# CSS

## Inheritance

Some properties are inherited by default no matter how deep nested elements

<div id="box">

    <section>

        <section>This is some text inside the box.<section>

    </section>

</div>

#box {

      background-color: brown;

      padding: 100px;

}

“This is some text…” will also have background color brown. The inheritance works on those properties for all selectors: tags, classes, id.

Some properties like `padding` are not inherited, to inherit them it should be set to `inherit`

#sec {

  padding: inherit;

}

But it inherits from close parent, not from grandparents:

        <div id="box">

            <section>

                <section id="sec" >This is some text inside the box.<section>

            </section>

        </div>

#box {

      background-color: brown;

      padding: 100px;

}

#sec {

  padding: inherit;

}

The padding of #sec will be 0px because the padding of parent <section> is default 0px.

## Select needed element

To select all descendants (children, grandchildren etc.) write

.class \* {

  padding: 10px;

}

To select only first-level children:

.class > \* {

  padding: 10px;

}

Select first elements B directly followed by A elements:

A + B {

  padding: 10px

}

Select all elements B followed by A element (on the same level of nesting):

A ~ B {

  padding: 10px

}

Select first child(or grandchild, etc.) of element:

div p:first-child{

  padding: 10px;

}

Select every element that is the 8th child of another element:

:nth-child(8){

  padding: 10px;

}

## FlexBox

In flex-box the `flex` property works bad:



It’s ignore 200px size.

So I prefer to use related parameters:

#div1{

  background-color: coral;

  width: 100%;

  flex-shrink: 1;

}

#div2{

  background-color: lightblue;

  width: 200px;

  flex-shrink: 0;

}

#div3{

  background-color: lightgreen;

  width: 200px;

  flex-shrink: 0;

}

The div1 take all the rest place and only div1 is shrink.

Don’t use flex-grow and width.

#div1{

  background-color: coral;

  min-width: 100px;

  width: 50%;

}

#div2{

  background-color: lightblue;

  min-width: 100px;

  width: 100%;

}

#div3{

  background-color: lightgreen;

  width: 200px;

  flex-shrink: 0;

}

div2 is twice wider than div1, but if shrink – 100px both:

## Width Height 100%

If ancestors heights are set to 100% - the elements will take the whole screen space.

But, if one of nested element are bigger than screen size, and the `overflow` of it parent is not set – it will EXPAND the screen size, and all ancestors will take the 100% of new screen size. This is because the default overflow is `visible`.

Also setting h/w in % works like shit, so try to use these parameters together:

**flex-grow: 1** – sets that the element can grow and how much relatively to other growing elements,

**flex-basis: 100px** – sets the minimal size (height if the parent is flex-direction column)

**overflow: hidden** (or another) ­– limits the size of children.

Then, when you have a growing box, if you put another element inside – you can operate on **max-width max-height** parameters.

# HTML

## Form input validation

There are a lot of input type like “text”, “password”, “number”, “tel”, “time”

`accept` set file filter when the file explore window is opened, but the user can select any file still.

<label> is binded to `id`.

<input> is binded to controller method parameters through `name`.

Additional fast validation in js:  
  
html and js validation is only on client side and can be avoided. To enhance security add server-side validation:



# JS

## Fundamentals of JS

Primitive DataTypes:   
- Number – any number 3, 2.5, -2. Number bigger than ~1015 loose precision, than use BigInt  
- BigInt numbers (without decimals) that can be bigger than ~1015  
to declare BigInt: let y = 9999999999999999n;  
BigInt can’t be used in arithmetic with Number, use conversion:  
let x = 5n;  
let y = Number(x) / 2;  
- String, can use both “ “ and ‘ ‘.

- Boolean  
- Null  
- Undefined (similar to null)  
- Symbol - A newer feature to the language, symbols are unique identifiers  
Other data types:  
- Object - collections of related data.

- function (it is an object at the same time)

foo = function() {};

console.log(typeof foo); // function

Difference between var, let, const – its not important.

Var variables have function scope. Let variables has the block scope. It can’t be accessible outside the particular code block ({block}).

**function** f() {

**if** (**true**) {

**let** b = 9

console.log(b); *// 9*

}

console.log(b); *// ReferenceError: b is not defined*

}

f();

console.log(b); *// error*

**function** f() {

**if** (**true**) {

**var** b = 9

console.log(b); *// 9*

}

console.log(b); *// 9*

}

f();

console.log(b); *// ReferenceError: b is not defined*

**let** a = 10

*// It is not allowed (but it’s ok for var)*

**let** a = 9

*// It is allowed*

a = 10

**Hoisting in JS for var:**

When JavaScript is executed, the interpreter moves or “hoists” all variable declarations to the top of their containing function / scope boundary, regardless of where they occur.



Transforms into this

This is ok:

**let** a = 10;

**function** f() {

a = 9

console.log(a) *// 9*

}

f();

  
Const is the same as let but can’t be changed.  
  
Instead of

use



Difference between loose equality “==” and strict equality “===”:  
both treat NaN != NaN,  
“==” performs a type conversion when comparing two things, e.g. these consts are loose equal:  
const num = 0;

const big = 0n;

const str = "0";

const obj = new String("0");

const bool = false;  
“===” doesn’t perform a type conversion, examples above are not strict equal.  
null == undefined, but null !== undefined

null != false

null != undefined

new String('foo') != new String('foo')

Only Object.is(NaN, NaN) return true for comparing NaN (or use custom solutions).

## C# and JavaScript/TypeScript

Nice article about C# and JS <https://mauricebutler.wordpress.com/2011/11/07/getting-started-with-javascript-with-a-c-background/> (but it’s from 2011)

- Same { } ;  
- Same if, else, switch



- Class and Interface keywords same in C# and TS

- same accessors (public, protected …) in C# and TS

- arrow => same only for lightweight functions

- async await. C# returns Task, JS and TS return Promise,

- garbage collection,

- C# have NuGet package manager, for JS similar role plays npm,

- TS have union types,

- TS have decorators, e.g. @sealed. Some of them are implemented in C# as keywords or attributes,

- In JS, the value `NaN` (Not a Number) occurs when the value is returned as a number type, but the value is not parseable as a number. The value `infinity` occurs when a number exceeds the upper limit 1.7976931348623157E+10308,

- The type of `Null` in JS is `object`,

- In JS anything that exists and has a value will evaluate as true unless the value is false, null, undefined, 0, NaN or an empty string,

- Instead of `someObject?.prop` in JS write ` if (someObject) `

- You can add properties to objects (in this case it’s a function object):

- C# foreach loop in JS: **in -> of**

- JS **in** iterates over properties:



x: fname, lname, age

## Display pdf on desktop and mobile

1. Download and configure pdfjs library.
2. Use code:

### Code

@model Invent.Repository.Model.View.InfoKierowcaViewModel

@using System;

@{

    Layout = null;

    var base64 = @Model.TemplatedPDFBase64;

}

<style>

    .iframe-container {

        overflow: hidden;

        position: relative;

    }

        .iframe-container iframe {

            border: 0;

            height: 100%;

            left: 0;

            position: absolute;

            top: 0;

            width: 100%;

        }

        .iframe-container object {

            border: 0;

            height: 100%;

            left: 0;

            position: absolute;

            top: 0;

            width: 100%;

        }

    .btn-space {

        margin-right: 3px;

    }

</style>

<input type="hidden" id="pdfFile" value="@base64" />

<div class="row">

    <div class="row text-center">

        <div class="btn-group" role="group" style="margin-bottom:4px;">

            <button class="btn btn-sm btn-space" id="prev">Poprzednia strona</button>

            <button class="btn btn-sm btn-space" id="next">Następna strona</button>

        </div>

        <div class="col-sm-12">

            <span>Strona: <span id="page\_num"></span> / <span id="page\_count"></span></span>

        </div>

    </div>

</div>

<div class="iframe-container text-center">

    <canvas style="width:100%; max-width:800px" id="the-canvas"></canvas>

</div>

<script>

    (function ()

    {

        let BASE64\_MARKER = ';base64,';

        let pdfjsLib = window['pdfjs-dist/build/pdf'];

        pdfjsLib.GlobalWorkerOptions.workerSrc = window['pdfjs-dist/build/pdf.worker/src/pdf.worker.js'];

        document.getElementById('prev').addEventListener('click', onPrevPage);

        document.getElementById('next').addEventListener('click', onNextPage);

        let pdfAsDataUri = $('#pdfFile').val();

        let pdfAsArray = convertDataURIToBinary(pdfAsDataUri);

        let pdfDoc = null;

        let pageNum = 1;

        let pageRendering = false;

        let pageNumPending = null;

        let scale = 3;

        let pdfCanvas = document.getElementById('the-canvas');

        let ctx = pdfCanvas.getContext('2d');

        pdfjsLib.getDocument(pdfAsArray).promise.then(function (pdfDoc\_)

        {

            pdfDoc = pdfDoc\_;

            document.getElementById('page\_count').textContent = pdfDoc.numPages;

            renderPage(pageNum);

        });

        function convertDataURIToBinary(dataURI)

        {

            let base64Index = dataURI.indexOf(BASE64\_MARKER) + BASE64\_MARKER.length;

            let base64 = dataURI.substring(base64Index);

            let raw = window.atob(base64);

            let rawLength = raw.length;

            let array = new Uint8Array(new ArrayBuffer(rawLength));

            for (let i = 0; i < rawLength; i++)

            {

                array[i] = raw.charCodeAt(i);

            }

            return array;

        };

        function renderPage(num)

        {

            pageRendering = true;

            // Using promise to fetch the page

            pdfDoc.getPage(num).then(function (page)

            {

                let viewport = page.getViewport({ scale: scale });

                pdfCanvas.height = viewport.height;

                pdfCanvas.width = viewport.width;

                // Render PDF page into pdfCanvas context

                let renderContext = {

                    canvasContext: ctx,

                    viewport: viewport

                };

                let renderTask = page.render(renderContext);

                // Wait for rendering to finish

                renderTask.promise.then(function ()

                {

                    pageRendering = false;

                    if (pageNumPending !== null)

                    {

                        // New page rendering is pending

                        renderPage(pageNumPending);

                        pageNumPending = null;

                    }

                });

            });

            // Update page counters

            document.getElementById('page\_num').textContent = num;

        }

        function queueRenderPage(num)

        {

            if (pageRendering)

            {

                pageNumPending = num;

            } else

            {

                renderPage(num);

            }

        }

        function onPrevPage()

        {

            if (pageNum <= 1)

            {

                return;

            }

            pageNum--;

            queueRenderPage(pageNum);

        }

        function onNextPage()

        {

            if (pageNum >= pdfDoc.numPages)

            {

                return;

            }

            pageNum++;

            queueRenderPage(pageNum);

        }

    })();

</script>

Where TemplatedFile is

 $@"data:application/pdf;base64," + Convert.ToBase64String(bytes);

bytes is byte[]

Result:



# TypeScript

Nice article about C# and TypeScript: <https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes-oop.html>

Types are just sets:



In TypeScript, this becomes very natural once you realize that every type is just a set. How do you describe a value that either belongs in the string set or the number set? It simply belongs to the union of those sets: string | number.

- Object doesn’t need to have a relation to interface if I want to use this object in place where suppose to be interface-implemented-object.