**JavaScript-theory**

* JavaScript can run either in browser for animated components or in NodeJS
* JavaScript(programming language conforming to ECMAScipt) vs ECMAScript(specification-defining standards)
* ECMAScript version 1 in 1997 and in 2015 ES2015 or ES6, and each year after is released a new version.
* The javascript is used between the <script></script> and it can be used either in head(for third party libraries) or usually at the end of the body section
* NodeJS is a runtime environment for executing JavaScript code
* Types in JavaScript: **string, boolean, number, undefined, null**
* Reference Types: object, array and function

JavaScript is a dynamic language: The variable name can be changed at runtime

arrays

- the arrays in javascript can store different types of data

- the arrays are dynamic , can be changed at runtime

functions

-Functions are used to calculate a value or to perfume multiple statement to run a task

- The functions have parameters and when calling that functions the same parameters are called arguments

**==OPERATORS ==**

::Arithmetic, Assignment, Comparison, Logical, Bitwise

==Logical operators==

//Falsy(falsy) values in JavaScript

//undefined

//null

//0

//false

//''

//NaN

// !!!! Anything that is not Falsy -> Truthy

-----Function should have only one responsibility-------------

* Break a function into multiple functions (calculateGrade split in calculateGrade and Calculate averaged of values)

//**this** keyword is a reference to the object that execute that piece of code

There are two ways to create a new object: **Factory Function and Constructor Function.**

//------Factory Functions----------------

//camel notation oneTwoThreeFour

//Pascal Notation: OneTwoThreeFour

function createCircle(radius){

    // const circle = {

    return {

        // radius: radius,

        radius,

        draw(){

            console.log('draw');

        }

        //  draw: function(){

        //      console.log('draw');

        //  }

        };

    //   return circle;

}

const circle1 = createCircle(1);

console.log(circle1);

circle1.draw();

const circle2 = createCircle(2);

console.log(circle2);

circle2.draw();

//----------Constructor Function-Pascal Notaion---------------

function Circle(radius){

    //this keyword is a reference to the object that execute that piece of code

    this.radius = radius;

    this.draw = function (){

        console.log('draw');

    }

}

const circle = new Circle(1);

**Objects are dynamic:**

//Objects are dynamic

const circleD = {

    radius:1

};

//we can add new properties

circleD.color = "yellow";

circleD.draw = function(){}

//we can DELETE properties

delete circleD.color;

delete circleD.draw;

console.log(circleD);

Every Object created have a constructor: let x={}(new Object()). There are constructors for string -> new String()//can be used just ‘’ “” ``, new Boolean(), new Number()…

**!!!Functions are objects in JS**

**Reference Types**

Object

Function

Array are Objects in JS

Objects and Primitives have different behaviour

Primitives are copied by their value

Objects are copied by their reference

… Spreading an array mean taking every individual element

**Functions**

**Function declaration and Function Expression**

Hoisting is the process where the JavaScript engine move the

// function declaration on top of the file

… Rest operator look the same as the Spread operator

In ES6 the function can be added to an object without the function keyword

**//"This" keyword is referencing the object that executes current function**

//The rest operator transform the parameters into an array

function sum(...items){

    console.log(items);

    if(items.length ===1 && Array.isArray(items[0]))

    //copying the array in array to a flat array with the spread operator (copy)

        items =[...items[0]];

    return items.reduce ((a,b) => a+b)

}

**UDEMY -JAVASCRIPT**

1. Javascript is a high-Level(We don’t have to worry about complex stuff like memory management), object-oriented(based on objects, for storing most kinds of data), multi-Paradigm(We can use different styles of programming-imperative,declarative) programming language(instruct computer to do things)
2. Js- Allowsus to add dynamic effects and web applications in the browser
3. Modern JavaScript -> from 2015(ECMA script 6) and every year is a new release with new features (ES6->ES7->ES8…)
4. Data Types: Number, String, Boolean, Undefined(value taken by a variable that is not yet defined(‘empty value’), Null(also mean ‘empty value’, Symol(ES2015):value that is unique and cannot be changed, BigInt(ES2020):larger integers than the Number type can hold
5. JavaScript has dynamic typing: We do not have to manually define the data type of the value stored in a variable. Instead, data types are determined automatically.
6. In JavaScript the value have the type , NOT the variable. That means that the value can change from number to string. It also can generate problems
7. Should use const for variables first and change after to let if needed
8. Operators in JavaScript: Arithmetic(+,-,\*,/,\*\*), Assignment operators (“=,+=,-=,++,- -"), Comparison operators(“<, > , >=,<=”)
9. Type conversion(we do it explicitly) and coercion(when the java convert the data to another type for us): Conversion => console.log(Number(“234”),243,”243”);, String(23);;

Coercion => console.log(“I am “ + 23 + “years old”) transforming numbers to string

1. 5 Falsy values : 0, “”, undefined, null, NaN(they are converted to Boolean false). Anything else is truthy(1, “sadf”, etc)(converted to true)
2. Are used by the JS when using type coercion(implicit transforming the types-IF..else)
3. Strict equality operator(“===”) and it doesn't do the type coercion
4. Loose equality(“==”) does type coercion: if ("18" == 18) console.log("using loose equality and JS using type coercion")
5. JavaScript Releases: -
   * Released in 1995(Brendan Eich-Mocha in 10 days).
   * 1996 renamed in LiveScript(and the name JavaScript was there to marketing reasons to attract the java developers).
   * In 1996 Microsoft launched IE, copying JavaScript from Netscape and caling it Jscript.
   * In 1997 we need to standardize the language, it was implemented by ECMA and it released ECMAScript 1(ES1), the first official standard for JavaScript(ECMAScript is the standard, JavaScript the language in practice)
   * In 2009 was released ES5(ECMAScript 5) with lots of great new features;
   * In 2015 was released the ES6/ES2015(ECMAScript 2015) and it was the **biggest update to the language ever!**
   * After that ECMAScript changed to an annual release cycle in order to ship less features per update.
   * Release of ES2016, ES2017, ES2018, Es2019, Es2020…..ES2102
   * It has backwards compatibility:Don’t break the web(from 2020 works on 1997)
6. Don’t break the web! :
   * + - 1. Old features are never removed
         2. Not really new versions, just incremental updates(releases)
         3. Websites keep working forever
         4. There are some old bugs but it made to work around in modern javascript
         5. It is used to keep the websites build forever woking
         6. What you can build with it? Fontend, Backend, Mobile.
7. During the development : Simply use the latest Google Chrome or Firefox.
8. During production: Use Babel to transpile and polyfill your code(converting back to ES5 to ensure browser compatibility for all users)
9. ES5 -Fully supported in all browsers (down to IE 9 from 2011). Ready to be used today
10. ES6/ES2015-ES2020: ES6+ => Well supported in all modern browsers; No Support in older browsers; Can use most features in production with transpiling and plyfilling
11. ES20201- \*\* : ESNext: Future versions of the language (new feature proposal that reach Stage4). Can already use some features in production with transpiling and polyfilling.
12. Functions in JS are values: Function declaration (func name(name){return name}) and Function expression(name = func(name){return name;}). **Calling the functions with the same sintax: name(name).**
13. Arrow function used primarly for one line functions: const calcAge3 = birthYear => 2037 - birthYear;; the arrow function does not get “this” keyword
14. Functions are values in javascript
15. Keep the code dry. Don’t repeat yourself principle.
16. 4 Stepps to solve any problem: 1. Make sure you 100% understand the problem. Ask the right questions to get a clear picture of the problem.

2. Divide and conquer: Break a big problem into smaller sub-problems.

3.Don’t be afraid to do a much research as you have to.

4. For bigger problems, write pseudo-code before writing the actual code.

1. DOM manipulation: Document Object Model: Structured representation of HTML documents. Allows javascript to access HTML elements and styles to manipulate them
2. The HTML page is generated by the browser as a DOM tree structure(child, parent, sibling of ELEMENTS)
3. DOM Methods and Properties for **DOM Manipulation are not a part of JavaScript(ECMA)**.
4. DOM Methods and properties is a library available through the API implemented by the Web Browesers and they are already ready to use. And the JavaScript can interact with the methods. Also there are many more libraries available through the APIs like: Timers, Fetch …
5. Doom elements are saved in string… We need to convert them
6. DOM:The style property is added as an inline style. The property in css need to be with camel naming
7. DRY(don't repeat yourself) principle

**Java-under the hood**

1. Javascript is a **high-level**, **prototype-based object-oriented**, **multi-paradigm**, **interpreted or just-in-time compiled(0s and 1s)**, **dynamic**, **single-threaded**, **garbage-collected** programming language with **first-class functions** and a **non-blocking event loop** concurrency model.
2. High-level the programmer don’t worry about creatin variables in memory(abstractization) vs low-level (C project) which are potentially faster;
3. Paradigm is an approach and mindset of structuring code, which will direct your coding style and technique: Procedural programming, Object-oriented programming(OOP), Functional programming(FP). Javascript have all that paradigms. Javascript is very flexible
4. In a language with first-class functions, functions are simply treated as variables. We can pass them into other functions, and return them from functions.
5. Dynamically-typed language: No data type definitions. Types becomes known at runtime. Data type of variable is automatically changed.
6. Concurrency model: hot the JavaScript engine handles multiple taks happening at the same time. Why do we need that? Javascript runs in one single thread, so it can only do one thing at a time. So what about a long-running task? Sounds like it would block the single thread. However, we want non-blocking behavior! How do we achieve that? By using an event loop: takes long running tasks, executes them in the “background”, and puts them back in the main thread once they are finished.
7. What is a javascipt engine? It is a program that executes javascript code. Example : V8 Engine empower google and nodeJS. Each browser have its own engine.
8. Js engine is composed by CALL STACK(Where our code is executed using the execution context) and HEAP(Object in memory, where the objects are stored)
9. Compilation:Entire code is converted into machine code at once, and written to a binary file that can be executed by a computer. (java, desktop application etc.)
10. Interpretation: Interpreter runs through the source code and executes it line by line.(old javascript)
11. **Just-in-time(JIT) compilation:** Entire code is converted into machine code at once, then executed immediatle. (**modern javascript**).
12. Modern just-in-time compilation of javascript: Parsing(reading the code) into AST-Abstract syntax tree, Compiling the code into 101001010, and right after compilation the code is executed in the CALL STACK. First the compilation is poor and during the execution of the program the compilation is optimized in special threads that we can’t access from code.
13. Javascript runtime composition: JS engine(with HEAP and CALL STACK), Web APIs(DOM ,timers, Fetch API etc) functionalities provided to the engine, accessible on global window object. CALLBACK QUEUE -> callback functions from DOM event listener( event functions). The callbacks function are put on the call stack when it is empty to be executed. The event loop is essential for non-blocking concurrency model.
14. What is an execution context? Environment in which a piece of JavaScript is executed. Stores all the necessary information for some code to be executed. (pizza box is the execution context, the pizza is the javascript code , and the fork and bill are the helpers so that javascript code to be executed).

After compilation the code is executed top-down. After compilation -> Creation of gloval execution context(for top-level code)-not inside a function -> Execution of top-level code(inside global EC) -> Execution of functions and waiting for callbacks(click event callback).

1. Exactly one global execution context(EC): Default context, created for code that is not inside any function (top-level). One execution context per function: For each function call, a new execution context is created;(this function are provided by the event loop).
2. What is inside execution context? 1. Variable Environment: let, const and var declarations, Functions, arguments object. 2. Scope chain. 3. This keyword. These are generated during “creation phase”, right before execution.
3. Arrow function don’t have arguments object and this keyword.
4. The javascript code is executed in call stack. The principle is LIFO. The functions , variables are pushed on the call stack on top on each other and after they are finished they are pop out (removed) from call stack and after, the function below continue the execution.
5. The functions are pushed on the CALL STACK only when the functions are calle, if not the functions are not pushed
6. Scope Concepts: 1. Scoping : How our program’s variables are organized and accessed. “Weher do variables live?” or “Where can we access a certain variable, and where not?”. 2. Lexical scoping: Scopin is controlled by plancement of funcions and blocks ain the code. 3. Scpace or environment in which a certain variable is declared (variable environment in case of efunction). There is global scope, function scope, and block scope; 4. Scope of a variable: Region of our code where a certain variable can be accessed.
7. The 3 types of scope in JavaScript: 1. Global Scope->-Outside of any function or block – Variables declared in global scope are accessible everywhere, 2. Function Scope -> -Variables are accessible only inside function, NOT outside, - Also called local scope , 3.Block Scope(ES6)-> -Variables are accessible only inside block( block scoped), -HOWEVER, this only applies to let and const variables!(the var variables are not block scoped), -Functions are also block scoped(only in strict mode)
8. The scope of a function declared into a block is available only in that block if the “use strict” mode is used. If not the functions are available outside the block scope.
9. The scope chain: The inner scope can access outer declared variables from all outer scopes. It is called variable lookup in scope chain. The outer scope variables can’t use inner declared variables.
10. The VAR variables are function scoped: even if is declared into a block the var variables behave like it is declared as a function variable(unlike the const and let variables)
11. The scope chain has nothing to do with the order in which functions were called. It does not affect the scope chain at all. (Scope chain vs. Call Stack)
12. The scope of a function declared into a block is available only in that block if the “use strict” mode is used. If not the functions are available outside the block scope.
13. If the global or outer variables are redeclared in the inner scoped functions there is not a problem because the javascript look for the first declaration of the variable and only if it doesn’t find it it will looks outside of its scope.
14. Hoisting makes some types of variables accessible/usable in the code vefore they are actually declared. “Variables lifted to the top of their scope”. Behind the scene = > Before execution, code is scanned for variagble declarations, and for each variable, a new property is created in the variable environment object. function declarations(hoisted, initial value:actual function, scope: block(in strict mode, oderwise function); var variables(hoisted, initial value: undefined(a lot of bugs), scope: function);let and const variables(not hoisted(technically yes, But not in practice), uninitialized,TDZ(Temporal Dead Zone->error), scope:block. For function expressions and arrows : depends if they are declared var or let/const.
15. The temporal dead zone for let and const is the zone from the scope where the variable is declared to the initializing the variable. The error will be ReferenceError: Cannot access “job” before initialization instead of ReferenceError: x is not defined\
16. Why TDZ? Makes it easier to avoid and catch errors: accessing variables before declarations is bad practice and should be avoided; Makes const variables actually work.
17. Why hoisting? Using functions before actual declaration; var hoisting is just a byproduct(it is old and can be replaced now by let/const)
18. How the “this” keyword works? “this” keyword/variable: Special variable that is created for every execution context (every function). Takes the value of (point to ) the “owner” of the function in which the this keyword is used. “this” is NOT static. It depends on how the function is called, and its value is only assigned when the function is actually called.
19. Types of calling functions. Method - this = <Object that is calling the method>; Simple function call – this = undefined(In strict mode! Otherwise it calls the global object window(in the browser); Arrow function – this = <this of surrounding function (lexical this)> The arrow function don’t have “this” keyword and when calling this for an arrow function the outer object is called; Event listener – this = <DOM element that the handler is attached to>; more ways to call functions with this(new, call, apply, bind).
    * “this” keyword does NOT point to the function itself, and also NOT to its variable environment
20. Don’t use arrow function for method because it doesn’t have this keyword.
21. Primitives(Number, String, Boolean, Undefined, Null, Symbol, BigInt) are stored in execution context in CALL STACK and objects(object literal, arrays, functions etc) are stored in HEAP.
22. When creating a primitive in call stack and changing the value it creates a new object to a new address because it is immutable . If a new primitive is creating to point to a value, and then changing the previous variable to a new value, a new address is created.
23. When creating an object it creates an address in call stack with reference to memory address( the value of the address) in the HEAP which have the value of the object. Object to large to stack in the call stack. When creating an object pointing to the same object it can manipulate the object.
24. Only primitive values are immutable, not CONST.
25. SPREAD OPERATOR(PACK the VALUE) and it is after the “=” => REST OPERATOR (UNPACK the VALUE) and it is before the “=”
26. Sets is a modern data structure added in ES6. It contains only unique elements, doesn’t have indexes. It has the methods: add, delete, has,clear
27. Maps are pairs of key: values. The difference between the maps and object is that the keys can be anything like: objects, maps, Booleans, arrays …. Instead of only strings.

Maps have methods like: set, get, has, size, clear, delete.

1. When we have to store a simple list we use Arrays or Sets and when we have Key/Value Pairs(ex. JSON) we use Objects or Maps.(Keys allow us to describe values)
2. Data structures build in javascript: arrays, set, object, maps, weakMap, WeakSet. NOT BUILT IN: Stacks, Queues, Linked lists, Trees, Hash Tables.
3. Arrays: - Use when you need ordered list of values(might contain duplicates); - Use when you need to manipulate data as arrays have handy methods.
4. Sets: -Use when you need to work with unique values; -Use when high-performance is really important; -Use to remove duplicates from arrays
5. Objects: -More “traditional” key/value store (“abused” objects); -Easier to write and access values with . and []; -Use when you need to include functions (methods); - Use when working with JSON(can convert to map)
6. Maps: -Better Performance; -Keys can have any data type ; -Easy to iterate ; -Easy to compute size; -Use when you simply need to map key to values; - Use when you need keys that are not strings
7. Boxing. Javascript is smart and takes the primitive string and converts behind the scene to an object string so we can call the methods. After calling the method it returns a primitive string again(unboxing)
8. String methods: indexOf(); lastIndexOf(); slice()-returning a string from a given position and an optional end position, slice(1),slice(-5), slice(2,6); toUpperCase(); toLowerCase(); trim()-removing spaces from end and start; replace(“x”,”x”.toUpperCase); replacesAll(all words); startsWith(“air”), endsWith(“r”) -returns Booleans; includes(); split(“ “)-returns an array based on the split sequence ; join(“+”) join the splitted words with the (“+”); padStart(25,”+”), padEnd(35,”-“); repeat(10) repeat a message n times
9. In javascript there is just passing by value. We still can use the reference if its address is passed as value. In javaScript there is no passing by reference like in the C++ programming.
10. JavaScript have First-class functions(they are values and is a concept). This means: =JavaScript treats functions a first-class citizens; =This means that functions are simply values; = Functions are just another “type” of objects; = Store functions in variables or properties; = Pass functions as arguments or OTHER functions(event handlers); = Return functions FROM functions; = Call methods on functions(bind)
11. JavaScript have Higher Order functions: = A function that receives another function as an argument, that returns a new function, or both; = This is only possible because of first-class function. 1.Function that receives another function (eventHandler-Higher-order function, and the function return on click( Callback function); 2. Function that returns a function(function name -Higher-order function and the return of that function is a returned function)
12. Higher-order function it take as a parameter a function
13. It adds a layer of abstraction- the high-order is at a higher level of abstraction, and it need a lower order function in order to do some processing
14. Call , Apply methods are used for function that need to set a context for “this”. Call uses a list of arguments and the apply uses an array of arguments.
15. Bind method returns a function. Bind method can be use to set a context for “this” but also can set it to null. It can be used to preset some arguments in order to create a more specific new function based on mother function.
16. **CLOSURES**-we don't create explicitly

a closer is a function that remembers all the elements of the function that created the inner function(closer) at birth date

every function have access to the variable environment ov the execution context where it was created even the function that created the inner function is no longer active in call stack

the variables are attached to the inner functions(closers)

Variable environment is attached to the function somewhere in the engine, exactly as it was at the time and place the function was created,

the closure doesn't loose connection with the environment where it was created in the first place

priority to look for variable in the closure variable environment even bigger priority than scope chain

1. CLOSURES definitions:
   1. 1. A closure is the closed-over variable environment of the execution context in which a function was created, even after that execution context is gone;
   2. A closure gives a function access to all the variables of its parent function, even after that parent function has returned. The function keeps a reference to its outer scope, which preserves the scope chain throughout time.
   3. A closure makes sure that a function doesn’t loose connection to variables that existed at the function’s birth place;
   4. A closure is like a backpack that a function carries around wherever it does. This backpack has all the variables that were present in the environment where the function was created.

We do NOT have to manually create closures, this a JavaScript feature that happends automatically. We can’t even access closed-over variables explicitly. A closure is NOT tangible JavaScript object.