**Minimum and Maximum Height and Width**

Because a web page can be viewed through displays of differing screen size, the content on the web page can suffer from those changes in size. To avoid this problem, CSS offers two properties that can limit how narrow or how wide an element's box can be sized to.

1. min-width — this property ensures a minimum width of an element's box.
2. max-width — this property ensures a maximum width of an element's box.

p { min-width: 300px; max-width: 600px; }

In the example above, the width of all paragraphs will not shrink below 300 pixels, nor will the width exceed 600 pixels.

Content, like text, can become difficult to read when a browser window is narrowed or expanded. These two properties ensure that content is legible by limiting the minimum and maximum widths of an element.

You can also limit the minimum and maximum *height* of an element.

1. min-height — this property ensures a minimum height for an element's box.
2. max-height — this property ensures a maximum height of an element's box.

p { min-height: 150px; max-height: 300px; }

In the example above, the height of all paragraphs will not shrink below 150 pixels and the height will not exceed 300 pixels.

**Overflow**

All of the components of the box model comprise an element’s size. For example, an image that has the following dimensions is 364 pixels wide and 244 pixels tall.

* 300 pixels wide
* 200 pixels tall
* 10 pixels padding on the left and right
* 10 pixels padding on the top and bottom
* 2 pixels border on the left and right
* 2 pixels border on the top and bottom
* 20 pixels margin on the left and right
* 10 pixels margin on the top and bottom

The total dimensions (364px by 244px) are calculated by adding all of the vertical dimensions together and all of the horizontal dimensions together. Sometimes, these components result in an element that is larger than the parent's containing area.

How can we ensure that we can view all of an element that is larger than its parent's containing area?

The overflow property controls what happens to content that spills, or overflows, outside its box. It can be set to one of the following values:

* hidden - when set to this value, any content that overflows will be hidden from view.
* scroll - when set to this value, a scrollbar will be added to the element's box so that the rest of the content can be viewed by scrolling.
* visible - when set to this value, the overflow content will be displayed outside of the containing element. Note, this is the default value.

p { overflow: scroll; }

In the example above, if any of the paragraph content overflows (perhaps a user resizes their browser window), a scrollbar will appear so that users can view the rest of the content.

The overflow property is set on a parent element to instruct a web browser how to render child elements. For example, if a div’s overflow property is set to scroll, all children of this div will display overflowing content with a scroll bar.

# Resetting Defaults

All major web browsers have a default stylesheet they use in the absence of an external stylesheet. These default stylesheets are known as user agent stylesheets. In this case, the term "[user agent](https://en.wikipedia.org/wiki/User_agent)" is a technical term for the browser.

User agent stylesheets often have default CSS rules that set default values for padding and margin. This affects how the browser displays HTML elements, which can make it difficult for a developer to design or style a web page.

Many developers choose to reset these default values so that they can truly work with a clean slate.

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

The code in the example above resets the default margin and padding values of all HTML elements. It is often the first CSS rule in an external stylesheet.

Note that both properties are both set to 0. When these properties are set to 0, they do not require a unit of measurement.

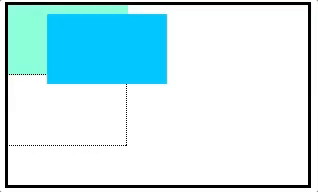
# Position: Absolute

Another way of modifying the position of an element is by setting its position to absolute.

When an element's position is set to absolute all other elements on the page will ignore the element and act like it is not present on the page. The element will be positioned relative to its closest positioned parent element.

.box-bottom { background-color: DeepSkyBlue; position: absolute; top: 20px; left: 50px; }

In the example above, the .box-bottom <div> will be moved down and right from the top left corner of the view. If offset properties weren't specified, the top box would be entirely covered by the bottom box. Take a look at the gif below:



The bottom box in this image (colored blue) is displaced from the top left corner of its container. It is 20 pixels lower and 50 pixels to the right of the top box.

In the next exercise, we will compare the scrolling of absolute elements with fixed elements.

**NOTES:**

1. The position property allows you to specify the position of an element in three different ways.
2. When set to relative, an element's position is relative to its default position on the page.
3. When set to absolute, an element's position is relative to its closest positioned parent element. It can be pinned to any part of the web page, but the element will still move with the rest of the document when the page is scrolled.
4. When set to fixed, an element's position can be pinned to any part of the web page. The element will remain in view no matter what.
5. The z-index of an element specifies how far back or how far forward an element appears on the page when it overlaps other elements.
6. The display property allows you control how an element flows vertically and horizontally a document.
7. inline elements take up as little space as possible, and they cannot have manually-adjusted width or height.
8. block elements take up the width of their container and can have manually-adjusted heights.
9. inline-block elements can have set width and height, but they can also appear next to each other and do not take up their entire container width.
10. The float property can move elements as far left or as far right as possible on a web page.
11. You can clear an element's left or right side (or both) using the clear property.

# Color Review

We've completed our extensive tour of the colors in CSS! Let's review the key information we've learned.

There are four ways to represent color in CSS:

* Named colors — there are 147 named colors, which you can review [here](https://msdn.microsoft.com/en-us/library/aa358802(v=vs.85).aspx).
* Hexadecimal or hex colors
  + Hexadecimal is a number system with has sixteen digits, 0 to 9 followed by "A" to "F".
  + Hex values always begin with # and specify values of red, blue and green using hexademical numbers such as #23F41A.
* RGB
  + RGB colors use the rgb() syntax with one value for red, one value for blue and one value for green.
  + RGB values range from 0 to 255 and look like this: rgb(7, 210, 50).
* HSL
  + HSL stands for hue (the color itself), saturation (the intensity of the color), and lightness (how light or dark a color is).
  + Hue ranges from 0 to 360 and saturation and lightness are both represented as percentages like this: hsl(200, 20%, 50%).
* You can add opacity to color in RGB and HSL by adding a fourth value, a, which is represented as a percentage.

Great job! Feel empowered to add a bit of color to each of your projects!

# Review

Great job! You learned how to style an important aspect of the user experience, typography.

Let's review what you've learned so far:

* Typography is the art of arranging text on a page.
* Text can appear in any number of weights, with the font-weight property.
* Text can appear in italics with the font-style property.
* The vertical spacing between lines of text can be modified with the line-height property.
* Serif fonts have extra details on the ends of each letter. Sans-Serif fonts do not.
* Fallback fonts are used when a certain font is not installed on a user's computer.
* Google Fonts provides free fonts that can be used in an HTML file with the <link> tag or the @font-face property.
* Local fonts can be added to a document with the @font-face property and the path to the font's source.
* The word-spacing property changes how far apart individual words are.
* The letter-spacing property changes how far apart individual letters are.
* The text-align property changes where text horizontally on a page.

**CSS Grid Essentials**

# minmax

So far, all of the grids that we have worked with have been a fixed size. The grid in our example has been 400 pixels wide and 500 pixels tall. But sometimes you might want a grid to resize based on the size of your web browser.

In these situations, you might want to prevent a row or column from getting too big or too small. For example, if you have a 100-pixel wide image in your grid, you probably don't want its column to get thinner than 100 pixels! The minmax() function can help us solve this problem.

.grid { display: grid; grid-template-columns: 100px minmax(100px, 500px) 100px; }

In this example, the first and third columns will always be 100 pixels wide, no matter the size of the grid. The second column, however, will vary in size as the overall grid resizes. The second column will always be between 100 and 500 pixels wide.

**Review**

At this point, we've covered a great deal of different ways to manipulate the grid and the items inside it to create interesting layouts.

* grid-template-columns defines the number and sizes of the columns of the grid
* grid-template-rows defines the number and sizes of the rows of the grid
* grid-template is a shorthand for defining both grid-template-columns and grid-template-rows in one line
* grid-gap puts blank space between rows and/or columns of the grid
* grid-row-start and grid-row-end makes elements span certain rows of the grid
* grid-column-start and grid-column-end makes elements span certain columns of the grid
* grid-area is a shorthand for grid-row-start, grid-column-start, grid-row-end, and grid-column-end, all in one line

You have seen how to set up and fill in a grid and you now have one more CSS positioning technique to add to your toolkit! Let's do some practice to solidify these skills.

**Grid Template Areas**

The grid-template-areas property allows you to name sections of your web page to use as values in the grid-row-start, grid-row-end, grid-col-start,grid-col-end, and grid-area properties.

<div class="container"> <header>Welcome!</header> <nav>Links!</nav> <section class="info">Info!</section> <section class="services">Services!</section> <footer>Contact us!</footer> </div>

.container { display: grid; max-width: 900px; position: relative; margin: auto; grid-template-areas: "head head" "nav nav" "info services" "footer footer"; grid-template-rows: 300px 120px 800px 120px; grid-template-columns: 1fr 3fr; } header { grid-area: head; } nav { grid-area: nav; } .info { grid-area: info; } .services { grid-area: services; } footer { grid-area: footer; }

You may want to expand this section of the website to view the code above more clearly.

1. In the example above, the HTML creates a web page with five distinct parts.
2. The grid-template-areas declaration in the .container rule set creates a 2-column, 4-row layout.
3. The grid-template-rows declaration specifies the height of each of the four rows from top to bottom: 300 pixels, 120 pixels, 800 pixels, and 120 pixels.
4. The grid-template-columns declaration uses the fr value to cause the left column to use one fourth of the available space on the page and the right column to use three-fourths of the available space on the page.
5. In each rule set below .container, we use the grid-area property to tell that section to cover the portion of the page specified. The header element spans the first row and both columns. The nav element spans the second row and both columns. The element with class .info spans the third row and left column. The element with class .services spans the third row and right column. The footer element spans the bottom row and both columns.
6. That's it! An entire page laid out in 40 lines of code.

# Overlapping Elements

Another powerful feature of CSS Grid Layout is the ability to easily overlap elements.

When overlapping elements, it is generally easiest to use grid line names and the grid-area property.

<div class="container"> <div class="info">Info!</div> <img src="#" /> <div class="services">Services!</div> </div>

.container { display: grid; grid-template: repeat(8, 200px) / repeat(6, 100px); } .info { grid-area: 1 / 1 / 9 / 4; } .services { grid-area: 1 / 4 / 9 / 7; } img { grid-area: 2 / 3 / 5 / 5; z-index: 5; }

In the example above, there is a grid container with eight rows and six columns. There are three grid items within the container — a <div> with the class info, a <div> with the class services, and an image.

The info section covers all eight rows and the first three columns. The services section covers all eight rows and the last three columns.

The image spans the 2nd, 3rd, and 4th rows and the 3rd and 4th columns.

The z-index property tells the browser to render the image element on top of the services and info sections so that it is visible.

# Justify Items

We have referred to "two-dimensional grid-based layout" several times throughout this course.

There are two axes in a grid layout — the column (or block) axis and the row (or inline) axis.

The column axis stretches from top to bottom across the web page.

The row axis stretches from left to right across the web page.

In the following four exercises, we will learn and use properties that rely on an understanding of grid axes.

justify-items is a property that positions grid items along the inline, or row, axis. This means that it positions items from left to right across the web page.

justify-items accepts these values:

* start — aligns grid items to the left side of the grid area
* end — aligns grid items to the right side of the grid area
* center — aligns grid items to the center of the grid area
* stretch — stretches all items to fill the grid area

There are several other values that justify-items accepts, which you can read about on the [Mozilla Developer Network](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout/Box_Alignment_in_CSS_Grid_Layout#Justifying_Items_on_the_Inline_or_Row_Axis). The definitions for these values can also be found in the [documentation](https://developer.mozilla.org/en-US/docs/Web/CSS/justify-items#Values). It is important to note that the page with the definitions includes some values that are not accepted in CSS Grid layout.

<main> <div class="card">Card 1</div> <div class="card">Card 2</div> <div class="card">Card 3</div> </main>

main { display: grid; grid-template-columns: repeat(3, 400px); justify-items: center; }

In the example above, we use justify-items to adjust the positioning of some elements on this web page.

1. There is a grid container with three columns that are each 400 pixels wide.
2. The container has three grid items that do not have a specified width.
3. Without setting the justify-items property, these elements will span the width of the column they are in (400 pixels).
4. By setting the justify-items property to center, the .card <div>s will be centered inside of their columns. They will only be as wide as necessary to contain their content (the words Card 1, etc).
5. If we specify a width for the .card elements, they will not stretch the width of their column.

# Justify Content

In the previous exercise, we learned how to position elements within their columns. In this exercise, we will learn how to position a grid within its parent element.

We can use justify-content to position the entire grid along the row axis.

It accepts these values:

* start — aligns the grid to the left side of the grid container
* end — aligns the grid to the right side of the grid container
* center — centers the grid horizontally in the grid container
* stretch — stretches the grid items to increase the size of the grid to expand horizontally across the container
* space-around — includes an equal amount of space on each side of a grid element, resulting in double the amount of space between elements as there is before the first and after the last element
* space-between — includes an equal amount of space between grid items and no space at either end
* space-evenly — places an even amount of space between grid items and at either end

There are several other values that justify-content accepts, which you can read about on the [Mozilla Developer Network](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout/Box_Alignment_in_CSS_Grid_Layout#Aligning_the_grid_tracks_on_the_block_or_column_axis). The definitions for these values can also be found in the [documentation](https://developer.mozilla.org/en-US/docs/Web/CSS/justify-content#Values). It is important to note that the page with the definitions includes some values that are not accepted in CSS Grid layout.

<main> <div class="left">Left</div> <div class="right">Right</div> </main>

main { display: grid; width: 1000px; grid-template-columns: 300px 300px; grid-template-areas: "left right"; justify-content: center; }

1. In the example above, the grid container is 1000 pixels wide, but we only specified two columns that are 300 pixels each. This will leave 400 pixels of unused space in the grid container.
2. justify-content: center; positions the columns in the center of the grid, leaving 200 pixels on the right and 200 pixels on the left of the grid.

This property is declared on grid containers.

# Align Items

In the previous two exercises, we learned how to position grid items and grid columns from left to right across the page. Below, we'll learn how to position grid items from top to bottom!

align-items is a property that positions grid items along the block, or column axis. This means that it positions items from top to bottom.

align-items accepts these values:

* start — aligns grid items to the top side of the grid area
* end — aligns grid items to the bottom side of the grid area
* center — aligns grid items to the center of the grid area
* stretch — stretches all items to fill the grid area

There are several other values that align-items accepts, which you can read about on the [Mozilla Developer Network](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout/Box_Alignment_in_CSS_Grid_Layout#Aligning_items_on_the_block_or_column_Axis). The definitions for these values can also be found in the [documentation](https://developer.mozilla.org/en-US/docs/Web/CSS/justify-items#Values). It is important to note that the page with the definitions includes some values that are not accepted in CSS Grid layout.

<main> <div class="card">Card 1</div> <div class="card">Card 2</div> <div class="card">Card 3</div> </main>

main { display: grid; grid-template-rows: repeat(3, 400px); align-items: center; }

In the example above, we use align-items to adjust the positioning of some elements on this web page.

1. There is a grid container with three rows that are 400 pixels tall.
2. The container has three grid items that do not have a specified width.
3. Without setting the align-items property, these elements will span the height of the row they are in (400 pixels).
4. By setting the align-items property to center, the .card <div>s will be centered vertically inside of their rows. They will only be as tall as necessary to contain their content (the words Card 1, etc).
5. If we specify a height for the .card elements, they will not stretch the height of their row even if align-items: stretch; is set.

This property is declared on grid containers.

# Align Content

In the previous exercise, we positioned grid items within their rows. align-content positions the rows along the column axis, or from top to bottom.

It accepts these positional values:

* start — aligns the grid to the top of the grid container
* end — aligns the grid to the bottom of the grid container
* center — centers the grid vertically in the grid container
* stretch — stretches the grid items to increase the size of the grid to expand vertically across the container
* space-around — includes an equal amount of space on each side of a grid element, resulting in double the amount of space between elements as there is before the first and after the last element
* space-between — includes an equal amount of space between grid items and no space at either end
* space-evenly — places an even amount of space between grid items and at either end

There are several other values that align-content accepts, which you can read about on the [Mozilla Developer Network](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout/Box_Alignment_in_CSS_Grid_Layout#Aligning_the_grid_tracks_on_the_block_or_column_axis). The definitions for these values can also be found in the [documentation](https://developer.mozilla.org/en-US/docs/Web/CSS/align-content#Values). It is important to note that the page with the definitions includes some values that are not accepted in CSS Grid layout.

<main> <div class="top">Top</div> <div class="bottom">Bottom</div> </main>

main { display: grid; height: 600px; rows: 200px 200px; grid-template-areas: "top" "bottom"; align-content: center; }

1. In the example above, the grid container is 600 pixels tall, but we only specified two rows that are 200 pixels each. This will leave 200 pixels of unused space in the grid container.
2. align-content: center; positions the rows in the center of the grid, leaving 100 pixels at the top and 100 pixels at the bottom of the grid.

This property is declared on grid containers.

# Justify Self and Align Self

The justify-items and align-items properties specify how all grid items contained within a single container will position themselves along the row and column axes, respectively.

justify-self specifies how an individual element should position itself with respect to the row axis. This property will override justify-items for any item on which it is declared.

align-self specifies how an individual element should position itself with respect to the column axis. This property will override align-items for any item on which it is declared.

They both accept these four properties:

* start — positions grid items on the left side/top of the grid area
* end — positions grid items on the right side/bottom of the grid area
* center — positions grid items on the center of the grid area
* stretch — positions grid items to fill the grid area (default)

align-self and justify-self accept the same values as align-items and justify-items. You can read about these values on the [Mozilla Developer Network](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout/Box_Alignment_in_CSS_Grid_Layout#Aligning_the_grid_tracks_on_the_block_or_column_axis). The definitions for these values can also be found in the [documentation](https://developer.mozilla.org/en-US/docs/Web/CSS/align-self#Values). It is important to note that the page with the definitions includes some values that are not accepted in CSS Grid layout.

These properties are declared on grid items.

**Grid Auto Rows and Grid Auto Columns**

CSS Grid provides two properties to specify the size of grid tracks added implicitly: grid-auto-rows and grid-auto-columns.

grid-auto-rows specifies the height of implicitly added grid rows. grid-auto-columns specifies the width of implicitly added grid columns.

grid-auto-rows and grid-auto-columns accept the same values as their explicit counterparts, grid-template-rows and grid-template-columns:

* pixels (px)
* percentages (%)
* fractions (fr)
* the repeat() function

<body> <div>Part 1</div> <div>Part 2</div> <div>Part 3</div> <div>Part 4</div> <div>Part 5</div> </body>

body { display: grid; grid: repeat(2, 100px) / repeat(2, 150px); grid-auto-rows: 50px; }

In the example above, there are 5 <div>s. However, in the section rule set, we only specify a 2-row, 2-column grid — four grid cells.

The fifth <div> will be added to an implicit row that will be 50 pixels tall.

If we did not specify grid-auto-rows, the rows would be auto-adjusted to the height of the content of the grid items.

These properties are declared on grid containers.

**Grid Auto Flow**

In addition to setting the dimensions of implicitly-added rows and columns, we can specify the order in which they are rendered.

grid-auto-flow specifies whether new elements should be added to rows or columns.

grid-auto-flow accepts these values:

* row — specifies the new elements should fill rows from left to right and create new rows when there are too many elements (default)
* column — specifies the new elements should fill columns from top to bottom and create new columns when there are too many elements
* dense — this keyword invokes an algorithm that attempts to fill holes earlier in the grid layout if smaller elements are added

You can pair row and column with dense, like this: grid-auto-flow: row dense;.

This property is declared on grid containers.

**Review**

Great work! You have learned many new properties to use when creating a layout using CSS Grid! Let's review:

* grid-template-areas specifies grid named grid areas
* grid layouts are two-dimensional: they have a row, or inline, axis and a column, or block, axis.
* justify-items specifies how individual elements should spread across the row axis
* justify-content specifies how groups of elements should spread across the row axis
* justify-self specifies how a single element should position itself with respect to the row axis
* align-items specifies how individual elements should spread across the column axis
* align-content specifies how groups of elements should spread across the column axis
* align-self specifies how a single element should position itself with respect to the column axis
* grid-auto-rows specifies the height of rows added implicitly to the grid
* grid-auto-columns specifies the width of columns added implicitly to the grid
* grid-auto-flow specifies in which direction implicit elements should be created

This is a great time to experiment with the code in the code editor and try any of the properties you want to practice more! When you're ready, move on!