Universal selector (*), attribute selectors, pseudo-classes (:hover), and pseudo-elements (::before, ::after).

1. Universal Selector (*)

The universal selector (*) in CSS selects all elements in a document, regardless of their type. It's useful when you want to apply a global style to every element, though it's typically used sparingly to avoid performance issues.

Example:

css

Copy code

```
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}
```

In this example, the universal selector resets the margin and padding for all elements to ensure consistent styling across browsers.

2. Attribute Selectors

CSS attribute selectors target elements based on the presence or value of an attribute. They are powerful when styling elements with specific attribute values, such as form inputs, links, or custom HTML data attributes.

Basic syntax:

css

Copy code

```
/* Select elements that have the 'href' attribute */a[href] {
  color: blue;
}
/* Select elements where the href starts with 'https' */a[href^="https"] {
  color: green;
}
/* Select elements where the href contains 'example' */a[href*="example"] {
  color: red;
}
```

In these examples:

- The first rule selects all <a> elements with an href attribute.
- The second rule targets links that start with "https".
- The third rule selects links that contain "example" in their href attribute.

3. Pseudo-Classes (:hover, :nth-child(), etc.)

Pseudo-classes define a special state of an element. They don't select elements directly but rather style them when they are in a certain condition or state.

Popular pseudo-classes:

- :hover – Applies styles when the user hovers over an element.

- :nth-child() – Selects elements based on their position within a parent.

Examples:

css

Copy code

```
/* Change link color on hover */a:hover {
  color: red;
}
/* Style every second list item */li:nth-child(2n) {
  background-color: #f0f0f0;
}
```

In the first example, the link changes color when hovered. In the second, every second <li> element is styled with a background color.

4. Pseudo-Elements (::before, ::after)

Pseudo-elements allow you to style specific parts of an element. They are often used to insert content before or after the content of an element.

Common pseudo-elements:

- ::before – Inserts content before an element's actual content.
- ::after – Inserts content after an element's actual content.

Examples:

css

Copy code

```
/* Add a decorative element before a heading */h1::before {
  content: "✨ ";
  font-size: 24px;
}
/* Add a bullet after list items */li::after {
  content: " •";
  color: gray;
}
```

In the first example, a sparkly emoji is added before every <h1>. In the second, a gray bullet is added after each list item.

Summary

- **Universal selector (*)** applies to all elements.
- **Attribute selectors** allow styling based on attribute presence or value.
- **Pseudo-classes** (:hover, :nth-child()) style elements in special states or conditions.
- **Pseudo-elements** (::before, ::after) allow insertion of content and styling of specific parts of an element.

Each of these selectors adds flexibility and precision to CSS, allowing you to style elements based on more complex conditions than simple element, class, or ID selectors.

specificity, inheritance, and cascading in CSS

1. Specificity

Specificity determines which CSS rule is applied when multiple rules target the same element. CSS assigns weights to different selectors, and the rule with the highest weight (specificity) is applied.

Example:

html

Copy code

```
<!DOCTYPE html><html><head>
 <style>
  /* Universal selector - lowest specificity */
  * {
   color: blue;
  }

  /* Element selector - higher specificity than universal selector */
  p {
   color: green;
  }

  /* Class selector - higher specificity than element selector */
  .important-text {
   color: red;
  }

  /* ID selector - highest specificity */
  #special-paragraph {
   color: purple;
  }
 </style></head><body>

 <p>This is a normal paragraph.</p>
 <p class="important-text">This is an important paragraph.</p>
 <p id="special-paragraph">This is a special paragraph.</p>
</body></html>
```

The first paragraph will be green because the p element selector overrides the universal selector *.

The second paragraph will be red because the .important-text class selector has higher specificity than the p element selector.

The third paragraph will be purple because the #special-paragraph ID selector has the highest specificity.

2. Inheritance

Inheritance is the process where some CSS properties, like color and font-family, are inherited from parent elements by their child elements. However, not all properties (e.g., margin, padding) are inherited.

Example:

html

Copy code

```
<!DOCTYPE html><html><head>
  <style>
    body {
      color: navy; /* Inherited */
    }
    p {
      font-size: 16px;
    }
    .non-inherited {
      background-color: yellow; /* Not inherited */
    }
  </style></head><body>

  <div class="non-inherited">
    <p>This paragraph inherits the color navy from the body element.</p>
    <p>However, the background color is only applied to the div, not the paragraphs inside.</p>
  </div>
</body></html>
```

In this example:

The paragraphs inside the div inherit the color: navy from the body element.

However, the background-color: yellow is applied only to the div, not to the p elements inside, because background properties are not inherited.

3. Cascading

Cascading refers to how the browser determines the final style for an element when multiple styles are applied. The "cascade" is a set of rules that dictate which styles take precedence. If two or more rules have the same specificity, the order in which they are defined matters.

Example:

html

Copy code

```
<!DOCTYPE html><html><head>
  <style>
    p {
      color: black; /* Rule 1 */
    }
    p {
      color: brown; /* Rule 2 - will override Rule 1 */
    }
  </style></head><body>

  <p>This paragraph will be brown due to the cascading effect.</p>
</body></html>
```

In this example:

Both CSS rules target the same element, but because Rule 2 comes later, it overrides Rule 1, and the paragraph text will be brown.

Combining All Three: Specificity, Inheritance, and Cascading

html

Copy code

```
<!DOCTYPE html><html><head>
  <style>
    /* 1. Universal selector */
    * {
      color: gray;
    }

    /* 2. Inherited from parent */
    body {
      color: navy;
    }

    /* 3. Element selector */
    p {
      color: green;
    }

    /* 4. Class selector */
    .important-text {
      color: red;
    }

    /* 5. ID selector */
    #special-paragraph {
      color: purple;
    }
    /* 6. Inline style (highest priority in the cascade) */
  </style></head><body>

  <p>This is a normal paragraph.</p>
  <p class="important-text">This is an important paragraph.</p>
  <p id="special-paragraph" style="color: orange;">This is a special paragraph with inline styling.</p>
</body></html>
```

What happens here:

The first paragraph will be green due to the element selector.

The second paragraph will be red because the class selector has higher specificity than the element selector.

The third paragraph will be orange because inline styles have the highest specificity in the cascade, even though the ID selector targets the same element with a purple color.

CSS Positioning:

CSS positioning is used to control the layout of elements in a webpage. There are five main positioning properties:

static (default) : Default behavior; elements follow normal document flow.

Relative: Positioned relative to its normal position; space remains in document flow.

Absolute: Positioned relative to the nearest positioned ancestor or browser window if none exist.

Fixed: Stays in one position relative to the browser window, even during scrolling.

Sticky: Behaves like relative until you scroll past a point, then acts like fixed.

Each of these properties controls how an element behaves in relation to the normal document flow or its parent container.

Let’s explore each with an example.

---

1. Static Positioning (Default)

By default, all elements are positioned statically. This means they follow the normal flow of the document, appearing one after the other.

Example:

html

Copy code

```
<!DOCTYPE html><html lang="en"><head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Static Positioning Example</title>
    <style>
        div {
            background-color: lightblue;
            padding: 10px;
            border: 1px solid black;
        }
    </style></head><body>
    <div>Div 1</div>
    <div>Div 2</div>
    <div>Div 3</div></body></html>
```

In this example:

All three div elements will be stacked vertically in the document flow as expected. They follow the natural flow, and their position can’t be changed using top, left, right, or bottom properties.

---

2. Relative Positioning

With relative positioning, the element is positioned relative to its original position in the document flow. If you move an element using top, left, right, or bottom, it will still occupy space in the normal flow of the document.

Example:

html

Copy code

```
<!DOCTYPE html><html lang="en"><head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Relative Positioning Example</title>
    <style>
        .relative-box {
            position: relative;
            top: 20px;
            left: 30px;
            background-color: lightgreen;
            padding: 10px;
            border: 1px solid black;
        }
    </style></head><body>
    <div>Div 1</div>
    <div class="relative-box">Div 2 (Relative)</div>
    <div>Div 3</div></body></html>
```

In this example:

Div 2 is shifted 20px down and 30px right, but it still occupies the same space it would have if it were static. Other elements (like Div 3) won’t be affected by this positioning.

---

3. Absolute Positioning

Absolute positioning allows an element to be placed exactly where you want it in relation to its nearest positioned ancestor (an element with relative, absolute, or fixed positioning). If there is no positioned ancestor, it will be positioned relative to the browser window.

Example:

html

Copy code

```
<!DOCTYPE html><html lang="en"><head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Absolute Positioning Example</title>
  <style>
    .container {
      position: relative;
      height: 200px;
      background-color: lightgray;
    }

    .absolute-box {
      position: absolute;
      top: 20px;
      right: 30px;
      background-color: salmon;
      padding: 10px;
      border: 1px solid black;
    }
  </style></head><body>
  <div class="container">
    <div>Div inside container</div>
    <div class="absolute-box">Div 2 (Absolute)</div>
  </div>
  <div>Div 3 outside container</div></body></html>
```

In This example:

Div 2 is positioned 20px from the top and 30px from the right of its closest positioned ancestor (.container).

Without the position: relative; on .container, the box would be positioned relative to the entire window.

---

4. Fixed Positioning

Fixed positioning keeps an element in a fixed position relative to the browser window, even when the user scrolls.

Example:

html

Copy code

```
<!DOCTYPE html><html lang="en"><head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Fixed Positioning Example</title>
  <style>
    .fixed-box {
      position: fixed;
```

```
            top: 10px;
            right: 10px;
            background-color: lightcoral;
            padding: 10px;
            border: 1px solid black;
        }

        body {
            height: 1500px;
        }
    </style></head><body>
    <div class="fixed-box">Div 2 (Fixed)</div>
    <div>Scroll the page and watch Div 2</div></body></html>
```

In this example:

Div 2 stays in the top-right corner of the browser window even when you scroll down the page. It doesn't move with the rest of the content.

---

5. Sticky Positioning

Sticky positioning is a mix of relative and fixed positioning. It behaves like relative positioning until you scroll past a certain point, at which it becomes fixed.

Example:

html

Copy code

```
<!DOCTYPE html><html lang="en"><head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Sticky Positioning Example</title>
    <style>
        .sticky-box {
            position: sticky;
            top: 0;
            background-color: lightblue;
            padding: 10px;
            border: 1px solid black;
        }

        body {
            height: 2000px;
        }
    </style></head><body>
    <div class="sticky-box">Div 2 (Sticky)</div>
    <div>Scroll the page and see when Div 2 sticks to the top</div></body></html>
```

In this example:

Div 2 behaves like a normal block element when you first load the page. However, when you scroll past it, it will "stick" to the top of the screen and remain there as you continue scrolling.

CSS Units: Absolute vs. Relative

CSS units are used to define the size of elements, fonts, margins, padding, etc. There are two main types of units in CSS: absolute units and relative units. Each has different use cases and behaviors, which impact layout and responsiveness.

---

1. Absolute Units:

Absolute units represent fixed values that do not change based on the context. Some commonly used absolute units include:

px (pixels): A fixed measurement that is equal to one dot on the screen.

pt (points): Commonly used in print media. 1 point is 1/72 of an inch.

Example:

html

Copy code

```
<!DOCTYPE html><html lang="en"><head>
  <style>
    .absolute {
      font-size: 16px; /* Fixed size */
      width: 300px;    /* Fixed width */
    }
  </style></head><body>
  <div class="absolute">This text is 16px, and the box is 300px wide.</div></body></html>
```

In the above example, both the text and the box width will remain the same across different screens. This can make it difficult for the design to adapt to various screen sizes, especially in responsive design.

---

2. Relative Units:

Relative units are based on the size of other elements, making them flexible and adaptive. Commonly used relative units include:

em: Relative to the font size of the element. If the parent element has a font size of 16px, then 1em = 16px.

rem: Relative to the root element’s (html) font size, which is usually 16px by default.

%: Relative to the parent element's size.

Example:

html

Copy code

```
<!DOCTYPE html><html lang="en"><head>
  <style>
    :root {
      font-size: 16px; /* Root font size */
    }
    .relative {
      font-size: 2em;   /* 2 times the parent font size */
      width: 50%;       /* 50% of the parent element width */
    }
  </style></head><body>
  <div class="relative">This text is 2em, and the box is 50% wide.</div></body></html>
```

In this example, the font-size: 2em means the text will be twice as big as the parent element’s font size. The width is set to 50%, so it will be half of the parent element’s width, making it more adaptive to screen sizes.

Key Difference:

Absolute units (px, pt) are fixed, and they do not scale based on the viewport or parent size.

Relative units (em, rem, %) change according to their environment, making them more useful for responsive designs.