TRAINING DRAGON

WEB DESIGN

TRAINER: EMILIANO

CSS INTRO, I

- why CSS?
- CSS structure
- where do we write CSS
- basic selectors
- styling text
- styling block elements
- the box model
- horizontally centering a block element

CSS INTRO, II

- floating and clearing content
- layout with flexbox
- css positioning with position
- z-index
- CSS3 intro I - border-radius
- CSS3 intro II - box shadow
- CSS3 intro III - nth-child

WHY CSS?

- more powerful and flexible than HTML presentation
- can control multiple HTML pages
- useful to avoid duplication
- promotes easier maintenance
- change an entire website with few touches

CSS STRUCTURE

- a css document is a collection of **rulesets**
- a ruleset is a collection of **rules**
- rules are made out of **properties** and **values**
- **properties** are presentational features, eg: color, size, position
- **values** are settings for properties

```css
/* ruleset: */
selector{
    /* rule */
    property: value;
    color: red;
}
```

WHERE DO WE WRITE CSS

there are different places where we can insert CSS code:

- **inline** css using **style attribute**
- **internal** css using **style tag**
- **external** css including external stylesheets

```
<p style="color:red;">this is a paragraph</p>
```

- **DEMO:** INLINE CSS
- listing properties and values inside a **style** attribute
- the most specific CSS, highly discouraged

```
<head>
    <style>
        h1{
            color: red;
        }
    </style>
</head>
```

- **DEMO:** INTERNAL CSS
- writing rulesets inside a **style** tag in the head of the document
- will be overridden by inline CSS, still discouraged

```html
<head>
    <link rel="stylesheet" href="path/to/stylesheet.css" media="screen"/>
</head>
```

- **DEMO:** invoking an EXTERNAL CSS stylesheet, best practice and highly encouraged
- **rel** specifies the relationship with the file we are invoking
- **href** provides path and filename of the stylesheet document
- **media** sets media type we want this stylesheet to be applied for

BASIC SELECTORS

IN THIS COURSE WE WILL SEE:

- **tagname** selectors
- **ID** selectors
- **CLASS** selectors
- **descendant** selectors

TAG NAME SELECTORS

```
/*** TAG NAME SELECTORS ***/
/* selecting ALL paragraphs */
p{
    text-align:justify;
}

/* selecting ALL images */
img{
    border:1px solid
}
```

- **DEMO:**using **tag names** to select all elements with that tag name
- **p** selects all paragraphs, **img** selects all images, **div** selects all divs etc..

ID SELECTORS

```html
<!-- in HTML IDs are attributes -->
<p id="aParagraph">some text here</p>
```

```css
/* in CSS, we use #idName notation */
/* to select elements via their id name */
#aParagraph{
    font-size: 1em;
}
```

- **DEMO:** using an **#ID** selector to select a specific element
- **IDs** are the most specific selectors
- we can only use an ID name **once** per page
- we can only give one ID to an element

CLASS SELECTORS

```html
<!-- in HTML CLASSes are attributes -->
<p class="classA classB">some text here</p>
<div class="classB">some more text here</div>
```

```css
/* in CSS, we use .className notation */
/* to select elements via their class name */
.classA{ font-size: 1em; }
.classB{ background-color:#666; }
```

- **DEMO:**using **class names** to select all elements with that class name
- we use classes to group and select multiple elements
- the same class name can be used as many times as we need
- we can apply multiple classes to same element
- we select classes using .className
- classes are less specific than IDs

DESCENDANT SELECTORS

```
<!-- leveraging hierarchy and element positions -->
<p class="classA">
    <img src="pic.jpg" width="100" height="100" alt="pic"/>
</p>
<div class="classB">
    <p>
        <img src="pic2.jpg" width="100" height="100" alt="pic"/>
    </p>
</div>
```

```
/**  SYNTAX: A B **/
/** selecting element(s) B nested inside element(s) A **/
p.classA img{ ... } /* selecting images inside ps with  .classA */
div img{ ... } /* selecting images nested inside divs */
```

- **DEMO:** using **descendant selectors** to select elements according to their position

STYLING TEXT, I

```css
/* sets the size of the text */
p{
    font-size: 16px | 15pt | 120% | 1.2em | 1rem ;
}
```

- **px** is a pixel, equal to one dot (fixed)
- **pt** is a point, traditionally used in print - 1/72 of an inch (fixed)
- **em** is a unit relative to the font-size of the element direct parent (scalable)
- **rem** is a unit relative to the root (html) font size (scalable)
- **%** is a unit relative to ascendants/ context (scalable)

STYLING TEXT, II

```
/* text alignment */
p{ text-align: left | right | center | justify; }
/* text decoration */
p{ text-decoration: underlined | overline | line-trough | none; }
/* making text bold */
p{ font-weight: [number] | normal | bold | bolder | lighter;}
/* making text italic */
p{ font-style: italic | normal | oblique; }
/* transforming text */
p{ text-transform: uppercase | lowercase | capitalize; }
```

STYLING TEXT, III

```
/* colour of the text */
p{ color: red | #f00 | #ff0000 | rgb(255,0,0); }
/* colour behind the text */
p{ background-color: red | #f00 | #ff0000 | rgb(255,0,0); }
```

STYLING BLOCK ELEMENTS, I

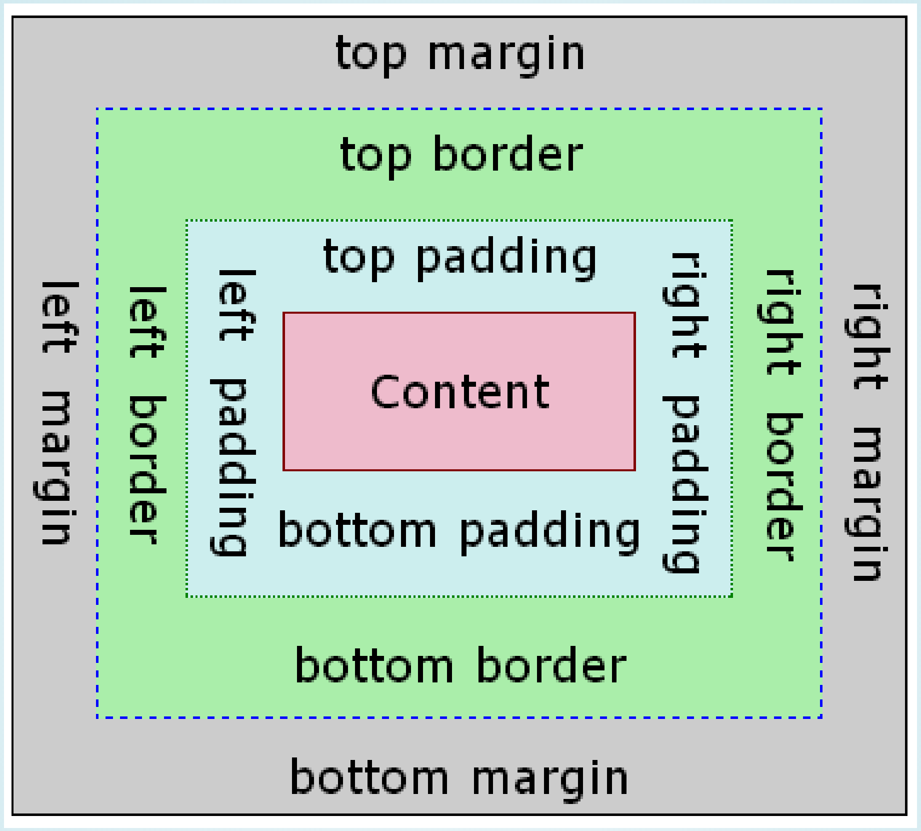
```
/* setting horizontal size */
div{
    width: 500px | 50% | 20em | 20 rem | 10vw | 10vh; /* default: 100% */
    max-width: 50px | 5% | 2em | 2 rem | 5vw | 5vh; /* default: 100% */
}

/* setting vertical size */
div{
    height: 500px | 50% | 20em | 20 rem | 10vw | 10vh; /* default: 0 */
    max-height: 500px | 50% | 20em | 20 rem | 10vw | 10vh; /* default: 0 */
}
```

- **1vw** is 1% of the view width
- **1vh** is 1% of the view width
- **1vmin** 1vw / vh, whichever is smaller
- **1vmax** 1vw / vh, whichever is larger

STYLING BLOCK ELEMENTS, II

```css
/* manipulating backgrounds */
div{
    background-color: red | #f00 | #ff0000 | rgb(255, 0, 0);
    background-image: url("pic.jpg");
    background-attachment: fixed | local | scroll;
    background-repeat: repeat | no-repeat | repeat-x | repeat-y;
    background-position: center | top | left | bottom | right;
}
```

THE BOX MODEL, I



THE BOX MODEL, II

```
/* box model */
div{
    width:400px;
    height:200px;
    padding: 20px;
    /* border: size style color; */
    border:10px solid green;
    margin: 50px;
}
```

- **width and height** define the size of the content
- **padding** is the spacing within the element, in between content and border [only positive values]
- **border** is a line around the element
- **margin** is the spacing outside the element, in between its border and other elements [positive or negative values]

THE BOX MODEL, III

MARGIN AND PADDING SHORTCUTS

```css
/* 4 different properties can be set */
div{
    margin-top:10px;
    margin-right:15px;
    margin-bottom:20px;
    margin-left:10px;
}
```

```css
/* or we can use shortcuts */
div{
    /* 1 parameter  => A: all sides */
    margin:A;
    /* 2 parameters => A: top&bottom, B: left&right */
    margin:A B;
    /* 3 parameters => A: top, B: left&right, C: bottom  */
    margin:A B C;
    /* 4 parameter => top, right, bottom, left */
    margin:A B C D;
}
```

- same rules will apply for padding shortcuts

THE BOX MODEL, IV

BOX-SIZING PROPERTY

```
/* box-sizing */
div{
    box-sizing: content-box | border-box
}
```

- **content-box** is the default value, width and height define the size of the **content**, no padding, no border
- **border-box** has to be set, width and height define the **whole size of the element**, including padding and border

HORIZONTALLY CENTERING A BLOCK ELEMENT

- set a width
- set margin left and right as auto

```css
.container{
    width: 90%;
    margin:0 auto;
}
```

FLOATING AND CLEARING CONTENT, I

- used to push images to one side and having text wrapping around them
- used to position block elements on the same line
- used to generate columns, layouts, navigations
- floating elements will be outside of the document flow
- floating multiple elements will generate issues
- float is blocked with **clear** property
- replaced by the more flexible **flexbox**

FLOATING AND CLEARING CONTENT, II

```
<!-- a navigation, vertical arrangement as default -->
<nav>
    <ul>
        <li><a href="home.html">home</a></li>
        <li><a href="blog.html">blog</a></li>
        <li><a href="contacts.html">contacts</a></li>
    </ul>
    <div class="clearFix"></div>
</nav>
```

```
/* floating nav items - float: left | right */
nav ul a{ float: left; }
/* clearing the float to prevent issues
    clear: left | right | both; */
.clearFix{ clear: both; }
```

- **DEMO:** floating nav items and preventing issues with .clearFix

LAYOUT WITH FLEXBOX

- much more effective and flexible than float
- controls rows, columns, positioning, spacing, elements order etc
- latest version is ready to be used
- fixes a number of old issues like holy grail, sticky footer etc
- makes creation of layouts much easier

FLEXBOX I, A NAVIGATION

```
<nav>
    <ul>
        <li><a href="#">home</a></li>
        <li><a href="#">gallery</a></li>
        <li><a href="#">about us</a></li>
        <li><a href="#">news</a></li>
        <li><a href="#">blog</a></li>
        <li><a href="#">contacts</a></li>
    </ul>
</nav>
```

```
nav ul{
    display: -webkit-box;
    display: -moz-box;
    display: -ms-flexbox;
    display: -webkit-flex;
    display: flex;
    flex-direction: row | column;
    justify-content: flex-start | flex-end | center
        | space-between | space-around;
    background-color: blue;
}

nav a{
    display: block;
    background-color: blue;
    color: #fff;
}
```

FLEXBOX II, A LIST OF IMAGES

```html
<div class="galleryHolder">
    <img src="pic1.jpg" width="100" height="100" alt="thumb">
    <img src="pic2.jpg" width="100" height="100" alt="thumb">
    <img src="pic3.jpg" width="100" height="100" alt="thumb">
    <img src="pic4.jpg" width="100" height="100" alt="thumb">
</div><!--!/ galleryHolder-->
```

```css
.galleryHolder{
    width: 500px;
    height:500px;
    display: -webkit-box;
    display: -moz-box;
    display: -ms-flexbox;
    display: -webkit-flex;
    display: flex;
    flex-direction: row | row-reverse |
        column | column-reverse;
    flex-wrap: wrap | no-wrap | wrap-reverse;
    /* horizontal arrangement */
    justify-content: space-between | space-around | center
        | flex-end | flex-start;
    /* vertical arrangement */
    align-content: space-between;
}
```

- **DEMO:** arranging images of a gallery using flexbox

FLEXBOX III, READINGS

- [css-tricks]
- [a cheatsheet]
- [MDN]
- [Solved by Flexbox]

CSS POSITIONING WITH POSITION

- advanced positioning system
- do not use to build a complete layout
- use **position** property as an "extreme measure"
- 5 values: **static**, **relative**, **absolute**, **fixed** and **sticky**
- 4 properties can be used as coordinates: **top**, **bottom**, **left** and **right**
- both positive and negative values can be used for coordinates

POSITION, I - STATIC

- all elements are static as a default
- normal behaviour of elements
- top, bottom, left, right and z-index property will not apply

POSITION, II - RELATIVE

- the element is still in the document flow
- the element will move in reference to its original position
- moving the element will not affect siblings or ancestor
- the original position will be preserved and kept empty, (a blank space will appear)

POSITION, III - ABSOLUTE

- the element is not in the document flow anymore
- the element will move in reference to its first non-static ancestor
- the original position will be lost, no blank space, other elements will replace it
- if there's no positioned ancestor, the reference is HTML
- we can make an ancestor relative to change the reference element

POSITION, IV - FIXED

- outside of the document flow
- original position will be lost
- the reference is the viewport
- fixed elements will not scroll with the rest of the page

POSITION, V - STICKY

- still not universally supported (but getting better)
- see caniuse.com for support tables
- the element starts with a relative position
- after passing a certain scrolling position, the element turns into fixed

```
#sticky{
    background-color: red;
    color:white;
    position: sticky;
    top:10px;
}
```

Z-INDEX

- only works with poritioned (non static) elements
- sets the z-order of elements
- elements with larger z-index cover elements with lower z-index
- takes integers as values
- animatable
- default value is AUTO

```css
.one{
    position:relative;
    z-index:10; /* will be covered by .two */
}

.two{
    position:relative;
    z-index:20; /* will appear on top of .one */
}
```

- **DEMO:** stacking an element on top of another

CSS3 INTRO I BORDER-RADIUS

- generates rounded corners via css
- belongs to CSS3
- used to need vendor prefixes, does not anymore
- can also generate circles, ellipses and similar shapes
- accepts one or four values

```
.box{
    /**
    border-radius: 4sides;
    border-radius:
        top-left-corner
        top-right-corner
        bottom-right-corner
        botton-left-corner
    ; **/
}
```

```
.circle{
    width:200px;
    height:200px;
    border-radius:50%;
}

.ellipse{
    width:400px;
    height:200px;
    border-radius:50%;
}
```

CSS3 INTRO II
BOX SHADOW

- generates shadows with css only
- does not need vendor prefixes anymore
- most basic syntax with 3 values and a color

```
.box{
    /* x-offset | y-offset | blur | colour */
    box-shadow: 3px 3px 5px red;
}
```

- **DEMO:** shadows around a box

CSS3 INTRO III NTH-CHILD

- selects the nth-child
- accepts an index or a multiple

```
nav li:nth-child(3){
    /* select the third list item in the navigation*/
}

.subset img:nth-child(3n){
    /* selects third, sixth, ninth, etc image of the subset */
}
```

- **DEMO:** selecting elements via their indexes