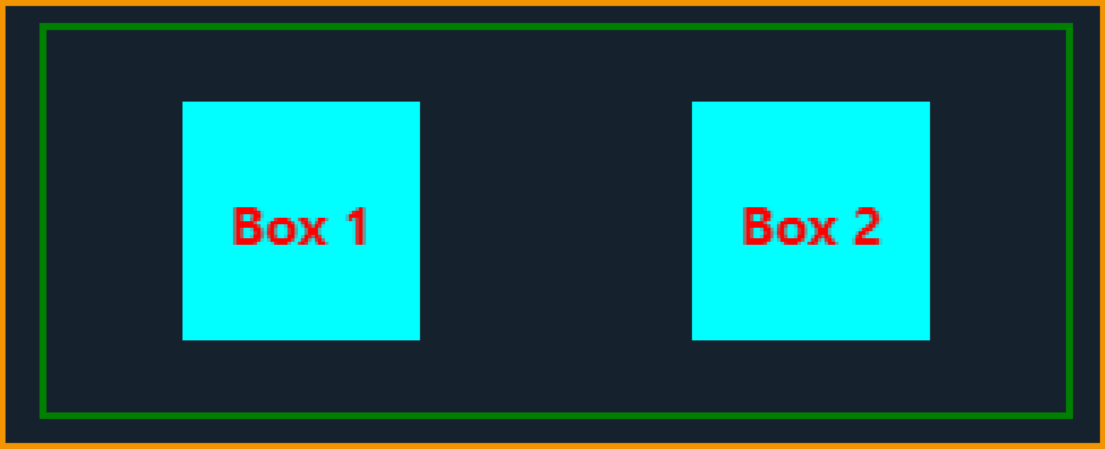
value: center



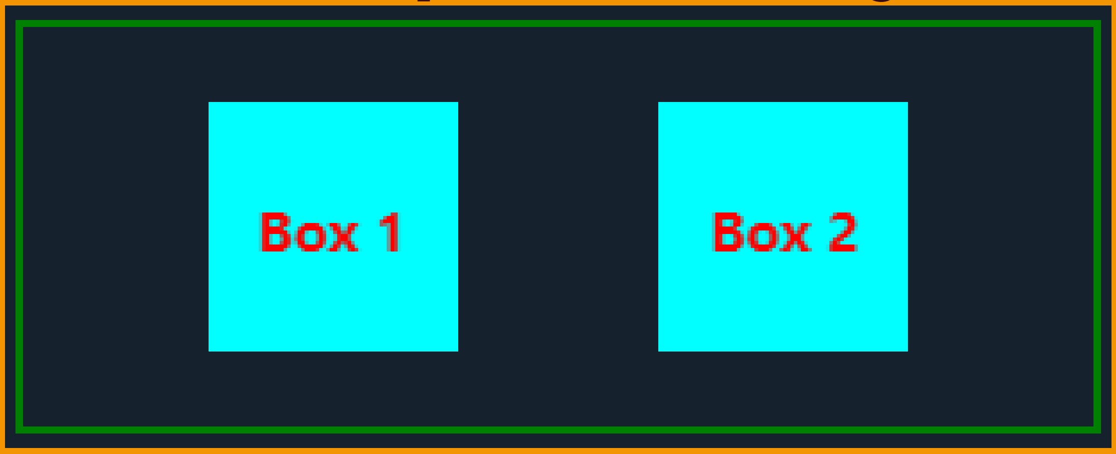
value: space-between



value: space-around



value: space-evenly



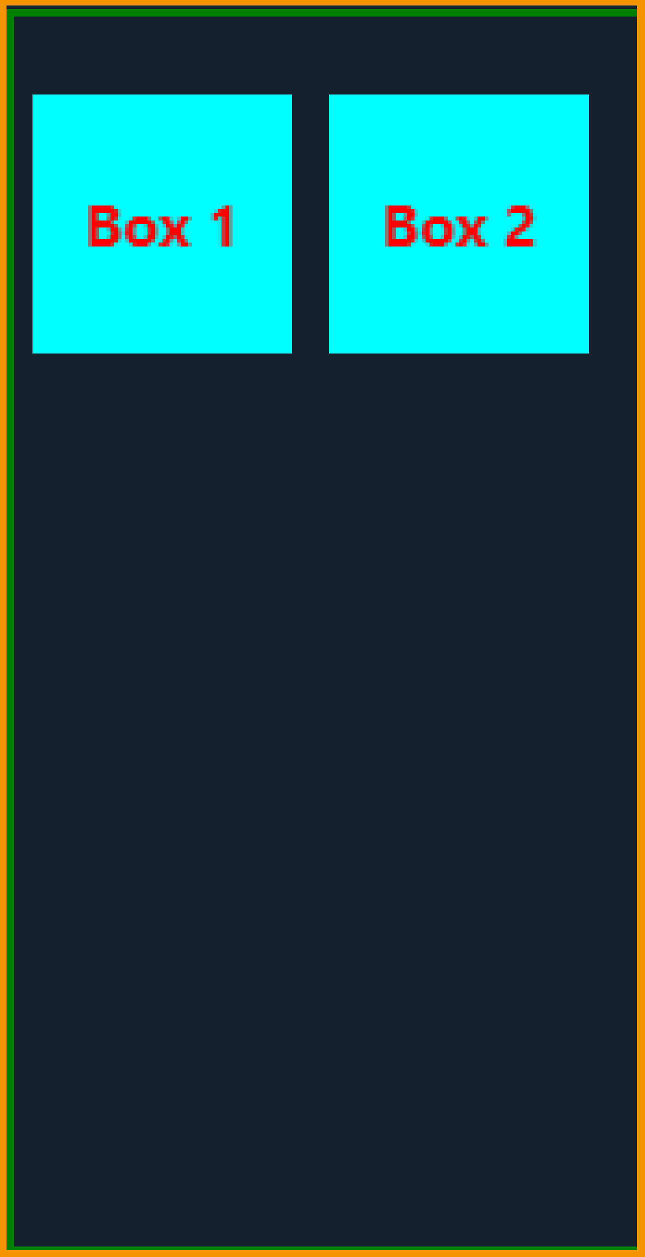
Step 4: Align Items Along the Cross Axis

This ensures items are alligns properly along the horizontal axis. The different values for **align-items** property are:

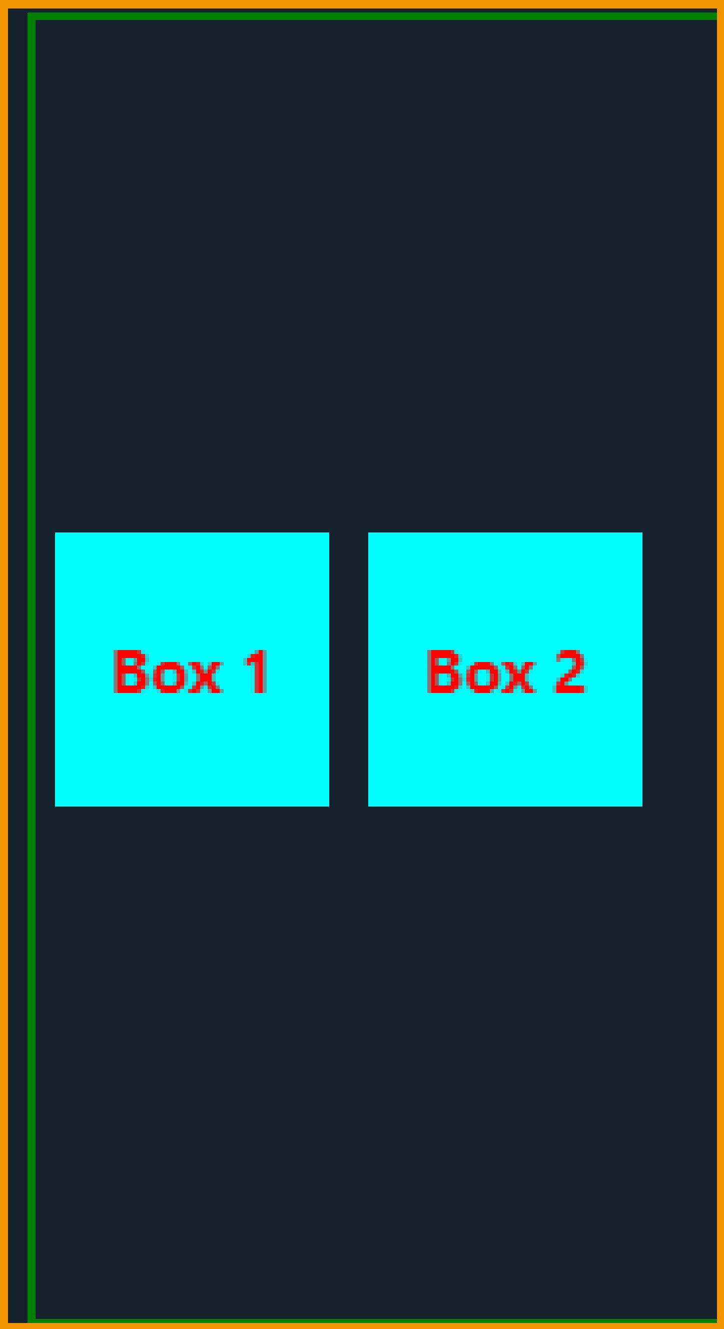
- **flex-start:** Align items to the start of the cross axis.
- **flex-end:** Align items to the end of the cross axis.
- **center:** Center items along the cross axis.
- **stretch:** Stretch items to fill the container. This is the default.

With the syntax: **align-items: {value},** we have:

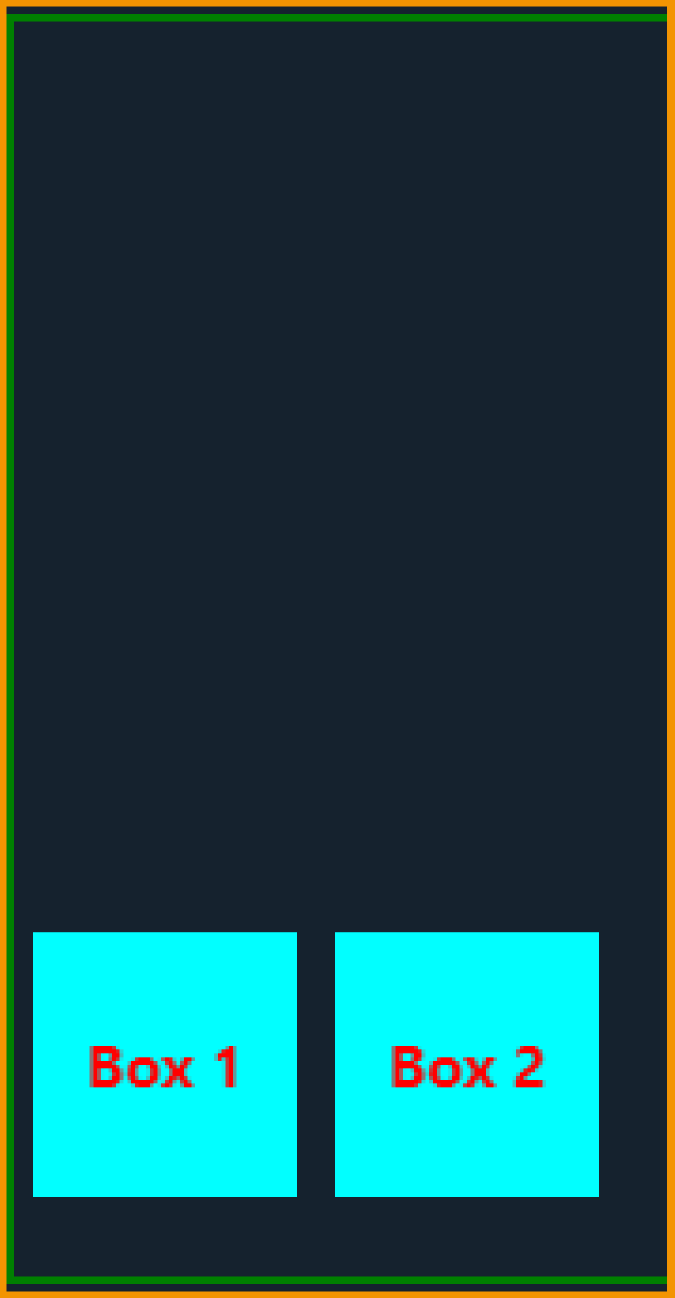
value: flex-start



value: center



value: flex-end



Step 5: Set flex-wrap to handle overflow

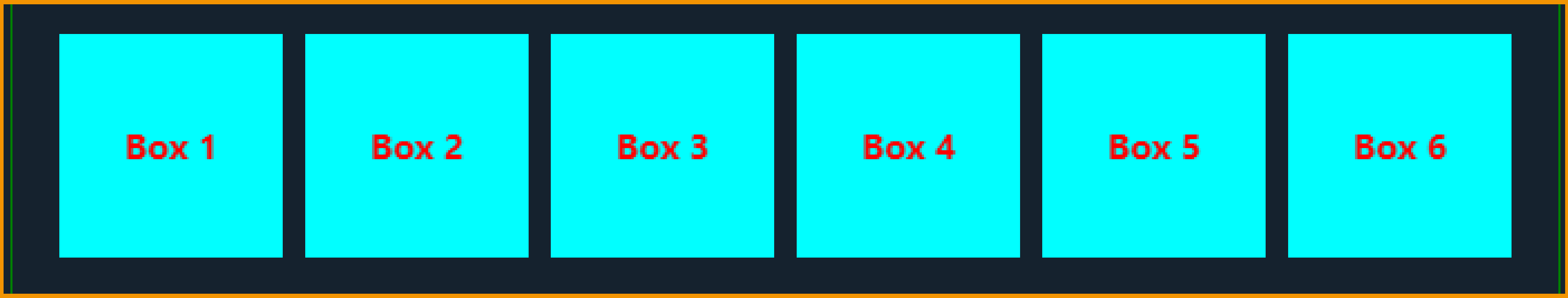
This property is crucial for managing the layout of items when there are too many to fit into one line, especially in responsive designs.

By default, flex items will try to fit into a single line. If they overflow, they may extend outside the container, which can cause layout issues.

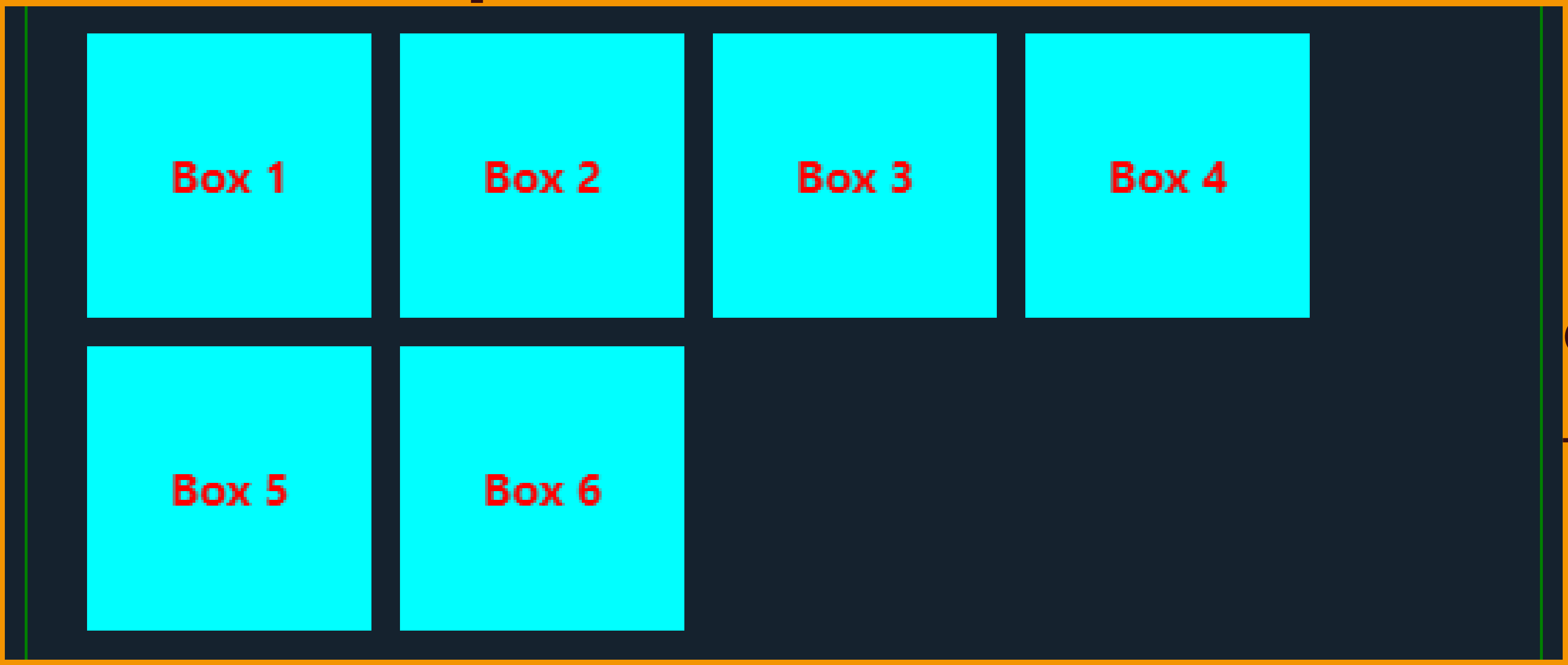
The **flex-wrap** property allows you to change this behavior, enabling items to wrap onto multiple lines, ensuring that the content stays within the boundaries of the container and maintains a more flexible and manageable layout.

With the syntax: **flex-wrap: {value},** we have:

value: nowrap(default) - Items are placed in a single line.

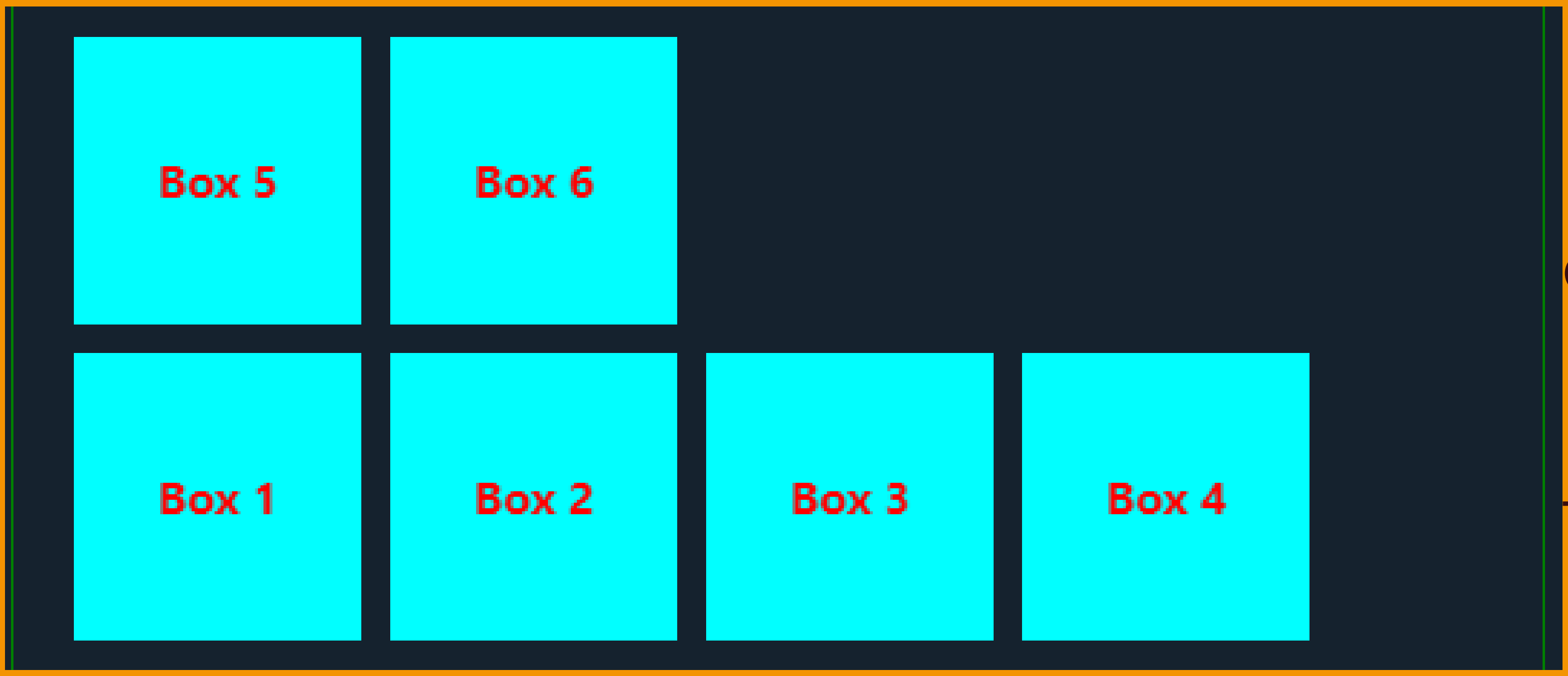


value: wrap



Flex items will wrap onto multiple lines, from top to bottom.

value: wrap-reverse



Flex items will wrap onto multiple lines, but in reverse order, from bottom to top.

Step 6: Control Item Growth with Flex Grow

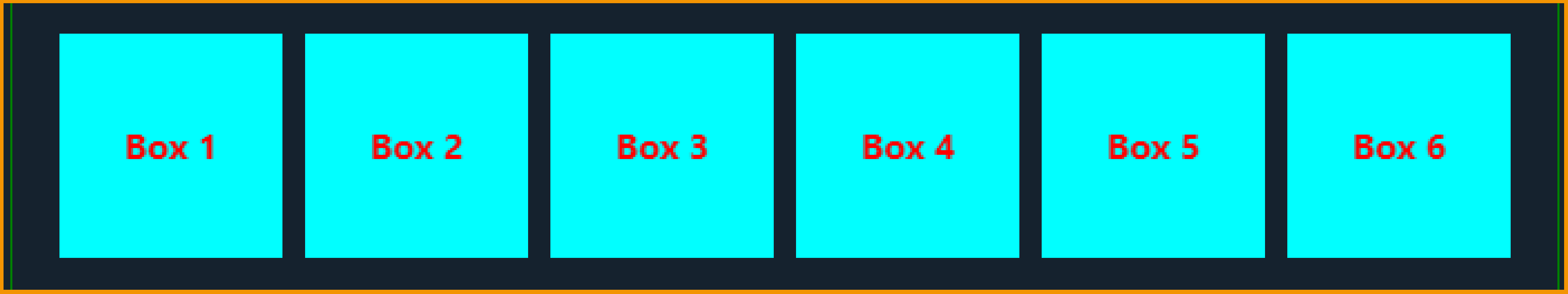
It defines the ability of a flex item to grow and occupy available space in the container.

flex-grow is a numeric property that determines how much a flex item will grow relative to other items in the flex container when there is extra space available. This property is a part of the flex shorthand property (flex-grow, flex-shrink, flex-basis) but can also be used independently.

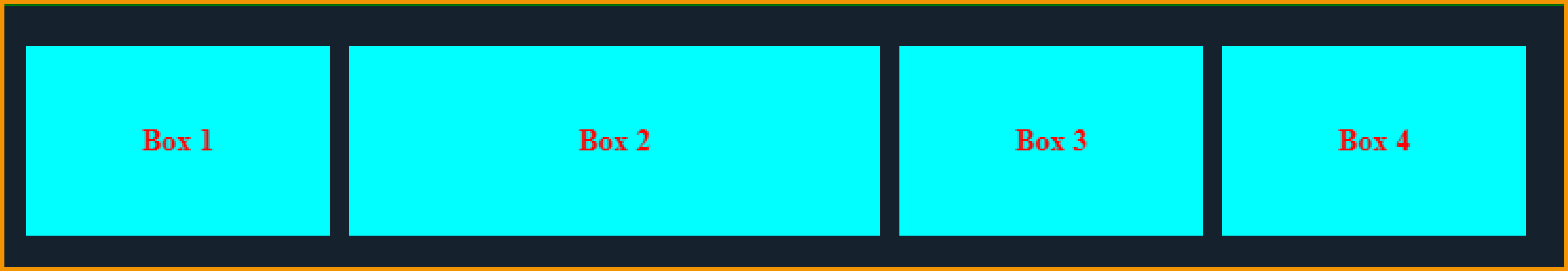
It usually has a value of a non-negative number representing the proportion of available space the item should take up. The default value is 0, meaning the item will not grow to fill the available space.

With the syntax: **flex-grow: {number},** we have:

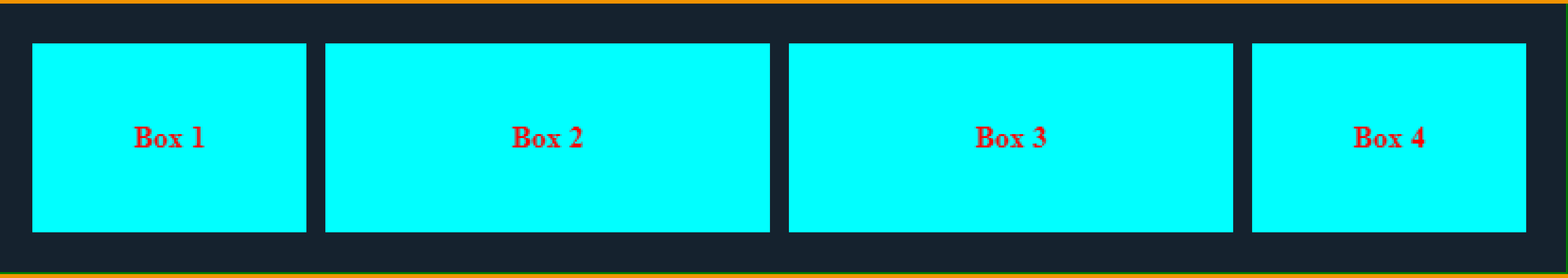
number: 0 (default) - No item is growing.



Box 2 has a flex grow with value of 3



Boxes 2 & 3 has a flex goal with value of 3, while the rest shares the remaining spaces equally.



Step 7: Control Item Shrinkage with Flex Shrink

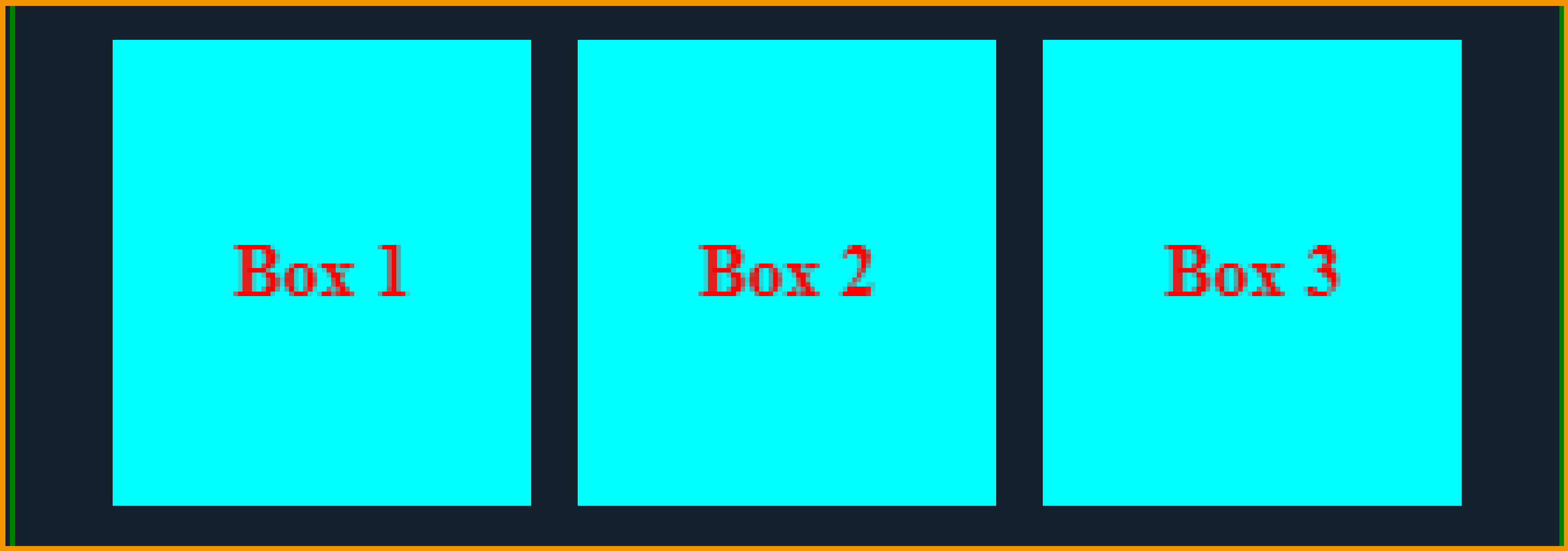
The **flex-shrink** property in CSS Flexbox is used to specify how much a flex item should shrink relative to the rest of the flex items inside the same container when there is not enough space available.

The shrinking only occurs when the flex container is too small to fit all the items at their natural sizes.

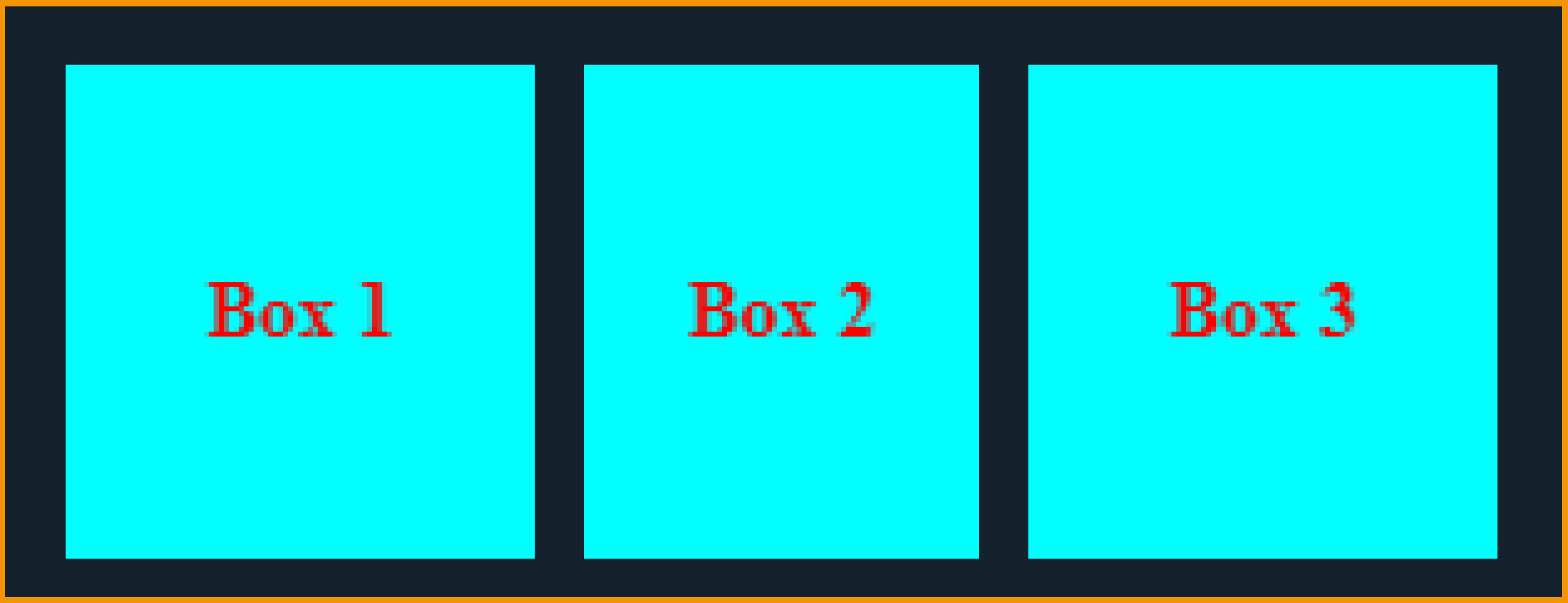
The space reduction is distributed based on the shrink factor of each item. The **flex-shrink** property also takes a non-negative number as it value.

With the syntax: **flex-shrink: {number},** we have:

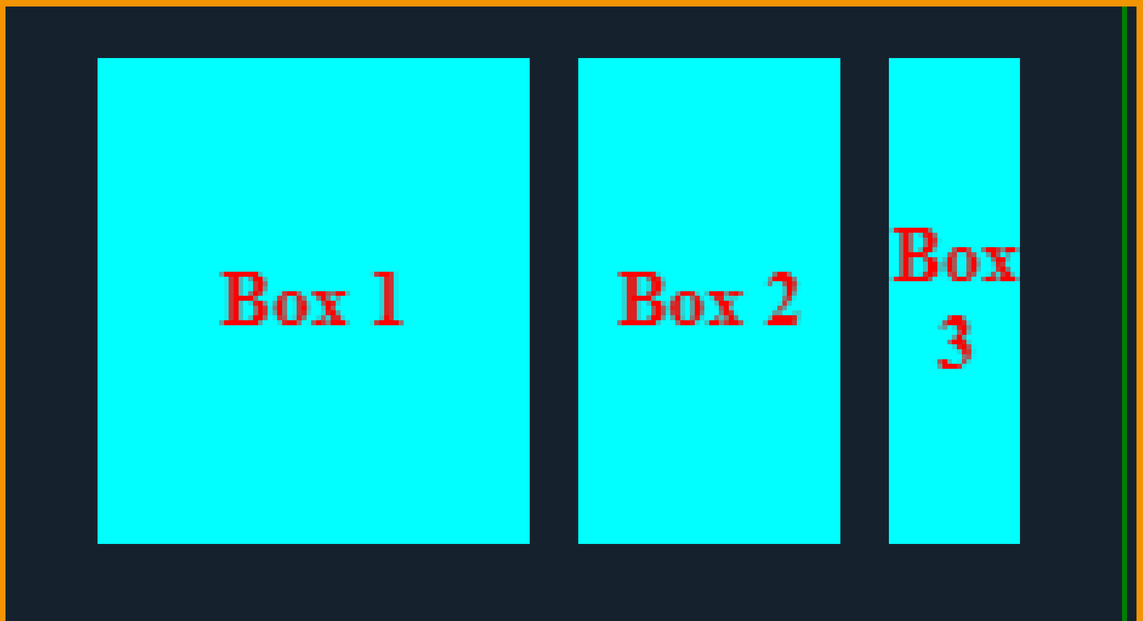
number: 0 (default) - No item shrinking.



Box 2 has a flex shrink value of 2 - shrinks than others



Box 2 has a value of flexshrink of 2 while box 3 has 3



Step 8: Set Initial Size with Flex Basis

The **flex-basis** property in Flexbox, determines the starting point for the size of a flex item. This is the size the item will take up before any extra space is added or taken away due to the flex-grow and flex-shrink properties.

The size is relative to the main axis of the flex container. If the **flex-direction** is **row** or **row-reverse**, flex-basis will define the **item's width**. If the **flex-direction** is **column** or **column-reverse**, it will define the **item's height**.

The values of flex-basis property are in:

- **Length values** (e.g., px, em, rem, %, etc.): These are explicit sizes. For example, flex-basis: 200px; sets the flex item's base size to 200 pixels.
- **auto:** This is the default value. It means the size is determined by the item's content or width/height properties.
- **content:** This value sets the base size to the size of the item's content. However, it's not widely supported and should be used cautiously.
- **0:** This sets the base size to zero, meaning the item will not take up any space initially and will rely entirely on the flex-grow property to get its size.

Step 9: Align Individual Items with Align Self

The **align-self** property in Flexbox allows individual flex items to override the align-items property of the container. This property is useful when you want to adjust the alignment of a single item differently from the others within the same container.

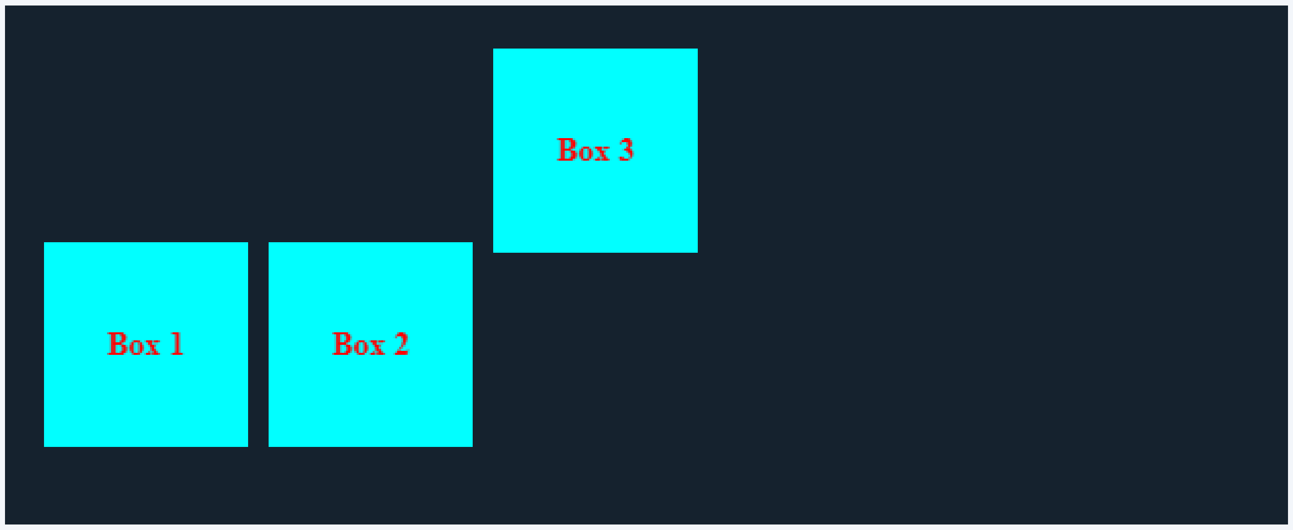
While align-items sets the alignment for all items in the container, align-self allows you to target individual items, giving them unique alignment properties.

The values of align-self property are:

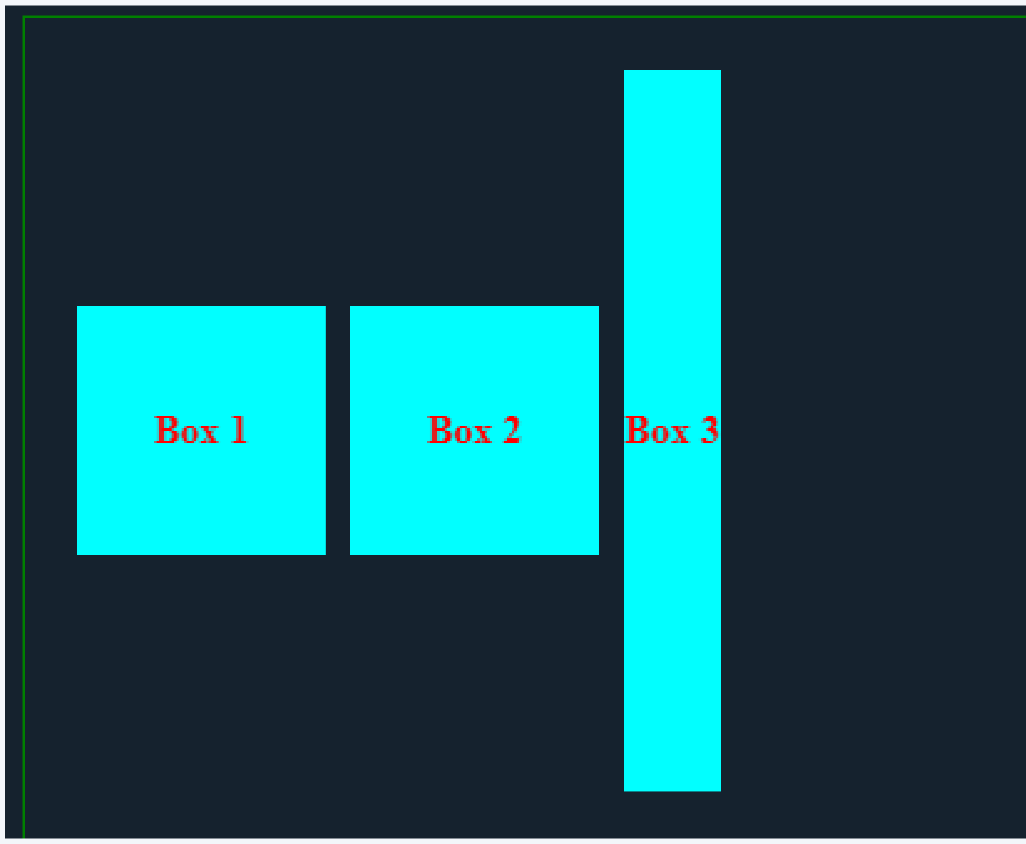
- **auto:** This is the default value, which means the item will follow the align-items value set on the flex container.
- **flex-start:** Aligns the item to the start of the cross axis.
- **flex-end:** Aligns the item to the end of the cross axis.
- **center:** Centers the item along the cross axis.
- **baseline:** Aligns the item's baseline with the baseline of the container.
- **stretch:** Stretches the item to fill the container along the cross axis.

With the syntax on Box3: **Align-self: value,** we have:

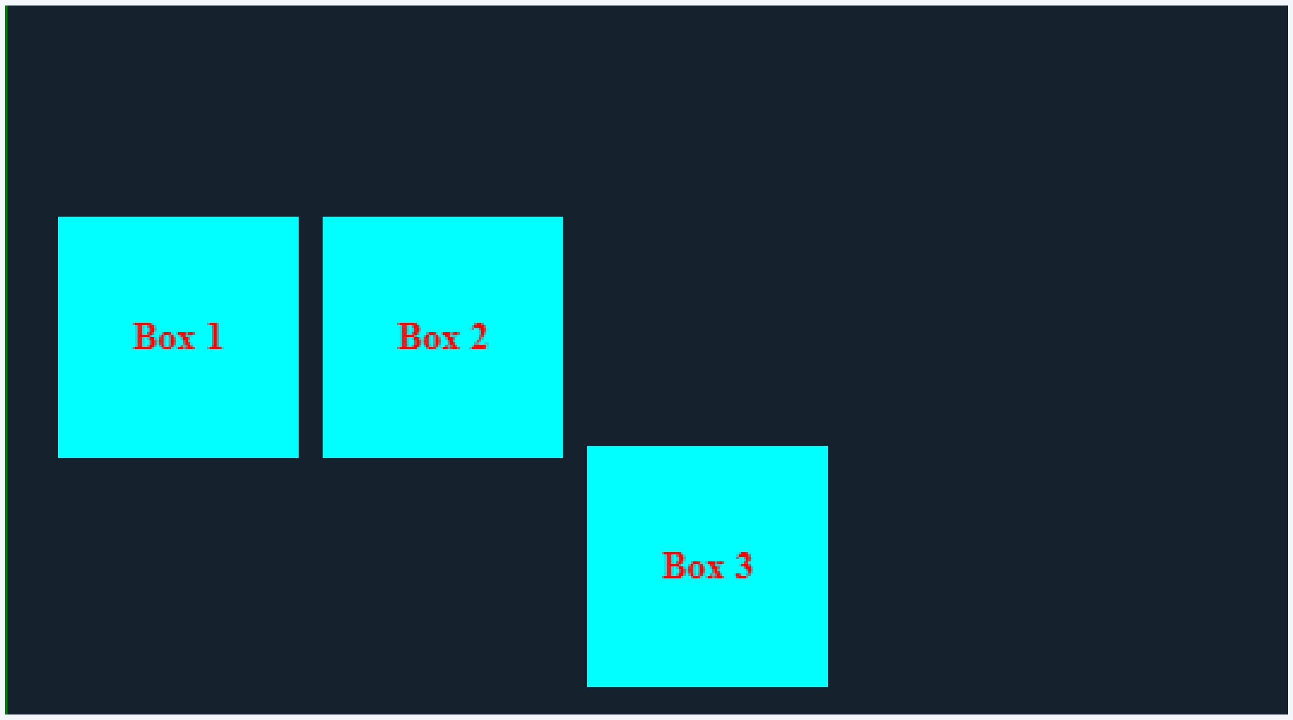
value: flex-start/baseline



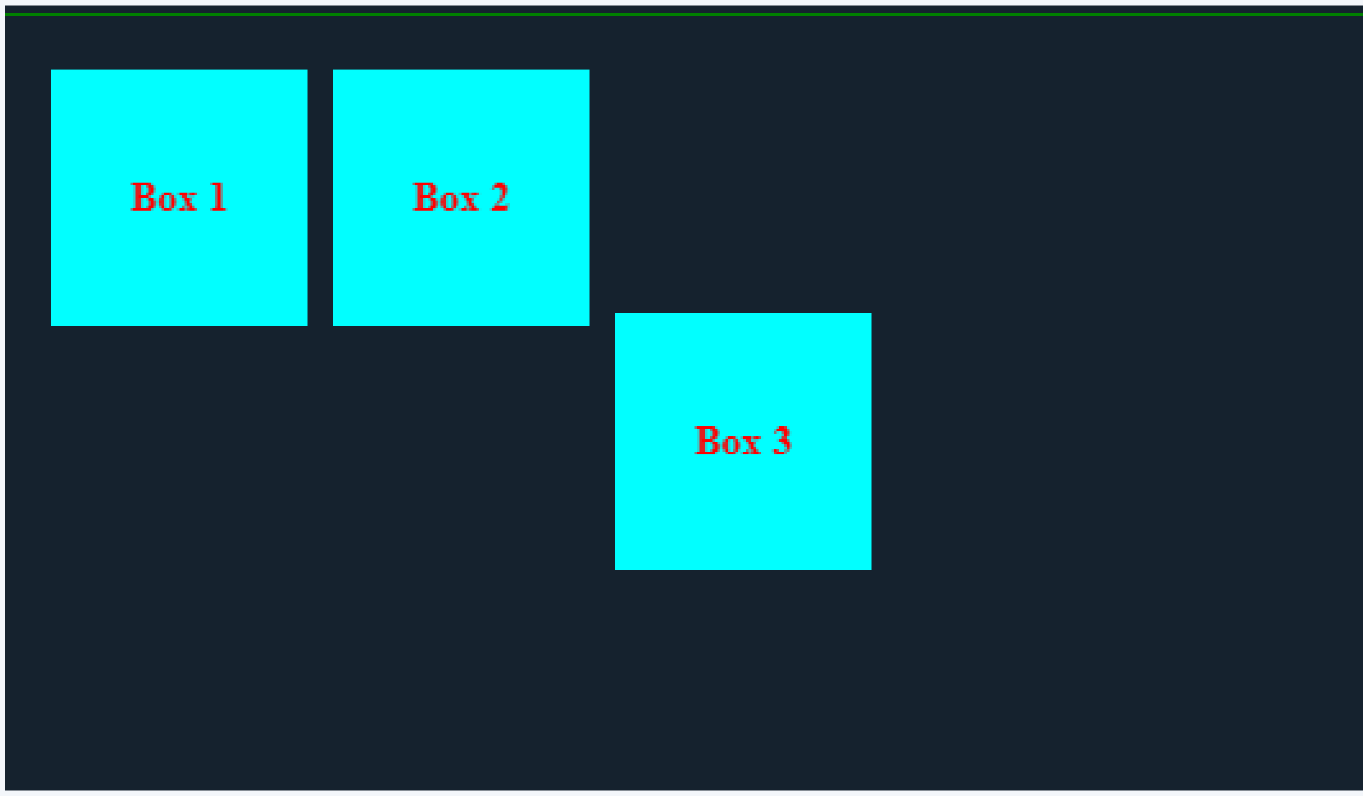
value: stretch



value: flex-end



value: center



Step 10: Add Space Between Items with gap

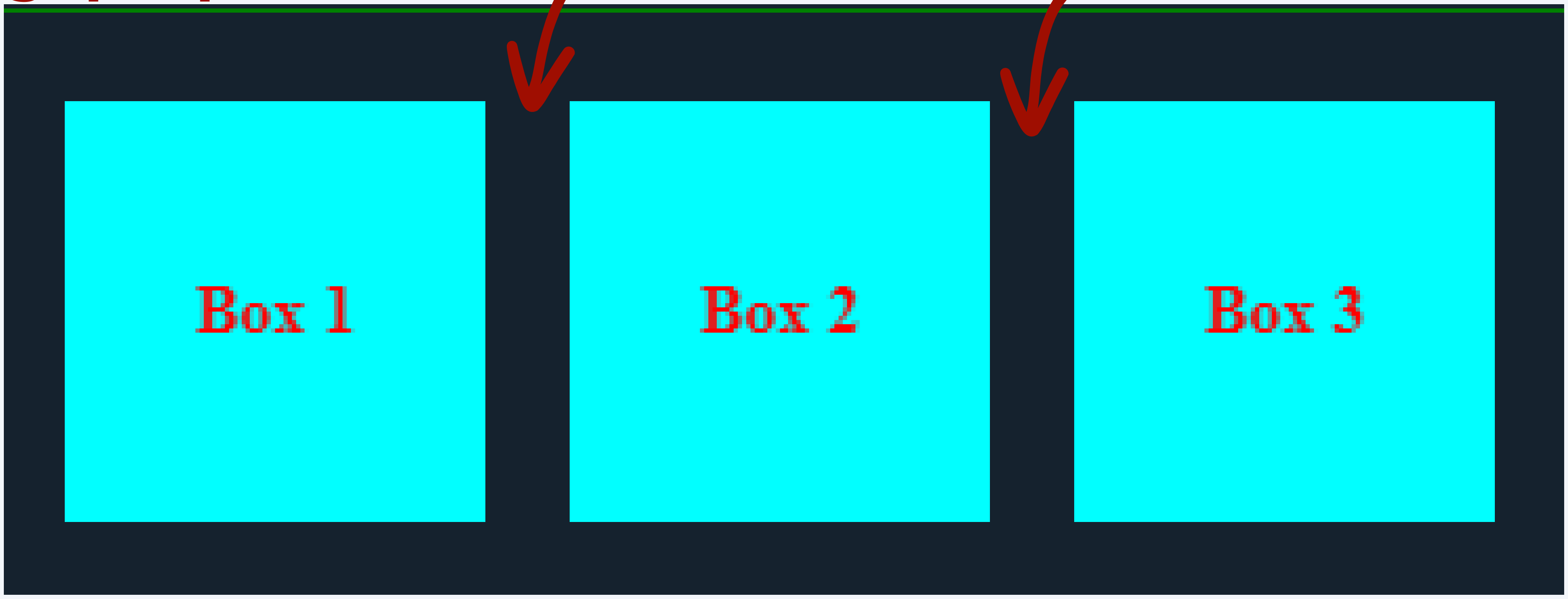
The **gap** property in CSS Flexbox provides a convenient way to control the spacing between flex items.
It eliminates the need for using margins to create space between items, resulting in cleaner and more maintainable code.

- **Single Value:** If you provide a single value, it sets the same space between both rows and columns in a grid or between flex items in a single line.
- **Two Values:** If you provide two values, the first value sets the row gap (vertical spacing), and the second value sets the column gap (horizontal spacing).

With the syntax:

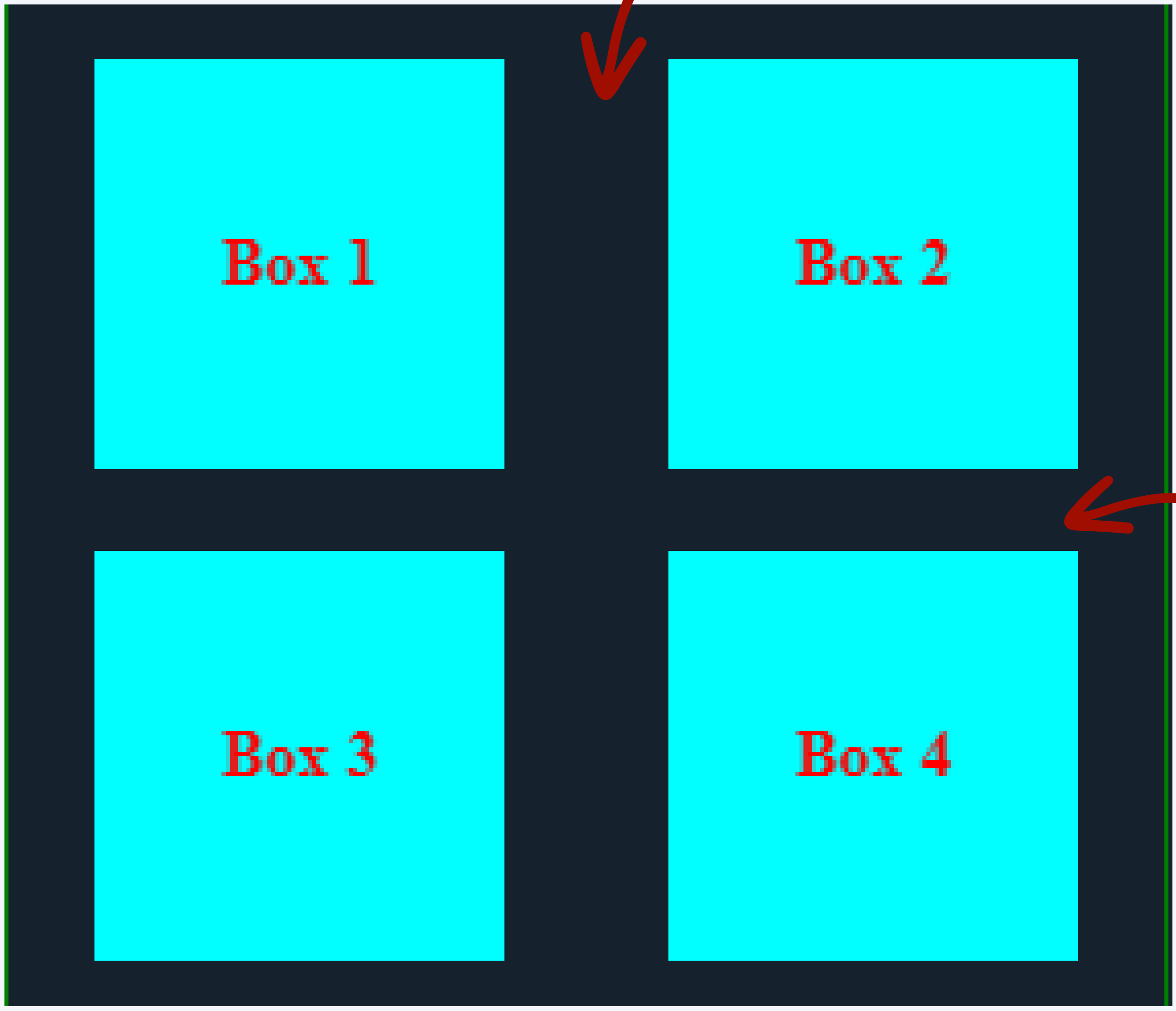
Same value between rows and columns

gap: 5px



row gap along vertical axis

gap: 5px 10px

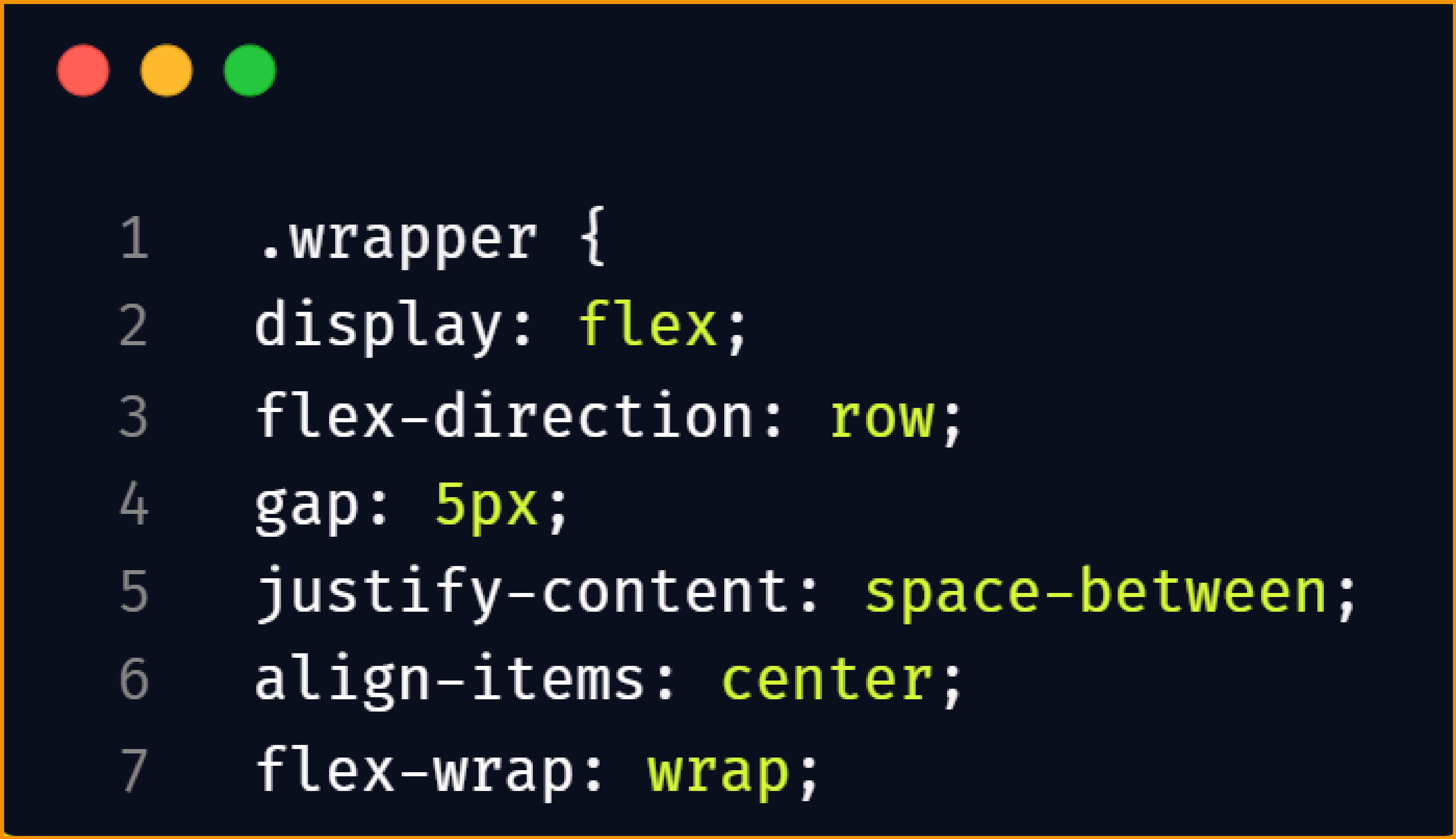


column gap along horizontal axis

Combine Properties for Complex Layouts

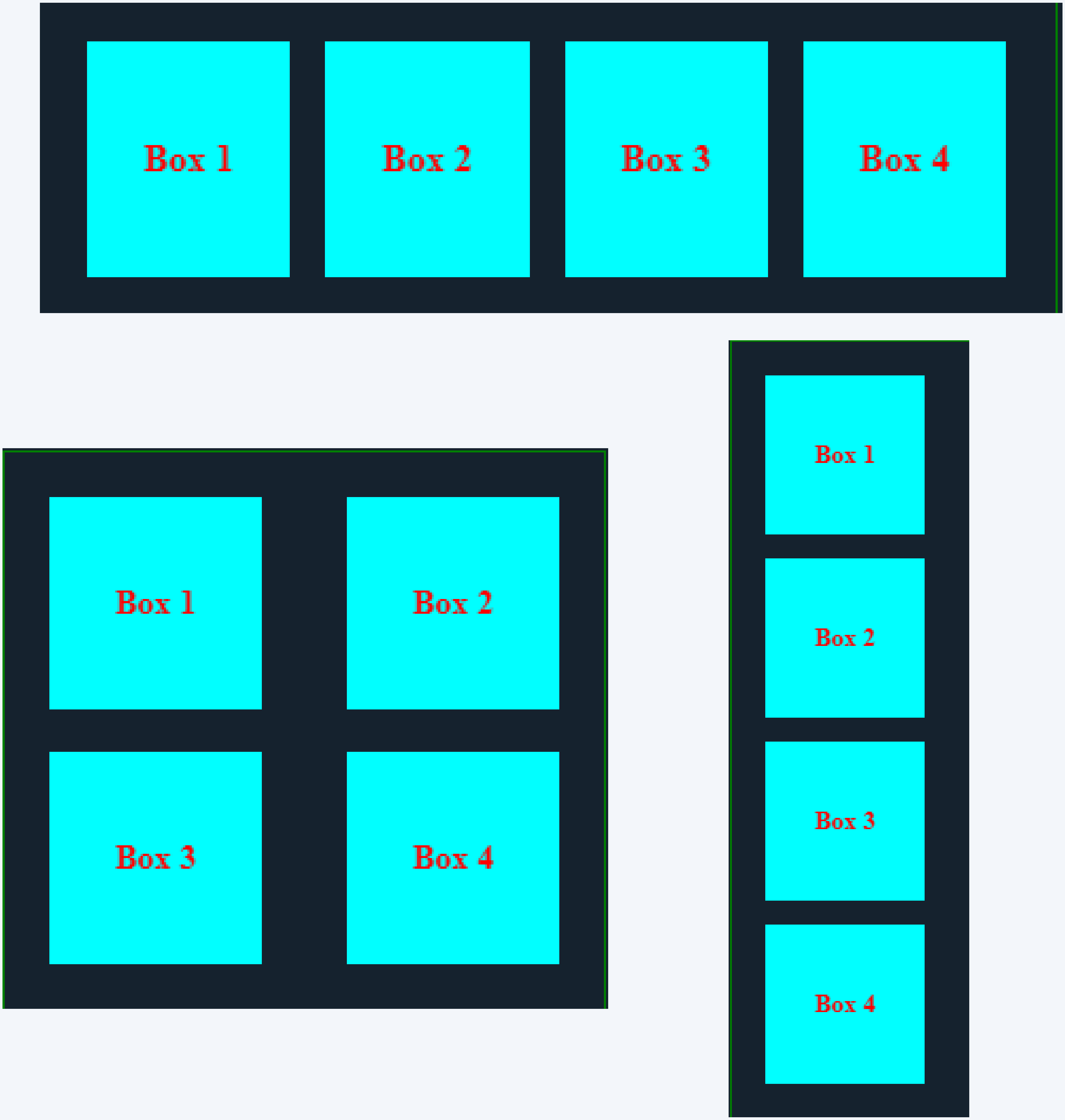
Now, from what we have learnt so far, let’s create a responsive layout by mixing and matching simple **Flexbox properties** to create an intricate layouts.

Check these combination of properties



```css
.wrapper {
display: flex;
flex-direction: row;
gap: 5px;
justify-content: space-between;
align-items: center;
flex-wrap: wrap;
```

Items adjusting based on screen or viewport width



Hi There! 👋

Thank you for reading through

Did you enjoy this knowledge?

💼 Follow my LinkedIn page for more work-life balancing and Coding tips.

🌐 LinkedIn: Oluwakemi Oluwadahunsi

kodemaven-portfolio.vercel.app