* To center bunch of elements/element we can use **margin: auto or 0 auto.**
* To position bunch of inline elements to single line we can use **display: inline-block**
* To position elements vertically we can use **vertical-align: top/center** etc.
* **Float** means we overwrite the default positioning and tell browser to push an element to the left or right. It takes the element out of the document flow, which can cause issues. That’s why it’s not used often anymore.
  + It is mostly used for images.
  + If we want to use it for elements, we have to use **clearfix**. We put an empty div after that float element, and in CSS we do **clear:both**. That will fix the issue.

**Z-Index:**

* this property controls the elements position in z-axis. It means, how overlapping elements will behave, who will be on top and who will be in background.
* Default value is auto (zero). If we want to place an element top of another, we increase the value. If we want in background, we decrease the value.
* If two elements have same z-index, then the element that **comes later** in the html code, it will be placed in front.
* This property doesn’t have any effect on static position elements. It only works where position property is defined something else (like fixed, absolute and so on).

**Stacking Context:**

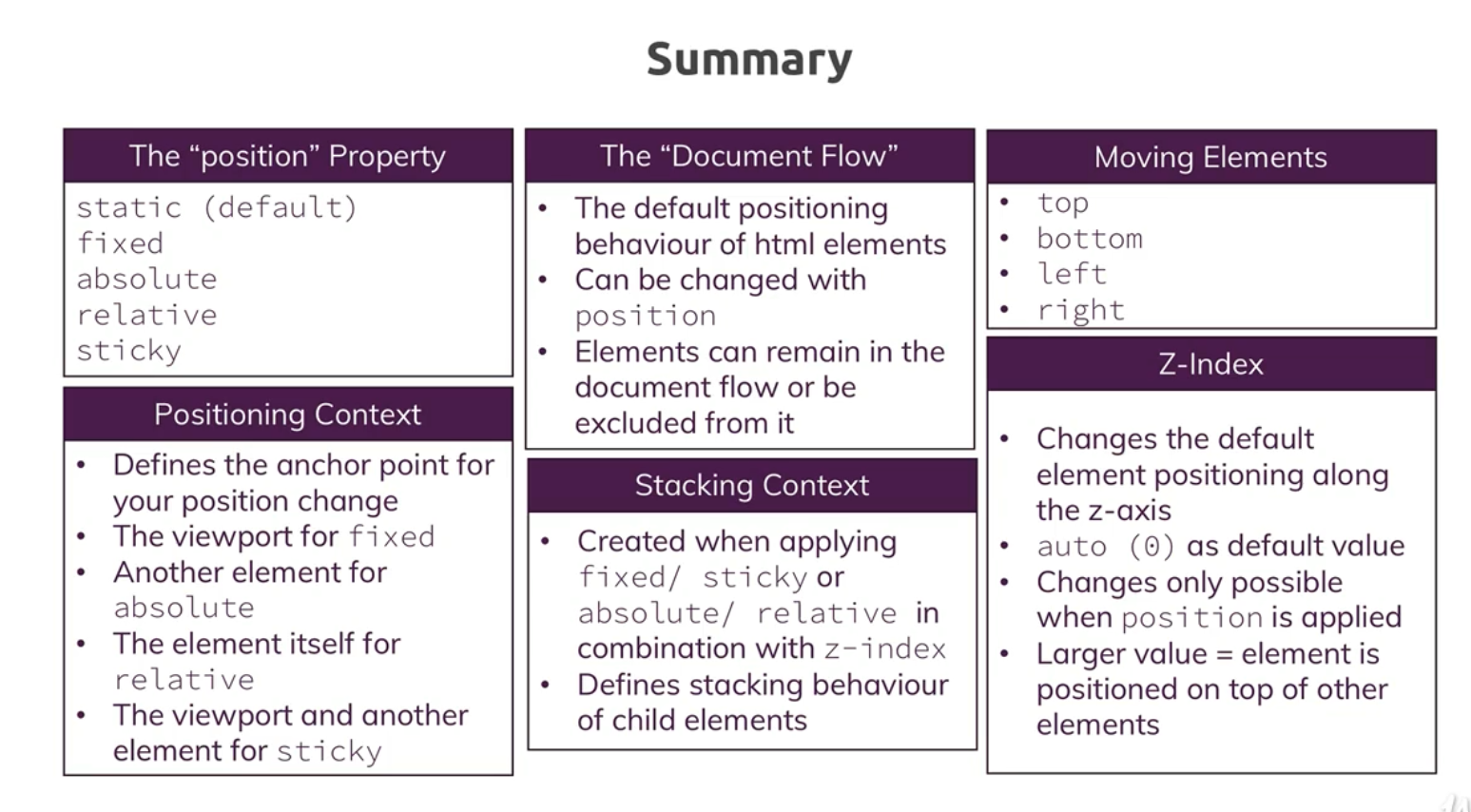
* We can stack bunch of positioned elements on top of each other, and we can define it by z-index, but what happens if those elements had child with z-index?
* In this case, the child z-index only works inside the parent element. So, for example, if a child had z-index of 100, and a sibling of its parent had a z-index of 50. In this case, still the sibling will be on top even though it has lower z-index.
* This behavior is due to stacking context. Read the full details in MDN

**Position**:

This property lets us change the position of some elements. We use it in conjunction with the **top, left, bottom, right** properties. They simply define how far from the respective position the element will be.

Note: When positioning an element relative to parent, positioning it from right works better than left.

1. **Static**:
   1. it’s the default value. which follow the document flow (top towards bottom)
   2. To be able to change the position of element, we need to change the position property value from static to other.
2. **Fixed**:
   1. It will make the element behave like inline block element (but it’s not technically inline-block). But we can use this property with same effect on both inline and block element.
   2. The element position depends only on the **viewport** (the user visible area of the device).
3. **Absolute:** 
   1. Its positioning context will be based on two cases.
      1. If none of the ancestor has a position property defined, then its context will be relative to **html element**.
      2. If one of the ancestors has a position property defined, then its context will be the **closest ancestor** element which as position defined.
   2. This takes the element completely out of document flow. The rest of the elements will act as if the element is not there. This is crucial difference
4. **Relative:** 
   1. It doesn’t take the element out of document flow
   2. Its positioning context is the **element itself**.
   3. So, we can basically push our element from its current position.
   4. We have to keep in mind that depending on how we set our top/left etc value, the element can leave its current parent.
   5. In this case, we can use **overflow: hidden.** This would hide the element once it leaves the parent, a good way to ensure our element don’t leave the container.
      1. Note: if we assign overflow: hidden to the body element, it will actually pass it to html element. This is an exception which might cause issues.
5. **Sticky:** 
   1. It is a combination of relative and fixed. **Bad browser support**.
   2. We can specify a distant between our element and viewport. As soon as this distance is reached, the element behaves as fixed. So, its context is the combination of viewport and the element.
   3. It stops again being fixed as soon as it reaches the end of its parent.

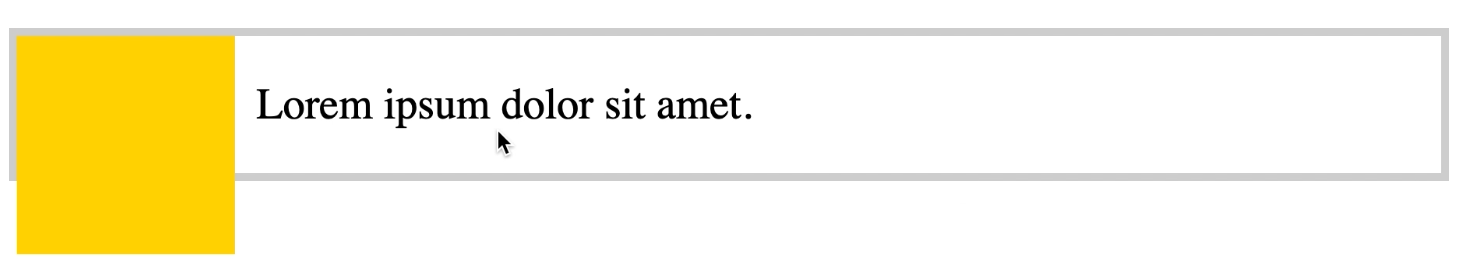
****

**Overflow:** When the container can’t fit the entire content, that is when overflow happens. This is shortcut for Overflow-x and overflow-y

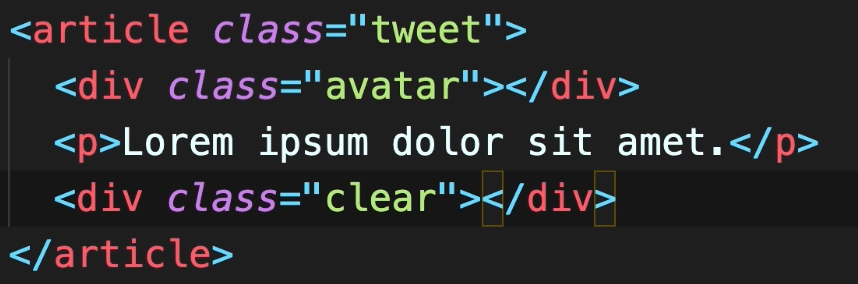
* **Visible:** default value, we see the extra content
* **Hidden:** We don’t see the extra content
* **Scroll:** we see horizontal and vertical scroll bars.
* **Auto:** Scrollbars only appear if overflow happens.

**Floating Element:** We can specify float property and all the subsequence elements will flow around it.

* **Clear:** We can specify float (values – left, right, both) to both. This helps with a bug that happens when we use float property. This causes the parent to ignore float element.

****

We can solve this, by adding empty div with clear:both right after the content.



But there is issue with this approach. It is a workaround and not semantic, there is a better approach. We get the empty div dynamically but without polluting our html markup. This is the preferred approach.

