
CSS
CSS
<!head>
<link rel="stylesheet href="style.css">
<!head>

<!head>
<container>
Sample Text
.container {
padding: 20px;
margin: 0 auto;
max-width: 1200px;
contmelign: colter;

Table of Contents

Introduction

Purpose

The purpose of this document is to provide a **comprehensive guide to CSS (Cascading Style Sheets)**, covering all essential concepts, properties, and techniques required to design and style modern web pages. This document aims to serve both as a **learning resource for beginners** and a **reference manual for intermediate developers**, detailing the usage, values, and practical implementation of CSS properties.

Scope

This document covers a wide range of CSS topics, including **selectors, pseudo-classes, box model, positioning, display types, flexbox, grid, backgrounds, typography, transitions, animations, filters, and responsive design**. Each section is accompanied by **examples, visual demonstrations, and explanations**, allowing the reader to understand the effect of each property in real-world layouts. The guide is structured to progress from **basic concepts to advanced techniques**, ensuring a clear learning path.

Procedure

The procedure followed in this document involves:

1. **Introduction of concepts** – explaining each property and its purpose.
2. **Listing possible values** – providing all common and advanced options.
3. **Practical examples** – using colored, numbered boxes, or live code snippets to visually demonstrate each property's effect.
4. **Comparison and clarification** – highlighting differences between similar properties or values, such as align-items vs align-self.

Brief Problem

Many developers and students face **difficulty visualizing the effects of CSS properties**, especially when properties interact or override each other. The cascading nature, specificity, and layout differences between Flexbox and Grid often lead to confusion. This document addresses these challenges by providing **clear, complete, and visually demonstrative examples** that allow learners to see the impact of each CSS property in isolation and in combination.

CSS Intro & Syntax

CSS (Cascading Style Sheets) is the language used to style and visually format HTML documents. While **HTML** defines the structure and content of a webpage (headings, paragraphs, images, buttons), **CSS** controls how that content looks — colors, fonts, spacing, layout, responsiveness, and animations.

In simple terms:

- **HTML = What is on the page**
- **CSS = How it looks**

CSS allows developers to:

- Separate content from design
- Reuse styles across multiple pages
- Make websites responsive on different screen sizes
- Improve user experience and accessibility

CSS Syntax

A CSS rule is made of **three main parts**:

```
selector {
property: value;
}
```

- **Selector** → selects the HTML element to style
- **Property** → what you want to change
- **Value** → how you want to change it

Example:

```
p {
color: blue;
font-size: 16px;
}
```

This rule:

- Targets all <p> (paragraph) elements
- Changes the text color to blue
- Sets the font size to 16 pixels

Every property-value pair must end with a **semicolon (;)**.

Types of CSS

1. Inline CSS

Inline CSS is written **directly inside an HTML element** using the style attribute.

Example:

```
<p style="color: red; font-size: 18px;">This is inline CSS</p>
```

Characteristics:

- Applied to **one element only**
- Has the **highest priority**

Made by Youssef Amgad Elkhatib

2. Internal CSS

Internal CSS is written inside a <style> tag placed in the <head> section of an HTML document.

Example:

```
<!DOCTYPE html>
<html>
<head>
<style>
p {
color: green;
font-size: 16px;
}
</style>
</head>
<body>
<p>This is internal CSS</p>
</body>
</html>
```

Characteristics:

- Affects **only one HTML page**

3. External CSS

External CSS is written in a **separate .css file** and linked to HTML using the <link> tag.

CSS File (styles.css):

```
p {
color: blue;
font-size: 16px;
}
```

HTML File:

```
<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet" href="styles.css">
</head>
<body>
<p>This is external CSS</p>
</body>
</html>
```

Characteristics:

- Best practice
- Styles multiple pages at once

CSS Priority (Cascade Order)

When **multiple CSS rules** target the same element, CSS follows a priority order:

- **Inline CSS** (highest priority, performed last)
- **Internal CSS**
- **External CSS** (lowest priority, performed first)

- If two rules have the **same type**, the **last one written** wins
- More **specific selectors** override general ones
- **!important** overrides everything (use carefully)

Example:

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

CSS Selectors

CSS selectors define **which HTML elements** a style rule applies to. Mastering selectors is essential because they control precision, reusability, and priority in CSS.

---

1. Element (Type) Selector

Targets all elements of a specific HTML tag.

```
p {
color: blue;
}
<p>This paragraph is blue</p>
```

---

2. Class Selector

Targets elements with a specific class attribute. Classes are reusable.

```
.highlight {
background-color: yellow;
}
<p class="highlight">Highlighted text</p>
<div class="highlight">Highlighted div</div>
```

---

3. ID Selector

Targets **one unique element** using the id attribute.

```
#main-title {
font-size: 32px;
}
<h1 id="main-title">Main Title</h1>
```

Rules:

- IDs must be unique
- Higher priority than classes

Made by Youssef Amgad Elkhatib

4. Universal Selector

Targets **all elements** on the page.

```
* {
margin: 0;
padding: 0;
}
```

---

5. Grouping Selector

Applies the same style to multiple selectors.

```
h1, h2, h3 {
font-family: Arial;
}
```

---

6. Attribute Selectors

Attribute selectors target elements based on the presence or value of an attribute.

6.1 Attribute Exists Selector

```
input[required] {
border: 2px solid red;
}
```

Targets inputs that have the required attribute.

6.2 Exact Attribute Value Selector

```
input[type="password"] {
background-color: lightgray;
}
```

6.3 Attribute Contains Selector

```
a[href*="google"] {
color: green;
}
```

Targets links that contain the word "google" in the URL.

6.4 Attribute Starts With

```
a[href^="https"] {
color: blue;
}
```

6.5 Attribute Ends With

```
a[href$=".pdf"] {
color: red;
}
```

---

7. Combinator Selectors

Combinators define relationships between elements.

7.1 Descendant Selector (space)

Targets elements inside another element.

```
div p {
color: purple;
}
```

7.2 Child Selector (>)

Targets **direct children only**.

```
div > p {
color: orange;
}
```

7.3 Adjacent Sibling Selector (+)

Targets the **next immediate sibling**.

```
h1 + p {
color: red;
}
```

7.4 General Sibling Selector (~)

Targets all siblings after an element.

```
h1 ~ p {
color: gray;
}
```

---

8. Pseudo-Classes

Pseudo-classes define a **special state** of an element.

Pseudo-class	Applies to	Description
:link	<a>	Unvisited links
:visited	<a>	Visited links
:hover	All	When user hovers with mouse
:active	All	While element is being clicked
:focus	Inputs, buttons	When element is focused
:focus-visible	Inputs	Focus shown only for keyboard navigation
:focus-within	Containers	When any child has focus
:target	All	Element targeted by URL fragment (#id)
:empty	All	Elements with no children or text
:not(selector)	All	Excludes specific selector
:root	html	The root element of the document
:default	Form elements	Default option (radio/checkbox)
:checked	Checkbox, radio	Checked inputs
:disabled	Form elements	Disabled elements
:enabled	Form elements	Enabled elements
:readonly	Inputs	Read-only inputs

:required	Inputs	Required inputs
:optional	Inputs	Inputs without required
:valid	Inputs	Input with valid value
:invalid	Inputs	Input with invalid value
:in-range	Inputs	Value within allowed range
:out-of-range	Inputs	Value outside allowed range
:placeholder-shown	Inputs	Placeholder text is visible
:first-child	All	First child of parent
:last-child	All	Last child of parent
:only-child	All	Element is the only child
:nth-child(n)	All	Matches nth child
:nth-last-child(n)	All	Matches nth child from end
:first-of-type	All	First element of its type
:last-of-type	All	Last element of its type
:only-of-type	All	Only element of its type
:nth-of-type(n)	All	nth element of its type
:nth-last-of-type(n)	All	nth element of its type from end
:is(selector)	All	Matches any selector in list
:where(selector)	All	Like :is() but zero specificity
:has(selector)	Containers	Parent selector (modern CSS)
:lang()	All	Elements with specific language

8. Pseudo-Elements

Pseudo-elements style **specific parts** of an element.

8.1 First Letter

```
p::first-letter {
font-size: 32px;
}
```

8.2 First Line

```
p::first-line {
font-weight: bold;
}
```

8.3 Before & After

```
h1::before {
content: " 🔥 ";
}

h1::after {
content: " 🚀 ";
}
```

---

10. Selector Priority (Specificity)

From lowest to highest priority:

1. Element selectors
2. Class selectors
3. ID selectors
4. Inline styles

Fonts & Text

This section covers **EVERY core font and text-related CSS property**, what it does, its possible values, and a **full visual HTML + CSS example** for each.

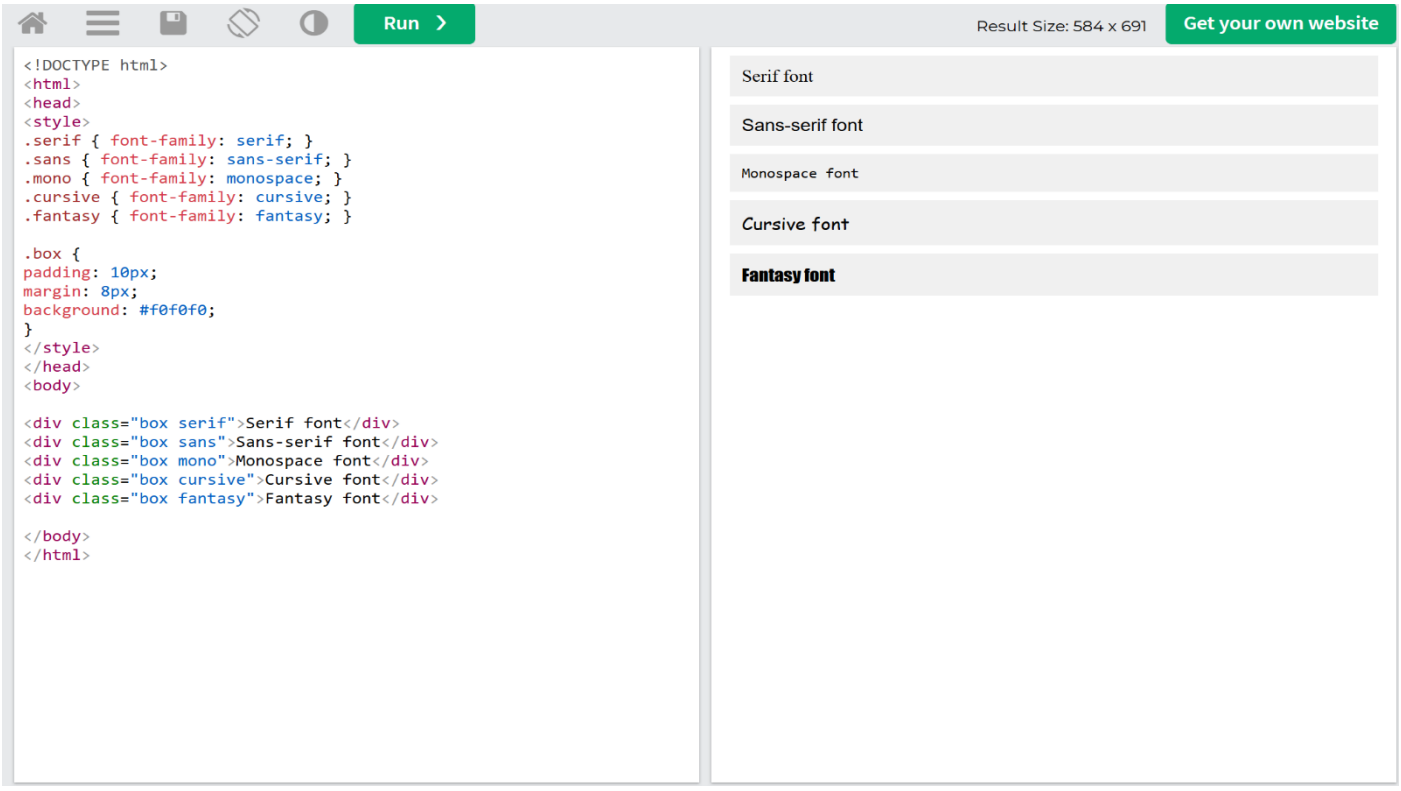
FONT-FAMILY

What it controls

Which font is used to render text.

Common values

- Specific fonts: Arial, Times New Roman, Roboto
- Generic families:
    - serif
    - sans-serif
    - monospace
    - cursive
    - fantasy



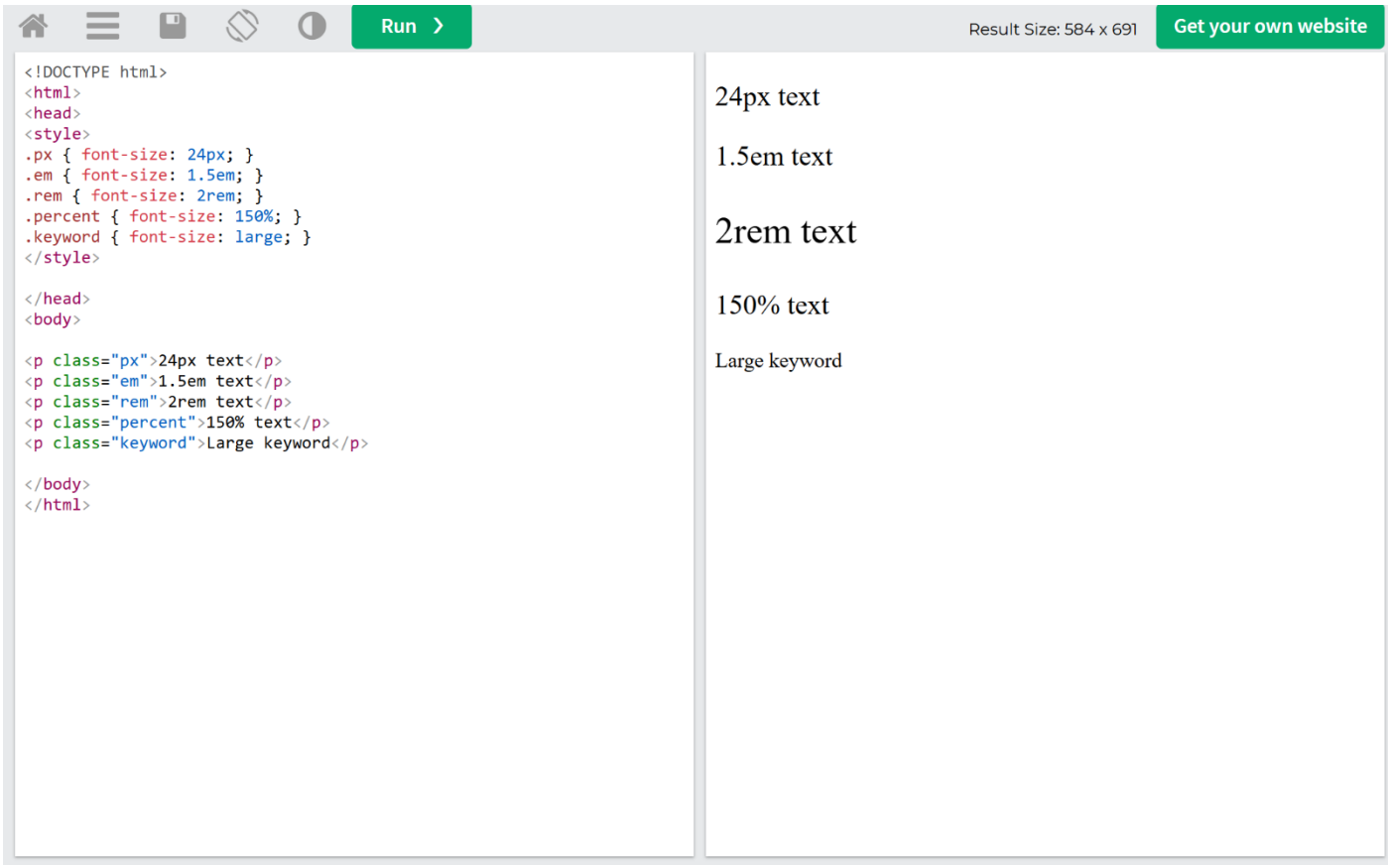
FONT-SIZE

What it controls

Size of the text.

Values

- px
- em
- rem
- %
- Keywords: small, medium, large



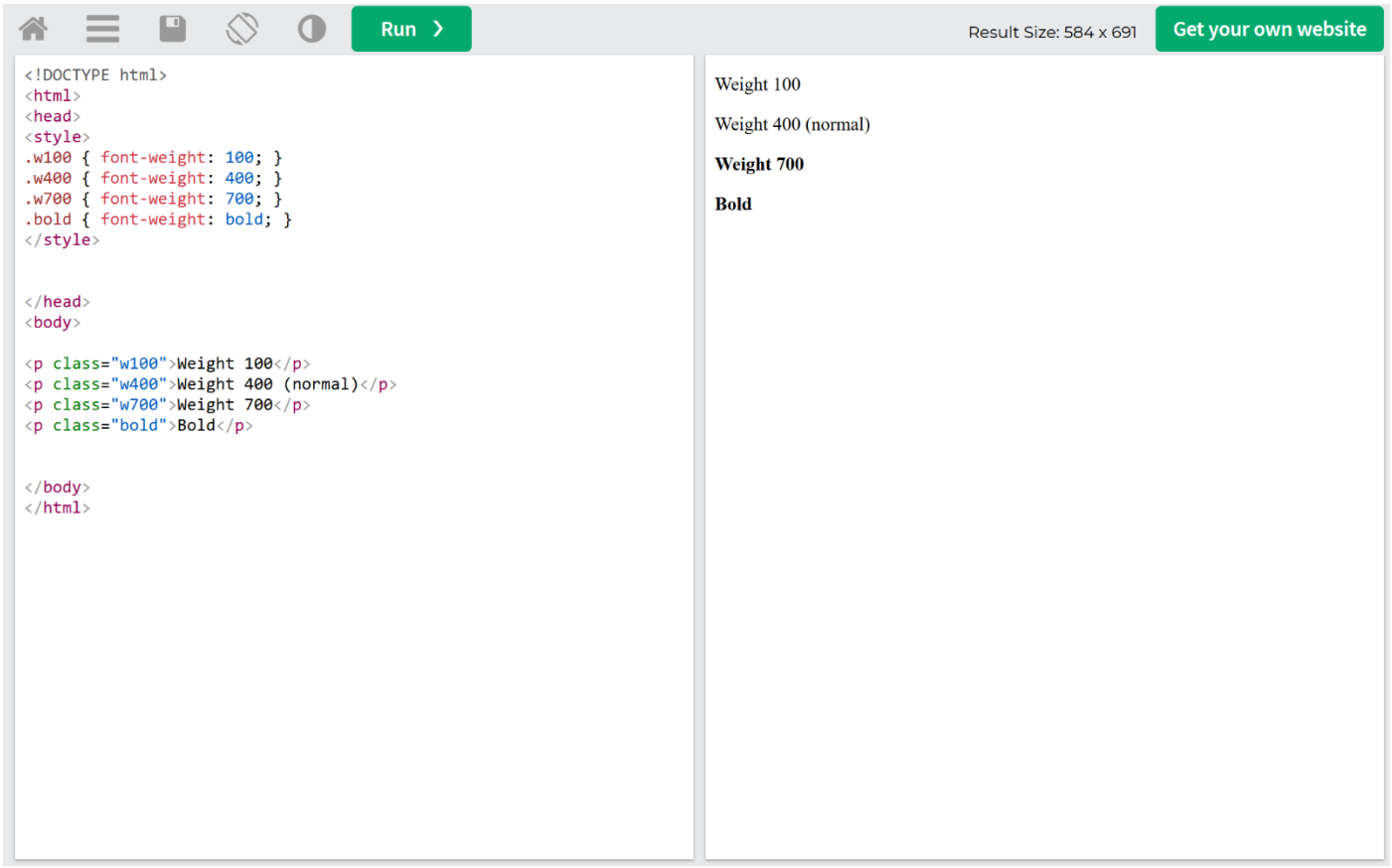
FONT-WEIGHT

What it controls

Thickness (boldness) of text.

Values

- Keywords: normal, bold, lighter, bolder
- Numbers: 100 → 900



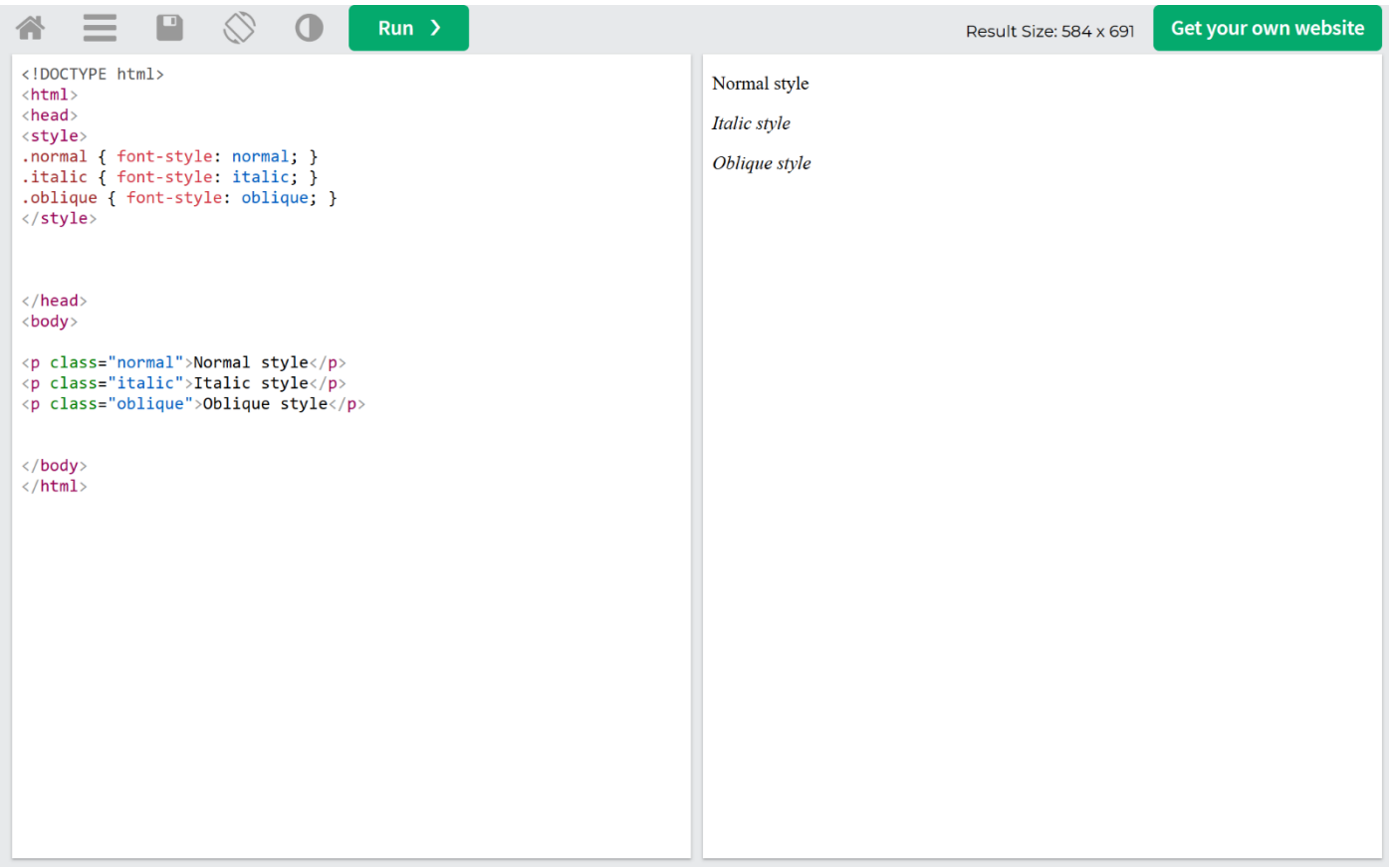
FONT-STYLE

What it controls

Italic or normal text.

Values

- normal
- italic
- oblique



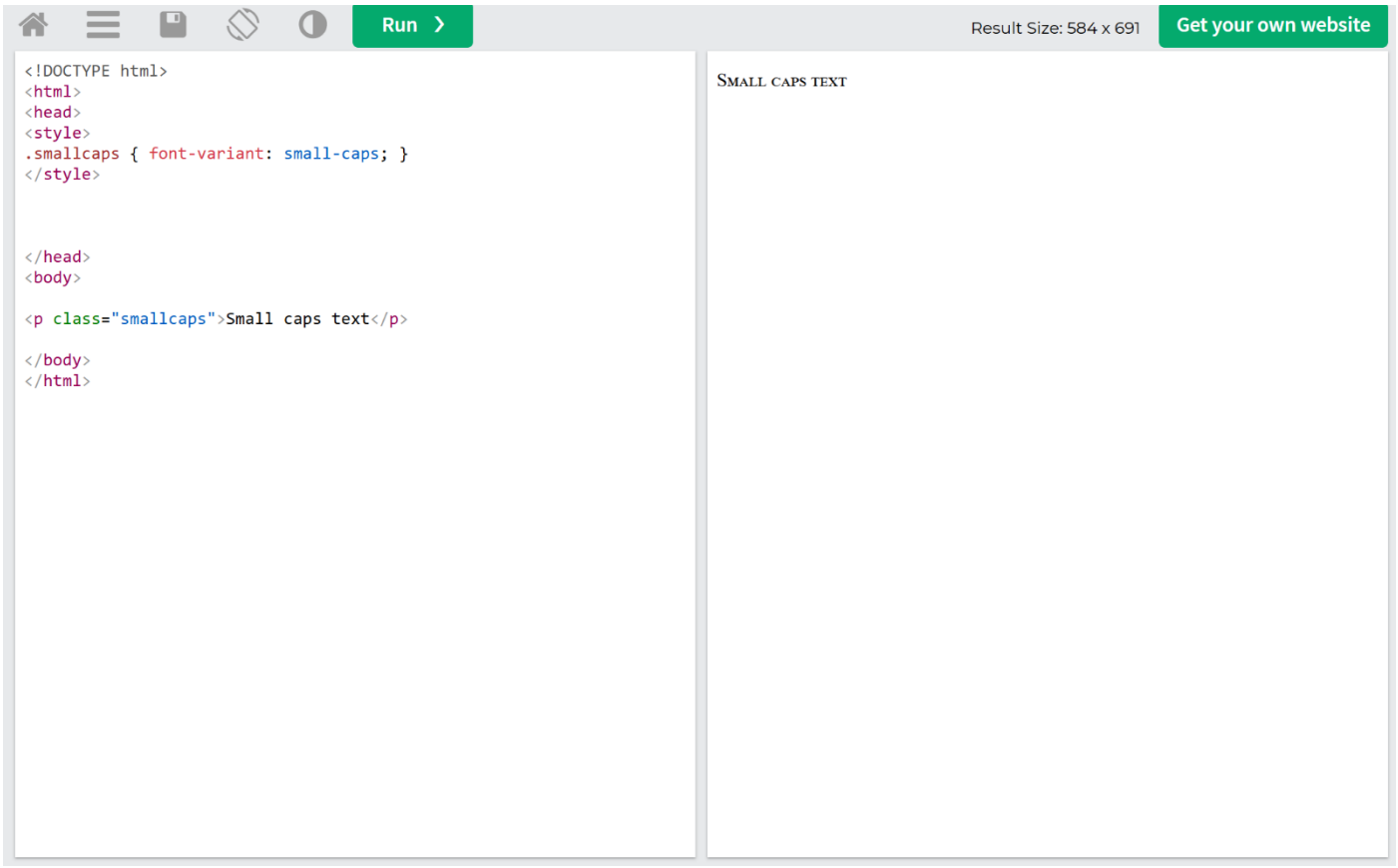
FONT-VARIANT

What it controls

Alternative glyphs like small caps.

Values

- normal
- small-caps



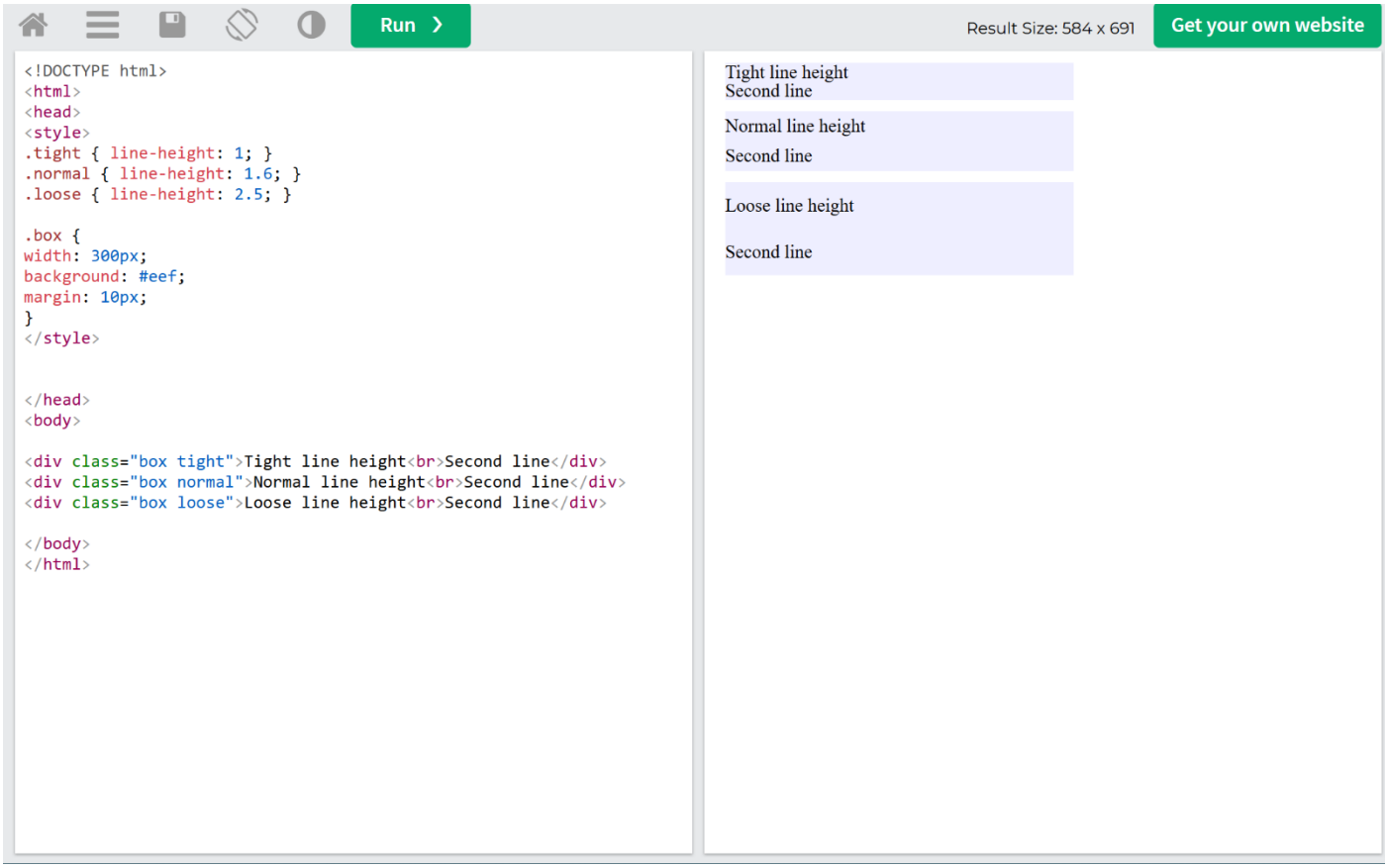
LINE-HEIGHT

What it controls

Vertical spacing between lines of text.

Values

- Unitless number
- px
- em
- normal



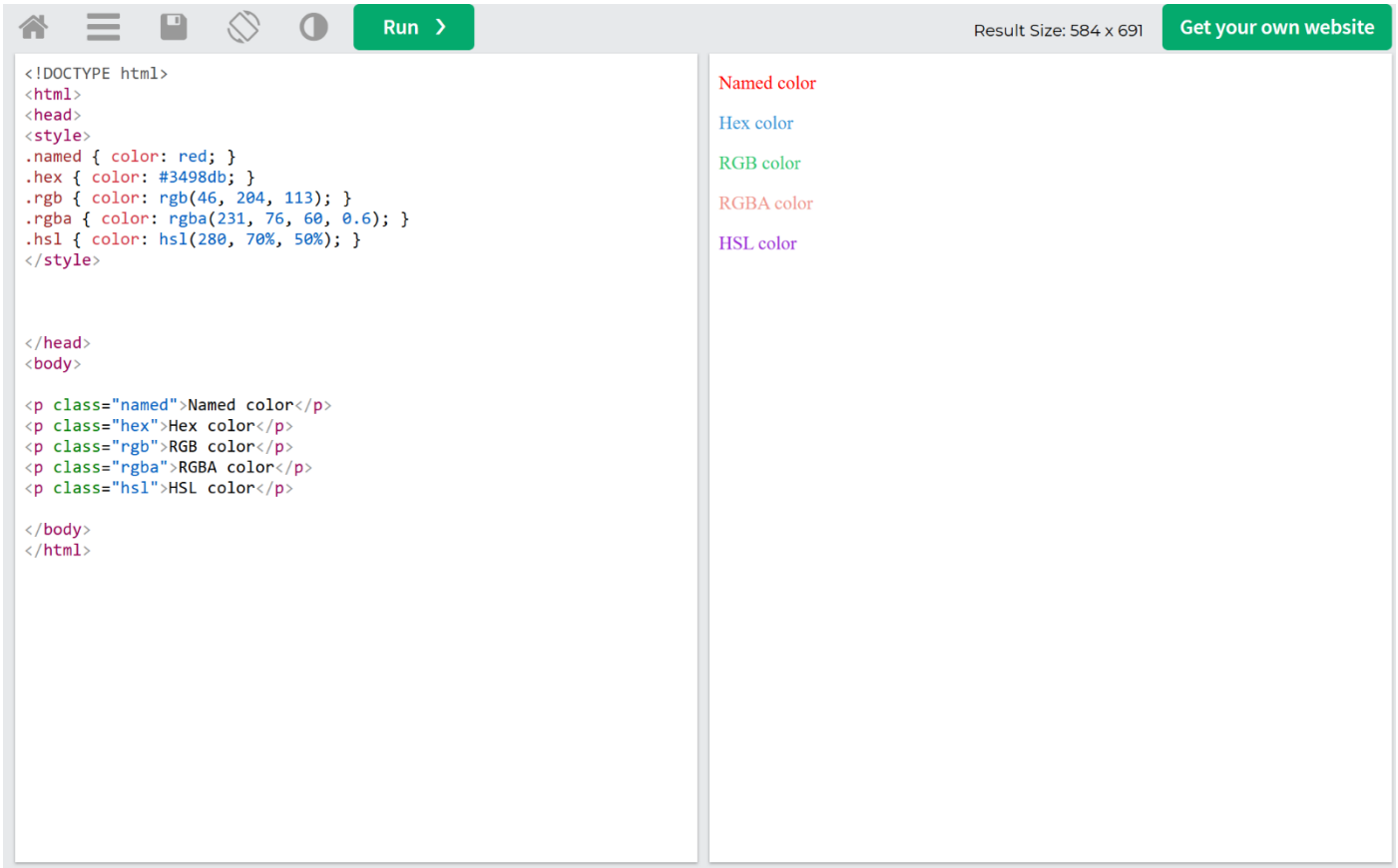
COLOR

What it controls

Text color.

Values

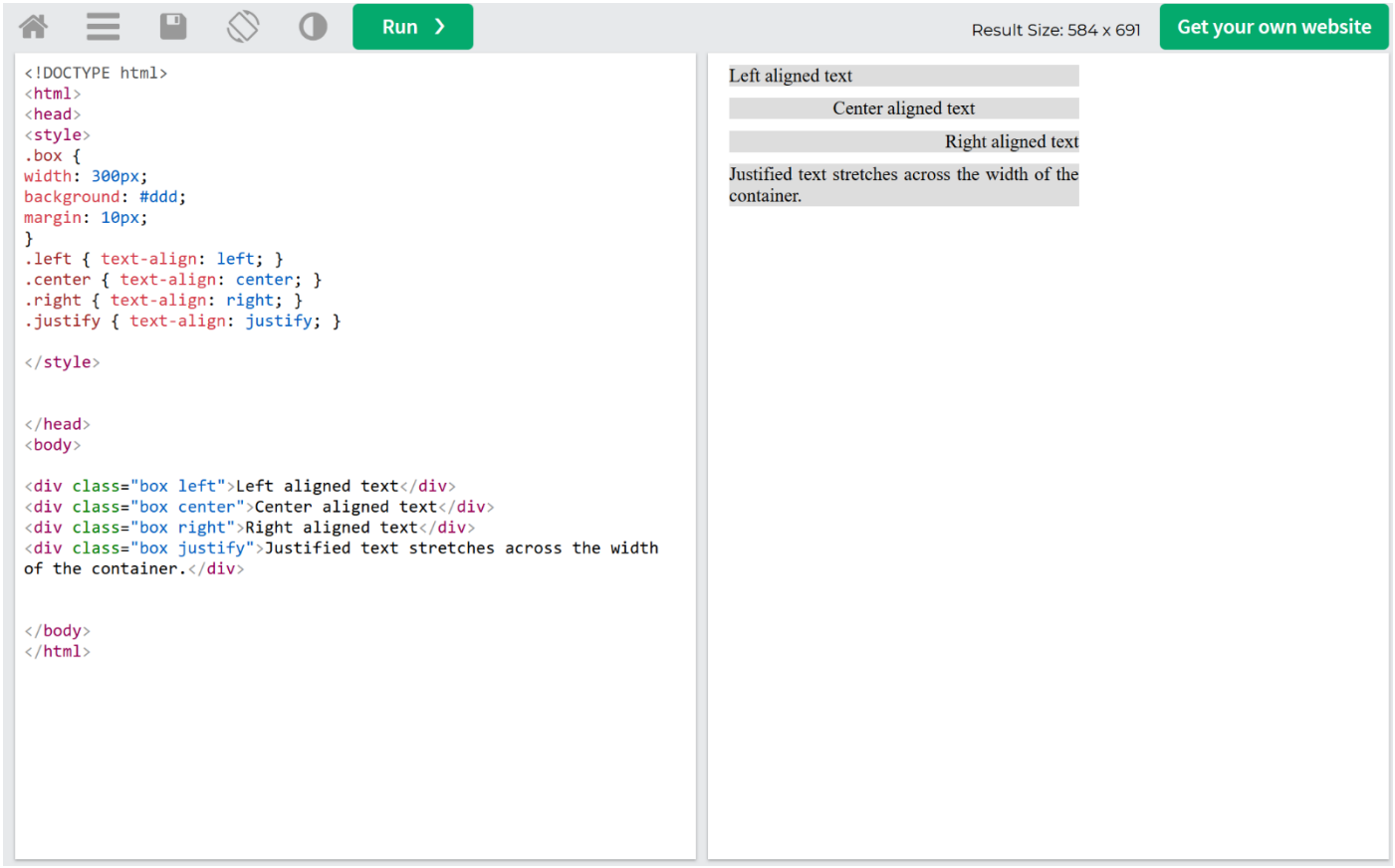
- Named colors
- hex
- rgb()
- rgba()
- hsl() (Hue, saturation, lightness)
- hsla() (Hue, saturation, lightness, alpha/opacity)



TEXT-ALIGN

Values

- left
- right
- center
- justify



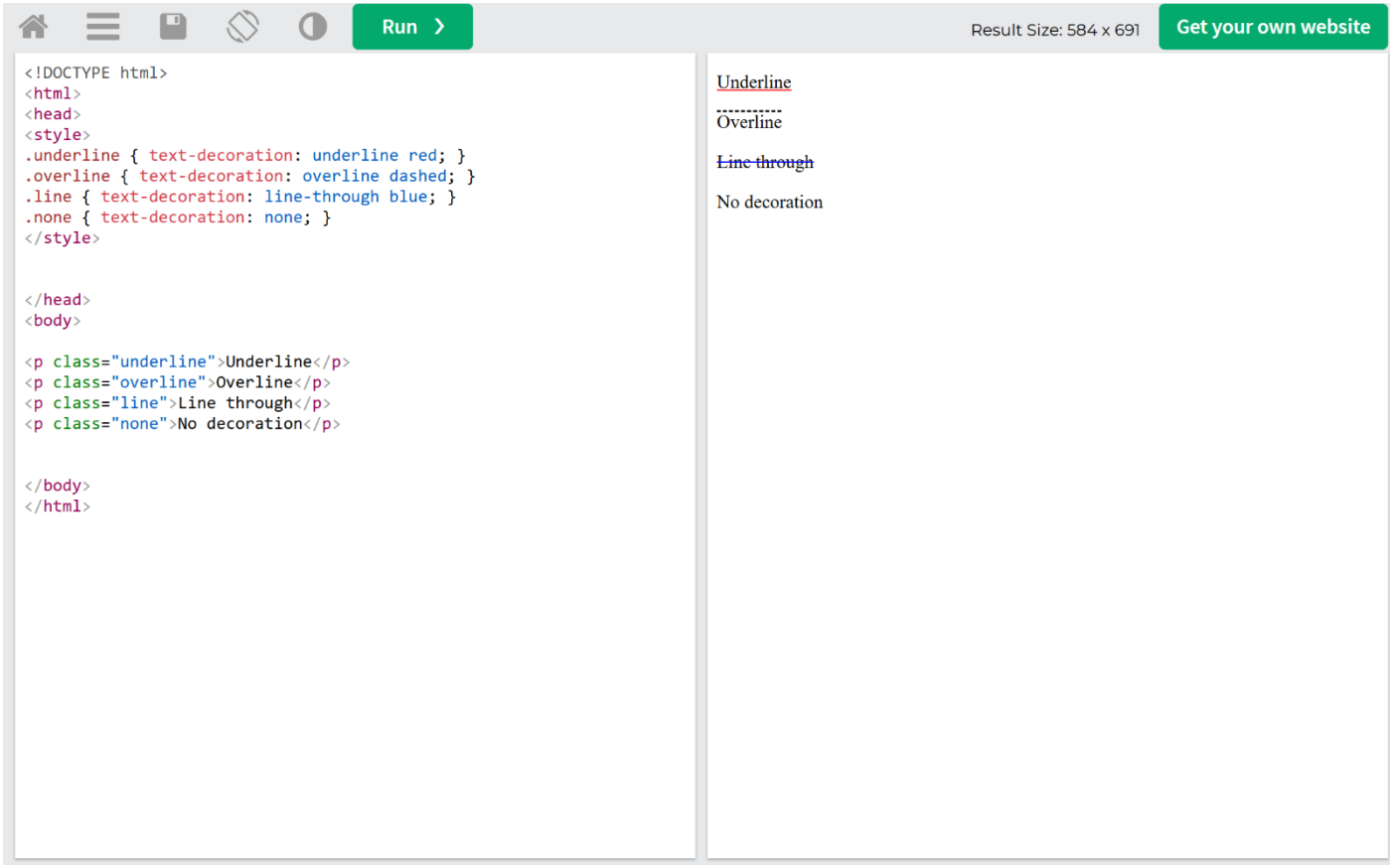
TEXT-DECORATION

What it controls

Lines on text.

Values

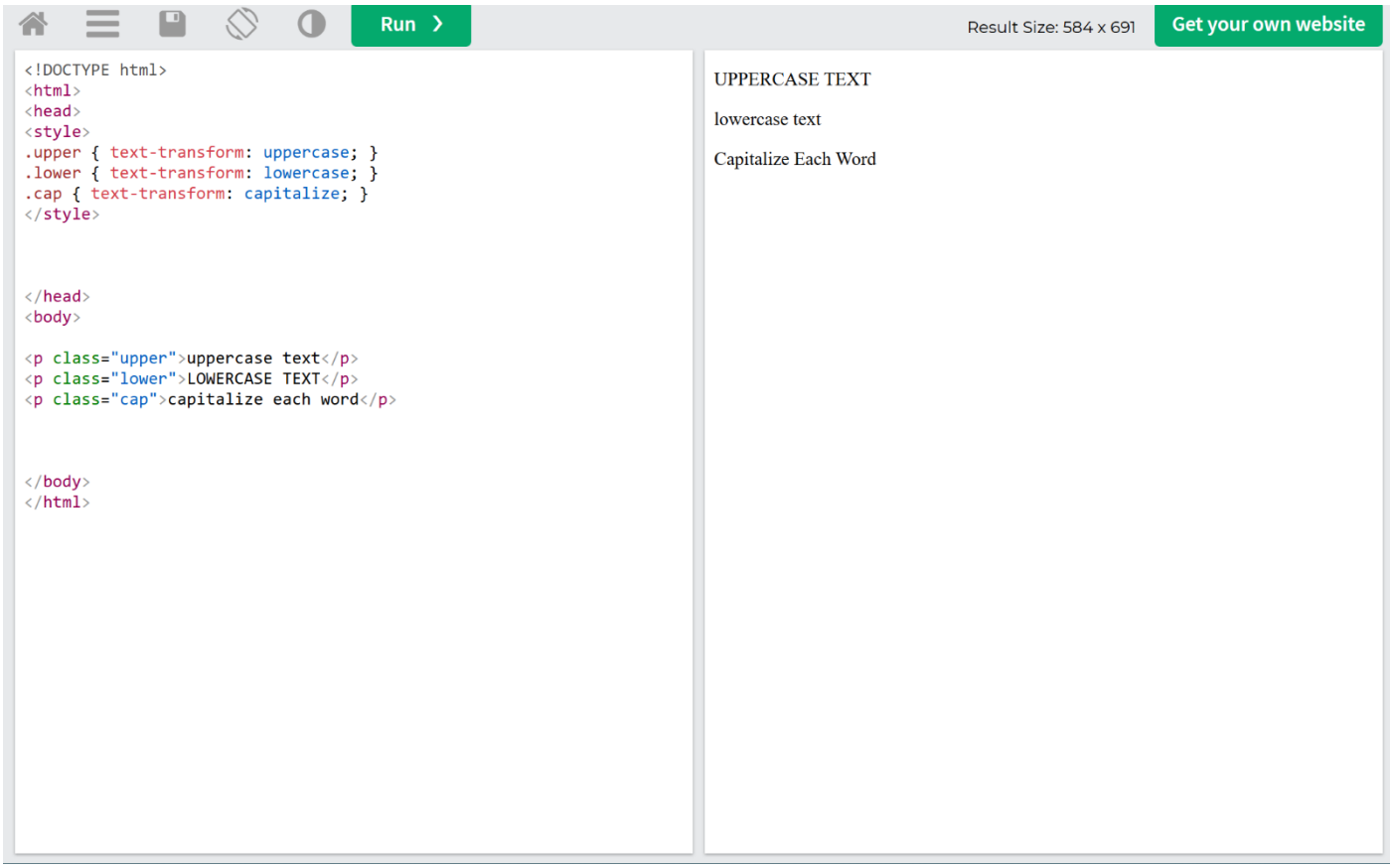
- underline
- overline
- line-through
- none



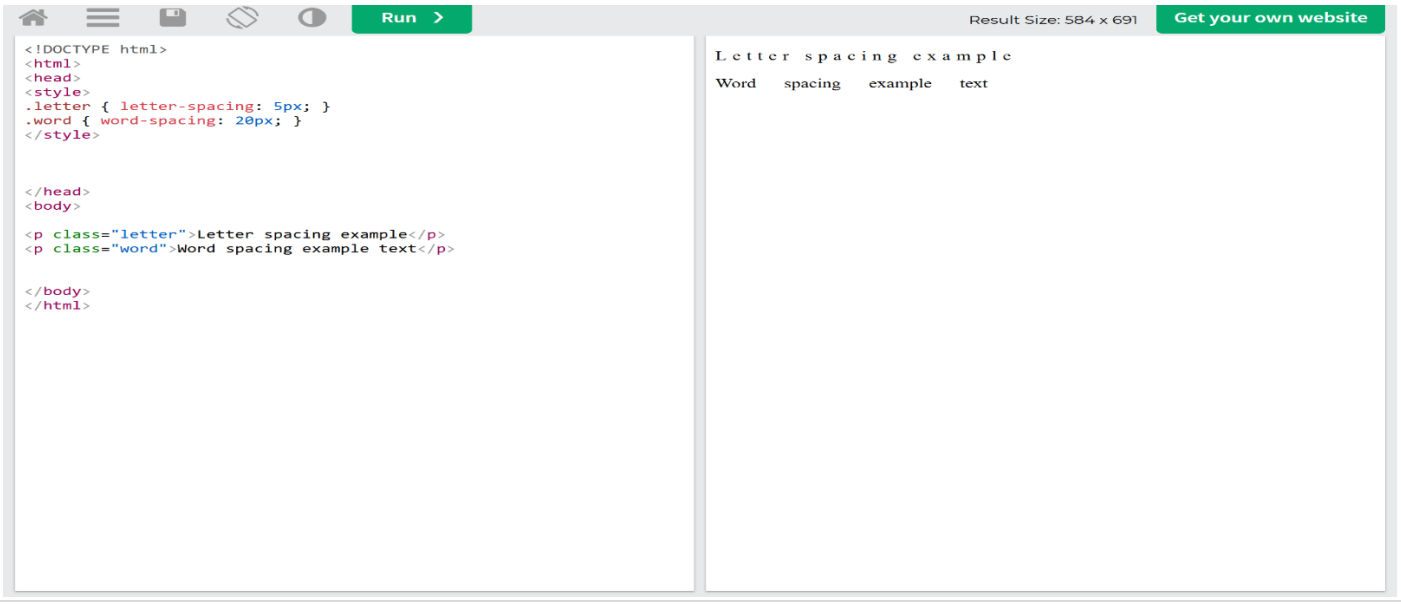
TEXT-TRANSFORM

Values

- uppercase
- lowercase
- capitalize



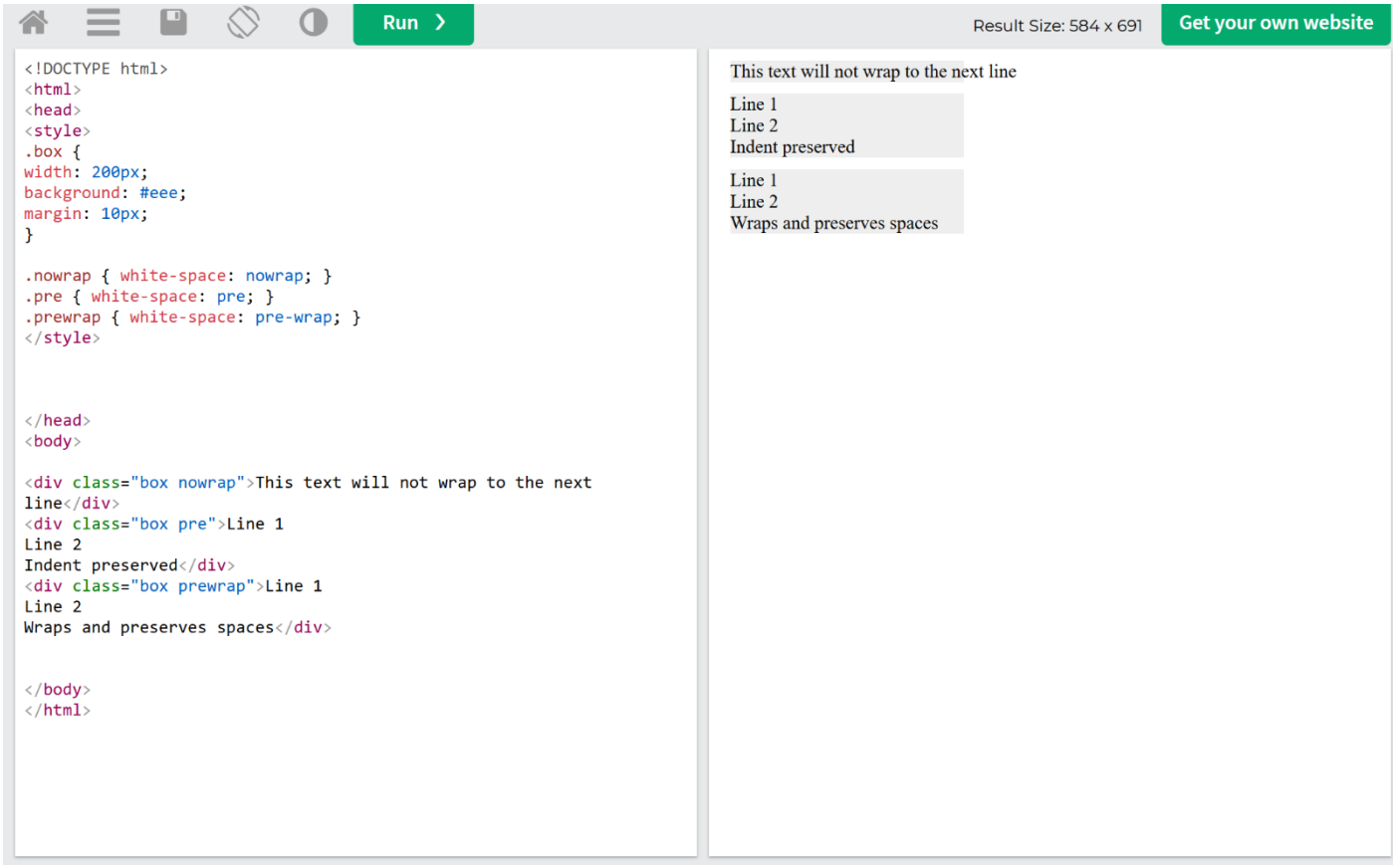
LETTER-SPACING & WORD-SPACING



WHITE-SPACE

Values

- normal
- nowrap
- pre
- pre-line
- pre-wrap

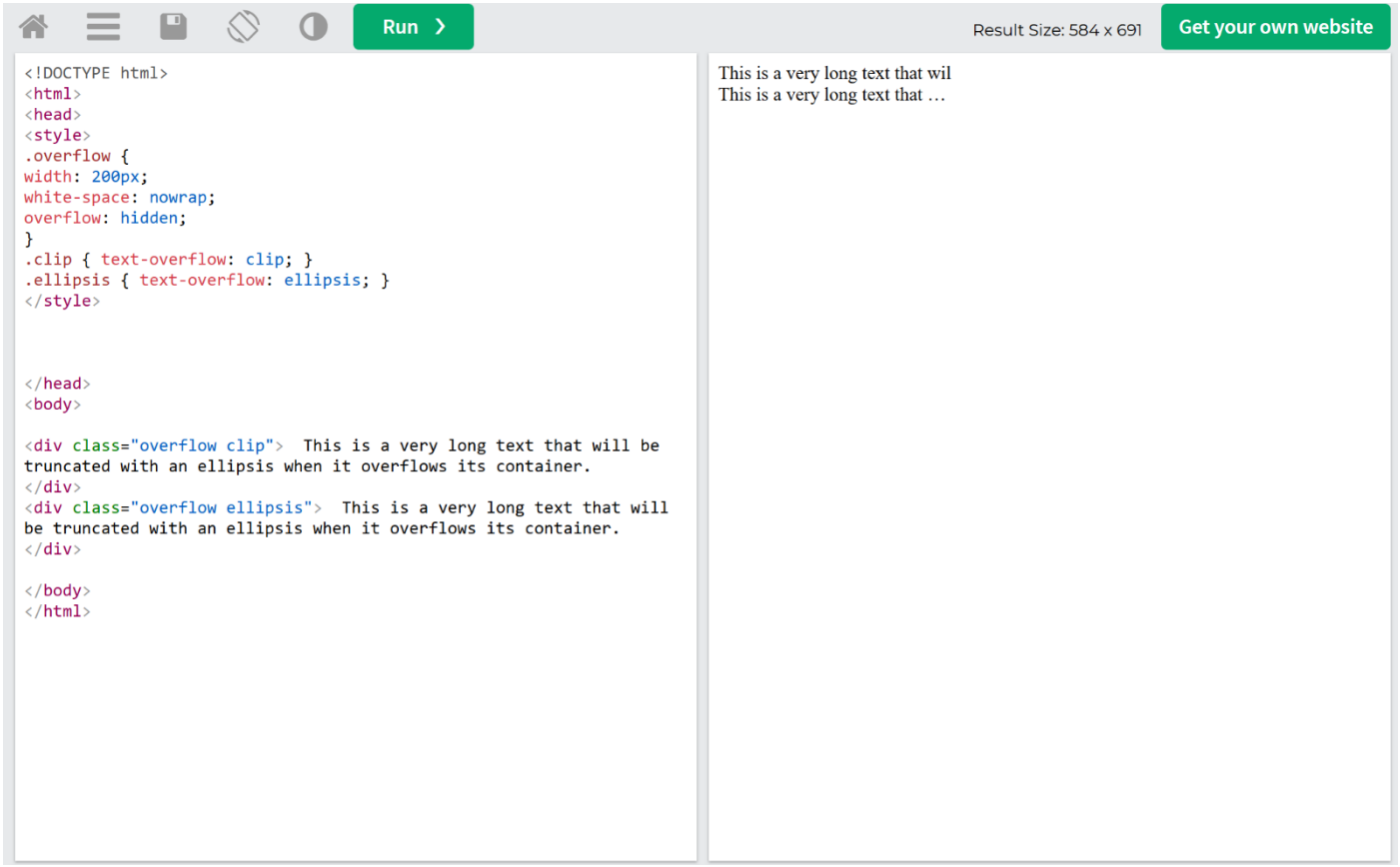


TEXT-OVERFLOW

Text overflow refers to how content is handled when it is too large to fit within its container box.

Values

- clip
- ellipsis



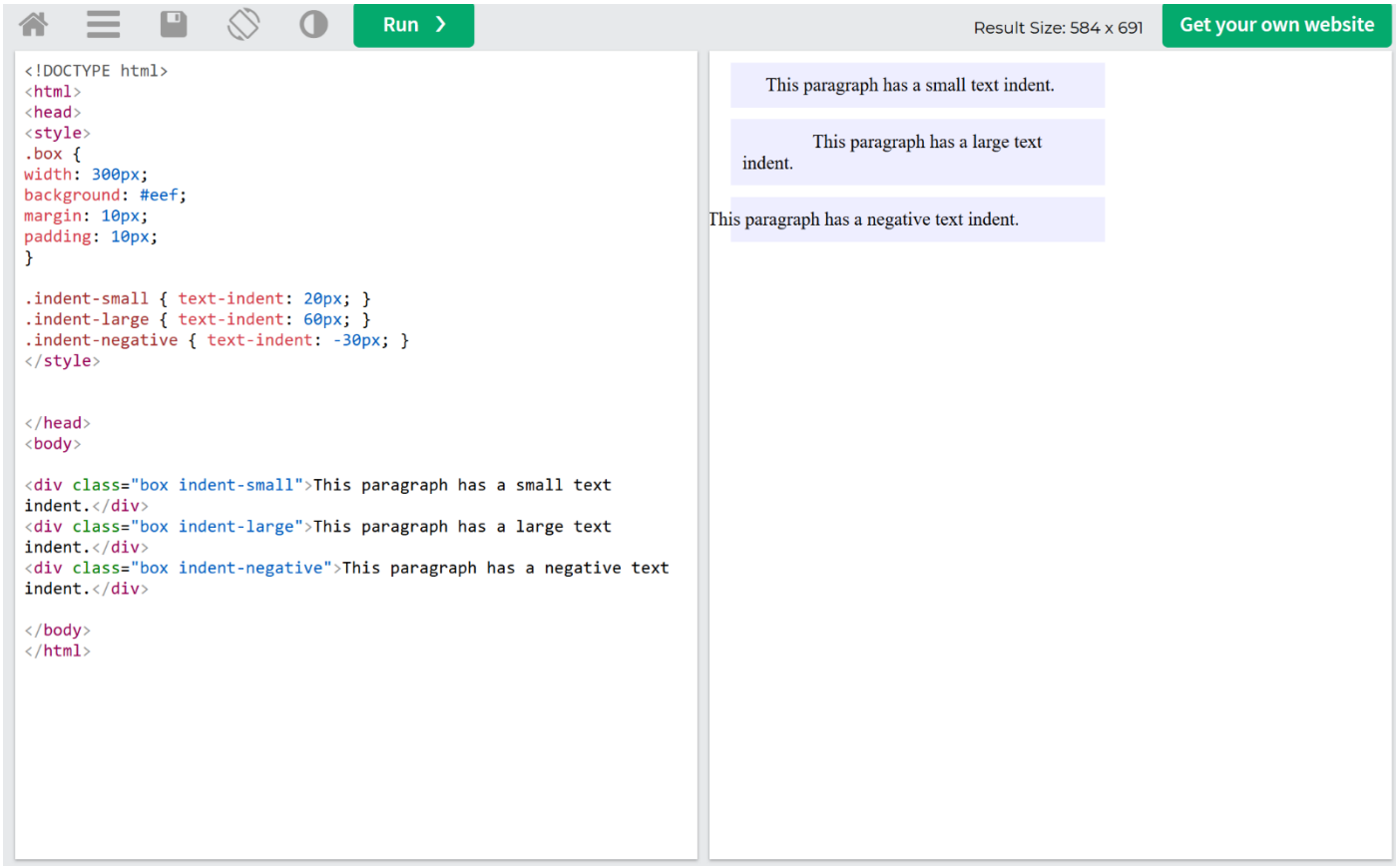
TEXT-INDENT

What it controls

Moves the **first line** of text inward from the left (or right in RTL).

Common values

- px, em, rem, %
- Negative values allowed



TEXT-SHADOW

What it controls

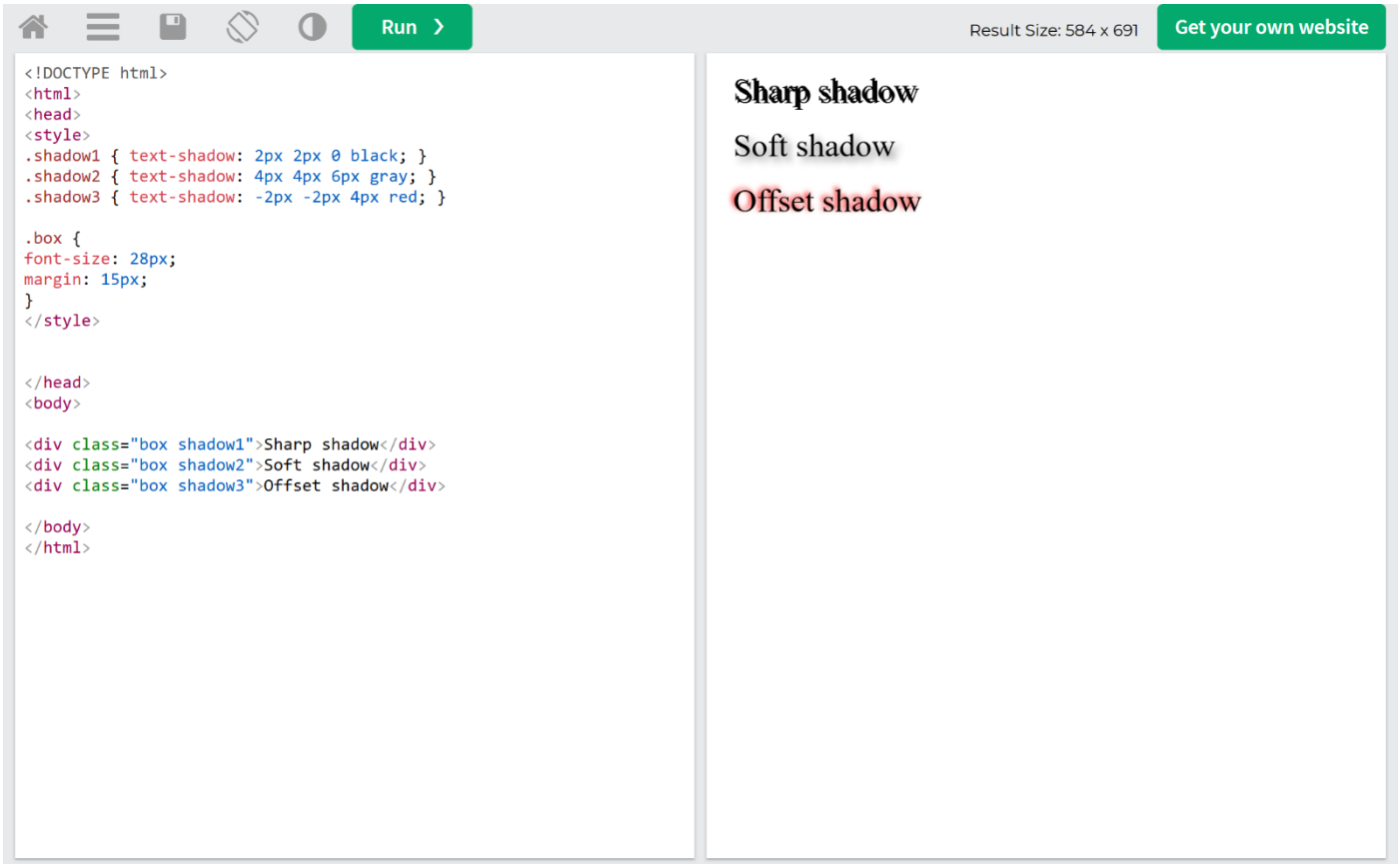
Adds shadow effects to text.

Syntax

text-shadow: x-offset y-offset blur color;

Values

- Horizontal offset
- Vertical offset
- Blur radius (optional)
- Color



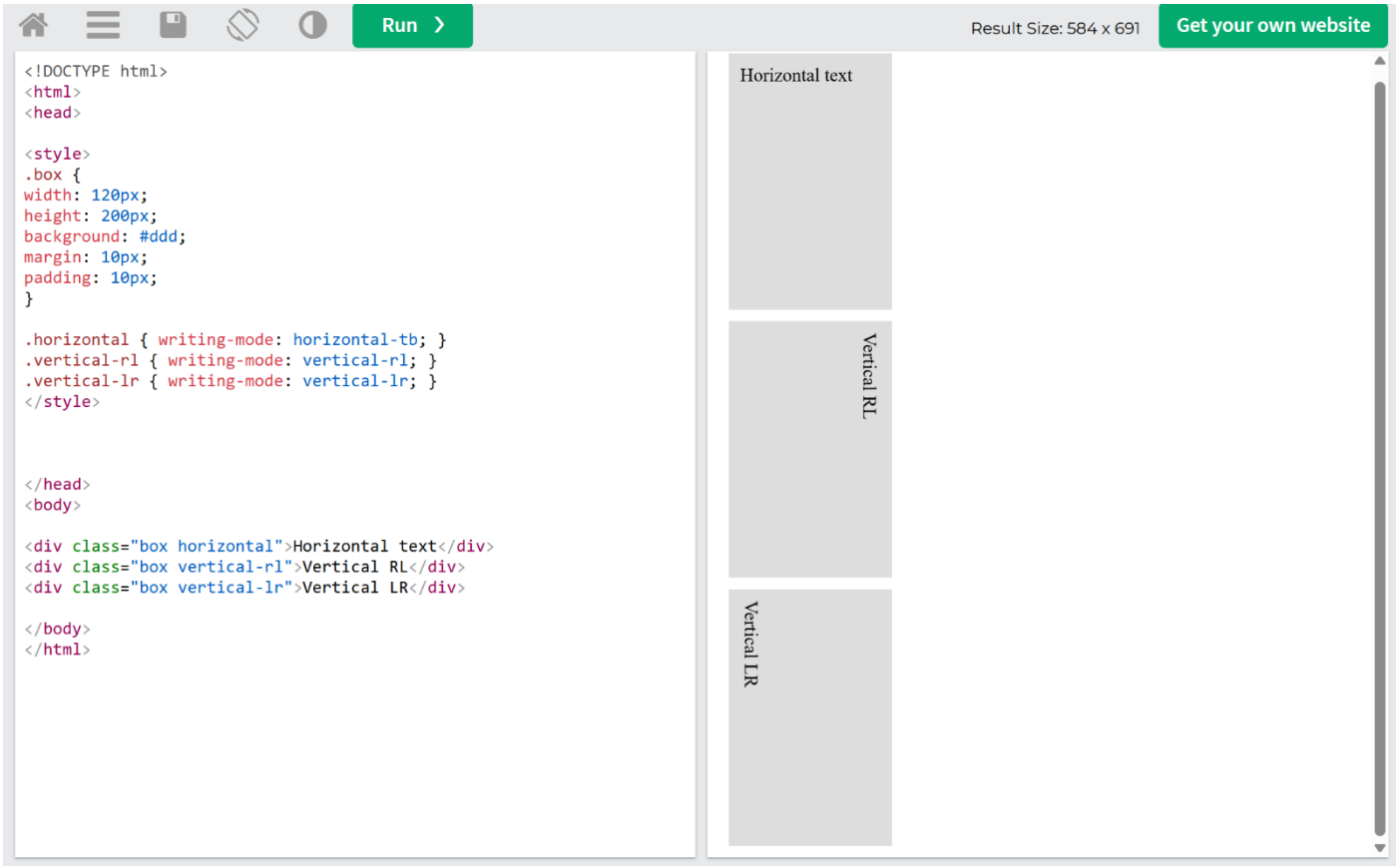
WRITING-MODE

What it controls

Sets **text flow direction** (horizontal or vertical).

Common values

- horizontal-tb (default)
- vertical-rl
- vertical-lr



TEXT-ORIENTATION

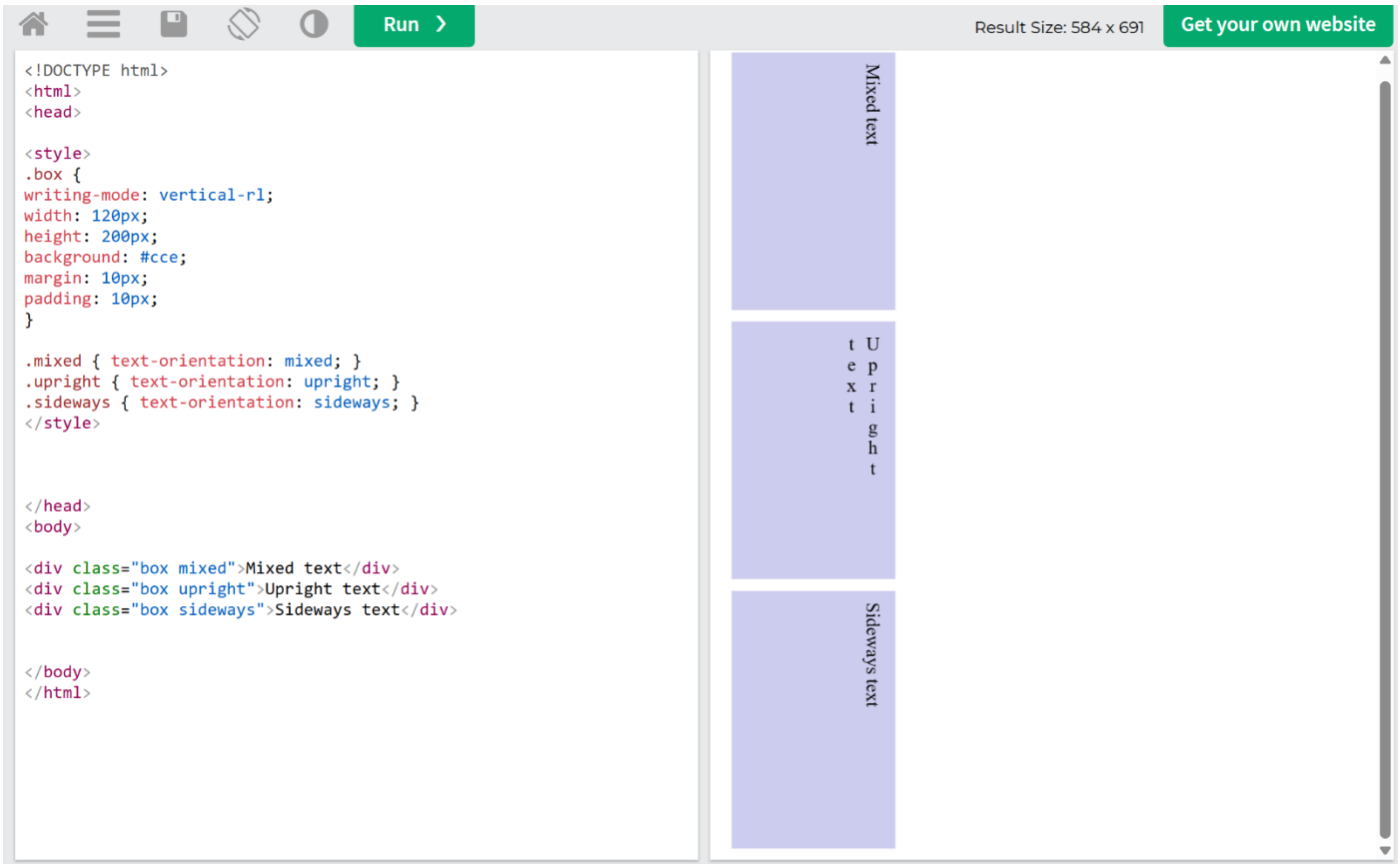
What it controls

How characters are oriented **inside vertical writing modes**.

⚠️ Has effect **only when writing-mode is vertical**.

Values

- mixed (default)
- upright
- sideways



Web Fonts

WHAT ARE WEB FONTS?

By default, browsers can only use **fonts installed on the user's system** (Arial, Times, etc.).

Web fonts allow you to:

- Use **any custom font**
- Ensure **consistent design** across devices
- Control typography precisely

They are downloaded by the browser just like images or CSS.

@FONT-FACE

What it does

Defines a **custom font** and gives it a name that CSS can use.

Once declared, the font behaves like any system font.

BASIC @FONT-FACE SYNTAX

```
@font-face {
font-family: "MyFont";
src: url("MyFont.woff2");
}
```

- font-family → name you choose
- src → font file location

FONT FORMATS (IMPORTANT)

Common formats:

- woff2 ✅ best (smallest, fastest)
- woff
- ttf
- otf

Browsers try sources **top to bottom**.

MULTIPLE FONT WEIGHTS

Each weight **must be declared separately**.

```
@font-face {
font-family: "MyFont";
src: url("MyFont-Regular.woff2");
font-weight: 400;
}

@font-face {
font-family: "MyFont";
src: url("MyFont-Bold.woff2");
font-weight: 700;
}
p { font-family: "MyFont"; }
strong { font-weight: 700; }
```

MULTIPLE STYLES (ITALIC)

```
@font-face {
font-family: "MyFont";
src: url("MyFont-Italic.woff2");
font-style: italic;
}
```

Browser automatically selects the correct file.

FONT-DISPLAY (VERY IMPORTANT)

Controls **what happens while the font is loading**.

Values

- swap ✅ recommended
- block
- fallback
- optional

```
@font-face {
font-family: "MyFont";
src: url("MyFont.woff2");
font-display: swap;
}
```

🧠 swap shows fallback text first, then swaps when font loads.

USING GOOGLE FONTS

Option 1 — <link> (most common)

```
<link href="https://fonts.googleapis.com/css2?family=Roboto:wght@400;700&display=swap"
rel="stylesheet">
body {
font-family: "Roboto", sans-serif;
}
```

Option 2 — @import (not recommended -> slower than link)

```
@import url('https://fonts.googleapis.com/css2?family=Roboto&display=swap');
```

CSS Box Model

The **CSS Box Model** describes how every HTML element is structured and how space is calculated around it. Every element is treated as a rectangular box made up of **four layers**, from inside to outside:

Content → Padding → Border → Margin

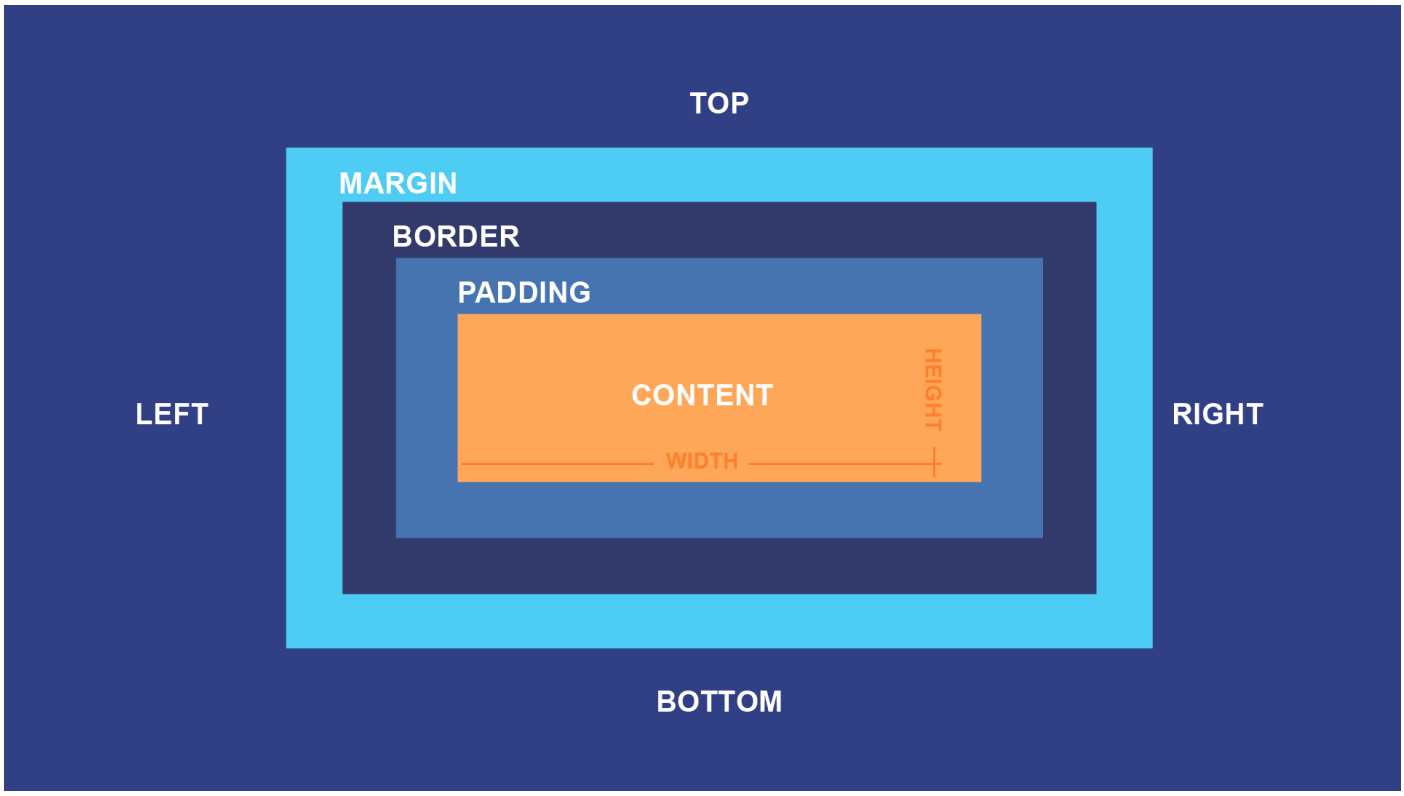
Understanding the box model is **critical** for layout, spacing, alignment, and responsive design.

Content

The **content area** holds the actual content of the element such as text, images, or other elements.

Properties that affect content size:

- width
- height
- min-width, max-width
- min-height, max-height



Padding

Padding is the **space between the content and the border**. It increases the internal spacing of an element.

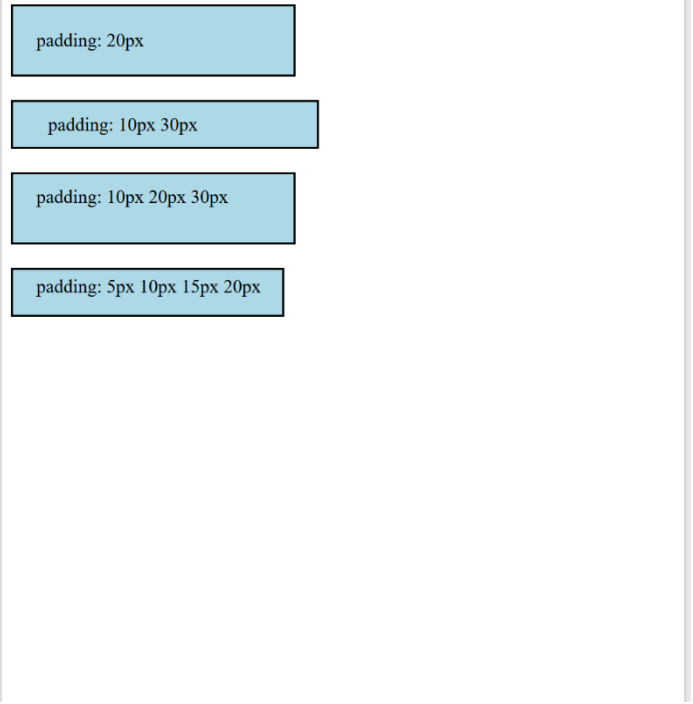
Padding properties:

- padding-top
- padding-right
- padding-bottom
- padding-left
- padding (shorthand)

Padding values:

- px, em, rem, %

```
<!DOCTYPE html>
<html>
<head>
<style>
.box {
    width: 200px;
    border: 2px solid black;
    background: lightblue;
    margin-bottom: 20px;
}


.pad-1 { padding: 20px; }
.pad-2 { padding: 10px 30px; }
.pad-3 { padding: 10px 20px 30px; }
.pad-4 { padding: 5px 10px 15px 20px; }
</style>
</head>
<body>


<div class="box pad-1">padding: 20px</div>
<div class="box pad-2">padding: 10px 30px</div>
<div class="box pad-3">padding: 10px 20px 30px</div>
<div class="box pad-4">padding: 5px 10px 15px 20px</div>


</body>
</html>
```



How values are applied:

- 1 value → all sides
- 2 values → top/bottom | left/right
- 3 values → top | left/right | bottom
- 4 values → top | right | bottom | left

Border

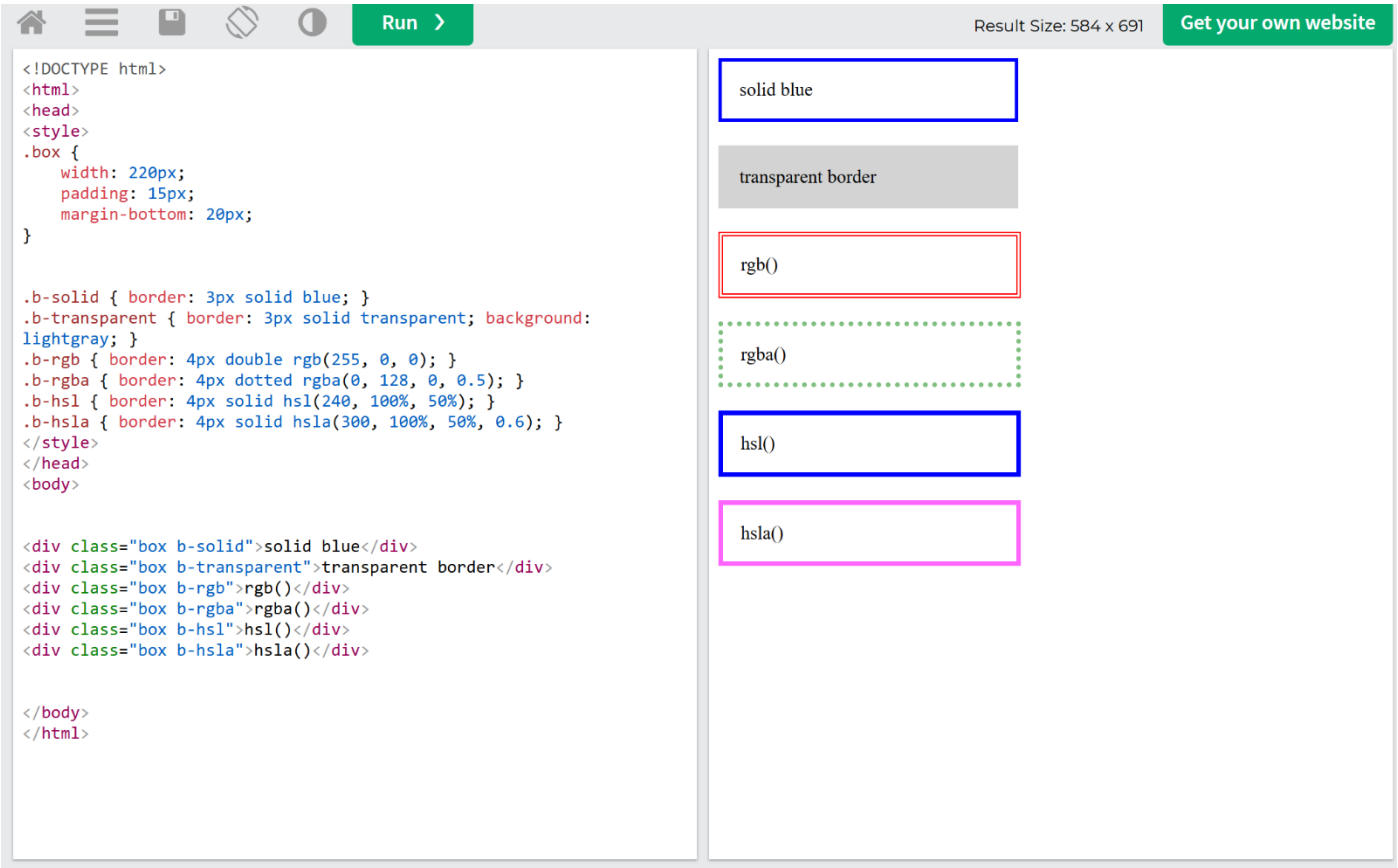
The border wraps around the padding and content.

Border properties:

- border-width
- border-style
- border-color
- border (shorthand)

Common border styles:

- solid
- dashed
- dotted
- double
- none



Border Radius

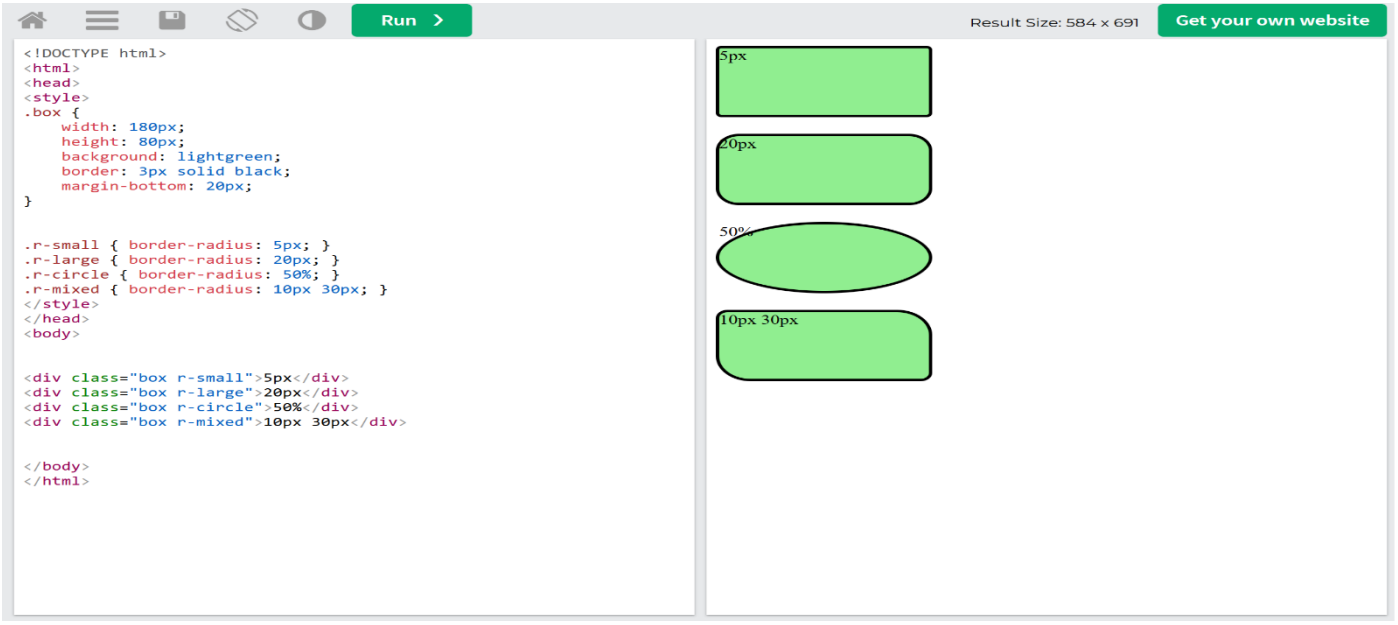
border-radius controls **how rounded the corners** of an element are. It can create anything from slightly rounded rectangles to perfect circles.

Properties & syntax:

- border-radius
- border-top-left-radius
- border-top-right-radius
- border-bottom-right-radius
- border-bottom-left-radius

Possible values:

- Lengths: px, em, rem
- Percentages: %
- Multiple values (1–4)



How values work:

- 1 value → all corners
- 2 values → top-left & bottom-right | top-right & bottom-left
- 3 values -> top-left | top-right & bottom-right | bottom-left
- 4 values → top-left | top-right | bottom-right | bottom-left

Border Image

border-image allows you to **use an image or gradient as the border** instead of a solid color.

Required setup:

- A visible border width
- border-style (or transparent border)

Main Properties

- **border-image-source**: Specifies the path (URL) to the image you want to use as the border.
- **border-image-slice**: Tells the browser how to cut the image into nine regions (corners, edges, and center) so it can be stretched or repeated.
- **border-image-width**: Defines how thick the image border should be, which can be different from the standard border-width.
- **border-image-outset**: Determines how far the border image area extends beyond the element's actual border box.
- **border-image-repeat**: Controls whether the image slices should be stretched to fit the area, repeated (tiled), or rounded to fill the space.
- **border-image (shorthand)**: A single property that allows you to set all the above values at once in one line.

Common values:

- Images (url())
- Gradients (linear-gradient, radial-gradient)
- Slice numbers or percentages
- Repeat options: stretch, repeat, round, space

Important notes:

- border-image replaces border-color
- Border radius may not work perfectly with images
- Mostly used for decorative UI elements

Run ›

Result Size: 584 x 691

Get your own website

```
<!DOCTYPE html>
<html>
<head>
<style>
.box {
width: 260px;
padding: 20px;
border: 10px solid transparent;
border-image: linear-gradient(45deg, red, blue, purple) 1;
}
</style>
</head>
<body>

<div class="box">Gradient Border Image</div>

</body>
</html>
```

Margin

Margin is the **outermost layer**. It creates space **outside** the element, separating it from other elements.

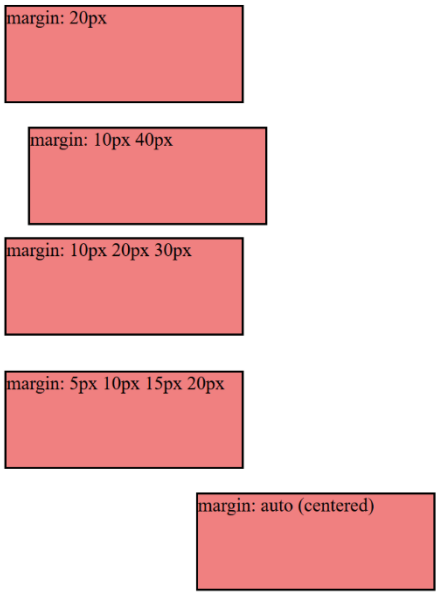
Margin properties:

- margin-top
- margin-right
- margin-bottom
- margin-left
- margin (shorthand)

Margin values:

- px, em, rem, %
- auto

```
<!DOCTYPE html>
<html>
<head>
<style>
.box {
    width: 200px;
    height: 80px;
    background: lightcoral;
    border: 2px solid black;
}


.m-1 { margin: 20px; }
.m-2 { margin: 10px 40px; }
.m-3 { margin: 10px 20px 30px; }
.m-4 { margin: 5px 10px 15px 20px; }
.m-auto { margin: 20px auto; }
</style>
</head>
<body>


<div class="box m-1">margin: 20px</div>
<div class="box m-2">margin: 10px 40px</div>
<div class="box m-3">margin: 10px 20px 30px</div>
<div class="box m-4">margin: 5px 10px 15px 20px</div>
<div class="box m-auto">margin: auto (centered)</div>


</body>
</html>
```



How values are applied:

- 1 value → all sides
- 2 values → top/bottom | left/right
- 3 values → top | left/right | bottom
- 4 values → top | right | bottom | left

Box Sizing

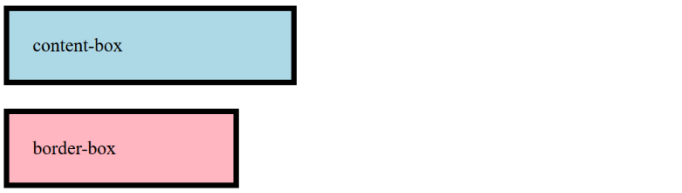
By default, width and height apply **only to content**.

box-sizing: content-box; **default**

box-sizing: border-box;

- content-box: padding and border add to size
- border-box: padding and border included in width/height (recommended)

```
<!DOCTYPE html>
<html>
<head>
<style>
.box {
    width: 200px;
    padding: 20px;
    border: 5px solid black;
    margin-bottom: 20px;
}


.content-box { box-sizing: content-box; background: lightblue; }
.border-box { box-sizing: border-box; background: lightpink; }
</style>
</head>
<body>


<div class="box content-box">content-box</div>
<div class="box border-box">border-box</div>


</body>
</html>
```



Box Shadow

box-shadow adds **shadow effects** around elements to create depth, elevation, or emphasis. Shadows do **not** affect layout.

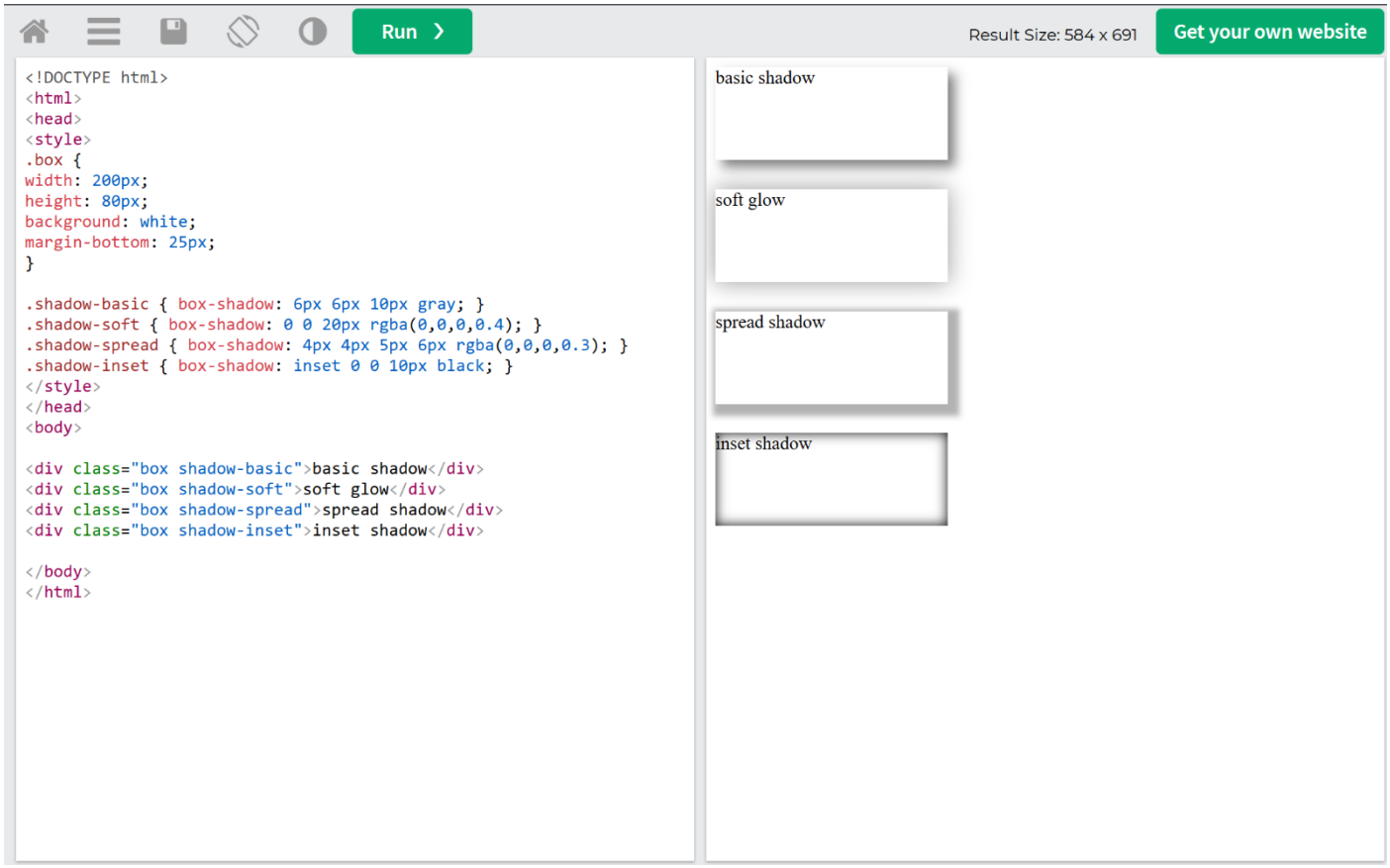
Syntax:

box-shadow: offset-x offset-y blur-radius spread-radius color;

Properties & values:

- offset-x → horizontal shift (required)
- offset-y → vertical shift (required)
- blur-radius → softness (optional)
- spread-radius → size expansion (optional)
- color → any CSS color format
- inset → shadow inside the element

Multiple shadows can be applied, separated by commas.



CSS Layout & Flow

DISPLAY

What display controls

- How an element participates in layout
- Whether it starts on a new line
- Whether width/height apply

Common values

- block
- inline
- inline-block
- none
- flex
- grid

Run ›

Result Size: 584 x 691

Get your own website

```
<!Doctype html>
<html>
<head>
<style>
.box {
background: lightblue;
padding: 10px;
margin: 5px;
width: 120px;
height: 60px;
border: 2px solid navy;
}

.block { display: block; }
.inline { display: inline; }
.inline-block { display: inline-block; }
.none { display: none; }
.flex { display: flex; gap: 10px; }
.grid { display: grid; grid-template-columns: repeat(3, 1fr); gap:
10px; }
</style>
</head>
<body>

<h3>Block</h3>
<div class="container">
<div class="box block">Block 1</div>
<div class="box block">Block 2</div>
</div>

<h3>Inline</h3>
<div class="container">
<div class="box inline">Inline 1</div>
<div class="box inline">Inline 2</div>
</div>

<h3>Inline-Block</h3>
<div class="container">
```

Block

Block 1

Block 2

Inline

Inline 1 Inline 2

Inline-Block

IB 1 IB 2

Display None

Visible

Still here

Run ›

Result Size: 584 x 691

Get your own website

```
</div>

<h3>Inline</h3>
<div class="container">
<div class="box inline">Inline 1</div>
<div class="box inline">Inline 2</div>
</div>

<h3>Inline-Block</h3>
<div class="container">
<div class="box inline-block">IB 1</div>
<div class="box inline-block">IB 2</div>
</div>

<h3>Display None</h3>
<div class="container">
<div class="box">Visible</div>
<div class="box none">Hidden</div>
<div class="box">Still here</div>
</div>

<h3>Flex (preview)</h3>
<div class="container flex">
<div class="box">1</div>
<div class="box">2</div>
<div class="box">3</div>
</div>

<h3>Grid (preview)</h3>
<div class="container grid">
<div class="box">A</div>
<div class="box">B</div>
<div class="box">C</div>
</div>

</body>
</html>
```

Inline-Block

IB 1 IB 2

Display None

Visible

Still here

Flex (preview)

1 2 3

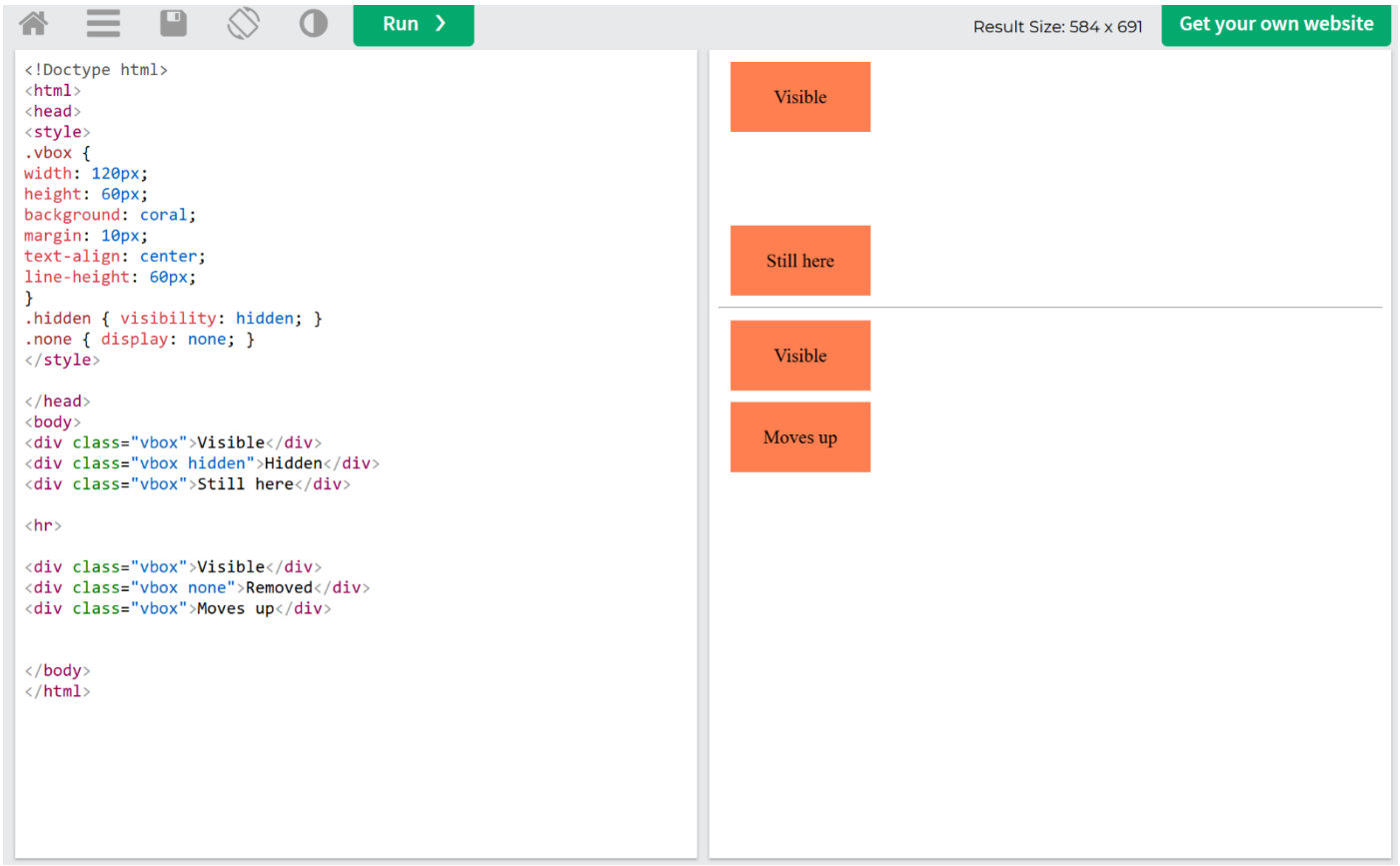
Grid (preview)

A B C

VISIBILITY

Key difference

- **visibility: hidden** → invisible **but space remains**
- **display: none** → removed completely

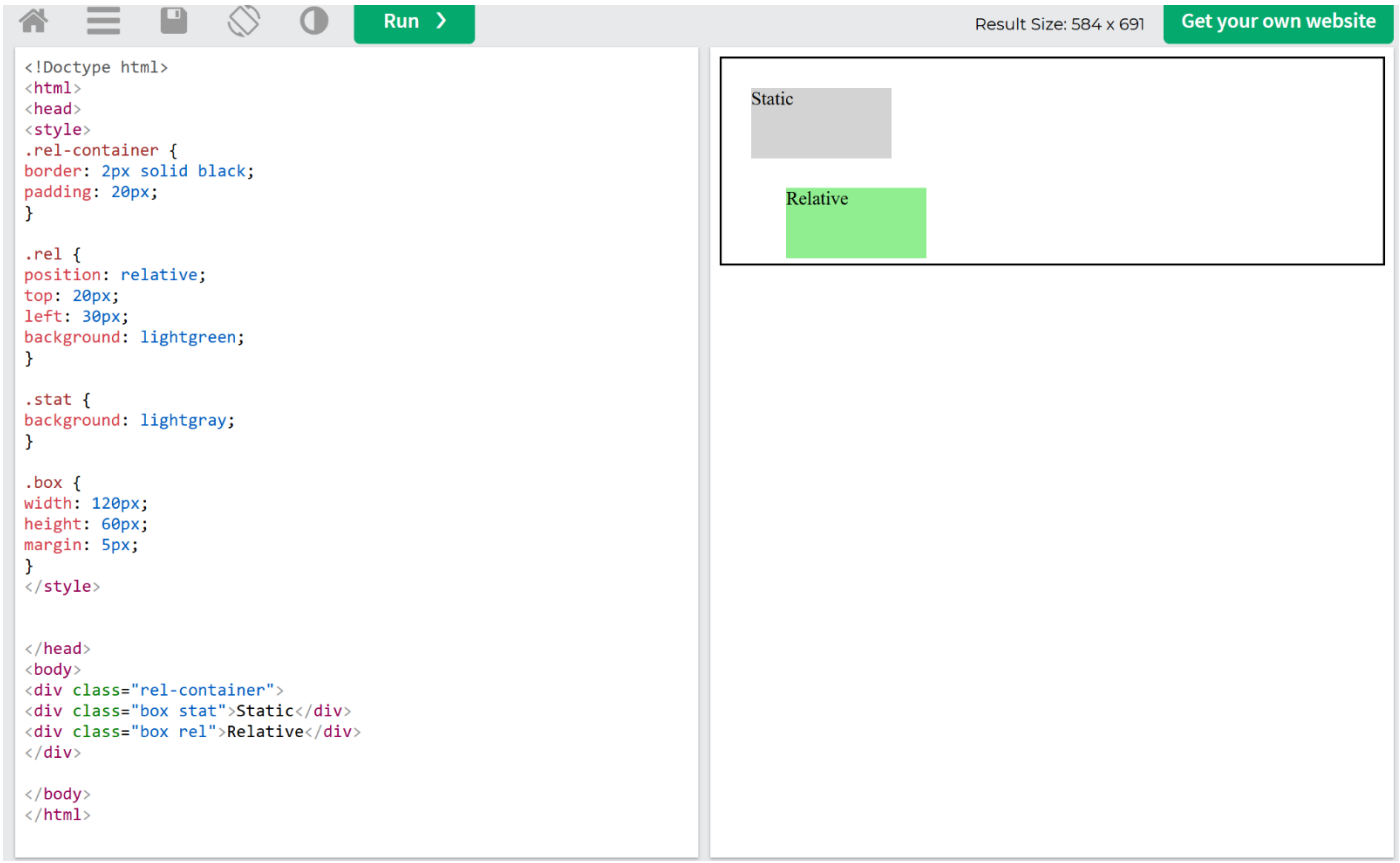


POSITIONING

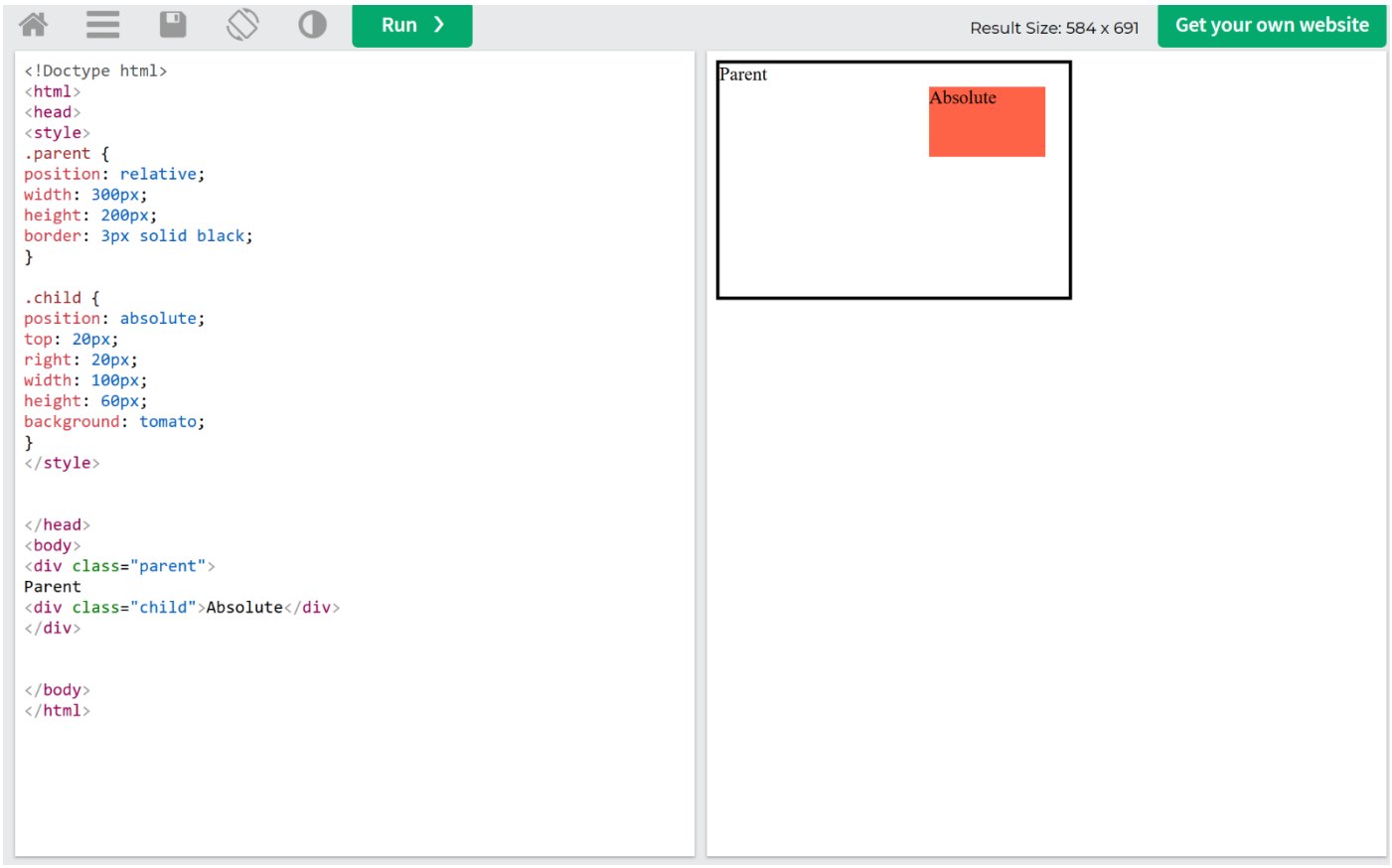
Position types

- **static** (default)
- **relative** (relative to Parent)
- **absolute** (relative to Body)
- **fixed** (relative to Viewport)
- **sticky** (relative to the viewport until parent ends)

Static vs Relative

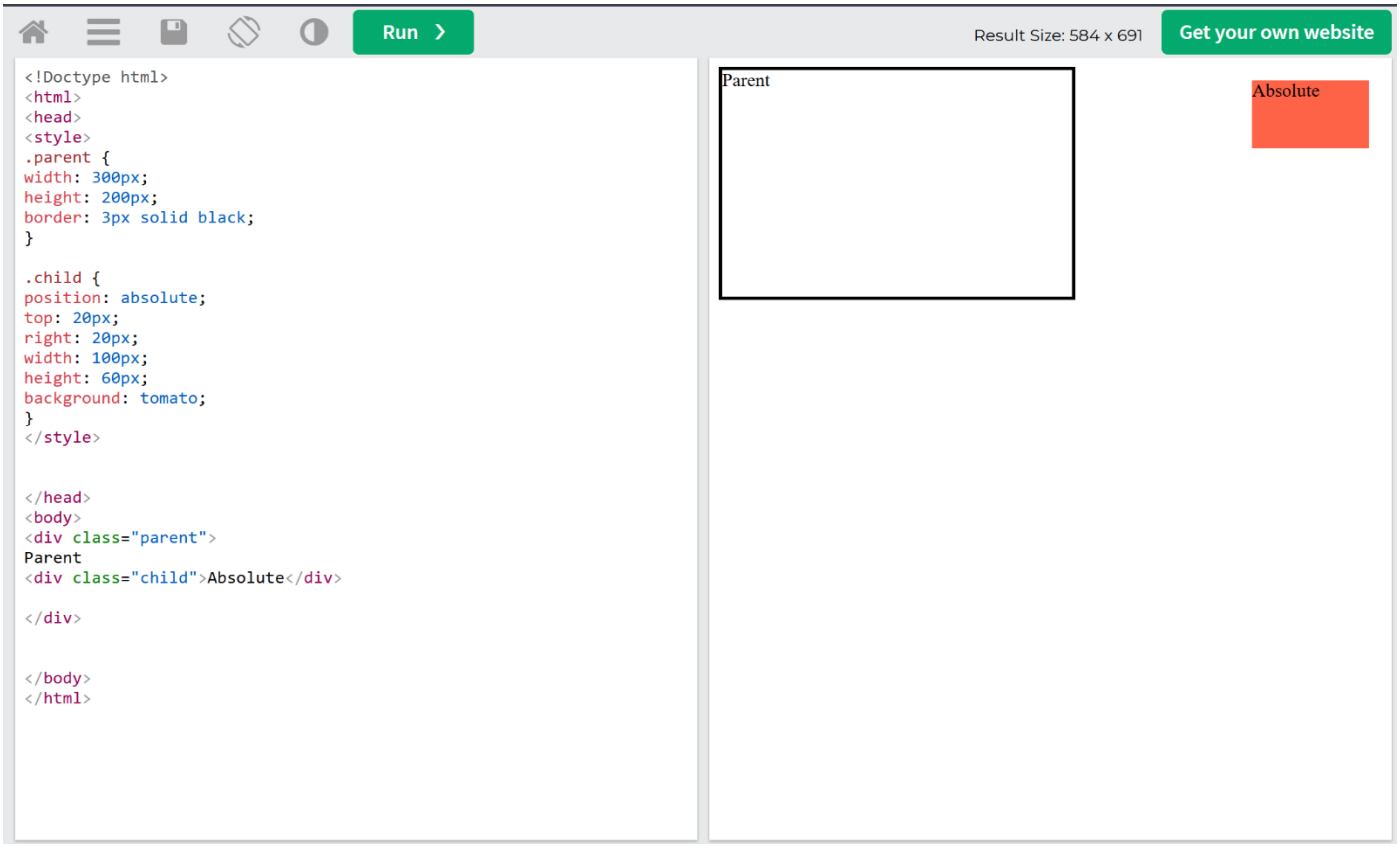


Absolute with Positioned Parent




```
<!Doctype html>
<html>
<head>
<style>
.parent {
position: relative;
width: 300px;
height: 200px;
border: 3px solid black;
}

.child {
position: absolute;
top: 20px;
right: 20px;
width: 100px;
height: 60px;
background: tomato;
}
</style>


</head>
<body>
<div class="parent">
Parent
<div class="child">Absolute</div>
</div>


</body>
</html>
```

Absolute without Positioned Parent




```
<!Doctype html>
<html>
<head>
<style>
.parent {
width: 300px;
height: 200px;
border: 3px solid black;
}

.child {
position: absolute;
top: 20px;
right: 20px;
width: 100px;
height: 60px;
background: tomato;
}
</style>


</head>
<body>
<div class="parent">
Parent
<div class="child">Absolute</div>

</div>


</body>
</html>
```

Fixed Position

Run › Result Size: 584 x 691 Get your own website

```
<!Doctype html>
<html>
<head>
<style>
.fixed {
position: fixed;
bottom: 20px;
right: 20px;
background: black;
color: white;
padding: 10px;
}
</style>


</head>
<body>
<div class="fixed">Fixed Button</div>

</body>
</html>
```

Sticky Position

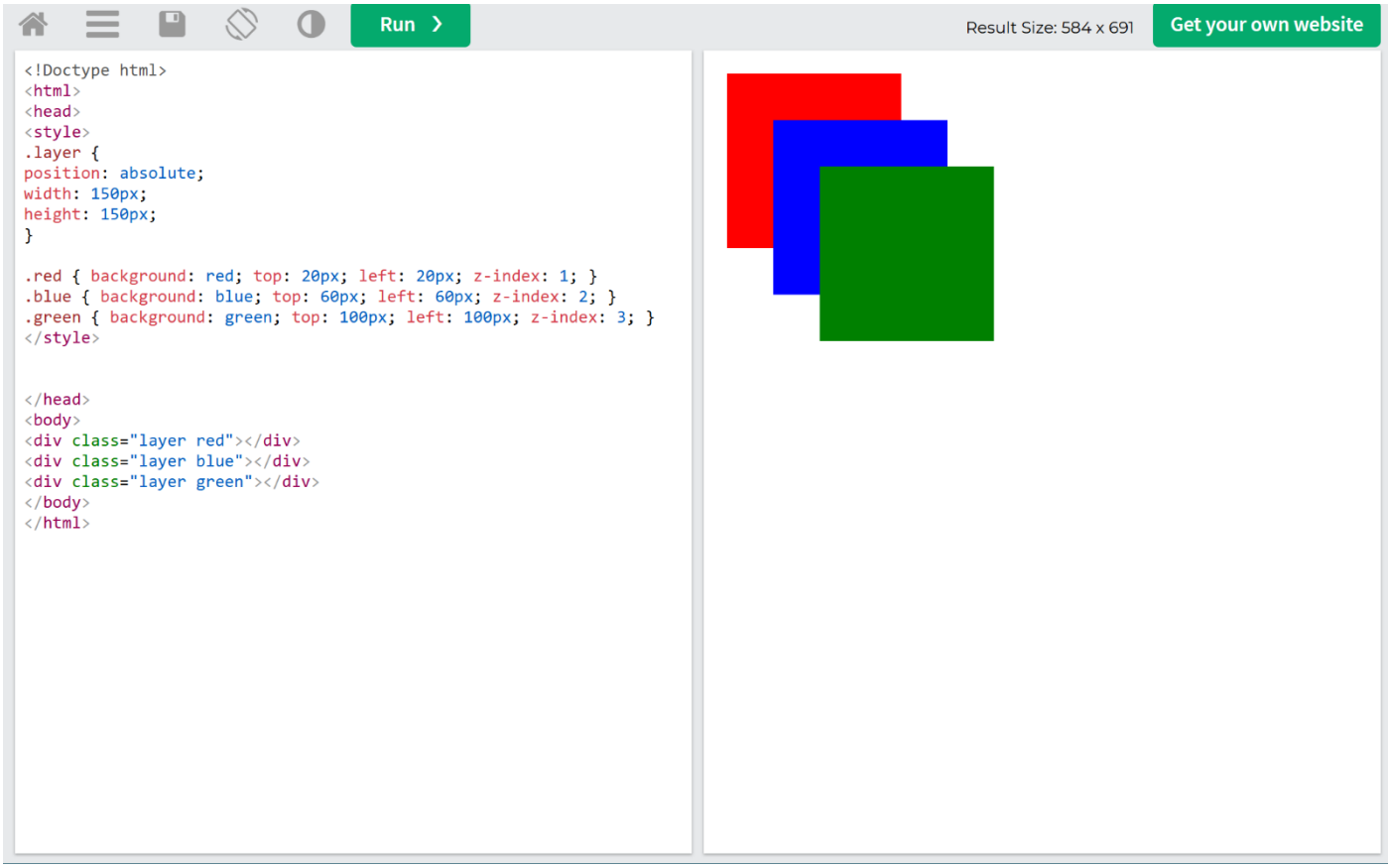
Run › Result Size: 584 x 691 Get your own website

```
<!Doctype html>
<html>
<head>
<style>
.sticky {
position: sticky;
top: 0;
background: gold;
padding: 10px;
}


.spacer { height: 800px; }
</style>

</head>
<body>
<div class="sticky">I stick on scroll</div>
<div class="spacer"></div>
</body>
</html>
```

Z-INDEX & STACKING CONTEXT

Rules

- Only works on positioned elements
- Higher z-index appears on top (0 is default), if indexes are equal -> first one created moves to back
- Parent stacking context limits children



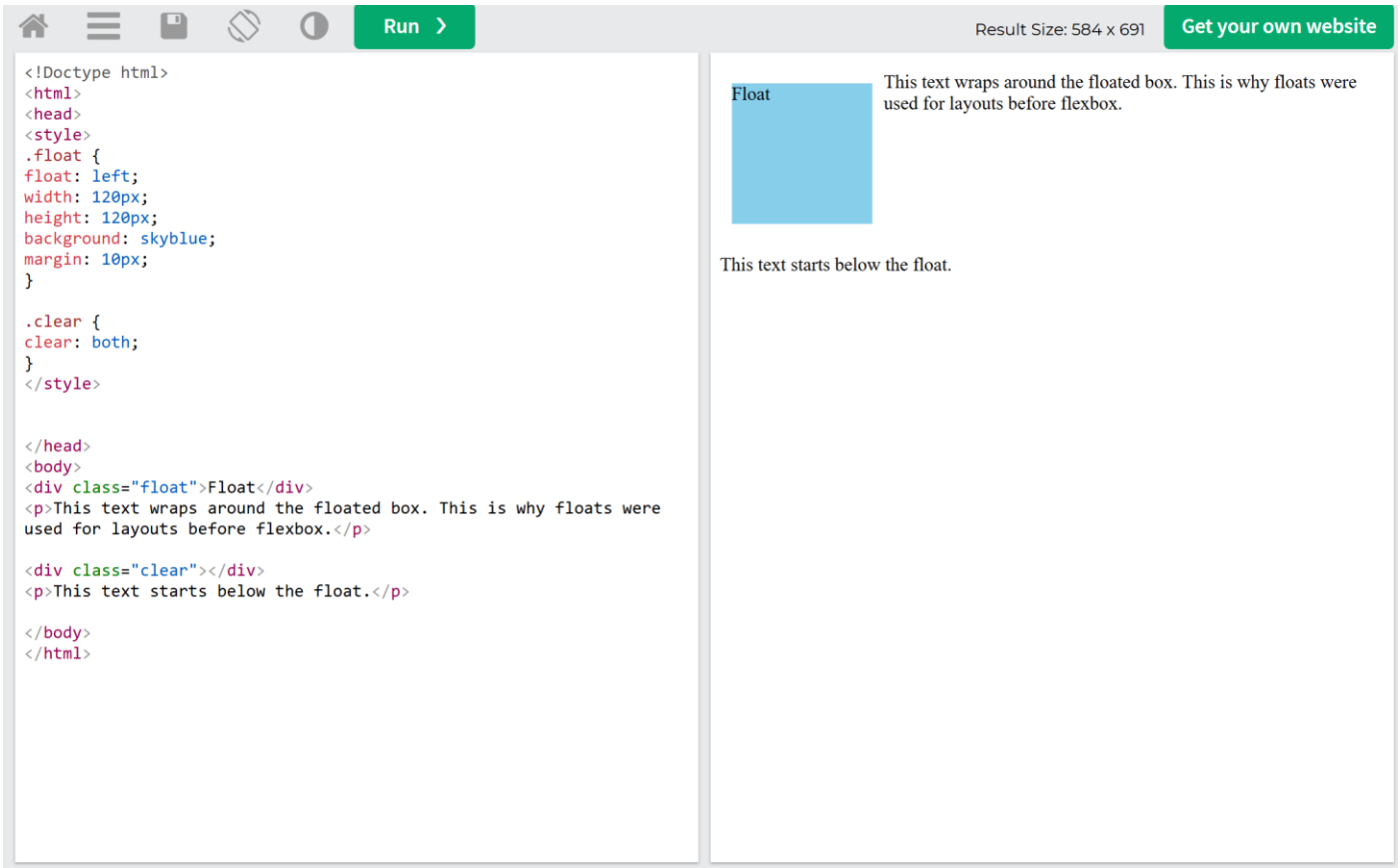
FLOAT & CLEAR

Float behavior

- Removes element from normal flow
- Text wraps around it
- Values of float: left, right, none(default)

Clear behavior

- don’t sit next to floated elements — start below them
- Values of clear: left, right, both, none(default)



CSS Flexbox

Flexbox is a **one-dimensional layout system**. Each property below lists **all possible values**, then demonstrates **all of them together** using colored, numbered boxes so differences are obvious.

Base Setup (Shared by All Examples)

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<style>
body {
font-family: Arial, sans-serif;
}

.section {
margin-bottom: 60px;
}

.title {
font-weight: bold;
margin-bottom: 10px;
}

.container {
display: flex;
background: #eee;
padding: 10px;
margin-bottom: 15px;
border: 2px solid #ccc;
}

.box {
width: 70px;
height: 70px;
color: white;
font-size: 20px;
display: flex;
align-items: center;
justify-content: center;
}

.b1 { background: crimson; }
.b2 { background: dodgerblue; }
.b3 { background: seagreen; }
```

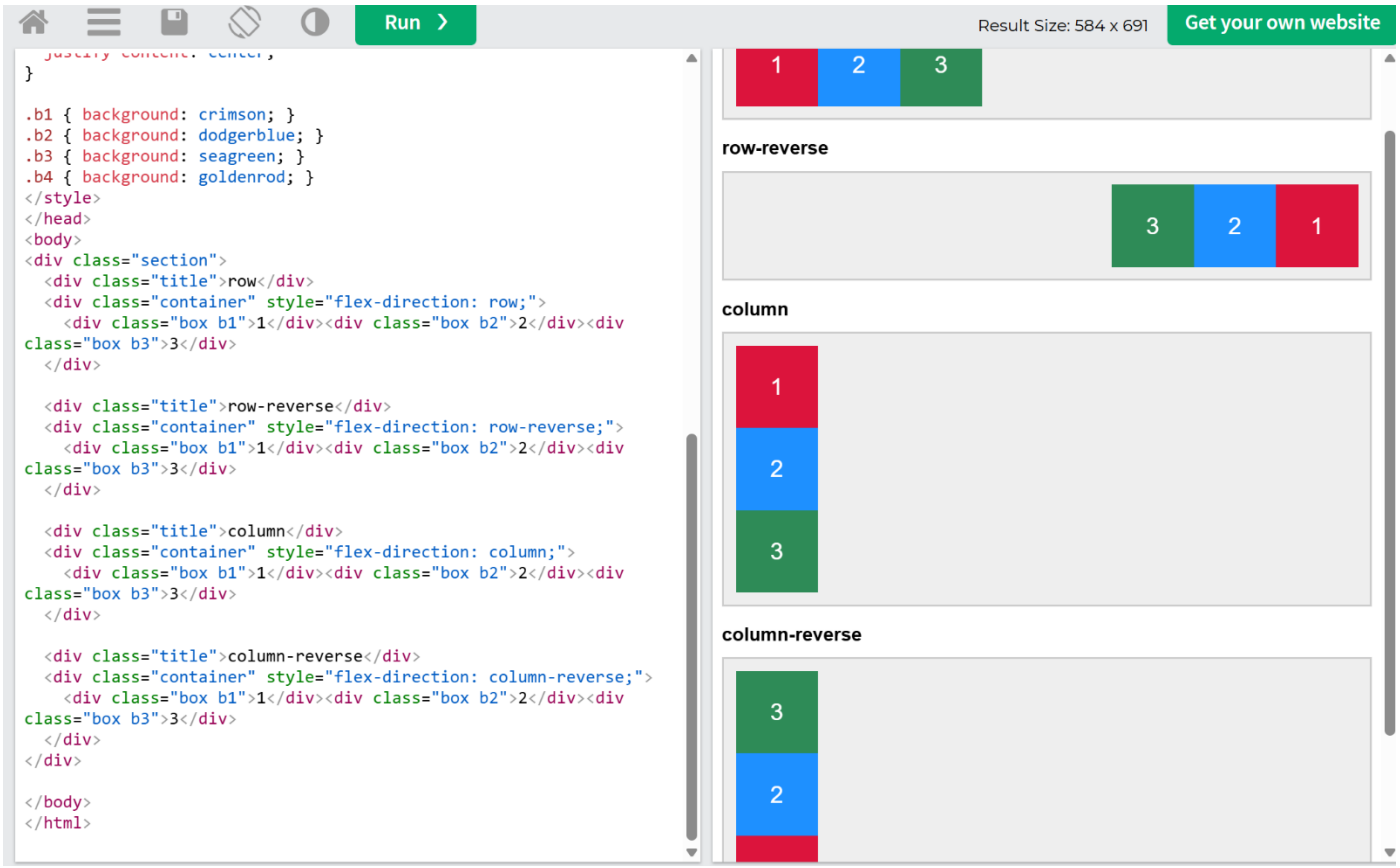
```
.b4 { background: goldenrod; }
</style>
</head>
<body>

</body>
</html>
```

---

flex-direction

Values

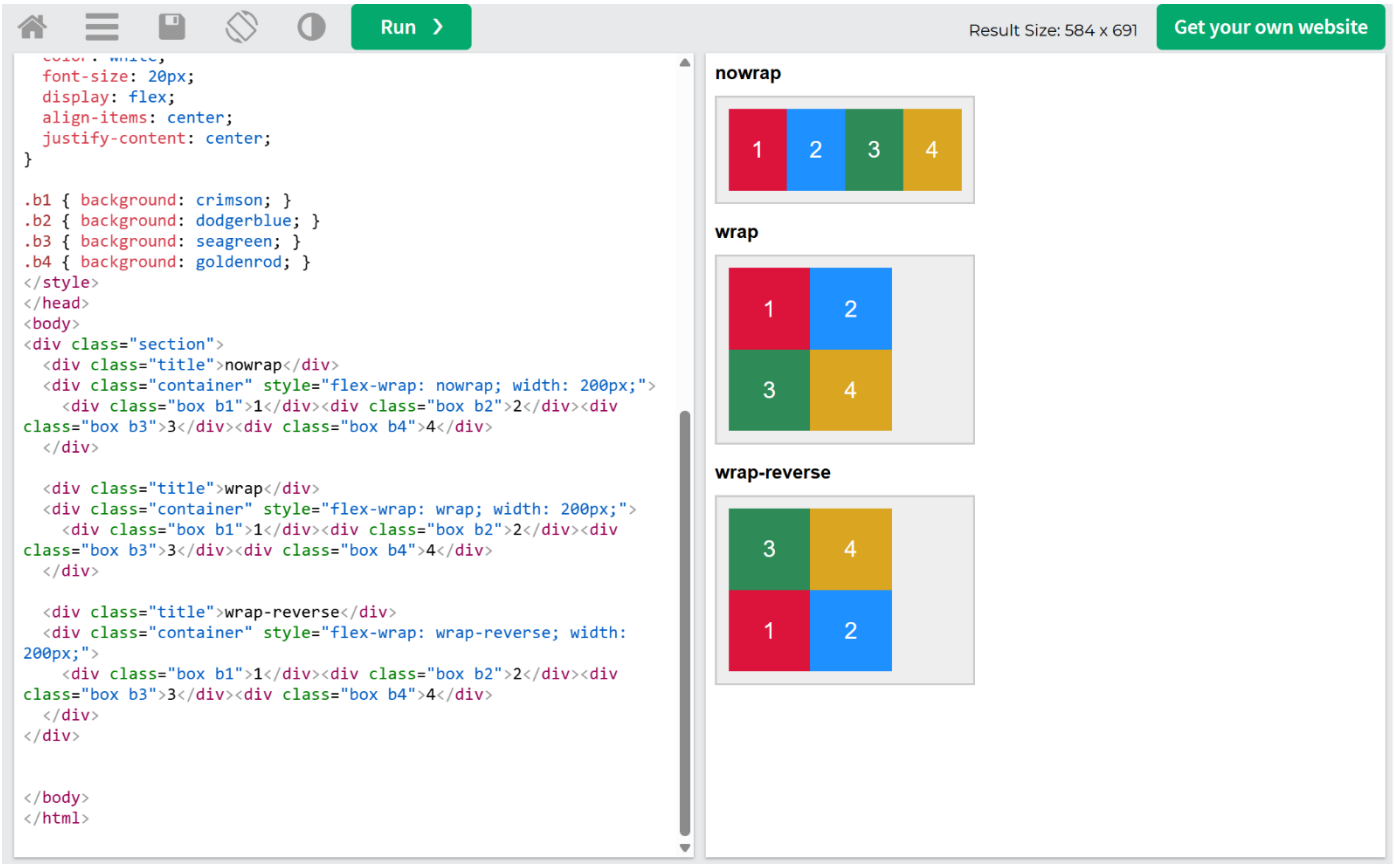
- row (default)
- row-reverse
- column
- column-reverse



flex-wrap

Values

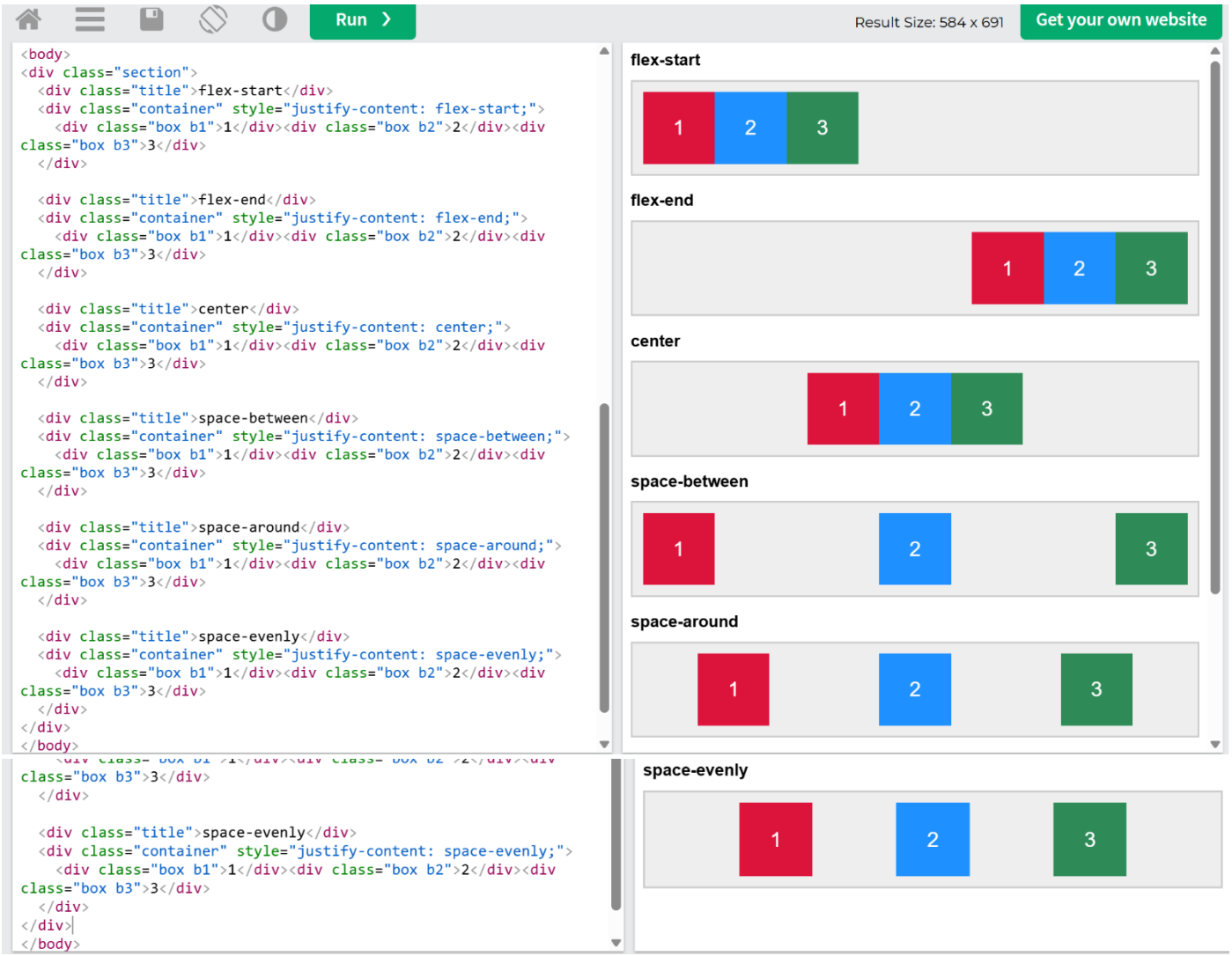
- nowrap (default)
- wrap
- wrap-reverse



justify-content

Values

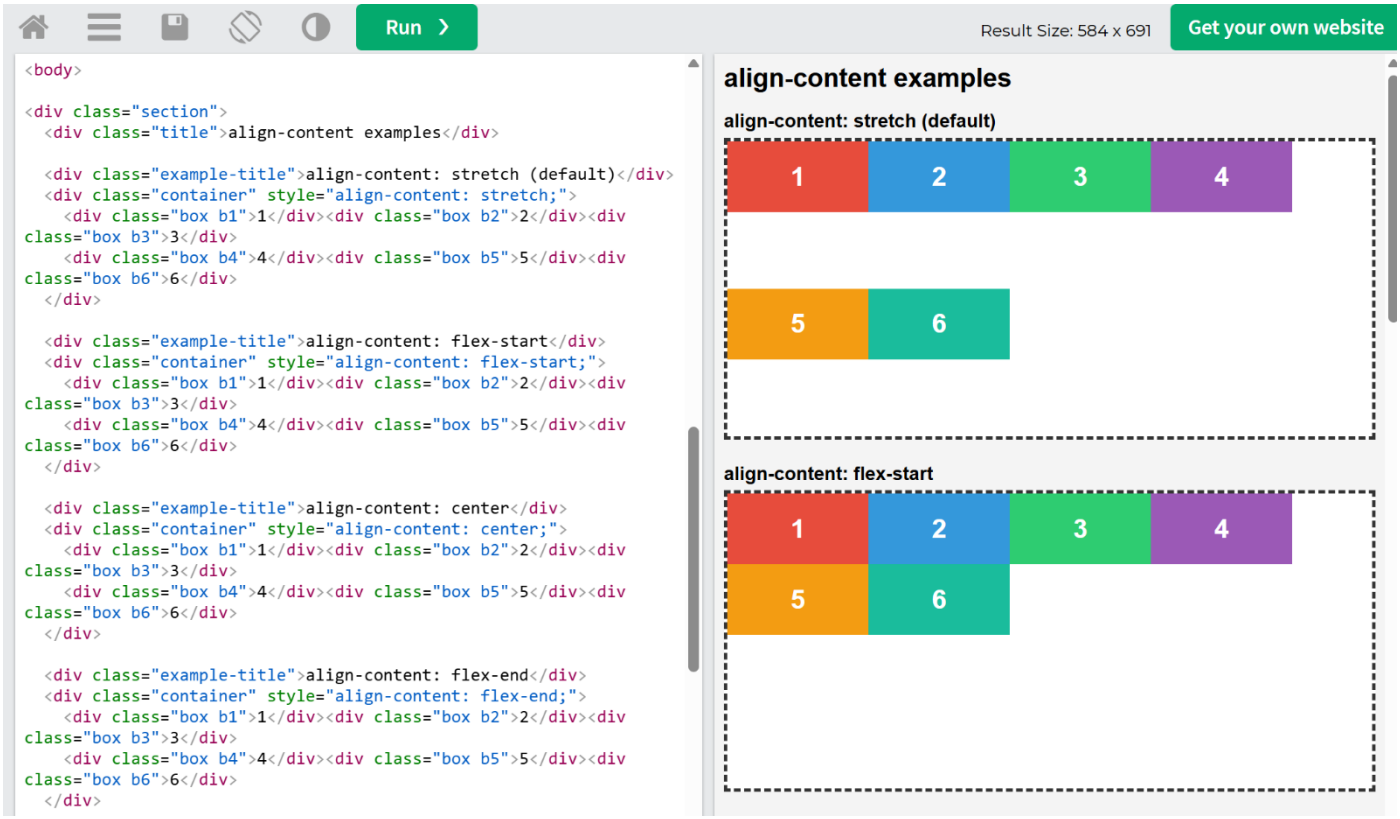
- flex-start
- flex-end
- center
- space-between
- space-around
- space-evenly

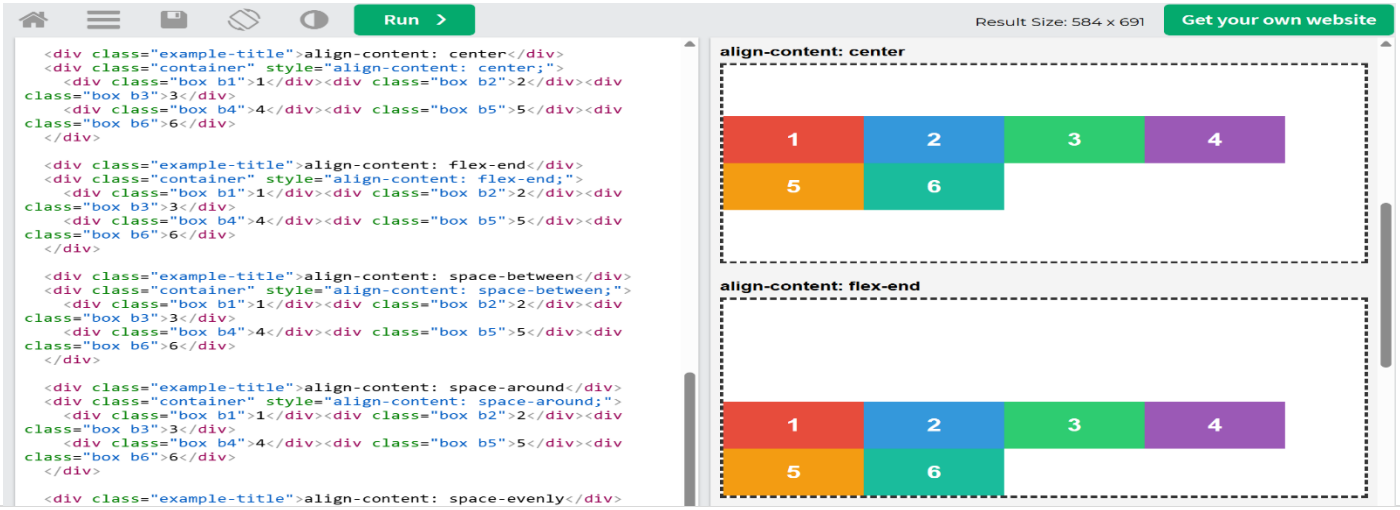


align-content (requires wrap)

Values

- stretch (default)
- flex-start
- flex-end
- center
- space-between
- space-around
- space-evenly



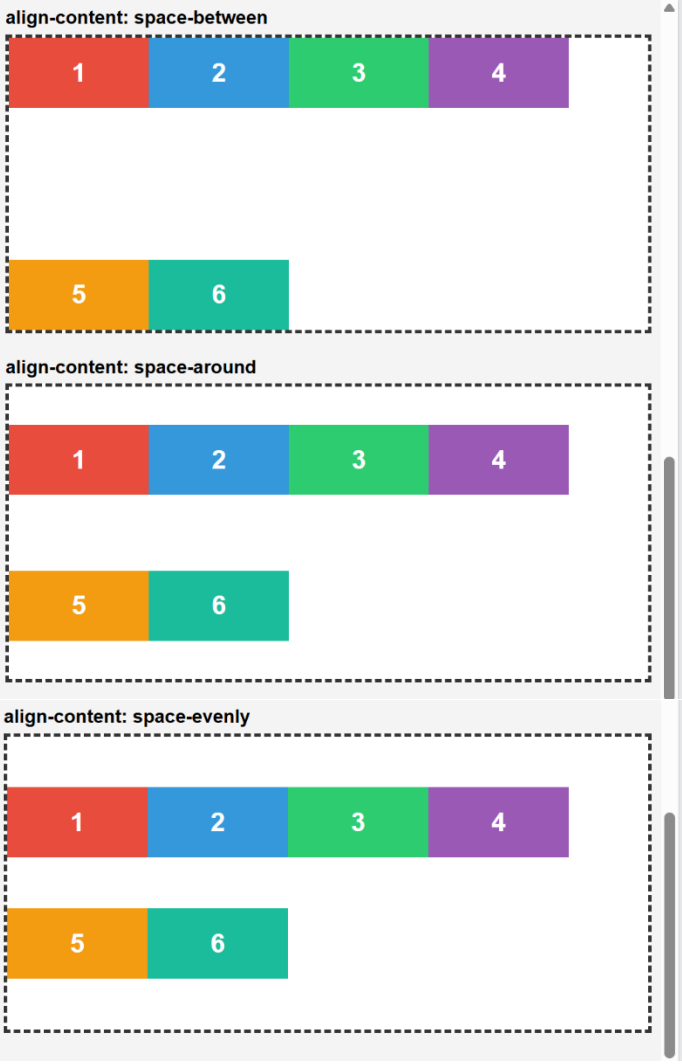


Run ›

Result Size: 584 x 691

Get your own website

```
  <div class="example-title">align-content: flex-end</div>
  <div class="container" style="align-content: flex-end;">
    <div class="box b1">1</div><div class="box b2">2</div><div
class="box b3">3</div>
    <div class="box b4">4</div><div class="box b5">5</div><div
class="box b6">6</div>
  </div>

  <div class="example-title">align-content: space-between</div>
  <div class="container" style="align-content: space-between;">
    <div class="box b1">1</div><div class="box b2">2</div><div
class="box b3">3</div>
    <div class="box b4">4</div><div class="box b5">5</div><div
class="box b6">6</div>
  </div>

  <div class="example-title">align-content: space-around</div>
  <div class="container" style="align-content: space-around;">
    <div class="box b1">1</div><div class="box b2">2</div><div
class="box b3">3</div>
    <div class="box b4">4</div><div class="box b5">5</div><div
class="box b6">6</div>
  </div>

  <div class="example-title">align-content: space-evenly</div>
  <div class="container" style="align-content: space-evenly;">
    <div class="box b1">1</div><div class="box b2">2</div><div
class="box b3">3</div>
    <div class="box b4">4</div><div class="box b5">5</div><div
class="box b6">6</div>
  </div>

class="box b3">3</div>
    <div class="box b4">4</div><div class="box b5">5</div><div
class="box b6">6</div>
  </div>

  <div class="example-title">align-content: space-evenly</div>
  <div class="container" style="align-content: space-evenly;">
    <div class="box b1">1</div><div class="box b2">2</div><div
class="box b3">3</div>
    <div class="box b4">4</div><div class="box b5">5</div><div
class="box b6">6</div>
  </div>

</div>

</body>
</html>
```

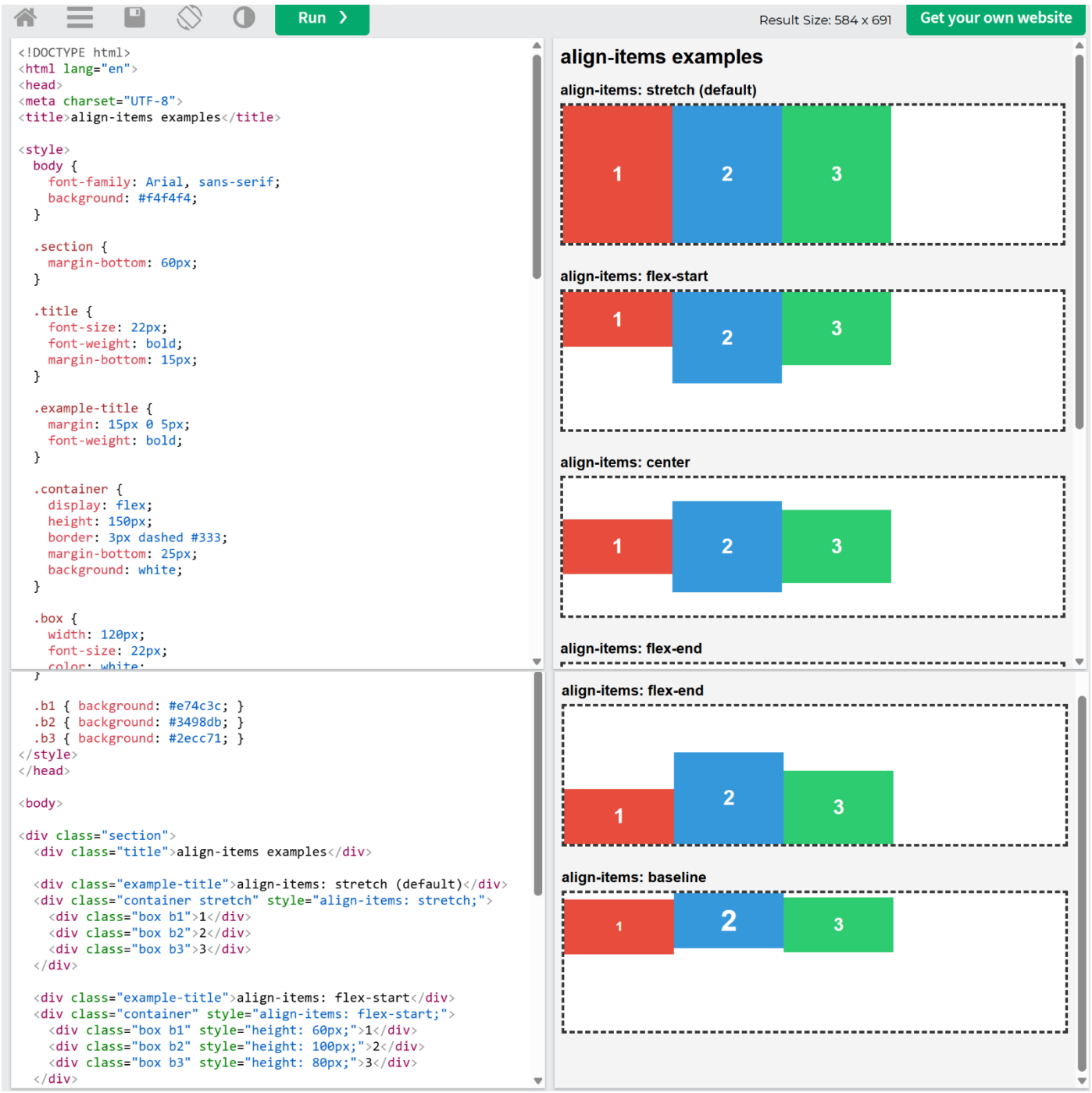


Made by Youssef Amgad Elkhatib

align-items

Values

- stretch (default)
- flex-start
- flex-end
- center
- baseline



align-self & justify-self

Align-self and justify-self **use the same values** as their container counterparts (align-items and justify-items), **but they apply to a single element only**.

They **override the container's alignment** for that specific item without affecting the rest.

```
.container {
  align-items: center;
}

.box.special {
  align-self: flex-end;
}
```

Flex Item Properties (grow, shrink, basis, order)

flex-grow

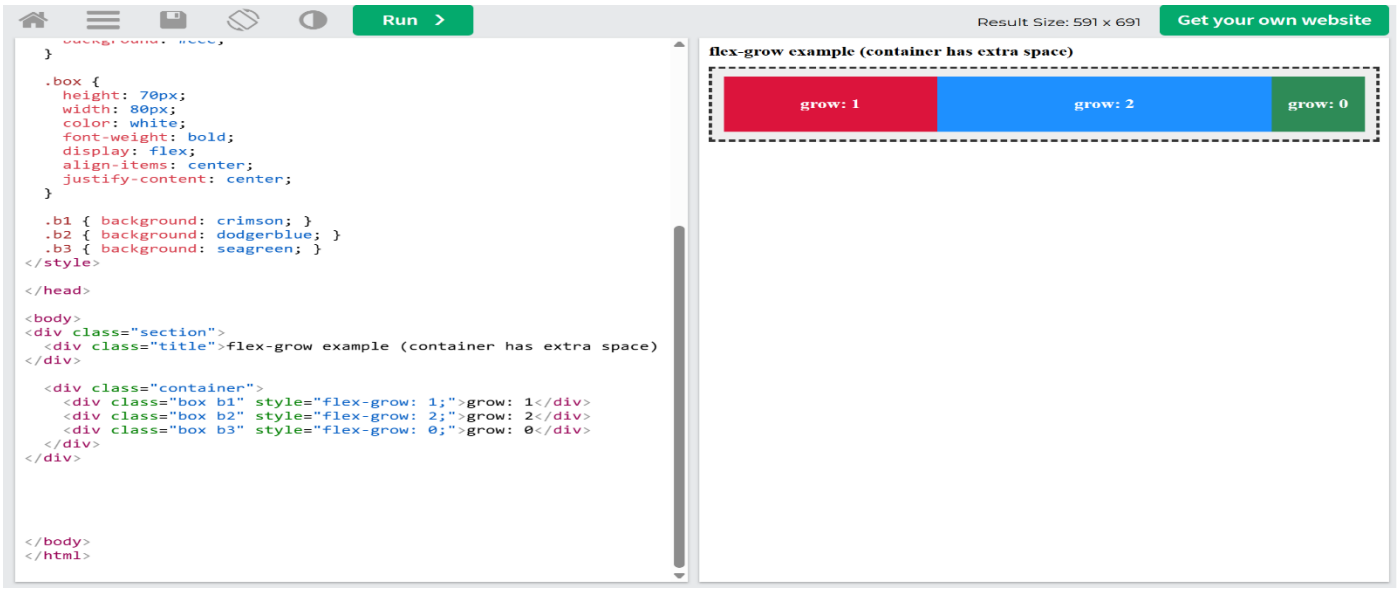
What it does

flex-grow controls **how much an item grows** relative to other items **when there is extra space** in the container.

- Default value: 0
- Works on the **main axis**
- Growth is **proportional**

Values

- 0 (do not grow)
- Any positive number (1, 2, 3, …)



flex-shrink

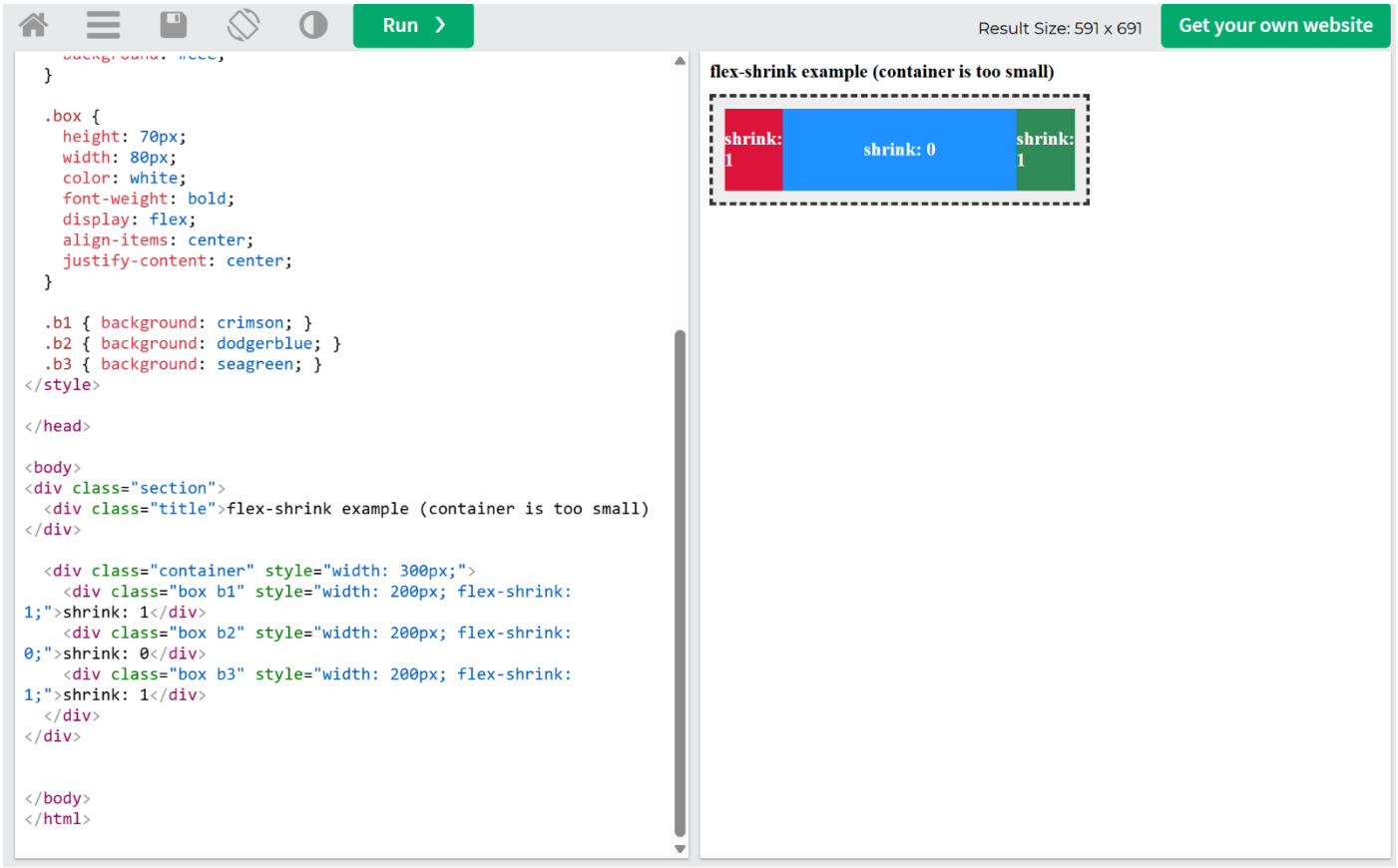
What it does

flex-shrink controls **how much an item shrinks** when there is **not enough space** in the container.

- Default value: 1
- Works only when items overflow
- Shrinking is **proportional**

Values

- 0 (do not shrink)
- Any positive number (1, 2, …)



flex-basis

What it does

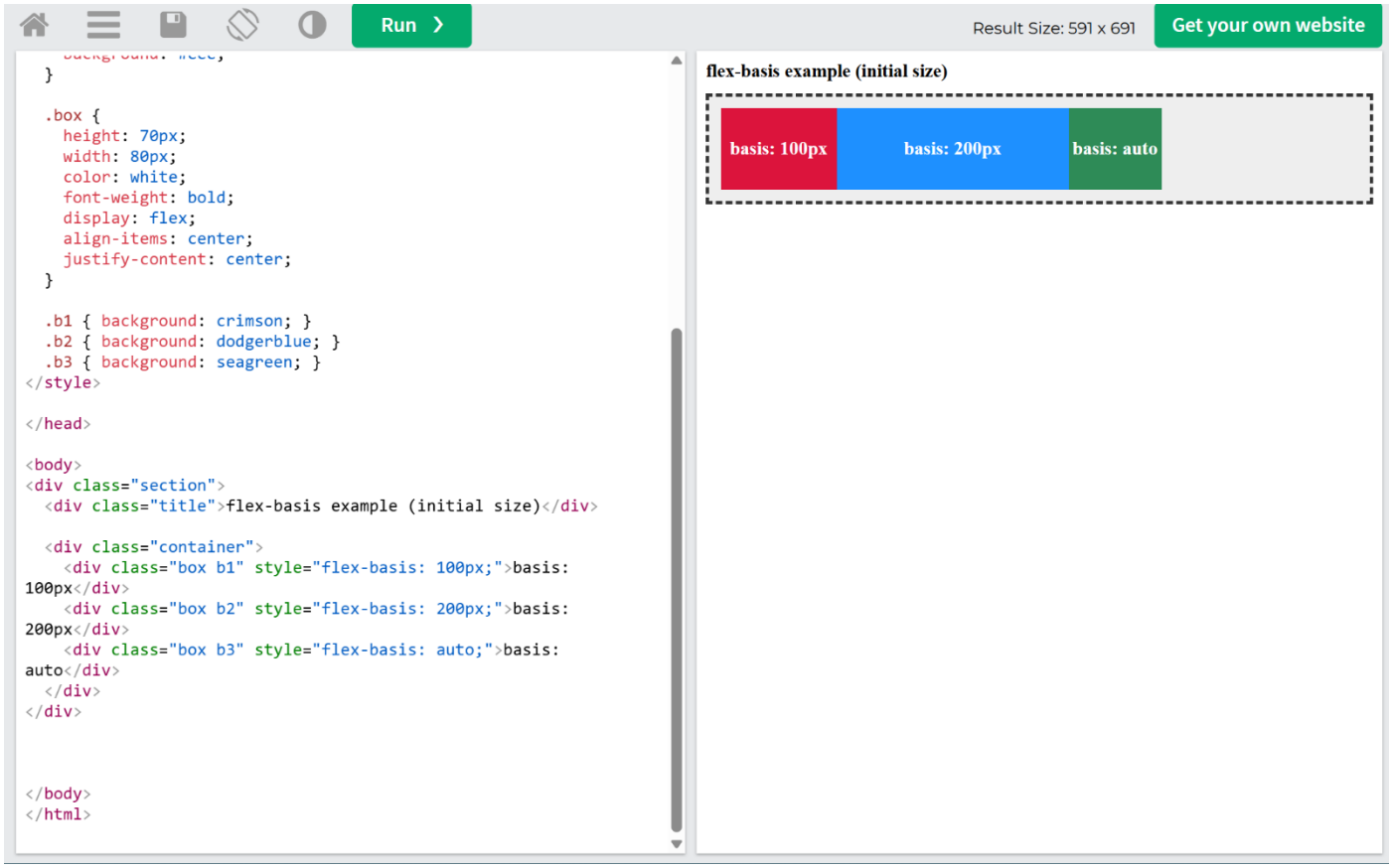
flex-basis defines the **initial size** of a flex item **before grow or shrink happens**.

Think of it as:

“My starting width (or height)”

Values

- Lengths: px, %, em, etc.
- auto (default)
- content



order

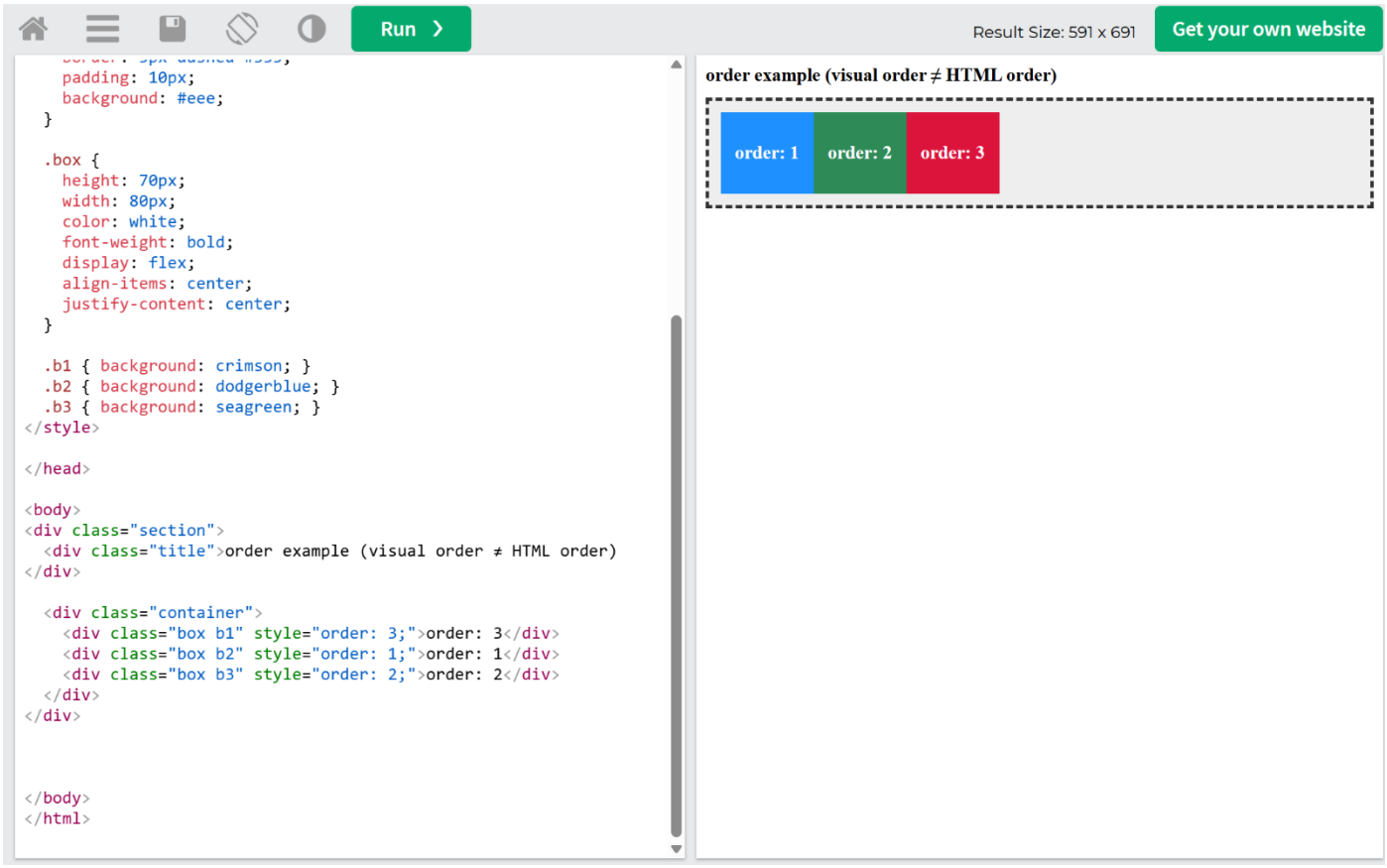
What it does

order controls the **visual order** of items **without changing HTML order**.

- Default value: 0
- Lower numbers appear first

Values

- Any integer (-1, 0, 1, 10, etc.)



CSS Grid Layout

CSS Grid is a **2D layout system** that lets you control **rows and columns at the same time**.

Grid Basics (must exist first)

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

Nothing works without this.

---

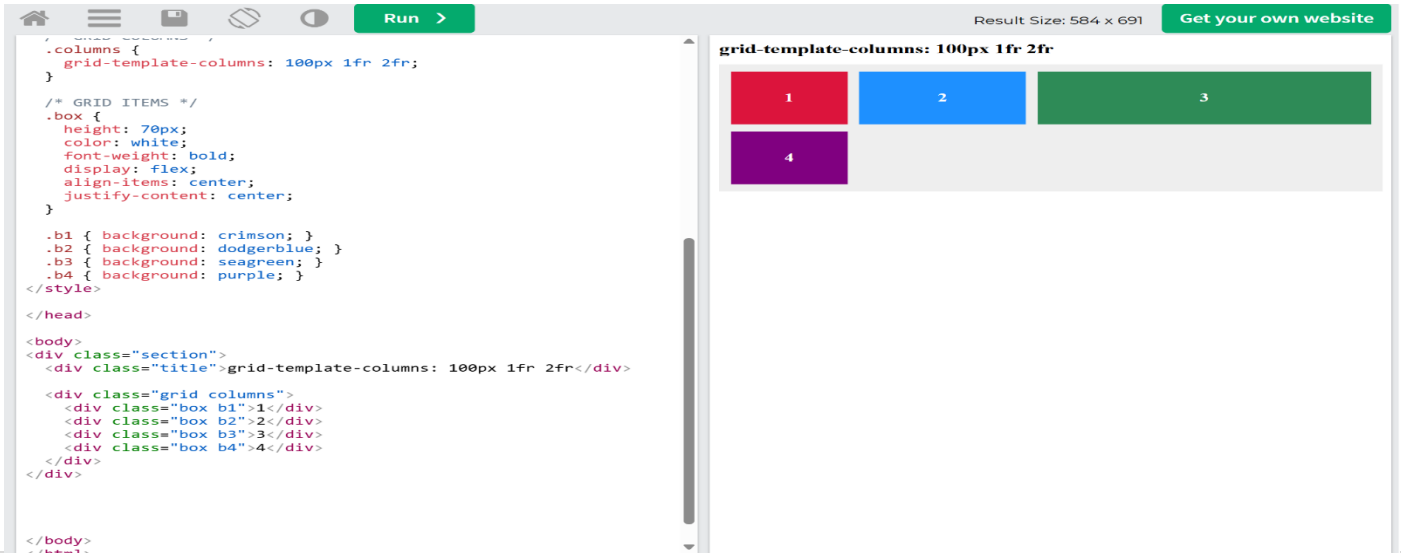
grid-template-columns

What it does

Defines **how many columns** the grid has and their sizes.

Common values

- Fixed: 100px, 200px
- Flexible: 1fr, 2fr -> **by ratio**
- Auto: auto
- Functions:
    - repeat(4, 25% 25%) -> **repeat four times, each time 25 %, resulting in 8 columns each 25%**
    - minmax(**minimum value, maximum value)**
    - fit-content(200px) -> fits data in 200px if it is more, it will wrap in the next line.



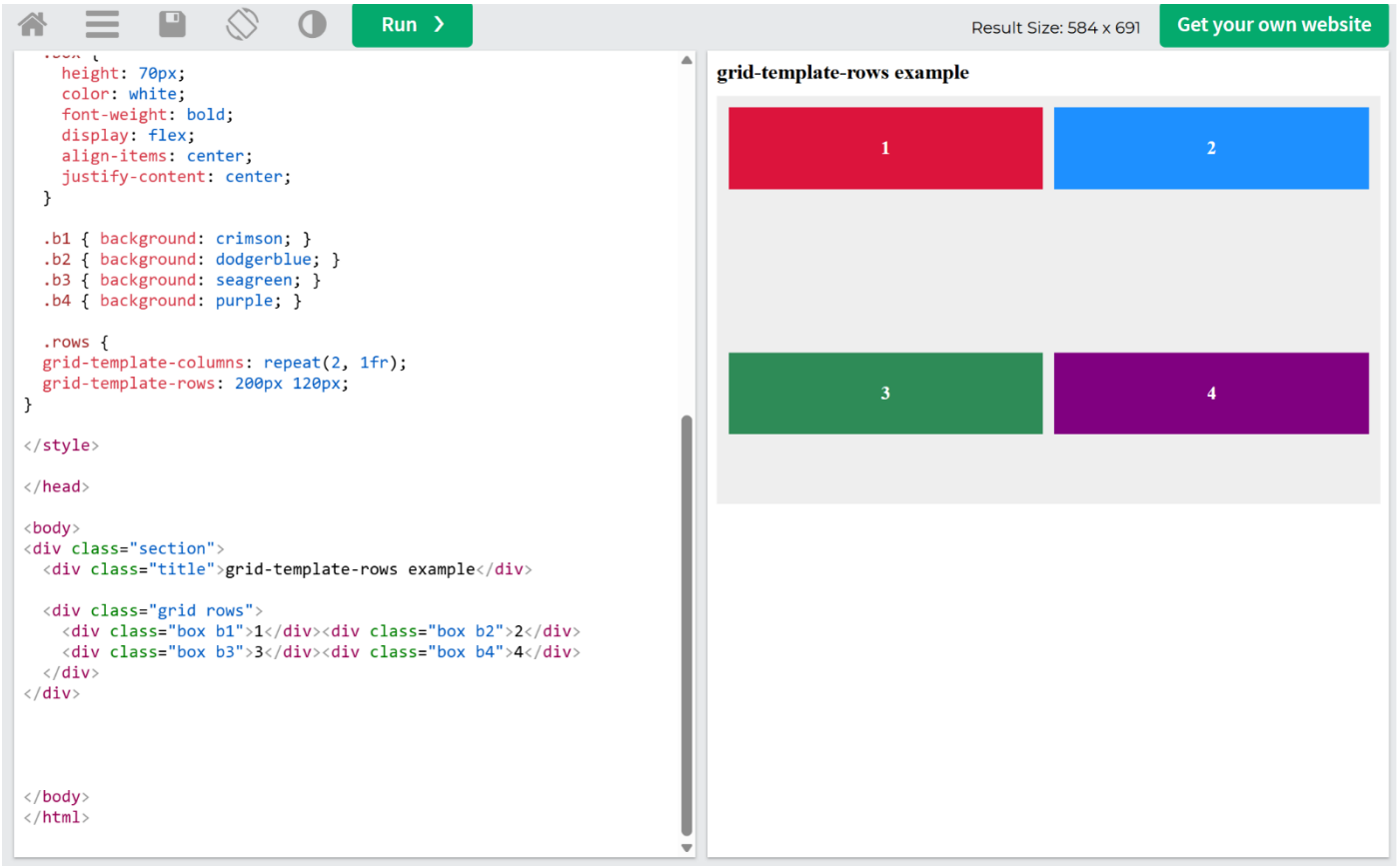
grid-template-rows

What it does

Defines **row heights**.

Values

- Same as columns: px, fr, auto, minmax()



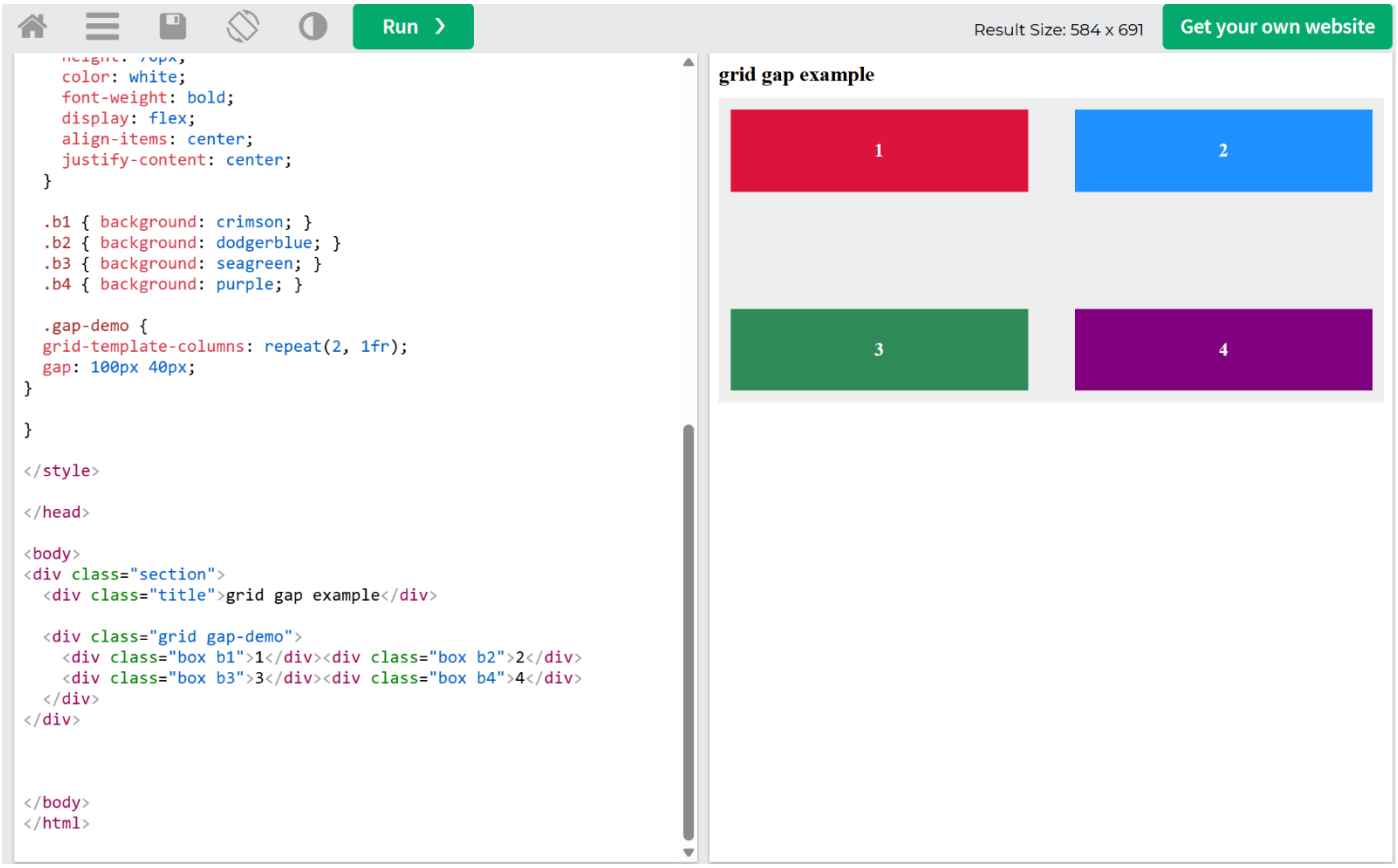
gap (row-gap / column-gap)

What it does

Controls **spacing between grid cells**.

Values

- gap
- row-gap
- column-gap



Grid Alignment (Container)

justify-items (horizontal alignment of items)

align-items (vertical alignment of items)

Values

- start
- center
- end
- stretch (default)

Grid Content Alignment (whole grid)

justify-content

align-content

These move the **entire grid inside the container**.

Values

- start
- center
- end
- space-between
- space-around
- space-evenly

Grid Item Alignment (per item)

justify-self

align-self

Same idea as Flexbox:
Overrides container alignment for one item

Grid Lines (Positioning by lines)

What are Grid Lines?

When you create a CSS Grid, the **columns and rows create invisible lines**:

- Vertical lines → column lines
- Horizontal lines → row lines

These lines are **numbered automatically** starting from **1**, counting from the **top-left** corner:

- Columns: left → right
- Rows: top → bottom

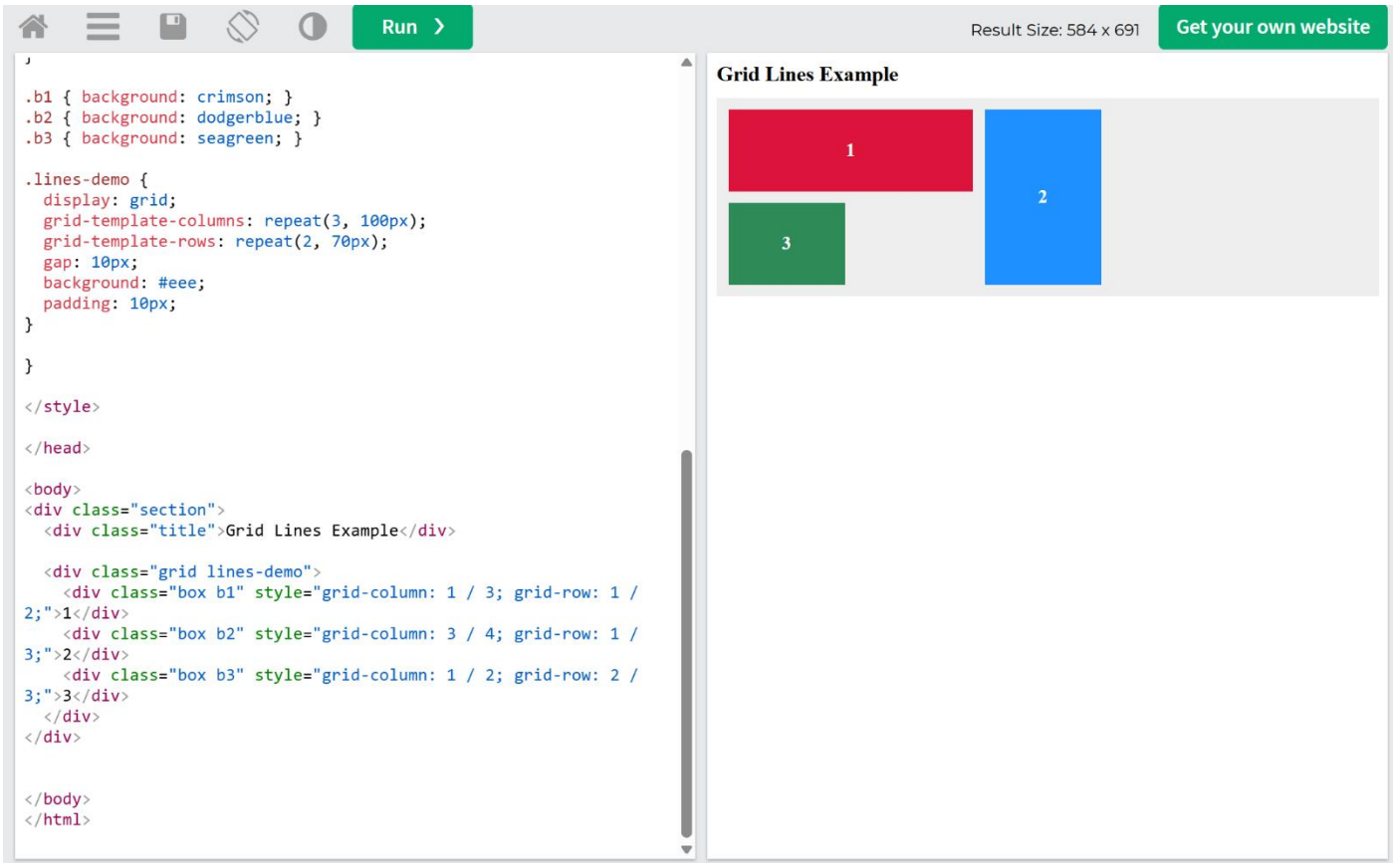
You can **position items by referring to these lines** using:

grid-column: start / end;

grid-row: start / end;

Where:

- start = line number to start at
- end = line number to end **before** that line

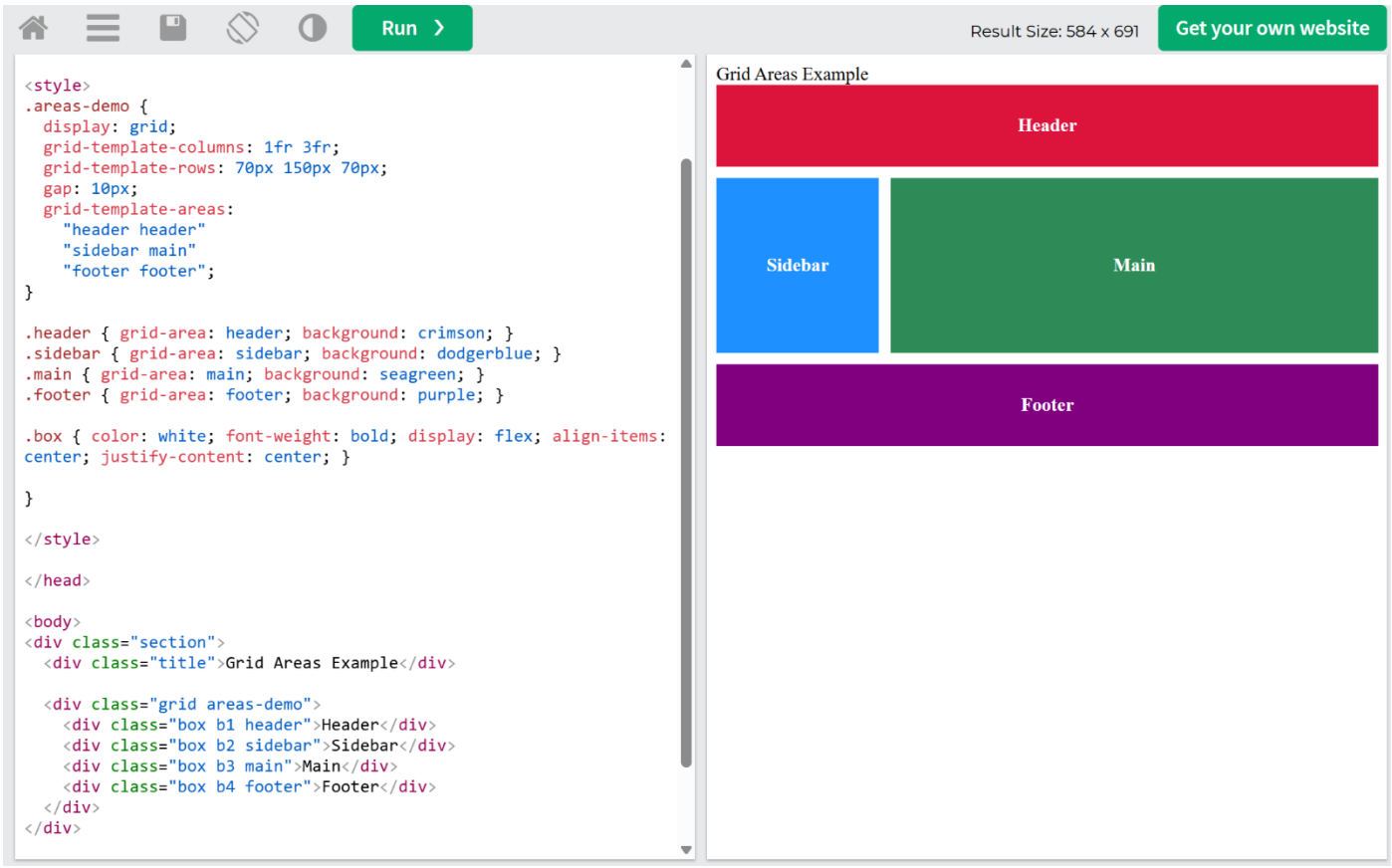


Grid Areas (named layouts)

What are Grid Areas?

Instead of using numbers, you can give parts of the grid a name and assign items to those areas.

- **Makes your layout more readable and maintainable**
- **Great for complex page layouts like header, sidebar, main, footer**



CSS Backgrounds

The background property is actually **a shorthand** for several background properties.

---

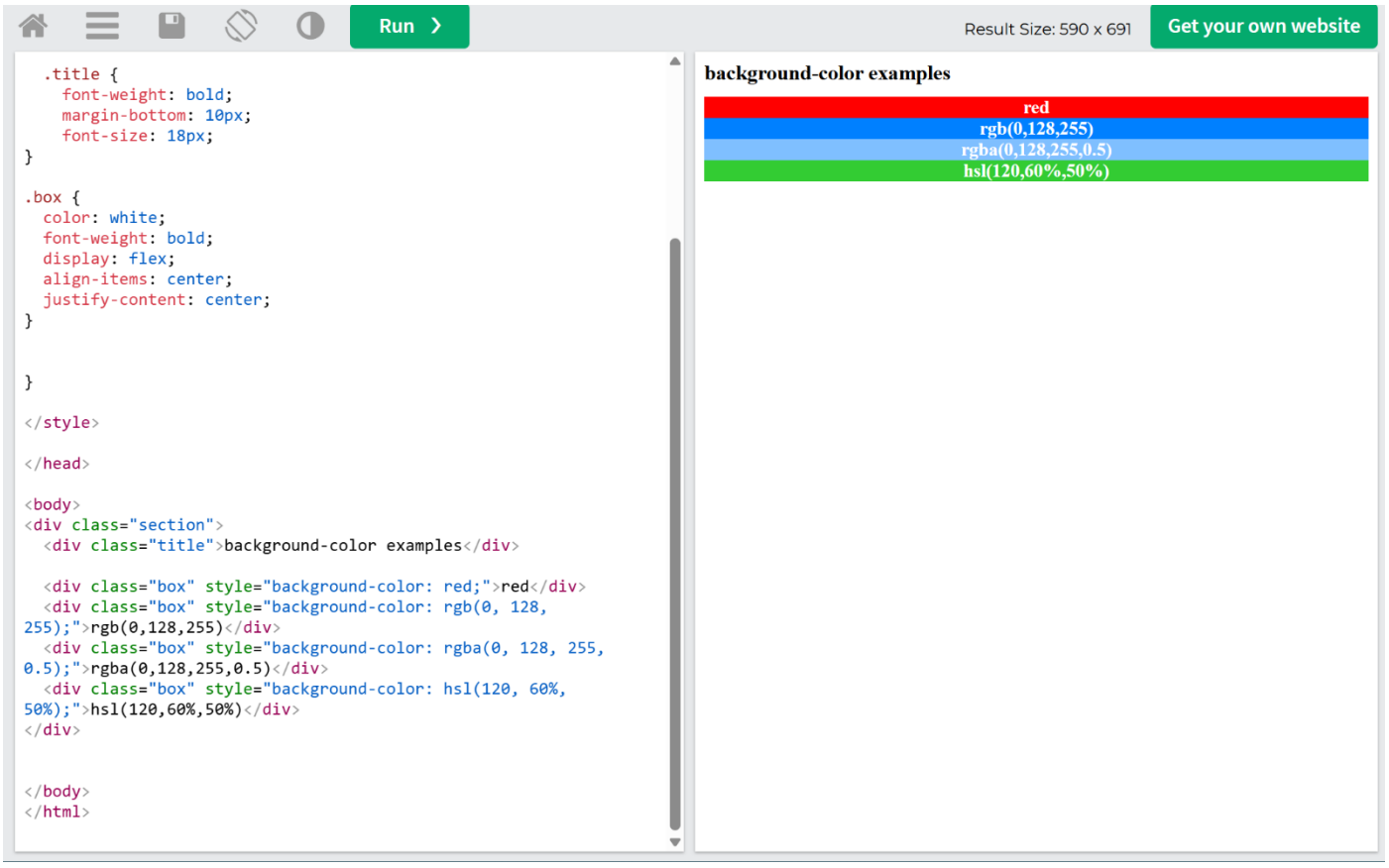
background-color

What it does

Sets the **background color** of an element.

Values

- Named colors: red, blue, yellow
- Hex: #ff0000
- RGB: rgb(255,0,0)
- RGBA: rgba(255,0,0,0.5)
- HSL: hsl(0, 100%, 50%)
- Transparent: transparent



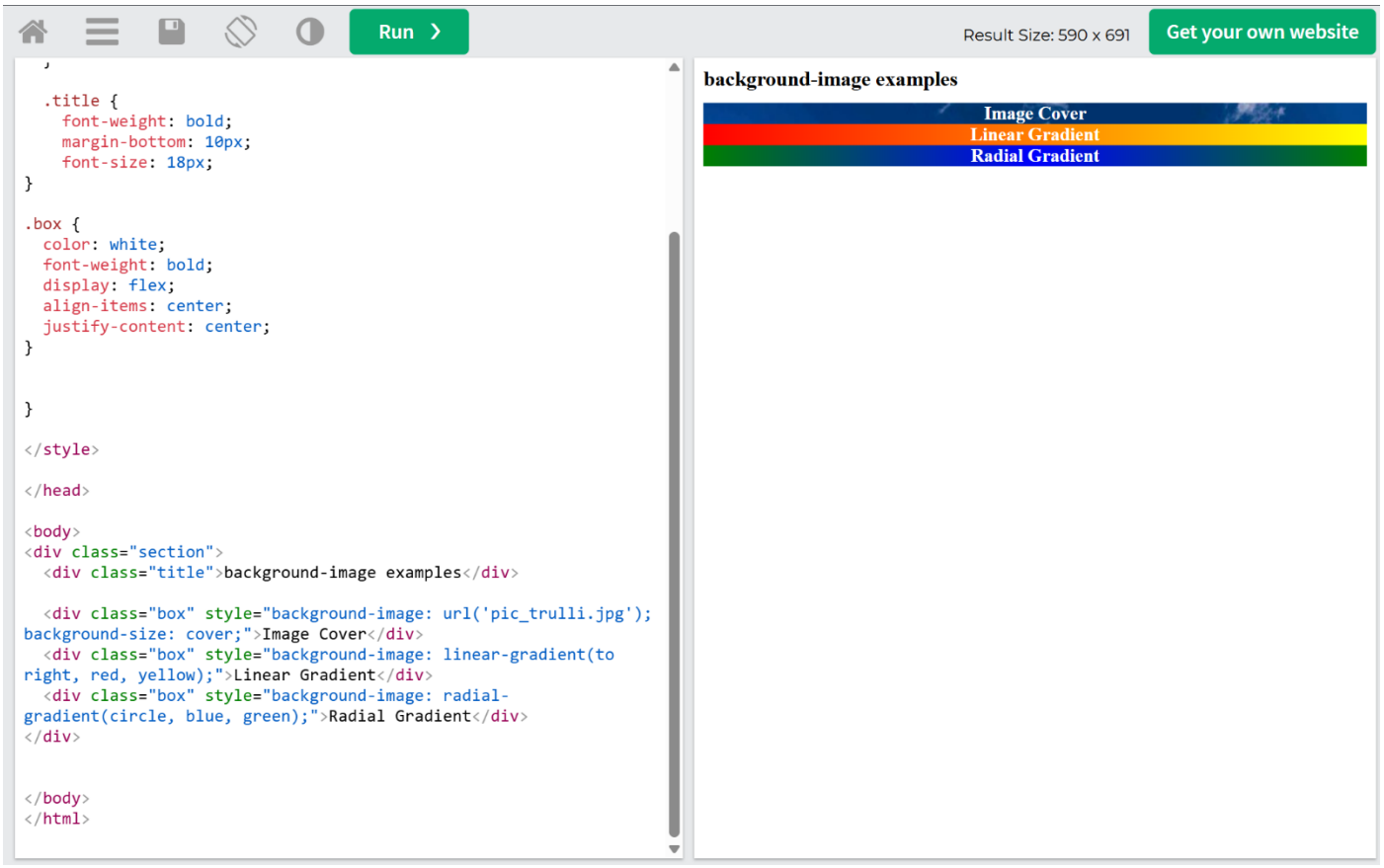
background-image

What it does

Sets an **image** as the background.

Values

- url("image.jpg")
- linear-gradient(to right or angle …, red, blue …)
- radial-gradient(circle or ellipse …, red, yellow, green …)
- Multiple images separated by commas



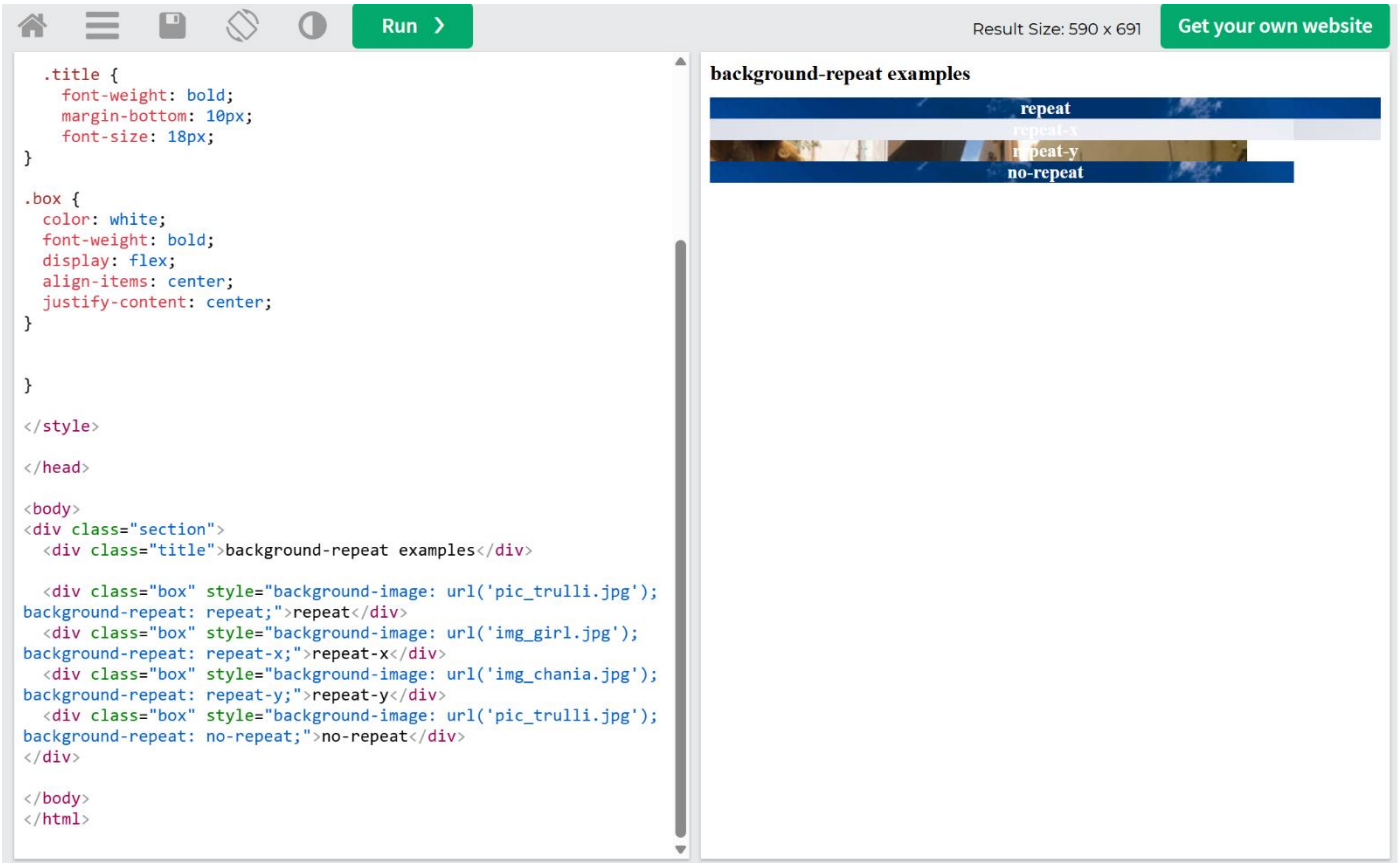
background-repeat

What it does

Controls **if/how the background image repeats**.

Values

- repeat (default) → both X & Y
- repeat-x → horizontal only
- repeat-y → vertical only
- no-repeat → no repeat



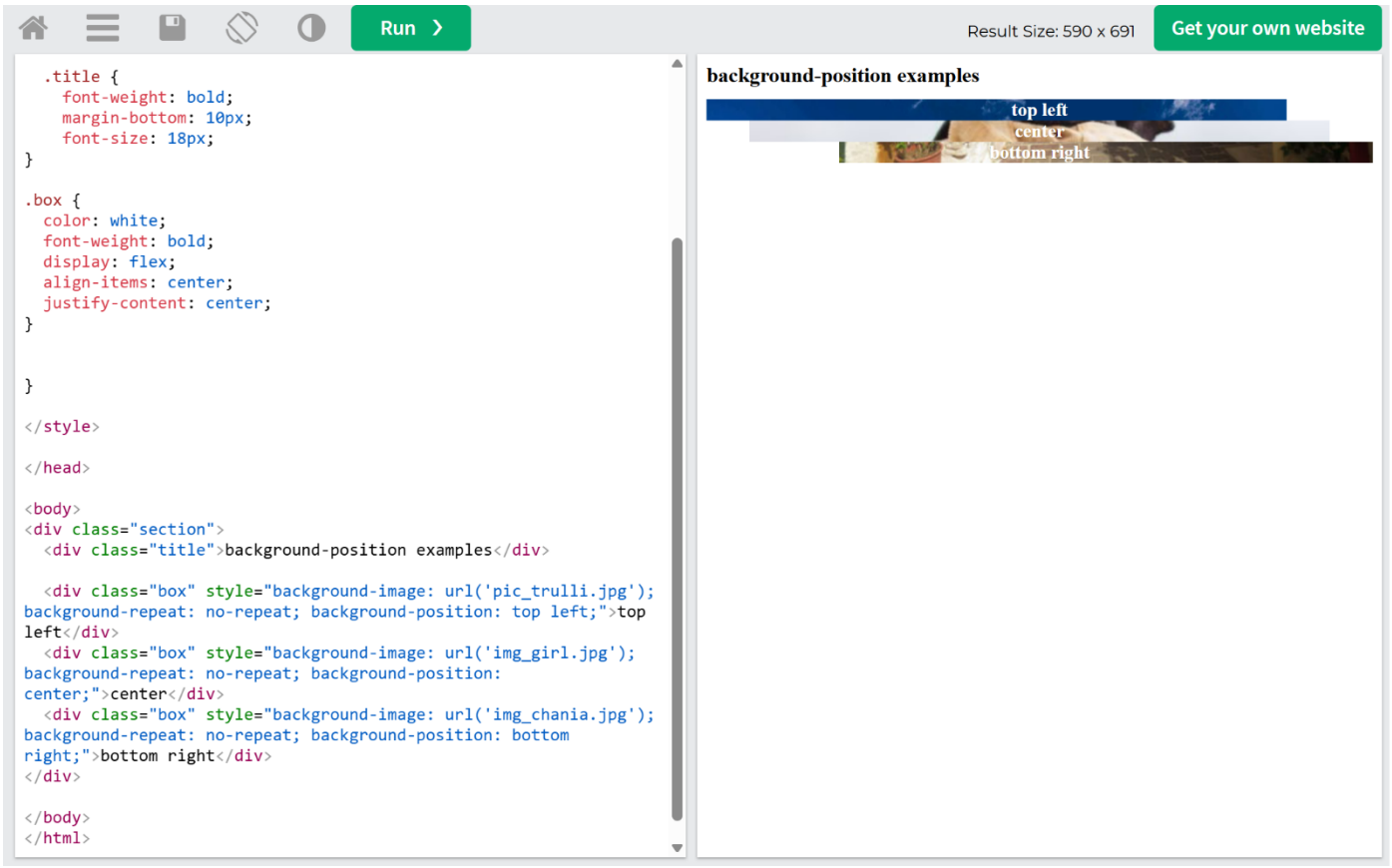
background-position

What it does

Sets **where the image starts** in the element.

Values

- Keywords: top, bottom, left, right, center, top left …
- Lengths: px, %
- Example: background-position: right 10px top 20px;



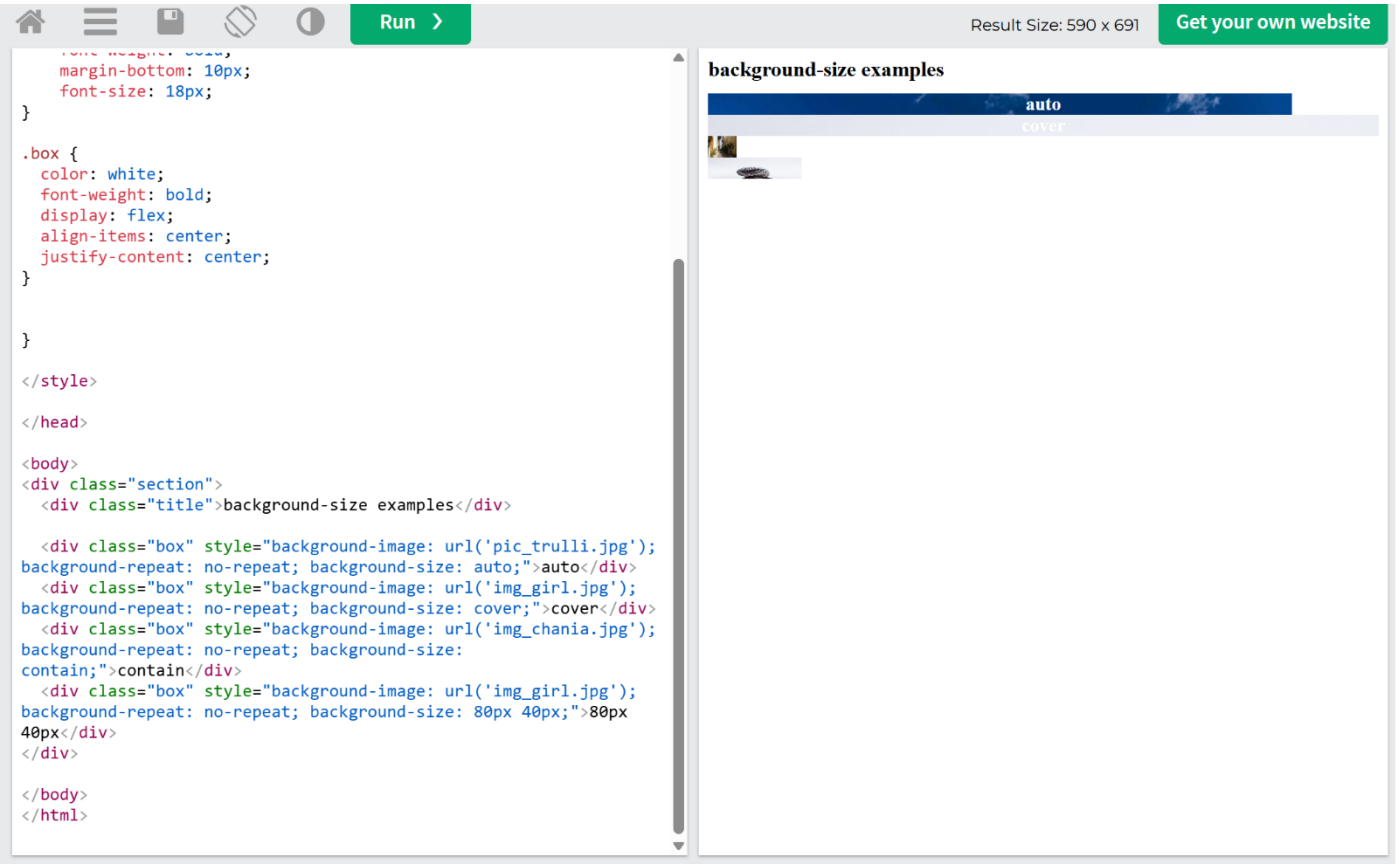
background-size

What it does

Controls the **size of the background image**.

Values

- auto (default) → original size
- cover → fills container, may crop
- contain → fits inside container
- Lengths: 100px 50px



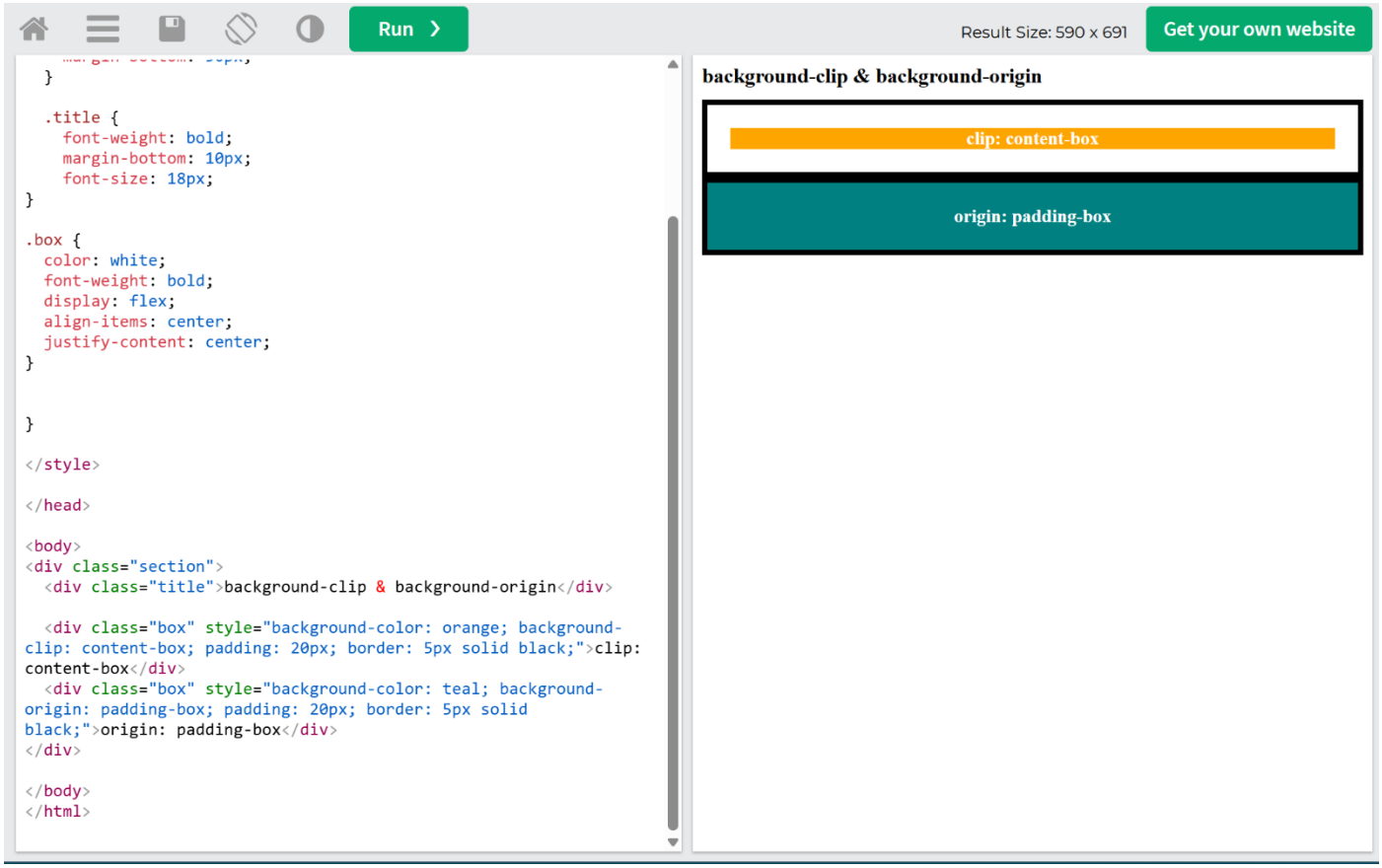
background-clip & background-origin

background-clip

- Determines **how far the background extends**
- Values: border-box(default), padding-box(clip outside padding), content-box(clip outside content)

background-origin

- Determines **where the background image starts**
- Values: border-box(default), padding-box(within border), content-box(from start of content)



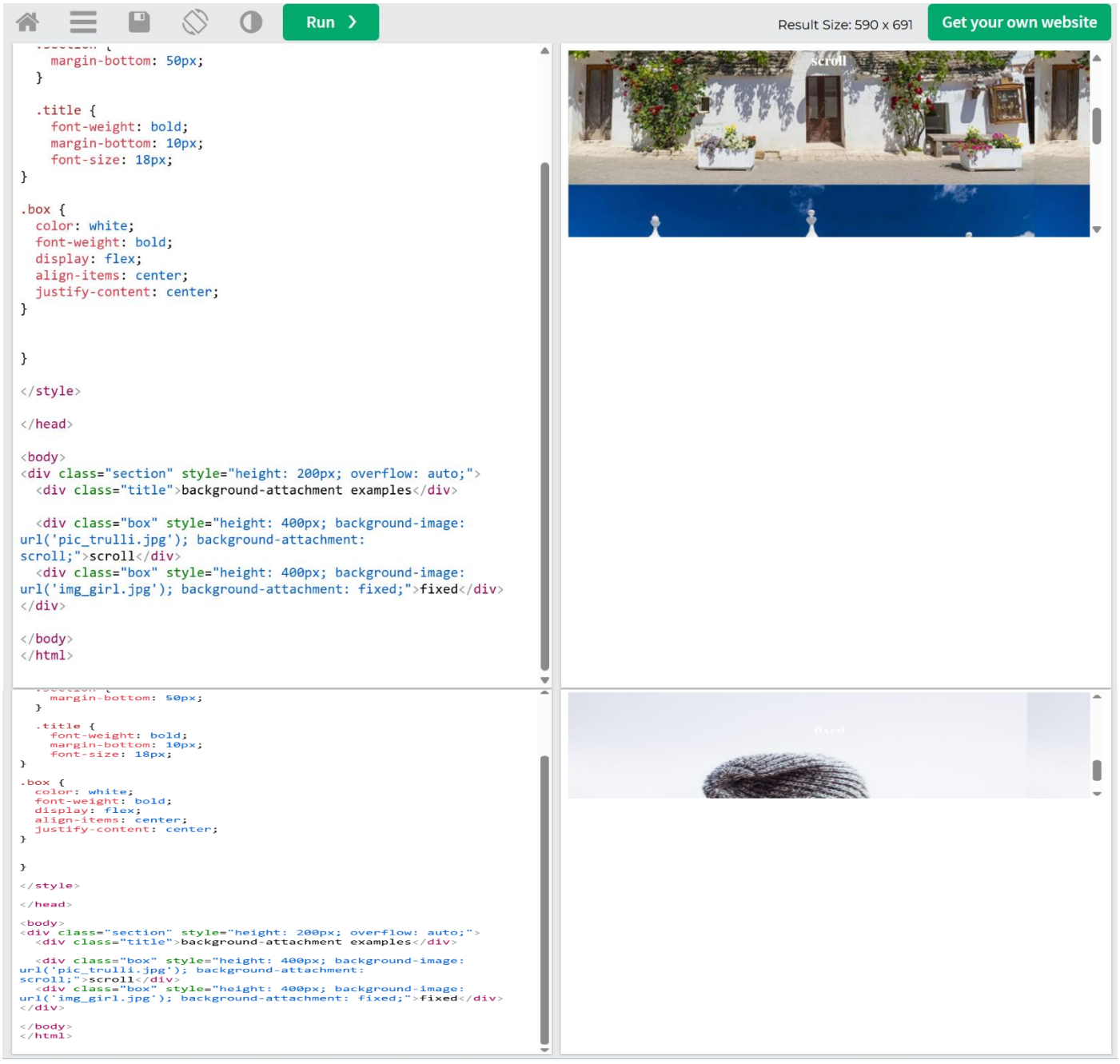
background-attachment

What it does

Controls **scrolling behavior** of background image.

Values

- scroll (default) → moves with content
- fixed → stays in viewport
- local → scrolls with element content

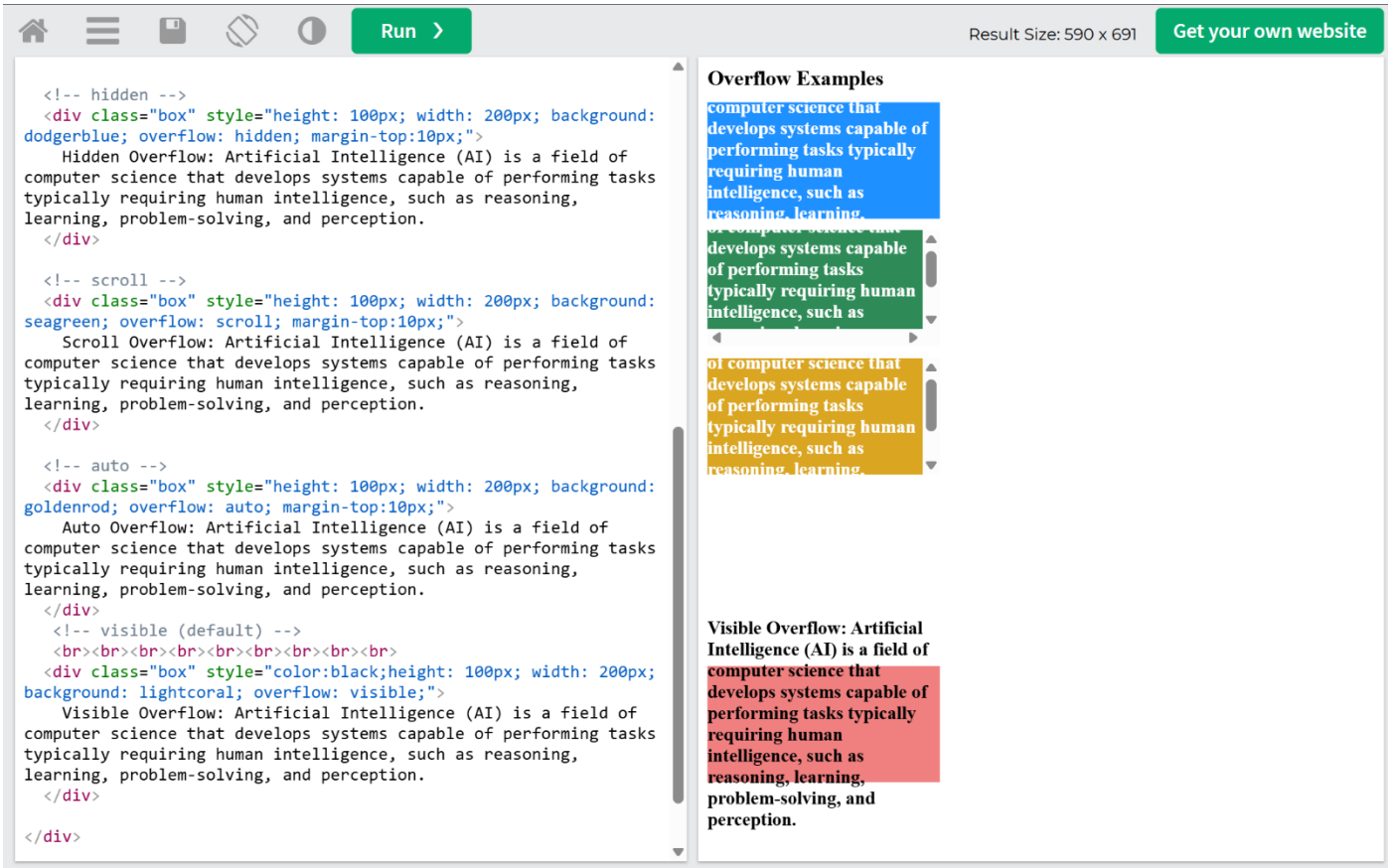


Overflow

Controls **what happens when content exceeds the element's box**.

Value	Meaning
visible	Default. Content **overflows** the box and is fully visible.
hidden	Content is **cut off** (clipped) at the boundary of the box.
scroll	Always shows **scrollbars** to access content.
auto	Shows **scrollbars only if needed**.

Note: overflow-x and overflow-y can control horizontal and vertical separately.

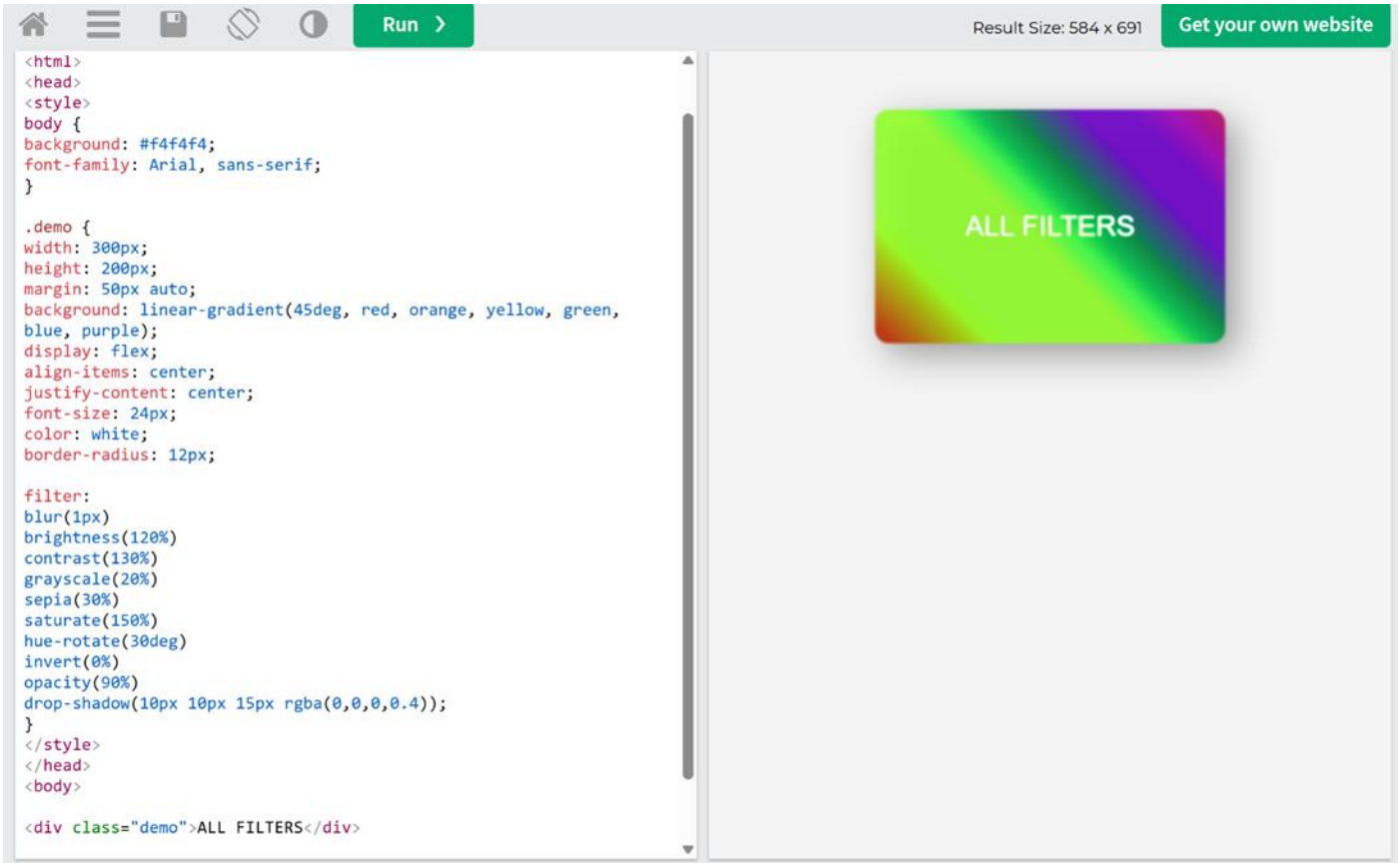


Filters

Filter	What it does
blur()	Blurs the element using a Gaussian blur
brightness()	Makes the element darker or brighter
contrast()	Increases or decreases difference between light and dark
grayscale()	Converts colors to grayscale
sepia()	Applies a warm brown tone
saturate()	Increases or decreases color intensity
hue-rotate()	Rotates colors around the color wheel
invert()	Inverts colors
opacity()	Controls transparency
drop-shadow()	Adds shadow following the element shape

Important notes

- Filters are applied **left → right**
- Order changes the result
- Filters do **not** affect layout or size




```
<html>
<head>
<style>
body {
background: #f4f4f4;
font-family: Arial, sans-serif;
}

.demo {
width: 300px;
height: 200px;
margin: 50px auto;
background: linear-gradient(45deg, red, orange, yellow, green,
blue, purple);
display: flex;
align-items: center;
justify-content: center;
font-size: 24px;
color: white;
border-radius: 12px;

filter:
blur(1px)
brightness(120%)
contrast(130%)
grayscale(20%)
sepia(30%)
saturate(150%)
hue-rotate(30deg)
invert(0%)
opacity(90%)
drop-shadow(10px 10px 15px rgba(0,0,0,0.4));
}
</style>
</head>
<body>

<div class="demo">ALL FILTERS</div>
```

CSS Transforms, Transitions & Animations

This section covers **motion and interaction in CSS**. These properties do **not affect layout flow** — they only affect how elements are visually rendered.

Transforms

Transforms change the **shape, position, size, or rotation** of an element in 2D or 3D space.

transform

Applies one or more transform functions.

Possible functions

- translate(), translateX(), translateY(), translateZ()
- scale(), scaleX(), scaleY(), scaleZ()
- rotate(), rotateX(), rotateY(), rotateZ()
- skew(), skewX(), skewY(), skewZ()
- matrix() takes exactly six parameters, all of which are <number> values: matrix(a, b, c, d, tx, ty). The parameters correspond to specific transformation types:
    - **a and d**: Scaling factors for the X-axis (a) and Y-axis (d).
    - **b and c**: Skew factors for shearing along the Y-axis (b) and X-axis (c).
    - **tx and ty**: Translation (movement) along the X-axis (tx) and Y-axis (ty), effectively in pixels.

Transforms are applied **from right to left**.

---

transform-origin

Defines the point around which the transform occurs.

Values:

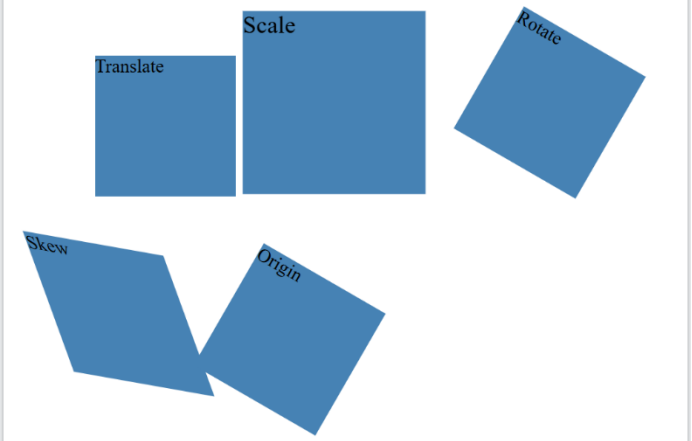
- center (default)
- top, bottom, left, right
- percentages
- px values

Run ›

Result Size: 584 x 691

Get your own website

```
<!DOCTYPE html>
<html>
<head>
<style>
.box {
width: 120px;
height: 120px;
background: steelblue;
margin: 30px;
display: inline-block;
}

.translate { transform: translate(40px, 20px); }
.scale { transform: scale(1.3); }
.rotate { transform: rotate(30deg); }
.skew { transform: skew(20deg, 10deg); }
.matrix{tranform: matrix;}

.origin {
transform-origin: top left;
transform: rotate(30deg);
}
</style>
</head>
<body>
<div class="box translate">Translate</div>
<div class="box scale">Scale</div>
<div class="box rotate">Rotate</div>
<div class="box skew">Skew</div>
<div class="box origin">Origin</div>
<div class="box matrix">Matrix</div>
</body>
</html>
```



3D Transforms

CSS 3D transforms allow elements to move and rotate in **three-dimensional space**, creating depth and realistic UI effects like card flips and cubes.

---

perspective

Defines the **distance between the viewer and the element** (camera depth).

What it does

- Controls strength of 3D depth
- Smaller value → stronger 3D distortion
- Larger value → flatter appearance

Values

- none (default)
- Length values (px only)

---

perspective-origin

Defines **where the viewer is positioned**.

What it does

- Moves the vanishing point
- Changes how depth tilts visually

Values

- center (default)
- left | right | top | bottom
- Percentages
- Length values

transform-style

Controls whether child elements **preserve 3D positioning**.

What it does

- Allows nested elements to stay in 3D space

Values

- flat (default)
- preserve-3d

---

backface-visibility

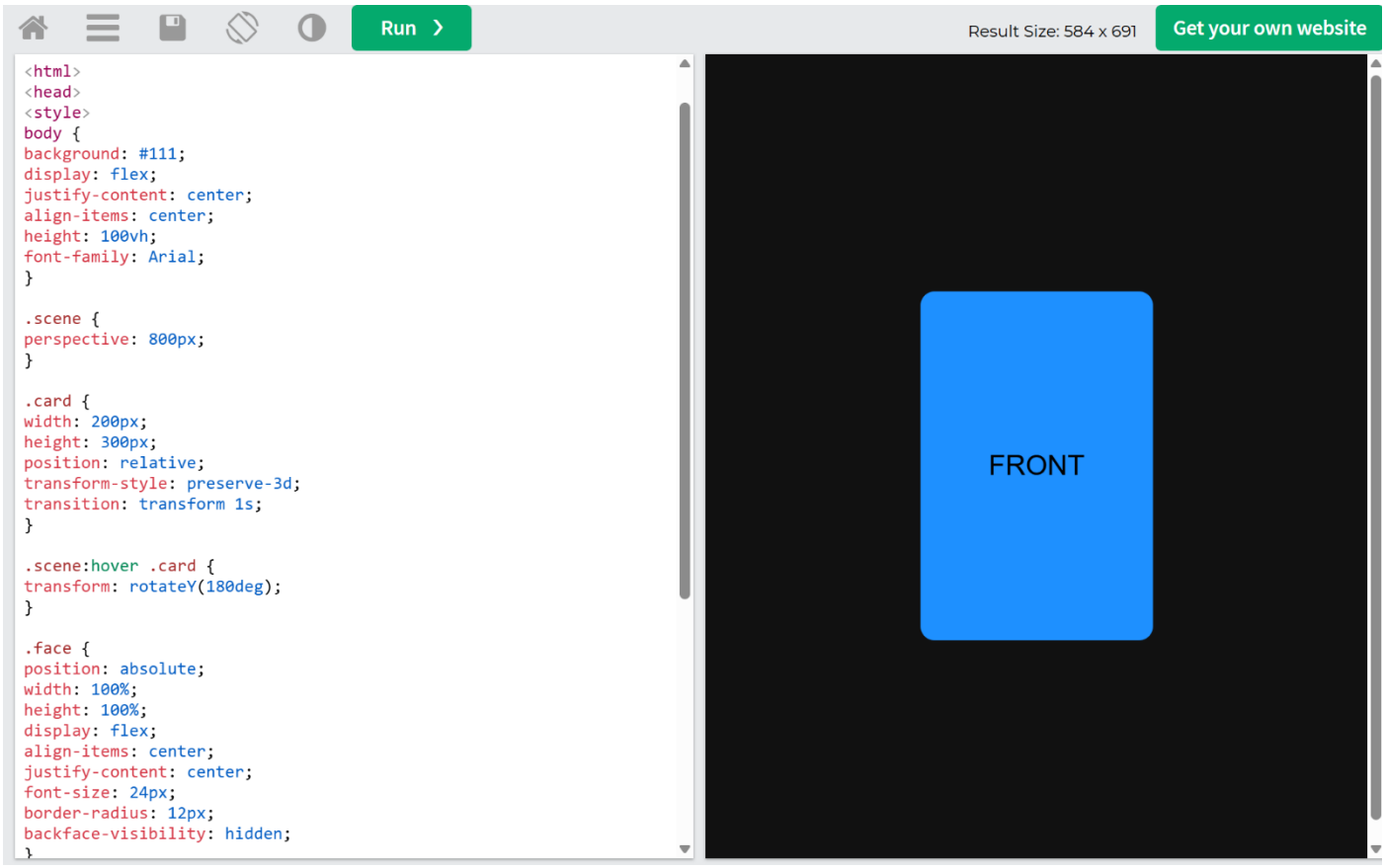
Controls visibility of the **back side** of a rotated element.

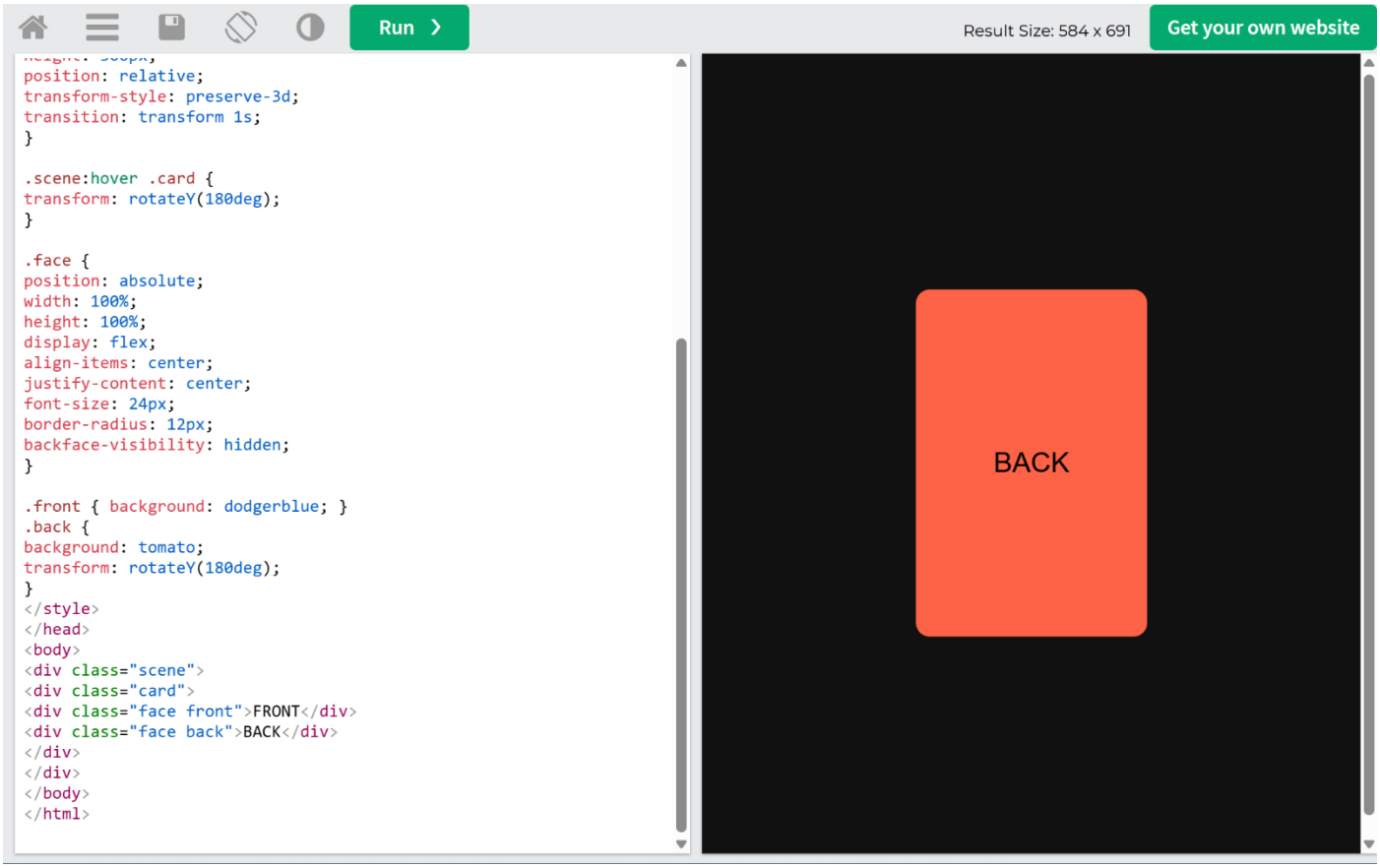
What it does

- Prevents mirrored text or reversed faces

Values

- visible (default)
- hidden




Run
Result Size: 584 x 691
Get your own website
position: relative;
transform-style: preserve-3d;
transition: transform 1s;
}
.scene:hover .card {
transform: rotateY(180deg);
}
.face {
position: absolute;
width: 100%;
height: 100%;
display: flex;
align-items: center;
justify-content: center;
font-size: 24px;
border-radius: 12px;
backface-visibility: hidden;
}
.front { background: dodgerblue; }
.back {
background: tomato;
transform: rotateY(180deg);
}
</style>
</head>
<body>

FRONT
BACK

</body>
</html>
BACK

Transitions

Transitions animate **property changes over time**.

transition

Shorthand for:

- transition-property
- transition-duration
- transition-timing-function
- transition-delay

---

transition-property

Which property to animate.

Values:

- all
- specific property (e.g. width, transform)

---

transition-duration

How long the transition takes.

Values:

- s
- ms

transition-timing-function

Controls speed curve.

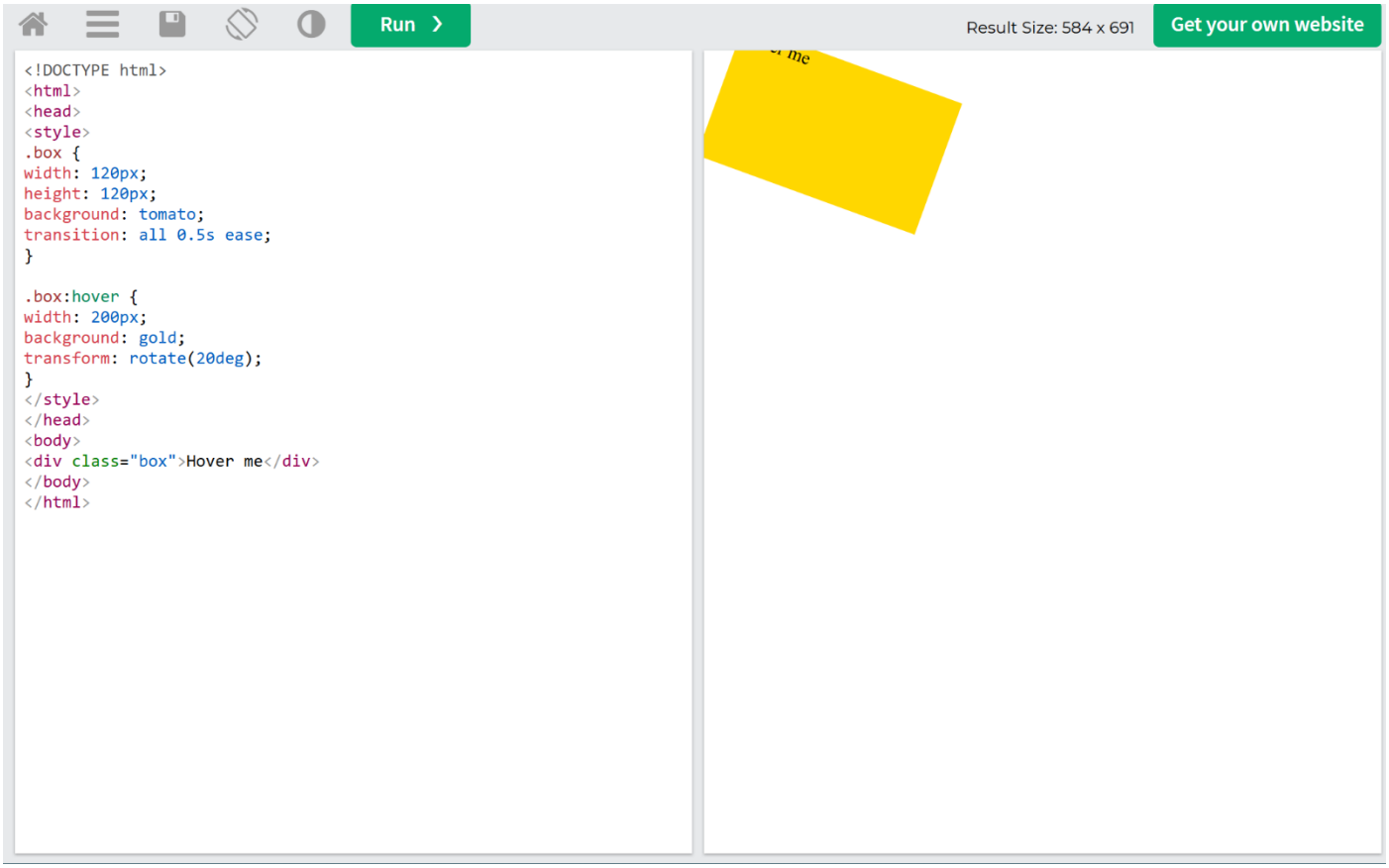
Values:

- ease (default)
- linear
- ease-in
- ease-out
- ease-in-out
- cubic-bezier()

---

transition-delay

Delay before animation starts.



Animations

Animations allow **automatic motion** using keyframes.

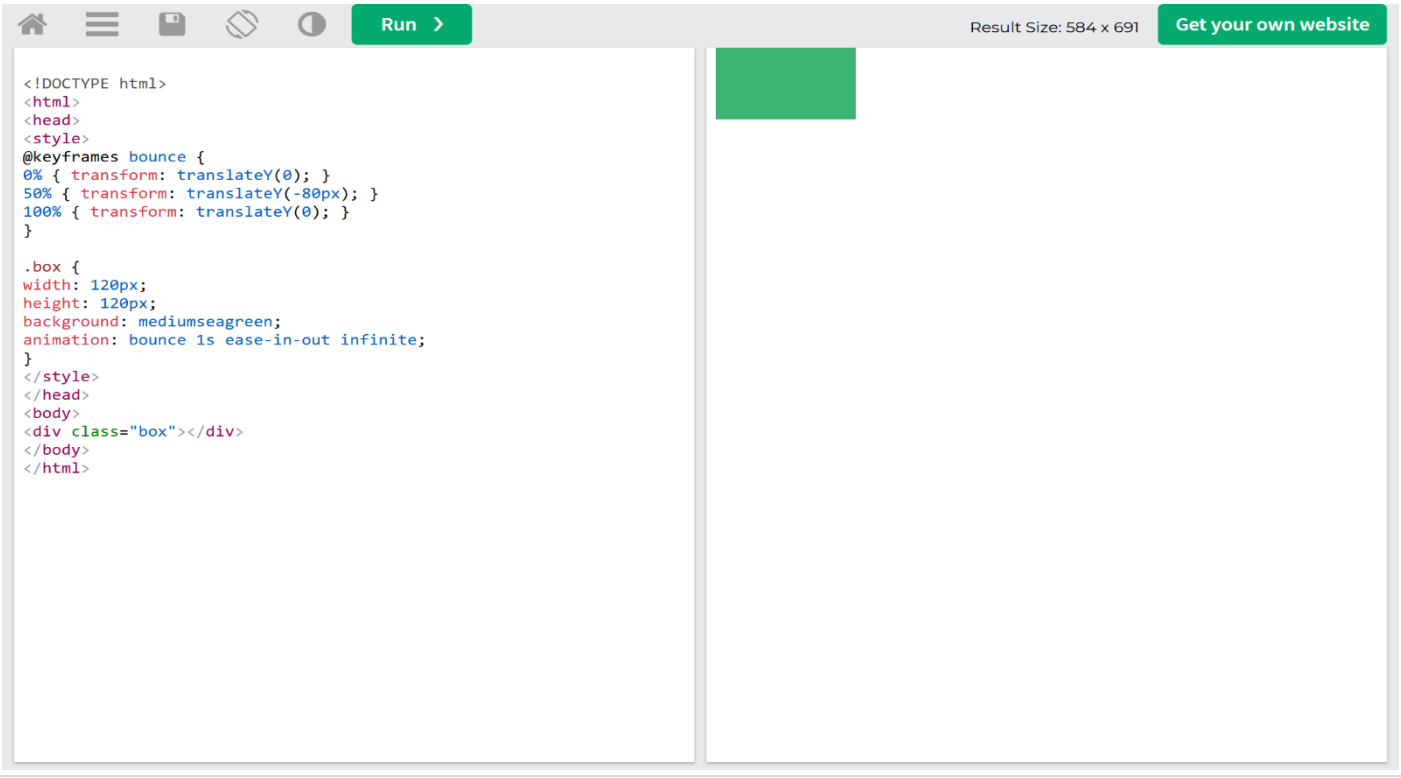
@keyframes

Defines animation steps.

```
@keyframes move {
from { transform: translateX(0); }
to { transform: translateX(200px); }
}
```

Animation Properties

- **animation-name**
- **animation-duration** (in seconds)
- **animation-timing-function** (ease, linear, ease-in, ease-out, ease-in-out, step-start, step-end, steps(n, start | end, cubic-bezier(x1, y1, x2, y2))
- **animation-delay** (in seconds)
- **animation-iteration-count** (any positive number)
- **animation-direction** (normal, reverse, alternate, alternate-reverse)
- **animation-fill-mode** (none, forwards, backwards, both)
- **animation-play-state** (running, paused)



Responsive Design

Responsive design means building layouts that **adapt to different screen sizes, devices, and orientations** without breaking usability or readability.

The goal is:

- One website
- One codebase
- Works everywhere

Core Principles of Responsive Design

- Fluid layouts (flexible widths)
- Flexible images & media
- Media queries
- Responsive typography
- Mobile-first thinking

Viewport

Controls how the page scales on mobile devices.

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

Without this, mobile browsers simulate desktop widths.

Responsive Units

Unit	Description
%	Relative to parent
vw	1% of viewport width
vh	1% of viewport height
vmin	Smaller of vw/vh
vmax	Larger of vw/vh
em	Relative to parent font-size
rem	Relative to root font-size

Responsive Images

```
img {
max-width: 100%;
height: auto;
}
```

Prevents images from overflowing containers.

Media Queries

Media queries apply CSS **based on device conditions**.

Syntax

```
@media (condition) {
/* CSS rules */
}
```

Breakpoints (Common Practice)

```
/* Mobile */
@media (max-width: 600px) {}

/* Tablet */
@media (min-width: 601px) and (max-width: 1024px) {}

/* Desktop */
@media (min-width: 1025px) {}
```

Responsive Typography

clamp(minimum, preferred, maximum)

Dynamically scales font sizes.

```
h1 {
font-size: clamp(1.5rem, 4vw, 3rem);
}
```

Orientation

```
@media (orientation: landscape) {
body {
background: #eee;
}
}
```

Hiding & Showing Content

```
.mobile-only {
display: block;
}

.desktop-only {
display: none;
}

@media (min-width: 768px) {
.mobile-only { display: none; }
.desktop-only { display: block; }
}
```

Conclusion

CSS is a powerful tool that enables developers to **create visually appealing and responsive web pages**. Understanding its full range of properties, behaviors, and interactions is essential for mastering web design. By combining **concept explanations, exhaustive value lists, and practical examples**, this document serves as a complete guide for learning and referencing CSS. It empowers developers to **solve layout challenges, style content effectively, and build professional web interfaces** with confidence.

Feedback & Contribution

This is a living document. If you have feedback, suggestions, or additional insights that could improve it, I welcome your input. Sharing ideas helps keep this material accurate, relevant, and valuable for everyone.

Copyright & Usage

Acknowledgments

This document is authored and copyrighted by **[Youssef Amgad Elkhatib]**.
Special thanks to the contributors who provided valuable feedback and suggestions for improvements:

Name	Contribution