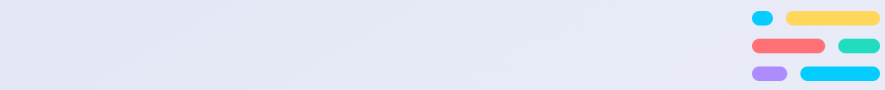


The Ultimate Guide to Logical Properties

Learn everything you need to know about Logical Properties and Values

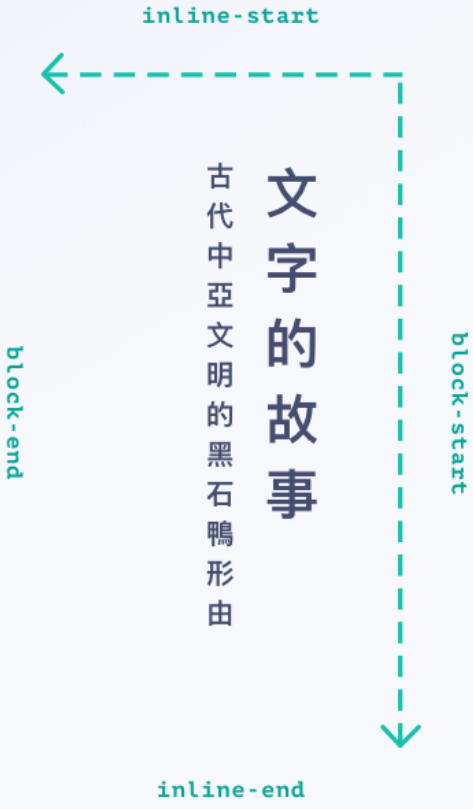


Table of Contents

Why learn Logical Properties?

The web development world changes rapidly. Since you're reading this I can tell you're the type of person who cares about staying on top of those changes.

There has begun to be a shift in the direction that CSS is going. It was slow at first with Flexbox and is now quickly picking up speed with CSS Grid and now Logical Properties.

So, what's the shift?

It's an awareness and realization that not everyone is going to be creating or reading your websites in English (or other left to right languages). And the way that we create websites is starting to take that into consideration.

It is the future of CSS. This is something you'll eventually have to learn if you want to stay on top of your game. This guide will help you learn it quickly and thoroughly.

About the Author

Lauralee Flores is the founder of CSS Academy. She has been designing and creating those designs on the web for the past 10+ years.

She knows that design matters and fell in love with CSS in 2008 when she created her first website.

It is her goal to not only help you discover how fun writing CSS can be. But, she will be doing her very best to help you fall in love with CSS, the design language on the web.

When you'll use Logical Properties.

For years the CSS community has been slowly moving in this direction. As time goes on you will see Logical Properties weaved throughout your CSS.

It's no surprise that for years the CSS Working Group has been aware of the need for Logical Properties. It only makes sense.

Think about it, where is your website traffic coming from? If you look in your analytics tool you'll likely see it's spread across the globe.

No matter what language your website is written in, it is highly probable that not everyone who visits your site is reading it in your language.

If you are someone who:

- wants to stay up to date with your CSS
- builds website in a language that is read right right to left and/or top to bottom
- builds website themes
- is a freelance web developer

then learning Logical Properties is extremely important.

In each of these scenarios, learning Logical Properties will

prepare you to build a website where that content can be translated into any language and continue to layout correctly.

It all started in 2008

Back in 2008 the CSS Working Group started discussing an idea that later became Flexbox. In mid-2009 Flexbox's working draft was published with elements that we now see as Logical Properties.

No one realized it at the time but this was not only the birth of Flexbox but also of Logical Properties.

Almost 10 years later in 2018 when CSS Grid and Logical Properties were released, the two were tied together.

Again, no one really saw this. Looking back though it becomes more clear.

Logical Properties impacts layout primarily and it is no surprise that the two biggest advancements in CSS Layout came with Logical Properties built into them.

Flexbox and CSS Grid

The real basis for understanding this shift toward Logical Properties can be seen in the very fundamental way that Flexbox and CSS Grid view the physical dimensions of alignment like left, right, top, and bottom.

Neither Flexbox nor CSS Grid take into account physical dimensions.

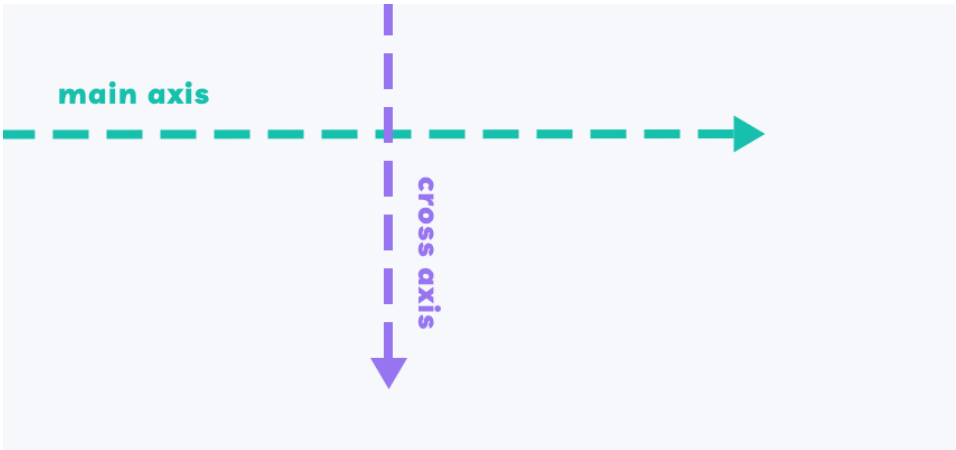
This is a huge shift.

Both Flexbox and CSS Grid use logical properties (start, end) instead of physical ones (top, bottom, right, left).

Let's look at some specifics.

Flexbox

Flex items flow horizontally or vertically depending on the direction of the main and cross axes.



If you've struggled learning Flexbox, I have good news. As you learn and understanding Logical Properties it will help you better understand Flexbox and vice-versa.

In Flexbox if you want to align something along the main axis you would use **justify-content**. With a value of **flex-start** or **flex-end** (there are a few others too but to understand this point I'm going to skip those for now).

And if you wanted to align something along the cross-axis you would use **align-items**. With a value, again of **flex-start** or **flex-end**.

Do you see how there is no physical dimensions mentioned? There is no top, bottom, left, or right.

A few other Flexbox properties that you've likely used many times and that align elements are **align-content** and **align-self**. Both use this flex-start or flex-end approach.

CSS Grid

CSS Grid follows the same logic. You won't see top, right, left or bottom anywhere.

Where Flexbox is one dimensional (the main axis) CSS Grid is two dimensional (rows and columns).

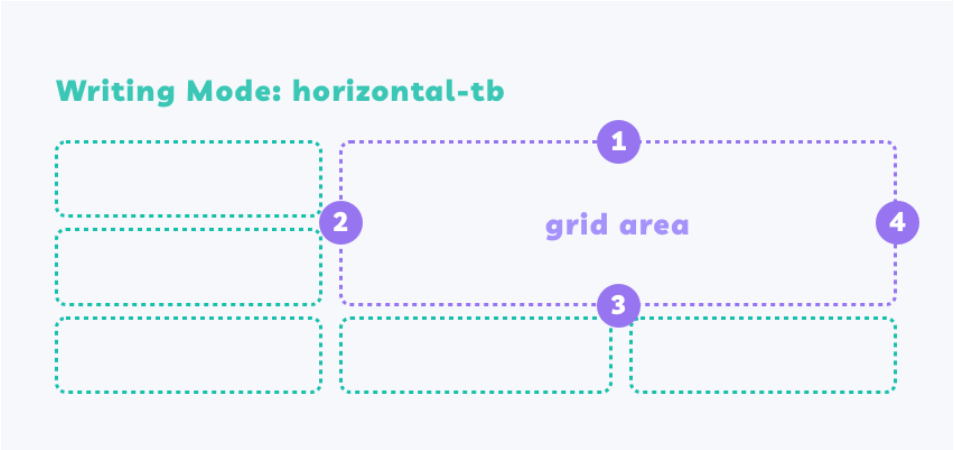
So, in CSS Grid you'll see **start** and **end** (not top, bottom, left, or right) values for rows and columns. Like this:

```
.grid-area {
    grid-row-start: 1;
    grid-column-start: 2;
    grid-row-end: 3;
    grid-column-end: 4;
}
```

These values are **relative to the grid area**.

If the writing mode on a website is in a regular horizontal mode (writing-mode: horizontal-tb;) these would map to the following positions on the box:

```
.grid-area {
    grid-row-start: 1;   /* top */
    grid-column-start: 2;   /* left */
    grid-row-end: 3;   /* bottom */
    grid-column-end: 4;   /* right */
}
```

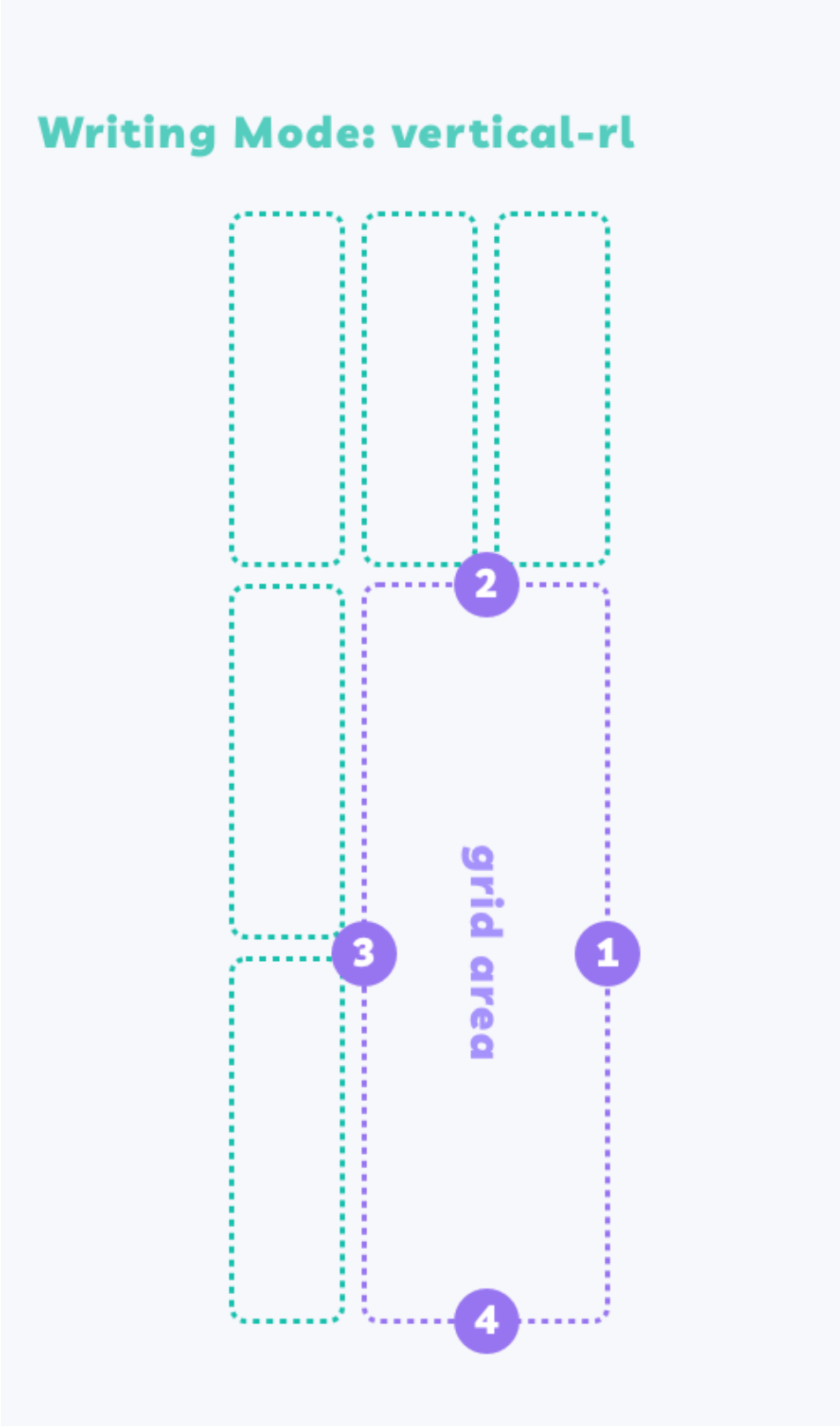


Because these are relative directions (not mapping specifically to the top, bottom, left or right of the box) if we changed the writing mode to a vertical direction, **the position of this grid will remain relative to the writing mode**.

Let's say we changed the writing mode to vertical right to left (writing-mode: vertical-rl;). These would now map to the following positions on the box:

```
.grid-area {
    grid-row-start: 1;   /* right */
    grid-column-start: 2;   /* top */
    grid-row-end: 3;   /* left */
    grid-column-end: 4;   /* bottom */
}
```

And that would look like this:



As you can see the start and end positions shift **relative** to

the writing direction not a specific physical location like top or bottom.

CSS Grid also understands **justify-items** and **justify-self** with their values as **start** or **end**. Again, not a physical location but a logical one.

As you can see, Logical Properties is the direction CSS is going. You will soon be using it not only to align items but it will be spread throughout all of your CSS.

Just as we shifted away from hard sizes like pixels to relative units, we are transitioning from physical properties like top, bottom, right, and left to logical properties.

You will be glad you learned this now as you will be using it for years to come.

What you need to know before learning Logical Properties

One of the underlying concepts of Logical Properties is something known as an Axis. It's not complicated but it's crucial you understand this concept. Once you grasp Block and Inline Axes you'll learn Logical Properties quickly.

As you work with new and emerging CSS you will begin to see block and inline dimensions show up more and more. This concept is key to understanding new CSS layouts (like CSS Grid). It is also fundamental in mastering Logical Properties.

The good news is, the concept of block and inline dimensions are easy to understand.

Let's look at a simple page that has a headline and three paragraphs.

```
<h1>Block and Inline Axes</h1>

<p>Each paragraph is a block level element. A block level element spans the whole width of the container.</p>

<p>A paragraph is laid out one below the other. The natural direction that the block level elements lay out is the block axis. </p>

<p>Inside the block level element, the paragraph text reads from left to right (in English). This is the inline axis.</p>
```

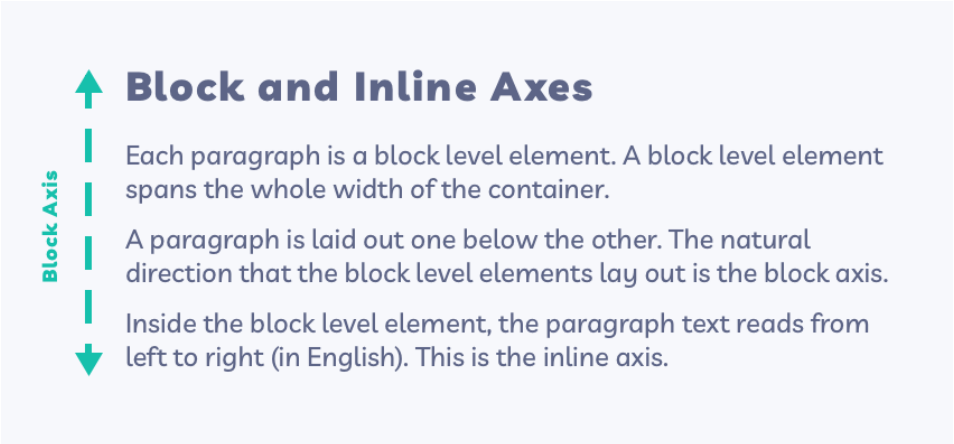
Block Axis

The header and three paragraphs are each **block level elements**.

What this means is that they always ***start on a new line*** and ***take up the full width available*** (they stretch out to the left and right as far as they can).

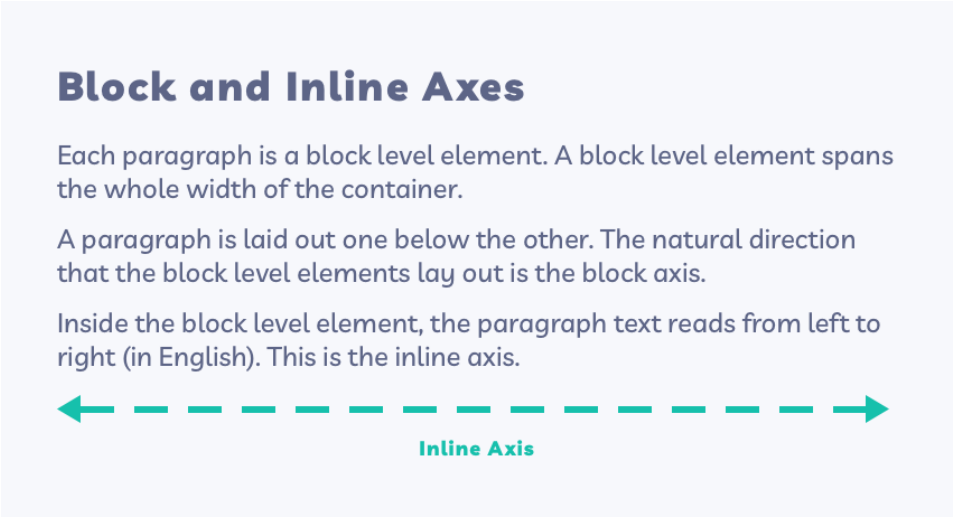
Each block level element like a header and a paragraph flow one right after the other, each taking up the full width.

This flow down the page is the block axis as it's the natural flow of the block level elements on the page.

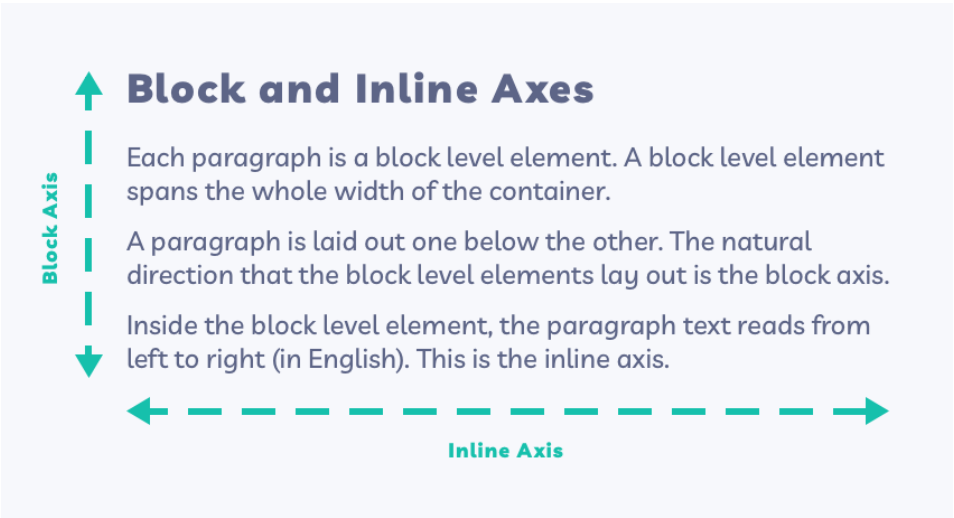


Inline Axis

The **inline axis runs across the block axis**. It is concerned primarily with the ***direction*** of the text as you are reading them.



Just as the block axis is concerned with which line comes next, inline axis is concerned with which word comes next.



Block Axis Customized

By default the block axis flows vertically down the page. Each block level element is horizontal (the element stretches horizontally across the container from the left edge to the right edge of the container) and top to bottom.

In CSS this looks like this:

```
.container {
    writing-mode: horizontal-tb;
}
```

This is the default writing-mode for browsers.

If, however, the language was a vertical based language. You could change it to vertically align and flow from the left to right:

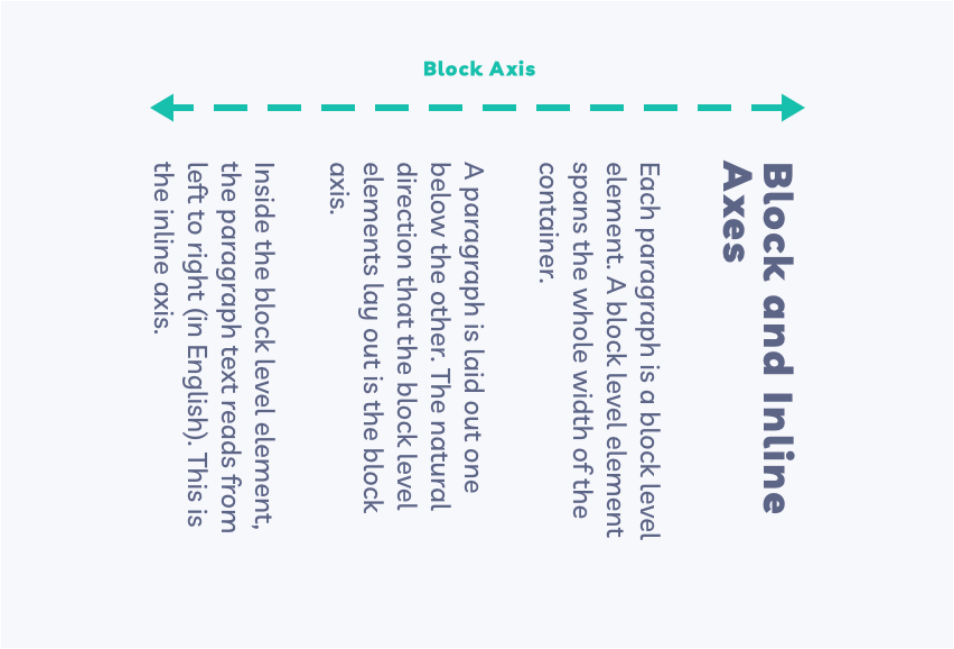
```
.container {
    writing-mode: vertical-lr;
}
```

The page would then look like this:

Or you could change it to vertically align and flow from the right to the left:

```
.container {
    writing-mode: vertical-rl;
}
```

The page would then look like this:



Inline Axis Customized

The default direction on the inline axis is left to right. It is the way we read text. You start on the left and read the next word to the right of it.

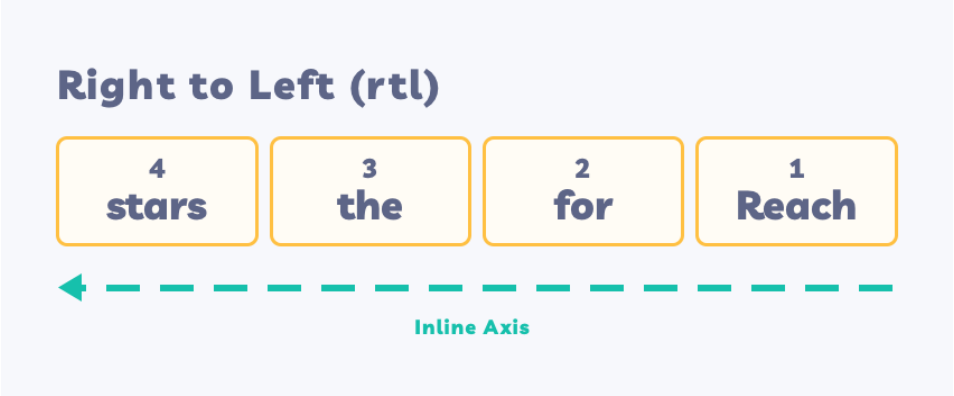
```
<html dir="ltr">...</html>
```

This looks just as you'd imagine it to look:

This is the default direction on browsers.

If you wanted to adjust this direction and have the words read from right to left you could change the direction like so:

```
<html dir="rtl">...</html>
```



In these examples the direction is assigned on the HTML element on the page.

You can also adjust direction as a CSS style like this:

```
.element {
    direction: rtl;
}
```

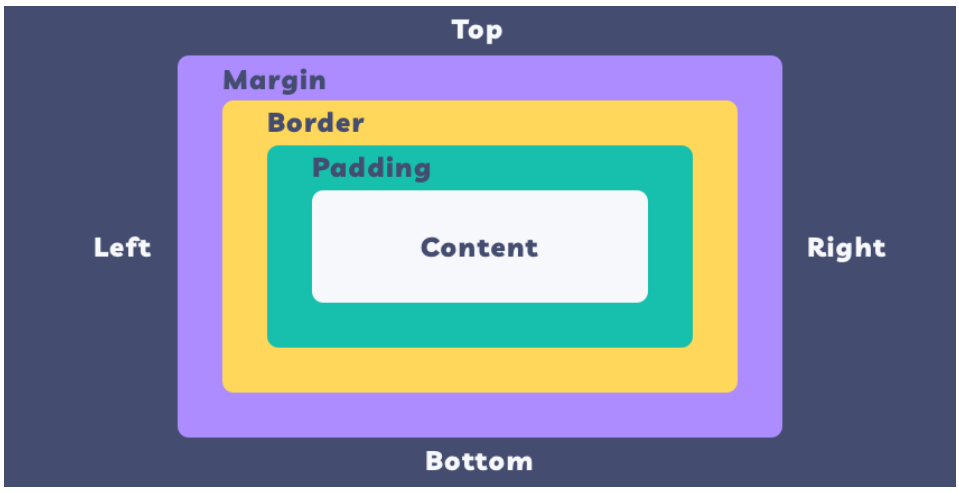
Even though you have access to both methods of assigning direction, the CSSWG's recommendation is to define it on the html root element. The reason they make this recommendation is because by assigning it as an HTML attribute it ensures the correct inline axis direction in the absence of CSS.

Logical Properties

Now that you understand block and inline axis let's cover the fundamentals of Logical Properties. You'll see how easy they are to understand and remember now.

Let's start with one of the most fundamental concepts in CSS: the box model. Simply put, the box model states that all elements have a box around them. Each box has a content area, padding, border, and margin.

We'll discuss the content area later in the chapter. Without focusing on the content area, here's what a traditional box model looks like:



This should look familiar.

If we wanted to add a top padding we'd do this in CSS:

```
.element {
    padding-top: 1rem;
}
```

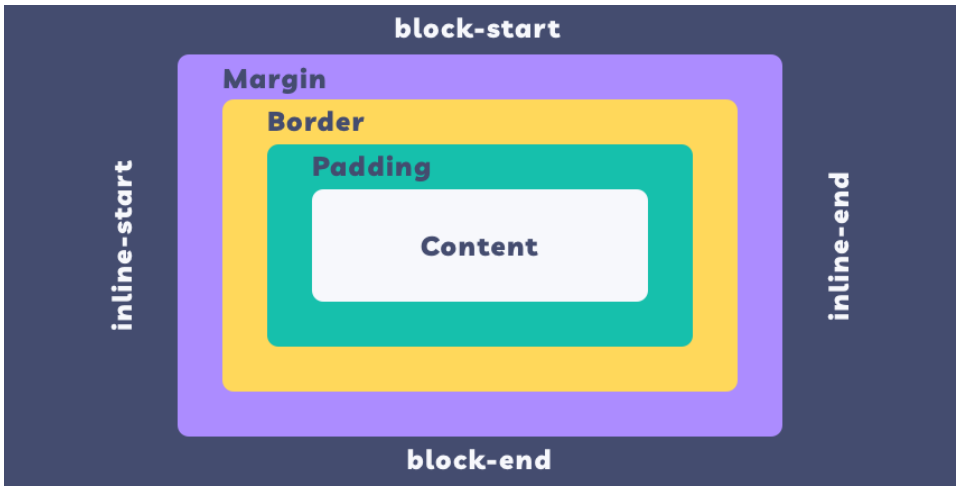
And if we wanted to add a margin on the right side:

```
.element {
    margin-right: 1rem;
}
```

Do you see how directly tied into the box model are those physical dimensions (top, bottom, left, right) that CSS is slowly going away from?

The Box Model

The new way to think about the box model using logical properties is like this:



Do you see how we use **block-start** and **block-end** along the **block axis** instead of top and bottom? And how we use **inline-start** and **inline-end** along the **inline axis** instead of left and right?

With those logical properties in place, if we wanted to add the same padding and margin as we did earlier, this is how we would do it:

```
.element {
    padding-block-start: 1rem;
}
```

```
.element {
    margin-inline-end: 1rem;
}
```

When done this way, you can change the block and inline axes and everything still works and displays correctly.

Let's jump in and learn the logical properties for padding, border, and margin. But, before we do, let's discuss if this is something you can use right now.

Browser Support

Before jumping in and learning these logical properties, can you use them right now? What is the browser support? The good news is that it's really good. Globally it has about 90% support and 96% when you include partial support (with a prefix). Internet Explorer and Opera Mini are the only browsers that don't currently support these properties *(updated 7 Sept 2020)*.

Padding

The logical properties for padding should make sense now that you understand the block and inline axes.

```
.element {
    padding-block-start: 1rem;   /* padding-top */
    padding-block-end: 1rem;   /* padding-bottom */
    padding-inline-start: 1rem;   /* padding-left */
    padding-inline-end: 1rem;   /* padding-right */
}
```

The formula for using these logical properties is:

```
padding-[block/inline]-[start/end]
```

Margin

Adding margin using logical properties is just the same as padding.

```
.element {
    margin-block-start: 1rem;   /* margin-top */
    margin-block-end: 1rem;   /* margin-bottom */
    margin-inline-start: 1rem;   /* margin-left */
    margin-inline-end: 1rem;   /* margin-right */
}
```

So, in addition to stating the border you would like to style, you can explicitly state the style, width, or color.

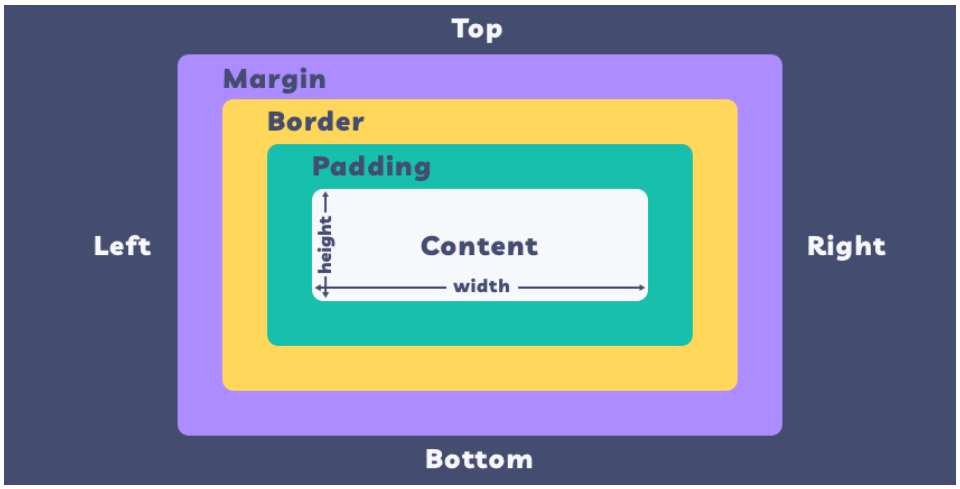
```
border-[block/inline]-[start/end]-[width/style/color]
```

All of these properties have the 90% global browser support coverage with 96% partial coverage (supported with a prefix).

Content Box Sizing

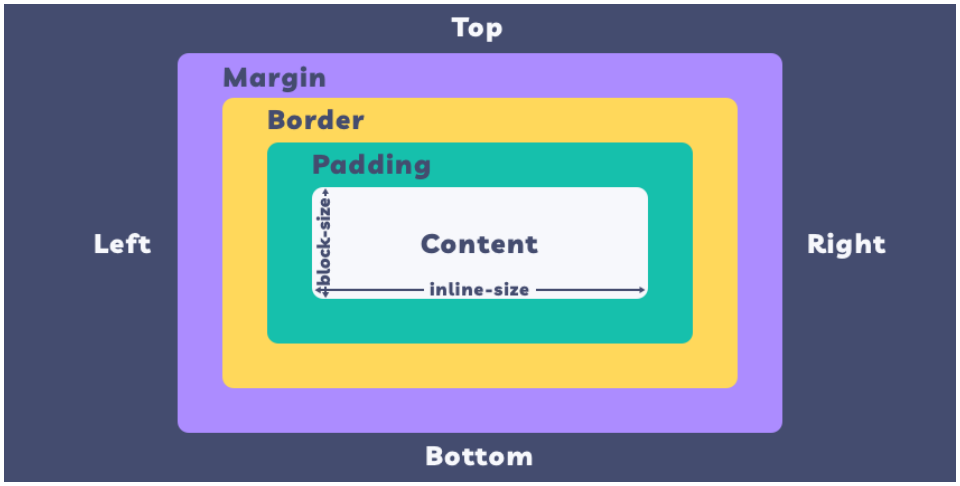
Let's go back to the traditional box model. This time let's look at the content area. Do you remember seeing this traditional box model?

Inside the content area we also have width and height to consider:



However, height and width are physical dimensions. If, for instance, the writing mode changed from horizontal to vertical the height and width would not be accurate descriptions. Hence, the need for logical properties.

The logical property for height is **block-size** and width is **inline-size**.



Now, no matter the writing mode the description of block-size or inline-size are now accurate in all conditions.

We can also set a **max-block-size**, **min-block-size**, **max-inline-size**, and **min-inline-size** just like you would if you set max-height, min-height, max-width, or min-width.

If the page has a writing mode of horizontal, top-to-bottom.

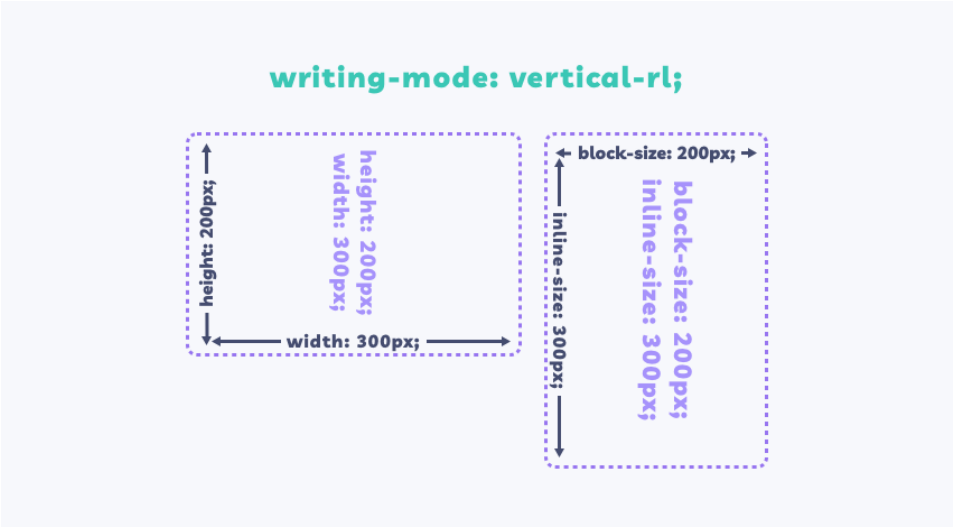
```
.container {
    writing-mode: horizontal-tb;
}
```

It would look like this:

And if the writing mode changed to vertical, right-to-left.

```
.container {
    writing-mode: vertical-rl;
}
```

It would look like this:



Text-Align

With text-align, instead of right or left you will only need to now state start or end:

```
.element {
    text-align: start;   /* left */
}
```

```
.element {
    text-align: end;   /* right */
}
```

Now you should have a strong foundational understanding of Logical Properties.

Even if you are not planning on creating a website in a language that reads vertically or right-to-left, it is good

practice to begin creating your websites using logical properties rather than physical dimensions.

If, for any reason, the writing mode or the inline direction changes on your website it will continue to display and flow correctly.

You will also better understand flexbox and CSS Grid, as well as any future updates to CSS that include logical properties.

Unsupported Logical Properties

Logical Properties is still very new. In the previous chapter you learned about the well supported aspects of Logical Properties. It's good to know what's coming down the pipeline though. And that's what this chapter is going to cover.

The following logical properties are good to know about and be aware of but not something I recommend using on your websites. At this point there is only about 4% browser support (only Firefox currently supports most of the following properties or values). In some cases there is no browser support, it's something they're working on right now.

Because there is value in knowing about what is shortly to come, here is a quick introduction and explanation of some new and upcoming Logical Properties you should know about.

Positioning

There are times when you want to absolutely position something on your page. You might do something like this:

```
.element {
    position: absolute;
    top: 0;
    right: 0;
    bottom: 0;
    left: 0;
}
```

You probably see a pattern here where physical position of top, right, bottom, and left are being replaced with logical properties. That is because if the writing mode changes from a horizontal to a vertical writing mode everything shifts naturally.

If you set position to absolute or another position value other than the default of static then you can follow this pattern:

- **inset-block-start** = **top**
- **inset-block-end** = **bottom**
- **inset-inline-start** = **left**
- **inset-inline-end** = **right**

So, it might look something like this:

```
.element {
    position: absolute;
    inset-block-start: 0;   /* top */
    inset-block-end: 0;   /* bottom */
    inset-inline-start: 0;   /* left */
    inset-inline-end: 0;   /* right */
}
```

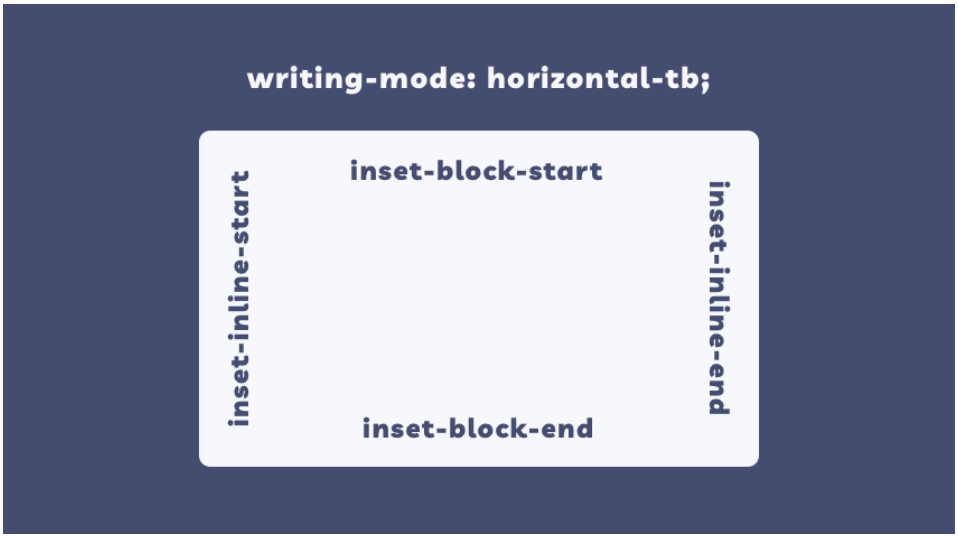
And you could even use the inline shorthand:

```
.element {
    position: absolute;
    inset-block: 0;   /* top and bottom */
    inset-inline: 0;   /* right and left */
}
```

Or even more shorthand:

```
.element {
    position: absolute;
    inset: 0;   /* top, bottom, right, left */
}
```

When the writing mode is horizontal, top-to-bottom this should look familiar:



NOTE: If you see **offset**, **offset-block**, **offset-inline**, **offset-block-start**, **offset-block-end**, **offset-inline-start**, or **offset-inline-end** these are all deprecated properties. Instead of **offset** now it is **inset**.

Box Model Shorthand Options

Just like inset offers some short hand options, there will soon be some short hand options for the box model.

[padding/margin/border]-block:

Quickly add top and bottom padding/margin/border with this shorthand. If you wanted to add a padding of 1rem on the top and the bottom of your element the shorthand would look like this:

```
.element {
    padding-block: 1rem;   /* top and bottom */
}
```

You could, of course, also do this with margin or border as well.

[padding/margin/border]-inline:

Just like the block shorthand, the inline shorthand allows you to add padding/margin/border on the left and right automatically:

```
.element {
    margin-inline: auto;   /* left and right */
}
```

Border also offers you a number of other shorthand options:

- border-block-color
- border-inline-color
- border-block-style
- border-inline-style
- border-block-width
- border-inline-width

Float and Clear

Another place we are used to using physical dimensions is when we float or clear things. In those cases we would use logical versions of the left and right values.

```
.element {
    float: inline-start;   /* left */
}
```

```
.element {
    float: inline-end;   /* right */
}
```

Both float and clear use the **inline-start** and **inline-end** keywords as values.

Other Properties

There are a few other properties worth briefly mentioning.

Border Radius

When specifying one corner at a time to adjust the border-radius you will soon be able to use logical properties to select which corder to target.

- border-start-start-radius
- border-start-end-radius
- border-end-start-radius
- border-end-end-radius

Caption Side

This is something you'll only use when working on a table's caption. It adjusts where the caption will be located on the table (the top, bottom, left, or right).

caption-side: [block-start, block-end, inline-start, inline-end];

Overflow

Overflow-x and overflow-y will soon be overflow-inline and overflow-block. The values remain the same: visible, hidden, clip, scroll, or auto.

Overscroll Behavior

Determine the browser's behavior when a block direction boundary of a scrolling area is reached. This is particularly useful on mobile when you want to customize the pull-to-refresh action that is now so common on mobile applications. The options are:

- overscroll-behavior-block
- overscroll-behavior-inline

Resize

This property sets whether an element is resizable and if so, in which direction. This is common on a <textarea>

where the default is to be resize-able. The values are:

- None
- Both
- Horizontal / Inline
- Vertical / Block

Download the Logical Properties Cheatsheet

Now that you understand everything you need to know about logical properties, download this logical properties cheatsheet to help you until you've fully mastered and memorized the logical properties naming and syntax.

Download Logical Properties Cheat Sheet