Java Script
R3 CODE

كتب بواسطة: عبد الرحمن عرفات

Written by: Abdul Rahman Arafat

لغة جافا سكريبت تعد اشهر لغة في برمجة الويب سواء Back End او Front End ولها العديد من ال Frameworks القوية في كلا المجالين يمكنك تعلمها وقتما تشاء ولكن اذا كنت هنا لتصبح Web Developer فيفضل أن يكون لديك خلفية في كلاً من HTML و CSS لتكون على الطريق الصحيح للإحتراف تطوير الويب

اهلاً بيك معايا في الرحلة ويلا بينا يا بطل نكتب سطر جديد في كتب التاريخ.

امر الطباعة في الكونسول في javascript هو

```
console.log()
```

تحويل انواع البيانات بين البيانات

التحويل الى نص :

```
toString()
```

تعريف المتغيرات

ممكن نستعمل اكتر من كلمة في جافا سكريبت

var
const
let

```
let name = "abdo";
```

Math Object

round

تقوم بالتقريب حسب الكسر تحت ام فوق 0.5 فإذا كان 0.5 او اعلي فإنها تقربة للعد الصحيح الأعلي والعكس صحيح

```
console.log(Math.round(3.6));
// 4
console.log(Math.round(3.4));
// 3
```

ceil

تقوم بالتقريب للعدد الأكبر قيمة

```
console.log(Math.ceil(6.3));
// 7
```

floor

تقوم بالتقريب للعدد الأقل قيمة

```
console.log(Math.floor(6.9));
// 6
```

min

تقوم باختيار العدد الأقل قيمة من بين عدة بدائل

max

تقوم باختيار العدد الأعلى قيمة من بين عدة بدائل

```
console.log(Math.min(6, 9, 6, 5));
// 5
console.log(Math.max(6, 9, 6, 5));
// 9
```

pow

تقوم بحساب الأس لرقم فهي تأخذ رقمين الأول هو الأساس والثاني هو الأس

```
console.log(Math.pow(2, 4));
// 16
```

random

تقوم باختيار رقم عشوائي

```
console.log(Math.random());
// random number
```

trunc

تقوم بحذف الكسر

```
console.log(Math.trunc(2.6));
// random number
```

string methods

البحث بال index ال index معروفة يبدأ من 0

يمكننا استخدام اما

charAt()

```
let TheName = "abdo";
console.log(TheName[2])
console.log(TheName.charAt(2))
// d
// d
```

length تقوم بحساب عدد عناصر السلسلة

```
let TheName = "abdo";
console.log(TheName.length)
// 4
```

trim تقوم بإزالة المسافات من بداية ونهاية المقطع

```
let TheName = "   abdul rahman  ";
console.log(TheName.trim())
// abdul rahman
```

toUpperCase

تقوم بتحويل النص الى كابيتال

toLowerCase

تقوم بتحويل النص الى سمول

```
let TheName = "   abdul rahman  ";
console.log(TheName.toLowerCase());
console.log(TheName.toUpperCase());
// ABDUL RAHMAN
// abdul rahman
```

indexOf

تقوم بالبحث عن اول ظهور لمقطع معين من بداية المصفوفة

lastIndexOf

تقوم بالبحث عن اول ظهور لمقطع معين من نهاية المصفوفة

slice

تقوم بأخذ مقطع من النص باستخدام الفهرس

repeat

تقوم بعمل تكرار للسلة النصية وتاخذ عدد مرات التكرار

split

تقوم بقص النص بناء علي حزف مقطع معين وانشاء مصفوفة من باقي السلاسل النصية

```
let TheName = "abdul rahman";
console.log(TheName.indexOf("a"));
// 0
console.log(TheName.lastIndexOf("a"));
// 2
console.log(TheName.slice("rahman"));
// 7
console.log(TheName.repeat(2));
// abdul rahman abdul rahman
console.log(TheName.split(" "));
// ['abdul', 'rahman']
```

supString

تقوم بأخذ مقطع من النص باستخدام اللفهرس

```
let TheName = "abdul rahman";
console.log(TheName.substring(3, 5));
// ul
```

Included

لتحقق اذا كانت القيمة المعطاة موجودة ام لا في النص المكتوب

StartWith

التحقق اذا كانت القيمة المعطاة موجودة ام لا في النص المكتوب وتحسب من البداية

EndWith

التحقق اذا كانت القيمة المعطاة موجودة ام لا في النص المكتوب وتحسب من النهاية

```
// includes()
let text1 = "Welcome to ABN's website";
console.log(text1.includes("ABN"));        // true ✅
لأن الكلمة موجودة داخل النص
console.log(text1.includes("website"));    // true ✅
console.log(text1.includes("hello"));      // false ❌
مش موجودة

// startsWith()
let text2 = "JavaScript is powerful";
console.log(text2.startsWith("Java"));     // true ✅
يبدأ بـ "Java"
console.log(text2.startsWith("Script"));   // false ❌
مش في البداية
console.log(text2.startsWith("java"));     // false ❌
حساس لحالة الحروف (case-sensitive)

// endsWith()
let text3 = "Learning with ABN";
console.log(text3.endsWith("ABN"));        // true ✅
بينتهي بـ "ABN"
console.log(text3.endsWith("with"));       // false ❌
مش في النهاية
console.log(text3.endsWith("n"));          // true ✅
آخر حرف هو "n"
```

comparison operators

```
console.log(10 == "10"); // Compare Value Only
console.log(-100 == "-100"); // Compare Value Only
console.log(10 != "10"); // Compare Value Only

console.log(10 === "10"); // Compare Value + Type
console.log(10 !== "10"); // Compare Value + Type
console.log(10 !== 10); // Compare Value + Type

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

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

console.log(typeof "Osama" === typeof "Ahmed");
```

Logical operators

```
!     NOT
&&    AND
||    OR
```

Conditional Statments

الجمل الشرطية هي واحدة من الأمور التي كثيراً ما ستستخدمها

if => لو
else if => وإلا لو
else => بخلاف ذلك

if

تأخذ الشرط والكود الذي سيتم تنفيذة اذا تحقق هذا الشرط

else if

تأخذ الشرط والكود الذي سيتم تنفيذة اذا تحقق هذا الشرط
وتستخدم في حالة تواجد عدة شروط ويمكنك استخدام العدد الذي نريدة منها

else

لا تأخذ شرط كغيرها فهي توضع في نهاية الشروط التي تضعها
وتأخذ الكود الذي سيتم تنفيذة في حالة عدم تحقق اي شرط

```
var age = 21;
if (age <= 17  ){
    console.log("you can't use our website")
}
else if (age >= 18  ){
    console.log("you can use our website")
}
else {
    console.log("you must enter your age")
}
```

Nested If Condition

هي ما يمكننا من عمل شروط متداخلة للتحق من توافر عدة شروط في عنصر معين كالمثال التالي

```
var age = 21;
var contury = "egypt";
var female = true
if (age <= 17) {
    if (contury != "egypt") {
        if (female != true)
            console.log("you cn't use this website")
    }
```

switch case

```
let day = "Friday";
switch (day) {
  case "Monday":
    console.log("بداية الأسبوع 💼");
    break;
  case "Friday":
    console.log("نهاية الأسبوع 🎉");
    break;
  case "Saturday":
    console.log("راحة واستجمام 😌");
    break;
  default:
    console.log("يوم عادي");
}
```

Array

المصفوفة وهي قائمة تحتوي على العناصر وتشبه تماماً ال list في بايثون

فيما يلي مثال يوضح كيفية انشاء مصفوفة والبحث داخلها

```
let arraysub = ["languages","programing"]
// انشاء مصفوفة
let myArray =
["abdulrahman","saeed","omar",["adam","mohamed"],["ganna"]];
// انشاء مصفوفة متداخلة
console.log(arraysub)
// ستطبع المصفوفة الأولي كاملةً
console.log(myArray[0])
// ستطبع العنصر الاول في المصفوفة
console.log(myArray[3][0])
// ستطبع العنصر الاول في المصفوفة المتداخلة الاولي
console.log(myArray[4][0][0])
//  0 ستطبع من القائمة المتداخلة الثانية من العنصر صاحب الفهرس
// 0ستطبع من السلسلة النصية الخاصة بة الحرف صاحب الفهرس
```

هناك بعض الدوال للتعامل مع المصفوفات

منها **length** تقوم بحساب طول المصفوفة

```
let arraysub = ["languages", "programing"]
console.log(arraysub.length); //2
```

منها **unshift** تقوم بإضافة العناصر لبداية المصفوفة

```
let arraysub = ["languages", "programing"]
arraysub.unshift("biology", "math");
console.log(arraysub);
// ['biology', 'math', 'languages', 'programing']
```

push تقوم بإضافة العناصر لنهاية المصفوفة

```
let arraysub = ["languages", "programing"]
arraysub.push("piology", "math");
console.log(arraysub);
// [ 'languages', 'programing','biology', 'math']
```

shift تقوم بحذف أول عنصر

pop تقوم بحذف آخر عنصر

```
let arraysub = ["languages", "programing","math"]
arraysub.shift();
arraysub.pop();
console.log(arraysub);
// ['programing']
```

indexOf

تقوم بالبحث عن اول ظهور للعنصر من بداية القائمة وتعطيك رقم الفهرس الموجود بة

ويمكنك اعطائها رقم فهرس تبدء البحث من عندة

lastIndexOf

تقوم بالبحث عن اول ظهور للعنصر من نهاية القائمة القائمة وتعطيك رقم الفهرس الموجود بة

ويمكنك اعطائها رقم فهرس **بالسالب** تبدء البحث من عندة

```
let arraysub = ["languages", "programing", "math", "languages", "biology"]
console.log(arraysub.indexOf("languages"));
// 0
console.log(arraysub.lastIndexOf("languages"));
// 3
```

sort تقوم بترتيب المصفوفة ابجدياً

reverse تقوم بعكس المصفوفة

```
let arraysub = ["languages", "programing", "math", "languages",
"biology"]
console.log(arraysub);
console.log(arraysub.sort());
console.log(arraysub.reverse());
// ["programing","biology","math", "languages", "languages"]
// ['languages', 'languages', 'math', 'biology', 'programing']
// ['programing', 'biology', 'math', 'languages', 'languages']
```

slice

تقوم بأخذ نسخة من القائمة

اذا اعطيناها رقم فهرس فهي تبدء قطع النسخة من عندة

واذا اعطيناها رقمين فهي تأخذ نسخة بداية من الفهرس للرقم الاول حتي فهرس الرقم الثاني ولكن العنصر في فهرس الرقم الثاني لا يكون ضمنها

splice

splice(index to start,how many items you want to remove,items you want add)

```
let arraysub = ["languages", "programing", "math", "languages", "piology"]
arraysub.splice(0, 0, "adam")
console.log(arraysub);
```

هنتعلمها قدام شوية join

Array.from():

تحويل سلسلة نصية إلى مصفوفة

```
Array.from("hello"); // ["h", "e", "l", "l", "o"]
```

Sets: يتم تحويل العناصر إلى مصفوفة

```
Array.from(new Set([1, 2, 3])); // [1, 2, 3]
```

يتم تحويل الأزواج إلى مصفوفة Maps

```
Array.from(new Map([[1, 'a'], [2, 'b']])); // [[1, "a"], [2, "b"]]
```

كائنات شبيهة بالمصفوفة مثل:

```
const obj = {0: 'a', 1: 'b', length: 2};
Array.from(obj); // ["a", "b"]
```

array.some()

هي دالة تُستخدم للتحقق مما إذا كان هناك عنصر واحد على الأقل في المصفوفة يحقق شرطًا معينًا. إذا تحقق الشرط لأي عنصر، ترجع `true`. وإذا لم يتحقق لأي عنصر، ترجع `false`.

الصيغة العامة:

```
array.some(callback(element, index, array), thisArg)
```

- `callback`: دالة يتم تنفيذها على كل عنصر.
- `element`: العنصر الحالي.
- `index`: فهرس العنصر (اختياري).
- `array`: المصفوفة الأصلية (اختياري).
- `thisArg`: قيمة اختيارية لتحديد السياق داخل الـ callback.

مثال عملي:

```
const numbers = [1, 3, 5, 8, 9];
const hasEven = numbers.some(function(num) {
  return num % 2 === 0;
});
console.log(hasEven); // true لأن الرقم 8 زوجي
```

في هذا المثال، الدالة some ( ) تبحث عن رقم زوجي واحد على الأقل. بمجرد أن تجد الرقم 8، ترجع `true` وتتوقف عن البحث.

🛒 مثال آخر: التحقق من وجود منتج غير متوفر

```
const products = [
  { name: 'Laptop', inStock: true },
  { name: 'Mouse', inStock: true },
  { name: 'Keyboard', inStock: false }
];

const hasOutOfStock = products.some(product => !product.inStock);

console.log(hasOutOfStock);
```

array.every()

هي نفس ال some لكن ال some كانت بتتحقق اذا فيه شرط متوفر على عنصر واحد فقط هنا لازم الشرط يكون متوفر في كل العناصر

Spread Syntax And Use Cases

تقوم بتفكيك العناصر ليصح كل عنصر منفرداً

```
const name = "abdo";
myArray = [...name];
console.log(myArray);
// Expected output: ['a', 'b', 'd', 'o']
```

Set

تستخدم لتخزين البيانات ولكنها لا تسمح بتكرار البيانات ولا تتعامل مع بالفهرس (index)

```
// إنشاء Set جديد
const mySet = new Set();

// إضافة عناصر
mySet.add(1);
mySet.add(2);
mySet.add(2); // لن تُضاف لأن 2 موجودة بالفعل

// التحقق من وجود عنصر
console.log(mySet.has(1)); // true

// حذف عنصر
mySet.delete(1);

// عدد العناصر
console.log(mySet.size); // 1

// المرور على العناصر
mySet.forEach(value => {
  console.log(value);
});

// حذف كل العناصر
mySet.clear();
```

Weak Set

تخزن الكائنات

Map

الماب في بايثون هي ال Dictionary وتستخدم لتخزين قيم لكل قيمة مفتاح

```
 // map إنشاء
const m = new Map();
// set إسناد القيم باستخدام
m.set("book", "video");
m.set("video", "book");
// get الوصول إلى القيم باستخدام
console.log(m.get("book"));
console.log(m.get("video"));
// ----------------------------------------
// map إنشاء
const m2 = new Map([
    // اسناد القيم مباشرة
    // key, value
    [1, "book one"],
    [2, "book two"],
    [3, "book three"]
]);
// get الوصول إلى القيم باستخدام
console.log(m2.get(1));
console.log(m2.get(2));
console.log(m2.get(3));
// حذف عنصر باستخدام المفتاح
m2.delete(2);
// طباعة المحتوى
console.log(m2);
// حذف جميع العناصر
m.clear();
console.log(m);
//التحقق من وجود مفتاح معين
console.log(m2.has(1));
console.log(m2.has(2));
console.log(m2.has(3));
// true true true
```

Weak Map

تخزن الكائنات

loop

for loop
while loop
Do while loop
GENERATORS 🔎

تستخدم الحلقات لعمل تكرار بدلا من كتابة نفس الكود عدة مرات
فهي تقوم بتكرارها

```
for (the index to start; the condition to stop;
Amount of increase per cycle) {
 block of code
}
```

1- for loop

تُستخدم عندما تعرف عدد التكرارات مسبقًا.

```
for (let i = 0; i < 5; i++) {
  console.log(i);
}
```

- التهيئة: `let i = 0` تبدأ المتغير.
- الشرط: `i < 5` طالما الشرط صحيح، تستمر الحلقة.
- الزيادة: `i++` تزيد قيمة `i` بعد كل تكرار.
- النتيجة: تطبع الأرقام من 0 إلى 4.

2-while loop

تُستخدم عندما لا تعرف عدد التكرارات مسبقًا، وتريد التكرار طالما الشرط محقق.

```
let i = 0;
while (i < 5) {
  console.log(i);
  i++;
}
```

- تتحقق من الشرط أولًا.
- إذا كان الشرط صحيحًا، تنفذ الكود داخل الحلقة.
- تستمر حتى يصبح الشرط خاطئًا.

3-Do while loop

تُستخدم عندما تريد تنفيذ الكود مرة واحدة على الأقل، حتى لو كان الشرط خاطئًا.

```
let i = 0;
do {
  console.log(i);
  i++;
} while (i < 5);
```

- تنفذ الكود أولًا.
- ثم تتحقق من الشرط.
- تستمر طالما الشرط صحيح.

⚠️ 1. التحكم في الشرط لتجنب الحلقات اللانهائية

إذا نسيت تحديث المتغير داخل الحلقة أو كتبت شرط خاطئ، ممكن الحلقة تشتغل إلى ما لا نهاية وتوقف البرنامج.

```
let i = 0;
while (i < 5) {
  console.log(i);
  // هنا، الحلقة مش هتوقف لو نسيت i++
}
```

🔄 2. استخدام break و continue

- break: يوقف الحلقة تمامًا.
- continue: يتخطى التكرار الحالي ويبدأ التكرار التالي.

```
for (let i = 0; i < 5; i++) {
  if (i === 3) break;       // يوقف عند 3
  console.log(i);
}

for (let i = 0; i < 5; i++) {
  if (i === 3) continue;    // يتخطى 3
  console.log(i);
}
```

🧠 3. اختيار نوع الحلقة المناسب

- `for`: لما تعرف عدد التكرارات.
- `while`: لما الشرط يتغير أثناء التنفيذ.
- `do...while`: لما لازم تنفذ الكود مرة واحدة على الأقل.
- `for...in`: للمرور على خصائص الكائنات.
- `for...of`: للمرور على عناصر المصفوفات أو الـ iterable.

```
const person = {name: "Ali", age: 30};
for (let key in person) {
  console.log(key, person[key]); // name Ali, age 30
}

const colors = ["red", "green", "blue"];
for (let color of colors) {
  console.log(color); // red, green, blue
}
```

🧹 4. تحسين الأداء

- الحلقات داخل الحلقات (nested loops) ممكن تبطئ الكود.
- حاول تقلل العمليات داخل الحلقة قدر الإمكان

```
for (let i = 0; i < bigArray.length; i++) {
  // تجنب العمليات الثقيلة هنا
}
```

Function

1-built in function

وهي مجموعة من الدوال الجاهزة المدمجة مثل

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

2-user function

وهذة هي موضوعنا الذي سنتكلم عنة

“user function”

. كيف ننشئ دالة(function) ؟ .

1-نكتب الكلمة المدمجة function ثم بعدها نسميها اسم يدل علي وظيفتها ثم نفتح قوسين ثم نفتح قوسين مجموعة للكود الذي سيتم تنفيذه

```
function function_name(){
block of code
}
```

2-نقوم باستدعاء الدالة بالشكل التالي ليتم تنفيذها

```
function_name()
```

خلينا ندخل في بعض المتعة اكتر.

مثال :

```
function sayhello() {
    console.log("hello world");
}
sayhello()
```

كيف نضيف (parameter) ؟

```
function sayhello(name) {
    console.log(`hello ${name}`);
}
sayhello("abdulrahman")
```

المعامل هو سد خانة يمثل متغير لتعلم الدالة انها ستتلقي بياناً معيناً فكما تري عند انشاء الدلة كتب name

لكنها لم تظهر مجدداً وكتب بدلاً منها نوع البيانات المناسب للعملية التي انشئت الدالة من اجلها

كيف نستعمل (return) مع ال(function)؟

بتستخدم عشان تعلي الدالة ترجعلي نتيجة معينة ولاحظ انا قلت ترجع مش تطبع

يلا نعطي مثال يوضح

```
function calc(n1, n2) {
    return n1 + n2;
}
console.log(calc(5, 9));
```

ويمكننا ايضا تخزين ناتج الدالة في متغير وطباعتة

```
function calc(n1, n2) {
    return n1 + n2;
}
var result = calc(5, 9);
console.log(`result is ${result}`);
```

خلينا نفترض اننا عاملين دالة المستخدم هو الي هيدخل البيانات ولو المستخدم مدخلش بيان معين يظهرلة كلمة زي unknown المثال التالي يوضح

```
function say_hello(name = "unkown", age = "unkown") {
    return `hello ${name} your age is ${age}`;
};
console.log(say_hello())
// hello unkown your age is unkown
```

. rest parameter .

تستخدم في حالة ان الدالة هتاخد بيانات من المستخدم بس هي بيانات احنا منعرفش عددها فهنا بنستخدمها وهي بتاخد بيانات نوعها array

لنعرفة ك parameter نقوم بكتابة ثلاث نقاط قبل اسم الـ rest parameter

ويمكن ايضاً اضافة اكثر من PARAMETER معها ولكن يجب ان تكون ال rest parameter الأخيرة

```
function skills(...skills) {
    if (skills.length === 0) {
        return ["no skills"]
    }
    else {
        return skills
    }
}
sk = skills("html", "css", "javascript", "python")
console.log(sk)
```

. function local variables .

بإختصار اي variable بنعرفة داخل function لا يمكن استخدامة خارجها

```
function sayHello(name) {
    const app_name = "soko";
    return `Hello, ${name}\nWelcome to ${app_name}!`;
}
console.log(sayHello("ganna"));
```

عند تعريف local variable لدالة او ل if او اياً كان لا تستخدم var لتعريفها فهي تصنع بعض الاخطاء يفضل استخدام const

Higher Order Function

هل دوال تتعامل مع دوال

تأخذ دوال كمعاملات "parameters"

-هي من الأشياء المهمة جداً في عالم البرمجة ويجب تعلمها اذا عدت الي هذة النقطة يوماً فقم بتعلمها

امثلة عليها :

الدالة	الوظيفة
map	**تطبق دالة علي كل عنصر في القائمة**
filter	**ترجع العناصر التي تحقق شرط معين**
reduce	**تدمج عناصر القائمة في قيمة واحدة**
forEach	**تطبق دالة علي كل عنصر في المصفوفة**

object

🧱 ما هو الـ Object؟

الكائن هو مجموعة من الخصائص (properties)، وكل خاصية تتكون من:

- اسم (key): وهو اسم الخاصية.
- قيمة (value): يمكن أن تكون رقم، نص، مصفوفة، كائن آخر، أو حتى دالة.

✍️ طريقة تعريف كائن:

```
let person = {
  name: "Ali",
  age: 30,
  isStudent: false
};
```

في هذا المثال:

- `name`, `age`, `isStudent` هي أسماء الخصائص.
- `Ali"`, `30`, `false"` هي القيم المرتبطة بها.

🔍 الوصول إلى خصائص الكائن:

يمكنك الوصول إلى القيم بطريقتين:

```
console.log(person.name);          // الطريقة الأولى: dot notation
console.log(person["age"]);        // الطريقة الثانية: bracket notation
```

⚙️ إضافة دوال داخل الكائن:

يمكن للكائن أن يحتوي على دوال تُسمى **methods**:

```
let car = {
  brand: "Toyota",
  start: function() {
    console.log("Car started");
  }
};

car.start(); // استدعاء الدالة داخل الكائن
```

🧠 استخدام الكلمة المفتاحية `this`:

داخل الكائن، `this` تشير إلى الكائن نفسه:

```
let user = {
  name: "Sara",
  greet: function() {
    console.log("Hello, " + this.name);
  }
};

user.greet(); // Hello, Sara
```

📦 مثال عملي:

```
let book = {
  title: "JavaScript Basics",
  pages: 200,
  printInfo: function() {
    console.log(`${this.title} has ${this.pages} pages.`);
  }
};

book.printInfo(); // JavaScript Basics has 200 pages.
```

BOM
Browser Object Model

🟢 `alert(message)`

- الغرض: عرض رسالة للمستخدم فقط.
- التفاعل: لا يوجد سوى زر "موافق" (OK).
- النتيجة: لا تُعيد أي قيمة.

مثال:

```
alert("مرحبًا بك!");
```

-

🟡 `confirm(message)`

- الغرض: سؤال المستخدم للحصول على تأكيد (نعم أو لا).
- التفاعل: زران "موافق" (OK) و"إلغاء" (Cancel).
- النتيجة: تُعيد `true` إذا ضغط المستخدم "موافق"، و`false` إذا ضغط "إلغاء".

مثال:

```
let isSure = confirm("هل أنت متأكد؟");
if (isSure) {
  alert("تم التأكيد!");
} else {
  alert("تم الإلغاء!");
}
```

🔵 prompt(message, default)

- الغرض: طلب إدخال من المستخدم.
- التفاعل: حقل إدخال + زران "موافق" و"إلغاء".
- النتيجة: تُعيد النص الذي أدخله المستخدم، أو null إذا ألغى.

مثال:

```
let name = prompt("ما اسمك؟" ,"ضيف");
alert(`مرحبًا، ${name}!`);
```

مثال عام :

```
/*
  BOM [Browser Object Model]
  - alert(Message) => Need No Response Only Ok Available
  - confirm(Message) => Need Response And Return A Boolean
  - prompt(Message, Default Message) => Collect Data
*/

// alert("Test");
// console.log("Test");

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

// console.log(confirmMsg);

// if (confirmMsg === true) {
//   console.log("Item Deleted");
// } else {
//   console.log("Item Not Deleted");
// }

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

console.log(promptMsg);
```

اتخاذ إجراءات بإستخدام الوقت

set time out

وهي تقوم بتنفيذ كود معين عند وقت معين

```
setTimeout(function () {
console.log("Msg");
 }, 3000);
```

في هذا المثال سيتم تنفيذ الدالة بعد 3 ثواني الي هما مكتوبين 3000 ودول معناهم 3 ثواني

مثال بطريقة اخري لعرض رسالة بعد 3ث:

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

اذا كانت الدالة التي سننفذها بعد وقت تأخذ معاملات فإننا نضيف هذة المعاملات بعد معاملات الsaytomeout

المثال التالي يوضح:

```
function sayMsg(user, age) {
    console.log(`Iam Message For ${user} Age Is : ${age}`);
```

```
}
setTimeout(sayMsg, 3000, "Osama", 38);
```

clearTimeout

تقوم بإيقاف الset time out اي انه بعد مرور الوقت المخصص لها لن تظهر نتيجتها

المثال التالي يوضح اضافة زر عند الضغط علية يوقف العد

```
let btn = document.querySelector("button");
btn.onclick = function () {
    clearTimeout(counter);
};
function sayMsg(user, age) {
    console.log(`Iam Message For ${user} Age Is : ${age}`);
}
let counter = setTimeout(sayMsg, 3000, "Osama", 38);
```

setInterval & clearInterval

تقوم بنفس ما سبق ولن ما سبق كانت بتعمل الحدث عند مرور الوقت مرة واحدة لا هنا كل ما الوقت هيعدي هي هتكررة لحد ما يتنفذ clearInterval

```
/*
  BOM [Browser Object Model]
  - setInterval(Function, Millseconds, Additional Params)
  - clearInterval(Identifier)
*/

// setInterval(function () {
//   console.log(`Msg`);
// }, 1000);

// setInterval(sayMsg, 1000);

// function sayMsg() {
//   console.log(`Iam Message`);
// }

// setInterval(sayMsg, 1000, "Osama", 38);

// function sayMsg(user, age) {
//   console.log(`Iam Message For ${user} His Age Is: ${age}`);
// }

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

function countdown() {
    div.innerHTML -= 1;
    if (div.innerHTML === "0") {
        clearInterval(counter);
    }
}
```

```
let counter = setInterval(countdown, 1000);
```

Location

Window Open And Close

تفتح صفحة وتغلق الصفحة التي فتحت من خلالها فقط ولا تملك القدرة لإغلاق اي صفحات اخري
ابحث عنها"ليست مهمة جداً ولكن قد تحتاجها"

History

```
console.log(history.length);
// تعطي عدد الصفحات المفتوحة في الجلسة
history.back();
// الرجوع للصفحة السابقة في الجلسة
history.forward();
// الانتقال للصفحة التالية في الجلسة
history.go();
// الانتقال لصفحة معينة في الجلسة
// اذا كان الرقم موجب ينتقل للامام واذا كان سالب ينتقل للخلف
```

Scroll, ScrollTo, ScrollBy, Focus, Print, Stop

```
/*
  BOM [Browser Object Model]
  - stop()
  - print()
  - focus()
  - scrollTo(x, y || Options)
  - scroll(x, y || Options)
  - scrollBy(x, y || Options)
*/

let myNewWindow = window.open("https://google.com", "",
"width=500,height=500");
//لتفتح الصفحة الجديدة في المساحة التي تم تحديدها
window.scrollTo({
  left: 500,
  top: 200,
  behavior: "smooth"
});
//تقوم بالذهاب الي المسافة التي تم تحديدها
// behavior: "smooth"
// تجعل الحركة للعنصر سلسة واكثر راحة بصرياً
```

Local storage

```
/*
  BOM [Browser Object Model]
  Local Storage
  - setItem
  - getItem
  - removeItem
  - clear
  - key

  Info
  - No Expiration Time
  - HTTP And HTTPS
  - Private Tab
*/

// Set
window.localStorage.setItem("color", "#F00");
window.localStorage.fontWeight = "bold";
window.localStorage["fontSize"] = "20px";

// Get
console.log(window.localStorage.getItem("color"));
console.log(window.localStorage.color);
console.log(window.localStorage["color"]);

// Remove One
// window.localStorage.removeItem("color");

// Remove All
// window.localStorage.clear();

// Get Key
console.log(window.localStorage.key(0));

// Set Color In Page
document.body.style.backgroundColor =
window.localStorage.getItem("color");

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

Session Storage

شرحها نفس الشرح السابق الفرق بينهم انها تخزن البيانات في الجلسة فقط علي عكس ال

local storage

Destructuring

مثال

```
myArra = ["abed", "adel", "nader", "walid"]
let [a, b, c, d] = myArra
console.log(a)
console.log(b)
console.log(c)
console.log(d)
// abed
// adel
// nader
// walid
```

مثال أعقد شوية

```
/*
  Destructuring
  - Destructuring Array Advanced Examples
*/

let myFriends = ["Ahmed", "Sayed", "Ali", ["Shady", "Amr",
["Mohamed", "Gamal"]]];

// console.log(myFriends[3][2][1]);

// let [, , , [a, , [, b]]] = myFriends;

let [, , , [a, , [, b]]] = myFriends;

console.log(a); // Shady
console.log(b); // Gamal
```

swapping

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

Destructuring Object

```
/*
  Destructuring
  - Destructuring Object
*/

const user = {
  theName: "Osama",
  theAge: 39,
  theTitle: "Developer",
  theCountry: "Egypt",
};

// console.log(user.theName);
// console.log(user.theAge);
// console.log(user.theTitle);
// console.log(user.theCountry);

// let theName = user.theName;
// let theAge = user.theAge;
// let theTitle = user.theTitle;
// let theCountry = user.theCountry;

// console.log(theName);
// console.log(theAge);
// console.log(theTitle);
// console.log(theCountry);

// ({ theName, theAge, theTitle, theCountry } = user);
const { theName, theAge, theCountry } = user;

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

مثال أخر

```
/*
  Destructuring
  - Destructuring Object
  --- Naming The Variables
  --- Add New Property
  --- Nested Object
  --- Destructuring The Nested Object Only
*/

const user = {
  theName: "Osama",
  theAge: 39,
  theTitle: "Developer",
  theCountry: "Egypt",
  theColor: "Black",
  skills: {
    html: 70,
    css: 80,
  },
};

const {
  theName: n,
  theAge: a,
  theCountry,
  theColor: co = "Red",
  skills: { html: h, css },
} = user;

console.log(n);
console.log(a);
console.log(theCountry);
console.log(co);
console.log(`My HTML Skill Progress Is ${h}`);
console.log(`My CSS Skill Progress Is ${css}`);

const { html: skillOne, css: skillTwo } = user.skills;

console.log(`My HTML Skill Progress Is ${skillOne}`);
console.log(`My CSS Skill Progress Is ${skillTwo}`);
```

```
/*
  Destructuring
  - Destructuring Mixed Content
*/

const user = {
  theName: "Osama",
  theAge: 39,
  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: ${n}`);
console.log(`Your Age Is: ${a}`);
console.log(`Your Last Skill Is: ${three}`);
console.log(`Your Live In: ${e}`);
```

Regular Expressions

هي سلسلة من الرموز تُستخدم لتحديد نمط معين داخل النصوص، مثل البحث عن كلمات، التحقق من صحة مدخلات المستخدم، أو استبدال أجزاء من النص

أهم الرموز في عالم ال Regular

الرمز	المعني
i	تجاهل حالة الأحرف الكبيرة والصغيرة
g	اذا لم تستعملها سيعيد اول نتيجة فقط
|	OR
-	من العدد قبلها إلى العدد بعدها وموضحة في الأمثلة التالية
^	NOT
\w	تعيد [a-z] و [A-Z] و _ و [9-0]
\W	تعيد ما هو عكس \w
\d	تعيد [9-0]
\D	تعيد ما هو عكس \d
\s	تعيد المسافات البيضاء
\S	تعيد كل شئ الا المسافات البيضاء
\b	إذا وضعت قبل اي شئ فهي تعني ابحث عنه في بداية مقطع إذا وضعت بعد اي شئ فهي تعني ابحث عنه في نهاية مقطع
\B	إذا وضعت قبل اي شئ فهي تعني ابحث عنه لكن ليس في بداية مقطع

إذا وضعت بعد اي شئ فهي تعني ابحث عنه لكن ليس في نهاية مقطع	
one or more وتكتب بعد شرط مثال \w+	+
zero or more	*
zero or one	?
حرف n بيرمز لرقم بنحط مكانة رقم وبتبحث لينا عن العدد الي بنحددة من نوع البيانات الي قبلة	\d{n}
بنحط رقمين بين الأقواس وبترجع لينا القيم في رينج من كذا لكذا	\d{x,y}
بنحط فيها رقم وهي بترجع لينا عدد القيم لا يقل عن الرقم ولكن قد يزيد عادي	\d{x,}
End with something?	$
Start with something?	^
followed by something?	?=
Not followed by something?	?!

أهم الدوال في عالم ال Regular

تقوم بتغيير اول قيمة لقيمة جديدة(يتم تحديد كلا القيمتين بالمثال)	replace
تقوم بتغيير كل القيم التي نحددها بقيمة جديدة اي انة اذا تكرر في السطر كلمة معينة 20 مرة فإننا نحدد الكلمة ونعطية الكلمة الجديدة وسيقوم بتغييرهم جميعا وسيتم التوضيح اكثر في المثال	replace all

Pattern
Flag

```
let myString = "Hello R3 Web School I Love R3";

let regex = /R3/ig;
/* هو الجزء الذي نبحث عنه <= Pattern
/ / هو علامتي  <= Delimiters
PATTERN  ويكتب داخلها التعبير النمطي
i  <= Flag
g  <= Flag
*/
//  تجاهل حالة الأحرف الكبيرة والصغيرة <= i
/* اذا لم نستعملها سيقوم بإعادة اول نتيجة فقط <= g
أما عند استعمالها فهو سيبحث عن كل التطابقات
*/
console.log(myString.match(regex));
```

Ranges

```
let tld = "Com Net Org Info Code Io";
let tldRe = /(info|org|io)/ig;
// OR يعني | الرمز
console.log(tld.match(tldRe));
// تعيد التطابقات match الدالة

let nums = "12345678910";
let numsRe = /[0-2]/g;
// 2 إلي 0 من المعنى فيصبح إلي يعني - الرمز
console.log(nums.match(numsRe));

let notNums = "12345678910";
let notNsRe = /[^0-2]/g;
//  not يعني  ^ الرمز
console.log(notNums.match(notNsRe));

let specialNums = "1!2@3#4$5%678910";
let specialNumsRe = /[^0-9]/g;
console.log(specialNums.match(specialNumsRe));

let practice = "Os1 Os1Os Os2 Os8 Os8Os";
let practiceRe = /Os[5-9]Os/g;
/*
في السطر ده انت بتقول هات لي
Os8Os
*/
console.log(practice.match(practiceRe));
```

```
/*
  Regular Expression
  Character Classes
  . => matches any character, except newline or other line
terminators.
  \w => matches word characters. [a-z, A-Z, 0-9 And Underscore]
  \W => matches Non word characters
  \d => matches digits from 0 to 9.
  \D => matches non-digit characters.
  \s => matches whitespace character.
  \S => matches non whitespace character.
*/

let email = 'O@@@g...com O@g.com O@g.net A@Y.com O-g.com o@s.org
1@1.com';
let dot = /./g;
let word = /\w/g;
let valid = /\w@\w.(com|net)/g;
console.log(email.match(dot));
console.log(email.match(word));
console.log(email.match(valid));
```

```
/*
  Regular Expression
  Character Classes
  \b => matches at the beginning or end of a word.
  \B => matches NOT at the beginning/end of a word.

  Test Method
  pattern.test(input)
*/

let names = "Sayed 1Spam 2Spam 3Spam Spam4 Spam5 Osama Ahmed Aspamo";
let re = /(\bspam|spam\b)/ig;
console.log(names.match(re));

console.log(re.test(names));
console.log(/(\bspam|spam\b)/ig.test("Osama"));
console.log(/(\bspam|spam\b)/ig.test("1Spam"));
console.log(/(\bspam|spam\b)/ig.test("Spam1"));
```

OOP

How to Creat Object and Methods

```
// Object انشاء
let myobj = {
  // properties انشاء
  my_name: "abdo",
  my_age: 21,
  // method انشاء
  sayHello: function () {
    return `hello ${this.my_name} you ${this.my_age} that's great`
  }
}

/*
فية طريقتين تتحقق بيهم من القيم
Dot Notation
Bracket Notation
*/
console.log(myobj.my_name);
// "abdo"
console.log(myobj["my_age"])
// 21
console.log(myobj.sayHello());
// hello abdo your age is 21 that's great
```

How to Creat Object and Methods 2

```
let myobj = new Object();
myobj.name = "ABDO";
myobj.name1 = "ganna";
myobj.love = function () {
  console.log(`${this.name} love ${this.name1}`);
  myobj2 = new Object();
  myobj2.name2 = "body";
  myobj2.name3 = "ganona";
  myobj2.love = function () {
    console.log(`${this.name2} love ${this.name3}`);
  }
}
```

Nested objects

```
let myobj1 = {
  name: "ABDO",
  name1: "ganna",
  love: function () {
    console.log(`${this.name} love ${this.name1}`);
    myobj2 = {
      name2: "body",
      name3: "ganona",
      love: function () {
        console.log(`${this.name2} love ${this.name3}`);
      }
    }
  }
}
console.log(myobj1.love());
console.log(myobj2.love());
```

Defining Object With Object.create

```
mainobj = {
  name1: "abdo",
  age: 21,
  say_welcome: function () {
    if (this.age === 21) {
      console.log(`welome ${this.name1} to our website`);
    } else {
      console.log(`you can't use our website`);
    }
  }
}
// انشاء كائن من كائن موجود مسبقاً
let otherobj = Object.create(mainobj);
otherobj.name1 = "ganna";
otherobj.age = 19;
console.log(mainobj.name1);
// abdo
console.log(mainobj.age);
// 21
console.log(mainobj.say_welcome());
// welcome abdo to our website
console.log(otherobj.name1);
// ganna
console.log(otherobj.age);
// 19
console.log(otherobj.say_welcome());
// you can't use our website
```

object.assign

1.اضافة قيم من كائن لكائن اخر

```
mainobj = {
  name1: "abdo",
  age: 21,
}
myobj = {
  work: "programer",
}
Object.assign(myobj, mainobj);
console.log(myobj);
/*
abdo
21
programer
*/
```

كما هو موضح في المثال فإنها تأخذ الكائن الذي سننسح إلية الصفات كأول عنصر ثم نضع الكائنات التي سننسخ منها ويمكن وضع اكثر من كائن لتتم وراثتهم

2.انشاء كائن جديد

```
mainobj = {
  name1: "abdo",
  age: 21,
}
myobj = {
  work: "programer",
}
Object.assign(myobj, mainobj);
console.log(myobj);
const notherobj = Object.assign({}, mainobj, myobj);
console.log(notherobj);
/*
21
"abdo"
"programer"
*/
```

كما هو موضح بالمثال فإننا نقوم بإنشاء متغير فارغ ثم نستعمل معة الدالة ثم نسند لها أقواس كيرلي فارغة ثم ننسخ له كائنات اخري

3.إنشاء property جديدة

```
mainobj = {
  name1: "abdo",
  age: 21,
}
myobj = {
  work: "programer",
}
Object.assign(myobj, mainobj, { hight: 186 });
console.log(myobj);
/*
abdo
21
programer
186
*/
```

object.freeze

تقوم بإنشاء كائن لا يقبل الحذف او الاضافة

```
mainobj = Object.freeze({
  name1: "abdo",
  age: 21,
})
```

delete

تقوم بحذف property

```
mainobj = Object.freeze({
  name1: "abdo",
  age: 21,
})
delete mainobj.age
console.log(mainobj)
// abdo
```

Constructor Function

هي دالة تكتب لتنشئ قالب لكائن يمكن من خلاله إنشاء كائنات لها نفس صفات الكائن الأول المثال يوضح:

```
function phone(name, price, battary, madein, expire) {
  this.name = name
  this.price = price - 500
  //هذا السطر يقوم بعمل خصم 500 علي اي كائن يتم انشائة
  this.battary = battary
  this.madein = "china"
  // تم اسناد قيمة ثابتة للبلد المنشئ لكل الكائنات التي سيتم انشاؤها
  this.expire = expire
}
let phone1 = new phone("iphone", 60000, 5200, "", "2 years")
// تم وضع علامتي تنصيص في مكان الخاصية التي تمتلك قيمة ثابتة
console.log(phone1.name);
console.log(phone1.price);
console.log(phone1.battary);
console.log(phone1.madein);
console.log(phone1.expire);

let phone2 = new phone("samsung", 60000, 5200, "", "3 years")
console.log(phone2.name);
console.log(phone2.price);
console.log(phone2.battary);
console.log(phone2.madein);
console.log(phone2.expire);
```

Constructor Function method

```
let player = function (name) {
  this.name = name
  this.walk = function () {
    console.log("player is walking")
  }
}
let player1 = new player("r3");
console.log(player1.name);
// r3
console.log(player1.walk())
// player is walking
```

في المثال السابق وضحنا انه يمكن انشاء method ونسخها لأي عدد من الكائنات نريد إنشائه وايضاً انه اذا كان الproperty او الmethod يمتلكون قيمة ثابتة فإنه لا يشترط وضعها في الأقواس اي انه يمكنك اضافة اي عدد من ال methods & properties دون الحاجة لإضافتها في الأقواس سيفيدنا ذلك أننا لن نضطر لإضافة علامات تنصيص لكل property & method ثابت عند إنشائها كائنات جديدة

class

هو يمثل الأب الذي من خلاله يمكنك إنشاء أي عدد من الكائنات

```
class player {
  // الكلاس واسمها
  constructor(name) {
    this.name = name
  }
//properties الـ تعريف فيها يتم التي المنطقة وهو  constructor ال
  walk() {
    console.log("player is walking")
  }
// methods ال تعريف فيها يتم التي المنطقة هذه
}
let player1 = new player("r3");
// class من كائن إنشاء
console.log(player1.name);
console.log(player1.walk())
```

Inheritance

الوراثة التي من خلالها يمكن لكلاس أن يرث خصائص و methodes من كلاس آخر

```
class animal {
  constructor(name, speed) {
    this.name = name;
    this.speed = speed;
  }
  run() {
    console.log(`the ${this.name} is run at speed ${this.speed} km/h`)
  }
}
class bird extends animal {
  constructor(name, speed) {
    super(name, speed)
  }
  run() {
    console.log(`the ${this.name} is fly at speed ${this.speed} km/h`)
  }
}
let bee = new bird;
bee.name = "bee"
bee.speed = 2
console.log(bee.run())
// the bee is fly at speed 2 km/h
```

Encapsulation

```
/*
  Encapsulation
  - Class Fields Are Public By Default
  - Guards The Data Against Illegal Access.
  - Helps To Achieve The Target Without Revealing Its Complex
Details.
  - Will Reduce Human Errors.
  - Make The App More Flexible And Manageable.
  - Simplifies The App.
*/

class User {
  // Private Property
  #e;
  constructor(id, username, eSalary) {
    this.i = id;
    this.u = username;
    this.#e = eSalary;
  }
  getSalary() {
    return parseInt(this.#e);
  }
}

let userOne = new User(100, "Elzero", "5000 Gneh");

console.log(userOne.u);
console.log(userOne.getSalary() * 0.3);
```

DATE & TIME

```
/*
  Date And Time
  - Date Constructor

  Static Methods
  - Date.now()

  - To Track Time You Need Starting Point
  - Epoch Time Or Unix Time In Computer Science Is The Number of
Seconds Since January 1, 1970.
  - Why 1970 [829 Days To 136 Years]

  Search For
  - Year 2038 Problem in Computer Science.
*/

let dateNow = new Date();

console.log(dateNow);

console.log(Date.now()); // 1000 Mill = 1 Second

let seconds = Date.now() / 1000; // Number Of Seconds
console.log(`Seconds ${seconds}`);

let minutes = seconds / 60; // Number Of Minutes
console.log(`Minutes ${minutes}`);

let hours = minutes / 60; // Number Of Hours
console.log(`Hours ${hours}`);

let days = hours / 24; // Number Of Days
console.log(`Days ${days}`);

let years = days / 365; // Number Of Years
console.log(`Years ${years}`);
```

```
/*
  Date And Time
  - getTime() => Number Of Milliseconds
  - getDate() => Day Of The Month
  - getFullYear()
  - getMonth() => Zero Based
  - getDay() => Day Of The Week
  - getHours()
  - getMinutes()
  - getSeconds()
*/

let dateNow = new Date();
let birthday = new Date("Oct 25, 82");
let dateDiff = dateNow - birthday;

console.log(dateDiff);
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());
console.log(dateNow.getDay());
console.log(dateNow.getHours());
console.log(dateNow.getMinutes());
console.log(dateNow.getSeconds());
```

```
/*
  Date And Time
  - setTime(Milliseconds)
  - setDate() => Day Of The Month [Negative And Positive]
  - setFullYear(year, month => Optional [0-11], day => Optional
[1-31])
  - setMonth(Month [0-11], Day => Optional [1-31]) [Negative And
Positive]
  - setHours(Hours [0-23], Minutes => Optional [0-59], Seconds =>
Optional [0-59], MS => Optional [0-999])
  - setMinutes(Minutes [0-59], Seconds => Optional [0-59], MS =>
Optional [0-999])
  - setSeconds(Seconds => [0-59], MS => Optional [0-999])
*/

let dateNow = new Date();
console.log(dateNow);

console.log("#".repeat(66));

// dateNow.setTime(0);
// console.log(dateNow);

// console.log("#".repeat(66));

// dateNow.setTime(10000);
// console.log(dateNow);

// console.log("#".repeat(66));

// dateNow.setDate(35);
// console.log(dateNow);

// dateNow.setFullYear(2020, 13);
// console.log(dateNow);

dateNow.setMonth(15);
console.log(dateNow);
```

```
/*
  Date And Time
  - Track Operations Time
```

```
  Search
  - performance.now()
  - performance.mark()
*/

// Start Time
let start = new Date();

// Operation
for (let i = 0; i < 100000; i++) {
  // document.write(`<div>${i}</div>`);
  let div = document.createElement("div");
  div.appendChild(document.createTextNode(i));
  document.body.appendChild(div);
}

// Time End
let end = new Date();

// Operation Duration
let duration = end - start;

console.log(duration);
```

Modules

1-اضافة الملفات لل html

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width,
initial-scale=1.0" />
    <link rel="stylesheet" href="main.css" />
    <title>Learn JavaScript</title>
  </head>
  <body>
    <script src="main.js" type="module"></script>
    <script src="app.js" type="module"></script>
  </body>
</html>
```

2-انشاء ملف module

```js
/*
  Modules
  - Import And Export
*/

let a = 10;
let arr = [1, 2, 3, 4];

function saySomething() {
  return `Something`;
}

export { a, arr, saySomething };
// كلمة export تعني تصدير
```

3-استیراد ملف module

```
import { a, arr, saySomething as s } from "./main.js";

console.log(a);
console.log(arr);
console.log(s());
```

JSON

java script object notation

هي عبارة عن سيرفر لمشاركة البيانات بين السيرفر والعميل

وهي مستمدة من الجافا سكريبت وهي بديل للغة xml

📦 JSON (JavaScript Object Notation)

أبسط وأخف من حيث البنية والكتابة.

أسهل في القراءة والكتابة للبشر والمبرمجين.

أسرع في المعالجة من قبل البرامج الحديثة.

مدعوم بشكل ممتاز في JavaScript وواجهات الويب.

يستخدم بكثرة في واجهات REST APIs وتطبيقات الويب الحديثة.

📄 XML (eXtensible Markup Language)

- أكثر تعقيداً من json

- يدعم السمات (Attributes) والتحقق من البنية عبر DTD أو XSD.

مناسب أكثر في الأنظمة القديمة أو التي تحتاج إلى تنسيق صارم

يستخدم في بعض المجالات مثل الوثائق والأنظمة الحكومية

SOAP APIs

إنشاء الـ json بنعمل ملف امتداده json

الجيسون يخزن البيانات على شكل قيمة ومفتاح داخل كائن

```
{
    "string": "Elzero",
    "number": 100,
    "object": { "EG": "Giza", "KSA": "Riyadh" },
    "array": ["HTML", "CSS", "JS"],
    "boolean": true,
    "null": null
}
```

Parse And Stringify

التحويل بين الجيسون والجافا سكريبت

```
*/
JSON
  - JSON.parse => Convert Text Data To JS Object
  - JSON.stringify => Convert JS Object To JSON
*/

// Get From Server
const myJsonObjectFromServer = '{"Username": "Osama", "Age": 39}';
console.log(typeof myJsonObjectFromServer);
console.log(myJsonObjectFromServer);

// Convert To JS Object
const myJsObject = JSON.parse(myJsonObjectFromServer);
console.log(typeof myJsObject);
console.log(myJsObject);

// Update Data
myJsObject["Username"] = "Elzero";
myJsObject["Age"] = 40;

// Send To Server
const myJsonObjectToServer = JSON.stringify(myJsObject);
console.log(typeof myJsonObjectToServer);
;(console.log(myJsonObjectToServer
```

XML

Promise

ال promise هو كائن object يمثل عملية غير متزامنة asynchronous قد تكتمل في المستقبل بنجاح او تفشل يعني بدل ما نستخدم call back function الي ممكن تسبب مشاكل زي الـ call back hell نستخدم promise لتنظيم الكود بشكل أنظف وأسهل

الـ Promise عنده 3 حالات

الحالة	الوصف
pending	العملية لسه شغالة وما خلصتش
fulfilled	العملية نجحت ورجعت نتيجة
rejected	العملية فشلت وحصل خطأ

style

تستخدم لتعديل الcss من خلال جافا سكريبت