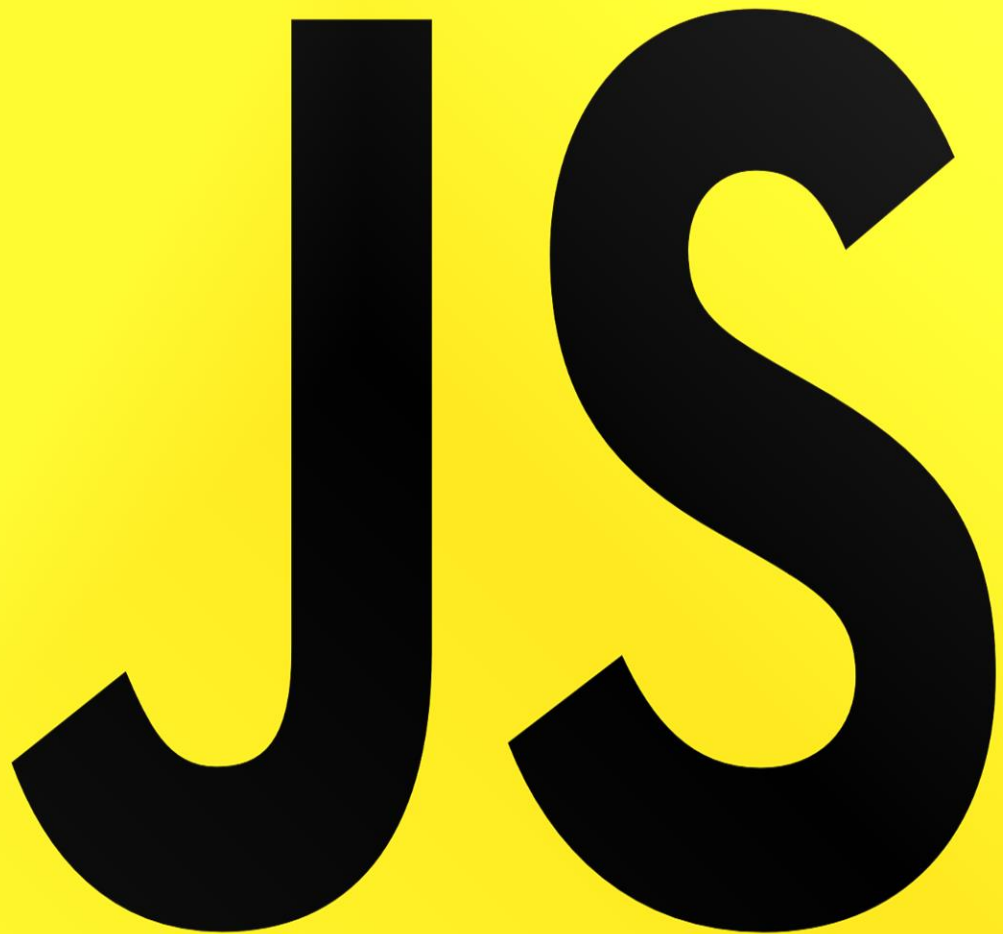


# JavaScript

Elzero Web School

WebSite Development

Front End

A large, bold, black 'JS' logo is centered on a bright yellow background. The letters are thick and stylized, with the 'J' having a curved bottom and the 'S' being a simple, rounded shape.



# BASICS JAVASCRIPT

## 1- window.alert

```
window.alert("Hello From JS File");
```

ظهور رسالة ترحيبية بأختيارك تكتب داخل القوس

## 2-document.write

```
document.write("<h1>Suhaib omar</h1>");
```

لكتابة كود داخل الصفحة بتنسيق تصميم موقع

## 3-console.log

```
console.log("omaradad");
```

للطباعة داخل Console

## 4- console.error

```
console.error("error");
```

لظهور رسالة تحذير للمستخدم داخل console

## 5- console.table

```
console.table(["suhaib","omar","Ahmad"])
```

لمجموعة معلومات معموله زي جدول داخل Console

## 6- console.log("%c ","color:red ;font-size:80px")

```
console.log("omar %c suhaib","color:red ;font-size:80px")
```

لتعديل على المتغير او الكلام المكتوب بنستخدم %c بنضيفها قبل الكلام المراد تعديله داخل console بمعنى لو عندي جملة وبدي اعدلها على كلمة واحده منهم فقط

## -----أنواع البيانات-----

7-

```
console.log(typeof("suhaib"));
```

نص - String

8-

```
console.log(typeof(5));
```

ارقام = Number

9-

```
console.log(typeof(false))
```

صح او خطأ = Boolean

10-

```
console.log(typeof([]))
```

جدول متغيرات == Array == Object

11-

```
console.log(typeof undefined)
```

12-

```
console.log({Name:"suhaib" , age:20 , country:["amman" , "Jordan" , "gaza"] , Active:(true) })
```

String

Number Array

Boolean

## -----المتغيرات-----

13-

```
var Name = "Suhaib",  
age = 23 ;  
console.log(Name)  
console.log(Name + " Omar ")  
console.log(Name + age)
```

المتغيرات و قيمهم و عند استخدام المتغير يجب كتابتها اولا قبل كود الاستعادة عند المتغيرات حرف كبير او صغير بفرق معك طريقة تسمية المتغيرات في جافا سكربت على طريقة camelCase اذا كانت من مقطعين الكلمة الاولى الاحرف صغيرة والكلمة الثاني اول حرف منها كبير هاذ هو تسمية المتغيرات بطريقة صحيحة في جافا سكربت

## -----أنواع المتغيرات-----

14-

### Var

Redeclare • (Yes)

Access• Before Declare • (Undefined)

Variable Scope Drama • [Added • To Window] • ()

### Let

Redeclare • (No •=>• Error)

Access Before Declare • (Error)

Variable Scope Drama • ()

### Consit

Redeclare • (No•=>• Error)

Access Before Declare • (Error)

15-

```
console.log('Suhaib omar "Halabe" ')
```

```
console.log("Suhaib omar ' halabe' ")
```

عند كتابة جملة داخل علامة تنصيص مزدوجة وتريد تمييز كلمة من ضمن الجملة ضعتها داخل علامة تنصيص منفردة والعكس صحيحة

```
console.log('Suhaib omar \"Halabe\"')
```

```
console.log("Suhaib omar \"halabe\"")
```

او بتقدر تحط هاذي العلامة \ بتعامل اشارة التنصيص على انها من ضمن النص وليس علامة برمجية

```
console.log("Suhaib
```

```
omar
```

```
halabe")
```

في هاي الحالة راح يعطيك غلط برمجي عشان تتخلص منو حظ العلامة ( / )

```
console.log("Suhaib\
```

```
omar\
```

```
halabe")
```

```
console.log('Suhaib omar \\"Halabe\"')
```

في حال حبيت تطبع العلامة \ مع علامة التنصيص بنحط اثنين منها وحده للطباعة و وحده بتعامل العلامة على انها من ضمن النص وليس علامة برمجية

```
console.log('Suhaib omar \n Halabe')
```

بنزل سطر جديد

16-

```
let a="we Love";  
let b="Javascript";  
let c="And";  
let d="Programming";  
console.log(a + " " + b + " \\" + c + " " + d );
```

طريقة الطباعة او كتابة الكود القديمة مقارنة بالجديدة ونفس النتيجة الي فوق لقديم ولي تحت لجديده

```
console.log(`${a} ${b} " " \\ ${c} ${d}`);
```

بمعنى انو بنحط ادخل Console.log() الاشارة الي عند حرف ذ الي هي ( ` ) وبعدين بنحط اشارة الدولار \$ بعدين هذول {} وداخل هذول الاشارتين المتغير بمعنى هيك بتصير Console.log(~\${a}~) وهاي هي الطريقة لجديده

17-

```
let title="Hello Elzero";  
let prgr="Elzero Web School";  
let sp="25/10";  
let markUp =`  
<div class="card">  
  <h3>${title}</h3>  
  <p>${prgr}</p>  
  <span>${sp}</span>  
</div>  
`;  
document.write(markUp.repeat(4));
```

حطينا 3 متغيرات كل متغير الو جملة

بعدين عملنا اخر متغير وهو الى راح يصمملنا صفحة ويب سايت

طبعا هاد المتغير ما بنحطوا بين علامتين تنصيص ("" ) بنحط علامة التنصيص هاي (~ ~ ) موجودة على حرف ذ

وعند Tags بنحط المتغيرات بين اشارتين علامة الدولار \$ وبين هذول القوسين { } وبالنص المتغير

واخر سطر برجمي بنقلوا اطلعنا المتغيره هاد وهون استخدمنا (repeat(Number))

repeat(4)

عشان ينفذ الي بدنا اياه 4 مرات بدل ما نضل نكتبهم كتابة

18-

let a=10;

a=20;

a=a+20;

a += 10 == a=a+10      نفس المعنى

console.log(a);

----- امتحان -----

19-Exame:-

let a=10;

b="20";

c= 80;

سؤال الاول:-

console.log(++a + +b++ + c++ - +a++);

//      11 + 20 + 80 - 11

//      31 + 71

//      100

هون اول اشي عند اول متغير زاد واحد وصار 11

المتغير الثاني بدل ما هو نص صار رقم بسبب علامة الزائد الي قبله

ثالث متغير ضل زي ما هو

المتغير الرابع هو كان 10 وصار 11 في المسألة



السؤال الثاني :-

```
console.log(++a + -b + +c++ - -a++ + +a);
```

السؤال الثالث:-

```
console.log(--c + +b + --a * +b++ - +b * a + --a - +true)
```

20-

```
console.log(1000000);
```

```
console.log(1_000_000); // جافا سكربت بتجاهل الاتدر سكور
```

```
console.log(1e6) // حرف الاي بعدها رقم يعني كم سفر بدك
```

```
console.log(10 *10 *10 *10 *10 *10);
```

```
console.log(1000000.0)
```

كل الاشكال الي ممكن تعطيك فيها جافا سكربت رقم مليون

```
console.log(Number.MAX_SAFE_INTEGER)
```

هاد اقصى حد لارقام الامنه الي ممكن تعطيك اياها جافا سكربت وما يصير شغلات غريبه

```
console.log(Number.MAX_VALUE)
```

اقصى رقم ممكن تعطيك اياه جافا سكربت

21-

```
console.log(Math.round(99.2));
```

```
console.log(Math.round(99.5));
```

بجبر الكسور فوق 5 بزيد العدد الصحيح واحد , تحت ال5 بخليه زي ما هو

```
console.log(Math.ceil(99.1));
```

قد ما كان عدد الكسور بجبرها ليزيد العدد الصحيح

```
console.log(Math.floor(99.9999));
```

بخلي العدد الصحيح زي ما هو

```
console.log(Math.min(10 , 20 , 10 , -100 , 90))  
console.log(Math.max(10 , 20 , 100 , -100 ,90 ))
```

أكبر عدد وأصغر عدد

```
console.log(Math.pow(2, 5))
```

رقم 2 مرفوع لرقم 5 يعني  $2^5 = 32$

```
console.log(Math.random())
```

أرقام عشوائية

```
console.log(Math.trunc(99.9))
```

بحذف الكسور

-----Exame-----

## 22-Exame

```
let a=1_00;//100  
b=2_00.5;//200.5  
c=1e2;//100  
d=2.4;//2.4
```

Find Smallest Number in All Variables And Reeturn Integer

```
console.log(Math.floor(Math.min(a , b ,c ,d)));
```

أعطيني أصغر عدد بينهم وشيل الكسور

//Use Variable a + d One time To Get The Needed Output

```
console.log(Math.pow(a , Math.trunc(d))); //10,000
```

أعطيني قيمة د بدون كسور وخلي حرف ا قيمة الأوس لالو

Get Integer "2" From d Varianle With 4 Methods

```
console.log(Math.trunc(d));
```

حذف الكسر

```
console.log(Math.floor(d));
```

أعطيني العدد صحيح

```
console.log(Math.round(d));
```

أجبر الكسر

```
console.log(Math.ceil(d-1));
```

أجبر العدد لعدد صحيح مهما كان الكسور بعيننا 3 ونقص من واحد

/Use Variables b + d To get This Values

```
let y =Math.pow(b , 0.9)- Math.pow(d , 4.5);
```

```
console.log((parseInt(b)/Math.ceil(d)).toFixed(2)); // 66.67 => String
```

```
console.log(Math.ceil(y)); //67 => Number
```

23-

```
let theName="Ahmad";
```

```
console.log(theName);
```

```
console.log(theName["Number"])
```

بتعطيك الحرف حسب ترتيبه بالارقام

```
console.log(theName[5])//undefined
```

اعطاك undefined لانو العدد المطلوب اكبر من عدد الارقام

بتعطيك الحرف حسب ترتيبه بالارقام

```
console.log(theName.charAt(1))
```

```
console.log(theName.charAt(5))
```

```
console.log(theName.length);
```

عدد الاحرف ( كم حرف )

```
let myName=" suhaib  ";
```

```
console.log(myName.trim());
```

بتشيل المسافات قبل وبعد لان المسافات بتتسب

```
console.log(theName.toUpperCase());
```

بتعطي النص كامل حروف كبيره

```
console.log(theName.toLowerCase());
```

بتعطي النص كامل حروف صغيره

```
console.log(theName.trim().charAt(2).toUpperCase())
```

المتغير + شيل المسافات + اعطيني الحرف الى رقمو 2 + وحولو الى حرف كبير

24-

```
let a="Elzero Web School";
```

```
console.log(a.indexOf("Web"))
```

أبحثلي على الكلمة الي موجودة بين القوسين وعطيني رقم موقعها

إذا كان الجواب سالب واحد يعني الكلمة موجوده

إذا موجوده بعطيك رقمها

```
console.log(a.indexOf("Web", 8))
```

عد 8 خانات وأبحثلي على الكلمة الي موجودة بين القوسين وعطيني رقم موقعها

```
console.log(a.lastIndexOf("Web",8))
```

أبحثلي على الكلمة الي موجودة بين القوسين وعطيني رقم موقعها بس بلش من نهاية الجملة

```
console.log(a.slice(5 , 12))
```

```
console.log(a.slice(-5 , -3 ))
```

ببدا من الحرف الي بيمثل الرقم الى اعطيتو اياه لحد الرقم الي اعطيتو اياه وبطبعهم

```
console.log(a.repeat( 5))
```

تكرار العنصر لعدد المرات الي تطلبها

```
console.log(a.split(" "))
```

بحطلك المتغير ادخل اريه وبقسمها حسب ما تطلب انا فوق طالب من عند المسافات

```
console.log(a.split("",5))
```

بحطلك المتغير داخل اريه وبقسمها حسب ما تطلب انا فوق طالب من عند المسافات + عدد التقسيمات

25-

```
let a="Elzero Web School"
```

```
console.log(a.length)
```

يعدلك كم حرف

```
console.log(a.substring(2,6));
```

بتعطيك الكلمة من بداية الحرف 2 الى 6

```
console.log(a.substring(6,2));
```

اذا كان الرقم الاول اكبر من الرقم الثاني يتم اعادة تصفيط الرقمين الى من 2 الى 6 واعطاء النتيجة

```
console.log(a.substring(-10,6))
```

اذا كان الحرف الاول بسالب ببدا من الصفر

```
console.log(a.substr(0, 6));
```

بتعطيك الكلمة من بداية الحرف 2 الى 6

```
console.log(a.substr(17));
```

بتعد من حرف رقم 17 الى اخر الجملة

```
console.log(a.substr(-3));
```

بتبليشي عكسي من اخر الجملة الى بدايتها

```
console.log(a.substr(-5, 2));
```

حرف رقم 5 من نهاية الجملة عدلي بعدو خانتين

```
console.log(a.includes("Web"));
```

بدورك على الكلمة بين القوسين بالجملة وبعطيك صح او غلط

```
console.log(a. includes("Web" , 8));
```

بعد الخانه رقم 8 درولي على الكلمة الي بين القوسين

```
console.log(a.startsWith("E"));
```

اذا بداية المتغير حرف E اعطيني صح

```
console.log(a.startsWith ("E" , 2));
```

بعد الخانه الثانيه اتاكد اذا اول المتغير حرف E

```
console.log(a.startsWith("zero",2));
```

بعد الخانه الثانيه اتاكد اذا اول المتغير الكلمة بين القوسين

```
console.log(a.endsWith("o" , 5)) ;
```

بعد الخانه الثانيه من نهاية الجملة اتاكد اذا اول المتغير الكلمة بين القوسين

26-Exam:-

```
let a="Elzero Web School";
```

```
console.log(a.slice(2,3).toUpperCase() + a.slice(3,6));
```

```
console.log(a.charAt(13).toUpperCase().repeat(8))
```

```
console.log(a.split(" ",1 ))
```

```
console.log(a.substr(0 , 6) + " " + a.substr(11))
```

```
console.log(a.charAt(0).toLowerCase() +a.substring(1 , a.length-1).toUpperCase()+a.substring(a.length-1).toLowerCase() )
```

27-

```
console.log(10 == "10")
```

مقارنة بين القيمة

```
console.log(10 === "10")
```

مقارنة بين القيمة و النوعيه

```
console.log(10 != "-10")
```

هل لا تساوي القيمة

```
console.log(10 !== "10")
```

هل لا تساويه القيمة و النوع

```
console.log(10 > 20);
```

```
console.log(10 < 20);
```

```
console.log( "suhaib" === "omar")
```

سؤال اجى لواحد في مقابلة شغل :: انو متساويات بالنوع ولكن مختلفات بالقيمة ف الجواب راح يكون غلط انهم مش متساويين

كيف بدنا نخليهم متساويين

الجواب

```
console.log(typeof "suhaib" === typeof "oamr")
```

بنضيف

```
typeof
```

28- Logical Operators

## Logical Operators

**! == Not**

**&& == And**

**|| == or**

**\*/**

```
console.log(true);    =true
```

```
console.log(!true);  = false
```

```
console.log(!(10 == "10"));
```

هل القيمة داخل القوس بتعطي صح , اه بتعطي طيب اعكس جوابها

```
console.log(10 == "10" && 10 < 9);
```

```
console.log(10 == "10" || 10 > 80);
```

## 29- Control Flow ( If & else if & else)

```
if (Condition) {
```

```
    // Block Of Code
```

```
}
```

```
let price = 100;
```

```
discount = false;
```

```
discountAmount = 30 ;
```

```
country = "KSA";
```

```
if (discount == true) {
```

```
    price -= discountAmount
```



```
    }else if (country === "Jordan"){  
        price -= 40;  
    }else if(country === "Jordan"){  
        price -= 60;  
  
    }else{  
        price -= 10;  
    }  
    console.log(price);
```

### 30- IF inside IF

```
    if (discount == true){  
        price -= discountAmount  
    }  
  
    else if (country === "Jordan"){  
        price -= 50;
```

```
        if (student === true) {  
            price -= 10  
        }  
    }  
}
```

### 31- Forms IF

```
let theName="Ahmad";  
let theGender ="Male";  
let theAge = 30;
```

الشكل الاول

```

if(theGender === "Male"){
    console.log("Mr");
}else{
    console.log("Mrs");
}

```

الشكل الثاني  
الشرط بشكل مختصر

Condition ? if True : IF False

```

theGender === "Male" ? console.log("Mr") : console.log("Mrs");

```

الشكل الثالث

```

let result = theGender === "Male" ? "Mr": "Mrs";

```

```

document.write(result);

```

الشكل الرابع

```

console.log(`Hello ${theGender === "Male" ? "Mr": "Mrs"} ${theName}`)

```

الشكل الخامس

```

//IF

```

```

theAge < 20 // 20 لو العمر اصغر من

```

```

? console.log(20) // 20 الشرط < اطبع التالي لو اصغر من

```

```

//Else If

```

```

: theAge > 20 && theAge < 60 // 60 لو العمر اكبر من 20 واصغر من

```

```

? console.log("20 To 60") // أطبعلي التالي

```

```

//Else If

```

```

: theAge > 60 // 60 لو العمر اكبر من

```

```
أطبعلي التالي // console.log("Larger Then 60")  
//Else  
لو ولا وحده اطبعلي التالي // console.log("Unknown")
```

### 32- null & undefined & Number

```
ممکن يكون المتغير واحد من هذول //  
// null // undefined // Number
```

```
let price = null;
```

```
إذا كان الجواب الراجع للطباعه احدى المتغيرات الي فوق راح يعطيك المتغير الي حاطينوا بعد هذول الاشارتين //  
// ||  
console.log(`The Price Is ${price || 200}`)
```

```
اما اذا كان الجواب صفر ولا صح ولا غلط ما راح يتم تغير المتغير الراجع للطباعة //  
بس بغيرها للقسم الثاني من السعر اذا كان يرجعك احدى المتغيرات التاليه //  
// null OR undefined  
console.log(`The Price Is ${price ?? 200}`)
```

### 33-Exam:-

```
let a = 10;  
  
if (a<10) {  
    console.log(10);  
}else if (a >= 10 && a<= 40) {  
    console.log("10 To 40");  
}else if (a> 40) {  
    console.log(">40");  
}
```

```
}else{  
    console.log("Unknown");  
}
```

```
a < 10  
? console.log(10)  
: a >= 10 && a <= 40  
? console.log("10 To 40")  
: a > 40  
? console.log(">40")  
: console.log("Unknown")
```

---

#### Video 37

```
let st = "Elzero web School";  
if("?????" === "34"){  
    console.log("Good");  
}  
//W Poition May Change  
if("?????" === "W"){  
    console.log("Good");  
}  
if("?????" !== "String"){  
    console.log("good")  
}  
if("?????" === "ElzeroElzero"){  
    console.log("Good");  
}
```

حل السؤال

```
let st = "Elzero web School";
```

```

er = st.length*2;
if(er.toString() === "34"){ console.log("Good");}

//W Poition May Change
if(st[st.indexOf("w")] === "w"){console.log("Good");}
if(st.length !== "String"){ console.log("Good")}

if(st.slice(0,6)+st.slice(0,6) === "ElzeroElzero"){console.log("Good");}

```

### 34- switch

```

let day = 0;
switch (day) {
  case 0:
    console.log("StaurDay")
    break;
  case 1:
    console.log("SunDay")
    break;
  case 2:
    console.log("MonDay")
    break;
  default:
    console.log("UnKnown Day")
    break;
}

```

Exame:-

### Video 39

المطلوب حول if الى switch

### 35-

```
let job = "Designer";
```

```
let salary = 0;

if (job === "Manager") {
    salary = 8000;
}else if(job === "IT" || job === "Support"){
    salary = 6000;
}else if(job === "Developer" || job === "Designer"){
    salary = 7000;
}else{
    salary = 4000;
}
console.log(salary)
```

تم التحويل

```
switch (job) {
    case "Manager":
        salary = 8000;
        console.log(salary)
        break;
    case "IT":
    case "Support":
        salary = 6000;
        console.log(salary)
        break;
    case "Developer":
    case "Designer":
        salary = 7000;
        console.log(salary)
        break;
    default:
```

```
salary = 4000;  
console.log(salary)  
break;  
}
```

شرح السؤال اعطانا جملة IF واحنا مطلوب منا نحولها الى Switch المتغير الرئيسي هو نوع الوظيفة وليس الراتب

Exame 2:-

```
let holidays = 0;  
let money = 0;  
switch (holidays) {  
  case 0:  
    money = 5000;  
    console.log(`my money is ${money}`)  
    break;  
  case 1:  
  case 2:  
    money = 3000;  
    console.log(`my money is ${money}`)  
    break;  
  
  case 3:  
    money = 2000;  
    console.log(`my money is ${money}`)  
    break;  
  case 4:  
    money = 1000;
```

```

    console.log(`my money is ${money}`)

    break;
case 5:
    money = 0;
    console.log(`my money is ${money}`)
    break;
default:
    money = 0;
    console.log(`my money is ${money}`)
    break;
}

```

شرح السؤال اعطانا جملة Switch واحنا مطلوب منا نحولها الى if المتغير الرئيسي هو نوع الوظيفة وليس الراتب

تم التحويل

```

if (holidays === 0) {
    money = 5000;
    console.log(`my money is ${money}`)
}else if(holidays === 1 || holidays === 2){
    money = 3000;
    console.log(`my money is ${money}`)

}else if(holidays === 3){
    money = 2000;
    console.log(`my money is ${money}`)
}

else if(holidays === 4){
    money = 1000;

```



```
console.log(`my money is ${money}`)  
}
```

```
else if(holidays === 5){  
    money = 0;  
    console.log(`my money is ${money}`)  
}  
else{  
    money = 0;  
    console.log(`my money is ${money}`)  
}
```

### 36- Array:

```
let myFriends = ["suhaib" , "omar" , "Al-halabe" , ["sdsdsdsd" , 'qweqweqe']];
```

بتقدر تحط اثنين Array

```
console.log(`Hello ${myFriends[0][5]}`)
```

بتقدر تختار الكلمة الي بذك تطبعها مثال myFriends[0][5] المربع الاول بتختار منو الكلمة والمربع الثاني بتختار الحرف من الكلمة الي بذك تطبعوا

```
console.log(myFriends[3][0])
```

```
myFriends[1] = "swswsw";
```

هون بتبدل الكلمة الي رقمها بال array 1 بالكلمة من عندك

```
console.log(myFriends.length)
```

```
console.log(myFriends)
```

```
myFriends[4] = ["faten" , "Wateen"]
```

هون بتضيف array داخل array بصير عندك array داخل array

```
console.log(myFriends);
```

```
console.log(typeof myFriends);
```

هون JavaScript بتتعرّف على array انها Object مش array

```
console.log(Array.isArray (myFriends));
```

Array.isArray بتفحص اذا المتغير array او لا

### 37- Arrays Methods

```
let myFriends = ["omar2", "suhaib", "oamr", "faten", "majde", "Al-halabe" ];
```

```
myFriends[myFriends.length] = "Gamal" ;
```

```
console.log(myFriends);
```

هون بنعمل اضافة على اخر array او توماتيكية قد ما كان عددها

```
myFriends[myFriends.length - 1] = "Gamal2" ;
```

```
console.log(myFriends);
```

هون بنلغي اول عنصر في array وبنضيف محلوا العنصر الي بدنا اياه

```
myFriends[myFriends.length - myFriends.length - 1] = "Gamal2" ;
```

```
console.log(myFriends);
```

اضافة عنصر في بداية array

```
myFriends.length = 3;
```

```
console.log(myFriends)
```

قد ما كان طول الarray عن طريق length. بنحدد قديمش نطبع منها كما هو موضح بالمثال

### 38-Arrays Methods Adding And Removing

```
let myFriends = ["suhaib" , "omar" , "Ahmad" , "Watten"];  
console.log(myFriends);
```

```
myFriends.unshift("sss" , "ddd");  
console.log(myFriends);
```

unshift بمعنى اضافة عنصر في بداية ال array

```
myFriends.push("smah" , "Eman ");  
console.log(myFriends);
```

unshift بمعنى اضافة عنصر في نهاية ال array

```
let first = myFriends.shift();  
console.log(myFriends);  
console.log(first);
```

shift بشيل اول عنصر من array وبتقدر تضيفها في array ثاني

```
let last = myFriends.pop();  
console.log(myFriends);  
console.log(last);
```

shift بشيل اخر عنصر من array وبتقدر تضيفها في array ثاني

### 39-Arrays Methods Search

```
let myFriends = ["suhaib" , "omar" , "Ahmad" , "Watten" , "omar"];  
  
console.log(myFriends);  
  
console.log(myFriends.indexOf("omar"));
```

أبحثني على اسم omar من بداية array الى نهايتها واعطيني رقمو

```
console.log(myFriends.indexOf("omar" , 2));
```

أبحثي على اسم omar من بداية array الى نهايتها بعد ما تعد عنصرين واعطيني رقمو

```
console.log(myFriends.lastIndexOf("omar"));
```

أبحثي على اسم omar من نهاية array الى بدايتها واعطيني رقمو

```
console.log(myFriends.lastIndexOf("omar" , -2));
```

أبحثي على اسم omar من نهاية array الى بدايتها بعد ما تعد عنصرين واعطيني رقمو

```
console.log(myFriends.includes("omar"));
```

أبحثي على اسم omar في array واعطيني الجواب بعطيك 1 يعني موجود ام في حال -1 يعني مش موجود

```
console.log(myFriends.includes("omar" , 2));
```

أبحثي على اسم omar في array بعد ما تعد عنصرين واعطيني الجواب بعطيك 1 يعني موجود ام في حال -1 يعني مش موجود

```
if(myFriends.lastIndexOf("omar") === -1){
```

```
    console.log("Not Found");
```

```
}
```

ابحثي على العنصر Omar داخل array في حال ما كنا موجود اطبعلي Now Found .

#### 40- Arrays Merthods sort

```
let myFreands = [10 , "Sayed" , "mohamed" , "90" , 1000 , 100 , 20 , "10" , -20 , -10];
```

```
console.log(myFreands);
```

```
console.log(myFreands.reverse());
```

بتعمل على عكس ترتيب Array

```
console.log(myFreands.sort());
```

ترتيب Array على حسب الابدجية

```
console.log(myFreands.sort().reverse());
```

بنقدر نستخدم الترتيب مع بعض على ترتيبها على حسب الابدجية و نكس الترتيب لجديد

## 41-Arrays Methods Slicing

```
let myFreands = ["Ahmad" , "Sayed" , "Ali" , "Suhaib" , "Gamal" , "Omar"];
```

```
console.log(myFreands.slice());
```

بتعمل على اقتطاع جزء معين من arrays وبتعمل array جديدة فيهم بتبدأ من العنصر رقم 0

```
console.log(myFreands.slice(1));
```

ابداء من العنصر رقم 1 واقتطع من array من رقم 1 الي نهاية array

```
console.log(myFreands.slice(1,3))
```

ابداء من العنصر رقم 1 واقتطع من array من رقم 1 الي رقم 2 array رقم 3 لا يوخذ لانو النهاية

```
console.log(myFreands.slice(-3));
```

```
myFreands.splice(0,0,"ommmaarr" , "sssuuhhaaiib")
```

اول خانه داخل القوسين البداية وثاني خانه هي خانه الحذف بتكتب رقم العنصر الي بك تحذفه داخل array وثالث خانه بتكتب العناصر الي بك تضيفها

```
console.log(myFreands)
```

## 42 – Arrays Methods

```
let myFriends = ["Ahmad" , "Sayed" , "Ali" , "Suhaib" , "Gamal" , "Omar"];
```

```
let myNewFriends = ["samar" , "sameh"];
```

```
let schoolFriends = ["Haytham" , "Shady"];
```

```
let allFriends = myFriends.concat(myFriends,schoolFriends , "suhia" , [1,2])
```

```
console.log(allFriends);
```

ضفناهم جميع array مع بعض عن طريق concat وحطيناهم داخل array جديد

```
console.log(allFriends.join(" | ").toUpperCase())
```

عملنا بين عناصر array لجديد فواصل او فراغ او اي اشي بدنا اياه عن طريق Join()

43-Exam:-

```
let zero = 0;
```

```
let counter = 3;
```

```
let my = ["Ahmed" , "Mazero" , "Elham" , "omar" , "Gamal" , "Ameer"];
```

```
console.log(my);
```

```
Q1- console.log(my); → ["oamr" , "elham" , "mazero" , "Ahmed"]
```

```
A-console.log(my.reverse().slice(--counter));
```

```
Q2-console.log(my.slice("????")); → ["Elham" , "Mazero"]
```

```
A-console.log(my.reverse().slice(counter).splice(zero , --counter , )); // ["Elham" , "Mazero"]
```

```
Q3 console.log(); → "Elzero"
```

```
A-my.splice(zero,counter+counter ,'Elzero');
```

```
Q4-console.log(); → "rO"
```

```
A-console.log(my[0].slice(++counter).charAt().toLowerCase()+my[0].slice(++counter).toUpperCase());
```

الشرح المطلوب في السؤال انو بدك اطلع الاجوبة الى موجود بعد الاسعم بشرط انك تستخدم المتغيرات الي فوق فقط

#### 44- Foor Loop

```
for(let i =0 ; i <10 ; i++){  
    console.log(i);  
}
```

احنا قلنا انو i اصغر من 10 وما قلنا اصفر او تساوي بسبب انو ببدا عد من رقم صفر مش من واحد:

#### 45-

```
let myFriends = [1 , 2 , "Suhaib" , "omar" , "Ahmad" , "Ali" , "ssower", "oomarsa" , true , false];
```

عملنا Arrays تحتوي على Boolean و Number و Strings بدنا نستخدم if and for عشان نقسم العناصر كل مجموعة لحال

```
let Numbers = [];
```

array فاضيه للارقام

```
let onlyName = [];
```

array فاضيه للاسماء

```
let Boolean = [];
```

array فاضيه للقيم المنطقية

```
for(let i=0 ; i < myFriends.length ; i++){  
    if(typeof myFriends[i] === "string"){  
        onlyName.push(myFriends[i]);  
    }else if (typeof myFriends[i] === "number"){  
        Numbers.push(myFriends[i])  
    }else{  
        Boolean .push(myFriends[i])  
    }  
}
```

الشرح : استخدمنا for عشان نخل يعدي وعلى كل عنصر موجود داخل المصفوفة القسم الاول من for من وين يبداء العنصر والقسم الثاني عملنا اوتوماتك يعني قد ما زدنا المصفوفة ال for بتشتغل عادي , if الاولى بنفحص ال i الي وصل لعند عنصر معين في المصفوفة بعد ما ال i اخذ رقمو وساواه اذا هو عباره عن رقم او نص او قيم صحيحة بعدين بنحطوا داخل array لجديده وينطبعهم

```
console.log(onlyName);
```

```
console.log(Numbers);
```

```
console.log(bollen);
```

#### 46-Products Practice

```
let products = ["Keyboard" , "Mouse" , "Pen" , "Pad" , "Monitor" , "Iphone" , "PC" , "suhaib"];
```

```
let colors = ["red","Green","Blue","black"];
```

```
let showCount= 8;
```

```
if(showCount < products.length+1){
```

```
    document.write(`<h1>Show ${products.length} Products</h1>`)
```

```
    for(let i =0 ; i< showCount ; i++){
```

```
        if (i < products.length) {
```

```
            document.write(`<div>`);
```

```
            document.write(`<h1>${i + 1+ "-"} ${products[i]}</h1>`)
```

```
            for(let j=0 ; j < colors.length; j++){
```

```
                document.write(`<p>${j + 1+ "-"} ${colors[j]} </p>`)
```

```
            }
```

```
            document.write(`</div>`);
```

```
        }
```

```
    }
```

```
}else{
```

```
    document.write(`<h1>Sory you Cant find that</h1>`)
```

```
}
```

انا زودت على الشرح كثير (فيديو رقم 53 )



#### 47-Loop While

```
let products = ["Keyboard", "Mouse", "Pen", "Pad", "Monitor", "Iphone"];  
let index = 0;  
while (index < products.length) {  
  console.log(products[index]);  
  index += 1;  
};
```

While loop اذا ضل الشرط متحقق بضلها شغاله بدون ما توقف مثال

```
while (0 < 10 )  
{  
  console.log( index);  
};
```

#### 48- Loop Do / while

```
let i = 0;  
do{  
  console.log(i);  
  i++;  
}while(false);  
console.log(i);
```

الفرق بين do و while انو ال do بنفذ الشرط بالاول بعدين بتحقق من الاوامر

#### 49-Loop Challenge

```
let myAdmins = ["Ahmed", "Omar", "Sayed", "Stop", "Samera"];
```

```
let myEmployees = [
```

```
    "Amgad",
```

```
    "Samah",
```

```
    "Ameer",
```

```
    "Omareca",
```

```
    "Othman",
```

```
    "Amany",
```

```
    "Samia",
```

```
];
```

```
let x = 0;
```

```
let y = 0;
```

```
for (let i = 0; i < myAdmins.length; i++) {
```

```
    if (myAdmins[i] === "Stop") {
```

```
        break;
```

```
    }
```

```
    x += 1;
```

```
}
```

```
document.write(`<div>We Have ${x} Admins</div>`);
```

```
for (let i = 0; i < myAdmins.length; i++) {
```

```
    if (myAdmins[i] === "Stop") {
```

```
        break;
```

```
    }
```

```
    y += 1;
```

```
document.write(`<hr><div>`);
```

```
document.write(`The Admin for team ${y} is ${myAdmins[i]}`);
```

```

document.write('<h2>Team Members:</h2>');

let k = 1;

for (let j = 0; j < myEmployees.length; j++) {

    if (myAdmins[i][0] === myEmployees[j][0]) {

        document.write(`${k}- ${myEmployees[j]}<br><br>`);

        k++;

    }

}

document.write('</div>');
}

```

#### شرحو باستخدام الذكاء الاصطناعي

**\*\*أول إشي\*\***، عندنا مصفوفتين: وحدة اسمها `myAdmins` اللي فيها أسماء المدراء، والثانية `myEmployees` اللي فيها أسماء الموظفين. الهدف من الكود هو نعرض المدراء وعدد الموظفين اللي بيبدأ اسمهم بنفس الحرف الأول من اسم المدير.

**\*\*بالنسبة لأول حلقة\*\***، بنمر على المصفوفة `myAdmins` عشان نعدّ عدد المدراء لحد ما نوصل لكلمة "Stop". كل ما نلاقي اسم مدير، بنزيد العداد `x` بواحد. إذا وصلنا لكلمة "Stop"، بنوقف الحلقة.

**\*\*بعد ما نخلص الحلقة الأولى\*\***، بنعرض عدد المدراء اللي حسبناهم.

**\*\*في الحلقة الثانية\*\***، بنمر برضو على المصفوفة `myAdmins`، بس هاي المرة عشان نعرض تفاصيل كل مدير وفريقه. منبدأ بكتابة اسم المدير ورقم الفريق، وبعدين بنطلع على الموظفين.

**\*\*داخل الحلقة الداخلية\*\***، بنشيك إذا اسم الموظف بيبدأ بنفس الحرف الأول من اسم المدير. إذا كان نفس الحرف، بنعرض اسم الموظف تحت اسم المدير. إذا ما كان نفس الحرف، بنكمل للحلقة اللي بعدها.

المتغير `k` بنستخدمه عشان نعدّ عدد الموظفين لكل مدير بشكل منفصل، وكل ما نلاقي موظف بنفس الحرف الأول من اسم المدير، بنزيد `k` بواحد عشان نعمل رقم تسلسلي لكل موظف.

**\*\*بالمختصر\*\***، الكود بيعرض عدد المدراء وبيجيب كل مدير والموظفين اللي بينتموا لفريقه بناءً على أول حرف من اسمهم

```
function sayHello(userName) {
  console.log(`Hello ${userName}`);
}

sayHello("suhiab");
sayHello("omar");
sayHello("Ahmad");
```

## 51-Example function

### One Example

```
function sayHello(userName, age) {
  if (age < 20) {
    console.log(`App Is Not Suitable For You`);
  } else {
    console.log(`Hello ${userName} Your Age Is ${age}`);
  }
}

sayHello("suhiab", 30);
sayHello("omar", 50);
sayHello("Ahmad", 15);
```

### Tow Example

```
function generateYears(start, end, exclude) {
  for (let i = start; i <= end; i++) {
    if (i === exclude) {
      continue; } console.log(i);
  }
}

generateYears(1982, 2021, 2020);
```

## 52-Return Function

```
function generate(start, end) {
  for (let i = start; i <= end; i++) {
    if (i === 15) {
      return;
    }
    console.log(i);
  }
}
generate(10, 20);
```

استخدمنا **Function** عطينا لها اسم وبين القوسين بداية ونهاية بداخلها خلينا **start** و **end** تساوي **start** و **end** اصغر من او تساوي **end** عشان يصير يطبعنا الارقام الى راح نخطها محل قوسين ال **return** شغلها في **if** نفس شغل **break** بمعنى اطلع ولما توصلي رقم **15** وقف وطلع خارج **if**

**Return** ادخل **function** ما بنكتب اشي تحتها ولو نكتب فهو غلط وما راح يتاخذ فيه

### 53-Default Function Parameters

```
function sayHello(userName = "Unknown", age = "Unknown") {
  //if (age === "undefined"){age = "Unknown"}
  // age = age || "Unknown"
  return `Hello ${userName} Your Age Is ${age}`;
}
console.log(sayHello("Suhaib", 20));
```

في حال انو المستخدم ما اعطى البيانات الاسم او العمر في عنا 3 طرق بنقدر من خلالها نخط قيمة افتراضية غير القمية الي جافا سكربت خطاها , 1- بنفس **Function** بنخط لكل متغير قيمة افتراضي , 2- باستخدام **IF** 3- بنخط للمتغير قيمتين قيمة الي راح يدخل المستخدم || القيمة الافتراضية

### 54-Rest Parameters

```
function calc(...numbers) {
  let result = 0;
  for (let i = 0; i < numbers.length; i++) {
    result += numbers[i];
  }
  return `Final Result Is ${result}`;
}
console.log(calc(10, 20, 30, 40, 50, 60, 70, 80, 1));
```

عملنا function عشان تعمل عملية الجمع تلقائية من خلال انو اسم function لخيرها array عن طريق انو ضفنا (...) قبل الاسم  
فصارت .array

## 55-Function Advanced Practice

```
function showInfo(us = "Un", ag = "Un", rt = 0, show = yes, ...sk) {
  document.write(`<Div>`);
  document.write(`<h2>Welcome, ${us}</h2>`);
  document.write(`<p>Age:${ag}</p>`);
  document.write(`<p>Hour Rate : ${rt}</p>`);
  if (show === "yes") {
    if (sk.length > 0) {
      document.write(`<p>Skills : ${sk.join(" | ")}</p>`);
    } else {document.write(`<p>Skills : No Skills</p>`);}
  }
  } else { document.write(`<p>Skills Is Hidden</p>`); }
  document.write(`</div>`);}
showInfo("Osama", 38, 20, "yes", "Html", "Css");
```

الشرح

هاي دالة بجافاسكريبت اسمها `showInfo`. الدالة هاي وظيفتها إنها تعرض معلومات معينة على صفحة الويب باستخدام `document.write``، يعني بتضيف نصوص للصفحة مباشرة.

لما تنادي الدالة، بتقدر تعطيه أربع معلومات (أو معطيات) رئيسية: اسم المستخدم، العمر، معدل الأجر بالساعة، وإذا بدك تعرض المهارات ولا لا.

إذا ما أعطيت كل المعلومات، الدالة فيها قيم افتراضية؛ مثلاً، اسم المستخدم الافتراضي هو "Un"، والعمر الافتراضي كمان "Un"، ومعدل الأجر بالساعة صفر. بالنسبة لموضوع المهارات، الدالة بتأخذ كل المهارات اللي بتحب تضيفها وبتجمعهم مع بعض.

الدالة بتشتغل كالتالي:

- أول إشي بتفتح كود HTML وتكتب "أهلاً وسهلاً" وبتحط الاسم اللي أعطيته إياها.
- بعدين بتكتب العمر ومعدل الأجر بالساعة.
- بعد هيك، إذا أنت قتلتها بدك تعرض المهارات (`show` = "yes"`)، بتطلع إذا فيه مهارات أعطيتها إياها. إذا فيه، بتعرضهم. إذا ما فيه، بتكتب "No Skills".
- أما إذا قتلتها ما تعرض المهارات (`show`` مش "yes")، بتكتب "Skills Is Hidden".
- وبالنهاية، بتسكر الكود اللي فتحته بالبداية.

يعني، إذا استخدمتها زي المثال اللي أعطيتي إياه (`"Osama", 38, 20, "yes", "Html", "Css"`)، رح تظهر رسالة على صفحة الويب بتقول:

- أهلاً وسهلاً، Osama

- العمر: 38

- معدل الأجر بالساعة: \$20

- المهارات: Html | Css

وإذا قتلتها ما تعرض المهارات، بدل ما تعرض "Html | Css"، رح تكتب "Skills Is Hidden".

هيك بتكون الدالة بتظهر معلومات بشكل مبسط على صفحة الويب!

Exame:-

```
function showDetails(...data) {
  let name, age, status;

  for (let i = 0; i < data.length; i++) {
    if (typeof data[i] === "string") {
      name = data[i];
    } else if (typeof data[i] === "number") {
      age = data[i];
    } else if (typeof data[i] === "boolean") {
      status = data[i] === true ? "Available" : "Not Available";
    }
  }

  return `Hello ${name}, Your Age Is ${age}, You Are ${status}`;
}
```

```
console.log(showDetails("Ali", 25, true));
console.log(showDetails(30, "Sara", false));
```

### الشرح

#### 1. الدالة وقبول المدخلات:

- الكود عبارة عن دالة اسمها `showDetails`، هاي الدالة بتقدر تاخذ عدد غير محدود من المدخلات (زي الاسم، العمر، والحالة إذا متاح أو مش متاح) وتجمعهم في مصفوفة وحدة.

#### 2. تحديد نوع كل مدخل:

- جوه الدالة، في متغيرات جاهزة عشان نخزن فيها الاسم والعمر والحالة. بنستخدم حلقة تمر على كل عنصر من العناصر اللي دخلها المستخدم.

- بنفحص نوع كل عنصر:

- إذا كان نص (string)، بنحطه في المتغير الخاص بالاسم.



- إذا كان رقم (number)، بنحطه في المتغير الخاص بالعمر.
- إذا كان منطقي (boolean) يعني True أو False، بنقرر إذا الشخص "متاح" أو "غير متاح".

### 3. التحقق من حالة الشخص:

- إذا كانت قيمة الحالة `true`، معناته الشخص "متاح".
- إذا كانت قيمة الحالة `false`، معناته الشخص "غير متاح".

### 4. تكوين الجملة النهائية:

- بعد ما نحدد الاسم والعمر والحالة، بنرجع جملة مرتبة فيها كل المعلومات: "مرحبا، اسمك كذا، عمرك كذا، وأنت متاح/غير متاح".

### 5. الاستخدام:

- لما تستدعي الدالة وتعطيها المعلومات (زي الاسم والعمر والحالة بأي ترتيب)، الدالة تلقائياً بترتيبهم وتطبع الجملة النهائية بشكل صحيح.

### ### الخلاصة:

الكود هدفه إنه يجمع معلومات من المستخدم (زي الاسم والعمر والحالة)، يحدد نوع كل معلومة، ويرتبهم عشان يطلع جملة واضحة توضح هاي المعلومات. مش مهم الترتيب اللي دخلت فيه المعلومات، الدالة بترتيبهم صح وبتعطيك النتيجة بشكل سهل ومباشر.

```
let calculator = function (num1, num2) {
  return num1 + num2;
};
console.log(calculator(10, 20));
```

## 57-Function Inside Function

### المثال الاول

```
function sayMessage(fName, lName) {
  let message = `Hello`;
  //Nested Function
  function concatMas() {
    message = `${message} ${fName} ${lName}`;
  }
  concatMas();
  return message;
}
console.log(sayMessage("Suhaib", "oamr"));
```

الشرح :- عملنا function الاولى وبدها متغيرين الاسم الاول و الاسم الثاني عملنا متغير اسمو message وحطينا فيه hello ال function الثاني فيها المتغير message بساوي قيمتو الي هي hello مع الاسم الاول و الاسم الثاني طلعا من function وناديننا function الثاني ورجعنا قيم message لانو هاد المتغير بحتوي على متغيرات function الاول

المثال الثاني ((نفس شرح الي فوق بس غيرنا شو نادينا))

```
function sayMessage(fName, lName) {
  let message = `Hello`;
  //Nested Function
  function concatMas() {return `${message} ${fName} ${lName}`;
  }
```

```

return concatMas();
}
console.log(sayMessage("Suhaib", "oamr"));

```

### المثال الثالث

```

function sayMessage(fName, lName) {
  let message = `Hello`;
  //Nested Function
  function concatMas() {
    //Nested Function
    function getFulName() {
      return `${fName} ${lName}`;
    }
    return `${message} ${getFulName()}`;
  } return concatMas();
} console.log(sayMessage("Suhaib", "oamr"));

```

### 58-Arrow Function

```

Let print = function (num){
  Return num;};
===

```

```

Let print (num) => num;

```

ال function التين الي فوق متساويات ونفس الاشئ بس انتبهى الاختصار تبع function مش دايمًا يكون هيك هاد الو متغير واحد و امر واحد

```

Console.log(print(100));

```

```

let print = function (num1, num2) {

```

```
return num1 + num2;
};
```

الشكل الثاني من function المختصره

```
let print = (num1, num2) => num1 + num2;
console.log(print(10, 50));
```

## 58-Scope

```
var a = 1;
var b = 2;
function showText() {
  let a = 10;
  let b = 20;
  console.log(`Function-From Local ${a}`);
  console.log(`Function-From Local ${b}`);
}
showText();
console.log(`From Global ${a}`);
console.log(`From Global ${b}`);
```

المتغير في منو نوعين العام والخاص و العام الي كل الدالات بتقدر تستخدموا

الخاص ما حد بقدر يستخدموا بالمثال المتغير الي دخال Function هذول متغيرات خاصه الي بالخارج هذول متغيرات عامه

## 59-Block Scope

Let تعتبر من المتغيرات الخاصه Var من المتغيرات عالمي مثال:-

```
Var x = 10;
If(10 === 10){
  //Var x = 27;
  Let x = 27;
  Console.log(x);
}
```

Console.log(x)

الي راح يصير هون ال x راح تتغير قيمتها من 10 ل 27 وين مكان لانو var عالمي وليس خاص :  
اما لما استخدمنا let داخل if تغيرت الشغلة صارت صار عنا 2 x واحد عالمي وقيمتوا 10 و واحد خاص في if فقط الي هو قيمتو  
27

## 60 - Lexical Scope

```
function parent() {  
  let a = 10;  
  console.log(`From Parent A= ${a}`);  
  function child() {  
    console.log(`From Child A= ${a}`);  
    function grand() {  
      let b = 100;  
      console.log(`From Grand A= ${a}`);  
      console.log(`From Grand B= ${b}`);  
    } grand();  
  } child();  
} parent();
```

Function لما يكون فيها متغير بتطلع ادور على قيمتو في Function الخارجية , أما اذا كان Function فيها متغير وبدها ادور عليه ما بتفوت على Function الداخليه  
بمعنى انو المتغير بطلع برا دائرة عشانو يدور على قيمتو بس ما بتفوت بدائرة بداخلوا.

Exame:-

1-السؤال الاول :-

```
let names = function (){  
  //Parameter ?  
  Return "???"  
};  
Console.log(names("Suhaib" , "Omar" , "Ahmad","alhalabe"));  
//String [Suhaib] , [omar] , [ahmad] , [alhalane] => Done !
```

بدك تحط متغير تقدر من خلاله تطبع اخر جملة بنفس النظام والشكل

الجواب

```
let names = function (...Name) {  
  return `String [ ${ Name ( ) , ( ) } ] => Done ! `;  
};  
console.log(names("suhiab" , "omar" , "Ahmed" , "Al-halabe"));
```

2- السؤال الاول بدك تعيد صياغة الجواب ولكن بنظام Arrow Function

الجواب

```
let names = (...Name) => `String [ ${Name[0]} , [ ${Name[1]}] , [ ${Name[2]}] , [ ${Name[3]}] => Done ! `;  
console.log(names("suhiab" , "omar" , "Ahmed" , "Al-halabe"))
```

السؤال الثاني:- ممنوع استخدام اي رقم من عندك

```
Let myNumbers = [20 , 50 , 10, 60];  
Let calc =(one , tow , ...nums) => "????";  
Console.log(calc(10 , "????" , "????"));
```

الجواب الاول

```
let myNumbers = [20 , 50 , 10 , 60];  
let calc = (one , tow , ...num) => `${one + tow + num[Number(True)]}`;  
console.log(calc(10 , myNumbers[Number(Fales)] , myNumbers[Number(True)]))
```

الجواب الثاني

```
let myNumbers = [20, 50, 10, 60];  
let calc = (one, tow, ...num) => one + tow + num[Number(false)];  
console.log(calc(10, myNumbers[Number(true)], myNumbers[Number(false)]));
```

```
let myNums = [1, 2, 3, 4, 5, 6];
```

```
let newArray = [];
```

```
for (let i = 0 ; i<myNums.length ; i++){  
  newArray.push(myNums[i] + myNums[i])  
}console.log(newArray);
```

For بتعمل على جمع الرقم مع نفسه

Map بتعمل على تطبيق Function على كل جزء داخل array

Filter برتجلك العناصر الى نجحت من test

Same Idea With Map

```
let addSelf = myNums.map(function(element){  
  return element + element;  
});console.log(addSelf)
```

Arrow Function

```
let addSelf = myNums.map((element) => element + element);
```

```
console.log(addSelf);
```



## 62- Map -Swap Cases

```
let swqappingCases = "elZERo";  
let invertedNumbers = [1, -10, -20, 15, 100, -30];  
let ignoreNumbers = "Elz123er4o";
```

عنا 3 متغيرات

```
let sw = swqappingCases . split("") . map(function (ele) {  
    //Condition ? true : False  
    return ele === ele.toUpperCase() ? ele.toLowerCase() : ele.toUpperCase();  
}) . join("");  
console.log(sw);
```

السؤال الاول :- حول الحروف الكبيره لحروف صغيره و حروف الصغيره لكبيره:

```
let inv = invertedNumbers.map(function (ele) {  
    return -ele;  
});  
console.log(inv);
```

السؤال الثاني :- حول الارقام الموجبه لارقام سالبه والعكس صحيح :

```
let ign = ignoreNumbers . split("") . map(function (ele) {  
    return isNaN(parseInt(ele)) ? ele : "";  
}) . join("");  
console.log(ign);
```

السؤال الثالث :- الغي الارقام من الجملة و اطبعها

### 63-Filter Function Map

```
let frindes = ["Ahmed", "Hameh", "Sayed", "Asmaa", "Amged", "Israa"];  
let filterFrineds = frindes.filter(function (ele) {  
    return ele.startsWith("A");  
});  
console.log(filterFrineds);
```

ادخل على array وتفقد هم عنصر عنصر ولي ببداء بحرف A حطو داخل Array لحال:

```
let nubers = [11, 20, 2, 5, 17, 10];  
let filterNumber = nubers.filter(function (ele) {  
    return ele % 2 === 0;  
});  
console.log(filterNumber);
```

ادخل على array وتفقد هم عنصر عنصر ولي عنصر الي باقي قسمتوا صفر حطو في array لحال:

### 64-filter and map

السؤال كالتالي بذك تفصل بين الارقام و الحروف وتضرب الارقام في بعض

```
let mix = "A13BS2ZX";  
let filterMapMix = mix  
    .split("")  
    .filter(function (ele) {  
        return !isNaN(parseInt(ele));  
    })  
    .map(function (ele) {  
        return ele * ele;  
    }) .join("");  
console.log(filterMapMix);
```

استخدمنا map و filter سوا ال filter طلعتنا الارقام فقط من المتغير و map خليناها تضرب المتغيرات الي طلوعوا في بعض:

```
let sentence = "I LOVE FOOD CODE TOO PLAYING MUCH";
```

السؤال بقلك طلعي الكلمات الي عدد احرفها اقل او تساوي 4 حروف

```
let upperWorld = sentence.split(" ").filter(function(ele){  
    return ele.length <= 4  
}).join(" ");console.log(upperWorld)
```

## 65-Reduce

```
let nums = [10, 20, 15, 30];
```

```
let add = nums.reduce(function (acc, current, index, arr) {  
    console.log(`Acc => ${acc}`);  
    console.log(`current Element=> ${current}`);  
    console.log(`current Element Index=> ${index}`);  
    console.log(`array => ${arr}`);  
    console.log(acc + current);  
    console.log(`#####`);  
    return acc + current;  
}, 5);
```

**reduce** هي دالة موجودة على المصفوفات (Arrays) في JavaScript ، وفكرتها إنها بتساعدك تجمع أو تحسب قيمة وحدة من كل عناصر المصفوفة. بمعنى، بتأخذ كل عنصر بالمصفوفة وبتطبق عليه عملية معينة وبتطلع لك نتيجة وحدة المتغير الأول: (Accumulator) هذا المتغير هو اللي بيجمع النتيجة من كل عملية.  
المتغير الثاني: (Current Value) هذا المتغير هو قيمة العنصر الحالي في المصفوفة اللي عم نشغل عليه.

## 66- Reduce 2

```
let theBiggest = ["Bla", "Propagande", "Other", "AAA", "Battery", "Test", "Propagande_Tow"];
let check = theBiggest.reduce(function(acc, current){
    return acc.length > current.length ? acc : current;
});
console.log(check);
```

استخدمنا **reduce** على المصفوفة **theBiggest** عشان نلاقي أطول كلمة فيهم الفانكشن اللي داخل **reduce** بياخذ متغيرين:

**Acc** هذا المتغير هو "المجمع"، بيخزن أطول كلمة لحد اللحظة

**Current** هذا المتغير هو العنصر الحالي من المصفوفة اللي عم نشوف إذا أطول من اللي عندنا أو لا.

لفانكشن بفحص طول الكلمة الموجودة في **acc** مقارنة مع طول الكلمة في **current**.

إذا: **acc.length > current.length** بيرجع **acc**، لأنه الكلمة اللي فيها أطول.

إذا: **current.length > acc.length** بيرجع **current**، لأنه الكلمة اللي فيها أطول.

وهيك **reduce** بيكمل يمر على كل الكلمات بالمصفوفة، وبالأخير رح يرجع أطول كلمة فيهم.

```
let removeChars = ["E", "@", "@", "L", "Z", "@", "@", "E", "R", "@", "O"];
let carc = removeChars
    .filter(function (ele) {
        return !ele.startsWith("@");
    })
    .reduce(function (acc, current) {
        return `${acc}${current}`;
    });
console.log(car);
```

الكود اللي عندك بيعمل شغلتين رئيسيات:

أول إشي، بيثيل كل الرموز "@" من المصفوفة اللي فيها حروف ورموز. يعني بيخلي بس الحروف.

ثاني إشي، بيجمع الحروف اللي ظلوا بعد ما شال الرموز "@"، وبيحولهم لكلمة وحدة.

فالنتيجة النهائية رح تكون كلمة وحدة بتتكون من الحروف اللي كانت بالمصفوفة، وبتطبع على الشاشة. بهالحالة، الكلمة اللي بتطلع هي "ELZERO".

## 67- ForEach

`forEach` هي دالة بتساعدك تمر على كل عنصر في المصفوفة وتنفذ عليه إشي معين. مثلاً، إذا عندك قائمة أرقام وبك تطبع كل رقم، بتستخدم `forEach` عشان تمر على كل رقم بالقائمة وتنفذ عليه عملية الطباعة.

الميزة فيها إنها بتخليك تنفذ كود معين لكل عنصر بدون ما ترجعك نتيجة نهائية. يعني لو بك بس تطبع، أو تعدل على العناصر، أو أي عملية ثانية لكل عنصر بالمصفوفة، `forEach` بتكون مناسبة. لكن ما بتقدر تستخدمها إذا كنت بك ترجع قيمة معينة بعد ما تخلص من المرور على كل العناصر.

Exame:-

```
let myString = "1,2,3,EE,l,z,e,r,o,_W,e,b,_S,c,h,o,l,2,0,Z";
```

```
let splitString = [];
```

عندك متغير myString بك تلغي الارقام و الفواصل وكلشي تماما وتطبع كلمة ELZERO WEB SCHOOL فقط

اقدرت اوصل لحلين

----- الحل الاول -----

```
let solution = myString.split("").filter(function (ele) {  
  return isNaN(ele); }).map(function (ele) {  
    return ele !== "," ? ele : ""; }).reduce(function (ele, fun) {  
      return ele + fun; }).slice(true, -!false).split("_").join("");  
console.log(solution);
```

-----الشرح-----

تحويل النص إلى حروف فردية الكود يبدأ بتقسيم النص إلى حروف فردية، بحيث كل حرف أو علامة فاصلة يكون عنصر مستقل في قائمة. تصفية العناصر غير الرقمية بعد ما نحصل على الحروف، بنعمل تصفية للعناصر اللي هي أحرف فقط، ونتجاهل الأرقام. إزالة الفواصل بعد التصنيف، بنقوم بإزالة الفواصل (علامات `،`) من بين الأحرف. دمج الأحرف في نص واحد بعد إزالة الفواصل، بنجمع الأحرف كلها في نص واحد. تقسيم النص بناءً على الرمز بنقسم النص الناتج على أساس الرمز ` \_ `، بحيث نحصل على أجزاء مختلفة. دمج الأجزاء بمسافات بعد التقسيم، بنجمع الأجزاء مرة ثانية، ولكن هذه المرة نستخدم مسافات بدلاً من الرمز النص النهائي هو النص الذي يحتوي على الأحرف التي كانت موجودة في النص الأصلي، لكن بدون أرقام وبدون علامات فاصلة، وقسم بناءً على الرمز ` \_ `، وتم استبداله بمسافات.

-----الحل الثاني-----

```
let solution = myString.split("")
.filter(function(ele){
return isNaN(parseInt(ele));})
.map(function(ele){
return ele === "_" || ele === "," ? "" : ele ; })
.slice(!true,-true).reduce(function(ele , fun){
return `${ele}${fun}`
}).slice(true)
console.log(solution)
```

-----الشرح-----

تحويل النص إلى حروف فردية النص ينقسم إلى حروف منفردة وعلامات فاصلة تصفية العناصر لتبقى فقط الأحرف يتم الاحتفاظ بالعناصر التي هي أحرف فقط، ويتم تجاهل الأرقام إزالة الرموز الخاصة (مثل الفواصل والرمز `\_`) يتم استبدال الفواصل والرموز `\_` بنص فارغ، بحيث يتم إزالة هذه الرموز دمج الأحرف في نص واحد يتم دمج الأحرف المتبقية في نص واحد طويل تقطيع النص بناءً على الرموز تم استخدام `slice` لتقطيع النص.

## 68-Object

```
let user ={  
  //Properties  
  theName : "Suhaib",  
  lastName : "omar",  
  //Methods  
  saHello : function(){  
    return `Hello`;  
  },};  
console.log(user.theName);  
console.log(user.lastName);  
console.log(user.saHello());
```

## 69-Object – Bot / Bracket Notation

```
let myVar = "country";  
let user = {  
  theName: "Suhaib",  
  country: "Jordan",  
  "Country of": "Amman",  
};  
console.log(user.theName);  
console.log(user[myVar]);  
console.log(user["Country of"]);
```

بنقدر بنعمل Access على العناصر بهذول الطرق

## 70- Object – Nested Object And Trainings

شرح يكون في object داخل Object ثاني وكيف تعمل Access عليه

```
let user = {  
  name: "Suhaib",  
  age: 23,  
  skills: ["HTML", "CSS", "JS"],  
  available: false,  
  addresses: {  
    kas: "Riyadh",  
    egypt: {  
      one: "Cairo",  
      tow: "Giza",  
    },  
  },  
  checkAc: function () {  
    if (user.available === true) {  
      return `Free For Work`;  
    } else {  
      return `Not free`; } },  
};  
  
console.log(user.name);  
console.log(user.age);  
console.log(user.skills);  
console.log(user.skills.join(" | "));  
console.log(user.addresses.kas);  
console.log(user.addresses.egypt);  
console.log(user.addresses.egypt.one);  
console.log(user["addresses"]);  
console.log(user["addresses"]["egypt"]["one"]);  
console.log(user.checkAc());
```

71-Object – Create With New Keyword New Object



```

a- let user = new Object({
  age: 20,
});
console.log(user);
b-user.age = 23;
c - user["Country"] = "Jordan";
user.sayHello = function () {
  return `Hello`;
};
console.log(user);
console.log(user.age + "\n" + user.Country);
console.log(user["Country"]);
console.log(user.sayHello());

```

أشكال Object في 3 أشكال مشروحه هون

## 72-Object Function This keyword

This تعمل عمل استدعاء المالك لها في الامثله بالاسفل تشرح

```

console.log(this);
console.log(this === window);
myVar = 100;
console.log(window.myVar);
console.log(myVar);
function sayHello() {
  console.log(this);
  return this;
}
sayHello();
console.log(sayHello() === window);

```

```

document.getElementById("cl").onclick = function () {
  console.log(this);
};
let user = {
  age: 38,
  ageInDays: function () {
    return user.age * 356;
  },
};
console.log(user.ageInDays());
console.log(user.age);

```

**this** في السياق العام يشير إلى **window** المتغيرات التي بنعرفها بدون **var** أو **let** بتتضاف مباشرة لكائن **window**. عند استدعاء دالة في السياق العام، **this** يكون **window**. لما تضيف حدث **onclick**، **this** يشير للعنصر الذي ضغط عليه المستخدم. كائن **user** فيه دالة تحسب العمر بالأيام وتضربه بعدد أيام السنة.

### 73-Object – Create Object With Create Method

```

let user = {
  age: 40,
  doubleAge: function () {return user.age * 2;},
};
console.log(user);
console.log(user.age);
console.log(user.doubleAge());
let obj = Object.create({});
obj = 100;
console.log(obj);
let copyObj = Object.create(user);
copyObj.age = 50;
console.log(copyObj);
console.log(copyObj.age);

```

```
console.log(copyObj.doubleAge());
```

الكائن `user` عنده خاصية `age` ودالة `doubleAge`

-قدرنا نعدل العمر في النسخة الجديدة `copyObj` واستفدنا من الدوال والخصائص الموروثة من الكائن `user`.

-لما نطبع `copyObj.doubleAge()` بعد تعديل العمر، رح يعطينا 100 لأن الدالة بتضرب العمر الجديد في 2.

## 74-Object – Create Object With Assign Method

```
let obj1 = {  
  prop1: 1,  
  meth1: function () {  
    return this.prop1;  
  }  
};  
let obj2 = {  
  prop2: 2,  
  meth1: function () {  
    return this.prop2;  
  }  
};  
let targetObject = {  
  prop1: 100,  
  prop3: 3,  
};  
let finalObject = Object.assign(targetObject, obj1, obj2);  
finalObject.prop1 = 200;  
finalObject.prop4 = 4;  
console.log(finalObject);  
let newobject = Object.assign({}, obj1, { prop5: 5, prop6: 6 });  
console.log(newobject);
```



# DOM – JAVASCRIPT

ELZERO WEB SCHOOL

VIDEO (86)

## 75-What Is DOM

DOM Selectors / Find Element / title / body / images /....

الـ DOM في جافاسكربت هو الطريقة التي بتقدر من خلالها تتحكم بمحتوى صفحة الويب. لما الصفحة تتعرض، المتصفح يحول الـ HTML لشكل شجري (Tree)، وكل عنصر على الصفحة مثل العناوين، الصور، والأزرار بيصير عقدة (Node) داخل هاي الشجرة.

بجافاسكربت، بتقدر تستخدم الـ DOM عشان:

تعدل على العناصر \*\*: زي إنك تغير النص اللي في عنصر معين أو تضيف/تحدف عناصر جديدة.

تتحكم بالستايل \*\*: تغير ألوان، حجم، أو شكل العناصر.

تتعامل مع الأحداث \*\*: زي لما المستخدم يضغط زر، تغير إشي بناءً على الحدث.

مثلاً:

- إذا بدك تغير محتوى عنصر معين، بتستعمل `document.getElementById("id").innerHTML = "نص جديد";`.

باختصار، الـ DOM بيخليك تتحكم وتتفاعل مع صفحة الويب ديناميكياً.

```
let myIdElement = document.getElementById("ID");
let myTagElements = document.getElementsByTagName("p");
let myClassElement = document.getElementsByClassName("CLASS ");
let myQueryElementclass = document.querySelector(".CLASS ");
let myQueryElementId = document.querySelector("#ID");
let myQueryElementall = document.querySelectorAll(".CLASS / ID ");
console.log(myIdElement);
console.log(myTagElements[1]);
console.log(myClassElement[0]);
console.log(myQueryElementclass);
console.log(myQueryElementall[1]);
console.log(document.title);
console.log(document.body);
console.log(document.forms[0].one.value);
console.log(document.links[1].href);
```

## 76-DOM Get Set Elements Content And Attributes → innerText

```
let myElemnt = document.querySelector(".js");

console.log(myElemnt.innerHTML);
console.log(myElemnt.textContent);

myElemnt.innerHTML = "Text From <span>Main.js </span> File";
myElemnt.textContent = "Text From <span>Main.js </span> File";
document.images[0].src = "https://google.com";
document.images[0].alt = "Alternate";
document.images[0].title = "Alternate";
document.images[0].id = "pic";
document.images[0].className = "pic";
let myLink = document.querySelector(".li");
console.log(myLink.getAttribute("class"));
console.log(myLink.getAttribute("href"));
myLink.setAttribute("href", "https://twitter.com");
myLink.setAttribute("title", "https://twitter.com");
```

هذا الكود يستخدم الـ DOM عشان يتعامل مع عناصر الصفحة، يعدل المحتوى، ويغير خصائص العناصر زي الصور والروابط.

## 77-DOM → Check Attributes

```
console.log(document.getElementsByTagName("p")[0].attributes);
```

اعطيني الخصائص العنصر المراد

```
let myP = document.getElementsByTagName("p")[0];
```

```
if (myP.hasAttribute("data-src")) {
```

الدالة hasAttribute هي دالة في جافاسكريبت بتفحص إذا العنصر فيه Attribute معين أو لا.

```
if (myP.getAttribute("data-src") === "") {
```

ال دالة getAttribute في جافاسكريبت بتستخدم عشان تجيب قيمة Attribute معين

```
myP.removeAttribute("data-src");
```

ال دالة removeAttribute في جافاسكريبت بتستخدم عشان تشيل (ت حذف) Attribute معين

```
} else [myP.setAttribute("data-src", "New value")];
```

الدالة setAttribute في جافاسكريبت بتستخدم لإضافة أو تعديل Attribute

```
} else { console.log("Not Found") }
```

## 78-DOM → Create Elements

```
let myElement = document.createElement("div");
```

الدالة createElement في جافاسكريبت بتستخدم عشان تنشئ عنصر

```
let myAttr = document.createAttribute("data-custom");
```

الدالة createAttribute في جافاسكريبت بتستخدم عشان تنشئ Attribute جديد لعناصر HTML بشكل ديناميكي.

```
let myText = document.createTextNode("Product One");
```

انشاء نص عادي من غير عنصر html

```
let myComment = document.createComment("This is Div");
```

انشاء تعليق

```
myElement.className = "Product";
```

```
myElement.setAttributeNode(myAttr);
```

دالة `setAttributeNode` في جافاسكريبت تستخدم لإضافة `Attribute` (سمة) تم إنشاؤها باستخدام دالة `createAttribute` إلى عنصر HTML معين.

```
myElement.setAttribute("data-text", "Texting");
```

```
myElement.appendChild(myText);
```

في JavaScript، `appendChild` هو دالة تُستخدم لإضافة عنصر جديد كطفل لعنصر موجود في DOM

```
document.body.appendChild(myComment);
```

فأنت تقوم بإضافة عنصر جديد إلى نهاية عنصر `<body>` في صفحة الويب.

```
document.body.appendChild(myElement);
```

## 79- DOM → Create Elements → Practice Product With Heading And Paragraph

```
for (let i = 0; i <= 100; i++) {  
  let myMainelement = document.createElement("div");  
  let myHeading = document.createElement("h3");  
  let myParagraph = document.createElement("p");  
  let myHeadingText = document.createTextNode(`Product Title ${i}`);  
  let myParagraphText = document.createTextNode("Product Description");  
  let myLien = document.createElement("hr");  
  myHeading.appendChild(myHeadingText);  
  myMainelement.appendChild(myHeading);  
  myParagraph.appendChild(myParagraphText);  
  myMainelement.appendChild(myParagraph);  
  myMainelement.appendChild(myLien);  
  myMainelement.className = `product ${i}`;  
  document.body.appendChild(myMainelement);  
}
```

-----الشرح-----



الكود اللي عندك بيعمل تكرار 100 مرة\*\*، وفي كل مرة بيعمل الآتي:

1. بانشئ عنصر `div` جديد، اللي هو الحاوية الرئيسية لكل "منتج".
  2. بانشئ عنصر `h3` عشان يكون العنوان الرئيسي للمنتج.
  3. بانشئ عنصر `p` عشان يحط فيه وصف المنتج.
  4. بانشئ نص للعنوان بعنوان "Product Title" متبوع برقم المنتج اللي هو `i`، واللي هو رقم التكرار (من 0 إلى 100).
  5. بانشئ نص للوصف عبارة عن "Product Description".
  6. بضيف كل نص للعنصر اللي بيناسبه، يعني العنوان بـ `h3`، والوصف بـ `p`.
  7. بعد ما تجهز الـ `h3` والـ `p`، بضيفهم داخل الـ `div` الرئيسي.
  8. بضيف خط أفقي (`hr`) بين المنتجات للتفريق بينهم.
  9. بيعطي للـ `div` الرئيسي كلاس خاص فيه على شكل `product` مع رقم المنتج اللي هو `i`.
  10. وأخيراً، بضيف كل `div` على الصفحة باستخدام `appendChild`، وهيك المنتجات بتعرض على الشاشة.
- بالنهاية، الكود بيعمل 100 منتج، كل واحد منهم فيه عنوان، وصف، وخط أفقي للتفريق.

## 80- DOM → Deal With Childrens

```
let myElement = document.querySelector("div");
```

```
console.log(myElement);
```

```
console.log(myElement.children);
```

الابناء = Element + Text

```
console.log(myElement.children[0]);
```

```
console.log(myElement.childNodes);
```

الابناء = Text

```
console.log(myElement.childNodes[0]);
```

```
console.log(myElement.firstChild);
```

أول ابن سواء Element أو text

```
console.log(myElement.lastChild);
```

```
console.log(myElement.firstChild);
```

Element = اول ابن

```
console.log(myElement.lastElementChild);
```

Element = اخر ابن

## 81-DOM → Events

```
<button id="btn" onclick="console.log(`10`)">Button</button>
```

```
let myBtn = document.getElementById("btn");
```

```
myBtn.onclick = function () {
```

```
    console.log("text");
```

```
};
```

```
window.onscroll = function () {
```

```
    console.log("scroll");
```

```
};
```

- onclick
- oncontextmenu
- onmouseenter
- onmouseleave
- onload
- onscroll
- onresize
- onfocus
- onblur
- onsubmit

بعض الخصائص التي بتصير لما المستخدم يقوم بعملية معينة على الصفحة

## 82-DOM Events

```
let userInput = document.querySelector("[name = 'username']");
```

```
let ageInput = document.querySelector("[name = 'age']");
```

بجيب المدخلات لاسم المستخدم و العمر

```
document.forms[0].onsubmit = function e {
```

```
    let userValid = false;
```

```
    let ageValid = false;
```

لما نضغط على زر الارسال نفذ التالي وعرفنا متغيرين الشغل كلو عليهم

```
    if (userInput.value !== "" && userInput.value.length <= 10) {
```

```
        userValid = true;
```

```
    }
```

إذا Input مش فاضي ولا قيمته اكبر من 10 قيمة متغير الاسم true

```
    if (ageInput.value !== "") {
```

```
        ageValid = true;
```

```
    }
```

إذا Input مش فاضي خلي متغير العمر true

```
    if (userValid === false || ageValid === false) {
```

```
        e.preventDefault();
```

```
    } };
```

إذا واحد من المتغيرين ما اخذوا القيم الصحيحة الغي عمل زر الارسال عن طريق preventDefault()

```
document.links[0].onclick = function (event) {
```

```
    console.log(event);
```

```
    event.preventDefault();
```

```
};
```

هاذ زر link لما تضغط عليه جيب معلوماته والغى كبستوا الافتراضيه

----- الشرح -----

الكود اللي عندك بعمل فحص على مدخلات المستخدم قبل ما يسمح بإرسال النموذج (form)، ويمنع الرابط الأول في الصفحة إنه يشتغل لما تنقر عليه.

### 1. المدخلات (inputs):

- أول سطرين بجيبوا المدخلات من الـ HTML، وحدة لاسم المستخدم (`username`) والثانية للعمر (`age`) باستخدام `querySelector`. هذول هم الحقول اللي رح نتحقق منهم.

### 2. فحص النموذج (form validation):

- لما المستخدم يضغط على زر الإرسال (submit)، بنفذ الحدث `onsubmit` على النموذج (form).
- بيعمل فحص أول إشي على مدخلات اسم المستخدم (`userInput`):
- إذا الاسم مش فاضي وطوله أقل أو يساوي 10 حروف، بنعتبره صحيح (بتكون قيمة `userValid` صارت `true`).
- بعدين بفحص مدخل العمر (`ageInput`):
- إذا مش فاضي، بنعتبره صحيح (`ageValid` بتصير `true`).
- إذا واحد من الفحوصات كان غلط (سواء الاسم أو العمر)، بيمنع النموذج إنه ينرسل باستخدام `e.preventDefault`.

### 3. الرابط (link):

- الكود بحدد إنه لما تنقر على أول رابط في الصفحة (`document.links[0]`)، الحدث `onclick` بتنفذ دالة اللي بطلعك معلومات عن الحدث نفسه (بالكونسول).
- وأيضًا بيمنع الرابط إنه يفتح الصفحة اللي الرابط بيودي إليها باستخدام `event.preventDefault`، يعني الرابط ما رح يشتغل لما تنقر عليه.

باختصار:

الكود بيتأكد إنك تدخل اسم مستخدم صحيح (مش فاضي وأقل من 10 حروف) وعمر مش فاضي قبل ما ينرسل النموذج، وإذا الفحص فشل، النموذج ما بينرسل. وكمان الرابط الأول في الصفحة بينعمله تعطيل وما يشتغل لما تضغط عليه.

### 83- DOM → Events Simulation

Click

Focus

Blur

```
let tow = document.querySelector(".tow");  
let one = document.querySelector(".one");  
window.onload = function () {  
  tow.focus();  
  one.onblur = function () {  
    document.links[0].click();  
  };  
};
```

الكود اللي عندك بيعمل الآتي:

تحديد العنصرين

بيجيب عنصرين من الصفحة باستخدام `querySelector` :

**tow**: العنصر اللي عنده الكلاس "tow".

**one**: العنصر اللي عنده الكلاس "one".

**window.onload**

هذا الحدث بيشتغل لما الصفحة كلها تخلص تحميل.

في الحدث هذا، بمجرد ما الصفحة تنفتح وتخلص تحميل، الفوكس (focus) بينحط مباشرة على العنصر اللي فيه الكلاس "tow"، يعني إا كان هذا العنصر هو input، رح يصير جاهز للكتابة بمجرد فتح الصفحة.

**one.onblur**

هذا الحدث بيصير لما المستخدم يطلع من العنصر "one" (يعني يفقد العنصر الفوكس).

بمجرد ما المستخدم يطلع من العنصر اللي عنده الكلاس "one"، الكود بيعمل محاكاة لنقرة على أول رابط بالصفحة باستخدام document.links[0].click() يعني لو كان عندك رابط بالصفحة، رح ينضغط بشكل تلقائي لما العنصر "one" يفقد الفوكس.

أول ما الصفحة تنفتح، الفوكس بينحط على العنصر "tow".

وإذا طلعت من العنصر "one"، أول رابط بالصفحة بينضغط بشكل تلقائي.

## 84-DOM → Class List

```
let element = document.getElementById("my-div");
```

```
console.log(element.classList);
```

بعطيك جمعي classes الموجوده

```
console.log(element.classList.contains("show"));
```

```
console.log(element.classList.contains("Suhaib"));
```

إذا class موجود بعطيك صح او غلط إذا مش موجود

```
console.log(element.classList.item("3"));
```

بعطيك class رقم 3

```
element.onclick = function () {
```

```
    element.classList.add("add-one", "add-two");
```

بضيف class

```
    element.classList.remove("one", "two");
```

بحذف class

```
    element.classList.toggle("ss");
```

إذا class موجود بحذفو وإذا مش موجود بضيفو

```
};
```

## 85- DOM → CSS

```
let element = document.getElementById("my-div");
```

```
element.style.color = "red";
```

```
element.style.fontWeight = "bold";
```

```
element.style.cssText = "font-weight : bold ; color : green ; opacity : 0.9";
```

```
element.style.removeProperty("color");
```

```
element.style.setProperty("font-size", "40px", "important");
```

## 86-DOM → Deal With Elements → (before – After – append – prepend – remove )

```
let element = document.getElementById("my-div");
```

```
let createdP = document.createElement("p");
```

```
element.before("hello JS");
```

إضافة عنصر بعد العنصر الاساسي

```
element.after("hello JS");
```

إضافة عنصر قبل العنصر الاساسي

```
element.before(createdP);
```

```
element.append(createdP);
```

داخل العنصر

```
element.prepend(createdP);
```

```
element.remove();
```

حذف العنصر

## 87-DOM → Traversing

```
let span = document.querySelector(".tow");
```

```
console.log(span.nextElementSibling);
```

العنصر التالي مباشرة

```
console.log(span.previousElementSibling);
```

العنصر الذي يكون قبل العنصر مباشرة

```
console.log(span.parentElement);
```

الاب

```
span.onclick = function(){  
  span.parentElement.remove()  
}
```

## 88-DOM → Cloning (CloneNode (Deep));

استنساخ العنصر اذا كان بين القوسين true بنسخ العنصر مع المحتوى الى بداخله  
اذا كان فاضي او false بنسخ العنصر فقط

```
let myP = document.querySelector("p").cloneNode(true)
```

```
let myDiv = document.querySelector("div")
```

```
myP.id = `${myP.id}-clone`
```

استنساخ العنصر مع محتواه وغيرنا id عشان ما يصير اغلاط عنا

```
myDiv.appendChild(myP)
```

## 89-DOM → Add Event Listener

تقدر تضيف أكثر من حدث لنفس العنصر بدون ما تلغي أحداث ثانية. او تعطي لعنصر حدث هو لسا مش موجود

```
let myP = document.querySelector("p");
```

```
myP.onclick = function () {
```

```
  let newP = myP.cloneNode(true);
```

```
  newP.className = "clone";
```

```
  document.body.appendChild(newP);
```

```
};
```

```
document.addEventListener("click", function (e) {
```

```
  if (e.target.className === "clone") {
```

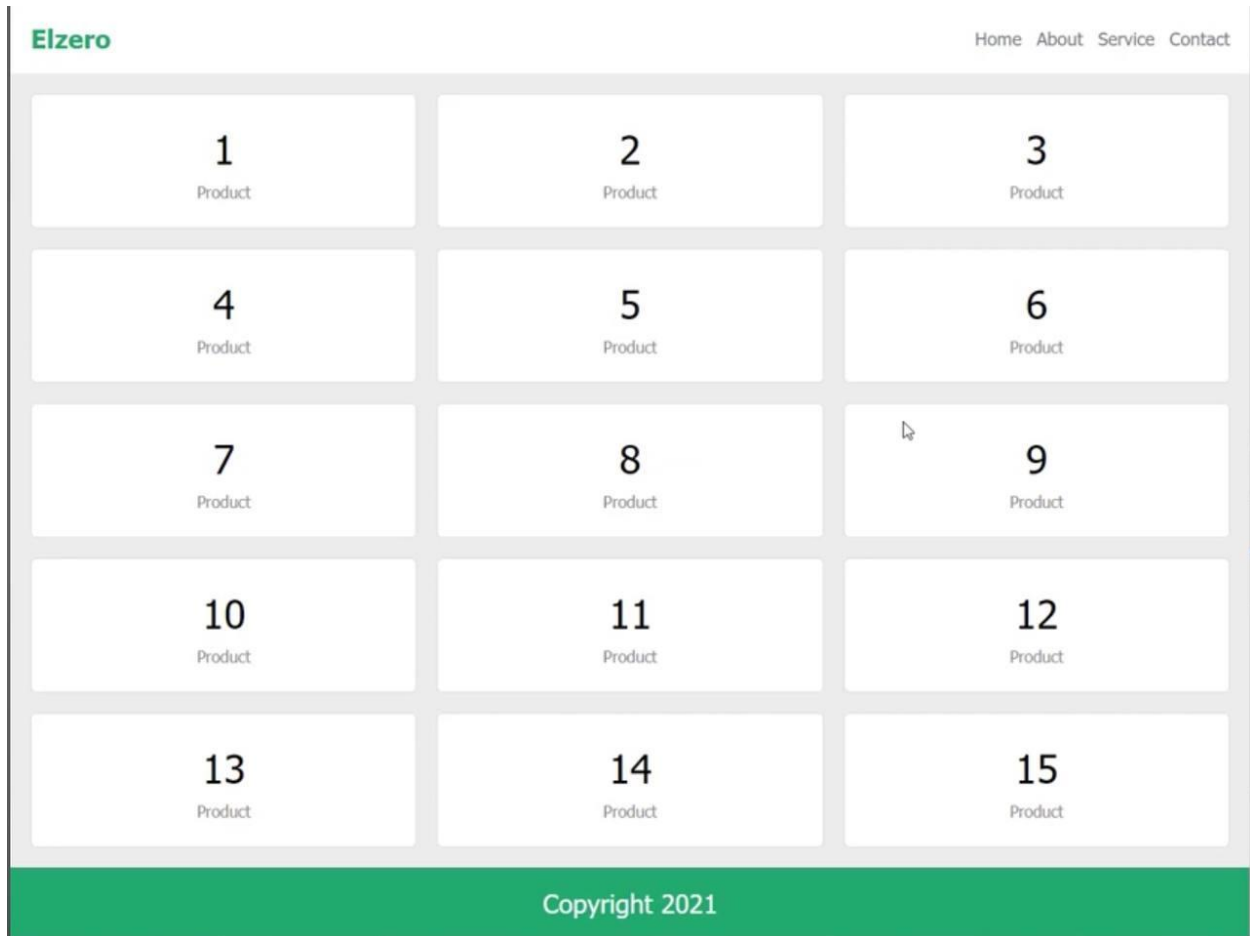
```
    console.log("Iam Cloned");
```

```
  } });
```



Exame:-

-----Video 101-----



← بذك تصميم نفس هاذ التصميم بدون ما تكتب اي اشي داخل ملف html او css فقط اكواد Javascript

حلي ولكن طوييل

```
document.body.style.cssText =  
  "background-color:#c5c5c5 ; padding : 0 ; margin : 0";  
  
/* Start Create Header */  
  
let header = document.createElement("div");  
header.className = "header";  
header.style.cssText =  
  "background-color:#ffffff ; height: 60px ; width: 100%;display: flex; place-items: center;justify-content:  
space-between;flex-direction: row-reverse;";
```

```

/* Start Create Logo Inside Header */

let logo = document.createElement("h3");
let textLogo = document.createTextNode("Elzero");
logo.appendChild(textLogo);
logo.style.cssText =
  "font-size: 40px; margin: 0 15px; color: green; font-weight: 900;";

/* End Create Logo Inside Header*/

/* Start Mune Inside Header */

let muneUi = document.createElement("ul");
muneUi.style.cssText =
  "list-style: none; display: flex; justify-content: space-between;";

let muneLi = document.createElement("li");
let muneLiText = document.createTextNode("Home");
muneLi.appendChild(muneLiText);
muneUi.appendChild(muneLi);
muneLi.className = "muneli";
muneLi.style.cssText =
  "color:rgb(117 117 117) ; cursor: pointer; padding: 0px 5px;";

let muneli1 = muneLi.cloneNode();
muneUi.appendChild(muneli1);
let muneLi1Text = document.createTextNode("About");
muneli1.appendChild(muneLi1Text);

let muneli2 = muneLi.cloneNode();
muneUi.appendChild(muneli2);
let muneLi2Text = document.createTextNode("Service");
muneli2.appendChild(muneLi2Text);

let muneli3 = muneLi.cloneNode();
muneUi.appendChild(muneli3);
let muneLi3Text = document.createTextNode("Contact");
muneli3.appendChild(muneLi3Text);

header.appendChild(muneUi);
header.appendChild(logo);
document.body.appendChild(header);

/* End Mune Inside Header */

/* End Create Header */

/* Start Create Product */

```

```

/* Start Create Product Div */
let productDiv = document.createElement("div");
productDiv.style.cssText =
  "width: auto; height: 700px ;margin: 15px 10px; display: grid; grid-template-columns: auto auto auto;
  justify-content: center;gap: 15px;";
/* End Create Product Div */
/* Start Create Product Card */

for (let i = 1; i <= 15; i++) {
  let productCard = document.createElement("div");
  productDiv.appendChild(productCard);
  productCard.style.cssText =
    "width: 425px; height: 125px; background-color: #ffffff; text-align:center";
  let productText = document.createElement("h3");
  let productPrg = document.createTextNode("product");
  productText.style.cssText = "opacity: 0.7;";

  productText.appendChild(productPrg);

  let productNumber = document.createElement("h1");
  let productNum1 = document.createTextNode(`${i}`);
  productNumber.appendChild(productNum1);
  let productAll = document.createElement("div");
  productAll.appendChild(productNumber);
  productAll.appendChild(productText);

  productCard.appendChild(productAll);
}

/* End Create Product Card */

document.body.appendChild(productDiv);
/* End Create Product */

/* Start Create Footer */
let footer = document.createElement("div");
let footerH = document.createElement("h3");

let footerHText = document.createTextNode("Copyright 2021");
footerH.appendChild(footerHText);

footer.style.cssText =
  "background-color: green;width: 100%;height: 50px;display: flex;justify-content: center;align-items:
  center;color: white;font-weight: 900;";
footer.appendChild(footerH);
document.body.appendChild(footer);
/* End Create Footer */

```

## -----الشرح-----

هالكود عبارة عن سكربت جافاسكربت لإنشاء صفحة ويب تحتوي على هيدر، قائمة من المنتجات وفوتر، وكل هالشئ معمول ديناميكياً بدون الحاجة لكتابة HTML مباشرة.

- أول إشي بيحدد لون خلفية الصفحة ويشيل الـ padding والـ margin منها.

- بعدين بنبدأ نعمل الهيدر (header)، اللي هو عبارة عن div فيه خاصية flex لتنسيق العناصر بداخله، متضمنين اللوجو "Elzero" والقائمة.

- اللوجو عبارة عن عنصر `<h3>` باللون الأخضر والخط السميك.

- القائمة عبارة عن قائمة غير مرتبة (`<ul>` - `unordered list`) فيها أربع عناصر ( Home, About, Service, Contact )، وكل عنصر من العناصر معمول باستخدام حلقة تكرار (clone) للكود الأساسي.

- القسم الثاني بيعمل div فيه 15 كرت للمنتجات، كل كرت فيه رقم المنتج وكلمة "product".

- بالنهاية الفوتر (footer) فيه حقوق الطبع، معمول أيضاً باستخدام div ونفس مبدأ الـ flex لتوسيط المحتوى فيه.

باختصار، الكود بيبنى الصفحة بشكل كامل ديناميكياً باستخدام JavaScript وعملية تنسيق المحتوى معمولية باستخدام CSS عبر الـ JavaScript مباشرة.

A yellow square containing the letters 'JS' in a bold, black, sans-serif font.

# Browser Object Model(BOM)

## BOM – JAVASCRIPT

ELZERO WEB SCHOOL

VIDEO (102)

## 90-BOM → Browser Object Model

BOM أو Browser Object Model في جافا سكريبت هو مجموعة من الكائنات (Objects) التي تتيح لك التعامل مع المتصفح نفسه. يعني من خلال الـ BOM يمكنك التحكم بأشياء مثل النافذة (window) ، العنوان (URL) ، وسجل التصفح (history).

ما الذي يمكنك التحكم به باستخدام الـ BOM ؟

window هو الكائن الأساسي الذي يحتوي على كل شيء داخل المتصفح. يحتوي على خصائص مثل:

alert() لعرض رسائل للمستخدم.

setTimeout() setInterval() لتشغيل كود بعد فترة زمنية أو بشكل متكرر.

document يمثل الصفحة نفسها ويستخدم للتحكم بالعناصر الموجودة داخلها (يرتبط بالـ DOM).

navigator يتيح لك معرفة معلومات عن المتصفح الذي يستخدمه المستخدم (مثل نوع المتصفح وإصداره).

location يستخدم للتحكم بالعنوان (URL) الحالي، مثل تغيير الصفحة أو إعادة تحميلها.

history يتيح لك التنقل بين الصفحات السابقة والمستقبلية التي تمت زيارتها.

باختصار:

الـ BOM هو الذي يسمح لك بالتحكم في المتصفح نفسه باستخدام جافا سكريبت، مثل فتح نافذة جديدة، تغيير العنوان (URL) ، أو عرض رسالة للمستخدم.

## 91-BOM → Alert --- Confirm --- Prompt

```
alert("Suhaib");
```

```
let confirmMsg = confirm("Are You Sure?");
```

```
console.log(confirmMsg);
```

```
if (confirmMsg === true) {
```

```
    console.log("Item Deleted");
```

```
} else { console.log("Item Not Deleted"); }
```

```
let confirmMsg = prompt("Good Day To You?" , "Write Day With 3 Characters");
```

```
console.log(confirmMsg);
```

## 92-BOM → setTimeout , clearTimeout

```
setTimeout(function sayMsg() {  
  console.log(`I am Message`);  
}, 3000);  
  
setTimeout(sayMsg, 3000);  
  
function sayMsg() {  
  console.log(`I am Message`); }  
}
```

دالة `setTimeout` في جافاسكربت تُستخدم لتأخير تنفيذ كود أو دالة معينة بعد فترة زمنية محددة (بالملي ثانية). يعني إذا بدك تشغل كود معين بعد وقت معين، بتستخدمها.

```
let counter = setTimeout(sayMsg, 3000);  
  
function sayMsg(user, age) {  
  console.log(`I am Message`); }  
  
let btn = document.querySelector("button");  
  
btn.onclick = function () { clearTimeout(counter); };
```

دالة `clearTimeout` بتستخدم لإلغاء مؤقت (`timer`) تم تعيينه باستخدام `setTimeout`. يعني إذا قررت إنك ما بدك الكود يتنفذ بعد ما تعيينته

## 93-BOM → setInterval , clearInterval

دالة `setInterval` في جافاسكربت بتستخدم لتشغيل كود أو دالة بشكل متكرر بعد فترة زمنية محددة (بالملي ثانية). يعني إذا بدك تشغل كود بشكل دوري كل فترة معينة، بتستخدمها، دالة `clearInterval` بتستخدم لإيقاف تكرار الكود اللي تم تشغيله بواسطة `setInterval`. يعني إذا بدك توقف المؤقت المتكرر بعد فترة، بتستخدمها.

```
setInterval(function() { console.log("SSuhaib") }, 1000)
```

المثال الي تحت حطينا `div` فيه رقم 5 كل ثانيه بنقص واحد

```
let div = document.querySelector("div");  
  
function countdown() { div.innerHTML -= 1;  
  
  if (div.innerHTML === "0") { clearInterval(counter); } }  
  
let counter = setInterval(countdown, 1000);
```

#### 94-BOM → Location Object (href Get / Set ..... / host / hash / protocol / reload() / Replace() / assign() )

```
console.log(location);
console.log(location.href);
location.href = "https://google.com";
location.href = "/#sec02";
console.log(`host : ` + location.host);
console.log(`hostname : ` + location.hostname);
console.log(location.protocol);
console.log(location.hash);
location.reload();
location.replace("https://google.com");
location.assign("https://google.com");
```

#### 95-BOM → Open Window

```
setTimeout(function(){
    window.open("https://google.com" , "_blank" , "Width=400 , height=400 , left=200 , top=350")
},2000)
```

اللي مكتوب هو كود JavaScript باستخدام الدالة `setTimeout` عشان يفتح نافذة جديدة (توجهك على رابط Google بعد مرور 2 ثانية). هيك الكود رح يفتح نافذة جديدة بعرض 400px وارتفاع 400px، وحيكون مكانها على بعد 200px من اليسار و350px من الأعلى بعد مرور 2 ثانية.

#### 96-BOM → History API



97-BOM → Stop , print , focus , scroll

```
window.focus()
```

```
window.print()
```

```
window.scrollBy(500 , 600)
```

```
window.scrollTo(500 , 600)
```

```
window.scrollTo({
```

```
    left : 500,
```

```
    top : 200,
```

```
    behavior:"smooth"})
```

98-BOM → ScrollX , scrollY

```
Scrollx === PageXOffset
```

```
Scrolly === PageYOffset
```

```
console.log(window.scrollX);
```

```
console.log(window.scrollY);
```

المثال الي تحت لما المستخدم ينزل تحت 600px يظهر زر يرجع اول الصفحة

```
let btn = document.querySelector("button");
```

```
window.onscroll = function () {
```

```
    if (window.scrollY >= 600) {
```

```
        btn.style.display = "block";
```

```
    } else {
```

```
        btn.style.display = "none"; } };
```

```
btn.onclick = function () {
```

```
    window.scrollTo({
```

```
        top: 0,
```

```
        left: 0,
```

```
        behavior: "smooth", }); };
```

## 99-BOM → Local Storage

هنا الكود يستخدم `localStorage` عشان يخزن، يرجع، ويمسح بيانات بسيطة في المتصفح، وهي بتظل محفوظة حتى لو سكرت المتصفح أو أعدت تشغيل الجهاز.

### //Set

```
window.localStorage.setItem("color" , "red")
```

```
window.localStorage.fontWeight = "bold"
```

```
window.localStorage["font-size"] = "20px";
```

تخزين بيانات: من خلال الـ `localStorage` بتقدر تخزن بيانات زي لون الخلفية أو حجم الخط على شكل مفتاح وقيمة. وهي البيانات بتظل موجودة حتى لو سكرت المتصفح.

### //Get

```
console.log(window.localStorage.getItem("color"))
```

```
window.localStorage.color
```

```
window.localStorage["color"]
```

استرجاع بيانات: بعد ما تخزن البيانات، بتقدر ترجعها بأي وقت باستخدام اسم المفتاح اللي خزنت تحته القيمة

### //Remove

```
window.localStorage.removeItem("color")
```

حذف بيانات: إذا بدك تمسح بيانات معينة، بتحدد المفتاح اللي بدك تحذفه وتتحذفه من الـ `localStorage`

### //Remove All

```
window.localStorage.clear();
```

مسح كل البيانات: إذا بدك تمسح كل شيء مخزن في الـ `localStorage` ، بتقدر تمسحه دفعة واحدة.

### //Get Key

```
console.log(window.localStorage.key(0));
```

الحصول على مفتاح محدد: بتقدر تجيب أول مفتاح مخزن في الـ `localStorage` ، إذا عندك أكثر من عنصر مخزن.

### //Set Color In Page

```
document.body.style.background = window.localStorage.getItem("color")
```

```
console.log(window.localStorage)
```

تغيير شكل الصفحة: إذا كنت مخزن لون مثلاً، بتقدر تستخدمه لتغيير لون الخلفية أو أي شيء ثاني في الصفحة بناءً على القيمة المخزنة.

الهدف من هاد كله إنك تقدر تخلي المتصفح يتذكر أشياء بسيطة عن المستخدم حتى لو رجع بعد فترة

## 100-BOM → Session Storage

**sessionStorage** في جافاسكربت بتشغل زي **localStorage** بس الفرق إنو البيانات اللي بتخزنها بتظل موجودة طول ما الصفحة أو التبويبة مفتوحة. يعني أول ما تسكر الصفحة أو تعمل إعادة تشغيل (refresh)، يتم مسح البيانات.

متى بنستخدمها؟

بنستخدم **sessionStorage** لما بدك تخزين معلومات مؤقتة للمستخدم أثناء الجلسة الحالية. مثلاً، إذا عندك فورم وبدك تخزين بيانات المستخدم لحد ما يكمل، وبعد هيك ما بدك تحتفظ فيها.

مثال بسيط:

تخيل إنك فاتح صفحة وفيها فورم. كتبت بياناتك، فالموقع بيخزن هالبيانات مؤقتاً باستخدام **sessionStorage** عشان لو تنقلت بين الصفحات أو عملت أي شيء ما تضيع، لكن أول ما تسكر الصفحة، يتم مسح.

إذا بدك تخزين اسم المستخدم مثلاً:

بتكتب اسمه وتخزنه في `sessionStorage`

بعدين تقدر تجيب اسمه وتعرضه في الصفحة.

لكن أول ما تسكر الصفحة أو التبويبة، الاسم بينمسخ وما يرجع.

ملخص: **sessionStorage** بتستخدم لتخزين بيانات مؤقتة يتم مسح تلقائياً أول ما تسكر الصفحة.

```
window.sessionStorage.setItem("color", "blue");
```

Exame:-

-----Video 114-----

```
let input = document.querySelector(".input");
let add = document.querySelector(".add");
let tasks = document.querySelector(".tasks");
function insertDelete(val) {
let task = document.createElement("div");
let del = document.createElement("span");
del.classList.add("delete");
task.classList.add("task");
del.addEventListener("click", function (e) {
e.currentTarget.parentElement.remove();
let index = tasksList.indexOf(
e.currentTarget.parentElement.innerText.slice(0, -6));
tasksList.splice(index, 1);
localStorage.setItem("items", JSON.stringify(tasksList));});
del.innerText = "delete";
let taskTxt = document.createTextNode(val);
task.append(taskTxt);
task.append(del);
tasks.append(task);}
if (localStorage.getItem("items")) {
var tasksList = JSON.parse(localStorage.getItem("items"));
for (let i = 0; i < tasksList.length; i++) { insertDelete(tasksList[i]); } } else { var tasksList = []; }
add.onclick = function () {
if (input.value.trim() !== "") {
insertDelete(input.value);
tasksList.push(input.value);
localStorage.setItem("items", JSON.stringify(tasksList));
input.value = ""; } };
```

## -----الشرح-----

هاد الكود عبارة عن تطبيق بسيط لـ "to-do list" (قائمة مهام) باستخدام **HTML**، **CSS**، و **JavaScript**. الفكرة إنك بتقدر تضيف مهام جديدة للقائمة، وتقدر تحذف المهام، والمهام بتتخفظ باستخدام **localStorage** عشان تظل موجودة حتى لو سكرت الصفحة ورجعتها.

كيف يشتغل الكود؟

1. **\*\*اختيار العناصر من الصفحة\*\***:

- أول إشي، الكود يختار العناصر اللي إلهها علاقة بإدخال المهام، زر الإضافة، ومكان عرض المهام. زي: حقل الإدخال، الزر اللي بيضيف المهام، والمكان اللي بيظهر فيه المهام.

2. **\*\*الدالة الإضافة والحذف (insertDelete)\*\***:

- هاي الدالة بتنشئ عنصر جديد للمهمة، مع زر "delete" لكل مهمة عشان تقدر تحذفها.

- لما تضغط على زر الحذف (delete)، المهمة بتتمسح من الصفحة وكمان من **localStorage**.

3. **\*\*جلب المهام من localStorage\*\***:

- الكود بفحص إذا فيه بيانات محفوظة مسبقاً في **localStorage** (إذا كان فيه مهام قديمة مخزنة).

- إذا لقي مهام، بيرجع يعرضها في الصفحة باستخدام نفس الدالة اللي بتحط المهام.

4. **\*\*إضافة مهمة جديدة\*\***:

- لما تضغط على زر الإضافة (add button)، الكود بفحص إذا الحقل مش فاضي.

- إذا فيه نص مكتوب، بضيف المهمة للقائمة ويحفظها في **localStorage**، وبعدين بفضي الحقل.

شو بصير لما تحذف أو تضيف مهمة؟

- لما تضيف مهمة، الكود بيخزنها في **localStorage**، ولما تحذفها بتتحذف كمان من **localStorage** عشان ما تظل تظهر لو سكرت الصفحة ورجعت.

الفكرة الرئيسية:

الكود يشتغل على إنه يخزن البيانات في المتصفح عشان حتى لو سكرت الصفحة ورجعتها بعدين، تظل المهام موجودة، وبتقدر تضيف أو تحذف بدون ما تفقد البيانات.

-----END BOM-----

## 101- Destructuring Array

الكود هذا يستخدم مفهوم "destructuring" في جافاسكربت، والتي بفكك عناصر المصفوفة وبوزعهم على متغيرات معينة.

```
let a = 1;
let b = 2;
let c = 3;
let d = 4;
let myFriends = ["suhaib", "sayed", "Ali", "Omar"];
[a, b, c, d, e = "Ahmad"] = myFriends;
console.log(a);
console.log(b);
console.log(c);
console.log(d);
console.log(e);
```

عندك مصفوفة فيها أسماء، وانت يدك توزع الأسماء على متغيرات مثل a و b و c وغيرها.

كل متغير رح ياخذ قيمة من المصفوفة بالترتيب مثلا: المتغير رح ياخذ الاسم الأول الاسم الثاني وهكذا

إذا عدد المتغيرات أكثر من عدد الأسماء بالمصفوفة، فالمتغير اللي ماله قيمة من المصفوفة رح ياخذ قيمة افتراضية (زي ما عملنا مع e وحطيناله اسم "Ahmad")

ولو كان المتغير مثل e ما الو قيمة بنحطلوا undefined

```
let [x, y, , z] = myFriends;
console.log(x);
console.log(y);
console.log(z);
```

فس الفكرة، لكن هون بنوزع بعض الأسماء فقط، وبنجاوز بعض العناصر من المصفوفة. مثلا: بنوزع الاسم الأول والثاني والأخير بس، والباقي ما بهمنا. ولو بدنا نأخذ الاسم الأول والثاني والرابع بنحط مكان الثالث فراغ زي ما هو بين معنا فوق

## 102 -Destructuring Array → Advanced Examples

```
let myFriends = [  
  "Ahmed",  
  "Sayed",  
  "Ali",  
  ["Shady", "Amr", ["Mohamed", "Gamal"]],  
];  
  
// console.log(myFriends[3][2][1])
```

```
let [, , [a, [b]]] = myFriends;  
console.log(a);
```

أعطيني هون أسم Shady

```
console.log(b);
```

وهون اسم Gamal

هذا الكود يستخدم destructuring بطريقة متقدمة عشان يوصل لعناصر موجودة داخل مصفوفة متداخلة (يعني مصفوفة داخل مصفوفة).

الشرح:

المصفوفة `myFriends` فيها 4 عناصر:

"Ahmed"

"Sayed"

"Ali"

مصفوفة متداخلة: `["Shady", "Amr", ["Mohamed", "Gamal"]]

بدنا نوصل للعنصرين "Shady" و "Gamal" باستخدام **destructuring** بدون الحاجة لاستخدام الفهرس مباشرة. شو بصير؟

لما نحط `[a, [b]]` : رح ياخذ أول عنصر بالمصفوفة المتداخلة اللي هو "Shady". رح ياخذ العنصر الثاني من المصفوفة اللي داخل المصفوفة المتداخلة، والتي هو "Gamal".

النتائج:

- `a` = "Shady"

- `b` = "Gamal"

بهذا الشكل بنكون قدرنا نوصل لعناصر متداخلة في المصفوفة بطريقة مرتبة وبدون استخدام الفهارس العادية.

### 103-Destructuring → Swapping Variables

هذا الكود يستخدم تقنية destructuring assignment لتبديل قيم متغيرين بدون الحاجة إلى استخدام متغير مؤقت.

```
let book = "video";  
let video = "book";  
[book, video] = [video, book];  
console.log(book);  
console.log(video);
```

عندك متغيرين book قيمته video و video قيمته book

باستخدام destructuring بنحط قيم المتغيرين داخل مصفوفة ونقوم بعكسهم book رح يأخذ قيمة video التي هي book و video رح يأخذ قيمة book الأصلية التي هي video بعد عملية التبديل القيم صارت book تساوي video و video تساوي video

---

### 104-Destructuring → Object

في هذا الكود بنستخدم destructuring عشان نستخرج قيم معينة من كائن (object) بشكل مباشر.

```
const user = {  
  theName: "Suhaib",  
  theAge: 39,  
  theTitle: "Developer",  
  theCountry: "egqpt", };  
const { theName, theAge, theCountry, theTitle } = user;  
console.log(theTitle);
```

عندك كائن اسمه `user` فيه مجموعة من الخصائص (properties) مثل الاسم، العمر، الوظيفة، والدولة. بدلاً من الوصول لكل خاصية باستخدام اسم الكائن، بنقدر نستخدم destructuring ونفكك القيم مباشرة.

فالكود بعمل التالي:

- بإنشاء متغيرات جديدة بنفس أسماء الخصائص الموجودة في الكائن `user` مثل `theName` و `theAge` و `theCountry` و `theTitle`.

- هذه المتغيرات بتأخذ القيم التي موجودة داخل الكائن `user` بشكل مباشر.

بعدين بيتم طباعة قيمة `theTitle` التي هي "Developer".



### 105-Destructuring ➔ (Naming The Variables , AddNew , Nested )

في هذا الكود، نستخدم destructuring للكائنات (objects) لاستخراج القيم من الكائن user بطريقة مختصرة. وأيضاً، نستخدم قيم افتراضية إذا ما كانت بعض الخصائص موجودة.

```
const user = {
  theName: "Suhaib", theColor: "black",
  Skills: { html: 70, css: 80, }, };
const {
  theName: name, theColor: co = "red", Skills: { html: ht, css: cs }, } = user;
console.log(name);
console.log(co);
console.log(ht);
console.log(cs);
const { html: skillOne, css: skillTwo = "5421" } = user.Skills;
console.log(skillOne);
console.log(skillTwo);
```

بهذا الكود نستخدم تقنية تفكيك الكائنات عشان نطلع القيم من الكائن بشكل مختصر بنطلع الاسم ونحطه في متغير اسمه "name" وينطلع اللون ونحطه في متغير اسمه "co" وإذا ما كان اللون موجود بالكائن بنحط لون افتراضي اللي هو أحمر بدين بنطلع قيم "HTML" و "CSS" من الكائن اللي اسمه "Skills" ونحطهم بمتغيرين بعد هيك بنعمل تفكيك مباشر من كائن "Skills" بنطلع قيمة "HTML" ونحطها بمتغير وينطلع قيمة "CSS" وإذا مش موجودة بنعطيه قيمة افتراضية

---

### 106-Destructuring ➔ Destructuring Function Parameters

```
const user = {
  theName: "Suhaib", theAge: 39,
  Skills: { html: 70, css: 80, }, };
showDetails(user);
function showDetails({ theName: n, theAge: c, Skills: { css } } = user) {
  console.log(`Your Name Is ${n}`);
  console.log(`Your Age Is ${c}`);
  console.log(`Your Css Skill Progress Is ${css}`); }
```

الكود اللي كتبته بستخدم فكرة التقطيعاو destructuring عشان ياخذ معلومات معينة من الكائن `user` لما نستدعي الدالة `showDetails` بنمرر كائن `user` اللي فيه الاسم والعمر والمهارات الدالة بتاخذ القيم بشكل مباشر من الكائن وتعيد تسميتهم جوا

الدالة الاسم بنسبته n والعمر c والمهارة التي بدنا ياها بنختارها التي هي css بعدين الدالة بتطبع اسمك وعمرك ومستوى مهارتك في css

#### تطبيق كامل -107

```
const user = {
  theName: "Suhaib", theAge: 23, skills: ["html", "css", "javascript"], addresses: {egypt: "Cairo",ksa:
  "Riyadh",}, };
const { theName: n, theAge: a, skills: [, , three],addresses: { egypt: e },} = user;
console.log(`Your name is ${a}`);
console.log(`Your Age is ${a}`);
console.log(`Your Skills is ${three}`);
console.log(`Your Live is ${e}`);
```

---

Exame:-

#### -----Video 122-----

طيب كيف ممكن نكتب كود بحيث لو عندي مصفوفة فيها معلومات عن اصدقائي وكل واحد فيهم عنده اسم وعمر اذا متاح او لا ومهارات وانا بدي اعرض معلومات صديقي بناءً على رقم انا باختياره يعني مثلاً اذا اخترت اول صديق او ثالث واحد كيف بدي اجيب الاسم والعمر واذا متاح او لا واعرض المهارة الثانية الي عنده شو الطريقة الافضل لتقليل التكرار في الكود واختصار السطور

باستخدام المعلومات التالية

```
let chosen = 3;
let myFriends = [
  { title: "Suhiab", age: 23, available: true, skills: ["Html", "Css"] },
  { title: "Omar", age: 25, available: false, skills: ["Python", "Django"] },
  { title: "Ahmed", age: 33, available: true, skills: ["Php", "Laravel"] },
];
```

حلي انا

```
if (chosen == 1) {
  var { title: n, age: a, available: ava, skills:sk } = myFriends[0];
  console.log(n);
  console.log(a);
  if (ava == true) {console.log(`available`);} else {console.log(`Not available`);}
```

```

    console.log(sk[1]);
} else if (chosen == 2) {
    var { title: n, age: a, available: ava, skills: sk } = myFriends[1];
    console.log(n);
    console.log(a);
    if (ava == true) {console.log(`available`);} else {console.log(`Not available`);}
    console.log(sk[1]);
} else if (chosen == 3) {
    var { title: n, age: a, available: ava, skills: sk } = myFriends[2];
    console.log(n);
    console.log(a);
    if (ava == true) {console.log(`available`);} else {console.log(`Not available`);}
    console.log(sk[1]);
} else { console.log(`Please Index number Corect`); }

```

---

حل ثاني

```

let {title:empName,age,available,skills:[,lastSkill]} = myFriends[chosen -1]
console.log(empName)
console.log(age)
console.log(available? "available" : "Not available")
console.log(lastSkill)

```

---

حل ثالث

```

if (myFriends[chosen - 1]) {
    let { title, age, available, skills } = myFriends[chosen - 1];
    console.log(title);
    console.log(age);
    console.log(available ? "Available" : "Not Available");
}

```

```
console.log(skills[1]);  
} else {console.log("Please enter a valid index number."); }
```

## 108-Set Data ➔ Type Methods

```
let myData = [1, 1, 1, 2, 3, "A"];
```

اربع طرق على استخدام Set

```
let myUnqueData = new Set([1, 1, 1, 2, 3]); // الاول
```

```
let myUnqueData = new Set(myData); // الثانية
```

```
let myUnqueData = new Set().add(1).add(1).add(1).add(2).add(3); // الثالث
```

```
let myUnqueData = new Set(); // الرابعة
```

```
myUnqueData.add(1).add(1).add(1);
```

```
myUnqueData.add(2).add(3).add("A");
```

```
console.log(myUnqueData.has("A"));
```

```
myUnqueData.delete(2);
```

```
myUnqueData.clear();
```

```
console.log(myData[0]);
```

```
console.log(myData);
```

```
console.log(myUnqueData);
```

```
console.log(myUnqueData.size);
```

عندك هون مثال بسيط عن كيف تستعمل Set في جافاسكربت الفكرة من الـ Set إنها بتحفظ البيانات وما بتسمح بتكرار القيم يعني إذا ضفت نفس القيمة أكثر من مرة ما رح تتضاف مرة ثانية

بالبداية أنشأنا Set جديد وضمنا قيم للمجموعة بعدين استخدمنا دالة has عشان نتأكد إذا القيمة موجودة أو لا بعدين استخدمنا delete عشان نحذف قيمة معينة وأخيرًا clear لحذف كل القيم بالمجموعة كاملة وأخيرًا حجم المجموعة بعد ما نستخدم clear رح يكون صفر

## 109-Set VS WeakSet

الفرق بين الـ `Set` و الـ `WeakSet` في جافاسكربت يتعلق بشكل رئيسي بكيفية إدارة البيانات واستخدامها.

الـ `Set`:

- يحتفظ بأي نوع من البيانات سواء **primitive** زي الأرقام أو النصوص أو **object** زي الكائنات. القيم اللي فيه فريدة، ما بكرر أي قيمة مضافة. القيم بتظل محفوظة في الذاكرة بشكل دائم إلا إذا مسحتها بنفسك. فيك تستخدم ميزات زي `size` عشان تعرف كم عنصر فيه أو `forEach` عشان تمر على كل العناصر.

الـ `WeakSet`:

- بيقبل فقط **object** كقيم، يعني ما فيك تضيف أرقام أو نصوص. القيم اللي فيه ما بتظل محفوظة بالذاكرة بشكل دائم. إذا صار ما في مراجع للكائنات الموجودة بالـ **WeakSet**، بيتم إزالتها تلقائيًا عن طريق الـ **Garbage Collection**، وهذا بسبب إن **WeakSet** بضعف الإشارة على الكائنات اللي داخله. ما فيك تستخدم ميزات زي `size` أو تمر على كل العناصر باستخدام `forEach`، لأنه مصمم للتعامل مع الكائنات بشكل خفيف ولتجنب استهلاك الذاكرة بشكل كبير. الـ `Set` مناسب لما تحتاج تخزن أي نوع من البيانات وتضمن إنها تظل محفوظة.

- الـ `WeakSet` مناسب لما بدك تتعامل مع كائنات بدون ما تقلق من موضوع استهلاك الذاكرة لأنه بيتم تنظيفه تلقائيًا من القيم غير المستخدمة.

//Type Of Data

```
let mySet = new Set([1, 1, 3, "A", "A"]);
```

```
console.log(mySet);
```

//Size

```
console.log(`Size Of Elements Inside Ts: ${mySet.size}`);
```

//Values + Keys [Alias For Values]

```
let iterator = mySet.keys();
```

```
console.log(iterator.next().value);
```

```
console.log(iterator.next().value);
```

```
console.log(iterator.next().value);
```

```
console.log(iterator.next());
```

```
console.log(`#####`.repeat(7));
```

//forEach

```
mySet.forEach((el) => console.log(el));
```

//Type Of Data

```
let myws = new WeakSet([{ a: 1, B: 2 }]);
```

```
console.log(myws);
```

## 110-Map Data Type Vs Object

الفرق بين `Map` و `Object` في جافا سكربت يتعلق بكيفية تخزين البيانات والتعامل معها.

### :Object

المفاتيح : المفاتيح في `Object` لازم تكون إما نصوص (strings) أو رموز (symbols)، يعني ما بتقدر تستخدم أنواع بيانات أخرى زي الأرقام أو القيم المنطقية كمفاتيح. الأداء: في التعامل مع عدد كبير من المفاتيح أو البيانات، ممكن يكون الأداء أبطأ مقارنة مع الـ `Map`. الترتيب : ما بيحافظ على ترتيب الإدخال. القيم بتكون مرتبة حسب كيفية إضافتها، لكن مش مضمونة. الوظائف : كائنات بتجي بميزات مدمجة، مثل `hasOwnProperty` وفوقها بتقدر تضيف دوال للكائن نفسه.

### :Map

- \*\*المفاتيح\*\*: بسمح لك تستخدم أي نوع من أنواع البيانات كمفاتيح، سواء كانت نصوص، أرقام، كائنات أو حتى دوال. الأداء: أداء أفضل بشكل عام في العمليات التي بتحتاج التحقق أو التعامل مع كمية كبيرة من البيانات، زي `has`, `get`, `set` مقارنة مع `Object`. الترتيب: بيحافظ على ترتيب الإدخال، يعني العناصر اللي بتضيفها بتضل مرتبة حسب إضافتك إياها. الوظائف: ما بضيف أي وظائف افتراضية مثل `Object`. بتتعامل معه عن طريق دوال محددة زي `has`, `get`, `set`, و `delete`.

**Object** : مناسب لما بتحتاج تخزين بيانات نصية كمفاتيح وبذلك تستفيد من الوظائف المدمجة.

**Map** : مناسب لو بتحتاج مفاتيح من أي نوع، وأداء أفضل مع عدد كبير من البيانات، وبتحتاج للحفاظ على ترتيب الإدخال.

```
let myObject = {};  
let myEmptyObject = Object.create(null);  
let myMap = new Map();  
console.log(myObject);  
console.log(myEmptyObject);  
console.log(myMap);  
let myNewObject = {  
  10: "Number",  
  "10": "Number",  
};  
console.log(myNewObject[10])  
let myNewMap = new Map();  
myNewMap.set(10,"Number")  
myNewMap.set("10","String")  
console.log(myNewMap.get(10))  
console.log(myNewMap.get("10"))
```

```
console.log("###")
```

### 111-Map Methods

```
let myMap = new Map([
  [10, "Number"],
  ["Name", "String"],
  [false, "Boolean"],
]);
console.log(myMap);
console.log(myMap.get(10));
console.log(myMap.get("Name"));
console.log(myMap.get(false));
console.log(myMap.size);
console.log(myMap.delete("Name"));
console.log(myMap.size);
console.log(myMap.has(false));
myMap.clear();
console.log(myMap);
```

---

### 112-Map Vs WeakMap

```
let mapUser = { theName: "Elzero" };
let myMap = new Map();
myMap.set(mapUser, "Onject Value");
mapUser = null;
console.log(myMap);
```

في الكود الأول يتم استخدام Map عشوائياً تخزن بيانات معينة. يتم إنشاء ماب جديدة وتخزين كائن فيها باستخدام set. بعد ذلك لما يغير قيمة mapUser يتنقل البيانات التي في الـ Map موجودة لأنها تعتمد على المرجع الأساسي للكائن مش على التغيير الذي صار عليه.

```
console.log("#".repeat(20));
```

```
let myWeakMap = new WeakMap();  
myWeakMap.set(myWeakMap, "Object Value");  
myWeakMap = null;  
console.log(myWeakMap);
```

أما بالنسبة للـ **WeakMap** فهي نوع خاص من الـ **Map** بس الفكرة هون إنه لما تغير قيمة المفتاح اللي جواتها بيتم تحرير الذاكرة المستخدمة لهذا المفتاح، ويتصير البيانات غير قابلة للوصول لأنه الـ **WeakMap** ما بتحتفظ بالمفتاح إذا صار عليه تعديل

---

### 113-Array Methods ➔ Array.from

تحويل سلسلة نصية "Suhiab" إلى مصفوفة

```
console.log(Array.from("Suhiab"));
```

تحويل سلسلة "12345" إلى مصفوفة وجمع كل رقم مع نفسه باستخدام دالة عادية

```
console.log(  
  Array.from("12345", function (n) {  
    return +n + +n; }) );
```

نفس العملية باستخدام Arrow Function

```
console.log(Array.from("12345", (n) => +n + +n));
```

تحويل مصفوفة فيها أرقام مكررة إلى Set وإرجاعها كمصفوفة

```
let myArray = [1, 1, 1, 2, 3, 4];  
let mySet = new Set(myArray);  
console.log(Array.from(mySet));
```

استخدام طريقة مختصرة مع الانتشار (...) لتحويل Set إلى مصفوفة

```
console.log([...new Set(myArray)]);
```

دالة تستخدم Array.from لتحويل arguments إلى مصفوفة

```
function af() {  
  return Array.from(arguments); }  
console.log(af("SUhiab", "omar", "Ahmed"));
```

----- الشرح -----



1. تحويل النص إلى مصفوفة  
`Array.from("SuhiaB")` يتحول كل حرف من الكلمة "SuhiaB" إلى عنصر منفصل داخل المصفوفة، يعني كل حرف بصير عنصر مستقل.
2. تحويل الأرقام إلى مصفوفة وجمع كل رقم مع نفسه  
 في الجزء الثاني، سلسلة الأرقام "12345" تتحول لمصفوفة وكل رقم يضاعف باستخدام دالة عادية `function (n)`.
3. استخدام Arrow Function  
 هاي الطريقة بتماثل الجزء السابق، لكنها بتستخدم `=>` عشان تكون الكتابة مختصرة.
4. تحويل المصفوفة إلى Set لإزالة التكرارات  
`Set` هو نوع خاص من البيانات في جافا سكريبت بيحذف أي عناصر مكررة. هون المصفوفة الأصلية بتحتوي على أرقام مكررة، لكن `Set` بيحذف التكرارات ويعدين بنحول النتيجة إلى مصفوفة باستخدام `Array.from`.
5. استخدام الانتشار مع Set  
 هاي طريقة ثانية لتحويل `Set` إلى مصفوفة باستخدام `...`، بتعمل نفس الشيء مثل `Array.from` بس بأسلوب أبسط وأوضح.
6. تحويل Arguments إلى مصفوفة باستخدام `Array.from`  
 الدالة `af` بتستقبل أي عدد من المتغيرات (`arguments`)، وتحوّلهم إلى مصفوفة باستخدام `Array.from`؛ هاي الطريقة مفيدة لما ما بتعرف عدد القيم اللي بتمررها للدالة.

#### 114-Array Methods ➔ Array.Copy

دالة بتسمحك تنسخ جزء من المصفوفة إلى مكان ثاني داخلها بدون تغيير طول المصفوفة.

```
let myArray = [10, 20, 30, 40, 50, "A", "B"];
```

```
myArray.copyWithin(3);
```

هون بتحكي للدالة إنها تنسخ العناصر من بداية المصفوفة (يعني من أول عنصر) وتلصقهم من الموضع الثالث في المصفوفة. فالعناصر الأولية: 10، 20، 30 بتتم نسخها من بداية المصفوفة وتلصق من المكان الثالث.

```
myArray.copyWithin(4,6)
```

هون بتبيلش النسخ من الموضع السادس) يعني من العنصر "B" وتلصق النسخة في الموضع الرابع، فبتصير المصفوفة فيها "B" بمكان الـ 40.

```
myArray.copyWithin(4, -1);
```

هون بتبيلش النسخ من الموضع الأخير) اللي هو -1 يعني آخر عنصر "B" وتلصقها في الموضع الرابع، فبتصير "B" بمكان العنصر 50.

```
myArray.copyWithin(1,-2);
```

النسخ هون ببيلش من الموضع اللي قبل الأخير (-2 يعني العنصر 50) وتلصق من الموضع الأول (يعني مكان 20)، فبتصير 50 بمكان 20.

```
myArray.copyWithin(1,-2,-1);
```

هون الدالة بتنسخ العنصر قبل الأخير (-2 يعني 50) وتلصق في الموضع الأول، لكن النسخ بيتوقف قبل آخر عنصر (-1 يعني مش راح ينسخ آخر عنصر).

```
console.log(myArray);
```

بالنهاية، النتيجة النهائية للمصفوفة بعد كل التعديلات بتبين كيف تم إعادة ترتيب القيم في المصفوفة باستخدام دالة `copyWithin`.

## 115-Array ➔ Array.Some

دالة `some` في جافاسكربت هي دالة تستخدم للتحقق إذا كان في عنصر واحد على الأقل في المصفوفة يحقق شرط معين. بمعنى آخر، الدالة بترجع `true` إذا كان في عنصر واحد أو أكثر يطابق الشرط الذي بمرره، وإذا ما كان في أي عنصر يطابق الشرط بتعطيك `false`.

```
let nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
```

```
let myNumber = 5;
```

```
let check = nums.some(function (e) {
```

```
  return e > this;
```

```
}, myNumber);
```

```
console.log(check);
```

بتحدد مصفوفة `nums` من الأرقام وبتخلي المتغير `myNumber` يساوي ٥. بعدين بتستخدم `some` للتحقق إذا كان في أي عنصر في المصفوفة أكبر من `myNumber`. الفكرة إنه بتستخدم قيمة `myNumber` كـ `this` داخل دالة `some` عن طريق تمريرها كـ `thisArg` بعد الدالة. لما تطبع النتيجة راح تكون `true` لأن في عناصر أكبر من ٥

```
function checkValues(arr, val) {
```

```
  return arr.some(function (e) {
```

```
    return e === val;
```

```
  }); }
```

```
console.log(checkValues(nums, 20));
```

```
console.log(checkValues(nums, 5));
```

دالة `checkValues` بتفحص إذا كان العنصر المطلوب موجود في المصفوفة أو لا. الدالة بتأخذ المصفوفة والعنصر اللي بدك تفحصه، وبترجع `true` إذا كان موجود و `false` إذا مش موجود.

```
let range = { min: 10, max: 20};
```

```
let CheckNumberInRange = nums.some(function (e) {
```

```
  return e >= this.min && e <= this.max; }, range);
```

```
console.log(CheckNumberInRange);
```

هون بتستخدم `some` للتحقق إذا كان في أرقام في المصفوفة `num` موجودة ضمن نطاق محدد `range` اللي قيمه من ١٠ إلى ٢٠. في هاي الحالة بمرر كائن `range` كـ `thisArg` لحتى يتم استخدامه داخل دالة `callback`.

## 116-Array → Array.every

```
const locations = {
  20: "place 1",
  30: "place 2",
  10: "place 3",
  40: "place 4",
};
let mainLocation = 15;
let locationsArray = Object.keys(locations);
console.log(locationsArray);
```

هون عم نستخدم map اللي هي دالة بتلف على كل عنصر في المصفوفة locationsArray  
الـ n يتمثل كل عنصر من عناصر المصفوفة والتي هي عبارة عن نصوص  
+n يستخدم علامة + عشان نحول النصوص لأرقام  
النتيجة بتكون مصفوفة جديدة اسمها locationsArrayNumbers فيها الأرقام بدل النصوص

```
let locationsArrayNumbers = locationsArray.map((n) => +n);
console.log(locationsArrayNumbers);
let check = locationsArrayNumbers.every(function (e) {
  return e > this;
}, mainLocation);
console.log(check);
```

هون عم نستخدم every للتحقق إذا كل العناصر في المصفوفة locationsArrayNumbers بتراضي شرط معين  
function (e) هي دالة بتفحص كل عنصر في المصفوفة  
e هو العنصر الحالي اللي عم نفحصه  
return e > this; يتحقق إذا العنصر e أكبر من القيمة اللي هي this  
this في الحالة بتكون قيمة المتغير mainLocation اللي هو 15  
every رح ترجع true إذا كل العناصر أكبر من 15، و false إذا في عنصر واحد على الأقل مش أكبر من 15

## 117-Array → Spread Operator → Iterable

الـ **Spread Operator** في جافاسكربت هو طريقة بتخليك تتعامل مع **iterable** زي المصفوفات (**Arrays**) أو النصوص أو حتى الكائنات (**Objects**) بطريقة مريحة وسهلة. فكر فيه كأنه طريقة لتوسيع (أو نشر) محتوى المصفوفة أو الكائن على عناصر فردية. مثال على **Array** مع الـ **Spread Operator**: لو عندك مصفوفة وبك تنقل عناصرها لمصفوفة جديدة أو تستخدمها كمعاملات دالة، بدل ما تكتب كل عنصر لحال، بتستخدم الـ **Spread Operator** (...) وبتنشر كل عناصر المصفوفة مرة وحدة. **Iterable** هو أي كائن في جافاسكربت بتقدر تستخدم معاه حلقة مثل **for...of** زي المصفوفات والنصوص و **Maps** و **Sets**. والـ **Spread Operator** بيشغل مع هاي الكائنات لأنه بفككهم لعناصرهم الفردية. باختصار، الـ **Spread Operator** بيسهل عليك التعامل مع البيانات القابلة للتكرار (**iterable**) زي المصفوفات بطريقة مرنة وسريعة بدل ما تحتاج تتعامل مع كل عنصر لحاله.

```
console.log("Suhaib");
console.log(..."Suhaib");
console.log([... "Suhaib"]);
// Concatenate Arrays
let myArray1 = [1, 2, 3];
let myArray2 = [4, 5, 6];
let allArrays = [...myArray1, ...myArray2];
console.log(allArrays);
//Copy Array
let copiedArray = [...myArray1];
console.log(copiedArray);
// Push Inside Array
let allFrineds = ["Omar", "Suhaib", "Ahmad"];
let thisYearFriends = ["Sameh", "Mahmuod"];
allFrineds.push(...thisYearFriends);
console.log(allFrineds);
// Use With Math Object
let myNums = [10, 20, -100, 100, 1000, 500];
console.log(Math.max(...myNums));
//Spread With Objects => Merge Objects
let objOne = {a: 1, b: 2,};
let objTow = {c: 3,d: 4, };
```

```
console.log({ ...objOne, ...objTwo, e: 5 });
```

### Exame :- Map And Set Challenge → Video (133)

بدك أطلع حل رقم 210 وممنوع تستخدم اي رقم او دلة

```
let n1 = [10, 30, 10, 20];  
let n2 = [30, 20, 10];  
let nums=Math.max(...n2)*[...n1,...n2].length;  
console.log(`Your Solution Here:${nums}`);
```

أول اشي عندك مصفوفتين :

```
n1 = [10, 30, 10, 20]
```

```
n2 = [30, 20, 10]
```

السطر اللي فيه `Math.max(...n2)` هون بنستخدم الـ `Math.max` عشان نجيب أكبر قيمة بالمصفوفة `n2` الـ `...n2` اللي هو `Spread Operator` بفك المصفوفة لعناصر فردية وبعطيها للدالة `Math.max` عشان تختار أكبر رقم. فبالنسبة لهاي المصفوفة، أكبر رقم هو `30`

`[...n1, ...n2].length` هون بنستخدم الـ `Spread Operator` كمان مرة عشان ندمج المصفوفتين `n1` و `n2` في مصفوفة وحدة. يعني لما تدمجهم بيصيروا: `[10, 20, 30, 20, 10, 30, 10]`. وبعدين بنستخدم `.length` عشان نحسب عدد العناصر اللي صارت بالمصفوفة المدمجة، والتي هو `7`.

السطر الأخير اللي بنعمله هون هو ضرب أكبر رقم من `n2` (اللي هو `30`) في عدد العناصر بالمصفوفة المدمجة (اللي هو `7`). الناتج بيكون `210`.

ببساطة الكود بحسب أكبر رقم بالمصفوفة الثانية، وبعدين بدمج المصفوفتين وبحسب عدد العناصر، وأخيراً بضرب العددين مع بعض.

## 118- Regular Expression

الـ **Regular Expressions** هي أدوات قوية تستخدمها عشان تبحث أو تتحقق أو تعدل على النصوص بشكل سهل وسريع. الفكرة منها إنك تكتب أنماط أو "قوالب" بتقدر تدور عليها داخل النصوص. كيف تشغل؟

بجافاسكربت، بنستخدم الـ **Regex** مع بعض الدوال الجاهزة مثل:

1. **test()**: بتفحص إذا النمط موجود بالنص.
2. **exec()**: بترجع أول تطابق للنمط مع النص.
3. **match()**: بترجع كل التطابقات داخل النص.

مثال بسيط: لو عندك نص وبك تدور على أي كلمة فيها أرقام (زي رقم هاتف أو عمر):

```
let myString; "أنا عمري 30 سنة";
```

```
let myRegex = /\d;/+
```

```
console.log(myRegex.test(myString)); // بتفحص إذا فيه رقم
```

هون الـ **d** معناها أي مجموعة أرقام (واحد أو أكثر)، وجافاسكربت بتفحص إذا النص فيه هاي الأرقام.

### شرح بعض الرموز المستخدمة:

- **d**: رقم واحد (0-9).

- **w**: حرف أو رقم (A-Z, a-z, 0-9).

- **+**: يعني "واحد أو أكثر" من اللي قبلها، مثل **d+** يعني أي مجموعة أرقام.

- **^**: بداية السطر.

- **\$**: نهاية السطر.

- **.**: أي حرف أو رمز.

مثال عملي على الإيميلات: لو بك تفحص إذا المستخدم دخل إيميل صحيح:

```
let email = "suhaib@example.com";
```

```
let emailPattern = /^[w-]+@[([w-]+\.)+([w-]);/2,4}
```

```
console.log(emailPattern.test(email)); // بفحص إذا الإيميل صحيح
```

هذا النمط بفحص إذا النص يبدأ بحروف أو أرقام، بعدين فيه @ وبعده الدومين، وهي بتقدر تتأكد من صحة الإيميل.

باختصار، Regular Expressions في جافاسكريبت سيساعدوك تتعامل مع النصوص بطرق مرنة وقوية، سواء كان البحث أو التحقق من صحة المدخلات أو حتى التعديل على النص.

الأمثلة الي فوق مش بالضروري تفهمها بالأمثلة الي تحت راح نشرحها شرح مفصل الهدف من الي فوق الاطلاع فقط

## 119-Regular Expression → Syntax

I → Case-insensitive

g → Global

m → Multilines

```
let myStiong = "Hello Elzero Web School I Love elzero";
```

```
let regex = /elzero/ig;
```

```
console.log(myStiong.match(regex))
```

```
console.log(myStiong.match(/Elzero/))
```

الكود اللي كتبته يستخدم Regular Expression للبحث عن كلمة elzero في النص myStiong

regex = /elzero/ig :

/elzero/: هذا النمط يبحث عن الكلمة elzero

I : يسمح للبحث يكون غير حساس لحالة الأحرف (case insensitive)، يعني بيدور على "elzero" سواء كانت بأحرف كبيرة أو صغيرة.

g : يعني البحث سيكون شامل لكل النص، وبيجيب كل التطابقات مش أول تطابق بس.

لما تستعمل match() مع regex , بيرجعك مصفوفة فيها كل الأماكن اللي لقي فيها الكلمة "elzero".

---

## 120-Regular Expression → Ranges → Part One

```
let tld = "com Net Org Info Code Io";
```

```
let tldRe = /(org|info|io)/gi;
```

```
console.log(tld.match(tldRe));
```

هون عم تستخدم Regular Expression للبحث عن الكلمات org و info و io داخل النص. الرمز g معناها عم يبحث بكل النص، والرمز i معناها عم يتجاهل الفرق بين الأحرف الكبيرة والصغيرة. يعني بيطلعك كل التطابقات سواء كانت مكتوبة بحروف كبيرة أو صغيرة.

```
let nums = "12345678910";
```

```
let numsRe = /[0-2]/g;
```

```
console.log(nums.match(numsRe));
```

هون عم تدور على الأرقام اللي بين 0 و 2 فقط. داخل النص nums. بمجرد ما يلاقي رقم بين هالحدود بيحطه بالمصفوفة ويبطلعك إياها.

```
let notNums = "12345678910";  
let notNsRe = /^[^0-2]/g;  
console.log(notNums.match(notNsRe));
```

هون عم تدور على كل الأحرف اللي مش من الأرقام 0 إلى 2. الرمز ^ معناه "ليس". فهون انت بتقوله "دورلي على كل شي مش بين الأرقام 0 إلى 2".

```
let specialNums = "1!2@3#4%12345678910";  
let specialNumsRe = /^[^0-9]/g;  
console.log(specialNums.match(specialNumsRe));
```

هون عم تستخدم [^0-9] يعني "دور على كل شي مش رقم". فبيطلعك الرموز الخاصة زي % # @ ! الموجودة داخل النص.

```
let practice = "Os1 Os10s Os2 Os8 Os8os";  
let practiceRe = /os[5-9]os/gi;  
console.log(practice.match(practiceRe));
```

هون عم تدور على كلمة os اللي بعدها رقم بين 5 و 9 وبعدين os مرة ثانية. مثلا، النص Os8os بيتطابق مع هالنمط، ولكن Os10s ما بيتطابق لأنه الرقم 10 مش بين 5 و 9.

#### الخلاصة

(org|info|lo): بيبحث عن واحد من الكلمات المحددة

[0-2]: بيدور على الأرقام من 0 إلى 2

[^0-2]: بيدور على أي شي غير الأرقام 0 إلى 2

[^0-9]: بيدور على أي شي غير الأرقام من 0 إلى 9

os[5-9]os: بيدور على كلمة os اللي بينها رقم من 5 إلى 9



## 121-Regular Expression → Ranges → Part Two

```
let myString = "AaBbcdefG123!234%^&*";
```

```
let atozSmall = /[a-z]/g;
```

```
console.log(myString.match(atozSmall));
```

هون عم تبحث عن كل الأحرف الصغيرة من a إلى z في النص، وبتطلعها.

```
let notAtozSmall = /^[^a-z]/g;
```

```
console.log(myString.match(notAtozSmall));
```

الرمز ٨ داخل الأقواس بيعمل استثناء. يعني هون عم تبحث عن أي شيء بالنص مش حرف صغير.

```
let atozCapital = /[A-Z]/g;
```

```
console.log(myString.match(atozCapital));
```

هون عم تبحث عن كل الأحرف الكبيرة من A إلى Z في النص.

```
let notAtozCapital = /^[^A-Z]/g;
```

```
console.log(myString.match(notAtozCapital));
```

مثل ما حكينا قبل، ٨ بيعمل استثناء. فهون عم تدور على أي شيء بالنص مش حرف كبير.

```
let aAndcAnde = /[ace]/g;
```

```
console.log(myString.match(aAndcAnde));
```

هون عم تدور على الأحرف a و c و e فقط داخل النص.

```
let NotaAndcAnde = /^[^ace]/g;
```

```
console.log(myString.match(NotaAndcAnde));
```

نفس المبدأ، هون عم تستثني الأحرف a و c و e وتدور على كل شيء غيرهم.

```
let ABC = /[a-zA-Z]/gi;
```

```
console.log(myString.match(ABC));
```

هون عم تبحث عن أي حرف سواء كان كبير أو صغير من a إلى z أو من A إلى Z. الرمز i بيخلي البحث غير حساس لحالة الأحرف.

```
let notABC = /^[^a-zA-Z]/gi;
```

```
console.log(myString.match(notABC));
```

هون عم تدور على كل شيء داخل النص مش حرف كبير أو صغير، زي الرموز أو الأرقام.

## 122- Regular Expression → Character Classes →Part One

```
let email = "O@@@g...com O@g.net A@Y.com O-G.com o@g.com o@s.org 1@1.com";
```

```
let dot = /. /g;
```

```
console.log(email.match(dot));
```

هون عم تستخدم النقطة .، اللي بتمثل أي رمز (يعني أي حرف أو رقم أو رمز آخر)، وبترجع كل الرموز اللي موجودة في النص بدون استثناء.

```
let word = /\w/g;
```

```
console.log(email.match(word));
```

هون عم تستخدم \w اللي بتمثل أي حرف من الحروف الإنجليزية، الأرقام، أو الشرطة السفلية (underscore). فهون بيبعث عن كل الرموز اللي بتطابق هذا النمط داخل النص.

```
let valid = /\w@\w.(com|net)/g;
```

```
console.log(email.match(valid));
```

هون عم تبحث عن الإيميلات اللي تتطابق مع نمط معين، وهو إنه يكون الإيميل فيه كلمة (حروف أو أرقام) قبل وبعد الرمز @، وينتهي بواحد من النطاقين .com أو .net.

---

## 123- Regular Expression → Character Classes →Part Tow

```
let names = "Sayed 1Spam 2Spam 3Spam Spam4 Spam5 Suhaib Ahmed Aspamo";
```

```
let re = /(\\bspam|spam\\b)/gi;
```

```
console.log(names.match(re));
```

هون عم تستخدم النمط اللي ببحث عن spam لما تكون إما بداية أو نهاية الكلمة /b هو اللي بحدد حدود الكلمة، يعني إذا كانت spam في بداية الكلمة أو نهايتها، النمط رح يتعرف عليها

```
console.log(re.test(names));
```

```
console.log(/(\\bspam|spam\\b)/gi.test("Suhaib"));
```

```
console.log(/(\\bspam|spam\\b)/gi.test("1Spam"));
```

```
console.log(/(\\bspam|spam\\b)/gi.test("Spam1"));
```

بتستخدم test عشان تشوف إذا النص يحتوي على تطابق للنمط. إذا كان موجود، بيرجع true، وإذا ما في، بيرجع false. تجارب على عدة نصوص

"Suhaib" رح يرجع false لأنه ما في كلمة "Suhaib" بالنص الي بتبحث عنو لانو المجال محصور في كلمة spam

"1Spam" رح يرجع true لأنه فيه كلمة "spam" بنهاية الكلمة.

"Spam1" رح يرجع true لأنه فيه كلمة "spam" ببداية الكلمة.

بالمجمل، النمط (\bspam|spam\b) مفيد إذا بدك تدور على كلمة "spam" وهي منفصلة أو جزء من كلمة لكن في بداية أو نهاية الكلمة.

#### 124- Regular Expression → Quantifiers → Part One

N+ → One Or More

N\* → Zero Or More

N? → Zero Or One

```
let mails = "o@nn.sa suhaib@gmail.com elzero@gmail.net Suhaib@mail.ru";
```

```
let mailsRe = /\w+@\w+.\w+/gi;
```

```
console.log(mails.match(mailsRe));
```

\w+: يعني أي عدد من الحروف أو الأرقام (الأحرف + الأرقام). @: عشان تحدد وجود إشارة @ في النص. \w+: بعد إشارة @ بدك دومين (مجموعة حروف/أرقام)

. : بعدها بدك النقطة اللي بتفصل اسم الدومين عن الامتداد \w+: وأخيراً بدك الامتداد زي .com أو .net.

```
let nums = "0110 10 150 05120 0560 350 00";
```

```
let numsRe = /0\d*0/gi;
```

```
console.log(nums.match(numsRe));
```

هون عم بدور على أي رقم ببداً وينتهي بصفر: 0 الرقم لازم يبدأ بصفر. \d\*: يعني أي عدد من الأرقام بعد الصفر الأول. 0: وينتهي بصفر. رح يمस्क أرقام زي 0110 و 05120 لأنها بتحقق الشروط.

```
let urls = "https://google.com http://www.website.ner web.com";
```

```
let urlsRe = /(https?:\/\/)?(www.)?\w+.\w+/gi;
```

```
console.log(urls.match(urlsRe));
```

هاد النمط مخصص لالتقاط الروابط، سواء كان فيها http أو https أو www:

- (https?://)? : يعني الرابط ممكن يبدأ ب http أو https أو ممكن ما يبدأ فيهم (اختياري).
- (www.)? : وجود www كمان اختياري.
- \w+: مجموعة من الحروف أو الأرقام.
- .: بعيدين النقطة.
- \w+: وأخيراً الدومين زي com أو net.

## 125- Regular Expression → Quantifiers → Part Two

N { X } → Number Of

N { X , Y } → Range

N { X, } → At Least X

```
let serials = "S100S S3000S S50000S S950000S";
```

```
console.log(serials.match(/s\d{3}s/gi)); //S[Three Number]S
```

هذا الجزء يدور على السلاسل التي تبدأ وتنتهي بحرف "S" وبينهم 3 أرقام.

```
console.log(serials.match(/s\d{4,5}s/gi)); //S[Four Or Five Number]S
```

هون بنبحث عن السلاسل التي تبدأ وتنتهي بحرف "S" وبينهم 4 أو 5 أرقام.

```
console.log(serials.match(/s\d{4,}s/gi)); //S[At Least Four]S
```

هذا بيدور على السلاسل التي تبدأ وتنتهي بحرف "S" وبينهم 4 أرقام أو أكثر (أي ما في حد أقصى للأرقام).

---

## 126- Regular Expression → Quantifiers → Part Three

\$ → End With Something

^ → Start With Something

?= → Followed By Something

?! → Not Followed By Something

```
let myString = "We Love Programming";
```

```
let names = "10samaZ 2AhmedZ 3Mohammed 4MoustafaZ 5GamalZ";
```

```
console.log(/ing$/gi.test(myString));
```

بمعنى شوفي المتغير اذا ينتهي ب ing

```
console.log(/^We/gi.test(myString));
```

بمعنى شوفي المتغير اذا بداء ب We

```
console.log(/lz$/gi.test(names));
```

```
console.log(/^d/gi.test(names));
```

```
console.log(names.match(/d\w{5}{?=Z}/gi));
```

اعطاك اوامر وقلك اذا يكون نهايتها Z اعطيني اياها

```
console.log(names.match(/\d\w{8}{?!Z}/gi));
```

اعطاك اوامر وقلك اذا يكون ما بتنتهي Z اعطيني اياها

(^ هاذي العلامة لما تكون بين [ ] يعني نفي بمعنى اعطيني كل شيء الا الداخول القوس لما تكون خارجهم بمعنى اذا تبتداء )

#### 127- Regular Expression → Replace / ReplaceAll

```
let txt = "We Love Programming And @ Because @ Is Amazing";
```

```
console.log(txt.replace("@", "Javascript"));
```

بتبدل @ ب javascript باول رمز بتلاقه فقط

```
console.log(txt.replace(/@/gi, "Javascript"));
```

بتبدل @ ب javascript بكل رموز @

```
console.log(txt.replaceAll("@", "Javascript"));
```

بتبدل @ ب javascript بكل رموز @

---

#### 128- Regular Expression → Input Form Validation Practice

→ Html ←

```
<form action="" id="register" method="get">
<input type="text" id="phone" name="the-phone" maxlength="15">
<input type="submit" value="register">
</form>
```

→ JavaScript ←

```
document.getElementById("register").onsubmit = function () {
  let phoneInput = document.getElementById("phone").value;
  let phoneRe = /\(\d{4}\)\s\d{3}-\d{4}/; // (1234) 567-8910
  let validationResult = phoneRe.test(phoneInput);
  if (validationResult === false) {
    return false;
  }
  return true;
}
```

};

## ← الشرح →

الجزء الذي يهتم فيه هو شكل رقم الهاتف المدخل، بحيث يتبع النمط التالي:

- 1234 567-8910، يعني لازم رقم الهاتف يبدأ بأربع أرقام داخل أقواس، متبوع بمسافة، بعدها ثلاث أرقام، بعدها شرطة، وأخيراً أربع أرقام.

الكود يعمل كالآتي:

1. `phoneNumber` يأخذ قيمة الحقل الذي دخله المستخدم.
2. يعتمد على: `regular expression \(\d{4}\)\s\d{3}-\d{4}`
  - `\(\d{4}\)` يعني يبدأ بأربع أرقام داخل أقواس.
  - `\s` بعدها مسافة.
  - `\d{3}` بعدها ثلاث أرقام.
  - `-` بعدها شرطة.
  - `\d{4}` وآخر شيء أربع أرقام.
3. `test()` يبيشوف إذا المدخل (`phoneNumber`) مطابق للنمط المحدد.
4. إذا التحقق فشل بيرجع `false`، وهيك رح يمنع إرسال النموذج، وإذا كان التحقق ناجح، بيكمل ويبسمح بإرسال النموذج.

## 129-Exame :-Regular Expression Challenge ➔ video 146

```
let url1 = "elzero.org";
let url2 = "http://elzero.org";
let url3 = "https://elzero.org";
let url4 = "https://www.elzero.org";
let url5 = "https://www.elzero.org:8080/articles.php?id=100&cat=topics";
console.log(url(1-5).match(re));
```

حلي انا

```
let re = /(https?:\/\/?\w+)?.\w+.(w+)?(:\d{4})\/\w+.php?id=\d{2,}&\w+=\w+)?/gi;
```

الشرح

**https?:** الجزء الأول يتحقق إذا كانت بداية الرابط تحتوي على "http" أو "https". علامة الاستفهام بعد الـ "s" معناها أن وجود الـ "s" اختياري.

**//?:**

علامة الـ "?" بعد "///" معناها أن وجود الـ "///" بعد "http" أو "https" اختياري.

**\w+:**

يتحقق من وجود كلمات بعد الـ "http://" أو "https://"، مثل "www".

**.\w+:**

الجزء هذا يتحقق إذا كان هناك أي كلمة بعدها نقطة، مثل "elzero".

**.org?:**

يتحقق إذا كانت نهاية الـ URL هي ".org" أو لا.

```
(:\d{4})\/\w+.php?id=\d{2,}&\w+=\w+)?:
```

هذا الجزء يختبر إذا كان الرابط يحتوي على port ورقم وأي بيانات إضافية (مثل "articles.php?id=100&cat=topics"). وجود هذا الجزء اختياري.

حل بشكل أدق و أوضح لو في حال كان في أكثر من نوع روابط

```
let url6 = "https://chatgpt.com/c/9c2e339b-188c-4d7f-97e9-c0e27ad189fb";
let url7 = "https://www.youtube.com/watch?v=pr7EstDv_tg&ab_channel=ElzeroWebSchool";
console.log(url(6-7).match(ree));
```

الحل

```
let ree = /https?:\/\/(www\.)?\w+\.\w+(\.\/\w+)*(\?\/\w+=\w+(\&\w+=\w+)*)?/gi;
```

الشرح

https?:

بتدور على "http" أو "https".

\/(www\.)?:

بتدور على "www." الذي هو اختياري.

\w+\.\w+:

بيدور على اسم الدومين زي youtube.com أو google.com.

(\/\w+)\*:

بيدور على أي مسارات في الرابط زي "/watch" أو "/articles".

(\?\/\w+=\w+(\&\w+=\w+)\*)?:

بيدور على query parameters زي "v=pr7EstDv\_tg" أو "id=100".





# OOP – JAVASCRIPT

ELZERO WEB SCHOOL

VIDEO (147)

### 130-Constructor Function

بتعمل func وبتعطيها أسم متغير هاي الطريقة بتسهل عليك كثير (بتعمل نموذج ومن خلاله بتقدر تشتغل وتكمل شغلك)\*

```
function User(id, username, salary) {  
  this.i = id;  
  this.u = username;  
  this.s = salary + 1000;  
}  
  
let userOne = new User(100, "Elzero", 5000);  
let userTwo = new User(101, "Hassan", 6000);  
let userThree = new User(102, "Sayed", 7000);  
console.log(userOne.i);  
console.log(userOne.u);  
console.log(userOne.s);  
console.log(userTwo.i);  
console.log(userThree.s);
```

- function User(id, username, salary) :

هاي دالة بنسُميها "constructor"، بنستخدمها عشان ننشئ object جديد كل ما نستدعيها. بتأخذ 3 قيم: id، اسم المستخدم (username)، والراتب (salary).

-this.i = id; :

هاي ببساطة بتخلي خاصية داخل ال object اسمها i وتخزن فيها القيمة اللي بتمررها بالدالة، اللي هي id.

بهالطريقة بتقدر تتعامل مع أكثر من object بسهولة باستخدام نفس الدالة.

### 131- Constructor Function ➔ New Syntax

الشكل لجديد منها وبدون Func وهيئ أسهل

```
class User {
```

- class User : هنا بنعرف class اسمها User. ال class عبارة عن قالب أو نموذج بنستخدمه لننشئ Objects.

```
  constructor(id, username, salary) {
```

```

this.i = id;
this.u = username;
this.s = salary + 1000;
} }

```

هذا هو الباني (constructor) للـ class. يعني لما ننشئ object جديد من الـ class، هاي الدالة بتنادي. بتأخذ 3 قيم: id، اسم المستخدم، والراتب.

```

let userOne = new User(100, "Elzero", 5000);
console.log(userOne.i);
console.log(userOne.u);
console.log(userOne.s);
console.log(userOne instanceof User);

```

هاي بتتحقق إذا كان userOne هو من نوع User. إذا صح، رح ترجع true.

```

console.log(userOne.constructor === User);

```

هاي بتتحقق إذا كان الباني (constructor) للـ userOne هو نفس الباني للـ class User. إذا صح، رح ترجع true.

هيك بتكون فهمت كيف تعمل الـ class وكيف تنشئ منها objects وتستخدمها.

## 132-- Constructor Function → Deal With Properties And Methods

```

class User {
  constructor(id, username, salary) {
    this.i = id;
    this.u = username || "Unknown";
    this.s = salary < 6000 ? salary + 500 : salary;
    this.msg = function () {
      return `Hello ${this.u} Your Salary Is ${this.s}`;
    };
  }
}

```

هنا إذا ما تم تمرير اسم المستخدم (username)، رح تخزن "Unknown" كاسم. يعني لو username كان undefined أو فارغ، رح تكون القيمة "Unknown".

هنا بنعرف دالة (method) داخل الـ constructor. هاي الدالة ترجع رسالة تحتوي على اسم المستخدم وراتبه.

```

}
writeMsg() {
  return `Hello ${this.u} Your Salary Is ${this.s}`;
}

```

هاي دالة أخرى معرفه داخل الـ class. ترجع نفس الرسالة مثل `msg`، بس مكتوبة بشكل مختلف.

```
let userOne = new User(100, "Suhaib", 5000);
console.log(userOne.i);
console.log(userOne.u);
console.log(userOne.s);
console.log(userOne.msg());
console.log(userOne.writeMsg());
console.log(userOne.msg); //Native Code
console.log(userOne.writeMsg); //Native Code
```

---

### 133- Constructor Function ➔ Update Properties Built In Constructors

```
class User {
  constructor(id, username, salary) {
    this.i = id;
    this.u = username;
    this.s = salary;
  }
  updateName(newName) {
    this.u = newName;
  }
}
```

المتغير الي عرفناه من قبل للاسم حظيت function بقدر المستخدم يغير الاسم حسب شو بدخل على الموقع

```
let userOne = new User(100, "Suhaib", 5000);
userOne.updateName("Omar");
console.log(userOne.u);
console.log(userOne.s);
let strOne = "Elzero";
let strTwo = new String("Elzero");
console.log(typeof strOne);
console.log(typeof strTwo);
```

```
console.log(strOne instanceof String);
```

`instanceof` هو أوبريتر بنستخدمه للتأكد إذا كان الكائن (object) هو instance من كلاس معين. لما نستخدم `instanceof` لا على `strOne`، رح يعطينا `false` لأنه `strOne` هو سترينغ عادي مش كائن.

```
console.log(strTow instanceof String);
```

أما لما نستخدم `instanceof` على `strTow`، رح يعطينا `true` لأنه `strTow` هو كائن من نوع `String`.

---

### 134-- Constructor Function ➔ Static Properties And Methods

```
class User {  
  //Static Property  
  static count = 0;  
  constructor(id, username, salary) {  
    this.i = id;  
    this.u = username;  
    this.s = salary;  
    User.count++;  
  }  
  //Static Methods  
  static sayHello() {  
    return `Hello From class`;  
  }  
  static countMembers() {  
    return `${this.count} Members Created36`;  
  }  
  let userOne = new User(100, "Suhaib", 5000);  
  let userTow = new User(101, "Elzero", 6000);  
  console.log(userOne.count);  
  console.log(User.count);  
  console.log(User.sayHello());  
  console.log(User.countMembers());
```

### الشرح

أول إشي عندنا كلاس اسمه **User** فيه خاصية ثابتة (static property) اسمها **count**، هاي الخاصية بتبدأ من الصفر. بعدين عندنا الكونستركتور (constructor) بياخذ ثلاث معطيات (parameters): **id**، **username**، و **salary**، وبيخزنهم في خصائص الكلاس باستخدام **this**. بعدين بيزيد قيمة **count** بواحد كل ما نعمل إنستانس (instance) جديد من الكلاس. عندنا كمان دوال ثابتة (static methods)، الأولى اسمها **sayHello** بترجع جملة "Hello From class"، والثانية اسمها **countMembers** بترجع عدد الأعضاء اللي تم إنشاؤهم. بعدين بنعمل إنستانسين (instances) من الكلاس **userOne** و **userTwo** ونعطيهم قيم مختلفة. لما نطبع **userOne.count**، ما رح يعطينا إشي لأنه **count** خاصية ثابتة وما بتتبع للإنستانس. لما نطبع **User.count**، رح يعطينا عدد الأعضاء اللي تم إنشاؤهم. لما نطبع **User.sayHello**، رح يعطينا الجملة "Hello From class". ولما نطبع **User.countMembers**، رح يعطينا عدد الأعضاء اللي تم إنشاؤهم.

---

## 135 - Class Inheritance

```
class User {
    constructor(id, username) {
        this.i = id;
        this.u = username;
    }
    sayHello() {
        return `Hello ${this.u}`;
    }
}

class Admin extends User {
    constructor(id, username, permissions) {
        super(id, username);
        this.p = permissions;
    }
}

class SuperAdmin extends Admin {
    constructor(id, username, permissions, anility) {
        super(id, username, permissions);
        this.a = anility;
    }
}
```

```

let SuperAdminOne = new SuperAdmin(111, "Ahmad", 2, "yes");
let adminOne = new Admin(110, "Omar", 1);
let userOne = new User(100, "Suhaib");
console.log(SuperAdminOne.u);
console.log(adminOne.u);
console.log(adminOne.sayHello());

```

عندك ثلاث كلاسات : User ، Admin ، و SuperAdmin . كل كلاس بيرث من الكلاس اللي قبله ويبضيف خصائص جديدة  
 User عنده id و username وميثود sayHello  
 Admin بيرث من User ويبضيف خاصية permissions.  
 SuperAdmin بيرث من Admin ويبضيف خاصية anility  
 أنشأنا كائنات من كل كلاس وطبعنا بعض المعلومات عنهم باستخدام الميثودز والخصائص اللي عرفناها

### 135 - Class Encapsulation

ببساطة، encapsulation في البرمجة هو إنك تحمي بيانات الكلاس (class) بحيث ما حدا يقدر يغيرها أو يوصلها إلا من خلال دوال (methods) معينة. في جافاسكربت بنستخدم private fields عشان نحقق مفهوم الـ encapsulation.

```

class User {
  #e;
  constructor(username , eSalary) {
    this.u = username;
    this.#e = eSalary; }
  getSalary() {
    return parseInt(this.#e);}
}

let userOne = new User( "Omar" , "5000 Suhaib");
console.log(userOne.getSalary() * 0.3);

```

أول إشي عنا كلاس اسمه User جواته في متغيرين واحد منهم خاص اللي هو #e يعني بس داخل الكلاس بنقدر نوصله المتغير الثاني اللي هو u هو عام فانت ممكن توصل له من برة لما تعمل كائن من هالكلاس بتحط له اسم مستخدم وشي اسمه eSalary اللي هو الراتب بس الراتب هنا مكتوب مع نص يعني مش رقم صافي عشان هيك لما بدك تجيب الراتب بنستخدم دالة اسمها getSalary هاي الدالة بتأخذ قيمة الراتب اللي جوات المتغير الخاص وتحوّلها لرقم صحيح باستخدام parseInt

بعدين لما بتعمل العملية الحسابية `console.log userOne.getSalary` مضروبة ب ٣٠٪ يعني قاعد بتحسب نسبة من الراتب

### شرح دقيق ل Encapsulation

المتغير "الخاص" (`private`) يعني إنه ما بنقدر نوصل له أو نعدله من خارج الكلاس. في الكود اللي عندك، المتغير `e##` هو متغير خاص لأنه محطوط قبله إشارة `##`. هاي الإشارة بتدل إنه هاد المتغير خاص وما حد يقدر يتعامل معه إلا من داخل الكلاس نفسه.

لو حاولت توصل للمتغير `e##` من برة الكلاس زي هيك:

```
;console.log(userOne.#e)
```

راح يطلع لك `**Error**` لأن المتغير محمي. الفائدة من هاي الحركة هي إنك تمنع أي حدا من خارج الكلاس إنه يغير أو يشوف هاد المتغير مباشرة، وبك تخليهم يستخدموا دوال معينة مثل `getSalary` عشان يوصلوا للبيانات بطريقة منظمة.

فالخاص هو زي صندوق مغلق، ما حد بقدر يفتحه إلا إذا عنده المفتاح، والمفتاح هون هو الدوال اللي بتسمحك توصل للبيانات.

---

### 136-Prototype ➔ Add To Prototype Chain & Extend

```
class User {  
  constructor(id, userName) {  
    this.i = id;  
    this.u = userName; }  
  sayHello() {  
    return parseInt(this.u);  
  }  
}  
  
let userOne = new User(100, "Elzero");  
  
console.log(userOne.u);  
  
console.log(User.prototype);  
  
console.log(userOne);  
  
User.prototype.sayWelcome = function () {  
  return `Welcome ${this.u}`;  
};
```

لما نضيف دالة للـ `prototype`، هاي الدالة بتكون متاحة لكل `objects` اللي بننشئهم من الكلاس `User`.

مثال: إذا عندك `object` مثل `userOne` اللي أنشأناه من الكلاس `User`، رح يقدر يستخدم دالة `sayWelcome` مباشرة بدون الحاجة لإضافتها للـ `object` نفسه. يعني لما تنادي `userOne.sayWelcome()`، رح تشتغل الدالة حتى لو ما كانت موجودة مباشرة داخل الـ `object`، لأنه رح يدور عليها في الـ `prototype`.



```
Object.prototype.love = "Elzero Web School";
String.prototype.addDotBeforeAndAfter = function (val) {
  return `.${this}.`;};
```

هون أضفنا دالة للـ `String.prototype` اسمها `addDotBeforeAndAfter`.

كل النصوص (strings) في JavaScript بقدرنا يستخدموا هاي الدالة، لأنه كل النصوص بتورث من `String`.  
 مثلاً، لما ننادي `myString.addDotBeforeAndAfter()`، رح نتضاف نقاط قبل وبعد النص لأنه `myString` هو عبارة عن نص (string) وورث الدالة من الـ `String.prototype`.

```
let myString = "Suhaib Halabe";
console.log(myString.addDotBeforeAndAfter());
```

- الـ `prototype` هو طريقة تخليك تشارك دوال أو خصائص بين كل الكائنات (objects) اللي بتتنتمي لنفس الـ class أو نفس النوع.
- لما تضيف شيء للـ `prototype`، الكائنات بتقدر تستفيد منه حتى لو ما كان موجود داخلها مباشرة.
- كل كائن في JavaScript عنده `prototype` معين بقدر يورث منه دوال وخصائص.
- هيك بتقدر تفهم كيف كل object بيستخدم الدوال والخصائص اللي موجودة في الـ `prototype` بدون ما

## 137-Object Meta Data And Descriptor

شرح كامل ولكن سوف يتم تجزئة الشرح مع Elzero Web school

في JavaScript، `Object Meta Data` و `Descriptors` هما مفهومين مهمين لتحكم أعمق بخصائص الكائنات (objects) من خلالهما، بتقدر نحدد كيفية عمل الخصائص بداخل الكائنات، مثل ما إذا كانت قابلة للتعديل، مرئية، أو قابلة للكتابة.

### 1.Object Meta Data (البيانات الوصفية للكائن)

كل خاصية (property) في كائن يتمثل مجموعة من القيم الوصفية أو "الميتاداتا". هاي البيانات الوصفية بتحدد سلوك الخاصية، واللي بنسميها "Descriptors".

### 2. Property Descriptors (وصف الخاصية)

وصف الخاصية في JavaScript بيعبر عن مجموعة من الخصائص اللي بتحدد كيفية تعامل الكائن مع الخصائص (properties). لكل خاصية في JavaScript بيكون عندها Descriptor، وهو عبارة عن معلومات توضح كيف بتعمل هاي الخاصية.

أنواع الـ Descriptors:

في نوعين رئيسيين من الخصائص في JavaScript:

#### 1. Data Properties (الخصائص البيانية):

هاي الخصائص يتمثل القيم العادية اللي بتخزن فيها البيانات. لكل خاصية بيانات في JavaScript عندها الـ descriptors التالية:

- value: لقيمة الحالية للخاصية.

- **writable:** إذا كانت **true** ، يعني بنقدر نعدل قيمة الخاصية. إذا كانت **false** ، بنقدر نقرأ الخاصية لكن ما بنقدر نعدلها.
- **enumerable:** إذا كانت **true** ، يعني الخاصية رح تظهر في عمليات التكرار مثل **for...in**. إذا كانت **false** ، ما بتظهر.
- **configurable:** إذا كانت **true** ، يعني بنقدر نحذف الخاصية أو نعدل الـ **descriptors** تبعها. إذا كانت **false** ، ما بنقدر نعدلها أو نحذفها.

## 2. Accessor Properties (خصائص الوصول):

هاي الخصائص بتستخدم دوال (**getters and setters**) لتحديد القيم، بدلاً من تخزين القيم مباشرة. إلهـ **descriptors** التالية:

- **get:** دالة بتستخدم لجلب قيمة الخاصية.
- **set:** دالة بتستخدم لتحديد أو تغيير قيمة الخاصية.
- **enumerable:** نفس الشيء، بتحدد إذا كانت الخاصية تظهر بالتكرار.
- **configurable:** نفس الشيء، بتحدد إذا كانت الخاصية قابلة للتعديل أو الحذف.

## 3. استخدام **Object.defineProperty()**

عشان نتحكم بوصف الخصائص في الكائنات، بنستخدم الدالة **Object.defineProperty()**. هاي الدالة بتخلينا نحدد الـ **descriptors** لأي خاصية في الكائن.

### مثال على **Data Property**:

```
let person = { name: "Suhaib" };
Object.defineProperty(person, "age", {
  value: 30,
  writable: false, // ممنوع تعديل القيمة
  enumerable: true, // الخاصية بتظهر عند التكرار
  configurable: false // ما بنقدر نغير أو نحذف الخاصية
});
console.log(person.age); // 30
person.age = 35; // محاولة تعديل القيمة ما بتنجح
console.log(person.age); // 30 رح يضل
```

- هون أضفنا خاصية **age** للكائن **person**، وخلينا الـ **writable: false**، يعني ممنوع نعدل قيمة **age** بعد تعريفها.

## نعيد الشرح مع Elzero Web School

## 138-Object Meta Data And Descriptor → Part One → Writable & Enumerable & Configurable

```
const myObject = {  
  a: 1,  
  b: 2,};  
Object.defineProperty(myObject, "c", {  
  writable: false,  
  enumerable: true,  
  configurable: false,  
  value: 3,  
});
```

استخدام `Object.defineProperty` أضفنا خاصية جديدة `c` للكانن `myObject` باستخدام `Object.defineProperty`، مع تحديد الخيارات التالية:

- `writable: false` يعني ما بنقدر نعدل قيمة الخاصية `c` بعد تعيينها.
- `enumerable: true` الخاصية `c` تظهر في عمليات التكرار (مثل `for...in`).
- `configurable: false` يعني ما بنقدر نحذف الخاصية أو نعدل الـ `descriptors` بعد تعيينها.
- `value: 3` قيمة الخاصية `c` هي 3.

```
myObject.c = 100;
```

حاولنا تغيير قيمة `c` إلى 100، لكن بسبب أن `writable` تم تعيينها إلى `false`، القيمة `c` تظل 3 وما تتغير.

```
console.log(delete myObject.c)
```

حاولنا نحذف الخاصية `c`، لكن بسبب أن `configurable` تم تعيينها إلى `false`، الحذف ما رح يتم، والنتيجة `c` تكون `false`.

```
for (let prop in myObject) {  
  console.log(prop, myObject[prop]);}
```

بما أن `enumerable` تم تعيينها إلى `true`، الخاصية `c` تظهر مع الخصائص الثانية في حلقة `for...in`.  
بالتالي، حلقة التكرار `c` تعرض:

c 3 | b 2 | a 1

```
console.log(myObject);
```

حاولنا تغيير قيمة `c` بس ما قدرنا لأنه `writable: false` و حاولنا نحذف `c` بس ما انحدفت لأنه `configurable: false` و الخاصية `c` ظهرت في التكرار لأن `enumerable: true`.

### 139-Object Meta Data And Descriptor → Part Tow → Define Multiple Properties & Check Descriptors

```
const myObject = {  
  a: 1,  
  b: 2, };  
Object.defineProperty(myObject, {  
  c: {  
    configurable: true,  
    value: 3, },  
  d: {  
    configurable: true,  
    value: 4, },  
  e: {  
    configurable: true,  
    value: 5, },  
});  
console.log(myObject);  
console.log(Object.getOwnPropertyDescriptor(myObject, "d"));
```

هذا السطر يستخدم `Object.getOwnPropertyDescriptor` عشان يعرض الوصف (descriptor)

```
console.log(Object.getOwnPropertyDescriptors(myObject));
```

هذا السطر يستخدم `Object.getOwnPropertyDescriptors` لعرض الوصف الكامل لكل الخصائص في الكائن `myObject`.

الخصائص اللي أضفناها (c, d, e) عندهم `configurable: true`، لكن ما حددنا لهم `writable` أو `enumerable`، فافتراضياً رح يكونوا:

- `writable: false` يعني ما بنقدر نغير القيم بعد تعيينها.
- `enumerable: false` يعني ما بيظهروا في عمليات التكرار مثل `for...in`.

## 140-Date And Time

ببداء حساب الايام و الوقت من تاريخ 1970

```
let dateNow = new Date();
console.log(dateNow);

console.log(Date.now());

let seconds = Date.now() / 1000;

console.log(`seconds: ${seconds}`);

let minutes = seconds / 60;
console.log(`Minutes: ${minutes}`);

let horus = minutes / 60;
console.log(`Hours: ${horus}`);

let days = horus / 24;
console.log(`days: ${days}`);

let Years = days / 365;
console.log(`Years: ${Years}`);
```

#### 141-Date And Time → GetTime & GetDate & getFullYear & GetMonth & getDay & getHours&.....

```
let dateNow = new Date();
```

```
let birthday = new Date("Oct 25 , 82");
```

```
let dateDiff = dateNow - birthday;
```

```
console.log(dateDiff / 1000 / 60 / 60 / 24 / 365);
```

```
console.log(dateNow);
```

```
console.log(dateNow.getTime());
```

```
console.log(dateNow.getDate());
```

```
console.log(dateNow.getFullYear());
```

```
console.log(dateNow.getMonth() + 1);
```

```
console.log(dateNow.getDay());
```

```
console.log(dateNow.getHours());
```

```
console.log(dateNow.getMinutes());
```

```
console.log(dateNow.getSeconds());
```

---

#### 142-Date And Time

```
let dateNow = new Date();
```

```
console.log(dateNow);
```

```
dateNow.setTime(0);
```

```
console.log(dateNow);
```

```
dateNow.setTime(10000);
```

```
console.log(dateNow);
```

```
dateNow.setDate(35);
```

```
console.log(dateNow);
```

```
dateNow.setFullYear(2024,8);
```

```
console.log(dateNow);
```

```
dateNow.setMonth(1);
```

```
console.log(dateNow);
```

**143-Date And Time → New Date(timestamp & Date String & Numeric Values)**

```
let date1 = new Date(0);
```

```
console.log(date1);
```

```
let date2 = new Date(1025298000000);
```

```
console.log(date2);
```

```
let date3 = new Date("06/29-2002");
```

```
console.log(date3);
```

```
let date4 = new Date("2002-06");
```

```
console.log(date4);
```

```
let date5 = new Date("02");
```

```
console.log(date5);
```

```
let date6 = new Date(2002, 6, 29, 11, 29, 0);
```

```
console.log(date6);
```

```
console.log(Date.parse("jun 29 2002"));
```

---

**144-Date And Time → Track Operations Time**

```
let start = new Date();
```

```
for (let i = 0; i < 100; i++) {
```

```
document.write(`<div>${i}</div>`);
```

```
}
```

```
let end = new Date();
```

```
let duration = end - start;
```

```
console.log(duration);
```

لحساب كم يحتاج وقت لإنشاء Div

## 145-Generators

الـ **\*\*Generators\*\*** في JavaScript هي نوع خاص من الدوال التي بتقدر توقف تنفيذها بشكل مؤقت وترجع قيمة معينة، وبعدين تكمل من نفس المكان الذي وقفت عنده. بنعرف الـ Generators باستخدام الكلمة المفتاحية **\*function\*** والتي بتعني "دالة مولدة".

أهم مميزات الـ: Generators

- بتقدر **\*\*توقف التنفيذ\*\*** باستخدام **`yield`**.

- بتسمح **\*\*بالتحكم بالتدفق\*\*** داخل الدالة، يعني مش لازم تنفذها مرة وحدة، ممكن تنفذها خطوة بخطوة.

- بترجع كائن خاص اسمه **Iterator** ، والتي بنقدر نستخدمه عشان ننادي القيم خطوة بخطوة.

كيف نكتب **Generator** ؟

بنكتب الـ **Generator** باستخدام **\*function\*** بدلاً من **`function`** العادية.

مثال بسيط:

```
function* myGenerator() {  
  توقف وترجع 1 // yield 1;  
  توقف وترجع 2 // yield 2;  
  توقف وترجع 3 // yield 3;  
}  
  
let gen = myGenerator(); // Iterator بترجع  
console.log(gen.next()); // { value: 1, done: false }  
console.log(gen.next()); // { value: 2, done: false }  
console.log(gen.next()); // { value: 3, done: false }  
console.log(gen.next()); // { value: undefined, done: true }
```

شرح:

1-تعريف الـ: Generator

- الدالة **`myGenerator`** فيها ثلاث قيم. **(1, 2, 3)** **`yield`**

**`yield`** - هي الكلمة التي بتوقف تنفيذ الدالة وترجع القيمة التي بعدها.

2 - استخدام الـ: Generator

- لما ننادي **`myGenerator()`** بيرجع كائن اسمه **\*\*Iterator\*\***.



- بعدين بنستخدم `next()` عشان نرجع القيم خطوة بخطوة.
- أول `next()` بترجع `{ value: 1, done: false }` ، والقيمة هي `1`.
- ثاني `next()` بترجع `{ value: 2, done: false }` ، وهكذا حتى توصل إلى `done: true` لما تخلص القيم.

`done` خاصية:

`done: false` - يعني ما خالصنا القيم، ولسه فيه قيم تانية بالـ generator.

`done: true` - يعني خالصنا كل القيم في الـ generator.

استخدام عملي:

```
function* range(start, end) {
  for (let i = start; i <= end; i++) {
    yield i; // توقف وترجع
  }
}

let numbers = range(1, 5); // 5 إلى 1
console.log(numbers.next().value); // 1
console.log(numbers.next().value); // 2
console.log(numbers.next().value); // 3
console.log(numbers.next().value); // 4
console.log(numbers.next().value); // 5
console.log(numbers.next().done); // true
```

شرح المثال:

`.for` داخل حلقة `yield` باستخدام `end` إلى `start` بتولد أرقام من `range` الدالة -

، بترجع كل رقم على حدة `numbers.next()` لما ننادي -

إيقاف واستكمال التنفيذ:

هي القدرة على إيقاف التنفيذ واستكمالها لاحقاً. هذا بيبكون مفيد في التعامل مع البيانات اللي Generators واحدة من أقوى ميزات الـ `infinite loops` بتيجي بشكل تدريجي أو في بناء خوارزميات معقدة مثل

مثال على infinite generator

```
function* infiniteNumbers() {
  let i = 0;
  while (true) {
```

```

    yield i++; // ما لا نهاية}}
    رح يستمر في توليد أرقام إلى ما لا نهاية}}

let numGen = infiniteNumbers();

console.log(numGen.next().value); // 0

console.log(numGen.next().value); // 1

console.log(numGen.next().value); // 2

```

**Generators** - يمكننا من توليد قيم بشكل تدريجي واستخدامها خطوة بخطوة .

-نوقف التنفيذ باستخدام `yield` ونقدر نكمل من نفس النقطة باستخدام `next()`

-مفيدة في التعامل مع البيانات التي بتوصل بشكل متتابع أو في بناء حلقات لا نهائية.

هيك بتقدر تستخدم الـ **Generators** في السيناريوهات التي بتحتاج فيها لتحكم أكبر بتدفق البيانات أو التنفيذ.

شرح Elzero Web School

```

function* generateNumbers() {

  yield 1;

  console.log("Hello After Yield 1");

  yield 2;

  yield 3;

  yield 4;}

let generator = generateNumbers();

console.log(generator);

console.log(generator.next().value);

console.log(generator.next().value);

console.log(generator.next().value);

console.log(generator.next().value);

for (let value of generateNumbers()) {

  console.log(value);}

```

تعريف الـ **Generator** : الدالة `generateNumbers` تستخدم `yield` لتوليد قيم مؤقتة. كل مرة تستدعي فيها `next()` بترجع القيمة الحالية وتوقف التنفيذ لحد ما تستدعي `next()` مرة ثانية.

تخزين الـ **Generator**: لما تخزن الـ **generator** في متغير، بتحصل على كائن من نوع `Iterator` ، وبتقدر تستدعي القيم من خلاله باستخدام `next()`

استدعاء `next()` كل استدعاء لـ `next()` بيولد قيمة جديدة حسب ترتيب الـ `yield` بالدالة. بعد أول `yield` ، بيوقف التنفيذ، ويكمل لما تستدعي `next()` مرة ثانية.

حلقة `for...of` تستخدم الـ `generator` تلقائيًا للتكرار على القيم التي بتولدها الدالة، بدون الحاجة لاستدعاء `next()` يدويًا، وبتعرض القيم بالترتيب.

#### 146- Generators → delegate Generator

```
function* generateNums() {
  yield 1;
  yield 2;
  yield 3; }

function* generateLetters() {
  yield "A";
  yield "B";
  yield "C"; }

function* generatorAll() {
  yield* generateNums();
  yield* generateLetters();
  yield* [4, 5, 6]; }

let generate = generatorAll();
console.log(generate.next());
console.log(generate.next());
console.log(generate.next());
console.log(generate.next());
console.log(generate.return("Z"));
console.log(generate.next());
```

أول إشي عندك ثلاث دوال مولدة أو `Generators`

`generateNums`: هاي الدالة بتولد الأرقام 1 2 3 باستخدام `yield`

`generateLetters`: هاي الدالة بتولد الأحرف A B C بنفس الطريقة باستخدام `yield`

`generatorAll`: هاي الدالة بتستخدم `yield*` عشان تستدعي الدوال الثانية وتولد منها القيم يعني هي زي حلقة بتجمع القيم من الدوال الثانية وتضيف كمان القيم `[4, 5, 6]`

لما نجي على الجزء اللي بنادي فيه الـ `generator`

```
let generate = generatorAll();
```

هون بنخزن الـ `generator` اللي بيجمع كل القيم من الدوال الثانية في متغير اسمه `generate`

بعدين نستخدم `next()` عشان نطلع القيم وحدة وحدة

`generate.next()`: أول استدعاء رح يرجع أول قيمة من `generateNums`` اللي هي 1

`generate.next()`: ثاني استدعاء يرجع ثاني قيمة اللي هي 2

`generate.next()`: ثالث استدعاء يرجع ثالث قيمة اللي هي 3

`generate.next()`: رابع استدعاء ببداً يجيب من `generateLeters`` ويبرجع أول حرف اللي هو A

`generate.return("Z")`` هون بنستخدم `return()` عشان ننهي الـ generator ونرجع القيمة Z وبوقف التنفيذ

`generate.next()`: حتى لو استدعينا `next()` بعد هيك رح يكون التنفيذ انتهى والنتائج رح يكون `{ value: undefined, done: true }`

---

## 147-Generators ➔ Infinite Numbers & User Return Inside Generators

```
function* generateNumbers() {
```

```
  let index = 0;
```

```
  while (true) {
```

```
    yield index++;
```

```
  }
```

```
}
```

```
let generate = generateNumbers();
```

```
console.log(generate.next());
```

```
console.log(generate.next());
```

```
console.log(generate.next());
```

```
console.log(generate.next());
```

هون عندك دالة مولدة اسمها `generateNumbers` وفكرتها إنها بتولد أرقام لا نهائية لأنك مستخدم `while(true)`` اللي هي حلقة بتضل تشتغل للأبد فكل مرة بتنادي `next()` بتطلع لك رقم جديد

أول ما تبدأ `index`` بتكون صفر بعدين أول `yield`` بيرجع قيمة `index`` اللي هي صفر وبعدين بيزيد `index`` واحد عشان المرة الجاي ترجع واحد وهكذا

لما تخزن الدالة المولدة في متغير `generate`` بتقدر تنادي `next()` كل مرة عشان تجيب رقم جديد

أول `next()` رح يرجع لك `{ value: 0, done: false }` اللي هو أول رقم وثاني `next()` يرجع `{ value: 1, done: false }` وهكذا

## 148-Modules ➔ Import And Export ➔ Namer VS Default Import And Export All

بجاء اسكربت الموديولز بتمكنك تقسم الكود تبعك لملفات متعددة كل ملف ممكن يحتوي جزء معين من الكود وبتقدر تشارك الكود بين الملفات باستخدام `import` و `export`

### Export

التصدير بنعمله عشان تقدر تشارك المتغيرات أو الدوال من ملف لملف ثاني وفي نوعين من الـ `export`

-الأول اسمه `Named Export` وهاي بتقدر تصدر فيها أكثر من إشي بنفس الملف بالاسم

-والثاني اسمه `Default Export` وهاي بتصدر فيها إشي واحد افتراضي بدون ما تحتاج تسميه بنفس الاسم لما تستورده

### Named Export

يعني تصدر المتغيرات أو الدوال باسم معين ولما تستوردهم لازم تستخدم نفس الاسم

وفيه `main.js` مثال بسيط اذا كان عندك ملف اسمه

```
// file: main.js
export const a = 10;
export const arr = [1, 2, 3];
export function saySomething() {
  console.log("Hello from main.js"); }
```

هون صدرت المتغير `a` والمصفوفة `arr` والدالة `saySomething`

### Default Export

في حال بدك تصدر إشي واحد كافتراضي وما تستخدم اسمه لما تستورده مثل

```
// file: main.js
const b = 20;
export default b;
```

هون صدرت المتغير `b` كتصدير افتراضي

### Import

الاستيراد يتم لما بدك تستخدم كود من ملف ثاني

## Named Import

عشان تستورد الأشياء اللي صدرتها بالاسم بتستخدم الأقواس المعقوفة ويكتب الأسماء زي ما هي موجودة بالملف الثاني مثل

```
// file: app.js
import { a, arr, saySomething } from "./main.js";
console.log(a); // 10
console.log(arr); // [1, 2, 3]
saySomething(); // Hello from main.js
```

## Default Import

لو بذك تستورد التصدير الافتراضي ما بتحتاج أقواس معقوفة وبتقدر تسميه بأي اسم بذك مثل

```
// file: app.js
import b from "./main.js";
console.log(b); // 20
```

فيك تخلط بين الـ Named والـ Default بنفس الملف يعني تستورد تصدير افتراضي مع تصدير مسمى مثال اذا كان عندك

```
// file: main.js
export const a = 10;
export const arr = [1, 2, 3];
export default function sayHello() {
  console.log("Hello from default export"); }
```

وبالملف الثاني تستورده هيك

```
// file: app.js
import sayHello, { a, arr } from "./main.js";
sayHello(); // Hello from default export
console.log(a); // 10
console.log(arr); // [1, 2, 3]
```

النقطة المهمة انه بالـ Named لازم تستورد الأسماء نفسها زي ما صدرتها والـ Default Export بتقدر تعمله مرة وحدة بكل ملف

الملف الاول :

```
let a = 10;

let arr = [1, 2, 3, 4];

function saySomething() {
  return `something`; }

export { a as myNumber, arr, saySomething };

export default function () {
  return `hello`;}
```

هون بنقلوا انو في function وبنعطيهها export افتراضي

الملف الثاني :-

```
import Elzero,{ myNumber, arr, saySomething as s } from "./main.js";
```

عشان تستدعي ال function الافتراضي بمحطها بعد import مباشره وبتحط اي اسم بعجبك عادي بتعرف عليها بس شرط تكون بعد import

اذا بدك تغير اسم الاستدعاء بتكتب بعدها as والاسم الي بدك اياه

```
console.log(Elzero())
console.log(myNumber);
console.log(arr);
console.log(s());
import * as all from "./main.js";
```

هون احنا استدعينا كامل الخواص الي موجوده في الصفحة الاول وبنعطيهها اسم من عنا بعد كلمة as بتغير اسم الاستدعاء

```
console.log(all);
console.log(all.myNumber);
console.log(all.arr);
```

عشان تستخدم الخواص بتكتب اسم الاستدعاء مع اسم الخاصيه

# {JSON}

## JSON – JAVASCRIPT

ELZERO WEB SCHOOL

VIDEO (169)



## 149-What is JSON

الـ JSON في جافاسكربت هو اختصار لـ JavaScript Object Notation وهو طريقة لتخزين ونقل البيانات على شكل نص منظم الهدف الرئيسي منه إنه يكون سهل القراءة والكتابة بالنسبة للبشر وأيضاً سهل الفهم للبرامج وببساطة بشكل أساسي على شكل أزواج من المفاتيح والقيم يعني مثلاً في المفتاح الذي يحدد اسم الخاصية ويكون مقابل إله القيمة التي تنتمي لها الخاصية البيانات التي في JSON ممكن تكون أرقام نصوص قوائم كائنات وهالشي بيخليها مرنة جداً وتقدر تتعامل معها بسهولة لما بدك تتبادل بيانات بين السيرفر والمتصفح

يتم إنشاء ملف والامتداد تبعو يكون json.

## 150-JSON → Syntax

```
{  
  "string": "Elzero",  
  "Number": 100,  
  "object": {  
    "Eg": "giza",  
    "KSA": "Riyadh"  
  },  
  "array": ["HTML", "CSS", "JS"],  
  "boolean": true,  
  "null": null  
}
```

يتم كتابة المتغيرات بين أشارتين اقتباس

## 151-JSON → API Overview

JSON في جافا سكربت هو طريقة لتخزين ونقل البيانات بين الأنظمة زي التطبيقات والسيرفرات شغله الأساسي إنه يحول البيانات لكائنات عشان يسهل فهمها والتعامل معها فمثلاً لو عندك معلومات عن مستخدم زي اسمه وعمره بتقدر تخزنهم أو ترسلهم بصيغة JSON لأنه منظم وسهل القراءة

استخدامه كثير مهم خاصة مع API لأن السيرفر أو النظام عادة بيعت ردود بصيغة JSON بعد ما يطلب التطبيق أو المتصفح معلومات منه زي بيانات الطقس أو المنتجات

السبب إنه مستخدم على نطاق واسع هو إنه خفيف وسهل ومفهوم لأغلب لغات البرمجة ويسمح بتبادل البيانات بسرعة بين الأنظمة

## 152- JSON ➔ JSON.parse & stringify

```
const myJsonObjectFromServer = `{"UserName" : "Suhaib" , "Age" : 23}`;
```

أول إشي عندك متغير اسمه myJsonObjectFromServer وهو عبارة عن نص عادي ما يعتبره جافا سكربت كائن أو JSON لما تكتب console.log لنوعه بتطلعك إنه string يعني نص

```
console.log(typeof myJsonObjectFromServer);
```

```
console.log(myJsonObjectFromServer);
```

```
const myJsObject = JSON.parse(myJsonObjectFromServer);
```

```
console.log(typeof myJsObject);
```

```
console.log(myJsObject);
```

بعدها بنعمل تحويل لهذا النص لكائن جافا سكربت باستخدام JSON.parse بنحول النص المكتوب بصيغة JSON لكائن حقيقي بجافا سكربت هيك بتقدر تتعامل مع البيانات اللي فيه زي ما بدك مثلا تغير القيم اللي داخله زي اسم المستخدم أو العمر , `parse` هو عبارة عن عملية تحويل نص مكتوب بصيغة JSON لكائن جافا سكربت عشان تقدر تتعامل مع البيانات اللي داخله مثلا إذا جبت بيانات من سيرفر بصيغة JSON لازم تحولها باستخدام `JSON.parse` عشان تصير كائن وتقدر تعدل عليها أو تستخدمها

```
myJsObject["UserName"] = "Elzero";
```

```
myJsObject["Age"] = 25;
```

```
const myJsonObjectToServer = JSON.stringify(myJsObject);
```

```
console.log(typeof myJsonObjectToServer);
```

```
console.log(myJsonObjectToServer);
```

بعدين لما نغير البيانات بالكائن اللي حولناه بنرجع نحول الكائن المعدل مرة ثانية لنص باستخدام JSON.stringify لأنه السيرفرات عادة بتتعامل مع البيانات بصيغة نصية مش كائنات جافا سكربت هيك بنعمل نص جديد يحتوي على البيانات المعدلة اللي بنبعثها للسيرفر , أما `stringify` فهي العملية العكسية بنحول كائن جافا سكربت لنص مكتوب بصيغة JSON عشان نبعثه للسيرفر مثلا السيرفرات بتتعامل مع النصوص مش مع الكائنات لذلك لما يكون عندك كائن بيانات بدك تبعثه للسيرفر بتحوله لنص باستخدام `JSON.stringify`

هيك بتقدر تشتغل مع البيانات بشكل مرن

## 153-JSON ➔ Asynchronous & Synchronous

لما نحكي عن Asynchronous و Synchronous جافا سكربت احنا بنحكي عن كيف الكود بينفذ التعليمات

بالنسبة لـ Synchronous الكود بينفذ خطوة خطوة وما ينتقل للخطوة اللي بعدها إلا لما بخلص اللي قبله يعني لازم تنتظر كل عملية تنتهي عشان تبدأ اللي بعدها وهاي الطريقة ممكن تسبب بطء في الكود إذا فيه عمليات طويلة زي القراءة من قاعدة بيانات أو الانتظار لطلب من السيرفر

أما Asynchronous يسمح للكود ينفذ العمليات بدون ما يستنى كل عملية تخلص فمثلاً ممكن يرسل طلب للسيرفر ويكمل تنفيذ باقي الكود ولما يجي الرد من السيرفر يتعامل معه بشكل منفصل وهاد بيكون باستخدام الـ callback أو async/await أو الـ async/await وهاي الطريقة بتخلي الكود أسرع وأكثر مرونة

---

## 154 - JSON ➔ Call Stack And Web API

في جافا سكربت الـ Call Stack والـ Web API بيشتغلوا مع بعض بطريقة بتنظم كيف الكود بينفذ .

### Call Stack

الـ Call Stack هو المكان اللي جافا سكربت بتحط فيه المهام بالترتيب. لما تستدعي دالة جديدة، بتتضاف على الـ Stack ولما تخلص بتتسأل من عليه. الفكرة إنها زي كومة بتزيد عليها الوظائف وبعدين بتشيلهم بعد ما تخلص كل واحدة منهم. جافا سكربت لغة Synchronous يعني بتنفذ كل مهمة بترتيبها، وهاي بتكون عن طريق الـ Call Stack.

### Web API

الـ Web API بتوفر أدوات المتصفح زي التعامل مع الـ DOM أو الشبكة أو التوقيعات أو عمليات Asynchronous زي الـ setTimeout أو استرجاع بيانات من السيرفر. هاي العمليات لما بتشتغل ما بتوقف الكود، بتتضاف للـ Web API اللي بيشتغل عليها لحالها بدون ما يعطل باقي الكود، ولما تخلص العملية بيرجعها للـ Call Stack عشان تكمل التنفيذ.

### كيف بيشتغلوا مع بعض

لما يكون فيه كود Asynchronous زي طلب بيانات من سيرفر، المتصفح بيخرج هذا الطلب من الـ Call Stack ويبيعته للـ Web API عشان يتنفذ بالخلفية، بعد ما يخلص بيرجع النتيجة للـ Callback Queue اللي بتنتظر دورها لينتهي الكود اللي شغال في الـ Call Stack، بعددين تدخل وتنفذ.

## أمثلة

### مثال على الـ Call Stack

```
function first() {  
  console.log("First"); }  
function second() {  
  first();  
  console.log("Second"); }  
function third() {  
  second();  
  console.log("Third"); }  
third();
```

هون كل دالة بتتضاف للـ Call Stack وتتسحب لما تخلص، بيطلع `First` ثم `Second` ثم `Third` حسب الترتيب.

### مثال على Web API (Asynchronous)

```
console.log("Start");  
setTimeout(() => {  
  console.log("Inside setTimeout");  
}, 2000);  
console.log("End");
```

هون الكود بيبدأ بطباعة `Start` وبيكمل تنفيذ الكود، بعدين بيخرج `setTimeout` للـ Web API ولما يخلص الـ 2000 ميلي ثانية بيرجع النتيجة للـ Callback Queue وينفذ `Inside setTimeout` بعد ما يكون طبع `End`

## 154 - JSON → Event Loop And Callback Queue

الـ Event Loop والـ Callback Queue في جافاسكريبت هما جزء مهم من كيف تعمل جافاسكريبت كبيئة برمجية غير متزامنة

لما بنكتب كود جافاسكريبت بيتم تنفيذه بشكل متسلسل يعني سطر بعد سطر لكن في بعض العمليات مثل طلب البيانات من السيرفر أو عمليات الوقت ممكن تأخذ وقت فهون تدخل فكرة الـ Callback Queue

لما تشتغل على مهمة طويلة أو عملية غير متزامنة جافاسكريبت بتخزن العملية في الـ Callback Queue وبتكمل تنفيذ الأسطر اللي بعدها بعد ما تخلص من المهمة الأولى بتروح تفحص الـ Callback Queue وبتشوف إذا في عمليات تحتاج تنفيذ فبتبدأ تنفذها وحدة وحدة

الـ Event Loop هو اللي يدير هاي العملية هو يتأكد أنه التنفيذ دائماً يكون في السطر الحالي وبعدها ينظر للـ Callback Queue عشان يشوف إذا في شغلات جاهزة للتنفيذ

```
console.log("بداية البرنامج")
setTimeout(() => {
  console.log("تم تنفيذ الـ")
}, 2000)
console.log("نهاية البرنامج")
```

لما تشتغل الكود هذا بتشوف أول شي "بداية البرنامج" بعدين "نهاية البرنامج" وبعدها بيكون فيه تأخير لمدة ثانيتين وبعدين بتشوف "تم تنفيذ الـ" `setTimeout`

هذا بسبب أن الـ `setTimeout` كانت عملية غير متزامنة خزنت في الـ Callback Queue وجافاسكريبت كملت تنفيذ الأسطر العادية بعدين رجعت لها بعد ما انتهت من التنفيذ العادي

بهيك بكون الشرح واضح لك عن الـ Event Loop والـ Callback Queue

## 155 - AJAX

**AJAX** هي طريقة بتخليك تتواصل مع السيرفر وتجيب بيانات بدون ما تعيد تحميل الصفحة يعني مثلاً لما تستخدم موقع وتضغط على زر ليوصل لك معلومات جديدة من السيرفر لكن الصفحة تبقى زي ما هي الفكرة الأساسية إنه تقدر ترسل طلب للسيرفر وتستقبل الرد في الخلفية بينما المستخدم يقدر يتفاعل مع الصفحة بهذه الطريقة التجربة بتكون أسهل وأسرع للمستخدم ومش مضطر يستنى تحميل كامل للصفحة

---

## 156-AJAX ➔ Request And Response From Real API

```
let myRequest = new XMLHttpRequest();
myRequest.open("GET","https://api.github.com/users/elzerowebsschool/repos",true);
myRequest.send();
console.log(myRequest);
myRequest.onreadystatechange = function () {
  console.log(myRequest.readyState);
  console.log(myRequest.status);
  if (this.readyState === 4 && this.status === 200) {
    console.log(this.responseText); }
};
```

أول شي بنبدأ بإنشاء **myRequest** وهي كائن من **XMLHttpRequest** هاد الكائن بنستخدمه لنعمل طلب للسيرفر بعدها بنستخدم **myRequest.open** عشان نحدد نوع الطلب في هاي الحالة نوع الطلب هو **GET** وبحدد كمان الرابط اللي بدنا نطلب منه البيانات والـ **true** معناها الطلب غير متزامن يعني الصفحة ما رح تتوقف عن الشغل بعدين بنستخدم **myRequest.send** عشان نرسل الطلب للسيرفر بعد هيك بنطبع **myRequest** في الكونسول لنتأكد إنه الكائن انعمل بشكل صحيح الخطوة التالية بنستخدم **myRequest.onreadystatechange** عشان نحدد وظيفة بتنفيذها لما تتغير حالة الطلب داخل هالوظيفة بنطبع **myRequest.readyState** اللي بتعبر عن حالة الطلب و**myRequest.status** اللي بتعبر عن حالة الاستجابة من السيرفر بعدين بنفحص إذا كانت **readyState** تساوي 4 يعني الطلب اكتمل والـ **status** تساوي 200 يعني كان الطلب ناجح إذا الشرط تحقق بنطبع **this.responseText** اللي فيها البيانات اللي جبنها من السيرفر وبهيك بنكون عملنا طلب ناجح وعرفنا كيف نتعامل مع الاستجابة من السيرفر

## 157- Loop On Data

```
let myRequest = new XMLHttpRequest();
myRequest.open("GET","https://api.github.com/users/elzerowebsschool/repos",true);
myRequest.send();
myRequest.onreadystatechange = function () {
  if (this.readyState === 4 && this.status === 200) {
    console.log(this.responseText);
    let jsData = JSON.parse(this.responseText);
    for (let i = 0; i < jsData.length; i++) {
      let div = document.createElement("div");
      let repoName = document.createTextNode(jsData[i].full_name);
      div.appendChild(repoName);
      document.body.appendChild(div);
    }
  }
};
```

### الشرح

#### 1- إنشاء myRequest

```
`let myRequest = new XMLHttpRequest();`
```

هنا بنعمل كائن جديد من XMLHttpRequest عشان نقدر نعمل طلب للسيرفر

#### 2- فتح الطلب

```
`myRequest.open("GET","https://api.github.com/users/elzerowebsschool/repos",true);`
```

بهاي السطر بنفتح الطلب ونحدد إنه نوعه GET وبحدد الرابط اللي بدنا نجيب منه البيانات والـ true يعني الطلب غير متزامن

#### 3- إرسال الطلب

```
`myRequest.send();`
```

هنا بنرسل الطلب للسيرفر

#### 4- تغيير حالة الطلب

```
`myRequest.onreadystatechange = function () {`
```

هاي السطر بنحدد دالة تتنفذ كلما تتغير حالة الطلب

## 5-فحص حالة الطلب

```
`if (this.readyState === 4 && this.status === 200) {`
```

هنا بنفحص إذا اكتمل الطلب وإذا كان الطلب ناجح

## 6- طباعة الاستجابة

```
`console.log(this.responseText);`
```

إذا الطلب ناجح بنطبع البيانات التي جلبناها من السيرفر

## 7- تحويل البيانات من JSON

```
`let jsData = JSON.parse(this.responseText);`
```

هنا بنحول النص الذي جلبته الاستجابة من السيرفر إلى كائنات جافاسكريبت

## 8- حلقة للتكرار على البيانات

```
`for (let i = 0; i < jsData.length; i++) {`
```

هنا بنبدأ حلقة للتكرار على كل عنصر في البيانات المستلمة

## 9- إنشاء عنصر جديد

```
`let div = document.createElement("div");`
```

هنا بنعمل عنصر **div** جديد عشان نضيف فيه اسم المستودع

## 10- إضافة نص إلى العنصر

```
`let repoName = document.createTextNode(jsData[i].full_name);`
```

بنجيب اسم المستودع من البيانات ونضيفه كالنص للعنصر **div**

## 11- إضافة النص للـ div

```
`div.appendChild(repoName);`
```

هنا بنضيف النص إلى العنصر **div**

## 12- إضافة العنصر للصفحة

```
`document.body.appendChild(div);`
```

وأخيرا بنضيف العنصر **div** للصفحة الرئيسية



## 158- Callback Hell Or Pyramid OF Doom

Callback Hell أو Pyramid of Doom هي مشكلة بتصير لما تكتب كثير من الـ Callbacks داخل بعضهم في جافاسكريبت، وبيصير الكود شكله معقد وصعب تقرأه وتفهمه لأنه بيصير متداخل بشكل هرمي كبير.

الفكرة إنه كل ما بدك تنفذ عملية غير متزامنة (زي طلب بيانات من السيرفر) وبدك تنتظر النتيجة عشان تنفذ عملية ثانية، بتحتاج تستعمل Callback. ومع تكرار هاي العملية، بتلاقي حالك بتحط Callbacks داخل Callbacks ، وبتصير كل ما تنزل بالكود بتزيد التداخل، وهذا هو اللي بيسموه Callback Hell.

ليش بيصير هيك؟ لأنه في البرمجة غير المتزامنة بتحتاج دايماً تنتظر نتيجة من شيء قبل ما تعمل شيء ثاني. فإذا كان عندك مثلاً 3 أو 4 عمليات غير متزامنة بتعتمد على بعض، رح تحتاج تحط كل عملية جوه الثانية، وبتصير البنية هرمية.

### مثال

```
function makeTired(e) {
  e.target.style.color = "red";
let p = document.querySelector(".text");
p.addEventListener("click", makeTired);
function iamACallback() {
  console.log("Iam A CallBack Function");
setTimeout(iamACallback, 2000);
setTimeout(() => {
  console.log("download a Photo From URL");
  setTimeout(() => {
    console.log("resize Photo");
    setTimeout(() => {
      console.log("Add Logo To The Photo");
      setTimeout(() => {
        console.log("Show The Photo In Website");
      }, 1000);
    }, 2000);
  }, 2000);
}, 1000);
```

## 159-Promise Intro And Syntax

الـ Promise في جافاسكريبت هو عبارة عن آلية للتعامل مع العمليات غير المتزامنة. يساعدك على تنفيذ كود ينتظر نتيجة من عملية بتأخذ وقت، زي طلب بيانات من السيرفر أو قراءة ملف، بدون ما توقف تنفيذ باقي الكود.

الـ Promise بمر بثلاث حالات:

- معلق (Pending): في البداية لما تكون العملية لسا ما خلصت ولا تمت.
- ناجح (Fulfilled): لما تكون العملية تمت بنجاح وتم الحصول على النتيجة.
- مرفوض (Rejected): لما تفشل العملية وبيتم إرجاع خطأ.

في الـ Promise بنستخدم دالتين أساسيتين:

- resolve : بتناديها لما العملية تكون ناجحة وبتعطيها النتيجة.
- reject : بتناديها لما العملية تفشل وبتعطيها الخطأ.

عشان نتعامل مع الـ Promise لما يتم تنفيذه، بنستخدم دالتين إضافيتين:

- then(): بتشتغل لما الـ Promise ينجح وبتتعامل مع النتيجة.
- catch(): بتشتغل لما الـ Promise يفشل وبتتعامل مع الخطأ.

هاي هي الفكرة العامة، والـ Promise ببسهل كتابة كود غير متزامن بطريقة أسهل للفهم والتعامل

### مثال مع الشرح :-

```
const myPromise = new Promise((resolveFunction, rejectFunction) => {  
  let connect = false;  
  if (connect) {  
    resolveFunction("connection Established");  
  } else {  
    rejectFunction(Error("connection Failed"));  
  }  
}).then(  
  (resolveValue) => console.log(`Good ${resolveValue}`),
```

```
(rejectValue) => console.log(`Bad ${rejectValue}`)  
)  
console.log(myPromise)
```

هلا خليني أشرحك الكود خطوة بخطوة وبالتفصيل الممل

أول شي عندنا ثابت اسمه myPromise وهو معرف كـ Promise جديد  
الـ Promise بياخذ دالة كتستقبل دالتين وهم resolveFunction و rejectFunction  
هاي الدالة بتحدد شو بصير إذا العملية نجحت أو فشلت

داخل الـ Promise بنعرف متغير اسمه connect وهو معطى كـ false  
فهذا يعني إنه ما في اتصال ناجح

بعدها بنفحص إذا كان connect قيمته true أو false  
إذا كانت true بنادي الدالة resolveFunction وبمرر رسالة وهي connection Established  
أما إذا كانت false بنادي الدالة rejectFunction وبمرر Error اللي يحتوي على رسالة connection Failed

بعدين بعد ما ينتهي الـ Promise عندنا دالة then  
الـ then بتمرر دالتين وحدة لمعالجة النجاح ووحدة لمعالجة الفشل

الدالة الأولى بتتعامل مع النتيجة إذا كان الاتصال ناجح  
فبتأخذ القيمة اللي مررتها دالة resolveFunction اللي هي connection Established  
وبتطبعها مع كلمة Good يعني بيطلع Good connection Established

الدالة الثانية بتتعامل مع النتيجة إذا كان الاتصال فاشل  
فبتأخذ القيمة اللي مررتها دالة rejectFunction اللي هي الخطأ Error connection Failed  
وبتطبعها مع كلمة Bad يعني بيطلع Bad connection Failed

وأخيراً بنطبع myPromise في الكونسول  
بس لأنه الـ Promise بيتنفذ بشكل غير متزامن يعني بشكل موازي مش رح تشوف النتيجة الفعلية إلا بعد ما ينفذ then  
بما إن connect كان false فالنتيجة النهائية اللي رح تطلع في الكونسول هي Bad Error connection Failed

## 160-Promise ➔ Then & Catch & Finally

```
const myPromise = new Promise((resolveFunction, rejectFunction) => {  
  let employees = ["Suhaib", "Ahmad", "Sayed", "Mahmoud"];  
  if (employees.length === 4) {  
    resolveFunction(employees);  
  } else {  
    rejectFunction(Error("Number OF Employees Is Not 4")); }  
});
```

هون بننشئ Promise وبنتحقق من عدد العناصر بالمصفوفة إذا كان 4 بنمرر المصفوفة من خلال الـ resolve وإذا مش 4 بنمرر خطأ باستخدام reject.

```
myPromise  
  .then((resolveValue) => {  
    resolveValue.length = 2;  
    return resolveValue;  
  })  
  .then((resolveValue) => {  
    resolveValue.length = 1;  
    return resolveValue;  
  })  
  .then((resolveValue) => {  
    console.log(`The Chosen Employee Is ${resolveValue}`);  
  })  
  .catch((rejectedReason) => console.log(rejectedReason))  
  .finally(console.log("The Operation Is Done"));
```

هون بنستخدم then لتعديل طول المصفوفة بالتدريج وبالنهية بنطبع الموظف المختار. إذا صار خطأ بنطبع الخطأ بالـ catch وأخيراً بننفذ finally بعد ما تخلص العملية.

## الشرح :-

في الكود هاد عم نستخدم الـ **Promise** في جافا سكريبت عشان نتعامل مع العمليات اللي ممكن تكون **Asynchronous** يعني ما بتخلص بنفس اللحظة وبتحتاج وقت لحد ما تكتمل زي جلب بيانات من سيرفر

أول شي بنعمل **Promise** ودينا بدالتين **resolveFunction** و **rejectFunction** هاي الدالتين بتمثلوا الحالة اللي فيها الـ **Promise** بقدر يرجع نتيجة إما تكون ناجحة أو تفشل حسب الشرط اللي بتوضع

في المثال هون شرطنا بيقول إذا كان عدد الموظفين 4 بننفذ **resolveFunction** ونمرر المتغير **employees** اللي هو عبارة عن مصفوفة فيها أسماء الموظفين أما إذا كان عددهم مختلف عن 4 بننفذ **rejectFunction** ونرجع خطأ

بعدين بنستدعي الـ **Promise** باستخدام **then** إذا الشرط اتحقق وتم تنفيذ **resolveFunction** أول **then** بياخذ القيمة اللي رجعت اللي هي المصفوفة وبيغير عدد العناصر داخل المصفوفة إلى 2 وبيرجع القيمة اللي عدلناها عشان يتم استخدامها في **then** اللي بعده في الـ **then** الثانية بنعمل نفس الاشئ لكن بنغير عدد العناصر إلى 1 فقط ونمرر القيمة المعدلة لـ **then** الثالثة

في النهاية آخر **then** بيطبع الرسالة اللي بتقول إن الموظف المختار هو العنصر الأول من المصفوفة اللي صار فيها عنصر واحد بعد التعديل في حال فشلت العملية ووصلنا للـ **rejectFunction** بيتم تنفيذ الـ **catch** اللي بتطبع الخطأ اللي حصل

أما الـ **finally** فهي بتشتغل دايماً سواء كانت العملية نجحت أو فشلت وبتطبع إنه العملية انتهت

## 161-Promise And XHR

**Promise** هي ميزة حديثة بجافاسكربت بتسهل التعامل مع العمليات التي يتأخذ وقت زي جلب البيانات من الإنترنت هي طريقة للتعامل مع الأكواد التي بتشتغل بشكل غير متزامن يعني العملية ممكن تبلىش وهلا وتخلص بعدين بس هي بتعطينا طريقة نعرف إذا العملية نجحت ولا فشلت الـ **promise** بمر بمراحل هي **pending** يعني العملية قيد التنفيذ **fulfilled** إذا نجحت **rejected** إذا فشلت بنقدر نتعامل مع كل مرحلة باستخدام **then** و **catch**

أما **XHR** أو **XMLHttpRequest** هو أداة قديمة بجافاسكربت عشان تعمل طلبات للسيرفر مثلاً تجيب بيانات من موقع تاني بدون ما تحدث الصفحة بس التعامل معها شوي معقد لأنه بتعتمد على الأحداث **callbacks** يعني بتضطر تكتب أكواد إضافية عشان تتحقق إذا الطلب نجح أو فشل ومع تطور الجافاسكربت طلعوا طريقة أفضل للتعامل مع هيك حالات اللي هي الـ **promises** والتي بتسهل التعامل مع الطلبات

باختصار **Promise** أحدث وأسهل من **XHR**

### مثال :-

```
const getData = (apiLink) => {
  return new Promise((resolve, reject) => {
    let myRequest = new XMLHttpRequest();
    myRequest.onload = function () {
      if (this.readyState === 4 && this.status === 200) {
        resolve(JSON.parse(this.responseText));
      } else { reject(Error("No Data Found")); }
    };
    myRequest.open("GET", apiLink, true);
    myRequest.send();
  });
};

getData("https://api.github.com/users/elzerowebsschool/repos")
  .then((result) => {
    result.length = 10;
    return result;
  }).then((result) => console.log(result[0].name)).catch((rej) => console.log(rej));
```

هذا الكود يعمل وظيفة لجلب بيانات من رابط باستخدام الـ XMLHttpRequest مع استخدام الـ Promise لتسهيل التعامل مع النتيجة. خليني أشرحك إياه خطوة خطوة:

أول شيء بنعمل دالة اسمها `getData` هاي الدالة بتأخذ رابط الـ API وتعمل عليه طلب باستخدام `XMLHttpRequest` داخل الدالة، إحنا بنرجع `Promise` عشان نتعامل مع النتيجة بسهولة بعدين.

بعدين بننشئ طلب جديد باستخدام `new XMLHttpRequest` ، وهذا الطلب بيشتغل ليحلب البيانات من السيرفر.

بنستخدم `onload` لتحديد شو يصير لما الطلب ينتهي. إذا كانت حالة الطلب `readyState` هي أربعة، يعني الطلب اكتمل، وإذا كانت حالة الاستجابة `status` هي ميتين، يعني الطلب ناجح، بنستخدم `resolve` لتحويل النص اللي رجعه السيرفر لـ `JSON` باستخدام `JSON.parse`.

إذا ما كانت الحالة ناجحة، بنستدعي `reject` وبنرسل رسالة خطأ.

بعدين بفتح الطلب باستخدام `myRequest.open` ونمرر رابط الـ API ونستخدم `myRequest.send` لإرسال الطلب.

لما نستدعي الدالة `getData` ونمرر إليها رابط الـ API ، بنستخدم `then` عشان نتعامل مع النتيجة إذا كانت ناجحة. بنقل طول النتيجة لعشر عناصر باستخدام `result.length = 10` وبعدين بنرجعها مرة ثانية. في `then` الثاني، بنعرض اسم أول عنصر من البيانات اللي حصلنا عليها.

إذا فشلت العملية لأي سبب، بنستدعي `catch` وبنطبع رسالة الخطأ.

## 162-Fetch API

**Fetch API** بجافا سكريبت هي طريقة جديدة وسهلة لجلب بيانات من السيرفرات زي ما كانت الطريقة القديمة XMLHttpRequest. الفرق إنه مع **fetch** التعامل أسهل وأوضح

بتستدعي **fetch** وبتعطيها رابط السيرفر أو الـ **API** اللي بدك تجيب منه البيانات. أول إشي **fetch** بتعمل طلب للسيرفر، وبرجعك **Promise** ، يعني ممكن تتعامل معها بطريقة سهلة باستخدام **then** و **catch**. بأبسط صورة، لما يخلص الطلب بتستخدم **then** عشان نحلل النتيجة أو نتصرف حسب المطلوب وإذا صار في خطأ بنمسك الخطأ باستخدام **catch**

الفرق الرئيسي إنه **fetch** بتستخدم **Promise** بشكل مباشر وبتخليك تتعامل مع الطلبات بشكل أنظف وأوضح من الطريقة القديمة

```
fetch("https://api.github.com/users/elzerowebsschool/repos").then((result) => {  
  let myData = result.json();  
  return myData  
}).then((myData)=>{  
  myData.length = 10  
  return myData  
}).then((myData)=>{  
  console.log(myData[0].name)  
})
```

هون الكود أول إشي بنادي على **fetch** عشان يعمل طلب على رابط الـ **API** اللي بيعطي معلومات عن مستودعات المستخدم من **GitHub**. أول خطوة بنستنى الرد باستخدام **then** أول ما يوصل الرد بنتأكد منه و بنحوله لبيانات باستخدام **json()** لأن الرد بيكون بصيغة **JSON**. بعدين في **then** الثاني بنحدد طول البيانات اللي جبتها لـ ١٠ بس. بعدين بنرجعها. في **then** الأخير بنطبع اسم أول مستودع موجود بالبيانات اللي رجعناها



### 163- Promise All & Settled & Race

```
const myFirstPromise = new Promise((res, rej) => {  
  setTimeout(() => {  
    rej("IAM THE FIRST PROMISE");  
  }, 3000);  
});
```

```
const mySecondPromise = new Promise((res, rej) => {  
  setTimeout(() => {  
    res("IAM THE SECOND PROMISE");  
  }, 1000);  
});
```

```
const myThirdPromise = new Promise((res, rej) => {  
  setTimeout(() => {  
    res("IAM THE THIRD PROMISE");  
  }, 2000);  
});
```

```
Promise.all([myFirstPromise, mySecondPromise, myThirdPromise]).then(  
  (resolvedValues) => console.log(resolvedValues),  
  (rejectedValue) => console.log(`Rejected ${rejectedValue}`)  
);
```

```
Promise.allSettled([myFirstPromise, mySecondPromise, myThirdPromise]).then(  
  (resolvedValues) => console.log(resolvedValues),  
  (rejectedValue) => console.log(`Rejected ${rejectedValue}`)  
);
```

```

Promise.race([myFirstPromise, mySecondPromise, myThirdPromise]).then(
(resolvedValues) => console.log(resolvedValues),
(rejectedValue) => console.log(`Rejected ${rejectedValue}`)
);

```

أول اشي الكود فيه ثلاث promises كل واحد فيهم بيعمل عملية معينة بعد مدة زمنية معينة عندك أول promise بيعمل reject بعد ثلاث ثواني وبيطبع IAM THE FIRST PROMISE والثاني بيعمل resolve بعد ثانية وبيطبع IAM THE SECOND والثالث بيعمل resolve بعد ثانيتين وبيطبع IAM THE THIRD PROMISE بالنسبة لأول جزء من الكود Promise.all بياخذ كل ال promises اللي موجودين وبيستنى كلهم يخلصوا إذا واحد منهم فشل بعمل reject وبيطبع اللي فشل فيه مثل Rejected IAM THE FIRST PROMISE بالنسبة ل Promise.allSettled هو ببيستنى كل ال promises يخلصوا بغض النظر إذا نجحوا أو فشلوا وبعدين بيعرض النتيجة لكل واحد فيهم على شكل status سواء كان fulfilled أو rejected أما Promise.race فبيطبع أول promise بيخلص سواء كان resolve أو reject يعني هون رح يطبع IAM THE SECOND PROMISE لأنه الثاني بيخلص أسرع من الباقيين

#### 164-Async

Async في البرمجة هو نمط بنستخدمه للتعامل مع العمليات اللي بتأخذ وقت طويل مثل طلبات الشبكة أو استرجاع بيانات من قاعدة البيانات الفكرة من async إنه يخلي البرنامج يكمل شغله بدون ما يستنى هذي العمليات تنتهي وبدل ما توقف الكود لغاية ما ترجع البيانات، بتستخدم ال async عشان تكمل باقي الكود ولما تخلص العملية بتقدر تشوف النتيجة لما تيجي تستخدم async في جافا سكريبت بتعمل دالة باستخدام async واللي بتسهل عليك التعامل مع الكود بدل ال promises بتستنى النتيجة باستخدام كلمة await فمثلاً لما بدك تجيب بيانات من API أو تعمل عملية تحميل بتستخدم await عشان تستنى العملية تنتهي لكن بدون ما توقف باقي الكود

```

async function getData() {
let users = ["Suhaib"];
if (users.length > 0) {
return "Users Found" ;
} else {
throw new Error("No Users Found");
}
}
getData().then( (resolvedValue) => console.log(resolvedValue),
(rejectedValue) => console.log(rejectedValue));

```

أول شي عندك دالة `async` اسمها `getData` فيها مصفوفة فيها اسم مستخدم واحد اسمه `Suhaib`. إذا كان في مستخدمين داخل المصفوفة يعني طولها أكبر من صفر، راح ترجع الكلمة `Users Found`. أما إذا المصفوفة كانت فاضية، راح ترمي `Error` بتقول `"No Users Found"`.

لما تستدعي `getData`، بتستخدم `then` عشان تتابع النتيجة. إذا الدالة رجعت قيمة، راح تظهر في `resolvedValue` وتطبعها، وإذا صار خطأ راح تظهر الرسالة في `rejectedValue` وتطبعها.

ببساطة، الفكرة من `async/await` إنها تسهل التعامل مع البيانات بدون ما تدخل في تعقيدات `promises`

---

## 165-Await

هي كلمة محجوزة في جافا سكريبت تستخدم مع `async functions` لتحسين التعامل مع العمليات غير المتزامنة لما تستخدم `await` قبل وظيفة ترجع `promise` رح توقف التنفيذ لحد ما تخلص العملية وترجع النتيجة ببساطة لما تنادي على دالة تحتوي `await` رح تنتظر لحد ما تخلص العملية وترجع النتيجة تأكد انك تستخدم `await` بس داخل `async function`

```
const myPromise = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("Iam The Good Promise");
  }, 3000);
});
```

```
async function readData() {
  console.log("Before Promise");
  console.log(await myPromise);
  console.log("After Promise");
}

readData();
```

أنت عرفت `promise` اسمها `myPromise` اللي بتخلص بعد 3 ثواني وبترد النص `"Iam The Good Promise"`

ثم عرفت دالة `async` اسمها `readData` بداخلها بتطبع `"Before Promise"`

بعدها بتستخدم `await` مع `myPromise` يعني رح تنتظر لحد ما `promise` تخلص وبعدين تطبع النتيجة بعدين بتطبع `"After Promise"` لما تنادي على `readData` رح يكون الناتج كالتالي

`"Before Promise"`

بعدين بعد 3 ثواني `"Iam The Good Promise"`

وأخيراً `"After Promise"`

هيك بتكون فهمت كيف تعمل `await` مع `promises`

## 166- Async ➔ Await With Try , catch Finally

async await مع try catch finally هي طريقة قوية للتعامل مع الأخطاء في البرمجة غير المتزامنة

1. try : يتخطى فيه الكود الذي يمكن أن يتسبب خطأ

2. catch : يتعامل مع الأخطاء إذا صار في مشكلة أثناء تنفيذ الكود في try

3. finally : الكود الذي رح ينفذ دائماً سواء صار خطأ أو لا

بهيك طريقة تقدر تضمن إنك تتعامل مع الأخطاء بشكل مناسب وتنفذ الكود الذي بدك إياه بعد العملية

إذا نجح كل شيء رح تكمل بشكل طبيعي

إذا صار خطأ رح يطبع لك رسالة توضح السبب

وفي النهاية دائماً رح تتأكد إنه فيه كود ينفذ بعد كل العمليات

```
async function fetchData() {  
  console.log("Before fetch");  
  try {  
    let myDate = await fetch( "https://api.github.com/users/elzerowebsschool/repos" );  
    console.log(await myDate.json());  
  } catch (reason) {  
    console.log(`reason ${reason}`);  
  } finally {  
    console.log("After fetch"); }  
}
```

fetchData();

أنت عرفت دالة async اسمها fetchData بداخلها بتطبع "Before fetch" بعدين بتستخدم await مع fetch عشان تجيب البيانات من API خاص بجيت هاب إذا العملية نجحت رح تتبع النتيجة باستخدام await myDate.json التي بتحول البيانات ل JSON إذا صار أي خطأ خلال عملية الجلب رح يدخل على catch ويطبع السبب بعدين في finally بتطبع "After fetch"

لما تنادي على fetchData رح يكون الناتج كالتالي

"Before fetch"

بعدين بعد ما تخلص العملية رح تطبع البيانات

وأخيراً "After fetch"

بهالطريقة بتكون تقدر تتعامل مع البيانات بشكل مرتب

# Index

Address	Page number
---------	-------------

<b>BASICS JAVASCRIPT</b>	3
19-Exame:-	8
22-Exame	10
26-Exame:-	14
28- Logical Operators	15
29- Control Flow ( If & else if & else)	16
30- IF inside IF	17
31- Forms IF	17
32- null & undefined & Number	19
33-Exame:-	19
34- switch	21
Exame:-	21
Exame 2:-	23
36- Array:-	25
37- Arrays Methods	26
38-Arrays Methods Adding And Removing	27
39-Arrays Methods Search	27
40- Arrays Merthods sort	28
41-Arrays Methods Slicing	29
42 – Arrays Methods	29
43-Exame:-	30
46-Products Practice	32
47-Loop While	33
48- Loop Do / while	33
49-Loop Challenge	34
50-Function	35
51-Example function	36
52-Return Function	36

<b>53-Default Function Parameters</b>	<b>37</b>
<b>54-Rest Parameters</b>	<b>37</b>
<b>55-Function Advanced Practice</b>	<b>38</b>
<b>Exame:-</b>	<b>39</b>
<b>56-Anonymous Function</b>	<b>41</b>
<b>57-Function Inside Function</b>	<b>42</b>
<b>58-Arrow Function</b>	<b>43</b>
<b>58-Scope</b>	<b>44</b>
<b>59-Block Scope</b>	<b>44</b>
<b>Exame:-</b>	<b>45</b>
<b>61-Higher Order Functions</b>	<b>47</b>
<b>62- Map -Swap Cases</b>	<b>49</b>
<b>63-Filter Function Map</b>	<b>50</b>
<b>64-filter and map</b>	<b>50</b>
<b>65-Reduce</b>	<b>51</b>
<b>66- Reduce 2</b>	<b>52</b>
<b>67- ForEach</b>	<b>52</b>
<b>Exame:-</b>	<b>53</b>
<b>68-Object</b>	<b>55</b>
<b>69-Object – Bot / Bracket Notation</b>	<b>55</b>
<b>70- Object – Nested Object And Trainings</b>	<b>55</b>
<b>71-Object – Create With New Keyword New Object</b>	<b>56</b>
<b>72-Object Function This keyword</b>	<b>57</b>
<b>73-Object – Create Object With Create Method</b>	<b>58</b>
<b>74-Object – Create Object With Assign Method</b>	<b>59</b>
<b>75-What Is DOM</b>	<b>61</b>
<b>76-DOM Get Set Elements Content And Attributes → innerText</b>	<b>62</b>
<b>77-DOM → Check Attributes</b>	<b>63</b>
<b>78-DOM → Create Elements</b>	<b>63</b>
<b>79- DOM → Create Elements → Practice Product With Heading And Paragraph</b>	<b>64</b>
<b>80- DOM → Deal With Childrens</b>	<b>65</b>
<b>81-DOM → Events</b>	<b>66</b>
<b>82-DOM Events</b>	<b>67</b>

83- DOM → Events Simualtion	69
84-DOM → Class List	70
85- DOM → CSS	70
86-DOM → Deal With Elements → (before – After – append – prepend – remove )	71
87-DOM → Traversing	71
88-DOM → Cloning (CloneNode (Deep));	72
89-DOM → Add Event Listener	72
Exame:-	73
90-BOM →Browser Object Model	78
91-BOM → Alert --- Confirm --- Prompt	78
92-BOM → SetTimeOut , clearTimeout	79
93-BOM → setInterval , clearInterval	79
94-BOM → Location Object (href Get / Set ..... / host / hash / protocol / reload() / Replace() / assign() )	80
95-BOM →Open Window	80
96-BOM →History API	80
97-BOM → Stop , print , focus , scroll	81
99-BOM → Local Storage	82
100-BOM → Session Stoarge	83
Exame:-	84
101- Destructuring Array	86
102 -Destructuring Array → Advanced Examples	87
103-Destructuring → Swapping Variables	88
104-Destructuring → Object	88
105-Destructuring → (Naming The Variables , AddNew , Nested )	89
106-Destructuring → Destructuring Function Parameters	89
107- تطبيق كامل	90
Exame:-	90
108-Set Data → Type Methods	92
109-Set VS WeakSet	93
110-Map Data Type Vs Object	94
111-Map Methods	95
112-Map Vs WeakMap	95

113-Array Methods ➔ Array.from -----	96
114-Array Methods ➔ Array.Copy -----	97
115-Array ➔ Array.Some -----	98
116-Array ➔ Array.every -----	99
117-Array ➔ Spread Operator ➔ Iterable -----	100
Exame :- Map And Set Challenge ➔ Video (133) -----	101
118- Regular Expression -----	102
119-Regular Expression ➔ Syntax -----	103
120-Regular Expression ➔ Ranges ➔ Part One -----	103
121-Regular Expression ➔ Ranges ➔ Part Tow -----	105
122- Regular Expression ➔ Character Classes ➔Part One -----	106
123- Regular Expression ➔ Character Classes ➔Part Tow -----	106
124- Regular Expression ➔ Quantifiers ➔ Part One -----	107
125- Regular Expression ➔ Quantifiers ➔ Part Tow -----	108
126- Regular Expression ➔ Quantifiers ➔ Part Three -----	108
127- Regular Expression ➔ Replace / ReplaceAll -----	109
128- Regular Expression ➔Input Form Validation Practice -----	109
129-Exame :-Regular Expression Challenge ➔ video 146 -----	111
130-Constructor Function -----	114
131- Constructor Function ➔ New Syntax -----	114
132-- Constructor Function ➔ Deal With Properties And Methods -----	115
133- Constructor Function ➔Update Properties Built In Constructors -----	116
134-- Constructor Function ➔ Static Properties And Methods -----	117
135 - Class Inheritance -----	118
135 - Class Encapsulation -----	119
136-Prototype ➔ Add To Prototype Chain & Extend -----	120
137-Object Meta Data And Descriptor -----	121
138-Object Meta Data And Descriptor ➔ Part One ➔Writable & Enumerable & Configurable -----	123
139-Object Meta Data And Descriptor ➔ Part Tow ➔ Define Multiple Properties & Check Descriptors -----	124
140-Date And Time -----	125
141-Date And Time ➔GetTime & GetDate & getFullYear & GetMonth & getDay & getHours&..... -	126
142-Date And Time -----	126



143-Date And Time → New Date(timestamp & Date String & Numeric Values)	127
144-Date And Time → Track Operations Time	127
145-Generators	128
146- Generators→delegate Generator	131
147-Generators →Infinite Numbers & User Return Inside Generators	132
148-Modules → Import And Export → Namer VS Default Import And Export All	133
149-What is JSON	137
150-JSON → Syntax	137
151-JSON → API Overview	137
152- JSON → JOSN.parse & stringify	138
153-JSON → Asynchronous & Synchronous	139
154 - JSON → Call Stack And Web API	139
154 - JSON → Event Loop And CallBack Queue	141
155 - AJAX	142
156-AJAX → Request And Response From Real API	142
157- Loop On Data	143
158- CallBack Hell Or Pyramid OF Doom	145
159-Promise Intro And Syntax	146
160-Promise → Then & Catch & Finally	148
161-Promise And XHR	150
162-Fetch API	152
163- Promise All & Settled & Race	153
164-Async	154
165-Await	155
166- Async → Await With Try , catch Finally	156

# JAVASCRIPT IS DONE