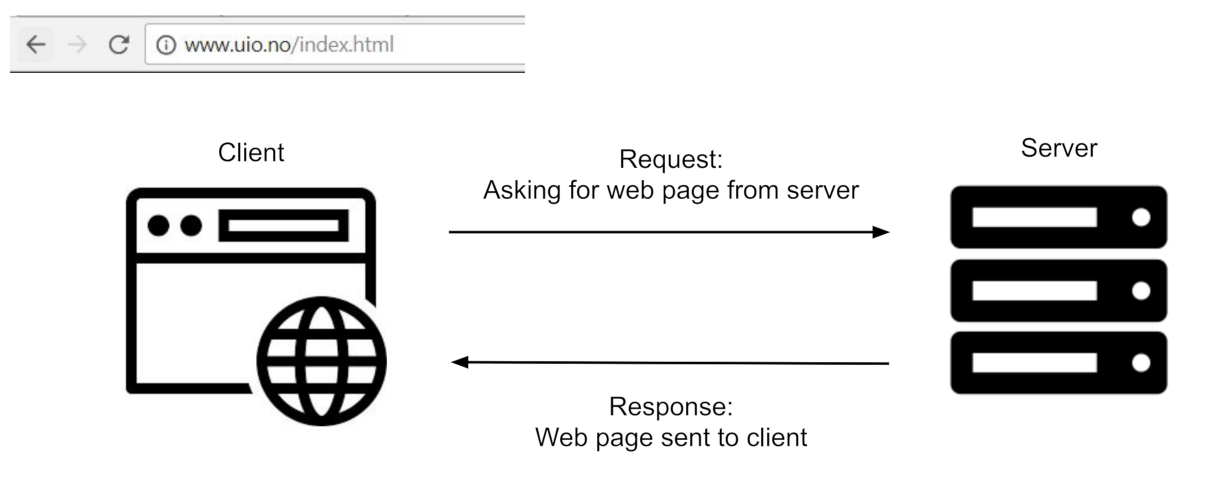
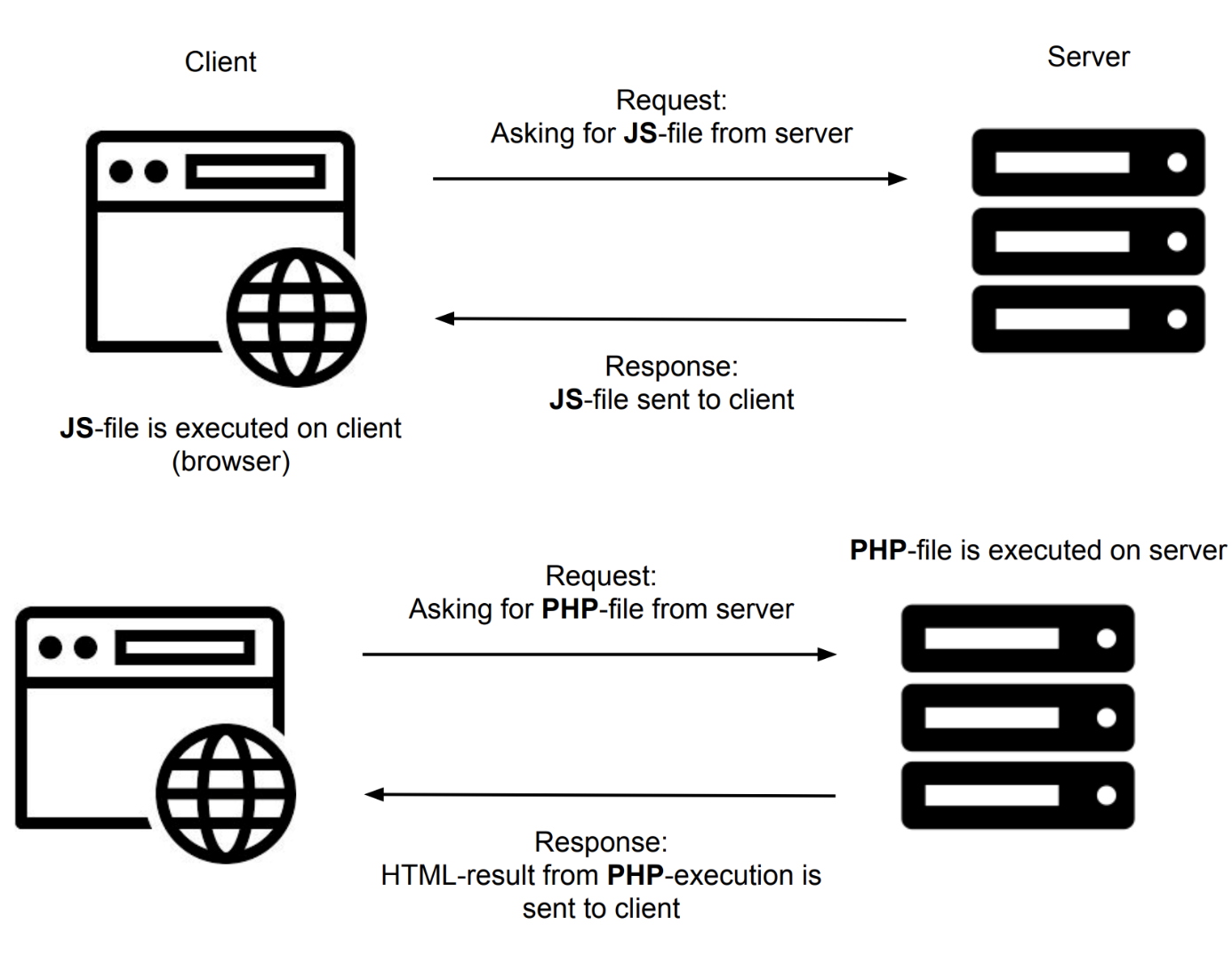
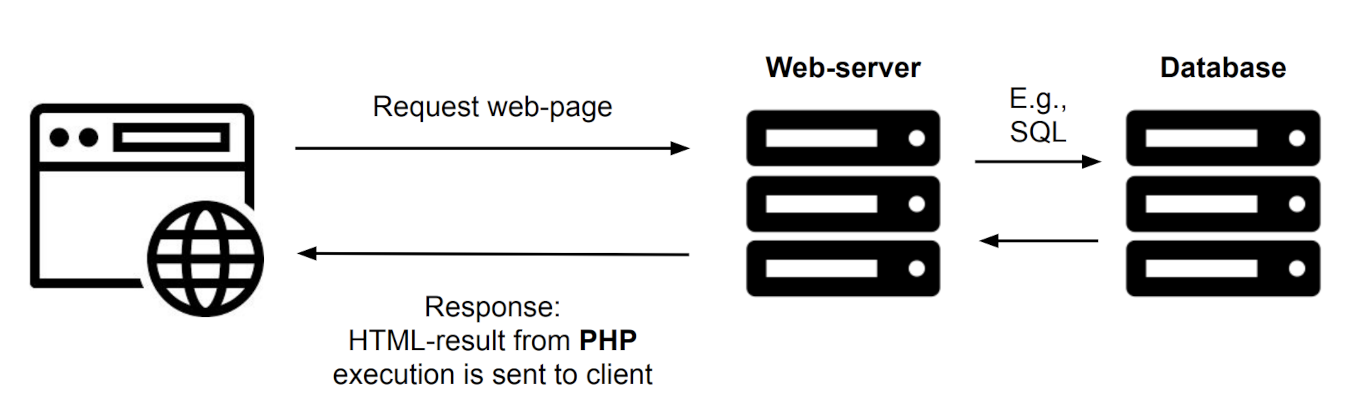
**Server and client-side**



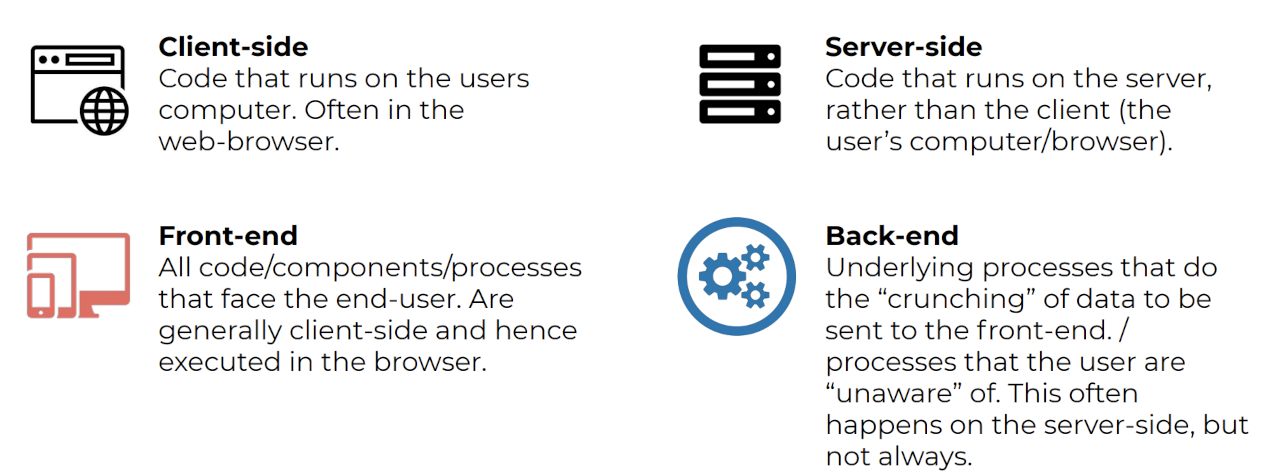
Figur 1Client requesting a resource from a server, and the server sending a response.



Figur 2Whereas a JavaScript or HTML-file requested from the server is returned to the client and executed there, server-side code in, for instance, PHP is executed on the server and the result from executing the code is returned as response to the client.



Figur 3Server fetching data from database and returning result to client.



Figur 4The concepts of client-side and server-side versus front-end and back-end.

# HTML

The <doctype> tag tells the browser that it's looking at an HTML document.

We specify the language of the document (English) next in the <html> tag.

The <head> element contains information about the current document, also known as metadata.

The <body> element defines the document's main body; what we can see.



# HTML Elements

## Headings

# We can create headings using the <h1>, <h2>, <h3>, <h4>, <h5> and <h6>

### Code example

<h1>My main title</h1>

<h2>Subtitle</h2>

## Paragraphs

Paragraphs are specified to be [block-level elements](https://developer.mozilla.org/en-US/docs/Web/HTML/Block-level_elements) and can be created using the <p> tag.

### Code example

<p>

Paragraphs are usually separated by blank lines

because they make it easier to scan through a page.

</p>

<p>This is a separate paragraph</p>

## Images

The HTML <img> tag embeds an image into a document.

| **Attribute** | **Description** | **Required** |
| --- | --- | --- |
| src | path of the image we want to embed | yes |
| alt | alternative description of the image | highly recommended, but image will still be shown if not specified |
| width | intrinsic width of the image | no |
| height | intrinsic height of the image | no |

### Code example

<img

alt="Picture of snow on a mountain top"

src="/learn/essential-front-end/html/elements/snow-mountain.jpg"

width="536"

height="354"

>

### Closing tag

The <img> tag is a so-called [void or empty element](https://developer.mozilla.org/en-US/docs/Glossary/Empty_element) which means that it **cannot** have any child nodes. Void element tags should not be closed in the usual fashion as shown in the example.

## Links

We use the HTML <a> (anchor) tag to create a hyperlink to another address.

| **Attribute** | **Description** | **Required** |
| --- | --- | --- |
| href | URL address of target location | no |
| title | tooltip (shown when hovered) | generally not needed |
| target | how to open the link (new tab, window, etc) | no |
| rel | relationship to the linked URL | no |

### Code example

<a

href="https://en.wikipedia.org/wiki/LLVM"

target="\_blank"

rel="noreferrer">

Example hyperlink to the wikipedia page about LLVM

</a>

## Lists

HTML provides us with two slighly different tags for creating lists: <ul> for unordered lists and <ol> for ordered lists.

We can nest lists as deep as we want, as long as we always be wrap the content in <li> tags (list items).

### Code example: unordered

<ul>

<li>The Lathe of Heaven</li>

<li>Rocannon's World</li>

<li>The Word for World is Forest</li>

</ul>

produces the unordered list below.

* *The Lathe of Heaven*
* *Rocannon's World*
* *The Word for World is Forest*

### Code example: ordered

<ol>

<li>Turn off the computer</li>

<li>Go outside</li>

<li>Contemplate the meaning of life</li>

</ol>

produces the ordered list below.

1. *Turn off the computer*
2. *Go outside*
3. *Contemplate the meaning of life*

### Code example: nested

<ul>

<li>Fantasy

<ol>

<li>The Eye of the World</li>

<li>The Lies of Locke Lamora</li>

</ol>

</li>

<li>Science fiction

<ol>

<li>Dune</li>

<li>Ender's Game</li>

</ol>

</li>

<li>Classics

<ol>

<li>Of Mice and Men</li>

<li>Mockingbird</li>

</ol>

</li>

</ul>

produces the nested list below.

* *Fantasy*
  1. *The Eye of the World*
  2. *The Lies of Locke Lamora*
* *Science fiction*
  1. *Dune*
  2. *Ender's Game*
* *Classics*
  1. *Of Mice and Men*
  2. *Mockingbird*

## Dividers

The HTML <div> tag is used to group relevant content together in a separate container.

The tag might seem useless right now, but it will show its full strength later on when combined with [CSS](https://dhis2-app-course.ifi.uio.no/learn/essential-front-end/css/), which makes it possible to target <div>s based on their id or class attributes and style them accordingly.

### Code example

<div class="shadowedBox greenBox">

<ul>

<li>Guide to Troubled Birds</li>

<li>Papillon</li>

<li>Foundation</li>

</ul>

</div>

has the potential of producing the box below (given some styling) 

## Buttons

Buttons that represents an action can be created using the HTML <button> tag. We can use buttons to submit forms, signal clickable actions and attach [JavaScript](https://dhis2-app-course.ifi.uio.no/learn/javascript/introduction/) triggers.

| **Attribute** | **Description** | **Required** |
| --- | --- | --- |
| disabled | whether the button should be disabled | no |
| type | type of the button (button, submit or reset) | no (defaults to regular button in most cases, but will default to submit button in forms) |

### Code example

<button>

Download game

</div>

has the potential of producing the button below (given some styling)



## Text formats

The HTML <strong> and <em> gives us a way of defining text with special meaning.

### Code example

<p>

<strong>

I am important and should be taken seriously.

</strong>

I love <em>carrots</em>.

Emphasis on carrots.

</p>

produces the paragraph below

***I am important and should be taken seriously.***

*I love*carrots*. Emphasis on carrots.*

## Inputs

HTML user <input>s are used to create interactive elements in web-based forms.

We usually use <input> elements inside a <form>, which makes it possible to validate input data client side.

| **Attribute** | **Description** | **Required** |
| --- | --- | --- |
| type | type of the input | no (defaults to text) |
| name | name of the input | yes (is submitted with the form as a name/value pair) |
| id | id of the input | no (useful when using labels) |
| placeholder | content to appear when input is empty | no (it is often better to use labels instead) |
| value | value of the input | no |
| disabled | whether the input should be disabled | no |

### Labels

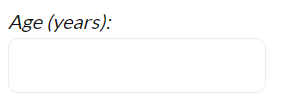
The HTML <label> element should be used to describe each and every visible <input>

#### Code example

<label for="age">Age (years):</label>

<input type="number" name="age" id="age" required>

which produces (give some styling)



#### Checkbox

Checkboxes should be used to toggle an option on or off.

<label for="displayEmptyColumns">Display empty columns:</label>

<input type="checkbox" name="displayEmptyColumns" id="displayEmptyColumns">

which produces (give some styling)



| **Attribute** | **Description** | **Required** |
| --- | --- | --- |
| checked | whether checkbox should be toggled on by default | no (defaults to false) |

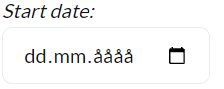
#### Date

Date <input>s are used to create input fields where the user can enter a date, either by picking it from a native browser date picker or typing it in.

<label for="startDate">Start date:</label>

<input type="date" name="startDate" id="startDate">

which produces (give some styling)



| **Attribute** | **Description** | **Required** |
| --- | --- | --- |
| max | latest acceptable date | no |
| min | earliest acceptable date | no |

#### File

The file input type is used for file selection.

<label for="license">License:</label>

<input type="file" name="license" id="license"

accept="image/png, image/jpeg">

which produces (give some styling)



| **Attribute** | **Description** | **Required** |
| --- | --- | --- |
| accept | what kind of file types to accept | no |
| multiple | whether to allow selection of multiple files | no (defaults to false) |

#### Number

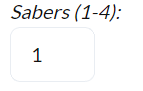
The number input type allows users to enter a number.

<label for="sabers">Sabers (1-4):</label>

<input type="sabers" name="sabers" id="sabers"

min="1" max="4" value="1" required>

which produces (give some styling)



| **Attribute** | **Description** | **Required** |
| --- | --- | --- |
| max | maximum acceptable number | no |
| min | minimum acceptable number | no |
| step | stepping interval when using the built-in arrows to adjust value | no |

#### Password

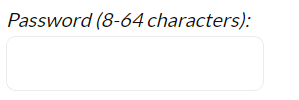
The password input type is a text field where user input is visually obscured such that it cannot easily be read.

<label for="password">Password (8-64 characters):</label>

<input type="password" name="password" id="password"

minlength="8" maxlength="64" required>

which produces (give some styling)



| **Attribute** | **Description** | **Required** |
| --- | --- | --- |
| maxlength | maximum length of password | no |
| minlength | minimum length of password | no |

#### Radio

We can use radio input type elements when we want to allow the user to select a single option from a choice of several.

<div>

Please select preferred shipping method:

<label>

<input type="radio" name="shippingMethod"

value="dhl" checked>

DHL

</label>

<label>

<input type="radio" name="shippingMethod"

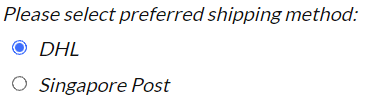
value="singapore-post">

Singapore Post

</label>

</div>

which produces (give some styling)



| **Attribute** | **Description** | **Required** |
| --- | --- | --- |
| checked | whether the radio is checked/selected | no |
| value | the value submitted when form is submitted paired with the name | no (defaults to 'on') |

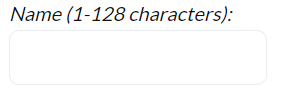
#### Text

The text is the default input type and is in many ways the most generic type.

<label for="name">Name (1-128 characters):</label>

<input type="text" name="name" id="name"

minlength="1" maxlength="128" required>



| **Attribute** | **Description** | **Required** |
| --- | --- | --- |
| maxlength | maximum length of text | no |
| minlength | minimum length of text | no |

## What is CSS?

CSS is a language used to describe how elements should be shown on a screen. The CSS standard is one of the open Web's core languages and all browser follow a common specification.

## How do we use CSS?

The last alternative, which is also the most commonly used one, is to write rules in a separate .css file and reference said file in our <head> element.

### Rules

CSS is based around rules. We define rules using selectors to target document elements, and put properties and their values inside curly braces.

p {

color: green;

font-size: 2em;

}

In the example above our selector is p which means that we want all <p> elements to have the specified style. Inside the curly braces, we have **declarations**. Each declaration is a property/value pair that sets a property to the specified value. Declarations must be separated by a semicolon and an optional new-line.

### Selectors

We select HTML document using selectors. These selectors can among other target tag types, class names and ids.

h1 {

text-transform: uppercase;

}

The code example above uses the h1 selector which targets all <h1> elements on a page.

.my-custom-class {

padding: 1rem;

}

Putting a . in front of a selector makes it a class selector which means that all elements with the given class gets that rule applied to them.

#my-custom-id {

margin: 1rem;

}

Putting a # in front of a selector makes it an id selector which means that all elements with the given id gets that rule applied to them.

p.my-custom-p-class {

border: 1px solid black;

}

We can combine these selectors as shown in the code example above where only <p> elements with the specified my-custom-p-class class gets the rule applied to them.

### Inline

Inlining CSS is often the quickest way to test if a certain property does what you want.

<h2 style="color: red;">

My text color is red!

</h2>

### Style tag

Putting all of our styling inside a <style> tag works for smaller web sites, but can be hard to maintain for larger web sites.

<style>

.large {

font-size: 5em;

}

.bold {

font-weight: 700;

}

</style>

<h2 class="large bold">

I am large and bold.

</h2>

### Separate file

Writing all of our css in separate .css files and referring to them using <link> inside our <head> element is the most common way of including CSS in HTML pages.

index.html

*<!doctype html>*

<html lang="en">

<head>

<meta charset="utf-8">

<title>Original title</title>

<link rel="stylesheet" type="text/css" href="styles.css">

</head>

<body>

<p class="brag">

I am using a class defined in the styles.css

file referred to in the <head> element.

</p>

</body>

</html>

styles.css

.brag {

color: red;

font-weight: 700;

}

# CSS Colors

There are essentially three ways of specifying colors in CSS:

**Keywords:** black, silver, gray, purple

**RGB:** **#RRGGBB, rgb(255, 10, 255, 50%)**

**HSL: hsl(270, 60%, 70%), hsl(270, 60%, 50%, 15%)**

# CSS Length Units

* **Pixels**: absolute length unit using the monitor's pixels as a base
  + 512px, 1024px
* **Percentage of parent**: relative length unit using the parent element's lengths as a base
  + 25%, 50%
* **em**: relative to the current element's font size
  + 2em
* **rem**: relative to the root element's font size (usually the <html> element)
  + 3rem
* **Percentage of viewport**: relative to the current device's viewport width and height
  + 50vw

## Pixels

### Code example

<style>

#blue-box {

width: 250px;

height: 250px;

background-color: blue,

color: white;

}

#red-box {

width: 200px;

height: 100px;

background-color: red,

color: black;

}

</style>

<div id="blue-box">

I am a blue box with a width of 250px and a height

of 250px

<div id="red-box">

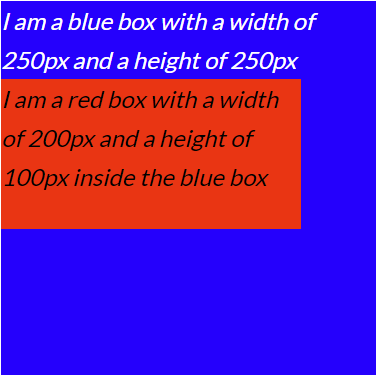
I am a red box with a width of 200px and a height

of 100px inside the blue box

</div>

</div>

which produces



## Percentage

We use percentages to specify lengths when we want to size on item relative to its parent element.

### Code example

<style>

#blue-box {

width: 250px;

height: 250px;

background-color: blue,

color: white;

}

#red-box {

width: 75%;

height: 50%;

background-color: red,

color: black;

}

</style>

<div id="blue-box">

I am a blue box with a width of 250px and a height

of 250px

<div id="red-box">

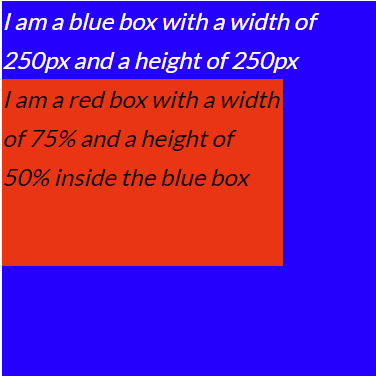
I am a red box with a width of 75% and a height

of 50% of inside the blue box

</div>

</div>

which produces



## Relative to font size

The em length unit is a relative length unit where the base length is calculated from the width of the character M of the current element.

### When to use em

We mainly use em for positioning, spacing and size purposes when we want to allow for reliable scalability within a surrounding context. We will later on see that em comes in extremely handy when writing re-usable components.

### Code example

<style>

#blue-box {

width: 250px;

height: 250px;

background-color: blue,

color: white;

*/\* 1.0em == 16px \*/*

font-size: 16px;

}

#red-box {

background-color: red,

color: black;

*/\* 1.0em == 16px (inherited from parent) \*/*

font-size: 1.5em;

padding: 0.25em;

}

</style>

<div id="blue-box">

I am a blue box with a font size of 16px

<div id="red-box">

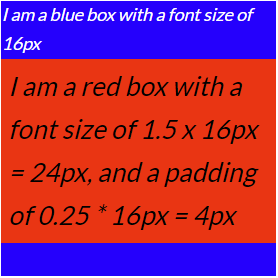
I am a red box with a font size of 1.5 x 16px = 24px,

and a padding of 0.25 \* 16px = 4px

</div>

</div>

which produces



## Relative to root font size

The rem length unit is a relative length unit where the base length is calculated from the width of the character M of the document's root element: the <html> element.

rem units do not have the same nesting complexity as em units since 1.0rem is the same absolute length regardless of context and depth.

### When to use rem

Everywhere else where scalability within surrounding context is not desirable. There are of course still situations where we want percentage lengths and absolute pixel lengths, but we should try to use rem as much as possible.

### Code example

<style>

#blue-box {

width: 250px;

height: 250px;

background-color: blue,

color: white;

font-size: 1.0rem;

}

#red-box {

background-color: red,

color: black;

font-size: 1.5rem;

}

</style>

<div id="blue-box">

I am a blue box with a font size of 1.0rem == 16px

<div id="red-box">

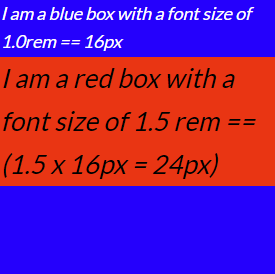
I am a red box with a font size of

1.5 rem == (1.5 x 16px = 24px)

</div>

</div>

which produces



# CSS Properties

## Text color

The CSS property color can be used to set the color of the text.

### Code example

h1 {

color: red;

}

### Accessibility

The contrast ratio between the background color and the foreground text color must be high enough such that it can be read without causing strain to our eyes.

To manually check our contrast ratios:

<https://webaim.org/resources/contrastchecker/>

## Background color

The CSS property background-color can be used to set the background color of an element.

### Code example

body {

color: white;

background-color: black;

}

## Border

The CSS property border can be used to add border to an element

### Common attributes

| **Attribute** | **Description** | **Required** |
| --- | --- | --- |
| style | the style of the border (none, solid, dotted, dashed, double, groove, ridge, inset, outset) | yes (defaults to **none**) |
| width | the width of the border [length unit] | no (defaults to **medium**) |
| color | the color of the border [color] | no (defaults to the **current color** of the element) |

### Code example: shorthand

.all-style {

border: solid;

}

.all-style-color {

border: dotted red;

}

.all-width-style {

border: 1em solid;

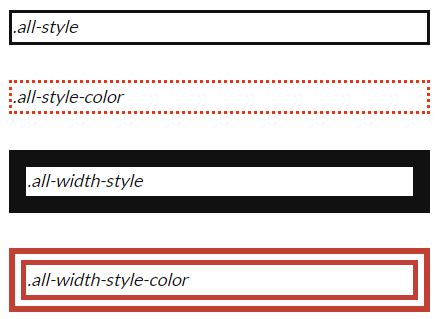
}

.all-width-style-color {

border: 1em double #d32f2f;

}

which produces



### Code example: extended

.rainbow {

*/\* top right bottom left \*/*

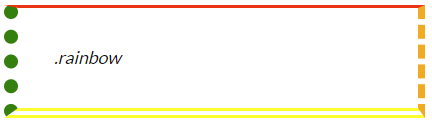
border-width: 0.2em 0.4em 0.6em 0.8em;

border-color: red orange yellow green;

border-style: solid dashed double dotted;

}

which produces



### Code example: border radius

.rounded {

border: 0.5em solid black;

*/\* all equivalent \*/*

*/\* all sides \*/*

border-radius: 1em;

*/\* top/bottom right/left \*/*

border-radius: 1em 1em;

*/\* top right bottom left \*/*

border-radius: 1em 1em 1em 1em;

}

which produces



## Padding

The CSS property padding can be used to create space between the content of an element and its border

### Code example: shorthand

.padded-box {

padding: 1em; */\* equivalent \*/*

padding: 1em 1em; */\* equivalent \*/*

padding: 1em 1em 1em 1em; */\* equivalent \*/*

border: 1px solid black;

}

which produces



### Code example: extended

.padded-box {

padding-top: 1em;

padding-right: 1em;

padding-bottom: 1em;

padding-left: 1em;

border: 1px solid black;

}

This .padded-box class and the previous one accomplish the exact same style

## Margin

The CSS property margin can be used to create space around an element's border and its surrounding elements

### Code example: shorthand

.lonely-box {

margin: 1em; */\* equivalent \*/*

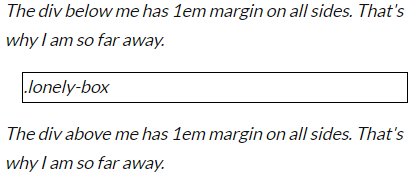
margin: 1em 1em; */\* equivalent \*/*

margin: 1em 1em 1em 1em; */\* equivalent \*/*

border: 1px solid black;

}

which produces



### Code example: extended

.lonely-box {

margin-top: 1em;

margin-right: 1em;

margin-bottom: 1em;

margin-left: 1em;

border: 1px solid black;

}

This .lonely-box class and the previous one accomplish the exact same style.

### Code example: centering

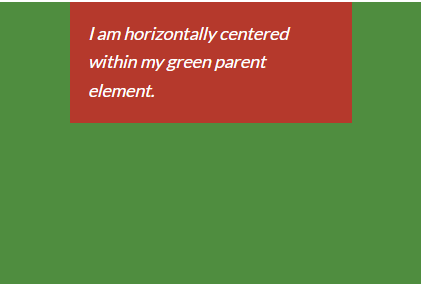
.centered {

width: 512px;

margin: 0 auto;

}

We can center elements horizontally within its parent element by explicitly specifying a width and setting the right and left margins to auto.



## Font size

The CSS property font-size can be used to specify the size of the font.

As briefly mentioned earlier, changing the font size of an element affects other properties using relative length units as well. If, for example, we are using the em unit to specify margin, changing the font size will directly scale the margin of the element proportionally.

### Accessibility

We should aim to always define font sizes using relative length units rather than the absolute px unit. Defining font sizes in px can create problems for some users in certain web browsers that don't allow them to increase the default HTML element font size.

### Code example

h1 {

font-size: 2.5rem;

}

which essentially says that all <h1> elements' font size must be 2.5 times the font size of the root HTML element.

# [CSS Position](https://developer.mozilla.org/en-US/docs/Web/CSS/font-size" \o "font-size" \t "_blank)

[The CSS property position allow us to override the default document flow behavior of an element.](https://developer.mozilla.org/en-US/docs/Web/CSS/font-size" \o "font-size" \t "_blank)

We can use position to create floating elements that float on top of all elements and stay in the same position even if we scroll further down on the same page.

The five main positioning types are:

* **Static**: the default positioning behavior where elements are put into their normal position in the document layout flow
* **Relative**: identical to static positionining except that a relative element can be moved horizontally and vertically using the top, bottom, left and right properties (relative positioning combined with absolute or sticky positioned elements can be very powerful)
* **Absolute**: completely break an element out of the normal document layout flow and position it relative to the containing element which is the nearest positioned parent (useful for menus, popup boxes, etc)
* **Fixed**: identical to absolute positioning except that it does not look for the nearest positioned ancestors, but rather immediately positions relative to the visible part of the viewport (useful for elements that must have the same position regardless of scroll position)
* **Sticky**: positions an element relatively until we reach some specified threshold where it becomes fixed (useful for headings that follow their respective paragraphs)

## Static

Static positioning is the default positioning behavior where elements are put into their normal position in the document layout flow

## Relative

Relative positioning is almost identical to the default static positionining. The only difference is that once the relative element has been positioned in its initial position, we can modify its final position by setting the top, bottom, left and right properties.

## Absolute

An absolutely positioned element will position itself relative to the nearest positioned ancestor. We can then modify its final position by setting the top, bottom, left and right properties. If no ancestor elements are explicitly positioned, the element will be displayed outside of the <html> element and be positioned relative to the viewport.

## Fixed

Fixed positioning is almost identical to absolute positioning except that it does not look for the nearest positioned ancestors, but rather immediately positions relative to the **visible part of the viewport**.

# CSS Flexbox

**justify-content:**

Aligns items horizontally and accepts the following values:

flex-start: Items align to the left side of the container.

flex-end: Items align to the right side of the container.

center: Items align at the center of the container.

space-between: Items display with equal spacing between them.

space-around: Items display with equal spacing around them.

**align-items:**

This CSS property aligns items vertically and accepts the following values:

flex-start: Items align to the top of the container.

flex-end: Items align to the bottom of the container.

center: Items align at the vertical center of the container.

baseline: Items display at the baseline of the container.

stretch: Items are stretched to fit the container.

**align-self:**

This property accepts the same values as align-items and its value for the specific item.

**flex-direction:**

Defines the direction of the main axis.

row: Items are placed the same as the text direction.

row-reverse: Items are placed opposite to the text direction.

column: Items are placed top to bottom.

column-reverse: Items are placed bottom to top.

**Order:**

Specifies the order of the flex item.

**flex-wrap:**

Specifies whether flex items are forced on a single line or can be wrapped on multiple lines.

nowrap: Every item is fit to a single line.

wrap: Items wrap around to additional lines.

wrap-reverse: Items wrap around to additional lines in reverse.

**flex-flow**

Shorthand property for flex-direction and flex-wrap. Takes in same inputs as flex-direction and flex-wrap

**align-content:**

Set how multiple lines are spaced apart from each other. This property takes the following values:

flex-start: Lines are packed at the top of the container.

flex-end: Lines are packed at the bottom of the container.

center: Lines are packed at the vertical center of the container.

space-between: Lines display with equal spacing between them.

space-around: Lines display with equal spacing around them.

stretch: Lines are stretched to fit the container.

**flex-basis:**

The flex-basis CSS property sets the initial main size of a flex item. It sets the size of the content box unless otherwise set with box-sizing.

**flex-grow:**

The flex-grow property specifies how much an element is allowed to grow in the main axis compared to the other sibling elements.

**flex-shrink:**

flex-shrink is the opposite of the flex-grow property and specifies how much an element is allowed to shrink in the main axis compared to the other sibling elements.

# CSS Grid

**grid-column-start:**

Specifies a grid item’s start position within the grid columns.

**grid-column-end**

Specifies a grid item’s end position within the grid columns.

span <integer>: spans the int amount

**grid-column**

Shorthand for grid-column-start and grid-column-end

You can also use span in instead in the end

**grid-row-start, grid-row-end & grid-row**

same concept as the columns one

**grid-area**

Shorthand for grid-row and grid-column

Eks: grid-area: 1 / 2 / span 3 / span 4;

**Order**

You can order class or id to sort

**grid-template-columns & grid-template-rows**

These commands are used to make the grids.

Eks:

grid-template-columns: 20% 20% 20% 20% 20%;

makes 5 columns with 20% space each

you can use px, em and %.

Also fr (fractional unit)

**grid-template**

Shorthand for grid-template-columns & grid-template-rows

Eks:

grid-template: 60% 40%/200px