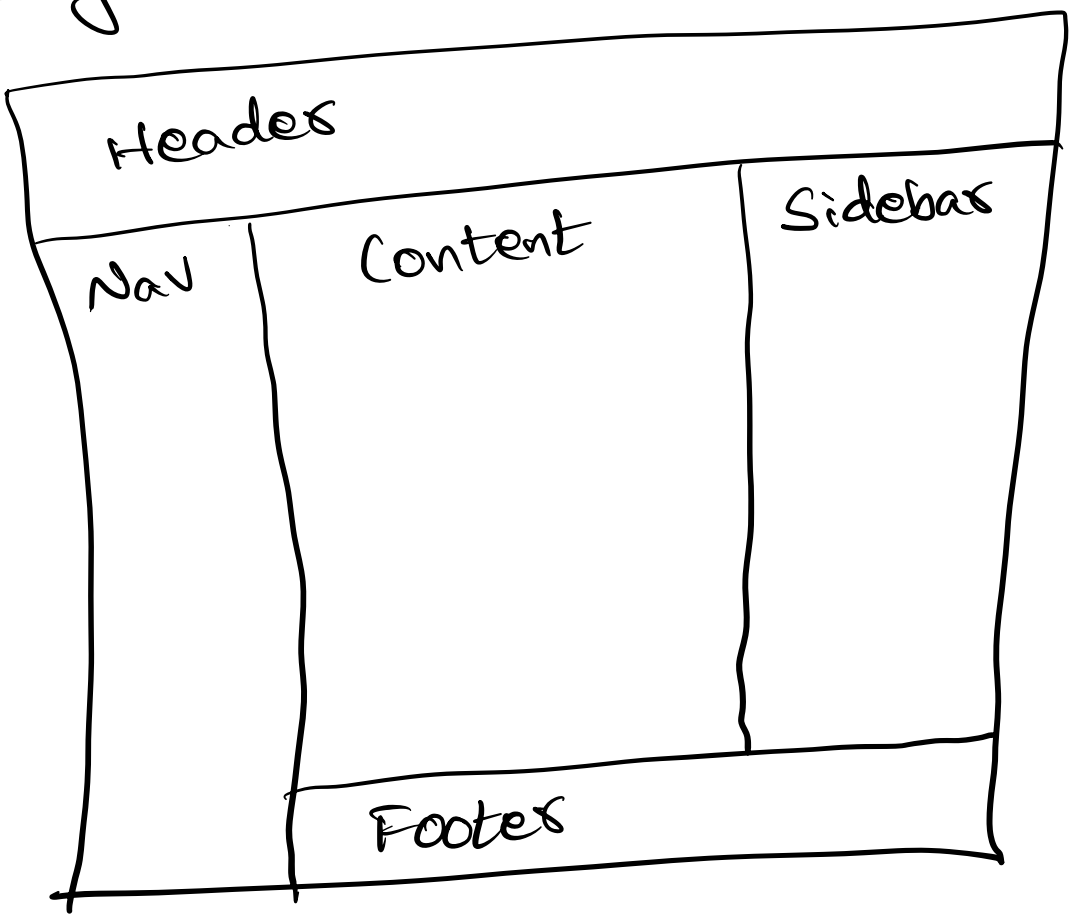
7. ADVANCED CSS (SPECIFICITY, INHERITANCE, POSITIONING)

<aside> - To have a side bar

Give names to grid item area.

Let's say we want to build this layout.



```
header {
    grid-area: top;
}
```

To my header component, add grid area as top. It is just giving a label.

In the layout we wanted to build, we have 3 rows & 3 columns,

top	top	top	
nav	content	sidebar	→ 9 Grid items
nav	bottom	bottom	

grid-template-areas: 'top top top'
'nav content sidebar'
'nav bottom bottom';

It is easier, better & efficient.
No need for row-span, column-span.
It is highly responsive.

grid-area name given to the grid item.

grid-template-rows: 80px 1fr 80px;

1st & 3rd row will have fixed size of 80 pixels. 2nd row will take the remaining space.

CSS Grid Garden → website

Flexbox vs Grid:

Both are used for creating responsive layouts.

Flexbox → 1D

Grid → 2D

Flexbox is ideal for content first approach.

↓

size of the flex container's content or no. of items is dynamic or unknown.

Items can be aligned horizontally or vertically and distribute space between

them.

Grid is ideal for layout first approach.

↓

when you have clear idea of layout, regardless of the content.

Offers control over row & column sizing & ability to align the entire grid.

Suitable for larger & complex layouts.

You can always use flexbox inside a grid & vice versa.

Users Agent Style Sheet:

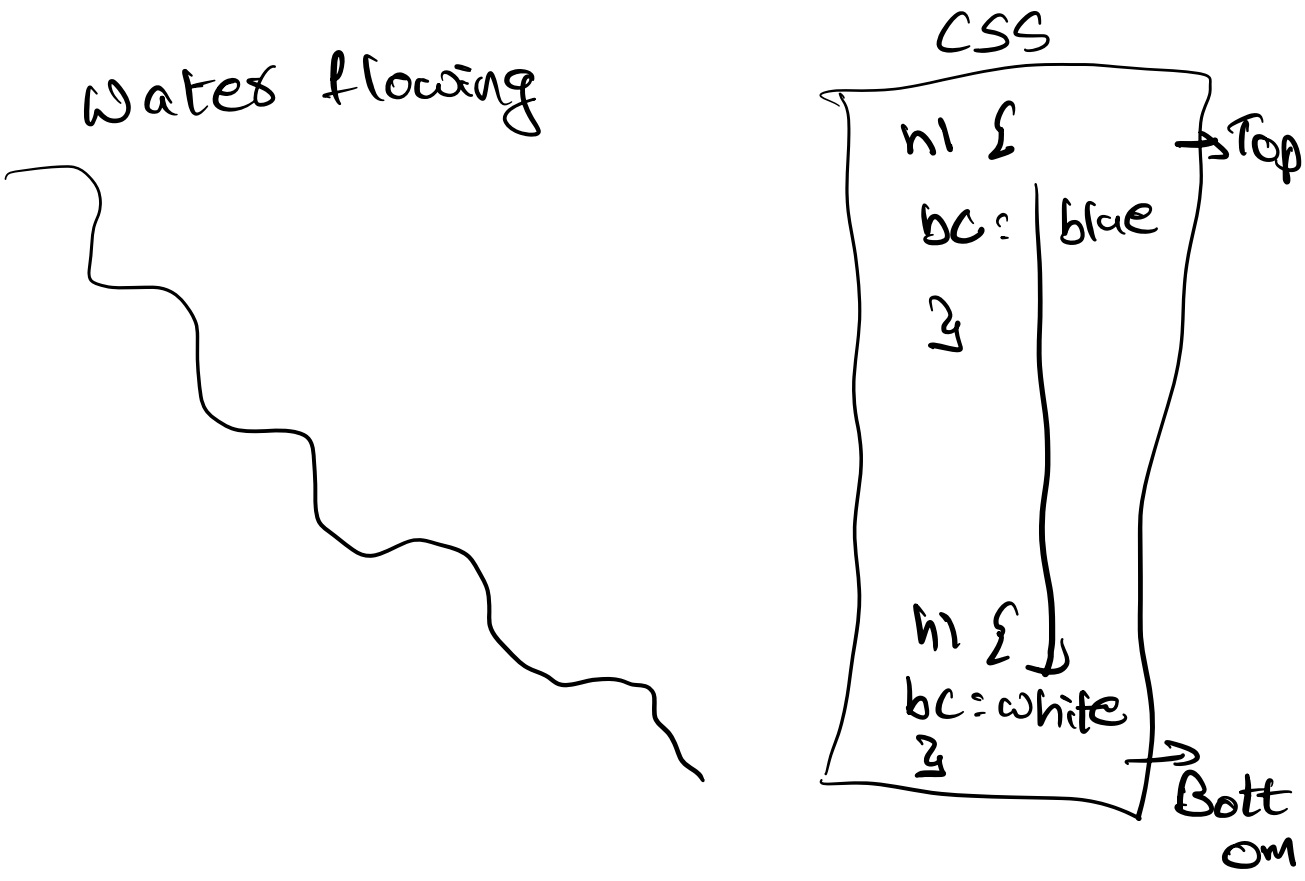
The style sheet is a set of default styles that browser applies to HTML document when author didn't give any.

Every browser has it's own UASS.
They can be overrided.

Cascading:

Like a waterfall flowing down a series of steps. Just as water flows from top and cascades down, styles in CSS are applied from top to bottom, with later styles flowing down and

potentially overriding the previous ones.



CSS follows top down approach

Inheritance:

```
<body>
    <h1> Welcome to CSS </h1>
    <p> lorem * 5 </p>
```

```
</body>

<style>
    body {
        font-family: Arial, ....
        color: pink
    }
</style>
```

The font & color of elements h1 & p have changed, even though style was not applied to them.

They inhesited from body.

However,

```
body {
    border: 3px solid red;
```

The borders is applied only to the body, not to each individual element inside it.

Borders property is not inhesited.

Inhesitance is nothing but an element getting property values from a parent.

Some properties can be inhesited & some can't.

Specificity:

When these are conflicting styles, the one with more details | specifics takes precedence.

```
<li id="fruits" class="favorite">
    Mango </li>
```

```
#fruits {
    color: red
}

.favorite {
    color: green
}
```

The specificity score is typically represented as a four-part value : [a, b, c, d]

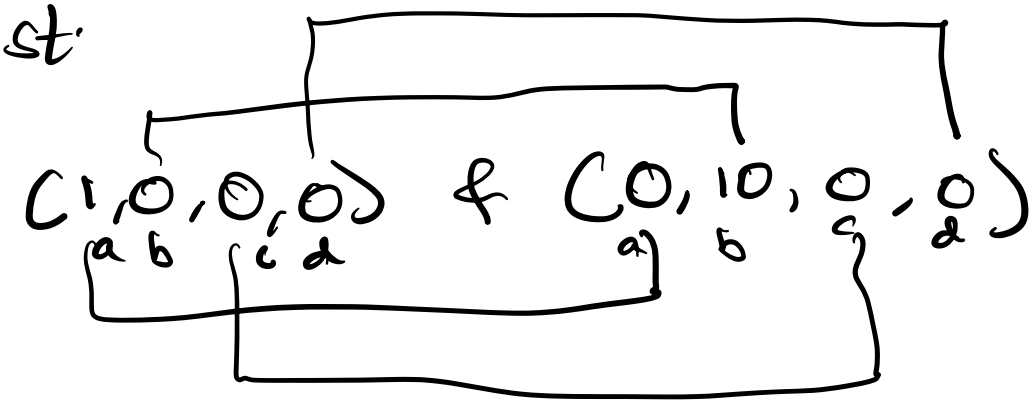
a -> Inline styling

b -> Id selectors

c -> classes, Attribute selectors, Pseudo class selectors

d -> Element selectors, Pseudo elements

When comparing specifity, the highest value in each segment is considered first.



It is not about highest value.

0 0 0 100

0 1 0 0 ✓

First compose 'a', if nothing go to 'b', then 'c', then 'd'.

ul#fruits

↓

ul that has id fruits

li.favorites

li that has class favorites

!important → Trump card

↓

Highest specificity

If two elements have same specificity

scope, the style applied last is the winner.

Position:

To control position of the element in it's containing element.

position: static;

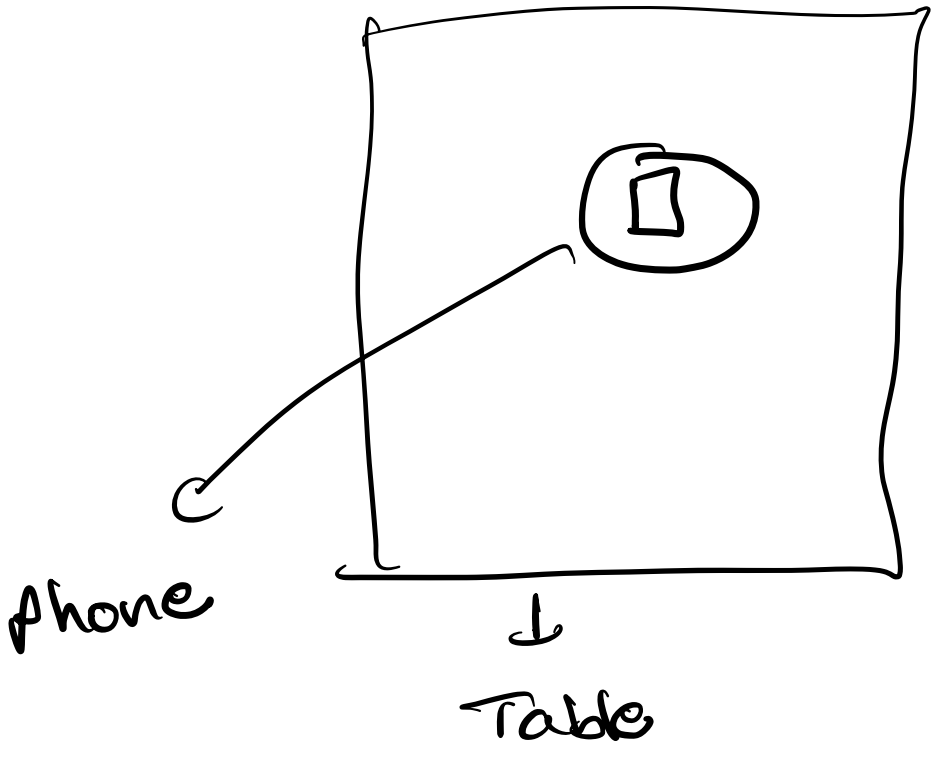
↓

Default value for all elements.

Elements are positioned in normal flow.

When position is static top, right, bottom, left have no effect.

position: relative;



Pick up the phone & shift it by 10 pixels.

Relative positioning is like giving an element a little wiggle room to move, without changing its place in line.

Let's say you are standing in a line,

→ Normal spot is where you are standing.

→ With relative positioning, you can take a small step in any direction (up, down, left, right)

→ Even if you move, everyone else stays where they are. Your original spot is saved.

→ You are still part of the line, just slightly out of place.

In web terms,

→ Element stays in its normal spot

in the webpage layout.

→ You can nudge it a bit from that spot without pushing the other elements around.

→ It's like the element is on a invisible leash from its original position.

Absolute:

Absolute positioning is like lifting an element out of the line and placing it wherever you want on the page.

Imagine arranging photos on bulletin

board,

→ You take a photo (the element) off board.

→ You can put it anywhere on the board.

→ Other photos (elements) on the board will ignore this photo & arrange themselves as if it wasn't there.

→ The photo's position is based on its nearest positioned ancestor (like the edge of the bulletin board), or the whole page if there isn't one.

In code terms,

→ Element is removed from normal flow.

→ You can place it anywhere on the page using top, right, bottom left.

→ Other elements behave as if it doesn't exist, potentially overlapping with it.

→ It's like the element is floating above the page, anchored to its nearest positioned page or the page itself.

Nearest positioned ancestor:

Browser looks for the nearest ancestor (parent, grandparent etc.) that has a position value other than static.

If no ancestor has a position set, it defaults to positioning relative to the entire document.

```
<div class="bookshelf">
    <div class="book">Book 1</div>
    <div class="book book2">Book 2
                            </div>

    <div class="book book3">Book 3
                            </div>
</div>
```

```
.book3 {
    position: absolute;
}
```

bookshelf is the first ancestor to book3.

If bookshelf has position,

```
.bookshelf {
    position: relative;
}
```

anything other than static, the browser will position the element relative to the ancestor.

If it doesn't find any ancestor, it will position the element relative to the entire document.

Fixed Positioning :

Sticks an element to a specific spot in the browser window, keeping it visible even when you scroll the page.

- The element is removed from the normal document flow.
- It's always positioned relative to the entire browser window.
- It stays in the same place on the screen, no matter how much

you scroll.

→ It's often used for elements that should always be visible, like navigation menus or "back to top" buttons.

Stick Positioning:

Allows an element to switch between relative & fixed positioning based on how far you've scrolled the page.

→ Imagine you are in a line, but you can stretch like rubber.

→ You stay in your spot until a certain point, then you stick

to the nearest wall (like the fixed element) until your group moves past you.

→ Then you snap back into your place in line.