CSS: Cascading Style Sheets

CPIT-405 Web Applications

Khalid Alharbi, PhD

Last updated: Fall 24

Table of Contents

Table of Contents

Introduction

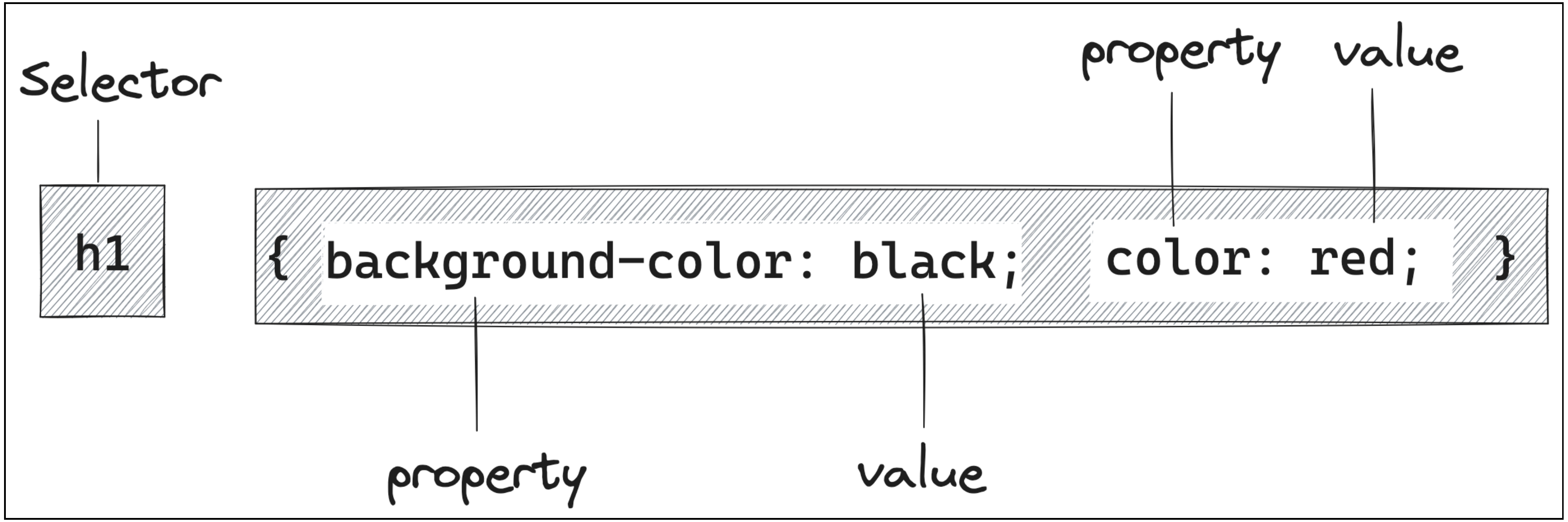
- **Cascading Style Sheets (CSS)** is a style sheet language used for specifying the presentation and styling of HTML documents.
- CSS is designed to support the principle of separation of content and presentation, including layout, colors, and fonts.
- HTML is utilized for structuring content, while CSS is employed for styling.
  - HTML is primarily dedicated to defining the structure of your document (e.g., headings, paragraphs, lists, links, etc.).
  - CSS is primarily dedicated to defining the visual styling of HTML elements (e.g., colors, fonts, layouts, animations, etc.).
- The cascade is the core of CSS and refers to the algorithm for solving conflicts where multiple CSS rules apply to an HTML element.
- Each web browser uses a layout engine to render web pages, and support for CSS functionality remains inconsistent between browsers.

History

- **1994**: CSS was proposed by Håkon Wium Lie, a Norwegian web pioneer and CTO of Opera Software, the company behind the Opera browser.
  - It was proposed as a way to separate content and presentation.
- **1996**: The first version of CSS was invented.
- **1998**: CSS 2 specification was released.
- **1999**: CSS 3 specification was released with the goal to modularize the CSS specification.
  - CSS no longer uses versions for the whole specs. Instead, it evolves into modules and levels (e.g, CSS Grid Layout Level 1, CSS Grid Layout Level 2, etc.)
- **2006-2009**: CSS preprocessors like `Sass` and `Less` released in response to stylesheets getting larger, more complex, and harder to maintain.
- **2011-Present**: Modern frameworks like Bootstrap, Foundation (2011), Semantic UI (2013), and Tailwind CSS (2017) received wide adoption.
- **2015-Present**: CSS continues to evolve with new features like animations, flexbox, and grid layout.

CSS Fundamentals

Basic Structure



```
h1 { background-color: black; color: red; }
```

Adding CSS to HTML

There are three main ways to apply CSS to an HTML document.

1. External CSS

```
<link rel="stylesheet" href="/path/to/style.css">
```

2. Internal CSS

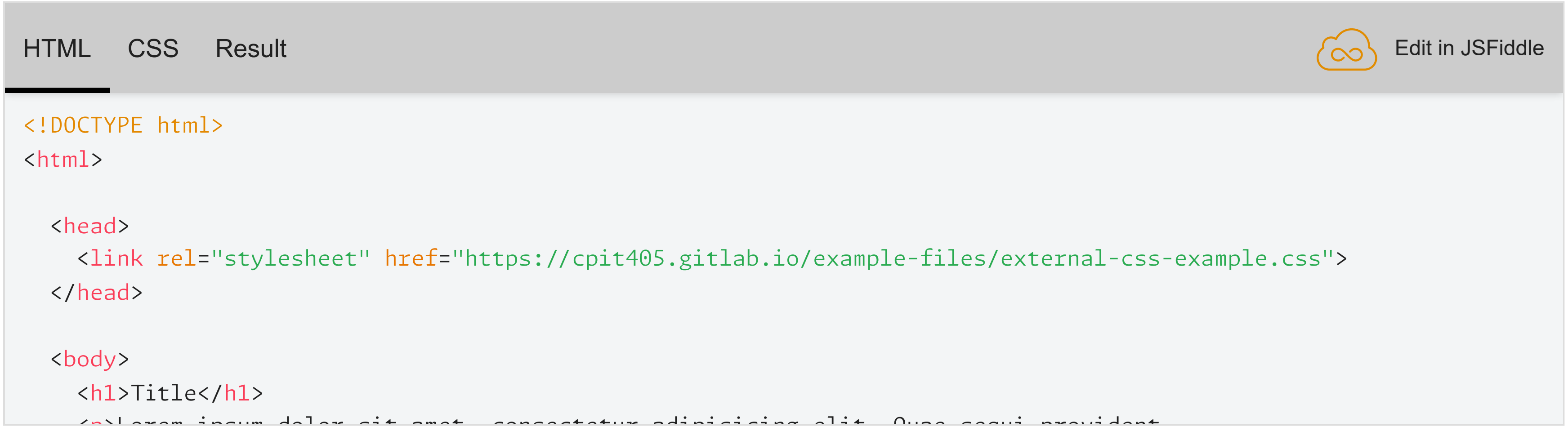
```
<head><style>body{width: 50%;}</style></head>
```

3. Inline CSS

```
<p style="color:red">foo</p>
```

External CSS

- External styles are defined within the `<link>` element, which is used to link external resource to the current HTML document.
- The `<link>` element is defined inside the `<head>` section of the HTML page, with an `href` attribute for the path to the stylesheet, and a `rel` attribute with a value of `stylesheet`:



```
<!DOCTYPE html>
<html>

  <head>
    <link rel="stylesheet" href="https://cpit405.gitlab.io/example-files/external-css-example.css">
  </head>

  <body>
    <h1>Title</h1>
    <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Quae sequi provident
```

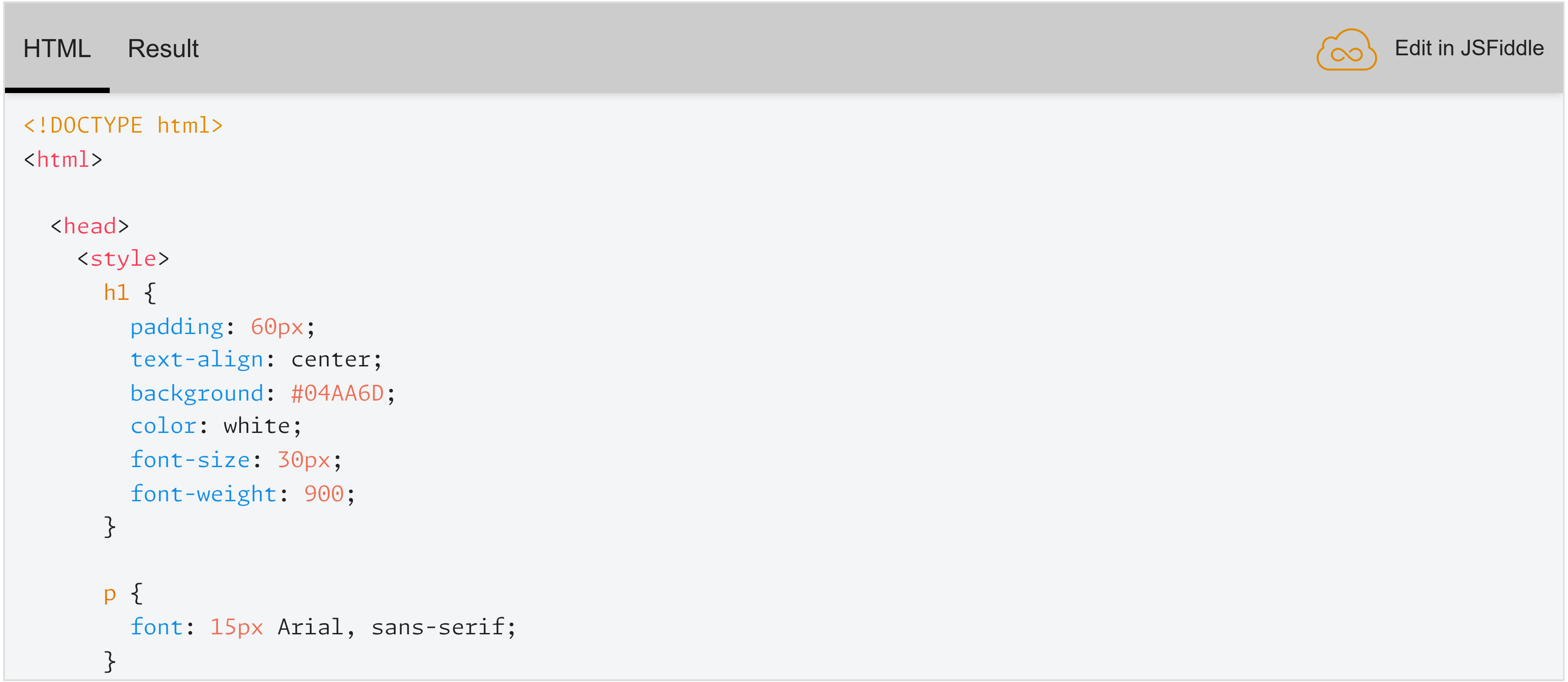
External CSS with media queries

- We can also conditionally load an external stylesheet with **media queries**.
- This works by setting a media type in the `media` attribute.
- In the following example, the external CSS file *print.css* will only be loaded if the media condition is true, which is a printer (i.e., for print preview and printing).

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" href="main.css">
    <link rel="stylesheet" href="print.css" media="print">
  </head>
    <body>
        <h1>Title...</h1>
        <p>Content...</p>
         <footer>
           <p>Khalid Alharbi</p>
           <p><a href="mailto:someemail@cpit405somedomain">Email</a></p>
         </footer>
    </body>
</html>
```

Internal CSS

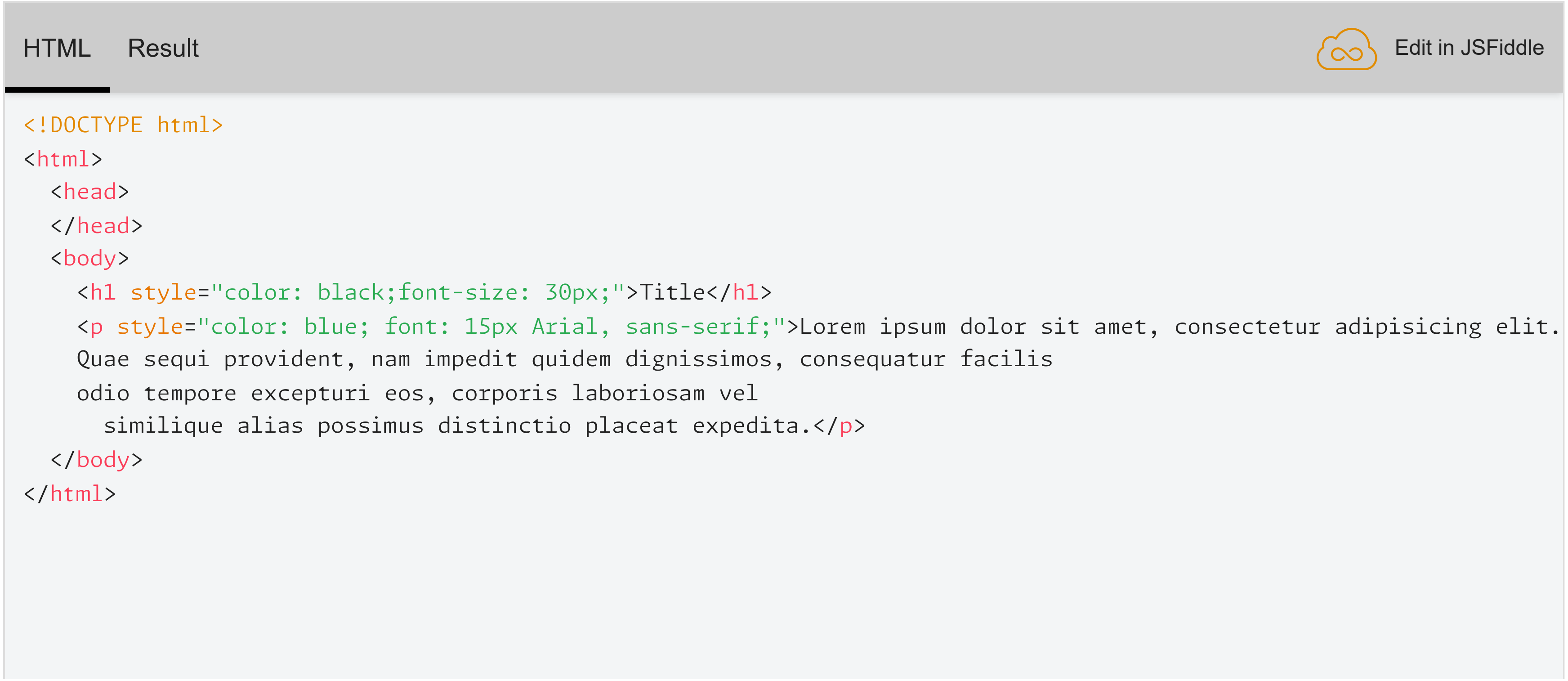
- Internal styles are defined within the `<style>` element, which is defined inside the `<head>` section of the HTML page:



```
<!DOCTYPE html>
<html>

  <head>
    <style>
      h1 {
        padding: 60px;
        text-align: center;
        background: #04AA6D;
        color: white;
        font-size: 30px;
        font-weight: 900;
      }

      p {
        font: 15px Arial, sans-serif;
      }
```

Inline CSS

- An inline style is defined within a single HTML element using the HTML attribute `style`.



```
<!DOCTYPE html>
<html>
  <head>
  </head>
  <body>
    <h1 style="color: black;font-size: 30px;">Title</h1>
    <p style="color: blue; font: 15px Arial, sans-serif;">Lorem ipsum dolor sit amet, consectetur adipisicing elit.
    Quae sequi provident, nam impedit quidem dignissimos, consequatur facilis
    odio tempore excepturi eos, corporis laboriosam vel
      similique alias possimus distinctio placeat expedita.</p>
  </body>
</html>
```

Browser Default / User-agent stylesheets

- The web browser may apply a default style when rendering an HTML page.
- Every browser has a set of default styles for HTML elements. These are known as the browser’s default styles or user agent styles.
- These default styles can vary from one browser to another. This is why the same webpage can look different when viewed in different browsers.
- Developers can override these default styles by specifying their own styles in CSS.
- It’s important to be aware of these default styles as they can affect the layout and appearance of your webpage, especially if you’re not explicitly setting all styles yourself.

User stylesheets

- Although not common, some browsers provide the capability for users to customize styles according to their preferences by using a personalized user stylesheet.
- For example, in FireFox, the configuration editor (`about:config` page) lists some settings known as advanced preferences.
- Changing advanced preferences can affect Browser’s stability and security. This is recommended for advanced users only.
- Some browser extensions offer the functionality to modify styles for enhanced accessibility or to create a more personalized user experience.

Cascading Order in CSS

- CSS stands for Cascading Stylesheets.
- The cascade is the algorithm used to resolve conflicts when multiple CSS rules apply to a single HTML element.
- The following steps apply to the cascading algorithm:

1. **Relevance:** first the algorithm filters all the rules from the different sources to keep only the rules that apply to a given element and the appropriate `media`.
2. **Origin and importance:** second the algorithm sorts elements by their importance, whether they have `important` flag.
3. **Specificity:** the specificity of a rule is considered when choosing one value or another.
4. **Scoping proximity:** styles defined closer to an element have more influence than those defined farther away.
5. **Order of appearance:** If you add multiple styles with the same power and location, the last one you write will actually be used.

More on Origin and importance

When there is more than one style defined in an HTML document, the style will be often applied as follows:

1. Inline style will take a higher priority.
2. External and internal style sheets will be applied depending on which one is defined **last** in the head section.
3. Browser default style.

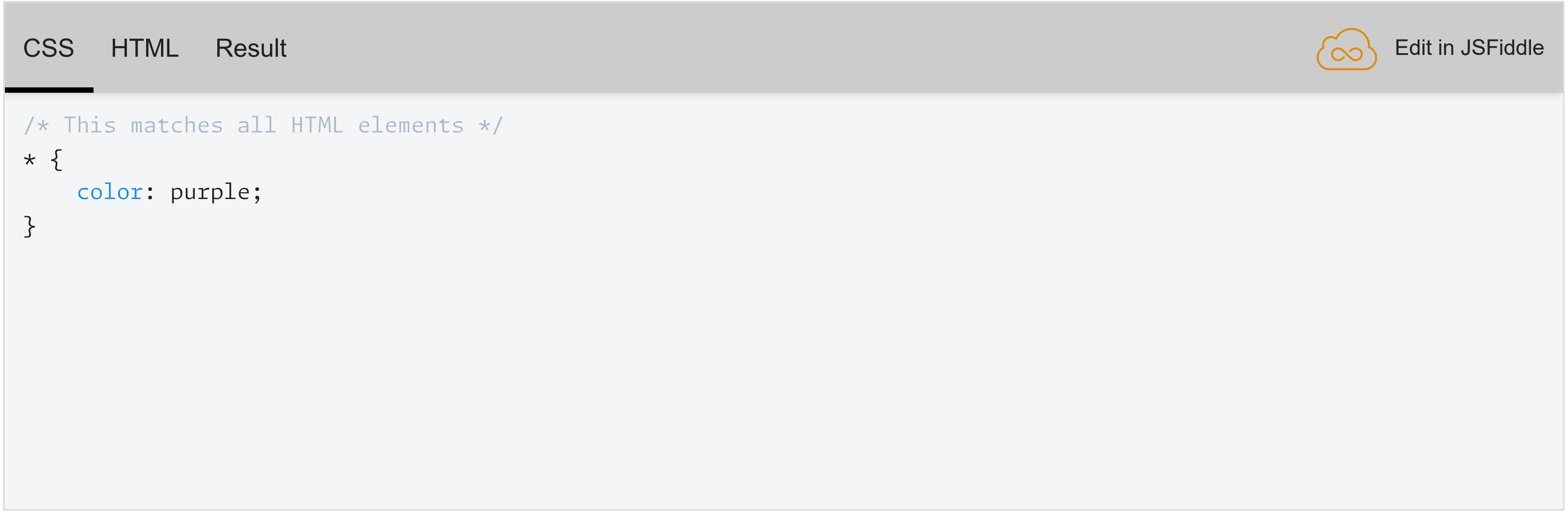
CSS Selectors

1. Universal Selector `*`
2. Element selector `element`
3. ID selector `#id`
4. Class Selector `.className`
5. Attribute Selector `element[attribute]`
6. Combinator selectors
   - Descendant `selector1 selector2`
   - Child `element1 > element2`
   - General sibling `selector ~ selector2`
   - Adjacent sibling `selector1 + selector2`
7. CSS Pseudo-classes
   - Visited Links `:visited`
   - Mouse is over elements `:hover`
   - Form input has received focus `:focus`.
   - Form input is required `:required`
   - Form input is optional `:optional`
   - First child of an element `:first-child`
   - Last child of an element `:last-child`
   - nth child of an element `:nth-child()`
8. CSS Pseudo-elements

1. Universal Selector

Syntax: `* { style properties }`

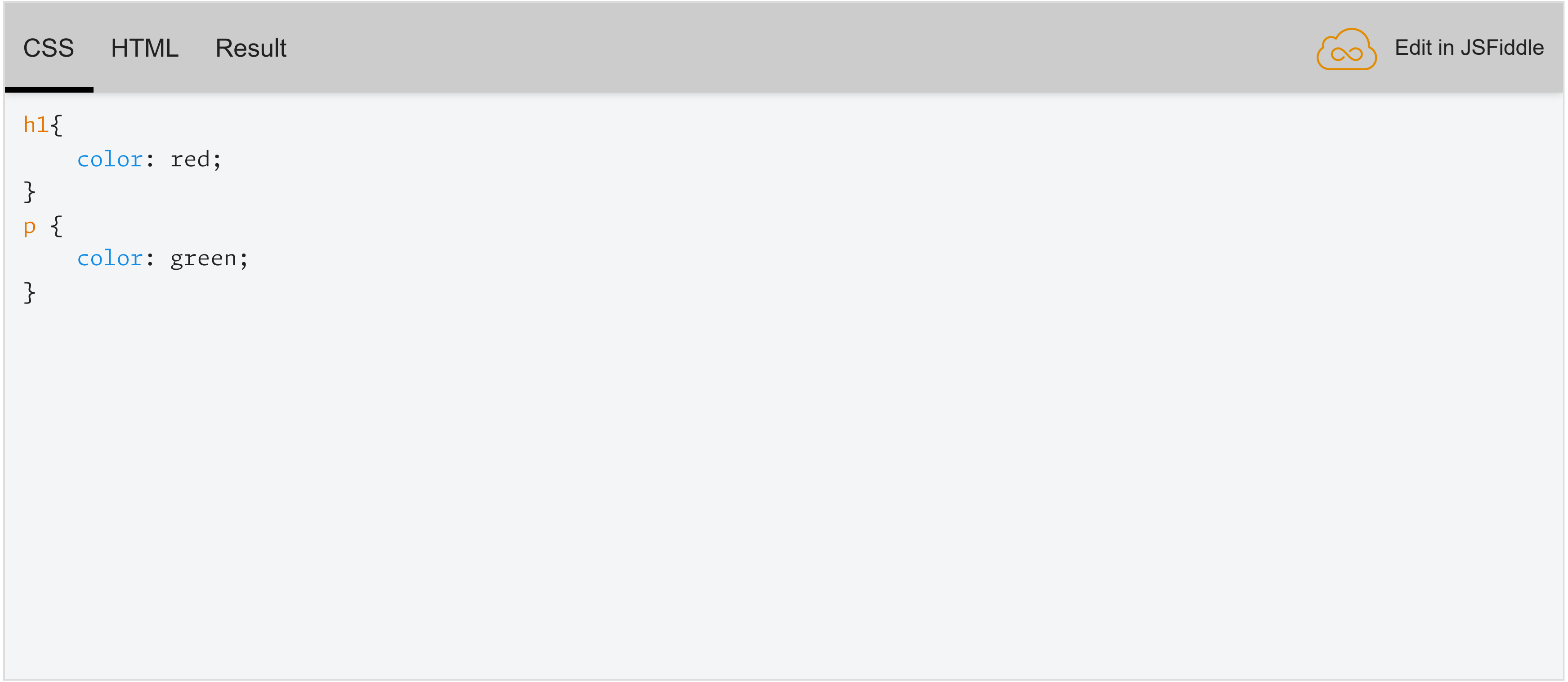
- The CSS universal selector `*` matches all HTML elements in the document.



```css
/* This matches all HTML elements */
* {
    color: purple;
}
```

2. Element selector

Syntax: `element { style properties }`

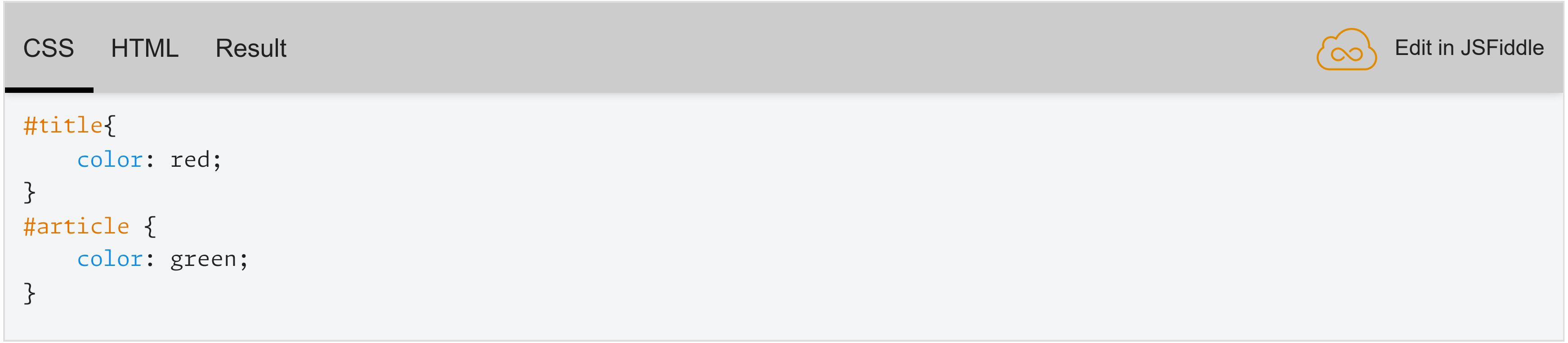


```css
h1{
    color: red;
}
p {
    color: green;
}
```

3. ID selectors

Syntax: `#id-value { style properties }`

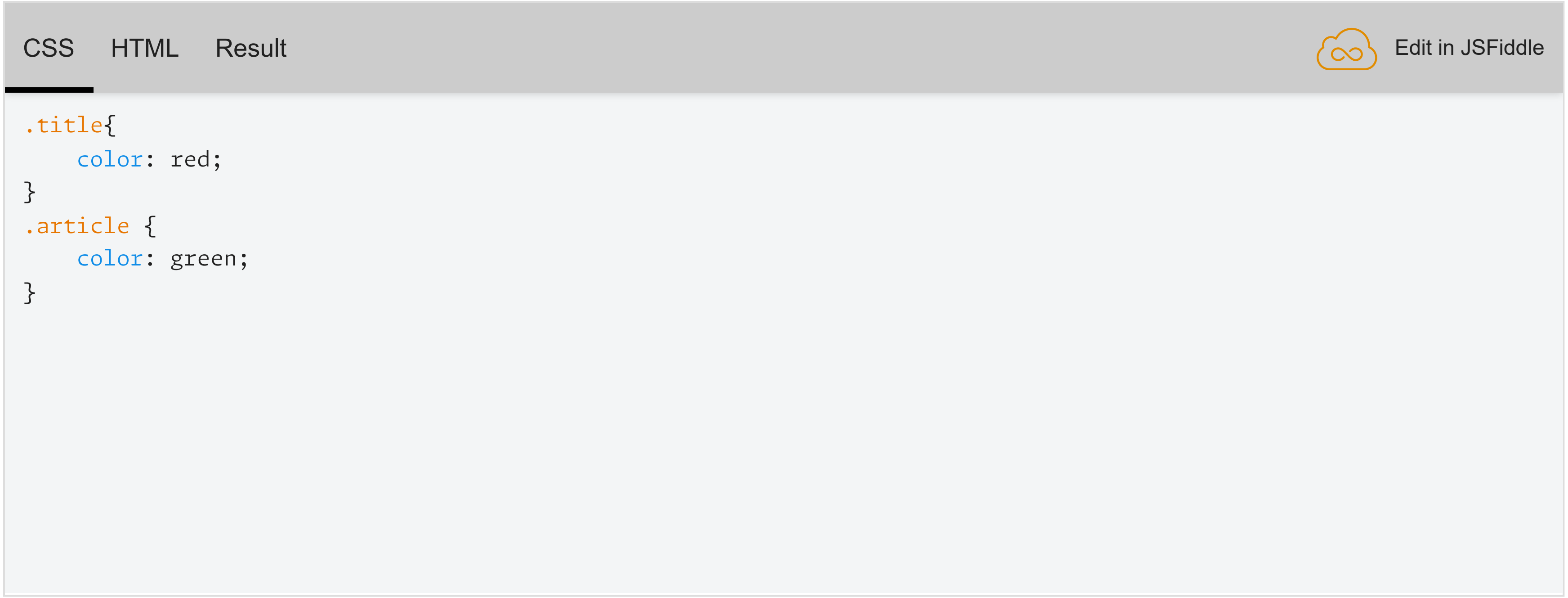
- In CSS, an ID selector is a name assigned to a specific style that is used to select an HTML element with the corresponding `id` attribute
- Each ID should be unique within a page and used only once.
- ID selectors have a higher specificity than class selectors, which means they will override conflicting styles from class selectors.



```css
#title{
    color: red;
}
#article {
    color: green;
}
```

4. Class Selector

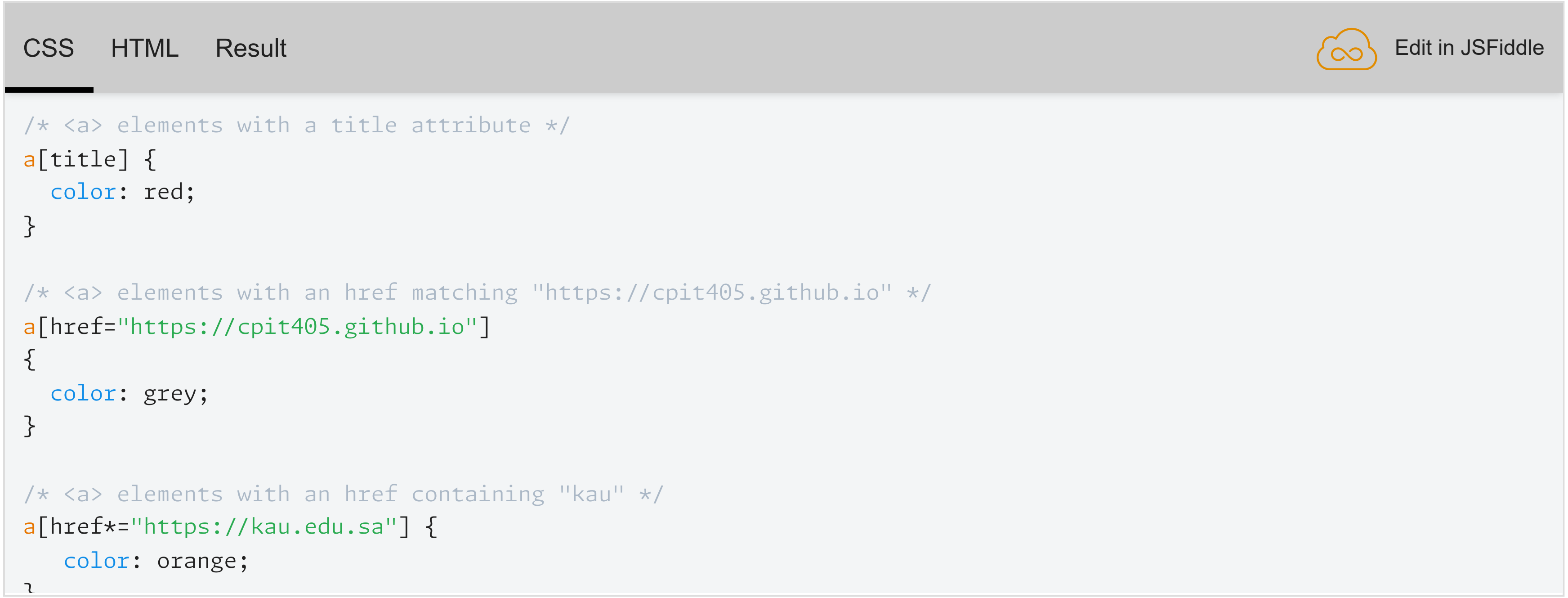
Syntax: `.className { style properties }`



```css
.title{
    color: red;
}
.article {
    color: green;
}
```

5. Attribute Selector (I)

Syntax: `element[attribute]{ style properties }`


CSS HTML Result Edit in JSFiddle

```
/* <a> elements with a title attribute */
a[title] {
  color: red;
}

/* <a> elements with an href matching "https://cpit405.github.io" */
a[href="https://cpit405.github.io"]
{
  color: grey;
}

/* <a> elements with an href containing "kau" */
a[href*="https://kau.edu.sa"] {
   color: orange;
```

5. Attribute Selectors (II)

Selector	Description	Example
`[attr]`	Matches elements that have the attr attribute declared	`input[required]`
`[attr=value]`	Matches elements with an attr attribute whose value is exactly value	`a[href="https://example.com"]`
`[attr~=value]`	Matches elements with an attr attribute whose value is exactly value, or contains value in its (space separated) list of values.	`p[class~="special"]`
`[attr^=value]`	Matches elements with an attr attribute, whose value begins with value.	`li[class^="box-"]`

5. Attribute Selectors (III)

Selector	Description	Example
`[attr$=value]`	Matches elements with an attr attribute whose value ends with value.	`li[class$="-box"]`
`[attr*=value]`	Matches elements with an attr attribute whose value contains value anywhere within the string.	`li[class*="box"]`

6. Combinator selectors

6.1 Descendant `selector1 selector2`

6.2 Child `element1 > element2`

6.3 General sibling `selector ~ selector2`

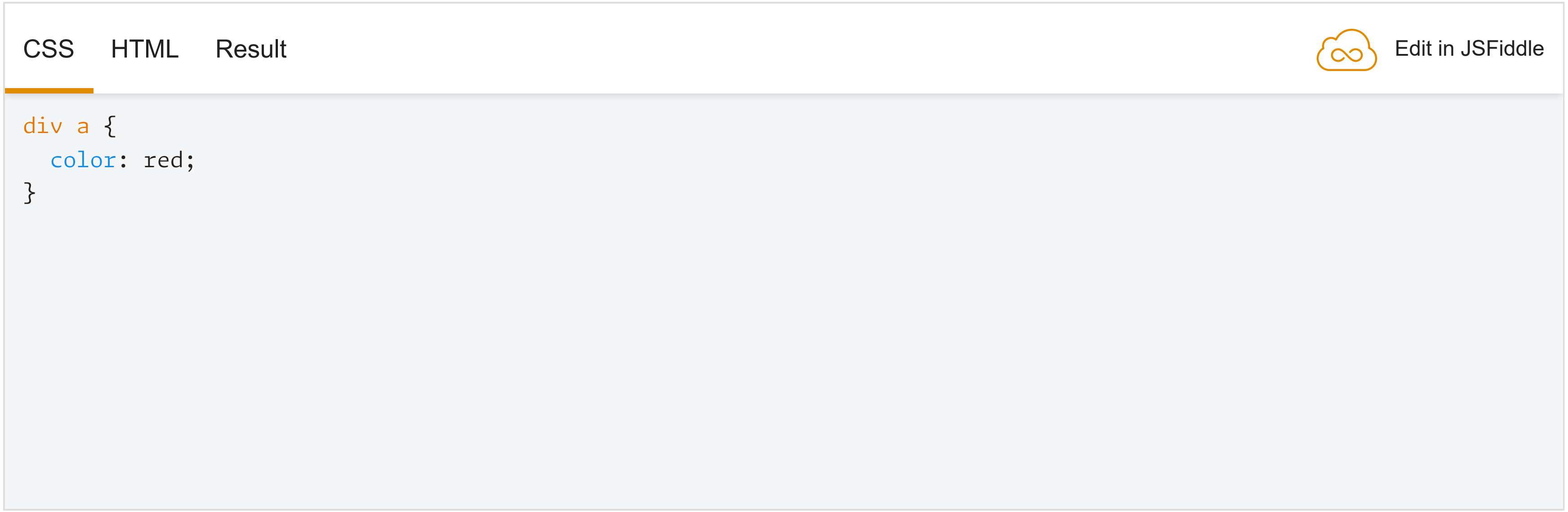
6.4 Adjacent sibling `selector1 + selector2`

6.1 Descendant combinator

Syntax: `element1 element2`

Matches the descendant element(s) of the first element.

Try it:

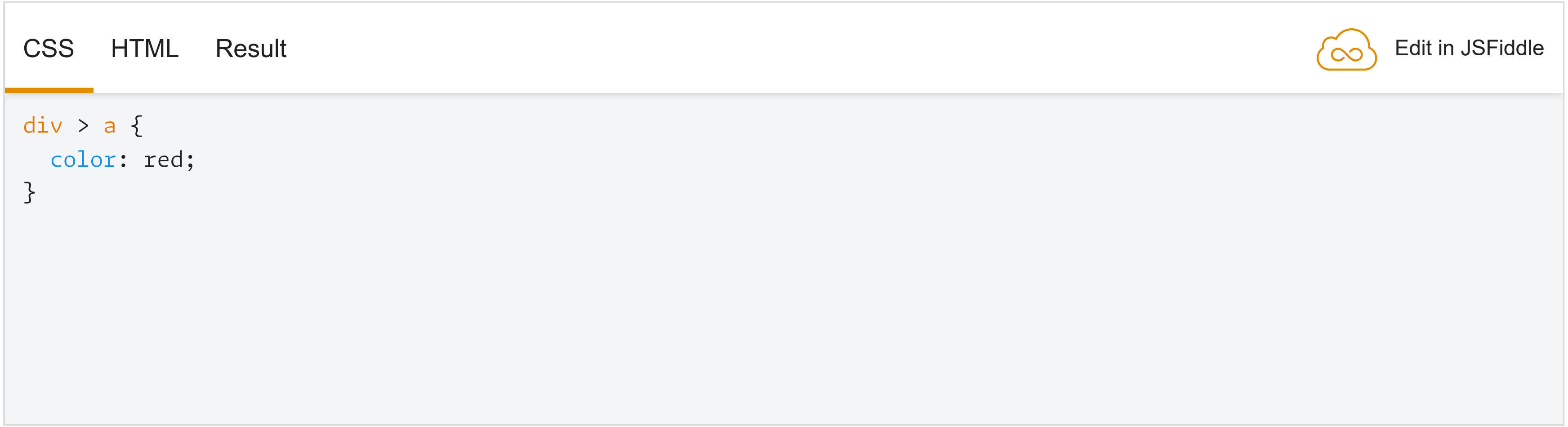


```css
div a {
  color: red;
}
```

6.2 Child combinator

Syntax: `element1 > element2`

Matches the direct child or children of the first element.

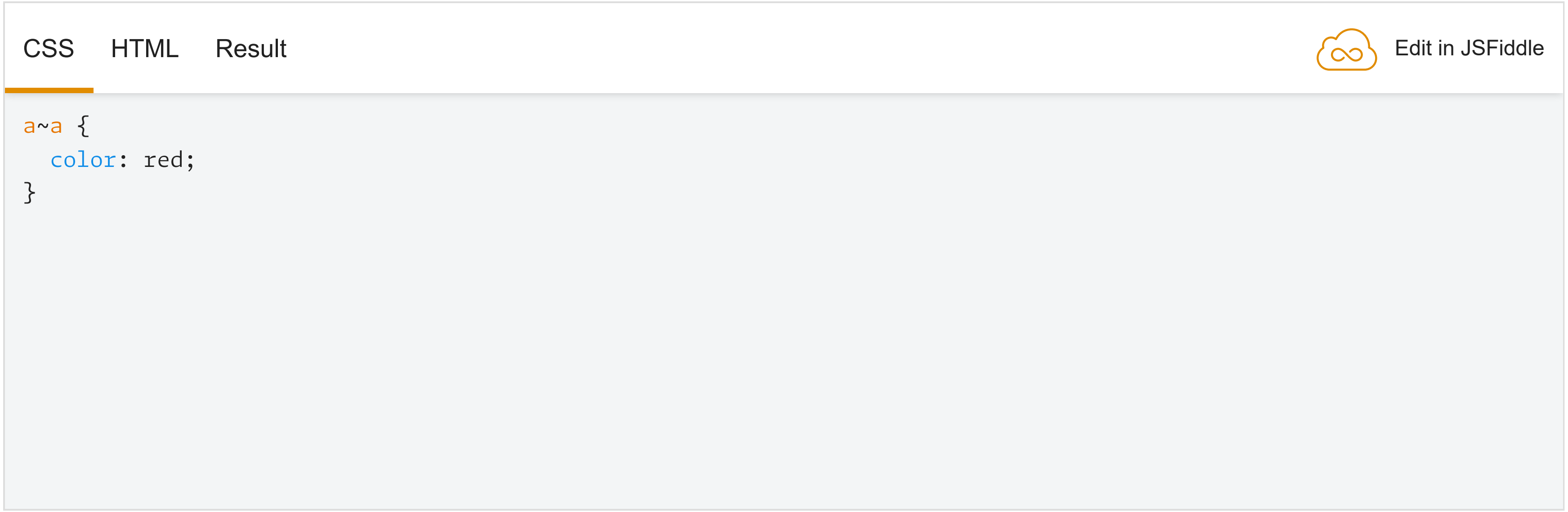


```
div > a {
  color: red;
}
```

6.3 General sibling combinator

Syntax: `element1 ~ element2`

Matches all siblings of the first element that are following the first element but not necessarily immediately.

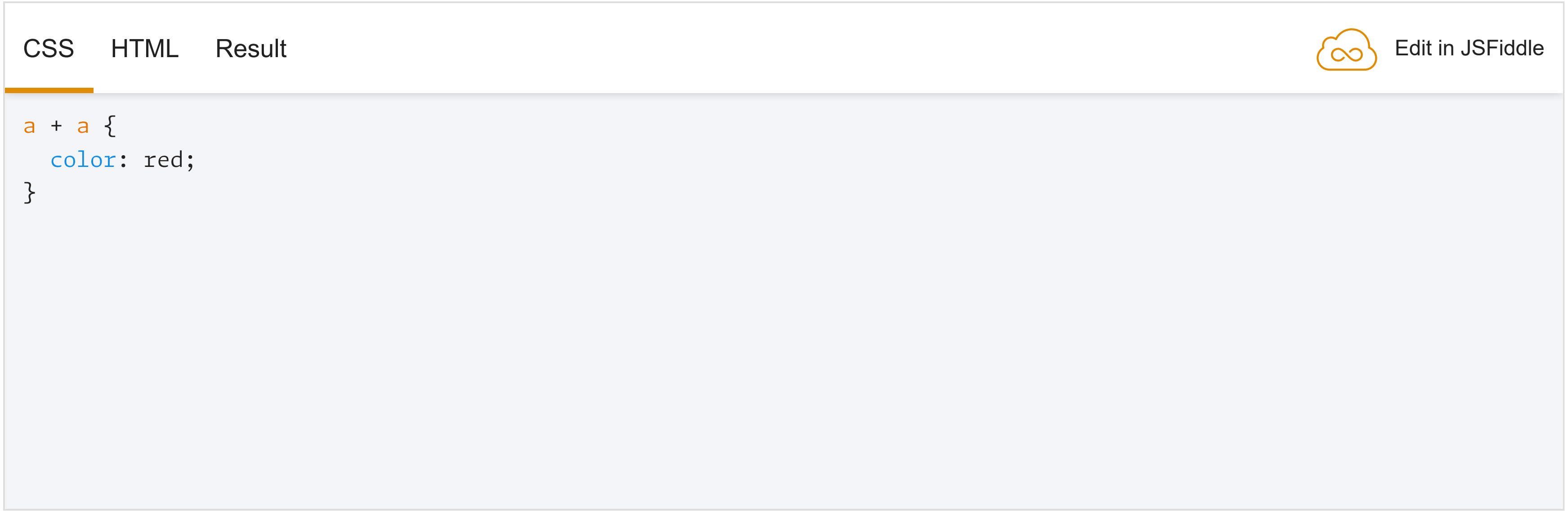


```css
a~a {
  color: red;
}
```

6.4 Adjacent sibling combinator

Syntax: `element1 + element2`

Matches the immediate sibiling of the first element that is immediately following the first element.



```css
a + a {
  color: red;
}
```

7. CSS Pseudo-classes (I)

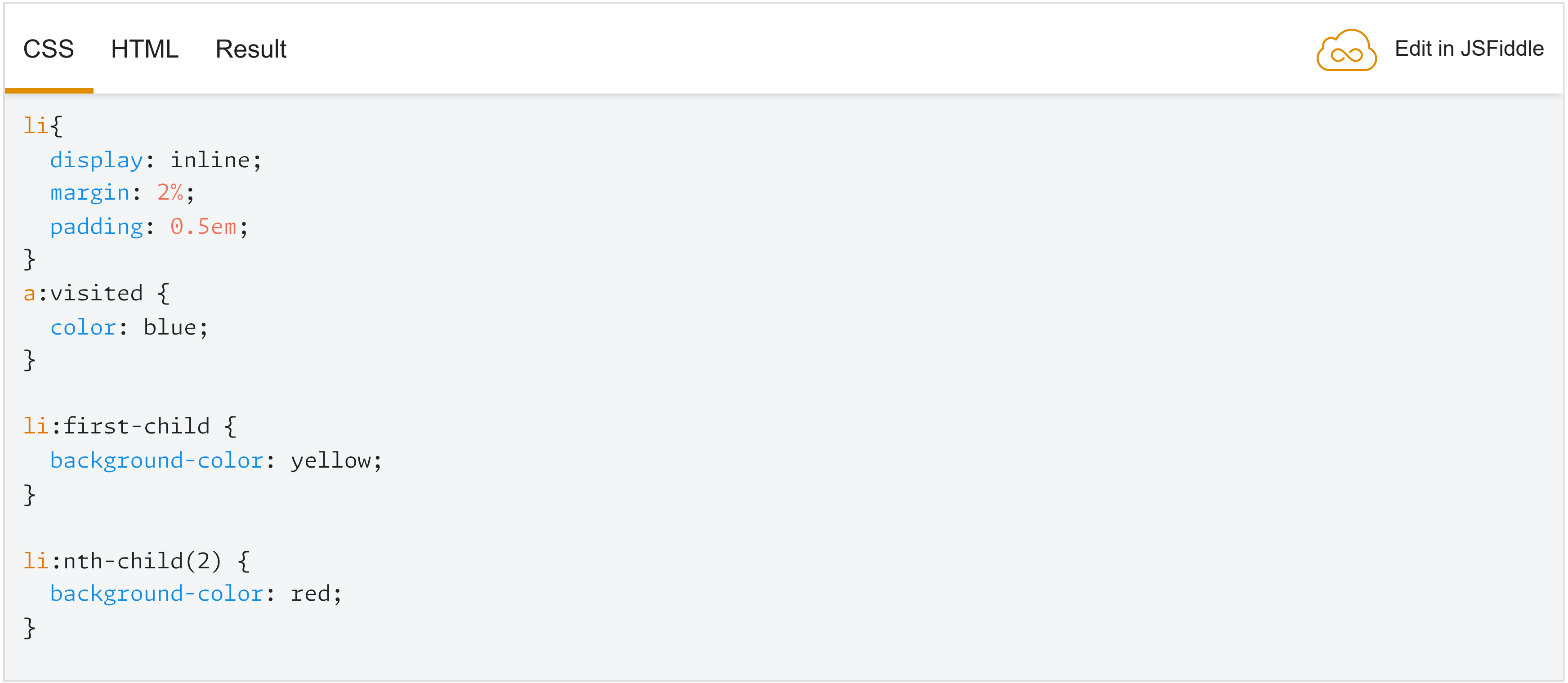
Syntax: `selector:pseudo-class { style properties }`

- In CSS, a pseudo-class is a keyword added to a selector to indicate a special state of the element (e.g. when the mouse cursor is over the element or the element was clicked on).
- There are several CSS pseudo-classes ↗.
- Next are some of the most commonly used pseudo-classes

7. CSS Pseudo-classes (II)

Pseudo-class	Description
`:visited`	Matches links that have been visited.
`:hover`	Matches when the mouse pointer is over an element.
`:focus`	Matches when an element such as an input form has received focus (e.g., clicked on or selected with the keyboard's **Tab** key)
`:required`	Matches when a form element is required.
`:optional`	Matches when a form element is optional.
`:first-child`	Matches an element that is the *first* of its siblings.
`:last-child`	Matches an element that is the *last* of its siblings.

7. CSS Pseudo-classes (III): Example 1



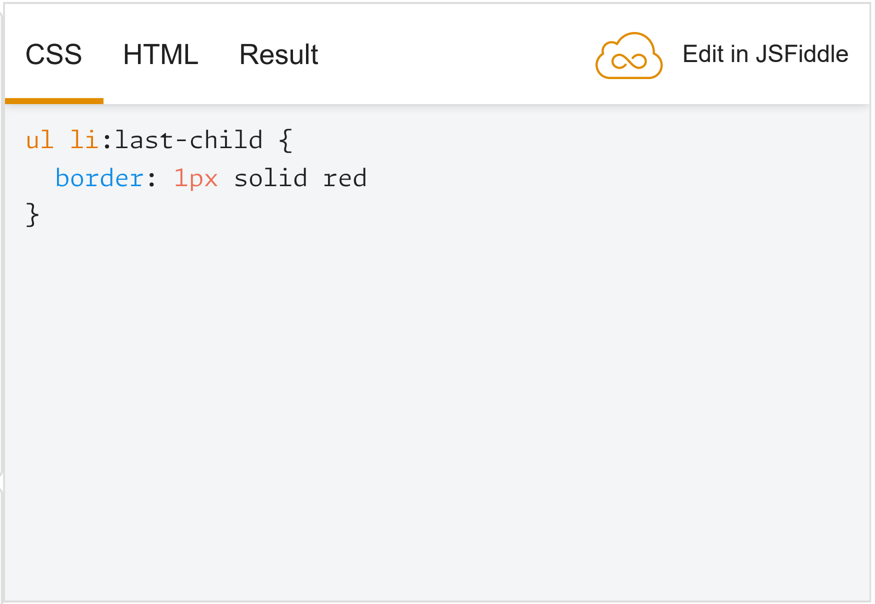
```css
li{
  display: inline;
  margin: 2%;
  padding: 0.5em;
}
a:visited {
  color: blue;
}

li:first-child {
  background-color: yellow;
}

li:nth-child(2) {
  background-color: red;
}
```

7. CSS Pseudo-classes (IV): Example 2

- Consider the following HTML and CSS code. Where would the border appear?

```html
<ul>
    <li>Item 1</li>
    <li>Item 2</li>
    <li>
        Item 3
        <ul>
            <li>Item 3.1</li>
            <li>Item 3.2</li>
            <li>Item 3.3</li>
        </ul>
    </li>
</ul>
```

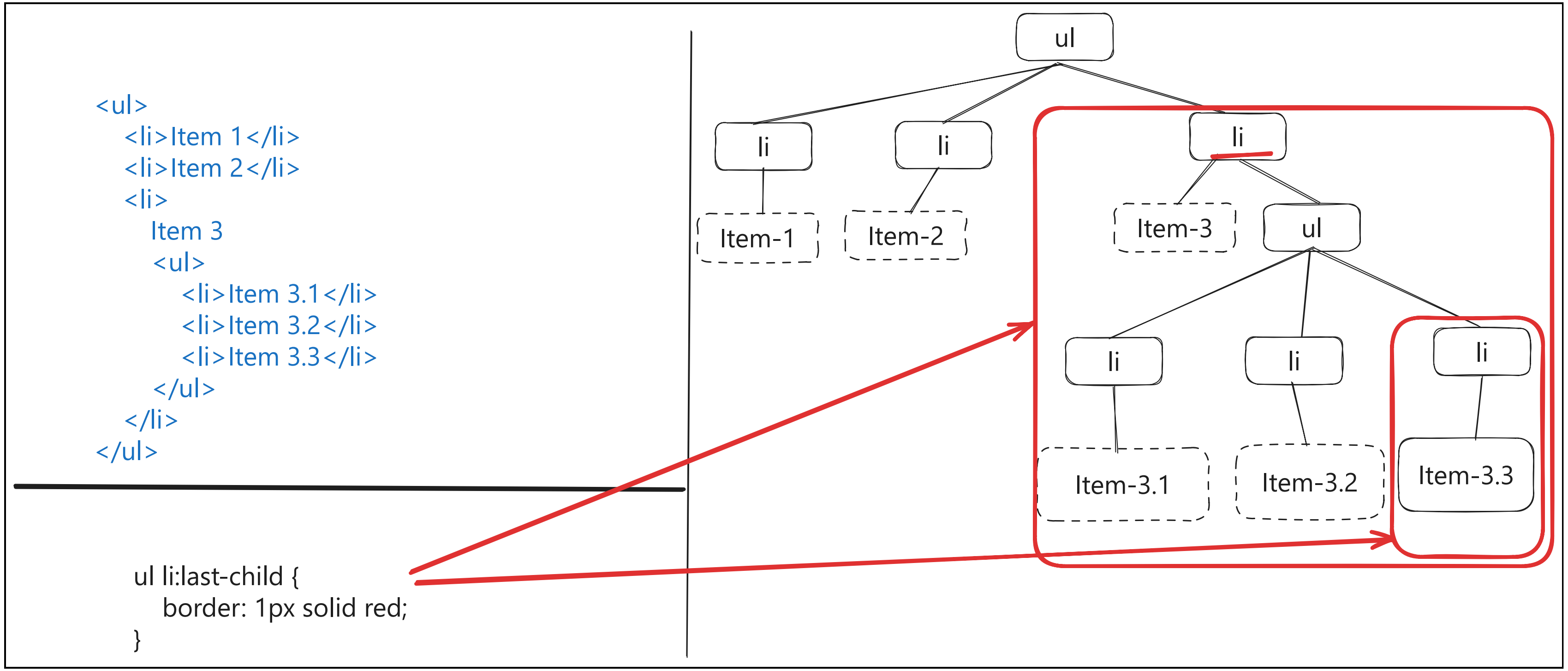
```css
ul li:last-child {
  border: 1px solid red
}
```



```css
ul li:last-child {
  border: 1px solid red
}
```

7. CSS Pseudo-classes (VI): Example 2 (Cont.)

- Here's the DOM tree illustrating how the CSS rule was applied.



```
<ul>
  <li>Item 1</li>
  <li>Item 2</li>
  <li>
    Item 3
    <ul>
      <li>Item 3.1</li>
      <li>Item 3.2</li>
      <li>Item 3.3</li>
    </ul>
  </li>
</ul>
```

```
ul li:last-child {
    border: 1px solid red;
}
```

7. CSS Pseudo-classes (VII): `first-child` Example (Question)

- Consider the following HTML code

```
<html><head></head>
<body>
  <a href="#">First Link</a>
  <a href="#">Second Link</a>
  <div>
    <p>
      <a href="#">Third Link</a>
    </p>
  </div>
  <a href="#">Fourth Link</a>
</body>
</html>
```

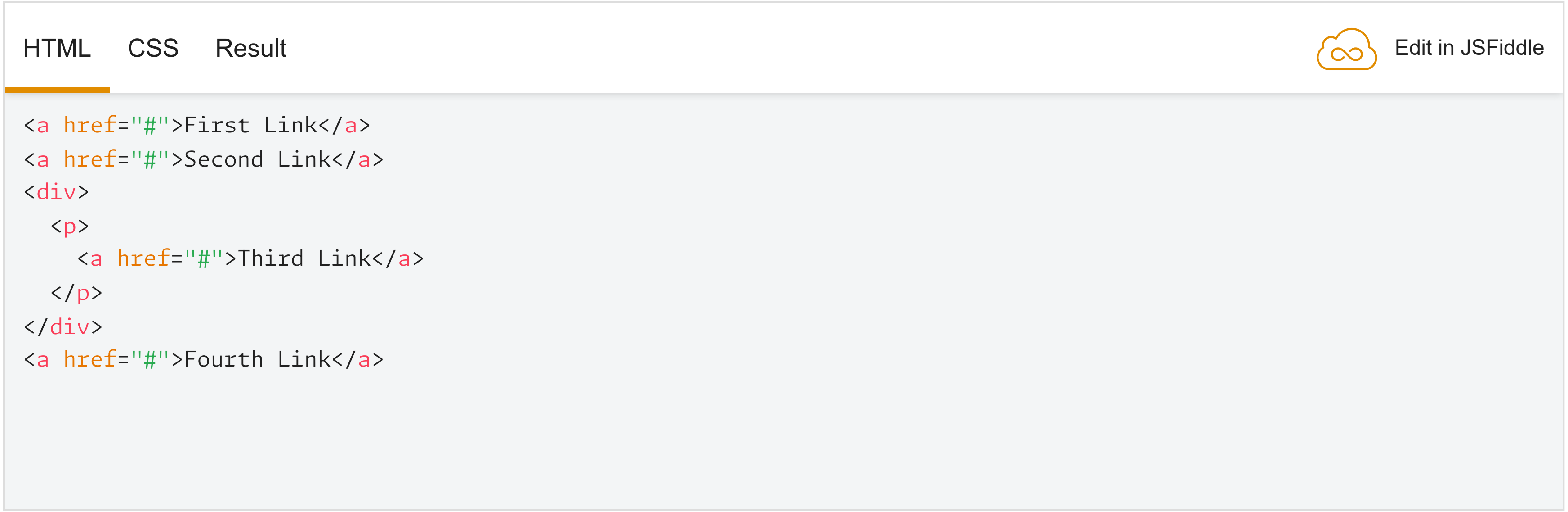
- and CSS code:

```
a:first-child {
  color: red;
}
```

Which link will appear red and why?

7. CSS Pseudo-classes (VII): `first-child` Example (Answer)

- Both the "First Link" and "Third Link" `<a>` tags are the ones that will appear red.



```html
<a href="#">First Link</a>
<a href="#">Second Link</a>
<div>
  <p>
    <a href="#">Third Link</a>
  </p>
</div>
<a href="#">Fourth Link</a>
```

7. CSS Pseudo-classes (VIII): `last-child` Example (Question)

- Consider the following HTML code

```html
<html><head></head>
<body>
  <a href="#">First Link</a>
  <a href="#">Second Link</a>
  <div>
    <p>
      <a href="#">Third Link</a>
    </p>
  </div>
  <a href="#">Fourth Link</a>
</body>
</html>
```

- and CSS code:

```css
a:last-child {
  color: red;
}
```

Which link will appear red and why?

7. CSS Pseudo-classes (VIII): `last-child` Example (Answer)

- The "Third Link" `<a>` tag is the only one that will appear red
- The `:last-child` pseudo-class represents an element that is last among its inclusive siblings.
  - An inclusive sibling is an object or one of its siblings (share the same non-null parent.). Source
  - There is a difference between the last element in the source code and the last child of a specific parent element in the DOM tree. `div` is considered the last child of the parent `body` tag.

HTML CSS Result Edit in JSFiddle

```
<a href="#">First Link</a>
<a href="#">Second Link</a>
<div>
  <p>
    <a href="#">Third Link</a>
  </p>
</div>
<a href="#">Fourth Link</a>
```

8. CSS Pseudo-elements (I)

- In CSS, pseudo-elements are keywords added to element(s) to style a specific part of the element(s). Below are examples of pseudo-elements:
- `::first-line`
  - This can be used to apply styles to the first line of a block-level element. For example, change the font of the first line of a paragraph.
  - The length of the first line is determined by many factors, including the width of the element, the width of the document or screen and the font size of the text".
- `::after`
  - This can be used to add a psudeo-element that is the last child of the selected element. for example, new content to the end of an element.
- `::before`
  - This can be used to add a psudeo-element that is the first child of the selected element. For example, new content to the beginning of an element.

8. CSS Pseudo-elements (II)

- `::selection`
  - This can be used to style an element that has been highlighted by the user (e.g., mouse clicking and dragging the mouse across text).
- For the complete list and description of CSS pseudo-elements, see MDN - Pseudo-elements

CSS Pseudo-elements (II)

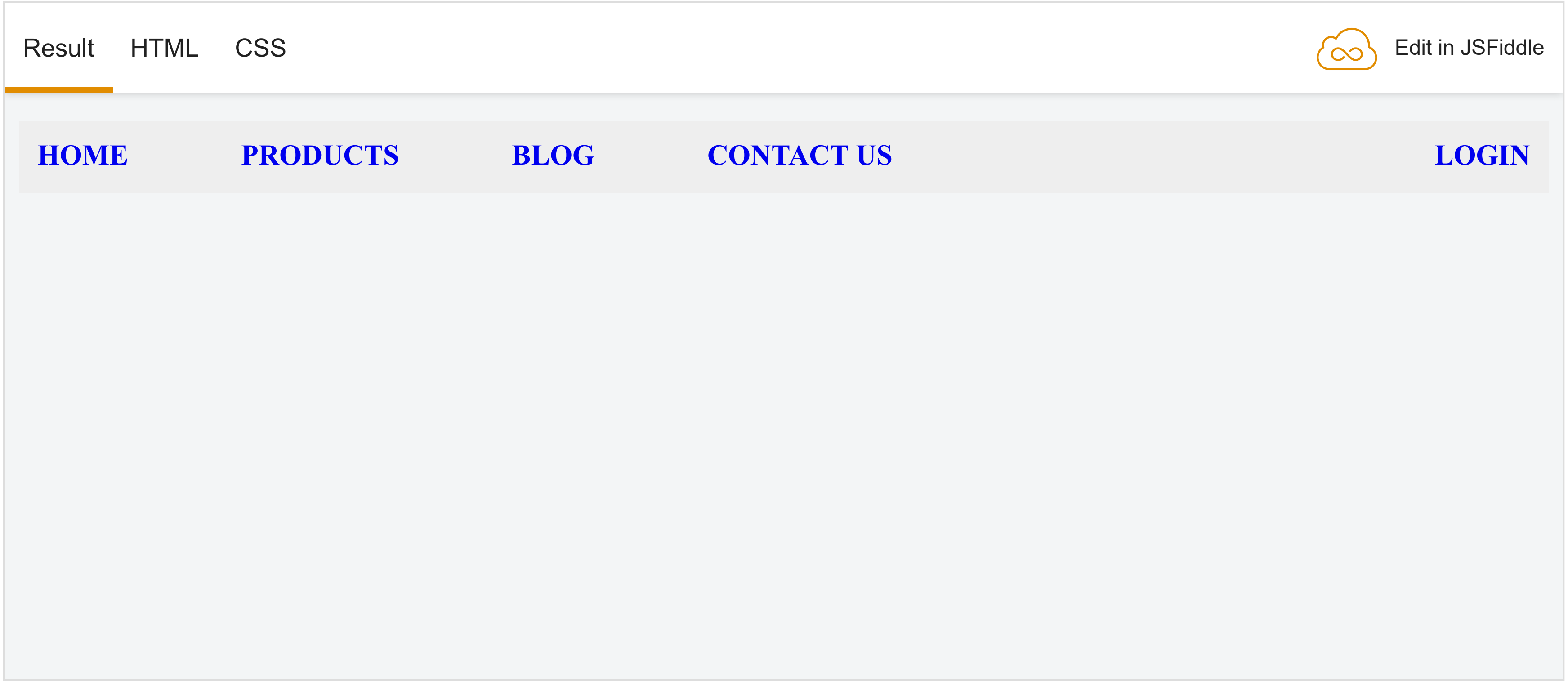
Example: The example below shows how to change the font style of the first line and add a wide-Headed north east arrow ↗ after an external link.

CSS HTML Result Edit in JSFiddle

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

/* Add a North East Arrow for extneral links*/
a.external::after {
  content: " ↗";
}
```

Example: Navigation Bar

Below is the HTML and CSS code for creating a horizontal navigation bar that looks like this:



CSS Properties (I)

CSS Property	Description
`color`	Sets the color of the text.
`background-color`	Sets the background color of an element.
`font-size`	Sets the size of the font.
`font-family`	Specifies the font face to be used.
`text-align`	Aligns the text within an element.
`margin`	Specifies the space around an element, outside of any defined borders.
`padding`	Specifies the space between the content of an element and its border.

CSS Properties (II)

CSS Property	Description
`border`	Sets the border around an element.
`border-radius`	Defines the radius of the element’s corners.
`width`	Sets the width of an element.
`height`	Sets the height of an element.
`display`	Specifies how an element should be displayed (block, inline, flex, grid, etc.).
`position`	Specifies the type of positioning for an element (static, relative, absolute, fixed, sticky).
`opacity`	Sets the opacity level for an element.

CSS Properties (III)

CSS Property	Description
`transition`	Specifies the speed curve of the transition effect.
`transform`	Applies a 2D or 3D transformation to an element.
`overflow`	Specifies what should happen if content overflows an element’s box.
`float`	Specifies how an element should float.
`clear`	Specifies on which sides of an element floating elements are not allowed to float.
`visibility`	Specifies whether or not an element is visible.
`line-height`	Sets the height of a line.

CSS Comments

- Comments in CSS start with `/*` and end with `*/` and can span multiple lines.
- Comments are ignored by the browser, so they don’t affect how the CSS is applied.

```css
h1 {color: red;} /* This is a CSS comment */
```

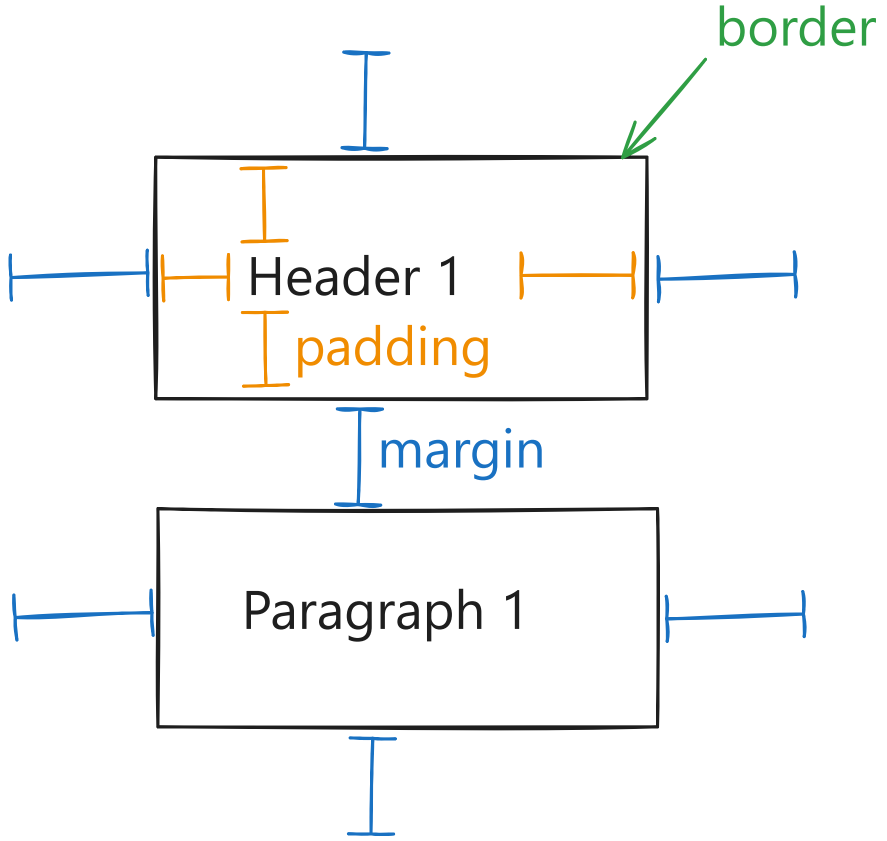
The CSS Box Model

Content, Padding, Border, and Margin

- The CSS Box Model is a fundamental concept in CSS that defines how elements are rendered on the screen.
- Each element is represented as a rectangular box, with the box’s **content**, **padding**, **border**, and **margin**.
- The **content** box contains the actual content of the element, such as text or an image.
- The **padding** box represents the space between the content and the border.
- The **border** box is the line that goes around the element (padding and content).
- The **margin** box is the space around the border.

The CSS Box Model (II)

```
<h1>Header 1</h1>
<p>Paragraph 1</p>


h1, p{
   margin: 25px;
   padding: 10px;
   border: 1px solid black;
}
```



CSS Units

- CSS units are measurements used in CSS to express the size, spacing, or other aspects of an element.
- CSS units can be absolute or relative.
- Absolute length units are not recommended for use on screens due to the wide variation in screen sizes.
- Absolute length units are often used for print fonts and layouts.
- Relative units are more flexible and responsive than absolute units.
- Relative units are the foundation of responsive design that work well across various devices and screen sizes.

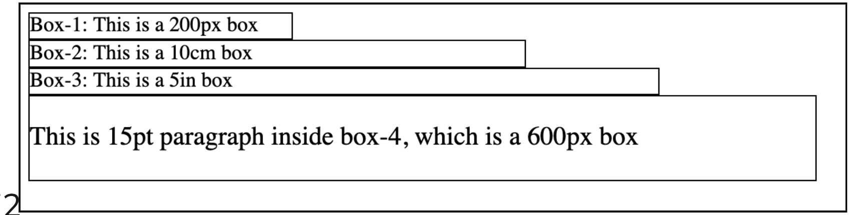
Absolute length units

- Absolute length units are often considered to always be the same size. Some of the most useful CSS absolute units are listed below:
- `px`: Pixels. 1px = 1/96th of 1in.
  - This unit is not an actual physical pixel. It’s amagical unit whose value varies by hardware and resolution. The size of 1px depends on the type of device and meant to be small but visible.
- `cm`: Centimeters. 1cm = 37.8px = 25.2/64in
- `in`: Inches. 1in = 2.54cm = 96p
- `pt`: Points. 1pt = 1/72th of 1 in.
  - This unit is often used for printed documents or ink-on-paper typography. Thus, it’s often used with the `font-size` property when creating a custom style for printers. For example, one may create a special CSS for printing only as `<link rel="stylesheet" media="print" href="print.css" />`

Absolute length units Example

```html
<div class="box-1">
  Box-1: This is a 200px box
</div>
<div class="box-2">
  Box-2: This is a 10cm box
</div>
<div class="box-3">
  Box-3: This is a 5in box
</div>
<div class="box-4">
  <p>This is 15pt paragraph inside box-4, which is a 600px box</p>
</div>
```

```css
div{
  border: 1px solid black;
}
.box-1{
  width: 200px;
}
.box-2{
  width: 10cm;
}
.box-3{
  width: 5in;
}
.box-4{
   width: 600px;
}
p{
  font-size: 15pt;
}
```


Box-1: This is a 200px box
Box-2: This is a 10cm box
Box-3: This is a 5in box
This is 15pt paragraph inside box-4, which is a 600px box

Relative length units

- Relative length units are relative to the size of the parent element, the parent element’s font, or the size of the viewport.
- These units are very helpful when designing web pages that will be rendered on devices of various screens as they scale relative to the size of the viewport.
- Some of the most useful CSS absolute units are listed below:
  - `%` Percentages
  - `vw` viewport width
    - 1% of the viewport’s width.
  - `vh` viewport height
    - 1% of the viewport’s height.
  - `em` element-relative
    - Relative to font size of the current element or its parent.
  - `rem` root-relative
    - Relative to font size of the root element.

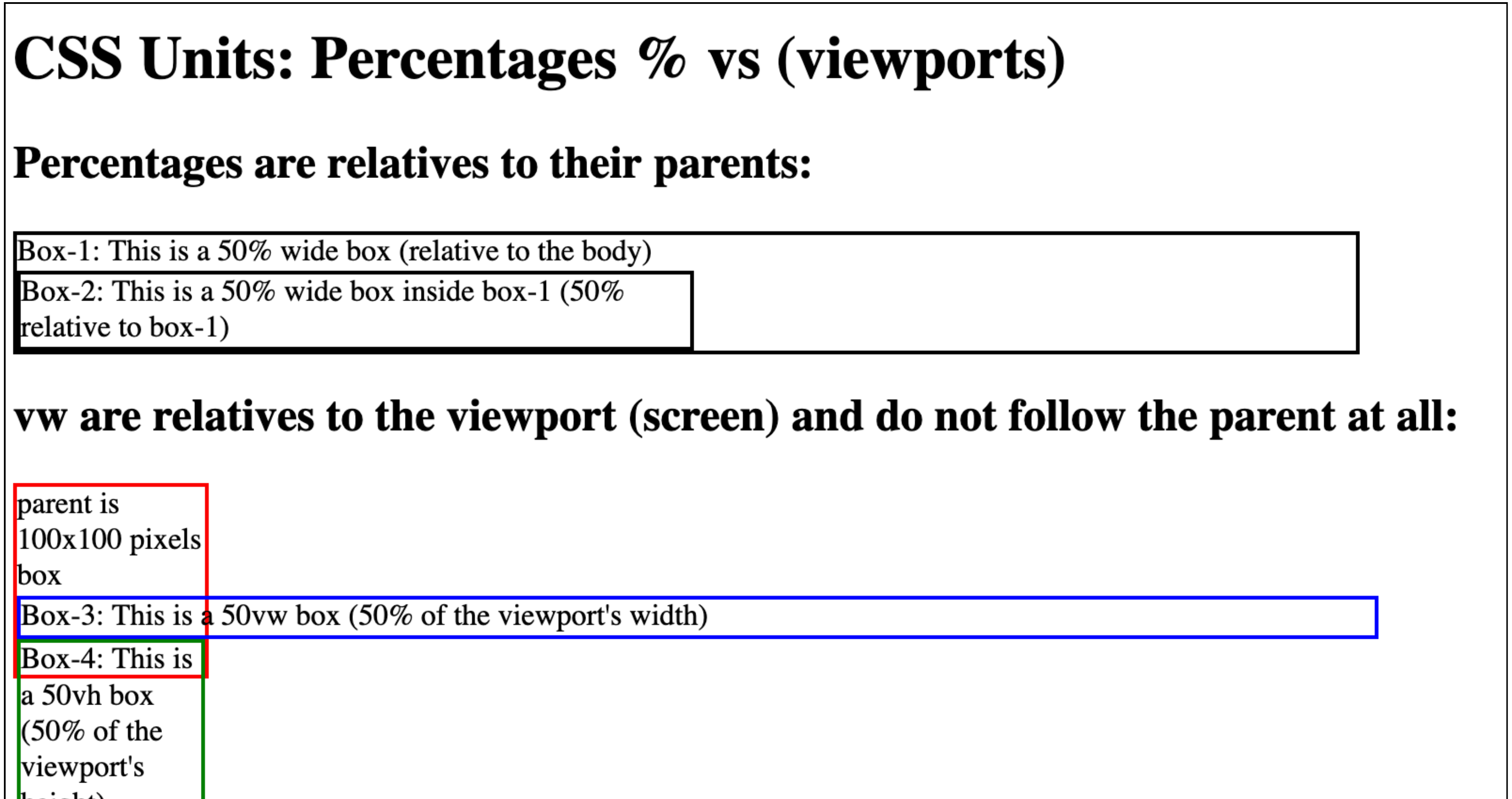
Example: percentages (`%`) vs viewports (`vw` `vh`)

Percentages are relative to the size of the parent element and viewports are relative to the size of the viewport.

```html
<h1>CSS Units: Percentages % vs (viewports)</h1>
<h2>Percentages are relatives to their parents:</h2>
<div class="box-1">
  Box-1: This is a 50% wide box (relative to the body)
  <div class="box-2">
    Box-2: This is a 50% wide box inside box-1 (50% relative to box-1)
  </div>
</div>
<h2>vw are relatives to the viewport (screen) and do not follow the parent at all:</h2>
<div class="parent">
  parent is 100x100 pixels box
  <div class="box-3">
    Box-3: This is a 50vw box (50% of the viewport's width)
  </div>
  <div class="box-4">
    Box-4: This is a 50vh box (50% of the viewport's height)
  </div>
</div>
```

```css
div{
  border: 2px solid black;
}
.box-1{
  width: 50%;
}
.box-2{
  width: 50%;
}
.parent{
  width:100px;
  height: 100px;
  border-color: red;
}
.box-3{
  width: 50vw;
  border-color: blue;
}
.box-4{
  height: 50vh;
  border-color: green;
```

Example: percentages (`%`) vs viewports (`vw` `vh`) (Cont.)



CSS Units: Percentages % vs (viewports)

Percentages are relatives to their parents:

Box-1: This is a 50% wide box (relative to the body)

Box-2: This is a 50% wide box inside box-1 (50% relative to box-1)

vw are relatives to the viewport (screen) and do not follow the parent at all:

parent is 100x100 pixels box

Box-3: This is a 50vw box (50% of the viewport's width)

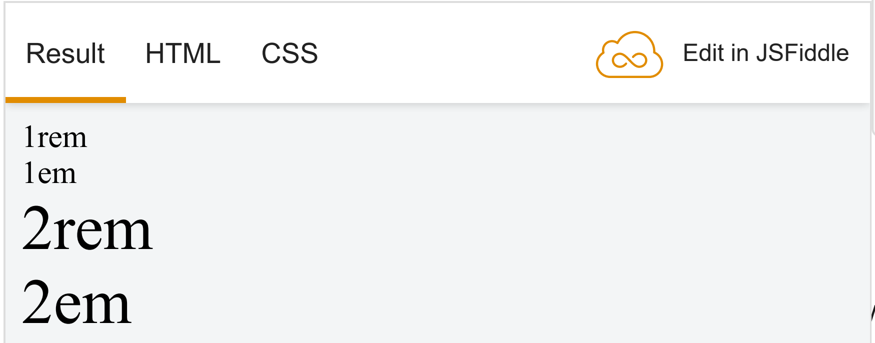
Box-4: This is a 50vh box (50% of the viewport's

Example: `rem` vs `em`

Both `rem` and `em` are relative to the defined font size. `rem` is relative the root's font size while `rm` is relative to the parent's font size.

```html
<div class="one-rem">1rem</div>
<div class="one-em">1em</div>
<div class="two-rem">2rem</div>
<div class="two-em">2em</div>
<hr>
<div class="parent">
    <div class="one-rem">1rem</div>
    <div class="one-em">1em</div>
    <div class="two-rem">2rem</div>
    <div class="two-em">2em</div>
</div>
```

```css
.parent{
    font-size: 40px;
}
.one-em{
    font-size: 1em;
}
.one-rem{
    font-size: 1rem;
}
.two-em{
    font-size: 2em;
}
.two-rem{
    font-size: 2rem;
}
```

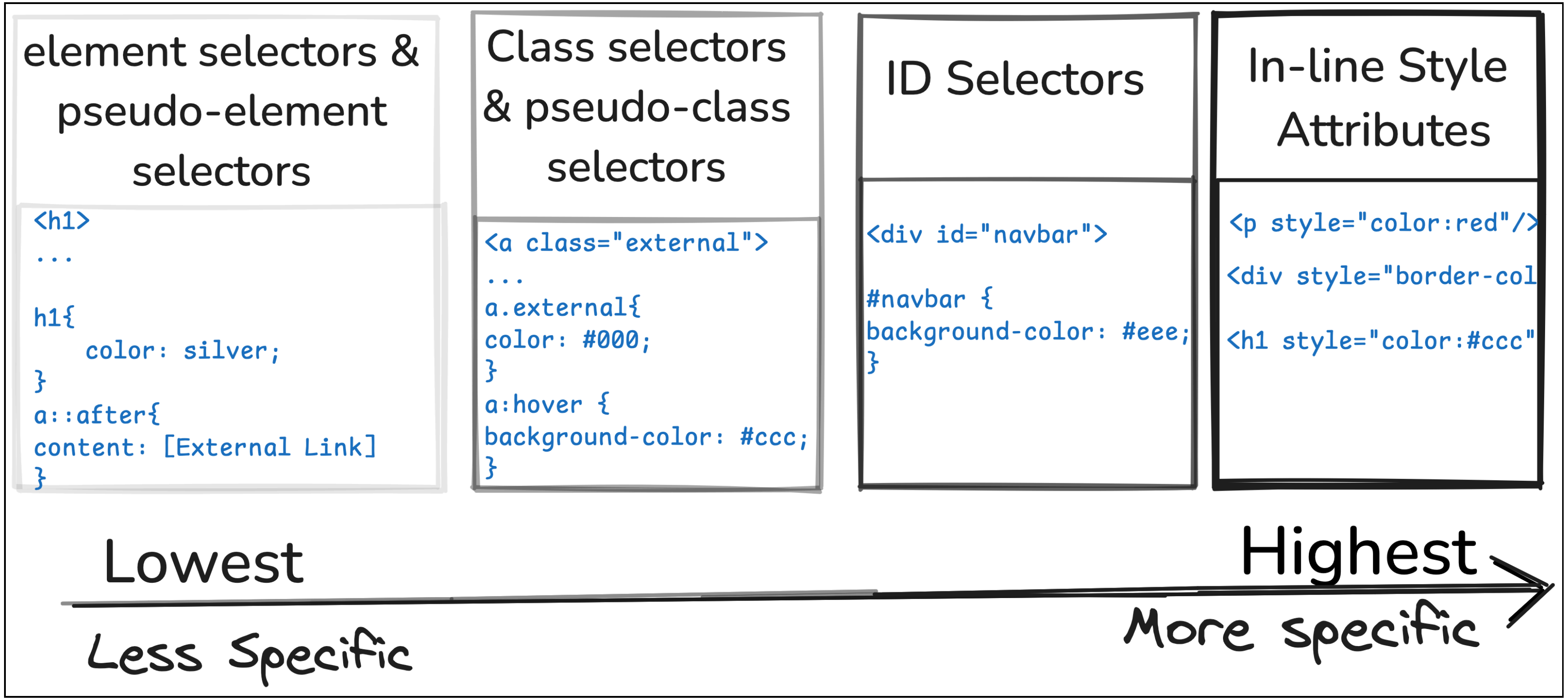


1rem
1em
2rem
2em

CSS Specificity

- Specificity is like a ranking system where elements with the highest specificity takes precedence.
- The selector with the highest specificity will win and be applied.
  - Inline styles have the **highest specificity**.
  - ID selectors (e.g., `#title`) are **more specific** than class selectors (e.g., `.title`).
  - **Less specific** selectors are class selectors (e.g., `.article`), pseudo-class selectors (e.g., `:hover`), and attribute selectors (e.g., [type="text"]).
  - Finally, the **least specific** selectors are element selectors (e.g., `div`, `h1`, `p`) and pseudo-element selectors (e.g., `::before`, `::after`).

CSS Specificity (II)



element selectors & pseudo-element selectors

```
<h1>
...

h1{
    color: silver;
}
a::after{
content: [External Link]
}
```

Class selectors & pseudo-class selectors

```
<a class="external">
...
a.external{
color: #000;
}
a:hover {
background-color: #ccc;
}
```

ID Selectors

```
<div id="navbar">

#navbar {
background-color: #eee;
}
```

In-line Style Attributes

```
<p style="color:red"/>

<div style="border-col

<h1 style="color:#ccc"
```

CSS Specificity (III)

- Consider the following HTML and CSS code, what would the color of the `<p>` element be?

```html
<p id="content" class="article">What is my font color?</p>
```

```css
#content{
  color: blue;
}
.article{
  color: green;
}
p {
  color: red;
}
```

Result

Edit in JSFiddle

What is my font color?

The `!important` Declaration

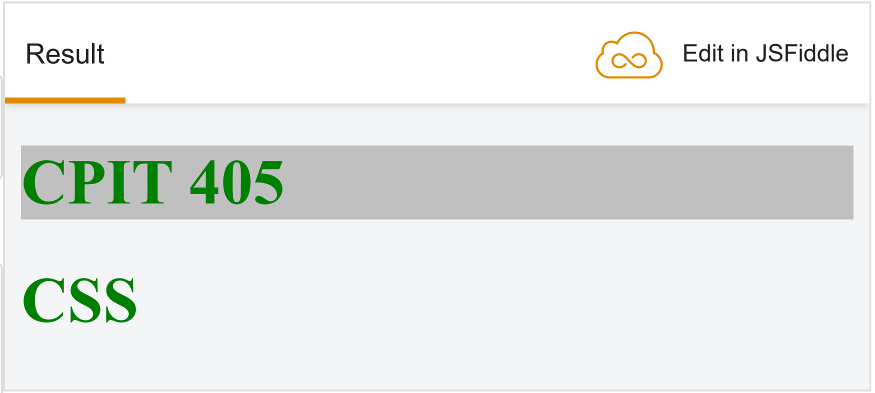
- `!important` is a CSS declaration that overrides other declarations regardless of specificity.
- `!important` takes precedence over all other declarations and specificity rules.
- Consider the following HTML and CSS code, what would the color of each `<h1>` element be?

HTML

```html
<h1 class="title">CPIT 405</h1>
<h1>CSS</h1>
```

CSS

```css
h1 {
  color: green !important;
}
h1.title {
  color: red;
  background: silver;
}
```



CPIT 405

CSS

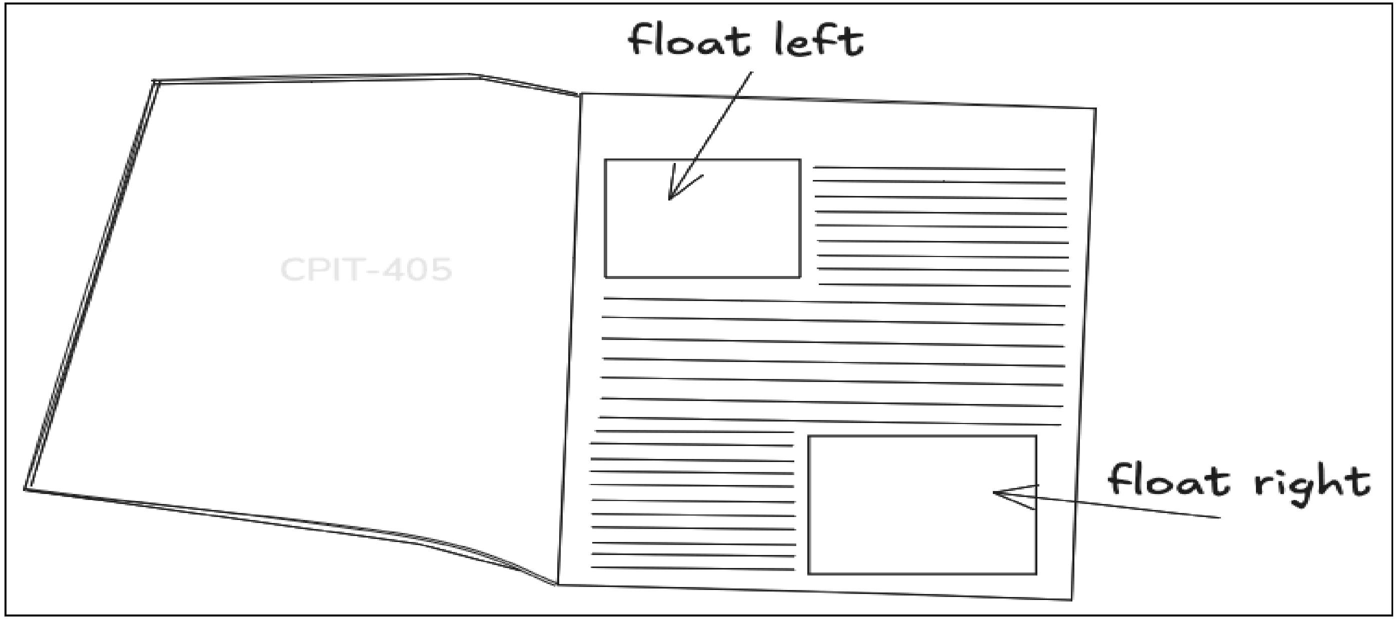
CSS Layouts

> The following topics are **optional, not in the exam,** and serve as supplemental materials to the CSS content.

- Floating and Positioning
- Flexbox
- Gridlayout

Floating

- The `float` property places an element on the left or right side of its container, allowing text and inline elements to wrap around it.
- The `float` property can be set to `left`, `right`, `inline-start`, `inline-end`, or `none`.



Floating Example

- The `float` CSS property:

```
img.left {float: left;}
```

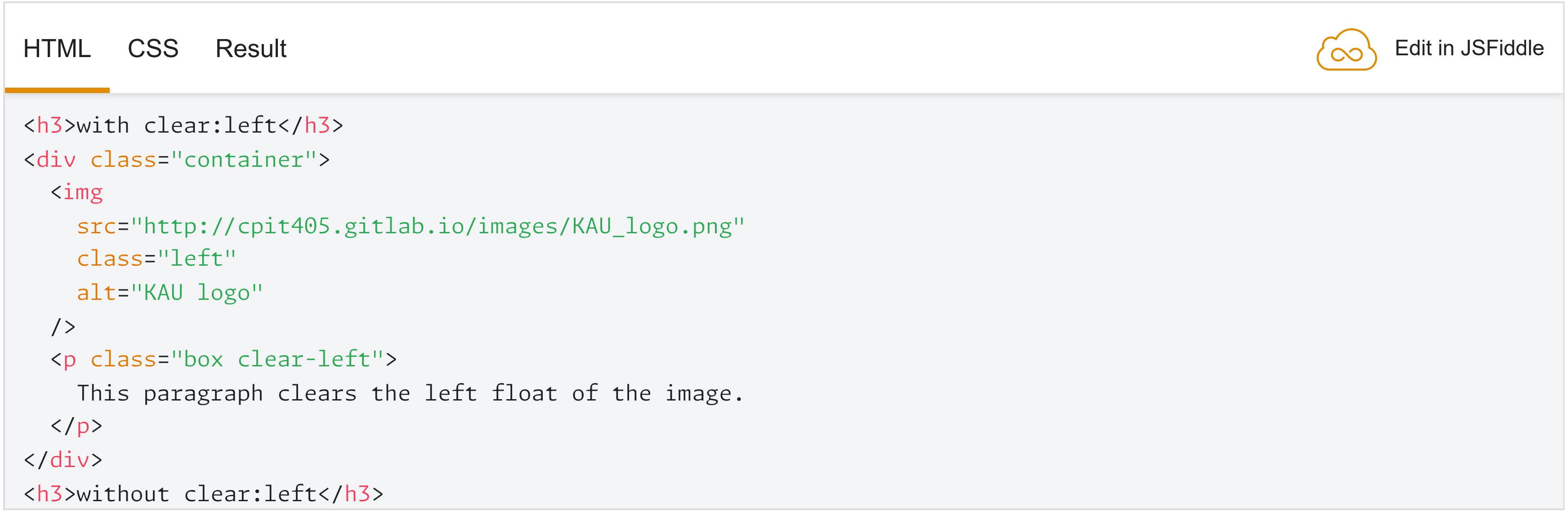
- Example:

HTML CSS Result Edit in JSFiddle

```
<div class="container">
  <img
    src="http://cpit405.gitlab.io/images/KAU_logo.png"
    class="left"
    alt="KAU logo"
  />
  <p>
    This paragraph will wrap around the floated elements. The float property
    places an element on the left or right side of its container, allowing text
    and inline elements to wrap around it.
  </p>
</div>
```

Clearing

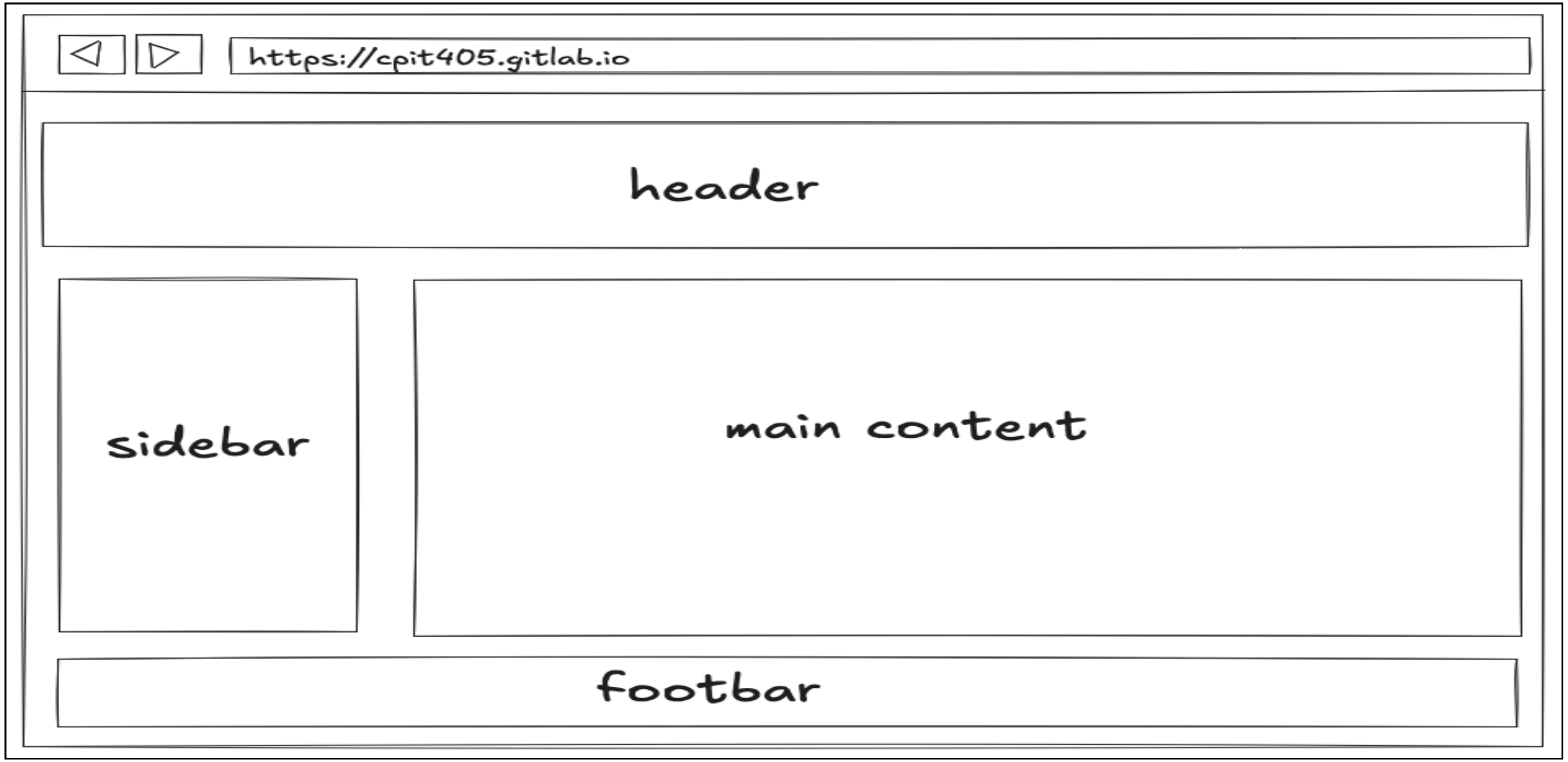
- The `clear` property places an element below any preceding floating elements.
  - It controls the behavior of elements next to floated elements.
  - It prevents elements from wrapping around floated elements
- The `clear` property can be set to `left`, `right`, `both`, or `none`.



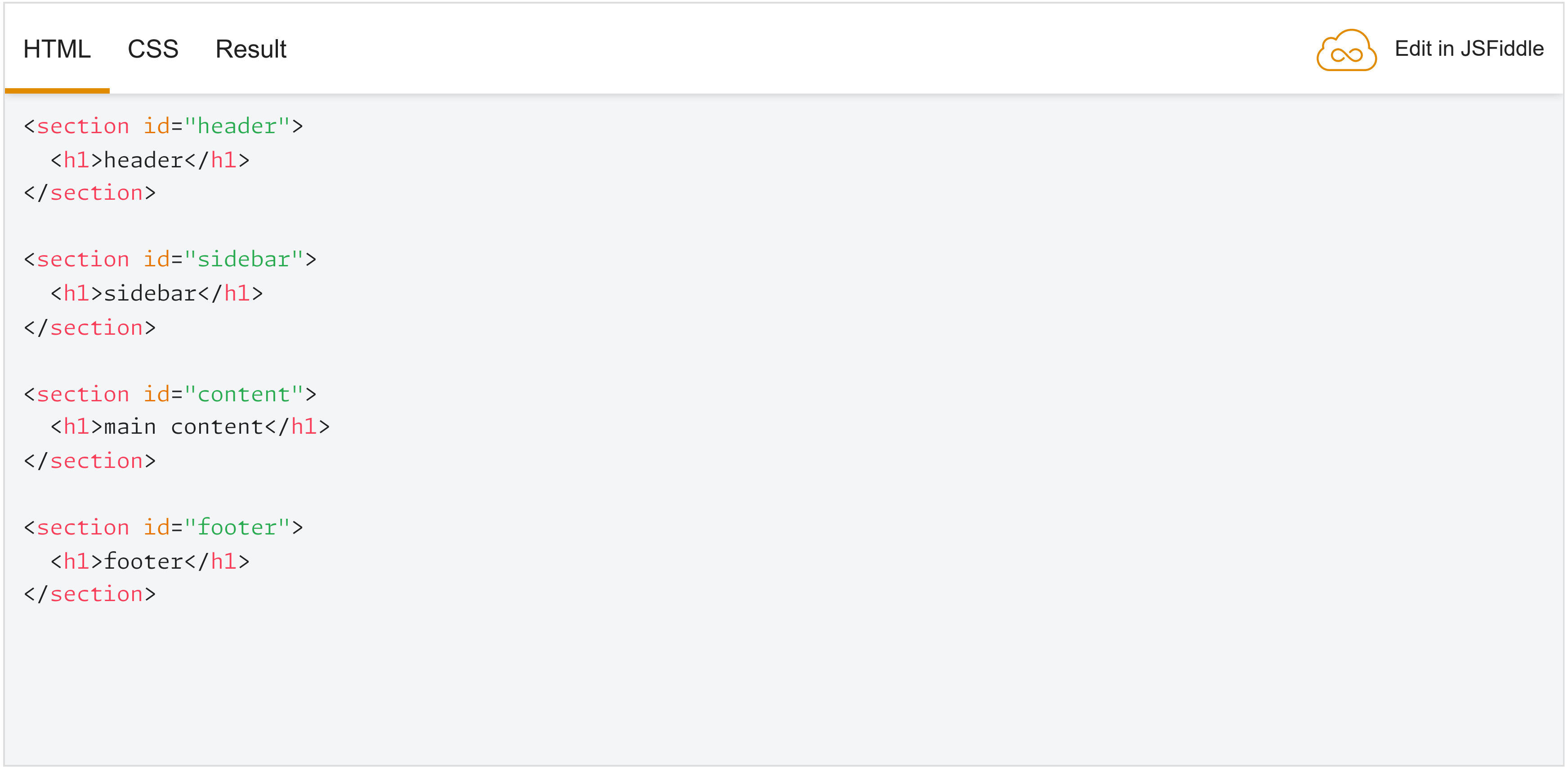
```html
<h3>with clear:left</h3>
<div class="container">
  <img
    src="http://cpit405.gitlab.io/images/KAU_logo.png"
    class="left"
    alt="KAU logo"
  />
  <p class="box clear-left">
    This paragraph clears the left float of the image.
  </p>
</div>
<h3>without clear:left</h3>
```

Float and Clear for layouts (I)

- Aside from being used for wrapping text around elements such as images, `float` and `clear` can be used to create web layouts.



Float and Clear for layouts (II)



HTML CSS Result Edit in JSFiddle

```
<section id="header">
  <h1>header</h1>
</section>

<section id="sidebar">
  <h1>sidebar</h1>
</section>

<section id="content">
  <h1>main content</h1>
</section>

<section id="footer">
  <h1>footer</h1>
</section>
```

Flexbox

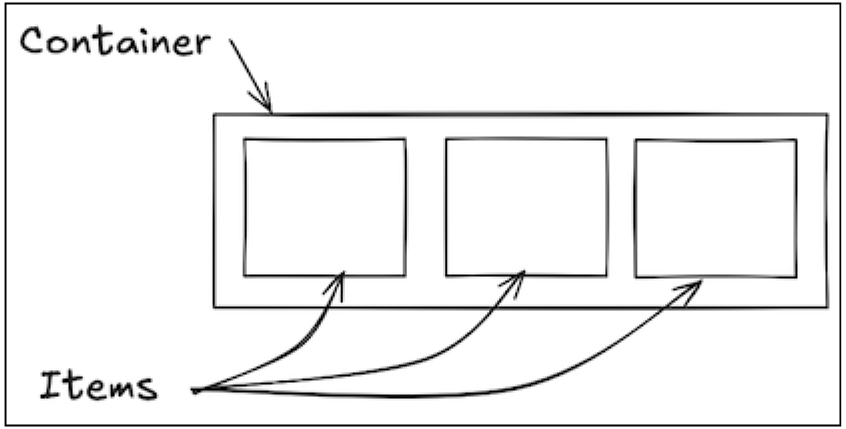
- Flexbox ↗, the flexible box layout, is an essential layout model for creating responsive layouts that adapt to different screen sizes and devices.
- It is a one-dimensional layout model for distributing space between items.
- It provides a flexible way to align and arrange items within a container, even when their sizes are unknown or dynamic.
- It is a **a single-direction layout model**, where items are layed out either in horizontal rows or vertical columns.

Flexbox (II)

- A Flexbox container is an element that has its display property set to `flex` or `inline-flex`.
- The `justify-content` property is used to distribute space around the flex items.
- The `align-items` property is used to align the flex items along the cross axis (vertically in this case).

```html
<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>
```

```css
  .flex-container {
    display: flex;
    justify-content: space-around;
    align-items: center;
  }
```



Flexbox (III)

- The `flex-direction` property in Flexbox defines the direction in which the flex items are placed in the flex container:
  - `row`: horizontal
  - `row-reverse` horizontal with items placed in a row from right to left.
  - `column`: vertical
  - `column-reverse`: vertical with items placed in a row from bottom to top.

Flexbox (IV)

HTML CSS Result

Edit in JSFiddle

```html
<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>
```

GridLayout

Wrapping up

- We have seen how CSS is used to define the style and layout of a web page.
- We have learned about the following CSS topics:
  - CSS Properties and Selectors
  - The CSS Box Model
  - CSS Units
  - Cascading Order in CSS
  - CSS Specificity
  - Floating and Positioning
  - Flexible Box Layout
  - Grid Layout
  - Transitions and Animations
- Coming up next: Learn JavaScript to add interactivity and dynamic functionality to webpages 👨‍💻 JS 👩‍💻

References

- Mozilla Developer Network (MDN) CSS. Retrieved from https://developer.mozilla.org/en-US/docs/Web/CSS.
- Meyer, E. and Weyl, E. 2023. CSS: The Definitive Guide. 5th Ed. O’Reilly Media.