## CSS

**IMPORTANT! = ALWAYS USE RESET STYLE CODE (RESETS BROWSER TO MAKE SURE IT DOESNT LOOK DIFFERENT IN DIFFERENT BROWSERS)**

* the styling of the content
* doesnt add anything, only modifies what is already added
* multiple stylesheet

## Styling through HTML

<style>

h2 {

color: red;

font-size: 70px; **how big the letters are**

margin: 0px; **the gapes/boxes**

}

</style>

### You should only style in your stylesheet (style.css)

## Styling through CSS

* in html (head) add a link to connect html document with css stylesheet **<link rel=“stylesheet“ href=“style.css“>**

h2 {

color: red;

font-size: 70px; **how big the letters are**

margin: 0px; **the outer gapes/boxes**

}

### Style specific text

(Use mostly the class atribute)

**Div**= every h2 in the div turns green

div h2 {

Color: green;

}

<div>

<h2>this is a text</h2>

<p>this is a text</p>

</div>

**Class and id atribute= class = .title id=#title (id only one per page, cannot use <h2 id=“two things“>This is the title</h2> and cannot have the same name in multiple elements)**

.title {

Color: red;

}

<h2 class=“title“> hey </h2>

## Comment

/\*comment\*/

## Box modeling using margin and padding

### The formula

**(top, right, bottom, left) or (top, right and left, bottom)**

Padding: 10px; = 10px space on all four sites (inside)

Margin: 10px; = 10px space on all four sites (oustide) (it adds on top of each other when next to each other, doesnt sumberge into one)

If two margins are touching the bigger one wins

Border: 10px solid gray; = adds 10px of gray borders

Or writte it as padding-bottom, margin-left: 10px, etc…

## Wrapper

* an important layout method
* makes space from left and right side outside the content

.wrapper {

Margin: 0 auto; (centers the content)

Width: 1000px; (limits the contents width)

}

## Styling properties

Font-family: Arial, „Times New Roman“ ( <= if Arial isn’t available) (the „“ are for making one word) **type of writting**

Font-size: 22px; **size of writting**

Font-style: *italic*; **style of writting**

Font-weight: 100; **the thickness of writting**

Text-align: left; **composition of text (to the left, right, up or down)**

Text-decoration: underline; **The decoration of text**

Text-indent: 30px; **pushes the text out (the first line)**

Text-transform: capitilize; **transforms text=> everything starts with capital lettel**

Color: red;

### : #FF0000 HEX

: rgb(255, 244, 96) **RGB** and :rgba(255, 244, 96, 0,5) **The fifth index is for opacity of color**

Letter-spacing: 10px; **Spacing between letters**

Word-spacing:-10px; **Spacing between words**

Line-height: 10px; **Spacing betwen lines**

## Font Importing

**from website (should use this one)**

Go to website and choose a font => copy <link>?</link> and paste it in index.html on top of the stylesheet (<link rel=“stylesheet“ href=“style.css“> => copy „font-family: #“ and put it in style.css

## Image

* image scales with the width (you dont need to set height)
* by using a background image, you can add text, and other content on top of the image

.img-eye {

Width: 300px;

Height: 200px;

}

### CSS image properties

Background-image: url(eye.jpg)

Background-size: 100% (cover=show only partialy, contain= the whole img)

Background-repeat: no-repeat (dont repeat the img if size 50%)

Background-position: center (center the img) center left (center up and down but pushed to the left)

## Video

Go to youtube or any other page=>copy the emblem/link and paste it into HTML=> for resizing the video, use a wrapper or div around it=> in css use this wrapper

.videowrapper{

Position: relative; (position is used for moving content)

Padding-bottom: 56.25%; (by using % we add height that is scaled with its width)

Padding-top: 25px;

Height: 0px;

}

Use .mp4

.videowrapper iframe{

Position: absolute; (?)

Left: 0px;

Top: 0px;

Right: 0px;

Down: 0px;

Width: 100%

Height: 100%

}

## Responsive website

**In header always use this meta tag for the responsive function => <meta name=“viewport“ content=“width=device-width, initial-scale=1.0“>**

### Phone

* you should always start with phone format, because when somethings goes wrong in your webpage it always looks better **on pc with phone format than a phone with pc format**
* dont use a wrapper

\* {

Margin: 0;

Padding: 0;

}

.wrapper {

Widht: 100%;

}

.Img-banner {

Display: none;

}

.img-info {

Width: 100%

}

.img-info h2 {

Padding: 30px 30px 20px; (top, left and right, bottom)

Font-family: arial;

Font-size: 50px;

Color: #111;

Line-height: 44px

.img-info p {

Padding: 0px 30px 20px;

Font-family: arial;

Font-size: 16px;

Color: #111;

Line-height: 24px;

}

.img-me {

Width: 100%;

}

### Tablet

# Show banner

@media only screen and (min-width: 768px) {

.Img-banner {

Display: block;

Width: 100%;

Padding-bottom: 30px;

}

.wrapper {

Widht: 600px;

Margin: 0 auto;

}

.img-info h2 {

Padding: 20px 0px 0px;

.img-info p {

Padding: 20px 0px 0px;

}

.img-me {

Padding-top: 30px;

}

#### PC

@media only screen and (min-width: 1000px) {

.wrapper {

Widht: 1000px;

}

.img-info {

Width: 50%;

Float: right;

.img-info h2 {

Padding: 20px 0px 0px 30px;

.img-info p {

Padding: 20px 0px 0px 30px;

}

.img-me {

Padding-top: 0px;

Width: 50%;

Float: left;

## Explanation of CSS properties:

**Position**: static (is set by default) = do not interfere with other elements, goes by the flow. All the directional properties are auto (top, left, right, bottom: auto;)

• relative = doesn’t change anything but allows you to use directional properties (top, left, right, bottom) to move around the object. If you give something position: relative and pull it up 10 pixels, it will move, but continue to occupy (in terms of its box) the 10 pixels below it that it previously occupied. **Doesnt affect other elements (can go on top of a element).**

If you give something margin and pull it up 10 pixels, it moves up and brings everything underneath it with it, too.

• absolute = destroys its enviroment. Can be movable but will move from its origin (top right corner of the page). By giving another element (container box) a position: relative; it will transfer to that containers top right corner.

• fixed = stays at its place and when moved, doesn’t interfere. Also destroys its environment. If making a header navigation you have to push down the content with padding so it doesnt interfere with added content.

• sticky = like fixed but is sticked to a certain place and when its hit by the top then it follows. By adding top: 100px; gives its 100px space before scrolling.

**ITS NOT SUPPORTED BY SAFARI =>** position: -webkit-sitcky; + position: sitcky;

(You shouldnt be using POSITION if you dont have a specific purpose in mind, if you want to move the content upwards, left, right, downwards use **margin and padding**)

(**z-index:** is the index which defines which element is on top or bottom

Example: z-index: 5; is goin to be on top of z-index: 3;)

**Display**:

• inline = will stay with other elements on the same line. Ignores height, a certain width, margin (vertical), padding (vertical) values. <span>, <img>, <em>, <strong> elements have inline value as default.

• block = gives the element its own space (line). You can manipulate using height, width, margin (all directions), padding (all directions), float, etc… Elements like <div>,<h1-h6>,<p> have a block value set as default.

• inline-block = stays inline and you can also manipulate

• none = hides the content completely

## Making a website => mmtuts tutorial 18.

* create index.html + use command (!) and organize by using spaces
* create style.css + copy and paste RESET STYLESHEET (add a commnet)
* creat the format for mobile first (the reason is that if something went wrong media responses then it alway appears better when a mobile format is in a desktop than a desktop format in mobile)
* in html create the skeleton (header, main, footer)
* add prefered FONTS (using google fonts)
* create a name and navigation (prepare for desktop version)
* in css set background-color for header and body
* set header (width and height)
* customize the header-brand
* customize the navigation and .header-cases
* navigation/header done
* add and customize photo (background)
* add and customize text on the background photo + centerize
* create link boxes (Cases, Porftfolio, Julian Xu, Instagram, About, Contact) + also prepare them for desktop version
* create footer (Home, Cases, Portfolio, About me, Contact + prepare for desktop version)
* create the desktop version (go to the header section)
* @media only screen and (min-width: 1000px) {}
* change the nav, banner, banner, text, box links, footer

## Form

* forms works with some kind of input (infromation)
* its used for login system, sending emails, etc..
* with HTML you can create the form but you cant work with the information without another programing language (PHP, Javascript, Python)

### In HTML (All the examples)

<**form** action=“contactform.php“ method=““>

<**input** type=“text“ name=“email“ placeholder=“Write your e-mail… “>

<**input** type=“text“ name=“email“ value=“Write your e-mail… “>

Male: <**input** type=“radio“ name=“gender“ value=“Male“ checked>

Female: <**input** type=“radio“ name=“gender“ value=“Female“>

<**input** type=“password“ name=“pwd“ placeholder=“Password“>

<**textarea** name=“message“ placeholder=“Write your message“><**/textarea**> (the type is text by default)

<**select** name=“car“>

<option value=“none“>None</option>

<option value=“car1“>Ford</option>

<option value=“car2“>Mercedes</option>

<option value=“car3“>Opel</option>

</select>

<**input** type=“submit“ name=“submit“>

<**button** type=“submit“ name=“submit“>Submit<**/button**>

</form>

### Explanation (elements, atributes, functions)

### Elements/tags

* form = the main element which contains the input, textarea, select, etc…
* input = the information (data = e-mail addres, username, password, tick boxes, sumbit function)  
  -textarea = the text (e-mail)
* select = dropdown option (choosable)
* button = a better way to make a submit button (it tells the browser that it is a button)

### Atributes

* type (input) = what type of input (**text**, **radio** = tick box, **submit** = to submit the info, **password** = for passwords)
* name (all) = name for the input (for sending emails => email)
* placeholder (input, textarea) = the light grey text thats shown in the box and fades when writting
* value (input, textarea, select = default text written in the box (when typping, you have to delete it / also could be used as tick box value)
* checked (input/radio) = automatically checked a certain tick box as default
* action (form) = where to send the submit (info)
* aria-label => for labeling the content
* method (form) = which method to use for sending the info. **Get** (3000 words limit) = not for sensitive data like passwords because you will be able to see the data in the UL, **Post** (no words limit) = doesnt show the data in the UL)
* for = what is the label for (name)

### Styling in CSS

Form {

width: 300px; (width limited to 300 pixels, makes them go onto another line)

}

# Input {

Width: 100%; (allows the fill boxes to go 300px)

}

Textarea { (styling of the textarea/email)

Resize: vertical; (can resize vertically only)

Max-height: 100px; (sets a height limit)

}

Input, textarea { (can style both at the same time)

Outline: none; (cancels the blue outline when typing)

Border: 1px solid #ddd; (customizes the border)

}

Input:focus, textarea:focus { (changes something when its focused on)

Box-shadow: 0(x = horizontal) 0(y = vertical) 4(blur value) rgba(255, 0, 0, 0.5); (give the box a shadow + color – light red)

Border: 1px solid rgb(255, 0, 0, 0.5); (if you want it to be fully red)

}

::placeholder { (**styles all elements with placeholders atribute**, this tag styles everything => if only input placeholders then use **Input::placeholder {}** )

Color: red;

Opacity: 1;

}

#### IMPORTANT!

* some browsers need their own individual tags for it to work

:-ms-input-placeholder { (styles Internet Explorer)

Color: red;

Opacity: 1;

}

::-ms-input-placeholder { (styles Micosoft Edge)

Color: red;

Opacity: 1;

}

## Table

* we mostly use tables to show data or informations on a website

### In HTML (all the examples)

<table>

<caption>Names list</caption>

<tr>

<th>Firt name: </th>

<th>Last name: </th>

<th>Gender: </th>

</tr>

<tr>

<td>Julian</td>

<td>Xu</td>

<td>Male</td>

</tr>

</table>

### Tag explanation

Table = the main structure

Caption = is the header tag for graphs and tables

Tr = table row

Th = table header (the header information), its centered by default

Td = table data (the result data information)

### Inside CSS styling

Table {

Width: 400px;

Border-spacing: 40px; (if theres more borders inside one border, it creates distance between all of them)

}

# Th, caption {

Text-align: left; (pushes the centered header to left)

}

# Table, th, td {

Border: 1px solid #000; (gives the table borders)

Border-collapse: collapse; (optional) (merges the borders into one)

}

# Th, td {

Padding: 10px; (give space from inside, so that its not so cramped together)

}

Tr:nth-child(even) { (every even row is white)

Background-color: #fff;

}

Tr:nth-child(odd) { (every even row is grey)

Background-color: #ddd;

}

#### Better HTML coding

* limit using <div> mistake: just so we can wrap an element to give it a class atribute, manipulate the structure so we can use CSS, markup content

Why: because we give it structure wich we shouldnt want, makes the document more complex, nonsymentic element = our machine doesnt recognize what its content is even with classes or ids => Hurts SEO, makes the files larger and slower to read with the browser

* dry coding mistake: using dry code (repeating code)

Why: because it will be much harder and more confusing in mantaining

#### Uploading website to the internet

**What do you need:** Website (finished website)

Domain (the name of the site = [www.seznam.cz](http://www.seznam.cz))

* you need to buy only one time

Server (the place you upload the website)

* you need to pay every month

### Websites which you will use:

* Hostgator.com (hosting site)
* Filezilla-project.org (so that you dont have to login on hostgator every time)

### Before you upload your website (Check list)

* make a proof check (spelling mistakes, hyperlinks, things I missed)
* meta tags
* website works in all browsers
* validate your website: (validator.w3.org – HTML)

(jigsaw.w3.org/css-validator/ – CSS)

* keep a copy of your website offline

#### Sitemap

Sitemap is for the user to go to your website a find your content easier

Helps with search engine SEO

There are many ways to create them (footer anchor tags/href, creating and putting a seperate html folder where all the links are, to the navigation or else where => **its the best way**, helps google and the users to find the wanted content), creating a XML page = not used for **users** but for **SEARCH ENGINES**)

You should always create HTML (for users) and XML (for SEO)

Should always be included, mainly for the website which are huge in content, websites that you have to click links to get to another link,

Xml-sitemaps.com (for creating a xml sitemap file)

* type in your URL website
* download file
* insert it in the main folder (index.html) – preferably using Filezilla
* do the following steps with google console to register into google search engine (from xml-sitemaps.com)

**404 page**

* a external page which you will be taken to if the the user goes to a page which does not exist (URL doesnt exist)
* create a html file for the 404 error page (usually you leave the header and footer and just add a text in the body) and name it **not-found-page.html**
* then create a new folder and name it **.htaccess** => its an configuration file for your server (**file of codes which will run before the server loads**) as long as you have a website on the internet you can use a .htaccess file (if the users somehow gets on a new page it this htaccess file tranfers him to the **not-found-page.html** file)

### Inside the .htaccess file

ErrorDocument 404 /not-found-page.html

#### Page file extension removal (?)

* uses the .htaccess file (if theres an error in the htaccess file, because its a server configuration file it will shut down the whole website untill the error is fixed)
* delete all the .html extensions in the hyperlinks inside index.html (it will work on those files without the extensions)
* create a .htaccess file (the file of codes which will run before the server loads) – put the 404 error code underneath everything

### Insite the .htaccess file

# mod\_rewrite starts here (# = comment)

RewriteEngine on (the code for rewritting)

(now we have to add rules and condition for the rewrite => rules = is the thing that actually changes the content, conditions = have to be true in order for the rules to run)

# does not apply to existing directories, meaning that if the folder exists on the server then dont change anything and dont run the rule! (If theres a folder named the same way as the file)

RewriteCond %{REQUESTED\_FILENAME} !-d

# if the file doesnt exist then it doesnt run the rule! (check for the file in the directiories with .html extension)

RewriteCond %{REQUEST\_FILENAME} /.html -f

# here we actually show the page that has the .html extension

RewriteRule ^(.\*)$ $1.html [NC,L]

(we went ahead and grabbed the entire URL if the CONDITIONS are true. If they are true then we still want to show the URL as it is, but the content on the page is going to be shown from the same name file.html)

([NC,L] are special conditions regarding the rule. NC = Capital letters wont disturb the URL, L = the conditions only work for the **RewriteRule ^(.\*)$ $1.html [NC,L]**)

## Flexbox

* an more complex method to manipulate layouts => we used float in the past (you had to set a content to and certain width and se float => there could be problems with float => you also had to use a lot of math and calculation with the width and margin/padding)
* responsive and mobile friendly
* you can change the order of contents without changing the HTML
* direction-agnostic => supports both block and inline elements

### Container

* use **display: flex;** on the wanted element (we **tell the element** that we want to **rearrange the content in a certain way** => we arrange the **children of the container**)
* even **flex-items** themselves can become **flex-containers** for their children elements

**flex-wrap**: nowrap; (is set by default => text doesnt go onto another line)

* wrap; (goes onto another line)
* wrap-reverse (jumps onto the next upward line)

**flex-direction**: row (from left- to right)

* row-reverse (right-left)
* column (top-bottom)
* column-reverse (bottom-top)

**flex-flow**: row wrap; (combination of flex-wrap and flex-direction)

**justify-content**: (styles horizontaly)

* flex-start (by default => the content gets pushed to the left)
* flex-end (the content gets pushed to the right side)
* center (centers the content)
* space-around (give space between each sentence)
* space-between (give space between each sentence and pushes to the sides)

**align-items**: (vertical moving of individual items) (**also deletes the same height function**)

* flex-start (all the content is pushed to the top)
* flex-end (all the content is pushed to the bottom)
* center (verticaly centered)
* stretch (set by default, strechtes the content/gives it height)
* baseline (centers all content with all kind of height)
* **align-content**: (styling of the whole row/**justify-conent but styles verticaly**)
* center (pushes all content together and centers it)
* space-around (all the content will get a even **top and bottom** margin)
* space-between (all the content will not get top margin and will get pushed to the top and bottom sides)

### Item

-Are the **content** **of the flex-containers** which you want to **manipulate** with

-you can use grow, shrink, basis, order **flex properties** for the **items**

* the order of how we want to show the items)

.Item {

Order: 1;

}

.Item2 {

Order: 2; (no matter the position it will always be after order 1)

}

Flex-grow: 2; (1 is set by default => gives more horizontal line space)

Flex-shrink: 2; (exact oposite)

Flex-basis: 60%; (manipulates the width)

or

Flex: 0(grow) 1(shrink) 100px(width)

## Flexbox in practice

### Example 1: (symetrical width and distribution)

* if we want to make the borders reach to the end
* the problem is that the borders are not symetrical due to the lenght of the labels (Name, Favorite Color)
* we would have to set two different width: ?;
* by using flex: 1; for the input elements they both will become the same length (ignores the length of the **label**)
* but if used flex: 1; also in the **label**, firstly it will distribute the width between the label and input (half and half), and the borders will become identical

Name: |border | ----------------------------------------------------------------------------

Favorite Color: |border| ------------------------------------------------------------------

#### HTML

<div class=“container“>

<form action=““>

<div class=“form-row“>

<label for=“name“>Name:</label>

<input type=“text“ id=“name“>

</div>

<div class=“form-row“>

<label for=“favColor“>Favorite Color:</label>

<input type=“text“ id=“name“>

</div>

</form>

</div>

#### CSS

.form-row {

Padding: 10px 0; (spaces between the forms)

Display: flex; (activates flexbox)

.form-row label { (Name)

Padding-right: 10px; (gives space between name and box)

}

.form-row input { (box)

Flex: 1;

}

### Example 2: (Width and order)

* we have 3 paragraphs, one of them has a lot of more text inside
* by adding the **display: flex;** to the parent element it will automatically change the layout (next to each other, the paragraph with more text will take more width by default)
* if we want the **3 paragraphs to be same width**, add to all elements the property **flex: 1;** (all the elements have the same width value)
* if we want the **main paragraph** (more text) to **be twice as big** as the other ones in width, add to the main element the property value flex: **2; (2x bigger), flex: 3; (3x bigger), etc…**
* the main paragraph is on the left side, the second is in the middle and the last is in on the right side. If we want to put the main paragraph in the middle and the second on the left and the last on the right side, we give the **main paragraph** a property **order: 2; Second one order: 1; Last one order: 3;**

**(you could do it in HTML where you would put the main paragragh underneath the second one, but because of SEO reasons it could be bad for your website)**

### Example 3: (Adding margin)

* we have 3 boxes (divs) with paragraphs inside and we want to put them next to each other so that they have the same width, but we want to add margin spacing between the boxes.
* We give the parent element **display: flex**; and they turn next to each other
* The 3 boxes will have the **same height** no matter the amount of the text, unlike **float.** This function is set by default, but if you dont want this function add **align-items: flex-start; (actually any: flex-end, center, baseline)** and the boxes will have a dependent **height based on the content inside**
* The boxes will begin at top due to **align-items: flex-start;** for them to start at the bottom use **flex-end,** for centering use **center,** the default value is **stretch**

**(you could give them a margin-right: 40px; but there would be left an akward space on the last box. For the akward space you could actually give the parent element a pseudo element :nth-child(3) or :last-child {margin-right: 0;} but using flexbox is easier)**

* Solution is that in flexbox we dont have to set width (**that doesnt mean we cant use a width**), so by giving the boxes (**using class**) a **width: 30%;** it will take only 90% of the complete width and leave a 10% space as margin. **The flexbox way of setting a width is flex-basis: 30%; (Use this)**

**(whenever you assign a padding and a width property together in a element, you should use box-sizing: border-box; so that our math stays as simple as possible)**

* now all the boxes will be pushed together onto the left side, with 10% of space on the right side. To distribute it equally (make margin spaces) use **justify-content: space-between; (other options: flex-start ,flex-end, center, etc…)**

### Example 4: (Line pushing)

* we have 5 items (boxes with a number text inside) which have a strictly set parameters of **width: 150px**; and a **height: 100px;** and aftarwards they are well distributed (**width**) using flexbox (**justify-content: space-around;**)
* The problem is that flexbox has sometimes problems recognizing when to push the content onto another line
* To solve the problem use the property flex-wrap: wrap; (other options no-wrap (by default), flex-start, flex-end)

### Example 5: (Text centering inside a container)

* we have a box in which we a have a piece of a text, we want to perfectly center the text (verticaly and horizontaly). Use **display: flex;** and for the element use **margin: auto;**

### Example 6: (

* we have a box with a paragraph and two layout boxes
* the layout that we want is that the main paragraph is on the right side and takes half of the width and the two boxes on the left with sharing width and height
* depending on the main paragraph (how much text is inside) it should change the height. The two paragraphs should also be adapting to the height while still sharing the same amount
* we give the parent element a display: flex; and then to the main paragraph a flex: 1; then we use display: flex; in his child element (flexbox in a flexbox) and we give his children element (child of the child) a flex: 1;
* the only problem now is that the two boxes are horizontaly next to each other, so we use flex-direction: column; (other options: row (by default), reverse-row, column, reverse-column)

#### CSS Variables

* variable = type of container that we can sign a value to (**WORKS ONLY FOR CHILD ELEMENTS)** => example: We have a color which we use for all the text inside our webpage. After finishing the website we decide to change the color of the text. We use one variable to change all the codes
* we use variables in a lot of other programming languages like PHP, Javascript
* very useful in CSS styling

### How to use it

:root { (pseudo element = a way for us to style specific part of an element)

* text-color: #ccc; (the variable + any name)

}

.section-one p {

Color: var(--text-color);

}

**.**section-two p {

Color: var(--text-color);

}

(If you want to change the color/styling just go to the **:root** and change the value)

### Other way of using it

.section-one {

* text-size: 14px;

}

.section-one p {

Font-size: var(--text-size);

}

#### Pseudo Elements

* a way for us to style a certain part of an element
* pseudo classes => changes the state of the element (when we hover on it, click)
* pseudo element => changes the part of the element, we select a part of it and then change it

### Pseudo classes

* If you put in front of the class a tag, it will only work for that tag. (example: h1:hover => if you hover to any h1 tag, then it will activate something)

:root = has the highest element priority (higher than <html>), is used for styling specific things in your website

:hover = activates something when you hover onto them

:active = activates while holding the click button

:visited = activates when visited or manipulated before

.list-example-1 p:last-child { (the last <p> element will be blue)

Color: blue;

}

.list-examples-2 p:nth-child(2n+0) { (2=second,even or odd, 2n+0=starts at 0 and applies at every other element)

Color:pink;

}

### Pseudo Element

* we use :: for pseudo elements
* styles a specific part of the element

.element-1::first-line { (styles the first line)

Background-color: yellow;

Color: red;

}

(can use both of them at the same time)

a

.element-3::selection { (when highlighted)

Background-color: green;

Color: blue;

}

**Animation and keyframes**

body {

background: #333;

}

.box {

background: white;

width: 200px;

height: 200px;

position: relative;

animation-name: myanimation;

animation-duration: 4s; (duration of the whole animation)

animation-iteration-count: 1; //other options = 2,3,4… or infinite (how many times should the animation repeat)

animation-fill-mode: forwards; (stops with the last keyframe style)

/\* animation-delay: 2s; \*/

/\* animation-direction: alternate; (goes one time normally and them goes the reverse way => repeat) \*/ //reverse = backwards (the direction of the animation)

/\* animation-timing-function: ease-in-out; \*/

}

/\* @keyframes myanimation {

0% {background-color: white; left: 0px; top: 0px; border-radius: 0 0 0 0;}

25% {background-color: red; left: 300px; top: 0px; border-radius: 50% 0 0 0;}

50% {background-color: green; left: 300px; top: 300px; border-radius: 50% 50% 0 0}

75% {background-color: blue; left: 0px; top: 300px; border-radius: 50% 50% 50% 0;}

100% {background-color: white; left: 0px; top: 0px; border-radius: 50% 50% 50% 50%;}

} \*/

@keyframes myanimation {

0% {background-color: white; border-radius: 0 0 0 0;}

25% {background-color: red; border-radius: 50% 0 0 0;}

50% {background-color: green; border-radius: 50% 50% 0 0}

75% {background-color: blue; border-radius: 50% 50% 50% 0;}

100% {background-color: white; border-radius: 50% 50% 50% 50%;}

}

#### Transitions

* makes the smooth, cool, transition efects
* in the past before css3 we could only use javascript for transitions
* can be used with hover and other trigger effects
* Examples: you can create a small box which when hovered on will pop up a menu or navigation box, image and logo transitions, animations, video transitions, etc…)

### In CSS

Transition-property: all; (i want all of the picture to have the transition)

Transition-duration: 2s; (trasition will take 2 seconds)

Transition-timing-funtion: ease-in-out; (ease in = slowly starts and then builds up, ease out, ease in out)

Transition-delay: 4s; (the transition will start after 4 sec when moved on with the mouse cursor)

**(Not all browsers support the transition effect, mainly the older ones. We have to use something called PREFIX for it to work)**

### Types of prefixes

* webkit- (Chrome)
* moz- (Firefox)
* o- (Opera)
* webkit-Transition-property: all;
* webkit-Transition-duration: 2s;
* webkit-Transition-timing-funtion: ease-in-out;
* webkit-Transition-delay: 4s;

### Easier way

Transition: all 2s ease-in 1s; (properties, duration, timing function, delay)

Transition: all 2s ease-in 1s;

* webkit-Transition: all 2s ease-in 1s;
* moz-Transition: all 2s ease-in 1s;
* o-Transition: all 2s ease-in 1s;