***JavaScript – Lab2 Report***

**Name:** Ahmed Mansour Mohamed  
**Track:** Open-Source Application Development

ـــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــ

**Content: -**

1. Notes on 10 methods of Object Data Type
2. Notes on 10 methods of Array Data Type  
   ( including the map and forEach methods)
3. Notes on closure/lexical Environment

**Notes on 10 methods of Object Data Type:**

1. **.Object( )**

* this method is the default constructor for creating an object data type.
* When I tried to give the method any type of arguments, it just returned an object of the data type I provided.

ـــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــ

1. **.assign(target,source,……..,sourceN)**

* Basically, it copies the data from more than one source object to a target object.
* It returns the target object.
* If the target and the source have the same key, target’s value will be overwritten with the source’s value.

ـــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــ

1. **.freeze(object)**

* Freezing and object means that the current values can’t be modified anymore.
* We can’t add new properties, or remove existing properties.
* It returns the same object it was given to freeze.

ـــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــ

1. **Object.values(obj)**

* This method returns the values of the keys inside the object.
* It takes an object, and returns an array with a random name which contains the values of the keys.

ـــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــ

1. **Object.entries(obj)**

* It returns an **array** which contains one key and its value.
* Could ne useful with the **for-of** loop, to get each key and its value, thus I think it can be useful for the phone book app.

ـــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــ

1. **Object.hasOwn(obj, ‘property’)**

* This function returns true(Boolean) if the given property was within the provided object.
* Helpful if you are looking for a certain property, and if it was there, then you can ge its value.

ـــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــ

1. **Object.seal(obj)**

* It returns the same object it was given.
* You can’t: add property, delete a property, configure a property.
* You can change the value of the existing properties, as long they are writable.

1. **Object.defineProperties(object, properties)**

* It returns the object that was passed to it.
* It adds a property, or modify it, if it already exists.
* We provide the value of the property in a curly brackets{ }, which contains the value of the property, and a Boolean value for the writable flag.

ـــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــ

1. **Object.is(value1, value2)**

* It’s a method used to compare the value and data type of two values, similar to ‘===’
* It returns a Boolean (true or false).

ـــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــ

**10-Object.definePropert(obj,prop,{descriptor})**

* It returns the object it was given.
* defines a new property directly on an object, or modifies an existing property on an object.
* The descriptors are more like a flags for this property.

**Notes on 10 methods of Array Data Type:**

1. **Array.reverse**

* This method reverses the order of the elements of a given array.
* It also changes the original array.

ـــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــ

1. **Array.sort( compare function)**

* It’s default behavior is to sort the original array ascendingly.
* It affects the original array, if we didn’t want to affect the original array we can use .toSorted().
* The compare function returns:   
  -ve if a < b.

+vs if a>b.

0 if a==b.

As (a) is the first element of the comparison, and (b) is the second element.

ـــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــ

1. **.map(callback Fn, thisArg)**

* It iterates on the array elements, passing every element to a function, and storing the result as a single element in a new array.
* It returns the new array, without modifying the original array.
* The function isn’t invoked for empty slots.

1. **.forEach(function)**

* It iterates on the array elements, executing the given function once.
* It doesn’t return anything.

Ex: array1.forEach((element) => console.log(element));

= the result will be printing each element from the console.log function, but the forEach itself does not return anything.

ـــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــ

1. **Array.from(arr , function)**

* It’s default behavior if it wasn’t given any function, then it will just split the elements of the given array in a new array.
* If it was given a function, then it will execute the function on every element of the given array, returning a shallow copy of the original array, modified with the given function.

ـــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــ

1. **Array.of( element1, element2,…..elementN)**

* Creates an array of the given elements, and return it.

ـــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــ

1. **.some(function)**

* it returns a **new array**, with the **element of the given index**, updated with the **new value we provided.**

ـــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــ

1. **.at(index)**

* It returns the element of the given index within the caller array.

ـــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــ

1. **.fill(value, start index, end index)**

* It updates the values of a given range of indices with the provided value.
* It modifies the original array.

ـــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــ

1. **Array.isArray( arr)**

* Checks if the provided variable was an array or not.
* Returns a Boolean value ( True or False).

ـــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــ

**Notes on closure/lexical Environment:**

* As we know, declaring a variable inside a function block, means that variable is bounded by this block of code, and can be accessed only in the local scope of the function.
* If the variable we were trying to call inside the function isn’t in the local scope, then the function would look up for it in the global scope.
* From the past two points, the conclusion would be that what’s inside the function(local scope) is valid only inside the function, and what’s outside the function(global scope) can be accessed inside the function.

**Lexical Scope:** is that mix between the function local scope and the global scope,Thus

**Lexical Environment:** would be the outer environment of a function and the inner environment of the function itself.

* When trying to write a nested function, for example a function that returns a function – as the example below can explain – even though the variable is declared in the local scope of the outer function, the inner function still can access it, How?

**Closure:** When we dive through the outer function to the inner function, as long as we are using the variable that was declared in the outer function, in the inner function, this variable is put in what’s called a **CLOSURE** to use it inside the inner function.

**Implementation**  
it can be used to apply the concept of “Encapsulation” to protect our data, How ?

**Example:**

Function fastest\_car(){

Let Bughati = 400;

Return function car(mycar){

Return Mycar >Bughati}

}

Var BMW=fastest\_car(); ***// now BMW has the function of car***

Console.log(BMW(300); ***//False***

* Here, if we tried to access the variable Bughati to modify it, we can’t, as it’s valid and accessible only in the local scope of fastest\_car function

Console.log (Bughati) //***undefined variable***