Quiz:-Create the following page inside a div element

Css Example

This is an example pagaraph Link one

This is an example pagaraph Link Two

This is an example pagaraph Link Three

This is an example pagaraph Link four

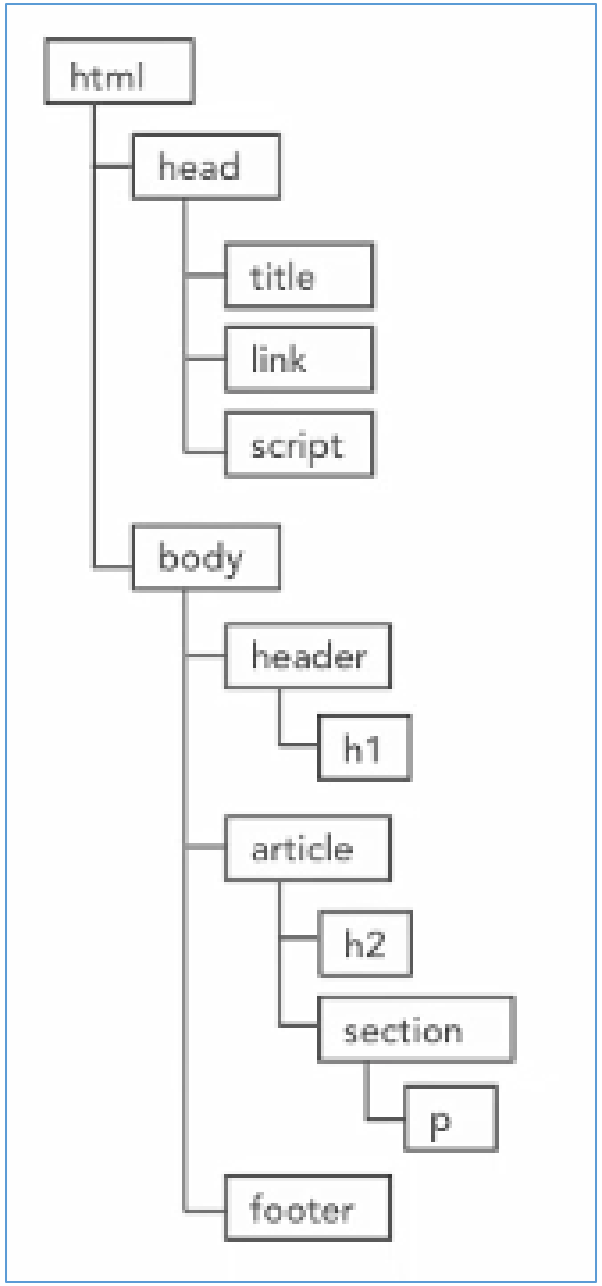
This is an example pagaraph Link five

This is an example pagaraph Link six

This is an example pagaraph Link seven

Document Object Model (DOM)

- **A structural representation of all elements** in an HTML document, often used by browsers and scripts to **traverse through the documents and retrieve content or apply behaviors.**
  - Browsers apply styles to elements by **traversing through the tree**.
- There are times, however, when the DOM alone cannot help you target elements, such as users hovering over an element or focusing on a form element.
- Pseudo-class Selectors: **allow us to target elements or instances that lie outside of the DOM or are too specific for simple selectors to target.**



CSS Selectors
> Pseudo-Class Selectors(PC selectors)

- **Dynamic pseudo-class selectors** target elements based on something other than attributes, content, or element type. Usually refers to something that can change over time, or that is based on user interaction.
- **UI element state pseudo-class selectors** target specific user interface elements in regards to whether or not they are currently enabled. (:enabled, :disabled, :checked).
- **Structural pseudo-class selectors** target elements based on specialized information within the DOM that simple selectors cannot target. This can be pattern-matching, child-to-parent relationships, or other structural information. (Example: **nth** child selector)
- **Other pseudo-class selectors** exist that can't really be grouped into any one category. These selectors give you even more specific targeting capabilities based on things like language or URL targeting.
- Please Read: https://www.w3.org/community/webed/wiki/CSS/Selectors

CSS Selectors
> Pseudo-Class Selectors

- Is used to define **a special state of an element**
- Can be used to style an element when a user mouse hovers over it, visited and unvisited links and getting focus.
- Pseudo-class selectors consists of a colon (:) followed by the selector.

  :hover pseudo-class selector
- Usually they are preceded by the element you wish to target based on its state.

  a:hover targeted pseudo-class selector
- And can be combined with other simple selectors as well.

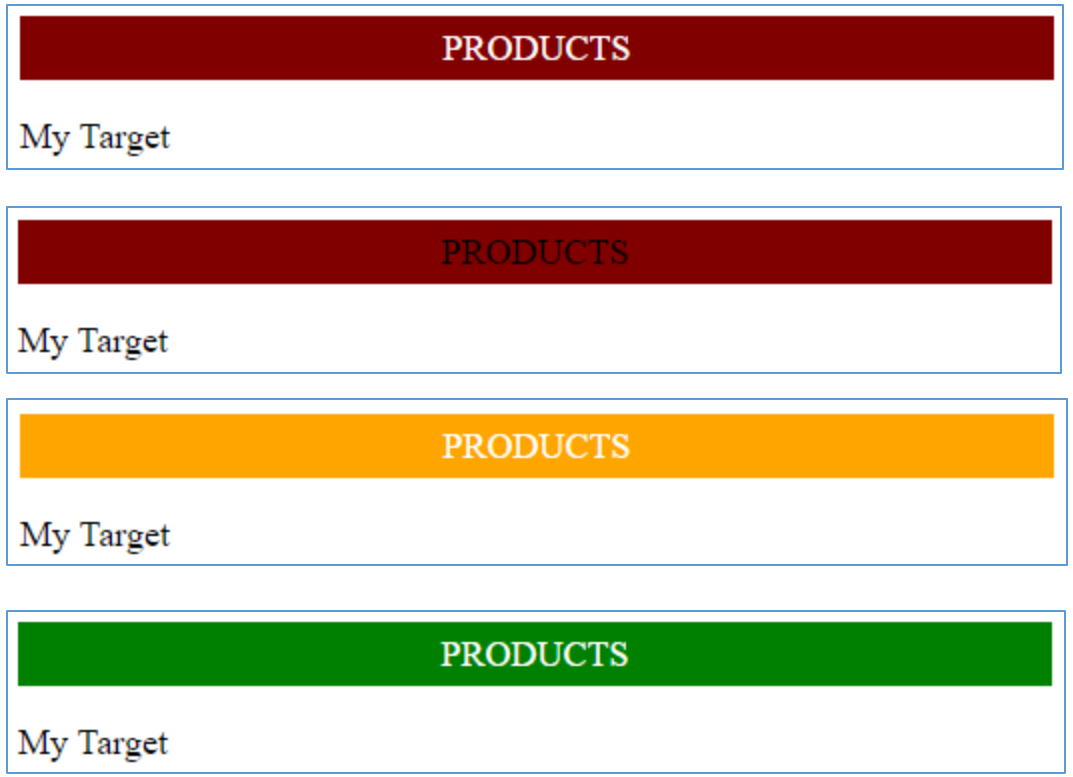
  ul a:hover descendent selector with a pseudo-class selector

CSS Selectors
> Dynamic PC Selectors

- **a:link** :- targets all links on the page. (Any link with an href attribute)
- **a:visited** :- targets all links with an address that the user already visited
- **a:hover** :- targets all links that is currently hovered over
- **a:active** :- targets all links that are either in focus or currently being clicked on.
- The order is very important. If you put the **visited styling after the hover styling, then the visited styling will override the hover style.**
- Developers usually group the link and visited styles together and the hover and active styles together.
- **a:target** :- has more to do with the URL than the actual page itself. If a URL is set to focus to a specific section (link) on the page, the style is applied.

CSS Selectors
> Dynamic PC Selectors

```
<a href="products.htm" title="products">Products</a>
<p><a id="section1">My Target</a></p>
```




```
<style>
    a:link {
        text-decoration: none;
        background: maroon;
        color: white;
        display: block;
        padding: 5px 10px;
        text-align: center;
        text-transform: uppercase;
        margin-bottom: 2px;
    }

    a:visited {
        color: black;
    }

    a:hover {
        background: orange;
    }

    a:active {
        background: green;
    }
</style>
```

CSS Selectors
> Dynamic PC Selectors

```
<style>
    a:target{
        color: red;
    }
</style>
```

- URL without fragment (http://localhost:57823/pseudo-class.htm)

My Target

- URL with fragment (http://localhost:57823/pseudo-class.htm#section1)

My Target

Quiz:-Apply Dynamic PC Selectors for a link one (hint:-first give a class name for link one)

- Link and visited =green color
- Hover and active=brown color with 40px font size

CSS Selectors
> Structural PC Selectors

- Allow you to target elements based on more complex patterns with in the DOM.
  - Example: target an element with a condition whether or not it is the first child of an element, or whether it is the only child of it and other factors.
- **first-child**

```
<style>
    span:first-child {
        font-size:1.3em;
        font-weight: bold;
        color: maroon;
    }
</style>
```

  - Find every span in the document and if it’s the first child of its parent element go ahead and give it the styling.

CSS Selectors
> Structural PC Selectors

- **first-of-type**

```
<style>
    h2:first-of-type{
        border-bottom: 1px solid #42403E;
    }
</style>
```

  - Find every h2 in the document and if it's the first h2 child of its parent element go ahead and give it the styling.

- **only-child**

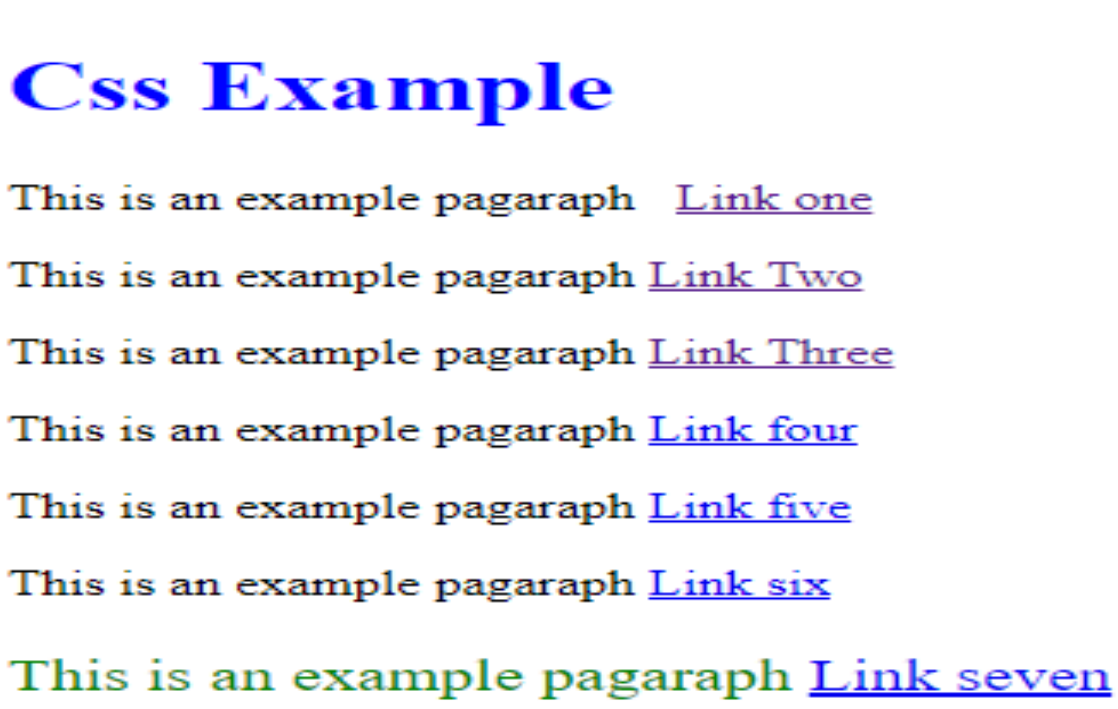
```
p:only-child{
    font-style: italic;
    text-align: center;
}
```

  - Whenever you find a paragraph, and that paragraph is the only child inside of a parent go ahead and give that styling.

CSS Selectors
> Structural PC Selectors

- **First-child**
- **last-child**
- **First-of-type**
- **last-of-type**
- **Only-child**
- **only-of-type**

Quiz:-Use Structural PC Selectors to change color & size of h1 & last paragraph



- **First-child**
- **last-child**
- **First-of-type**
- **last-of-type**
- **Only-child**
- **only-of-type**

CSS Selectors
> Structural PC Selectors

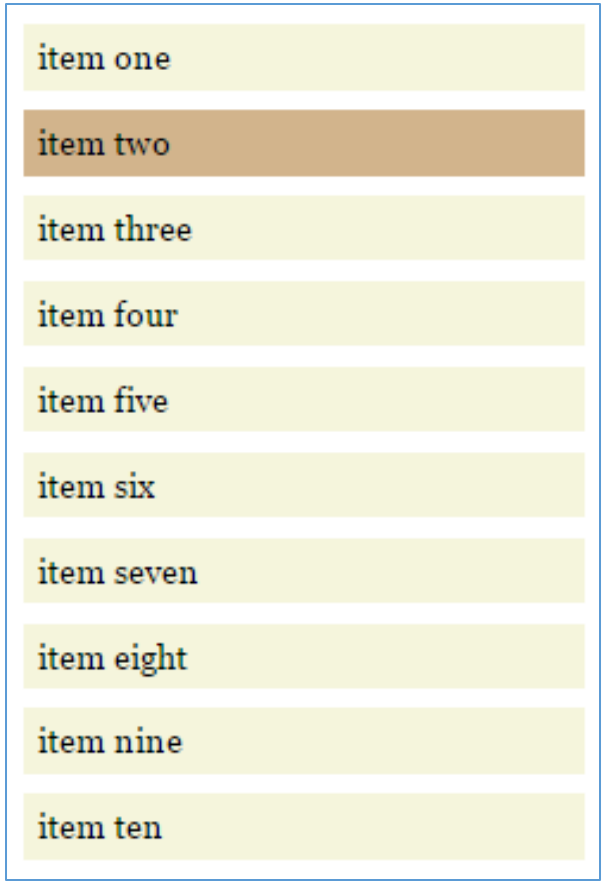
- Of all the structural pseudo-class selectors **:nth-child() pseudo-class selector is the most complex of them all.**
- nth-child selectors allow us to target elements based on patterns that describes which elements within a parent you should target.
- We have four flavors of nth-child selectors.
  - nth-child()
  - nth-of-type()
  - nth-last-child()
  - nth-last-of-type()

CSS Selectors
> Structural PC Selecto

- **nth-child(2)**

```html
<ol>
    <li>item one</li>
    <li>item two</li>
    <li>item three</li>
    <li>item four</li>
    <li>item five</li>
    <li>item six</li>
    <li>item seven</li>
    <li>item eight</li>
    <li>item nine</li>
    <li>item ten</li>
</ol>
```

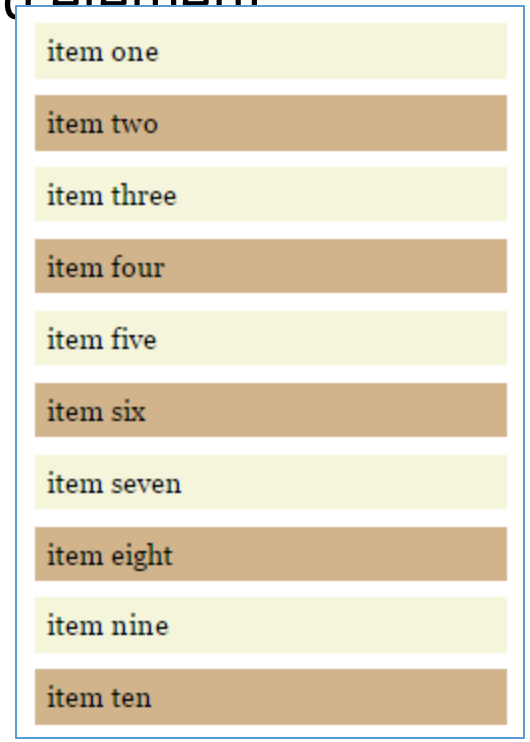
```css
<style>
    body {
        font-family: Georgia;
        font-size: 80%;
    }

    ol {
        list-style: none;
    }

    li {
        background: beige;
        padding: 5px;
        margin: .6em 0;
        width: 200px;
    }

    li:nth-child(2)
    {
        background: tan;
    }
</style>
```



CSS Selectors
> Structural PC Selectors

- **nth-child(2n)**
  - n is a grouping element
  - On the above element, target every second element

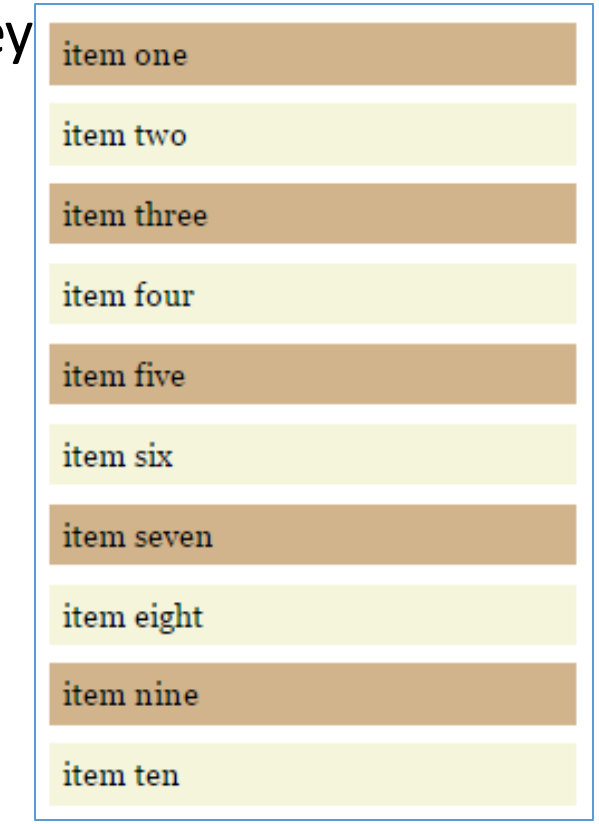
```
li:nth-child(2n)
{
    background: tan;
}
```



CSS Selectors
> Structural PC Selectors

- **nth-child(odd)**
  - Instead of grouping, one can use key
  - **even** is also another keyword

```
li:nth-child(odd)
{
    background: tan;
}
```



Quiz:-Use **nth-child to create the following**

Css Example

This is an example pagaraph Link one

This is an example pagaraph Link Two

This is an example pagaraph Link Three

This is an example pagaraph Link four

This is an example pagaraph Link five

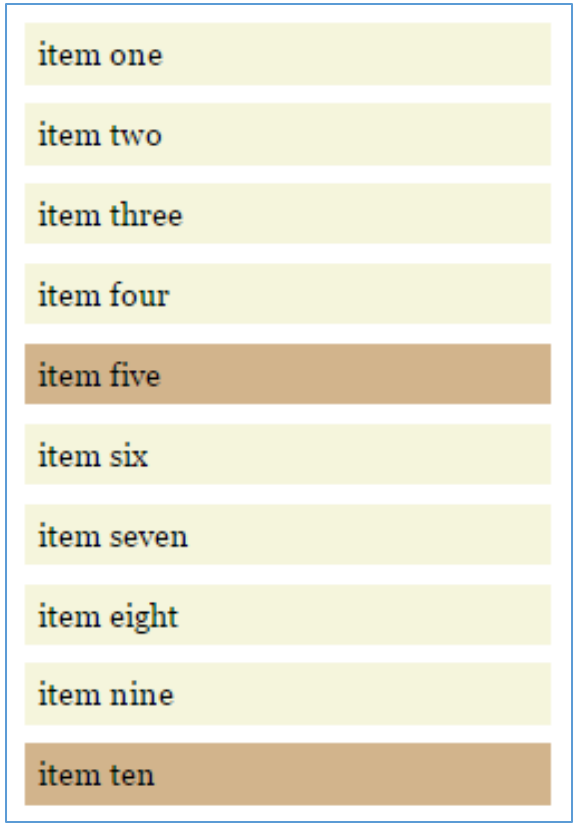
This is an example pagaraph Link six

This is an example pagaraph Link seven

CSS Selectors
> Structural PC Selectors

- **nth-child(2n+1)**
  - The 1 is called offset (beginning with this one)

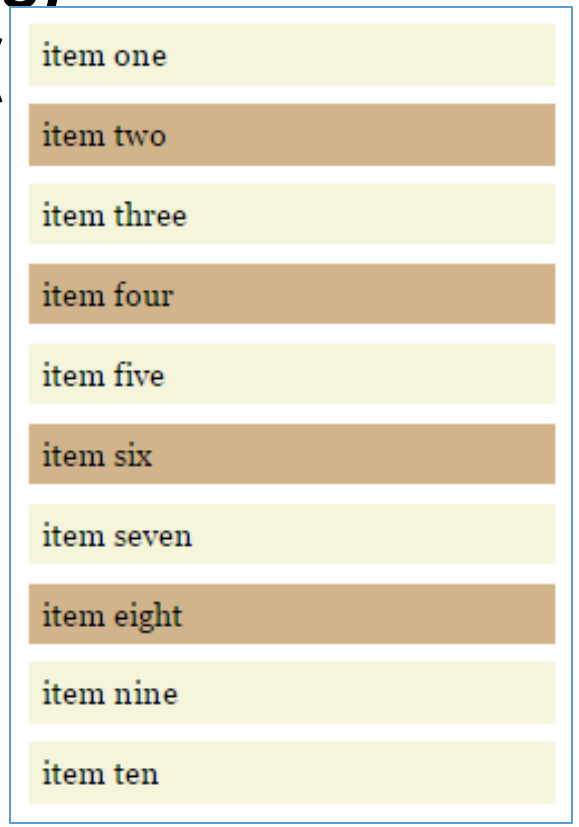
```
li:nth-child(5n+5)
{
    background: tan;
}
```



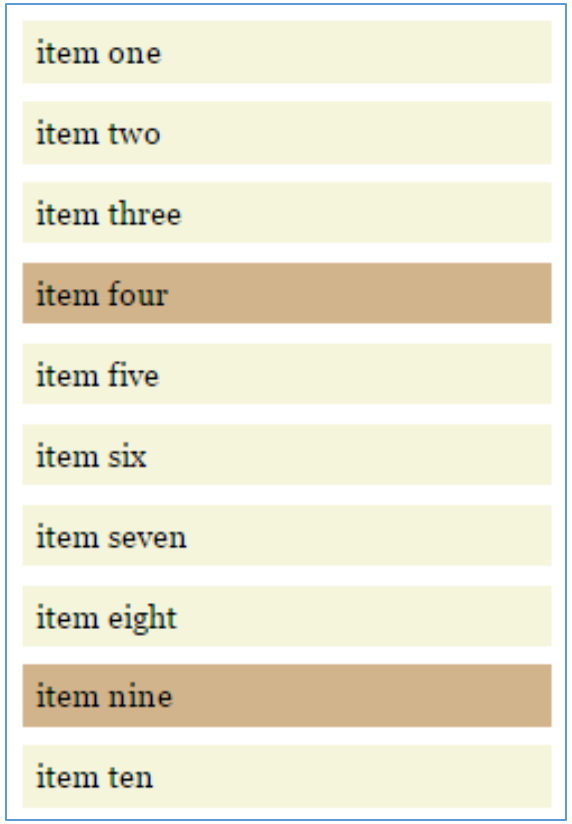
CSS Selectors
> Structural PC Selectors

- **nth-child(-2n+8)**
  nth-child(

```
li:nth-child(-2n+8)
{
    background: tan;
}
```




```
li:nth-child(5n-1)
{
    background: tan;
}
```



Quiz:-Use nth-child

Css Example

This is an example pagaraph Link one

This is an example pagaraph Link Two

This is an example pagaraph Link Three

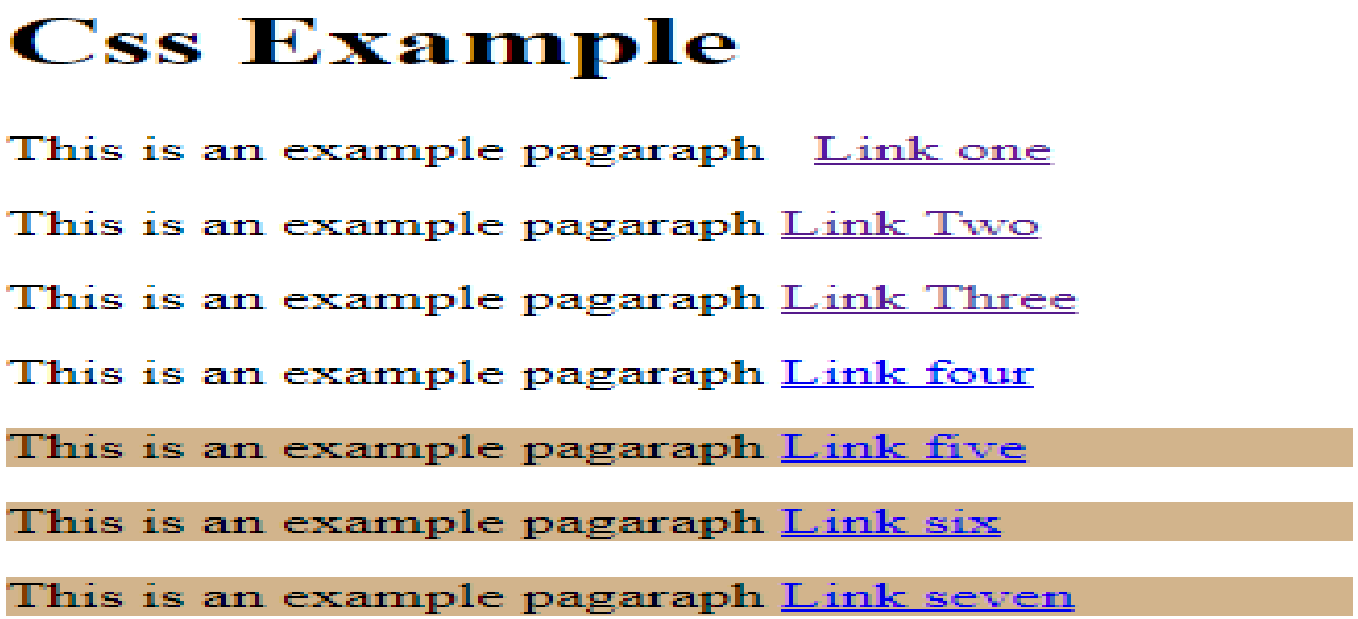
This is an example pagaraph Link four

This is an example pagaraph Link five

This is an example pagaraph Link six

This is an example pagaraph Link seven

Quiz:-Use nth-last-child



Quiz:-Use nth-child & nth-last-child

Css Example

This is an example pagaraph Link one

This is an example pagaraph Link Two

This is an example pagaraph Link Three

This is an example pagaraph Link four

This is an example pagaraph Link five

This is an example pagaraph Link six

This is an example pagaraph Link seven

Quiz:-Use nth-child & nth-last-child

Css Example

This is an example pagaraph Link one

This is an example pagaraph Link Two

This is an example pagaraph Link Three

This is an example pagaraph Link four

This is an example pagaraph Link five

This is an example pagaraph Link six

This is an example pagaraph Link seven

CSS Selectors
> Pseudo-elements

- Pseudo-elements are similar to pseudo-classes in many ways as they **allow you to access content beyond the normal capabilities of the DOM.**
  - For Example: Using pseudo-element selectors you could target the **first line or first letter of an element**.
- Pseudo-element selectors also allow us to create what is know as generated content.
  - That is actually place content on the page that is not contained in the structure of the document.

CSS Selectors
> Pseudo-elements

- **p::first-line** (written like p:first-line in CSS 2)

```
<style>
    p::first-line{
        font-family: Georgia;
        font-style: italic;
    }
</style>
```

This is a paragraph inside the main content. I would like to have a drop-cap for this paragraph, and italicize the first line. This should give me dramatic effect I'm looking for. It's not that I like drama, just that I need a little more text here, and wanted to show a more real-world example of pseudo-element selectors can help you write more efficient styles.

```
<p>
    This is a paragraph inside the main content.
    I would like to have a drop-cap for this paragraph,
    and italicize the first line. This should give me
    dramatic effect I'm looking for.
    It's not that I like drama, just that
    I need a little more text here,
    and wanted to show a more real-world example of
    pseudo-element selectors can help you write
    more efficient styles.
</p>
```

CSS Selectors
> Pseudo-elements

- **p::first-letter**

This is a paragraph inside the main content. I would like to have a drop-cap for this paragraph, and italicize the first line. This should give me dramatic effect I'm looking for. It's not that I like drama, just that I need a little more text here, and wanted to show a more real-world example of pseudo-element selectors can help you write more efficient styles.

```
<style>
    p::first-line{
        font-family: Georgia;
        font-style: italic;
    }
    p::first-letter{
        font-size: 2em;
        font-family: Georgia;
        float: left;
        padding: 0px 5px 0 0;
    }
</style>
```

```
<p>
    This is a paragraph inside the main content.
    I would like to have a drop-cap for this paragraph,
    and italicize the first line. This should give me
    dramatic effect I'm looking for.
    It's not that I like drama, just that
    I need a little more text here,
    and wanted to show a more real-world example of
    pseudo-element selectors can help you write
    more efficient styles.
</p>
```

CSS Selectors
> Pseudo-elements

- Generated content allows us to place content on the page.
  - **:before and :after**
  - Doesn’t show up on the DOM tree
- In order to work with :before and :after, we use content attribute.
  - Content can be: string (Text content), u, counters(chapters), open-quote and close-quote, and attr(x).

CSS Selectors
> Pseudo-elements

```
<a href="http://speedtest.net" title="visit speedtest.net">Internet speed test website</a>
```

```
<style>
    a:after{
        content: " (outside link)";
        color: #666;
    }
</style>
```

```
<style>
    a:after{
        content: attr(href);
        color: #666;
    }
</style>
```

Internet speed test website (outside link)

Internet speed test websitehttp://speedtest.net

Quiz:-Use :after and content & attr(x) to create the following

Css Example

This is an example pagaraph Link one [#frag]

This is an example pagaraph Link Two [#two]

This is an example pagaraph Link Three [#three]

This is an example pagaraph Link four [#four]

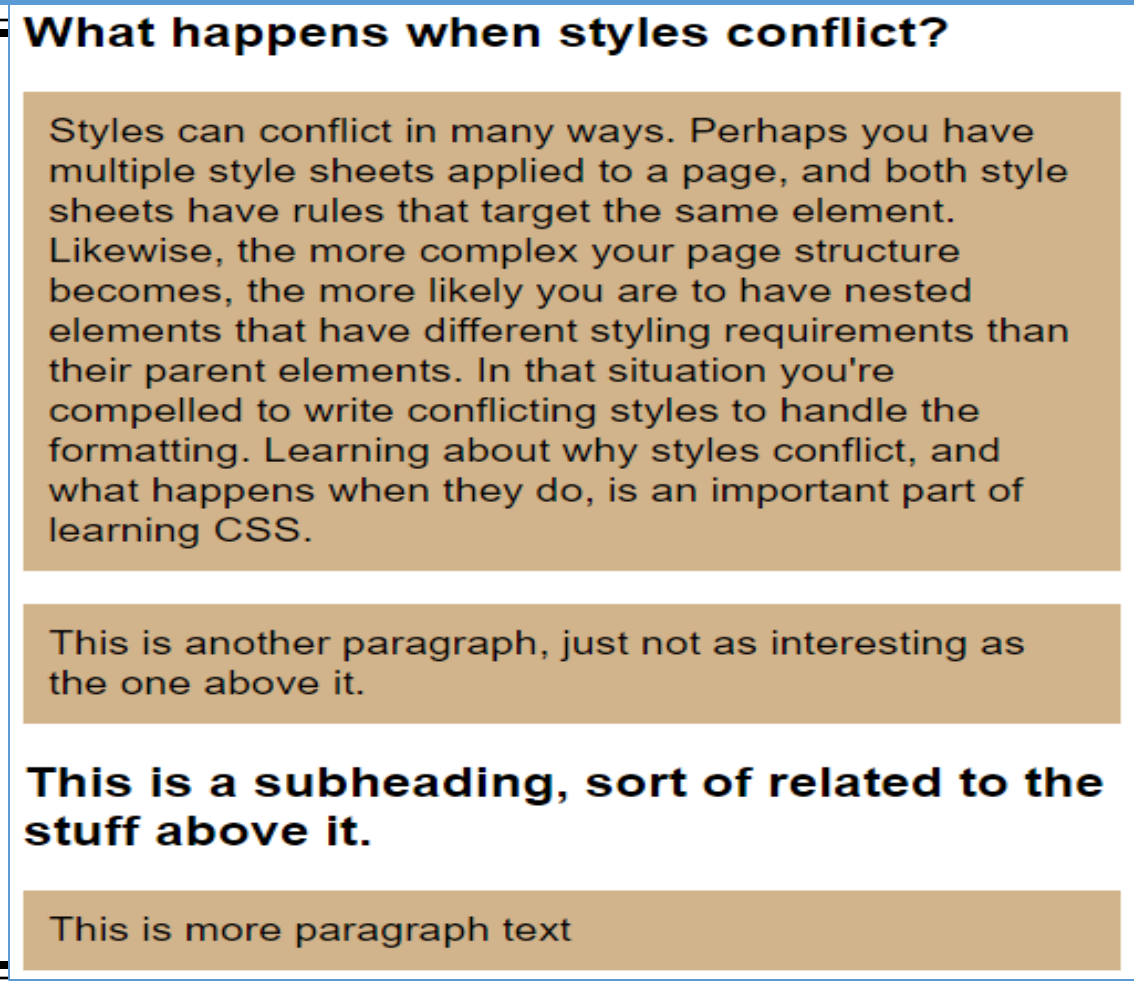
This is an example pagaraph Link five [#five]

This is an example pagaraph Link six [#six]

This is an example pagaraph Link seven [#seven]

CONFLICT IN CSS

1. The Cascade
2. Inheritance
3. Specificity



Conflict in CSS

- Unless every page you create only holds a single paragraph, you're eventually going to have to deal with styles that conflict with one another.
- Anytime you **have multiple rules that target the same element you have conflicting styles.**

```
<article>
    <h1>What happens when styles conflict?</h1>
    <p>Styles can conflict in many ways.
    Perhaps you have multiple style sheets applied to a page,
    and both style sheets have rules that target the same element.
    Likewise, the more complex your page structure becomes,
    the more likely you are to have nested elements that have different
    styling requirements than their parent elements.
    In that situation you're compelled to write conflicting styles to handle the formatting.
    Learning about why styles conflict,
    and what happens when they do, is an important part of learning CSS.</p>
    <p>This is another paragraph, just not as interesting as the one above it.</p>
    <h2>This is a subheading, sort of related to the stuff above it.</h2>
```

Conflict in CSS

- It is not always easy to catch conflicting styles.

```
<style>
    body {
        font-family: Arial;
        font-size: 90%;
    }

    p {
        font-size: 1em;
        color: white;
        background: tan;
    }

    p {
        font-size: 1.2em;
        color: black;
        padding: 10px;
        background: tan;
    }
</style>
```

What happens when styles conflict?

Styles can conflict in many ways. Perhaps you have multiple style sheets applied to a page, and both style sheets have rules that target the same element. Likewise, the more complex your page structure becomes, the more likely you are to have nested elements that have different styling requirements than their parent elements. In that situation you're compelled to write conflicting styles to handle the formatting. Learning about why styles conflict, and what happens when they do, is an important part of learning CSS.

This is another paragraph, just not as interesting as the one above it.

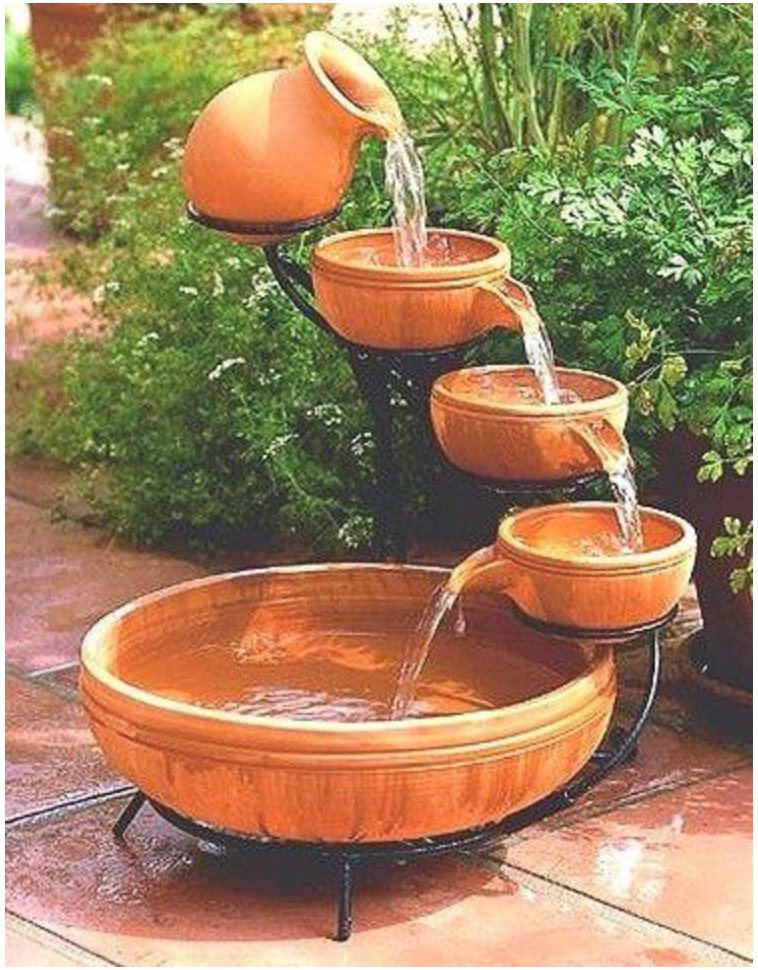
This is a subheading, sort of related to the stuff above it.

Conflict in CSS

- When the properties conflict the **properties of one rule will replace the other**.
- If they don’t conflict with one another, the properties are added and the rules are cumulative leaving you with a mixture of the two rules.
- **What determines which properties are used in the event of a conflict? That is driven by three principles.**
  - The Cascade
  - Inheritance
  - Specificity

The Cascade

- The cascade can be summed up as, the last rule applied wins.
  - Styles are applied in the order they are found, and in the event of a conflict the style that comes last is the one that is used.



CSS Rules Overriding

Here is the rule to override any Style Sheet Rule.

- Any inline style sheet takes the highest priority. So, it will override any rule defined in <style>...</style> tags or the rules defined in any external style sheet file.
- Any rule defined in <style>...</style> tags will override the rules defined in any external style sheet file.
- Any rule defined in the external style sheet file takes the lowest priority, and the rules defined in this file will be applied only when the above two rules are not applicable.

The Cascade
> Example

```
<!DOCTYPE HTML>
<html>
<head>
    <meta charset="UTF-8">
    <title>the cascade</title>
    <link href="cascade.css" rel="stylesheet" type="text/css" />
    <style>
        body {
            font-family: Arial;
            font-size: 120%;
        }
    </style>
</head>
<body>
    <p>I require styling</p>
</body>
</html>
```

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

I require styling

The Cascade
> Example…

```html
<!DOCTYPE HTML>
<html>
<head>
    <meta charset="UTF-8">
    <title>the cascade</title>
    <link href="cascade.css" rel="stylesheet" type="text/css" />
    <style>
        body {
            font-family: Arial;
            font-size: 120%;
        }

        p {
            color: blue;
        }
    </style>
</head>
<body>
    <p>I require styling</p>
</body>
</html>
```

css

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

I require styling

The Cascade
> Example…

-

cascade.css

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

```
<!DOCTYPE HTML>
<html>
<head>
    <meta charset="UTF-8">
    <title>the cascade</title>
    <style>
        body {
            font-family: Arial;
            font-size: 120%;
        }

        p {
            color: blue;
        }
    </style>
    <link href="cascade.css" rel="stylesheet" type="text/css" />
</head>
<body>
    <p>I require styling</p>
</body>
</html>
```

I require styling

The Cascade
> Example…

```html
<!DOCTYPE HTML>
<html>
<head>
    <meta charset="UTF-8">
    <title>the cascade</title>
    <link href="cascade.css" rel="stylesheet" type="text/css" />
    <style>
        body {
            font-family: Arial;
            font-size: 120%;
        }

        p {
            color: blue;
        }
    </style>
</head>
<body>
    <p style="color: green;">I require styling</p>
</body>
</html>
```

css

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

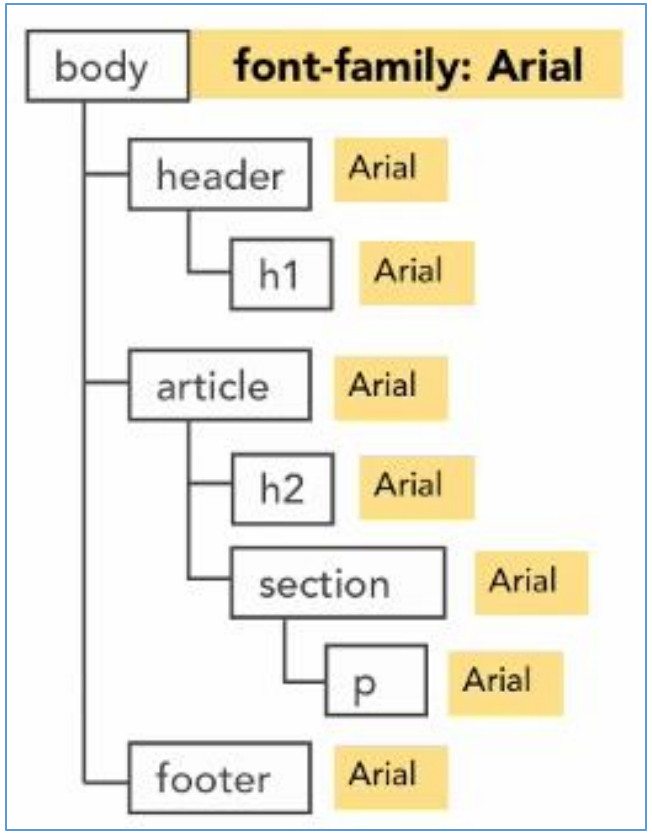
I require styling

Inheritance

- The process by which properties **are passed from parent to child elements without explicit definition.**
- Allows elements to “inherit” styles from parent elements.
- Helpful in reducing the amount of CSS to set styles for child elements.
- Unless a more specific style is set on a child element, the element looks to the parent element for its styles.
- Styles that relate to **text and appearance** are inherited by the descendant elements.
- Styles that relate to the appearance of boxes created by styling DIVs, paragraphs, and other elements, such as **borders**, **padding**, **margins** are not inherited.

Inheritance

- What if you want to apply Arial font for all of your elements in a given HTML document. You just have to apply the font to the body element as shown on the figure.
- In the event of a conflict**, child element styles will override parent element styles**



Inheritance > Example

```
<style>
    body {
        font-family: Georgia;
        font-size: 120%;
        color: green;
    }
</style>
```

```
<body>
    <section>
        I am just text within the section.
        <p>
            I am a paragraph nested inside a section. I also have a
            <strong>strong tag</strong> nested inside of me.
        </p>
    </section>
    <aside>
        <p>
            I am a paragraph nested inside an aside. I also have a
            <strong>strong tag</strong> nested inside of me.
        </p>
    </aside>
</body>
```

I am just text within the section.

I am a paragraph nested inside a section. I also have a **strong tag** nested inside of me.

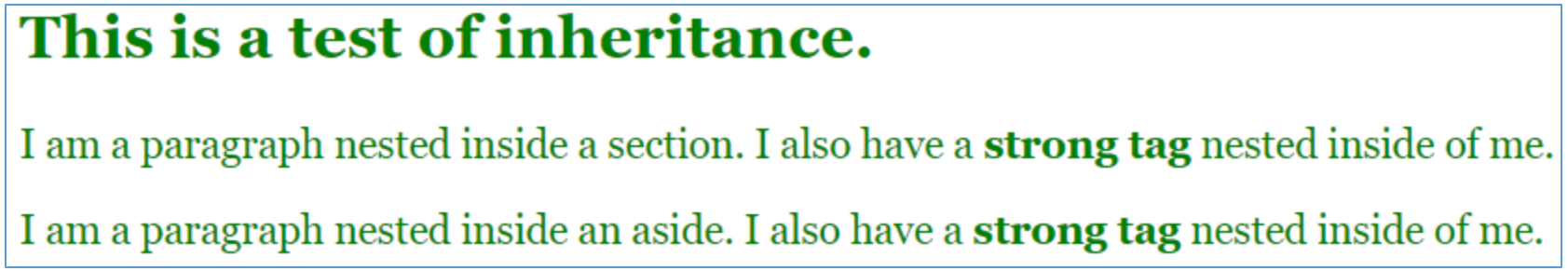
I am a paragraph nested inside an aside. I also have a **strong tag** nested inside of me.

Inheritance
> Example…

- **Not every property inherits**.
- When they have a property applied to them that conflicts with the parent element, **the child element's property wins.**

```
<body>
    <section>
        <h1>This is a test of inheritance.</h1>
        <p>
            I am a paragraph nested inside a section. I also have a
            <strong>strong tag</strong> nested inside of me.
        </p>
    </section>
    <aside>
        <p>
            I am a paragraph nested inside an aside. I also have a
            <strong>strong tag</strong> nested inside of me.
        </p>
    </aside>
</body>
```

```
<style>
    body {
        font-family: Georgia;
        font-size: 120%;
        color: green;
    }
</style>
```

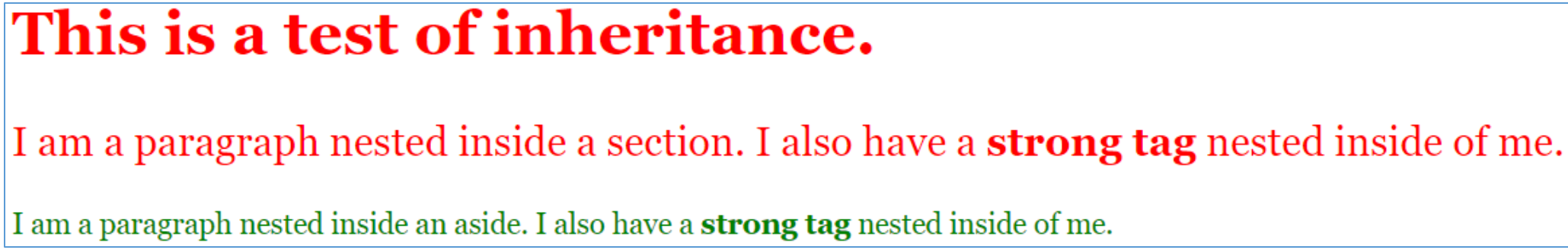


Inheritance
> Example…

- Cumulative Effect

```
<style>
    body {
        font-family: Georgia;
        font-size: 120%;
        color: green;
    }

    section{
        font-size: 1.4em;
        color: red;
    }
</style>
```

```
<body>
    <section>
        <h1>This is a test of inheritance.</h1>
        <p>
            I am a paragraph nested inside a section. I also have a
            <strong>strong tag</strong> nested inside of me.
        </p>
    </section>
    <aside>
        <p>
            I am a paragraph nested inside an aside. I also have a
            <strong>strong tag</strong> nested inside of me.
        </p>
    </aside>
</body>
```



Inheritance
> Example…

- *Property of the paragraph are going to override any properties of a parent element that conflict with it*

```
<body>
    <section>
        <h1>This is a test of inheritance.</h1>
        <p>
            I am a paragraph nested inside a section. I also have a
            <strong>strong tag</strong> nested inside of me.
        </p>
    </section>
    <aside>
        <p>
            I am a paragraph nested inside an aside. I also have a
            <strong>strong tag</strong> nested inside of me.
        </p>
    </aside>
</body>
```

```
<style>
    body {
        font-family: Georgia;
        font-size: 120%;
        color: green;
    }

    section {
        font-size: 1.4em;
        color: red;
    }

    p {
        color: blue;
    }
</style>
```

This is a test of inheritance.

I am a paragraph nested inside a section. I also have a **strong tag** nested inside of me.

I am a paragraph nested inside an aside. I also have a **strong tag** nested inside of me.

Working with Inheritance

- As sties become more complex, inheritance become more difficult to track.
- **Keeping styles organized will help you avoid conflicts caused by inheritance.**
- **Placing your most basic, default styles on top-level elements is an efficient way to style elements site-wide.**
- Remember that styles are cumulative; you must account for formatting inherited through parent elements.

Specificity

- Most of the time the cascade or inheritance can be counted on to resolve styling conflicts. However, there is a third concept. Specificity that can come into play as well.
- The specificity of a selector is, **how specific is it?**
- In the event of a conflict between two selectors that inheritance can not solve the specificity of the selector is used to determine which selector has precedence.

Quiz:-Show at least 9 ways to target the <h1> element

```
<div id="div">
<h1 id="one" class="one">Css Example</h1>
```

Example:-
H1{
Color:red
}

Specificity

- classes: Classes, Attributes, and Pseudo-classes
- elements: Element Types and Pseudo-elements

selector	ID	classes	elements	specificity
body	0	0	1	1
#mainContent	1	0	0	100
.quote	0	1	0	10
div p	0	0	2	2
#sidebar p	1	0	1	101

Specificity
> Example

- We have 101 and 1, the 101 always wins

```
<body>
    <section id="mainContent">
        <p>I am a paragraph nested inside a section.
        I also have a <strong>strong tag</strong> nested inside of me.</p>
    </section>
</body>
```

I am a paragraph nested inside a section. I also have a **strong tag** nested inside of me.

```
<style>
    body {
        font-family: Georgia;
        font-size: 120%;
    }

    #mainContent p {
        color: red;
    }

    p {
        color: blue;
    }
</style>
```

Specificity
> Example

- Programmers go crazy about this one.
- This is not working because there is a selector elsewhere with a **higher degree of specificity than the one you are trying to use.**
  - .green has a specificity of 10 and #mainContent p has a specificity of 101.

```
<body>
    <section id="mainContent">
        <p class="green">I am a paragraph nested inside a section.
        I also have a <strong>strong tag</strong> nested inside of me.</p>
    </section>
</body>
```

I am a paragraph nested inside a section. I also have a **strong tag** nested inside of me.

```
<style>
    body {
        font-family: Georgia;
        font-size: 120%;
    }

    #mainContent p {
        color: red;
    }

    p {
        color: blue;
    }

    .green{
        color: green;
    }
</style>
```

Specificity
> Example

- In this case, we know .green has a specificity of 10, which shouldn't override #mainContent p that has a specificity of 101.
  - Specificity works just fine until inheritance is involved.
- The strong element is nested inside the section and a paragraph. We know from inheritance that those properties like color wil
  But, if a child element has a style that differs or conflicts v
  parent styles, the child styles always wins. In this case we

```
<body>
    <section id="mainContent">
        <p>I am a paragraph nested inside a section.
        I also have a <strong class="green">strong tag</strong> nested inside of me.</p>
    </section>
</body>
```

I am a paragraph nested inside a section. I also have a **strong tag** nested inside of me.

```
<style>
    body {
        font-family: Georgia;
        font-size: 120%;
    }

    #mainContent p {
        color: red;
    }

    p {
        color: blue;
    }

    .green{
```

Specificity
> Example

- .green has a specificity of 10
- strong has a specificity of 1
- .green wins.

```
<body>
    <section id="mainContent">
        <p>I am a paragraph nested inside a section.
        I also have a <strong class="green">strong tag</strong> nested inside of me.</p>
    </section>
</body>
```

I am a paragraph nested inside a section. I also have a **strong tag** nested inside of me.

```
<style>
    body {
        font-family: Georgia;
        font-size: 120%;
    }

    #mainContent p {
        color: red;
    }

    p {
        color: blue;
    }

    .green {
        color: green;
    }

    strong {
        color: purple;
    }
</style>
```

Specificity
> Example

- .green has a specificity of 10
- strong.green has a specificity of 11
- strong.green wins.

```
<body>
    <section id="mainContent">
        <p>I am a paragraph nested inside a section.
        I also have a <strong class="green">strong tag</strong> nested inside of me.</p>
    </section>
</body>
```

I am a paragraph nested inside a section. I also have a **strong tag** nested inside of me.

```
<style>
    body {
        font-family: Georgia;
        font-size: 120%;
    }

    #mainContent p {
        color: red;
    }

    p {
        color: blue;
    }

    .green {
        color: green;
    }

    strong.green {
        color: purple;
    }
</style>
```

Specificity

- Remember
  - You always have **to keep track how specific a rule is** to know if you need something a little bit more specific to override something.
  - You want to try to keep the specificity of your **rules as low as possible. They are easier to parse that way**, and they are also a **lot easier to override later on**.
  - **And semantic naming matters. (a class named green, but it sets its color to purple)**
- Please go ahead and visit the following link and play with specificity
  - https://www.sitepoint.com/web-foundations/specificity/

Important

- In addition to the cascade, inheritance and specificity there is one more CSS technique that can be used to resolve conflicts and that is the **important** declaration.
- It doesn’t matter what else is going on in your styles, in the event of a conflict, the important declaration wins.
  - The only thing that can override an important declaration is **another more specific important declaration** or some kind of user controlled stylesheet.
- Don’t use important declaration unless it is absolutely necessary.
- Example: color: blue **!important**;

Important
> Example

- In the following case regardless of the cascade, *specificity or inline style the important declaration wins.*

```
<body>
    <section id="mainContent">
        <p style="color: purple">
            I am a paragraph nested inside a section.
            I also have a <strong>strong tag</strong> nested inside of me.
        </p>
    </section>
</body>
```

I am a paragraph nested inside a section. I also have a **strong tag** nested inside of me.

```
<style>
    body {
        font-family: Georgia;
        font-size: 120%;
    }

    p {
        color: red !important;
    }

    #mainContent p {
        color: green;
    }
</style>
```

Resolving Conflicts Through Planning > General Guidelines

- Avoid using local styles when possible.
  - They add unwarranted extra layer of **complexity** to a site.
  - Updating or modifying styles can become a chore.
  - Local styles are easy to miss, especially in a team environment.
- Do not use inline styles except for very specific circumstances.
  - They are almost impossible to detect and maintain.

Resolving Conflicts Through Planning > Develop a Strategy for Specificity

- Don’t mix class and ID selectors without having a **plan that guides when they are to be used.**
- Sections of ID selectors can be hard to override later if not accounted for.
- **Rule of Thumb:** If you find yourself writing descendent selectors with three or more selectors, consider revising your strategy.

Resolving Conflicts Through Planning > Use Inheritance to your Advantage

- If you are familiar with your page structure, you should be able to identify global formatting across your site.
- Those formatting needs can then be written as global styles on parent elements, and inherited by the rest of the site.
- **This results in fewer rules to write and easier styles to maintain.**

Resolving Conflicts Through Planning
> Think about How Styles Relate to One Another

- New designers often make the mistake of **styling each element individually, as it's encountered.**
- This can lead to bloated style sheets and hard to maintain styles.
- Thinking of styles **as related formatting allows you to plan and write organized style sheets.**

Example of Good Planning

```
body{padding:0;margin:0 2px;background-color:#000080;font-family:helvetica,arial,sans-serif;color:#222;line-height:1.4;
table{border:1px solid #999;margin:1em;}
table h2,h3{padding-top:0em;}
th{border-style:solid;border-width:1px;padding:1em;background-color:#f6f6f6;}
td{background-color:#fff;font-family:helvetica,arial,sans-serif;vertical-align:top;border:1px solid #999;padding:1em;}
td.grey{background-color:#c0c0c0;}
p{padding:0;}
p.center{text-align:center;}
p.adtop{text-align:center;}
h2.grey{text-align:center;background-color:#f0f0f0;font-weight:bold;padding:.5em;border:solid 1px #ccc;}
h1.center{text-align:center;}
h2,h3{padding-top:1em;}
h1,h2,h3,h4{color:#000080;}
a:link{color:#2222ff;text-decoration:none;}
a:link:hover{color:#2222ff;text-decoration:underline;}
a:visited{color:#800080;}
a:visited:hover{color:#800080;text-decoration:none;}
a.img:visited:hover{color:#ff0;background-color:#fff;}
a.img:link:hover{color:#ff0;background-color:#fff;}
a.button{color:#fff;text-decoration:none;font-weight:bold; background-color:#000080;padding:0.5em;}
a.button:hover{color:#000080;text-decoration:none; font-weight:bold;background-color:#fff;border:2px solid #000080;padd
a.butred{color:#fff;text-decoration:none;font-weight:bold; background-color:#ff0000;padding:0.5em;border:1px solid #000
a.butred:hover{color:#ff0000;text-decoration:none; font-weight:bold;background-color:#fff;border:1px solid #ff0000;padd
div.header{padding:0em 1em 0em 1em; float:left;background-color:#000080;}
div.header a:visited{text-decoration:none;}
div.header a:visited:hover{text-decoration:none;}
div.alphabet{line-height:2;font-size:150%;color:#ccc;}
```

Example of Good Planning. *Separate .css file for different font family,animations…*

```
/* vietnamese */
@font-face {
  font-family: 'Roboto';
  font-style: normal;
  font-weight: 700;
  src: local('Roboto Bold'), local('Roboto-Bold'), url(https://fonts.gstatic.com/s/roboto/v18/PwZc-YbIL414wB9rB1IAPRJtr
  unicode-range: U+0102-0103, U+0110-0111, U+1EA0-1EF9, U+20AB;
}
/* latin-ext */
@font-face {
  font-family: 'Roboto';
  font-style: normal;
  font-weight: 700;
  src: local('Roboto Bold'), local('Roboto-Bold'), url(https://fonts.gstatic.com/s/roboto/v18/97uahxiqZRoncBaCEI3aWxJtr
  unicode-range: U+0100-024F, U+0259, U+1E00-1EFF, U+20A0-20AB, U+20AD-20CF, U+2C60-2C7F, U+A720-A7FF;
}
/* latin */
@font-face {
  font-family: 'Roboto';
  font-style: normal;
  font-weight: 700;
  src: local('Roboto Bold'), local('Roboto-Bold'), url(https://fonts.gstatic.com/s/roboto/v18/d-6IYplOFocCacKzxwXSOFtXF
  unicode-range: U+0000-00FF, U+0131, U+0152-0153, U+02BB-02BC, U+02C6, U+02DA, U+02DC, U+2000-206F, U+2074, U+20AC, U+
}
```