JavaScript

ES8

Facts about JavaScript

- JavaScript (JS) - Dynamic language
- Scripting language for browsers and servers (JS for server side is run by node)
- JS is interpreted by the browser, no compilation
- Functional programming language
- Prototype version of JavaScript was written in **10 days** in1995
- Since 1996 ECMA is in charge of releasing the specification of JS, and browsers need to implement it.
- ES6 also known as ECMAScript2015 is the first major update since ES5 in 2009
- Since then every year there is a new version ES7-ECMAScript2016, ES8-ECMAScript2017
- Each browser implements the new version at his own pace

Facts about JavaScript

- Browsers Support ES6
  - New Desktop browsers > 96%
  - New mobile browsers > 99%
  - Unfortunately you don’t know which browser will run your program so you can’t assume that the user browser will support ES6
- We can transpile our code from ES6,ES7,ES8 to ES5
- Babel is a popular transpiler

JS - Hello World

- comments: //, /* */
- console - is used for debugging and we can use **console.log** to print a message
- **console.log** accepts a string to print
- **console.assert(<expression>)**
- JS scripts file ends with **js** suffix: <filename>.js
- Recommended IDE **JetBrains: WebStorm** (community edition is free)
- We can run the script by using our browser or using node
    - **node <my-script>**

Variable Declaration - var

- Since the first version of JS we defined a variable with: **var**
- Syntax: **var <variableName> = <assignment>;**
- assignment is optional
- **variableName** should be camelcased
- variable type can change dynamic language/loosly typed
- example:

```
var myString = ‘hello world’

myString = 10;
```

- What is the scope of var?

Variable Declaration - const, let

- Syntax:

```
const <variableName> = <assignment is a must>;
let <variableName > = <assignment is optional>
```

- const has a single assignment
- let can have multiple assignment
- Is single assignment mean immutable?
- The scope of let and const is inside the block
- What’s the result of the previous example when changing var to let ?

Basic Types - string

- Define a string: “”, ‘’, ``
- `` backticks are used for multiple lines
- `` with backticks you can inject javascript variables by using ${}
- you can concat strings with the **+**
- string is an array of characters so you can access a character like array syntax: **myStr[i]** (you can’t change the value, string are immutables)
- you can iterate on a string like array

- Some common functions: **indexOf, substr, split**

Basic Types - Numbers

- single number type that represents: float, positive, negative numbers
- Operators: +, -, *, \, %, **, ++, -- (** es6 - not supported by everyone)
- **toString** will convert number to string
- **parseInt, parseFloat** will convert from string to number if fails will return **NaN**
- numbers are immutable
- number constants: **NaN, Infinity, -Infinity**

Booleans

- true, false
- common tricks with boolean:
  - if (<var>) { … }
  - const myVar = expressionIfTrue || -1

- booleans are immutable
- logical operators: !, ==, ===, ||, &&, !=, !==

Basic Types - Miscellaneous

- undefined
- null
- NaN
- Infinity
- -Infinity

Advanced Types - Array

- Syntax: **const myArray = [‘bannana’, 10, true]**
- You don’t need to specify the size of the array
- You don’t need to specify the types the array can hold
- arrays are mutable
- common methods: **forEach, push, pop, splice,**
- common properties: **length**

Advanced Types - Object/Dictionary

- syntax: **var dict = {<string key>: <value>, <string key2>: <value2>}**
- access values: **dict.key1** or **dict[‘key1’]**
- add value: **dict[‘newkey’] = <new value>**
- get an array of all the keys: **Object.keys(dict)**
- delete a key: **delete dict[‘newkey’]**
- is key in object? **dict.hasOwnProperty(‘newkey’)**

less common advanced types

- **Map -** similar to **Object** but keys don’t have to be strings
- **Set -** set will have unique values

if… else

- syntax:

  ```
  if (condition) {
      ...
  } else {
      ...
  }
  ```

- Every value that is not: **undefined, null, 0, NaN, “”** is considered true

if … else if … else

- syntax:

```
if (condition) {

}

else if(condition2) {

}

else if(condition2) {

}

else {

}
```

switch

- syntax:

```
switch (expression) {
    case value1:
        …
        [break]
    case value1:
        …
        [break]
    default:
        ...
}
```

switch

- comparison is with ===
-

for

- syntax:

```
for([initialization]; [condition]; [final expression]){

    ...

}
```

- the initialization is run at first run, mainly will initialize a variable
- the loop will run while condition is true
- the final expression will execute at the end of each iteration
- all [initialization][condition][final expression] are all optional
- you can use **break** to exit the loop
- you can use **continue** to jump to next iteration

while

- syntax:

  while(condition){

  ….

  }

- break exit the loop
- continue move to next iteration

do… while

- Syntax:

  do {

  ...

  } while(condition);

- break and continue works as well

for...of, for...in

- Syntax:

  for(let i of <iterable>){ …}

  for(let i in <iterable>) { … }

- of will iterate over what iterable describe
- in will iterate on all enumerable properties including those added to the prototype of inherited elementes
- of is a loop on iterable objects
- in can loop on dictionary
- in is dangerous cause it can also iterate on prototype things

<script> - Loading scripts in the browser

- script element can be used to run a JS file or embed inline JS
- **src -** set this attribute to load script from file
- **type -** set this attribute to set what type of script it is (if omitted then default JavaScript)
- when browser see a script tag it will stop parsing the page and wait till the script is run (including downloading the script) unless placing the **async** attribute or **defer**
- the **defer** will run the script according to the order also defer will run after the document has been loaded
- not all browsers support async and defer
- let’s try and create an app that will alert an hello world message

<script> - Where to put the script tag

- place the <script> tag at the bottom of the <body> tag
  - Advantage: Browser will finish parsing the page
  - DisAdvantage: Script download won’t start immediately
- Place the <script> tag at the head and place a **defer/async** attribute
  - Advantage: Browser will start downloading the script immediately
  - Advantage: Browser won’t be blocked from further document parsing
  - DisAdvantage: Not supported by older browsers

window

- contains global properties and methods
- those properties can be access by: **window.<propertyName>** or just **<propertyName>**
- you can assign new properties that will be available global: **window.myProperty='hello world'**
- **localStorage -** associates key value with the domain (has no expiration time)
- **setTimeout, setInterval, clearInterval -** sets a functionto be executed periodically
- **alert** - will pop an alert message

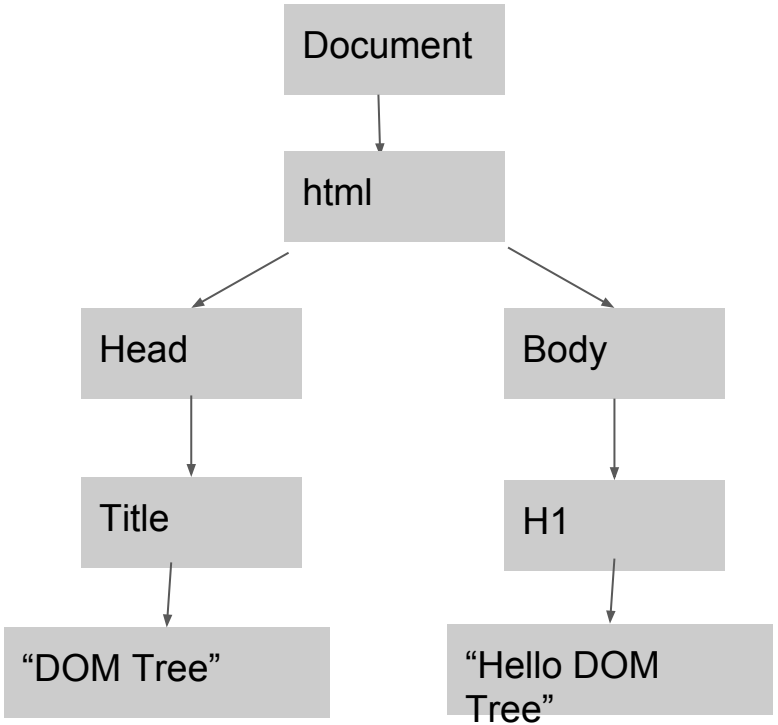
location

- Information and actions on the current location
- Location object contains the following useful methods:
  - assign - navigate to new page
  - reload - reload the current page
  - replace - same like assign only you can’t back to the current page
- Location object contains the following useful properties
  - pathname
  - hash
  - href
  - host
  - hostname
  - origin
  - pathname
  - port
  - protocol
  - search
  - hash
  - each of the properties can be changed and it will change the url

Document -DOM

- Browser loads HTML documents
- HTML document can be arranged in a tree structure

```
<html lang="en">
<head>
  <title>DOM Tree</title>
</head>
<body>
        <h1>Hello DOM Tree</h1>
</body>
</html>
```

Document - DOM

- The browser creates a tree of JavaScript Objects from the HTML tags
- That tree is called the DOM tree
- the DOM tree determines what will be displayed on the screen
- The root of the DOM tree is in the **document** object
- the DOM tree can be manipulated with JavaScript
- Let’s try to use the document cover the following:
  - DOM nodes selectors
  - Updating DOM nodes
  - Deleting DOM nodes
  - Creating a new DOM node

Document - DOM Selectors

- **getElementById**
  - returns a single element
  - id in HTML has to be unique
- **getElementsByClassName**
  - returns multiple elements with a shared class
- **getElementsByTagName**
  - returns multiple elements with the same tag

Document - Updating Delete Insert

- make an element **hidden**
- change inner text
- add a click event
- change the style
- add html
- get an attribute
- remove the dom node

Document - creating new elements

- **document.createElement**
- **document.createTextNode**
- **document.appendChild**

DOM element events

- You can use **addEventListener** to add events
- How do i know which events i can add?
- You can also add events by adding attribute to the HTML
- event functions will get the **Event** object as argument

<form>

- form can have a **method** attribute
- we can attach a submit event to form
- you can add **input, textarea, button** to a form
- by default form submit will send the request to the **action** attribute and will reload the page
- to prevent the page from reloading you can call **stopPropagation, preventDefault** on the **Event** object of the submit event

Chrome Developer Tools

- CDT is used for:
  - Style editing
  - debugging JS
  - Performance optimization
  - Request/Response inspection
  - JS console
  - Cookies/localStorage examine
  - Memory usage
- To open:
  - on windows: F12, Ctrl+Shift+I, Tools -> Developer Tools
  - on Mac:CMD + ALT + I, View -> Developer -> Developer tools

Chrome Developer Tools

- Let’s try to do the following common things with the developer tools
  - inspect HTML and add styles
  - print message to the console and interact with the page
  - setting a break point
  - using the **debugger** statement
  - See the Network and what requests and responses my apps are getting
  - Perform an audit on a page
  - How much memory does my app consume (Task Manager)

function - define a function

- Syntax:

```
function fooBar(arg1, arg2) {

    return arg1 * arg2;

}
```

- return is optional
- primitive values are passed by value, non primitive by reference
- functions can be created without a name - function expression
- it’s common for functions to accept functions as arguments

function - call a function

- call function: **sqrt(25);**
- function declaration can appear below where it is called
- variables defined in functions are only available in the scope of the function
- variables outside the function are available in the scope of the function
- You can call a function by using the **call** method in the function prototype
- In the previous version using call was a way to do inheritance
- You can call the functions by using the **Function.prototype.apply**
- apply is similar to call only the arguments are sent as an array
- call and apply will return the result of the function

function - arguments

- you can access the function arguments from: **arguments** array
- using arguments array you can deal with functions where you are not sure how many arguments you are going to get
- you can pass default value to arguments
- default arguments don’t have to be the last ones

this

- this behaves differently in JS then in other languages
- By default **this === window**
- when a function is called this is equal to window
- when a function has ‘use strict’ this is equal to undefined
- when a function is part of an object this will be the object
- when a function is called with the new then **this** will be the new object of the function (good for dealing with classes)
- you can use **bind** to set what **this** will be

Lambda Functions

- syntax
  - (arg1, arg2) => { … }
  - arg1 => { … }
  - (arg1, arg2) => 3 // return 3
  - arg1 => 3
- doesn’t have a **this**

Prototype

- JavaScript doesn’t have a subclass and inheritance like traditional languages
- JavaScript uses prototype to achieve this
- The base prototype is: **Object.prototype** nearly all object are instances of **Object**
- Some of the inherited methods: **toString, hasOwnProperty, create, getPrototypeOf, constructor**
- **Array** and **Function** has prototype as well which inherits from **Object.prototype**
- when searching for a property it will start from the nearest prototype and then search in the next one and next one (prototype chaining)
- the next prototype is saved in the **__proto__**
- we can use prototype to create classes and inheritance
- We can take advantage of prototype chaining and override methods in the chain

Class

- Class is a syntax sugar for creating a class and inheritance like common languages and not by using prototype
- The feature was added in ES6 (older browser do not support this)
- you can define **constructor** in the class
- inheritance is done with the **extend** keyword
- you can call base function by using **super** (in constructor it has to be the first statment)
- you can define static methods with the keyword **static**
- you can define getters and setters

Promises

- Promises are part of the language since ES6
- represent state of an async task
- promise can have 3 states: **pending, completed, rejected**
- the constructor gets an executor function with resolve reject methods
- The executor function will run immediately
- subscribing to a promise is done with **then**
- if a promise is already fulfilled when you call **then** then the subscriber will run immediately
- **then** returns a promise which allows us to do **Promise Chaining**
- It’s common to use **Promises** with server communication

XMLHTTPRequest, Fetch

- XMLHttpRequest/Fetch is used to interact with servers
- Used with ajax programming where you can retrieve information from server without doing refresh
- We will now practice using XMLHttpRequest/Fetch with our rest server

Todo Rest Server

- our rest server is located at this url: **https://nztodo.herokuapp.com**
- The server is connected to a database with a single table called task
- the task table api is in this path: **/api/task/**
- The server returns a **json** response

Task JSON

- A single task json looks this:

{"id":8529,"title":"mytitle","description":"mydescription","group":"mygroup","when":"2016-12-12T21:20:00Z"}

- **id** is the primary key and automatically created by the server
- **when** is an **ISOString** representing date time

CORS

- stands for **Cross-Origin Resource Sharing**
- As a security measure browsers restrict cross-origin HTTP requests initiated from within scripts
- using CORS spec we can do cross domain communication between browser and server
- CORS are used with HTTP headers
- CORS headers has **Access-Control-*** prefix
- **Access-Control-Allow-Origin** - is required in the response from the server
- Certain Requests for the server are considered simple and are sent directly to the server
- some requests like **PUT, DELETE** the browser will automatically send a preflight request

GET all tasks from server

host: https://nztodo.herokuapp.com

path: /api/task/?format=json

method: GET

- fetch will work with promise
- fetch will return promise even on bad response
-

Get a single task

host: https://nztodo.herokuapp.com

path: /api/task/:id/?format=json

method: GET

Insert new task

host: https://nztodo.herokuapp.com

path: /api/task/

method: POST

request body: {title: …, description: …, when: …, group: ...}

Delete

host: https://nztodo.herokuapp.com

path: /api/task/:id/

method: DELETE

UPDATE

host: https://nztodo.herokuapp.com

path: /api/task/:id/

method: PUT

request body: {title: …, description: …, when: …, group: ...}

Modules

- implemented natively only in Safari and Chrome
- when placing the **<script>** tag add the attribute **module**
- you can just place the entry script and not add **<script>** for every module

Modules - export

- using export you can expose **function, class, constant** that can be imported from other module
- you can use export to chain export from other files (good for barrel files)
- you can use export default then import name can change
- if using regular export then name is important

Modules - import

- you can import exported **functions, const, class**
- exported default items can be imported with any name
- export without default name should persist in import as well
- you can use **import * as name from …** to import everything in a module
- you can change the name of the import with alias

try...catch...finally…throw…

- **Syntax:**

  try{}

  catch(e) {}

  finally{}

- used to run statements and handle error event
- finally will always run used for cleanup
- you can use **throw** to jump to **catch**
- you can if on error type using **instanceof** to handle different types of exception
- 7 built in error object
- you can extend **Error** to create your own error object

EX1 - TODO no server

- In this ex we will practice manipulating DOM, add form, add events
- The app will have a single page
- At the top of the page there will be a form to create a new todo task
- the form will have the following input fields:
  - input for title
  - textarea for description
  - date input for the time of the task
  - submit button
- attach an event for the form submit
- when submitting the form the new task is added inside a **ul li** tags

EX2 - With Server

- Get the list of tasks from the server
- add a search at the top of the list
- the search should append a get param to the url
- when clicking an element in the list move to a new page with a get param of the list item you selected
- in the single task page you should query the server for a single task and display the details