Types of selectors in CSS — By Gagan Baghel

Introduction of Selector Types

Selector Type	Description	Example
Simple Selectors	Select elements based on their type, class, ID, or universally.	`p` , `.class` , `#id` , `*`
Combinator	Select elements based on their relationship to other elements.	`div p` , `ul > li` , `h1 + p`
Pseudo-Classes	Target elements based on a specific state or position.	`:hover` , `:nth-child(2)` , `:not()`
Pseudo-Elements	Target specific parts of an element's content.	`::before` , `::first-letter`
Attribute	Target elements based on their attribute values.	`[type="text"]` , `[href^="https"]`

▼ 1. Simple Selectors

Simple selectors are the most basic types of selectors used in CSS. They target elements based on basic characteristics like type, class, ID, or universal selection.

Types of Simple Selectors:

1. **Type Selector (Element Selector):**
   - Selects elements by their tag name.
   - Example:This will target all `<h1>` elements and apply the color blue.

```
h1 {
  color: blue;
}
```

2. **Class Selector (`.`):**

- Selects elements that have a specific class attribute. A class can be shared by multiple elements.
- Example:This will target all elements with the class `highlight`.

```
.highlight {
  background-color: yellow;
}
```

3. **ID Selector (`#`):**
   - Selects elements with a specific `id` attribute. The `id` must be unique within the HTML document.
   - Example:This will target the element with the `id="main-header"`.

```
#main-header {
  font-size: 24px;
}
```

4. **Universal Selector (`*`):**
   - Selects all elements on a webpage.
   - Example:This rule applies a `margin` and `padding` reset to all elements.

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

5. **Group Selector (`,`):**
   - Combines multiple selectors into one rule, allowing you to apply the same styles to different elements.
   - Example:This rule applies the same color style to all `<h1>`, `<h2>`, and `<p>` elements.

```
h1, h2, p {
  color: black;
}
```

▼ **2. Combinator Selectors**

Combinators are used to define the relationship between two or more selectors.

They specify how elements relate to each other in the document structure.

Types of Combinator Selectors:

▼ **Descendant Selector (Space):**

- Targets elements that are nested within other elements (not necessarily direct children).
- Example:This will select all `<p>` elements that are descendants of a `<div>`, regardless of how deep they are nested.

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

▼ **Child Selector (`>`):**

- Selects elements that are direct children of a specified parent.
- Example:This will target only the `<li>` elements that are direct children of a `<ul>`.

```
ul > li {
  list-style-type: none;
}
```

▼ **Adjacent Sibling Selector (`+`):**

- Selects an element that is immediately preceded by a specified sibling element.
- Example:This will target the first `<p>` element that comes immediately after an `<h1>`.

```
h1 + p {
  margin-top: 0;
}
```

▼ **General Sibling Selector (`~`):**

- Selects all siblings of a specified element that appear after it in the document.
- Example:This will select all `<p>` elements that are siblings of an `<h2>` element.

```
h2 ~ p {
  color: gray;
}
```

▼ **3. Pseudo-Classes** Selectors

In-Depth Guide on Pseudo-Classes in CSS

Pseudo-classes in CSS are keywords that are added to selectors to style elements based on their state or structure. They allow you to target elements that are not easily selectable using standard selectors. Pseudo-classes can be divided into different categories based on their function.

Let's explore **dynamic pseudo-classes**, **structural pseudo-classes**, and **UI element pseudo-classes** in the order you provided:

▼ **1. Dynamic Pseudo-Classes**

Dynamic pseudo-classes allow you to style elements based on user interactions such as clicking, hovering, or focusing. These pseudo-classes make your website dynamic and responsive to user behavior.

1.1 `:link`

- **Definition**: Targets unvisited links. It applies styles to all `<a>` elements that have an `href` attribute and haven't been clicked by the user.
- **Syntax**:

```
a:link {
  property: value;
}
```

- **Example**:
  This will style all unvisited links as blue.

```
a:link {
  color: blue;
}
```

1.2 `:visited`

- **Definition**: Targets links that have been visited by the user. The browser tracks visited links using history.
- **Syntax**:

```
a:visited {
  property: value;
}
```

- **Example**:
  This will change the color of visited links to purple.

```
a:visited {
  color: purple;
}
```

1.3 `:active`

- **Definition**: Styles an element while it is being activated (e.g., when the user is clicking on the link or button).
- **Syntax**:

```
element:active {
  property: value;
}
```

- **Example**:
  This will change the color of a link to red when it's being clicked.

```
a:active {
  color: red;
}
```

1.4 :focus

- **Definition**: Targets an element that has received focus, typically used for form elements like inputs. It allows you to change the appearance of an element when the user clicks into it or navigates to it using the keyboard.
- **Syntax**:

```
element:focus {
  property: value;
}
```

- **Example**:
  This will change the border color of an input field to green when the user focuses on it.

```
input:focus {
  border-color: green;
}
```

1.5 `:hover`

- **Definition**: Styles an element when the user hovers over it with the mouse. It is commonly used for interactive elements like buttons, links, or images.
- **Syntax**:

```
element:hover {
  property: value;
}
```

- **Example**:
  This will change the background color of a button to yellow when the user hovers over it.

```
button:hover {
  background-color: yellow;
}
```

▼ 2. Structural Pseudo-Classes

Structural pseudo-classes are used to style elements based on their position within the DOM structure, like being the first or last child, or an nth child in a list.

2.1 `:first-child`

- **Definition**: Targets the first child element within its parent.
- **Syntax**:

```
parent:first-child {
  property: value;
}
```

- **Example**:
  This will apply red color to the first `<p>` element inside its parent

container.

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

2.2 :last-child

- **Definition**: Targets the last child element within its parent.
- **Syntax**:

```
parent:last-child {
  property: value;
}
```

- **Example**:
  This will bold the last `<li>` in an unordered list.

```
li:last-child {
  font-weight: bold;
}
```

2.3 :nth-child(n)

- **Definition**: Targets elements based on their position within their parent. The `n` can be a number, keyword, or formula (`n` starts at 1).
- **Syntax**:

```
element:nth-child(n) {
  property: value;
}
```

- **Example**:

```
tr:nth-child(odd) {
  background-color: lightgray;
}
```

This will style every odd `<tr>` element in a table with a light gray background.

Another example:

```
li:nth-child(3) {
  color: blue;
}
```

This will style the third `<li>` in a list with blue color.

2.4 :first-of-type

- **Definition**: Targets the first element of a specific type (like `<p>`, `<div>`, etc.) within its parent, regardless of its position among siblings of other types.
- **Syntax**:

```
element:first-of-type {
  property: value;
}
```

- **Example**:
  This will apply a font size of 18px to the first `<p>` element in its parent, even if it's not the first child.

```
p:first-of-type {
  font-size: 18px;
}
```

2.5 :last-of-type

- **Definition**: Targets the last element of a specific type within its parent.
- **Syntax**:

```
element:last-of-type {
  property: value;
}
```

- **Example**:
  This will apply a light blue background to the last `<div>` in its parent container.

```
div:last-of-type {
  background-color: lightblue;
}
```

▼ 3. UI Element Pseudo-Classes

UI pseudo-classes are used to target user interface elements based on their state, such as whether a form control is enabled, disabled, or checked.

3.1 `:enabled`

- **Definition**: Targets form elements (like input fields, buttons, etc.) that are enabled (i.e., not disabled and able to be interacted with).
- **Syntax**:

```
input:enabled {
  property: value;
}
```

- **Example**:
  This will apply a green border to all enabled input fields.

```
input:enabled {
  border: 1px solid green;
}
```

3.2 :disabled

- **Definition**: Targets form elements that are disabled (i.e., elements that cannot be interacted with).
- **Syntax**:

```
input:disabled {
  property: value;
}
```

- **Example**:
  This will apply a light gray background to disabled input fields, indicating that they are inactive.

```
input:disabled {
  background-color: lightgray;
}
```

3.3 :checked

- **Definition**: Targets elements like checkboxes or radio buttons that are currently checked.
- **Syntax**:

```
input:checked {
  property: value;
}
```

- **Example**:
  This will style all checked checkboxes with a yellow background.

```
input[type="checkbox"]:checked {
  background-color: yellow;
}
```

Conclusion

CSS pseudo-classes allow you to apply styles based on the state or position of elements, offering great flexibility in creating interactive, dynamic, and well-structured designs. By combining dynamic, structural, and UI pseudo-classes, you can make your web pages more engaging and user-friendly.

Understanding when and how to use these pseudo-classes will enable you to target and style elements effectively, leading to a more responsive and intuitive user interface.

▼ **4. Pseudo-Elements** Selectors

CSS Pseudo-Element Selectors

CSS pseudo-elements allow you to style specific parts of an element or insert content without needing to alter the HTML structure. They are incredibly useful for adding decorative elements, emphasizing specific text parts, or managing advanced styling effects without additional HTML markup. Pseudo-elements are written with a double colon (`::`), although a single colon (`:`) can still be used in some cases for backward compatibility.

Types of Pseudo-Element Selectors:

1. `::before` Pseudo-Element

- **Description**: The `::before` pseudo-element allows you to insert content before the content of an element.
- **Use Case**: This is commonly used to add decorative content or icons without modifying the HTML itself.

Example:

```
p::before {
  content: "Note: ";
  font-weight: bold;
  color: red;
}
```

- **Explanation**: This rule adds the text "Note: " before every `<p>` element, emphasizing it with a bold red font. It's often used for decorative elements, such as icons or visual cues before text content.

---

2. `::after` Pseudo-Element

- **Description**: The `::after` pseudo-element is similar to `::before`, but it inserts content after the content of an element.
- **Use Case**: This is useful for adding closing elements, such as custom icons or other decorative content.

Example:

```
p::after {
  content: " ➡️";
  color: blue;
}
```

- **Explanation**: This rule appends an arrow (➡️) after the content of every `<p>` element, enhancing its visibility or implying a direction. The `::after` pseudo-element is commonly used for decorative purposes or to insert symbols after an element's content.

---

3. `::first-letter` Pseudo-Element

- **Description**: The `::first-letter` pseudo-element targets and styles the first letter of a block-level element (like a paragraph).

- **Use Case**: This is often used in typography to create a "drop cap" effect or emphasize the first letter in a sentence or block of text.

Example:

```
p::first-letter {
  font-size: 2em;
  color: darkgreen;
  font-weight: bold;
}
```

- **Explanation**: The first letter of every paragraph (`<p>`) will be styled with a larger font size, dark green color, and bold weight, creating a visually striking drop cap effect. This is often seen in news articles or blog posts for stylistic purposes.

---

4. `::first-line` Pseudo-Element

- **Description**: The `::first-line` pseudo-element applies styles to the first line of a block-level element's text.
- **Use Case**: This is useful when you want to differentiate the first line of a paragraph or block of text for readability or emphasis.

Example:

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

- **Explanation**: The first line of every paragraph will appear in bold and uppercase. This pseudo-element is often used to make the first line stand out, such as in a heading or introductory text.

---

5. `::placeholder` Pseudo-Element

- **Description**: The `::placeholder` pseudo-element styles the placeholder text inside form elements (e.g., `<input>`, `<textarea>`).
- **Use Case**: This is useful when you want to customize the appearance of placeholder text, such as changing its color or font.

Example:

```
input::placeholder {
  color: gray;
  font-style: italic;
}
```

- **Explanation**: This rule will make the placeholder text in `<input>` fields appear gray and italic. Customizing placeholder text improves the UX by giving form fields a more polished and distinctive appearance.

---

6. `::selection` Pseudo-Element

- **Description**: The `::selection` pseudo-element allows you to style the part of the document that the user has selected or highlighted (usually with a mouse or keyboard).
- **Use Case**: This is great for creating a custom highlight effect when users select text on a webpage.

Example:

```
::selection {
  background-color: yellow;
  color: black;
}
```

- **Explanation**: When users highlight any text on the page, the highlighted text will appear with a yellow background and black text. This can create a more engaging or brand-consistent selection effect, different from the default browser highlight.

---

7. `::marker` Pseudo-Element

- **Description**: The `::marker` pseudo-element styles the marker (bullet or number) of list items.
- **Use Case**: This is useful when customizing the bullets or numbers in ordered and unordered lists.

Example:

```
li::marker {
  color: orange;
  font-size: 1.5em;
}
```

- **Explanation**: The markers (bullets or numbers) in list items (`<li>`) will be orange and larger than the default size. This provides a way to customize the look and feel of list markers to match the design of the webpage.

---

8. `::backdrop` Pseudo-Element

- **Description**: The `::backdrop` pseudo-element targets and styles the backdrop of elements displayed in full-screen mode.
- **Use Case**: This is typically used when an element (such as a modal or video) is in full-screen mode and you want to style the backdrop behind the element.

Example:

```
::backdrop {
  background-color: rgba(0, 0, 0, 0.8);
}
```

- **Explanation**: When an element is in full-screen mode, the backdrop will appear with a dark semi-transparent background (`rgba(0, 0, 0, 0.8)`), providing better visibility for the content in focus.

---

9. `::file-selector-button` Pseudo-Element

- **Description**: The `::file-selector-button` pseudo-element styles the button of a file input element.
- **Use Case**: This is useful for customizing the appearance of the button used to upload files in forms.

Example:

```
input[type="file"]::file-selector-button {
  background-color: lightblue;
  border: 1px solid blue;
  padding: 5px 10px;
}
```

- **Explanation**: This rule customizes the button inside a file input field by giving it a light blue background, a blue border, and some padding. It's a great way to improve the appearance of the default file input button.

Practical Use Cases for Pseudo-Elements

1. **Custom Bullet Lists**: You can use `::before` and `::after` to add custom icons or decorative bullets to lists.

   ```
   li::before {
     content: "• ";
     color: purple;
   }
   ```

2. **Tooltip Creation**: Use `::before` or `::after` with the `content` property to create simple tooltips or additional information without changing the HTML.

   ```
   a::after {
     content: " (opens in a new tab)";
     font-size: 0.8em;
   ```

```
  color: gray;
}
```

3. **Drop Caps**: Use `::first-letter` to create classic "drop cap" effects, typically seen in printed media like books or magazines.

```
p::first-letter {
  font-size: 3em;
  float: left;
  margin-right: 10px;
}
```

4. **Input Placeholder Customization**: Customize form input placeholder text to match your website's design, improving user experience.

```
textarea::placeholder {
  font-style: italic;
  color: darkgray;
}
```

These are the ones we've already covered. Let's now expand on **other lesser-known pseudo-elements**.

---

10. `::cue` Pseudo-Element

- **Description**: The `::cue` pseudo-element is used to style WebVTT (Web Video Text Tracks) cues, such as subtitles or captions in HTML5 videos.
- **Use Case**: You can apply custom styles to captions or subtitles in videos.

Example:

```
css
Copy code
::cue {
```

```
  color: white;
  background-color: black;
  font-size: 1.2em;
}
```

- **Explanation**: This rule styles the subtitles or captions in video tracks with white text on a black background and enlarges the font size. It enhances the readability and appearance of video captions.

---

11. `::part` Pseudo-Element

- **Description**: The `::part()` pseudo-element is used in conjunction with the `part` attribute for styling Shadow DOM parts.
- **Use Case**: This pseudo-element allows you to style shadow DOM elements from the light DOM by targeting specific parts.

Example:

```
css
Copy code
my-element::part(button) {
  background-color: green;
  color: white;
}
```

- **Explanation**: If a custom element exposes a part called "button," this rule will apply styles to that part, even if it exists inside a shadow DOM. This helps developers create reusable components with customizable styling.

---

12. `::slotted` Pseudo-Element

- **Description**: The `::slotted()` pseudo-element is used to style elements that are placed into slots within shadow DOM.

- **Use Case**: It allows you to style elements passed into a custom element's slots.

Example:

```
css
Copy code
::slotted(p) {
  color: blue;
  font-style: italic;
}
```

- **Explanation**: This rule targets paragraphs (`<p>`) that are slotted into a custom element, changing their color to blue and italicizing them. It's useful for customizing content passed into a shadow DOM component.

---

13. `::grammar-error` Pseudo-Element

- **Description**: The `::grammar-error` pseudo-element is used to style text that has a detected grammar error (though not widely supported across browsers yet).
- **Use Case**: This pseudo-element can be used to apply custom styles to grammar errors highlighted by the browser.

Example:

```
css
Copy code
::grammar-error {
  background-color: yellow;
  text-decoration: underline red;
}
```

- **Explanation**: This rule highlights grammar errors with a yellow background and underlines them in red. It is useful for enhancing accessibility or custom grammar-checking features, although support is limited.

---

14. `::spelling-error` Pseudo-Element

- **Description**: The `::spelling-error` pseudo-element is used to style text that has a detected spelling error (though not widely supported).
- **Use Case**: You can use this to apply custom styles to spelling errors highlighted by the browser.

Example:

```
css
Copy code
::spelling-error {
  text-decoration: underline wavy red;
}
```

- **Explanation**: This rule applies a wavy red underline to spelling errors, making them more visually distinctive. This pseudo-element is especially useful in text-editing applications or websites with built-in spell-check functionality.

---

15. `::target-text` Pseudo-Element

- **Description**: The `::target-text` pseudo-element allows styling of the part of the document that is the target of a fragment identifier (like `#id` in URLs).
- **Use Case**: When you use a URL that points to a specific part of the page, this pseudo-element allows you to style the targeted text.

Example:

```
css
Copy code
```

```
::target-text {
  background-color: lightyellow;
  outline: 2px dashed red;
}
```

- **Explanation**: This rule highlights the text targeted by a URL fragment with a light yellow background and a red dashed outline. It's useful for making the targeted section of a page stand out when the user navigates to it.

---

16. `::placeholder-shown` (Special Mention)

- **Description**: This pseudo-class is different from pseudo-elements but often confused with them. It selects form elements when a placeholder is visible.
- **Use Case**: It's useful when you need to style form elements that still have their placeholder text displayed.

Example:

```
input:placeholder-shown {
  background-color: lightblue;
}
```

- **Explanation**: This rule applies a light blue background to input fields where the placeholder text is visible, providing a visual cue that the field is empty.

▼ **Conclusion**

CSS pseudo-elements are an essential tool for enhancing web design. They allow developers to style specific parts of an element or insert content without modifying the HTML structure. Whether you need to emphasize the first letter of a paragraph, add decorative content, or

customize form placeholders, pseudo-elements provide the flexibility to create visually appealing effects with minimal code.

By mastering pseudo-elements, you can greatly improve the aesthetics and user experience of your website, keeping your HTML clean and maintainable while applying advanced styles.

▼ Summary

Summarized List of Pseudo-Elements

1. `::before`: Insert content before an element.
2. `::after`: Insert content after an element.
3. `::first-letter`: Style the first letter of a block of text.
4. `::first-line`: Style the first line of a block of text.
5. `::placeholder`: Style the placeholder text in input fields.
6. `::selection`: Style the text that the user selects.
7. `::marker`: Style the bullets or numbers in lists.
8. `::backdrop`: Style the backdrop of elements in full-screen mode.
9. `::file-selector-button`: Style the file upload button in forms.
10. `::cue`: Style WebVTT (video subtitles or captions) cues.
11. `::part`: Style shadow DOM parts.
12. `::slotted`: Style elements inserted into slots within shadow DOM.
13. `::grammar-error`: Style detected grammar errors.
14. `::spelling-error`: Style detected spelling errors.
15. `::target-text`: Style the text targeted by a fragment identifier.
16. `::placeholder-shown`: Style form elements when the placeholder is shown (a pseudo-class, not a pseudo-element).

▼ **5. Attribute Selectors**

CSS Attribute Selectors

Attribute selectors in CSS allow you to style HTML elements based on the presence of specific attributes or the values of those attributes. These selectors provide a dynamic way to apply styles without needing to rely on classes or IDs. Attribute selectors can be especially useful when working with forms, links, or custom HTML data attributes (`data-*`), providing great flexibility.

Types of CSS Attribute Selectors:

1. `[attribute]` Selector

- **Description**: Selects elements that have a specified attribute, regardless of the attribute's value.
- **Use Case**: This selector is useful when you want to apply styles to any element that has a particular attribute, even if the value of the attribute varies.

Example:

The following rule will select any element that has the `type` attribute (e.g., `<input type="text">`, `<button type="submit">`):

```
[type] {
  border: 1px solid black;
}
```

- **Explanation**: This CSS rule applies a black border to all elements that contain the `type` attribute, regardless of the actual value of the `type` attribute. This is helpful when targeting all input types in a form or button elements.

2. `[attribute="value"]` Selector

- **Description**: Selects elements that have a specific attribute with an exact value.

- **Use Case**: This selector is useful when you want to style only elements where a particular attribute has a specific value.

Example:

The following rule targets only `<input>` elements where `type="text"`:

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

- **Explanation**: This rule will apply a light gray background to all `<input>` elements with a `type` attribute exactly equal to "text." It won't affect input types like `checkbox` or `password` because the values are different.

---

3. `[attribute~="value"]` Selector

- **Description**: Selects elements whose attribute value contains a specific word in a space-separated list.
- **Use Case**: This selector is typically used with class attributes to style elements that include a specific class name as one of several class names.

Example:

The following rule selects elements with a class attribute that includes the word `featured`:

```
[class~="featured"] {
  border: 2px solid red;
}
```

- **Explanation**: If an element has multiple class names like `class="featured post popular"`, the selector will match it because `featured` is one of the class names. This is useful for selectively targeting elements with multiple classes.

---

4. `[attribute^="value"]` Selector

- **Description**: Selects elements whose attribute value starts with a specific value.
- **Use Case**: This selector is often used to target links or elements where an attribute begins with a certain prefix.

Example:

The following rule targets `<a>` elements where the `href` attribute starts with "https":

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

- **Explanation**: This rule applies a green color to all links (anchor tags) where the `href` attribute starts with "https." This is useful for distinguishing secure links (those using HTTPS) from non-secure ones (HTTP).

---

5. `[attribute$="value"]` Selector

- **Description**: Selects elements whose attribute value ends with a specific value.
- **Use Case**: This selector is commonly used to target file extensions or elements that end with a particular string.

Example:

The following rule targets `<a>` elements where the `href` attribute ends with ".pdf":

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

- **Explanation**: This rule applies a red color to all links that point to PDF files. It's especially useful for indicating downloadable files with specific extensions.

6. `[attribute*="value"]` Selector

- **Description**: Selects elements whose attribute value contains a specific substring anywhere in the value.
- **Use Case**: This selector is versatile, useful for finding elements where a certain word or string is present anywhere in the attribute value.

Example:

The following rule targets `<a>` elements whose `href` attribute contains the word "example":

```
[href*="example"] {
  text-decoration: underline;
}
```

- **Explanation**: This will apply underlining to any link that contains "example" anywhere in its URL. It's great for selectively styling URLs or elements based on a part of the string.

---

7. `[attribute|="value"]` Selector

- **Description**: Selects elements where the attribute value is exactly the specified value or starts with it followed by a hyphen (commonly used for language attributes).
- **Use Case**: This selector is ideal for targeting language or region-specific attributes that use a hyphen to separate values (e.g., `en-US`).

Example:

The following rule targets elements with a `lang` attribute set to "en" or that starts with "en-", such as "en-US":

```
[lang|="en"] {
  font-style: italic;
}
```

- **Explanation**: This rule will italicize text in any element where the `lang` attribute is set to "en" (English) or a variation like "en-US" (American English). It's useful for internationalized websites.

Practical Use Cases for Attribute Selectors

1. **Form Styling**:
   You can easily style specific form elements like text fields, checkboxes, or buttons without adding extra classes.

```
input[type="submit"] {
  background-color: blue;
  color: white;
}
```

2. **Link Styling**:
   Customize links based on their file types (like PDFs) or external/internal URLs.

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

3. **Targeting Data Attributes**:
   Modern frameworks use `data-*` attributes extensively. You can style elements based on custom attributes like this:

```
[data-toggle="modal"] {
  cursor: pointer;
  font-weight: bold;
}
```

Conclusion

CSS attribute selectors provide a powerful way to dynamically style HTML elements based on the presence and value of their attributes. They allow you to target elements more efficiently, especially in cases where HTML structure might change, or you need more precise control over elements beyond just classes and IDs.

By using attribute selectors, you can apply styles in a flexible, reusable manner, making your CSS more adaptable to different situations without extra markup or JavaScript.