

JAVASCRIPT



الوظيفة



الرمز	اسم الوظيفة

ملاحظة:

introdaction

١. javascript هي لغة برمجية او سكربت (scripting or programming language)
٢. تعمل على client side (browser) and server side
٣. تعمل على جعل الموقع تفاعلي و ديناميكي
٤. يمكن التلاعب في css and html
٥. يمكن تحريك محتوى الموقع
٦. يمكن التلاعب في قواعد البيانات والتشيك على معطيات المستخدم
٧. يمكن إنشاء ألعاب وتطبيقات موبايل وديسكتوب
٨. متطلبات هي التطبيق الجيد على css and html

ملاحظة: هناك مواقع تستخدم لحل المشكلات (proplems solving) مثل leetcode/edaip/hackerrank

ملاحظة: إضافات مهمة vscode وهي

bracket pair colorizer 2/editorconfig for vs dode/eslint/indent-rainbow/live server
/material icon theme/path intellisense/prettier – code formatter

ملاحظة: يتم إضافة ملف الجافا سكربت ما خارجي أو داخلي في الهد أو آخر الباضي

ملاحظة: يتم استخدام الكونسول الذي في كروم للاختبارات مثل الكائن window الذي له أوامر

مكان وضع السكريبت



١. ملف الجافاسكريبت أما خارجي أو داخلي
٢. يتم وضع وسم <script> أما في آخر الهيد أو في آخر الباضي
٣. يقوم المتصفح بترجمة ملف html خطوة خطوة لذلك إذا كانت الأكواد مسؤولة عن تغيير عنصر ما والعنصر لم يتم إنشائه بعد فإن الكونسول يقوم بإعطاء خطأ
٤. لذلك نقوم بوضع وسم السكريبت في آخر الباضي أو نقوم بتحميل الصفحة ثم تشغيل الأكواد وذلك عن طريق الكود
٥. `{} window.onload = function () { }` نقوم بكتابته إذا كان السكريبت خارجي أو في الهيد

comments



الرمز	اسم الوظيفة
//	كوميونت سطري
/* */	كوميونت متعدد الأسطر
وظيفة الكوميونت	وضع الملاحظات عن الأكواد أو تهميش الكود لإيقاف عمله
إختصار التهميش	ctrl + /

output



الرمز	اسم الوظيفة
window.alert() alert()	وهو أمر من الكائن window يقوم بعرض مربع تنبيه يمكن كتابته داخله أي نص أو أمر
window.print()	وهو أمر يستخدم لطباعة صفحة html لذلك نفعله عن طريق onclick
document.write()	وهو أمر من الكائن document يمكن كتابته داخله أي نص أو عناصر html لكن لاستعملها لكتابة html ملاحظة: لا نقوم باستخدامه لأنه يحذف عناصر html تبع الصفحة يلي بعده
console.log()	وهو أمر من كائن console يقوم بطباعة نص في الكونسول
console.table (["", ""])	يقوم بطباعة البيانات عن طريق جدول
web API	https://developer.mozilla.org/en-US/docs/Web/API

طريقة كتابة الجمل و المعايير

١. نقوم بكتابة سيمي كولون ; بنهاية كل تعليمة برمجة
 ٢. يمكن كتابة أكثر من تعليمة برمجية في سطر واحد ولكن تفصل بينها السمي كولون
a. `var a = 5; var b = 6; var y = a + b;`
 ٣. يمكن وضع فراغات داخل التعليمة مثل المثال السابق
 ٤. يمكن كسر السطر البرمجي إلى سطر جديد وتكملة التعليمة البرمجية في السطر الجديد
 ٥. عدم تسمية المتغيرات بكلمات محجوزة
 ٦. يتم قراءات التعليمة البرمجية من اليسار لليمين
- a. `110 <-- document.getElementById("demo").innerHTML = (5 + 6) * 10;`

Ecmascript

١. وهي منظمة عالمية تقوم بوضع معايير لمعلومات لل javascript
٢. es6 وهو إصدار لجافاسكربت تم إصداره سنة 2015 وفيه تغيرات طفيفة وميزات كثيرة على الأكواد
٣. es6-features وهو موقع يقوم يعرض الفرق بين الإصدارات السابقة والإصدار es6
a. <http://es6-features.org/#Constants>
٤. الإصدارات الحديثة من المتصفحات هي فقط تدعم التحديثات الجديدة لجافا سكربت لذلك لحل هذا المشكلة نستخدم babel
a. https://babeljs.io/repl/#?browsers=&build=&builtIns=false&corejs=3.6&spec=false&loose=false&code_lz=Q&debug=false&forceAllTransforms=false&shippedProposals=false&circleciRepo=&evaluate=false&fileSize=false&timeTravel=false&sourceType=module&lineWrap=true&presets=react%2Cstage-3&prettier=true&targets=Node-6.12&version=7.14.4&externalPlugins=

Data types

الرمز	اسم الوظيفة
'typeof()' فراغ typeof	وهو فكتشن لتحديد نوع البيانات حيث نكتب البيانات داخله أو بعده بفراغ <code>console.log(typeof("ammar"));</code>
"ammar"	نوع نصي string
12345	نوع رقمي number

500.99	نوع رقمي number
[11,12,13] ["a","a","a"]	نوع مصفوفة object = array
[name:"",age:24]	نوع كائن object و يأتي على شكل مفتاح وبعده قيمة
true = 1 false = 0	نوع شرطي boolean
undefined	نوع قيمة undefined فارغة
null	نوع قيمة null غير موجودة
function myFunc(){}	نوع تابع أو دالة function

8

variabels

الرمز	اسم الوظيفة
لإستدعاء عنصر عن طريق ايدي	<code>document.getElementById("").innerHTML = "";</code> <code>id.innerHTML = "";</code>
قواعد تسمية المتغير	<ol style="list-style-type: none"> يمكن أن يحتوي على أحرف و أرقام و شريطة سفلية _ و الدولار \$ يجب أن يبدأ بحرف أو شريطة سفلية أو دولار فقط الأسماء حساسة لحالة الأحرف أن كانت كبيرة أو صغيرة أي الأسم ذو الحرف الكبير يختلف عن الأسم ذو الحرف الصغير مثل <code>User user</code> لا يمكن استخدام الكلمات المحجوزة لتسمية المتغيرات لا يمكن أن يبدأ برقم لا يمكن أن تحتوي على سببشيل كركتر أو مسافات إذا احتوى الأسم على مقطعين يجب أن يبدأ المقطع الثاني بحرف كبير مثل <code>userName camelCase</code>
var	<code>var namevariabels = value , namevariabels2 = value2 ;</code> <code>var namevariabels , namevariabels2;</code> <ol style="list-style-type: none"> لغة الجافاسكربت هي لغة loosely typed أي هي لغة لا تحتاج لتعرفها نوع المتغير يجب تعريف المتغير بعدها يتم استخدامه ويمكن استخدامه قبل تعريفه يمكن تعريف المتغير أكثر من مرة ولا يفقد قيمته إلا إذا عدلنا عليه عندما يتم استخدام المتغير قبل تعريفه يظهر undefined ولا يمكن معرفة سبب المشكلة عندما يتم إنشاء متغير بواسطة var فإنه يظهر داخل الكائن window ويمكن استخدامه يمكن استخدام المتغير في كل ملف الجافا سكربت
let	<ol style="list-style-type: none"> لا يمكن تعريف المتغير أكثر من مرة لأن يفقد قيمته فيخرج error و يمكن معرفة سبب المشكلة ومكانها عندما يتم استخدام المتغير قبل تعريفه يظهر error و يمكن معرفة سبب المشكلة ومكانها عندما يتم إنشاء متغير بواسطة let فإنه لا يظهر داخل الكائن window و لا يمكن استخدامه يمكن استخدام المتغير في الكتلة الواحدة
const	<ol style="list-style-type: none"> لا يمكن تعريف المتغير أكثر من مرة لأن يفقد قيمته فيخرج error و يمكن معرفة سبب المشكلة ومكانها عندما يتم استخدام المتغير قبل تعريفه يظهر error و يمكن معرفة سبب المشكلة ومكانها عندما يتم إنشاء متغير بواسطة const فإنه لا يظهر داخل الكائن window و لا يمكن استخدامه يمكن استخدام المتغير في الكتلة الواحدة

string

الرمز	اسم الوظيفة
"ammam" 'ammam'	double quotes single quotes
"ammam 'qassab' "	إظهار سنغل كوتس نكتبها داخل دبل كوتس
'ammam "qassab" '	إظهار دبل كوتس نكتبها داخل سنغل كوتس
\	أو نستخدم escape \ وهي تعمل لتجاهل الخطأ في العنصر يلي بعده مباشرة لذلك نستخدم لكتابة الكوتس أو لتقسيم السترينغ لعدة أسطر مثل "ammam \"qassab\" " 'ammam \'qassab\' ' "ammam \ abd alqader \ qassab" "ammam \\ qassab"
\n	سطر جديد
\t	مسافة إفقية
تحويل النص إلى كائن	var x = new string("ammam"); console.log(typeof x); لا تنشئ النص ككائن لأن بيطن الكود ويمكن إرجاع نتائج غير متوقعة
== ===	عندما نقارن نصين تقارن القيمة فقط عندما نقارن نصين تقارن القيمة مع النوع
" " + " " 10 + " "/" + 10	يتم الربط بين نصين أو متغيرين من النصوص يتحول الرقم إلى نص
` \${متغير} `	console.log(a+" "+b+"\n"+c); = console.log(`\${a} \${b} \${c}`); نلاحظ أن في template literals (template strings) لكتابة فراغ بين السلسلتين يكفي كتابة مسطرة أما لكتابة سطر جديد يكفي فقط البدء بسطر جديد مثال قوي let title = "hello"; let desc = "ammam qassab"; let page = ` <div ><br="" class="card"></div> <div class="child"> <h2>\${title}</h2> <p>\${desc}</p> </div> </div> `; document.write(page); هذه التقنية في es6 ولكن عندما يحولها المتصفح إلى الجافا سكريبت القديمة تحتوي على الكثير من الدوال و \n لكن سنستخدم لبناء الصفحة مكتبة رايكت بدل من ذلك
logical or	تستخدم لعرض قيمة بديلة في حال كان هناك متغير فيه القيم التالية : ١. Null ٢. Undefinde ٣. 0 ٤. "" ٥. false or true

	<pre>let price = Null or Undefinde or 0 or "" or false or true ; consol.log(`the price is \${price 333}`);</pre>
Nullish coalescing operator ??	<p>نستخدم ?? لعرض قيمة بديلة في حال كان هناك متغير فيه القيم التالية :</p> <p>١. Null</p> <p>٢. Undefinde</p> <pre>let price = Null or Undefinde ; consol.log(`the price is \${price ?? 333}`);</pre>

10

string method

الرمز	اسم الوظيفة
str	var str = "ammar qassab";
length()	تقوم بإرجاع طول السلسلة وعندما تكون السلسلة فارغة يرجع القيمة 0 مثل str.length(); النتيجة 12 لا تبدأ من صفر
slice(start,end)	لعرض جزء من السلسلة و نحدد جزء العرض من نقطة البداية إلى نقطة النهاية و يبدأ من ال 0 مثل str.slice(6,11); و إذا رقم سالب يبدأ العد من نهاية السلسلة مثل str.slice(-6,-1); و إذا حذفنا الجزء الثاني سيطلع من البداية إلى النهاية السلسلة str.slice(-6);\ str.slice(6); مثل ال slice() بكن لا تقبل المعامل السالبة
substring(start,end)	مثل ال slice() لكن الجزء الثاني يحدد طول السلسلة وينطبق عليها السالب أو حذف الجزء الثاني
substr(start,long)	مثل ال slice() لكن الجزء الثاني يحدد طول السلسلة وينطبق عليها السالب أو حذف الجزء الثاني str.substr(0,5);/ str.substr(-6,6);/ str.substr(6);/str.substr(-6);
replace("","")	تقوم باستبدال السلسلة التي في اليسار بالسلسلة التي في اليمين مع العلم أن السلسلة التي في اليسار يجب أن تكون هي نفسها سواء بالطول أو في حالة الأحرف إذا كانت صغيرة أو كبيرة وتقوم بتغيير أول سلسلة متطابقة تقع في طريقها مثل str.replace("qassab","abd"); لإلغاء حالة حساسية الأحرف نستخدم /i / مثل str.replace(/QASSAB/i,"abd"); لتغيير جميع الأجزاء المتطابقة التي في السلسلة دفعة واحدة نستخدم /g / مثل str.replace(/a/g,"h");
repeat(رقم)	يقوم بتكرار السلسلة بعدد مرات حسب الرقم مثل str.repeat(2);
toUpperCase()	لتحويل الأحرف لأحرف كبيرة مثل str.toUpperCase();
toLowerCase()	لتحويل الأحرف لأحرف صغيرة مثل str.toLowerCase();
concat("")	لضم سلسلة إلى سلسلة أخرى مثل str = str.concat("Hello");/str = "ammar".concat(" qassab"); و يمكن ضم أكثر من سلسلة لسلسلة واحدة مثل str = str.concat("Hello",str2);
trim()	يزيل المسافة الفارغة من جانبي السلسلة مثل str = " ammar qassab ";/str.trim();
padStart(4,0)	يقوم بوضع خانات من اليسار وحسب طول الرقم حيث 4 عدد الخانات و 0 الرقم الذي يوضع مثل let text = "5"; text.padStart(4,0); الناتج هو 0005
padEnd(4,0)	يقوم بوضع خانات من اليمين وحسب طول الرقم حيث 4 عدد الخانات و 0 الرقم الذي يوضع مثل let text = "55"; text.padEnd(4,0); الناتج هو 5500
str[رقم]	يرجع حرف واحد من السلسلة حسب الرقم و في حال كانت السلسلة أصغر من الرقم يرجع undeFined
charAt(رقم)	يرجع حرف واحد من السلسلة حسب الرقم و في حال كانت السلسلة أصغر من الرقم يرجع قيمة سلسلة فارغة مثل str.charAt(0); a
split("","رقم")	لتحويل النص إلى مصفوفة مع العلم نكتب داخل التابع النص الذي بين الأحرف سواء لا يوجد شيء أو , مثل let text = "a/b/c/d"; const myArrya = text.split("/"); لإستدعاء نستدعي مثل المصفوفات myArrya[1]; والرقم يستخدم لتحديد عدد المحارف المطلوبة و كتابته تكون اختيارية

string search

الرمز	اسم الوظيفة
<code>indexOf("رقم")</code>	تقوم بالبحث بالسلسلة على أول تطابق لكلمة البحث في السلسلة وتقوم بإرجاع موقعها وإذا لا يوجد تطابق تقوم بإرجاع قيمة 1- مثل <code>str.indexOf("qas")</code> النتيجة 7 العد من أول السلسلة تقبل رقم لتبدأ البحث منه من بداية السلسلة مثل <code>str.indexOf("a",6)</code>
<code>lastindex("رقم")</code>	تقوم بالبحث بالسلسلة على آخر تطابق لكلمة البحث في السلسلة وتقوم بإرجاع موقعها وإذا لا يوجد تطابق تقوم بإرجاع قيمة 1- مثل <code>str.lastIndexOf("qas")</code> النتيجة 7 العد من أول السلسلة تقبل رقم لتبدأ البحث منه من نهاية السلسلة مثل <code>str.indexOf("a",7)</code>
<code>search("")</code>	نفس <code>indexOf</code> لكن لا تقبل رقم لبدا السلسلة تقوم بالبحث بالسلسلة على أول تطابق لكلمة البحث في السلسلة وتقوم بإرجاع موقعها وإذا لا يوجد تطابق تقوم بإرجاع قيمة 1- مثل <code>str.search("qas")</code> النتيجة 7 العد من أول السلسلة لا تقبل رقم لتبدأ البحث منه من بداية السلسلة وهي حساسة لحالة الأحرف وبالتالي البحث قد يفشل لذلك نستخدم <code>/string/i</code>
<code>includes("")</code>	تقوم ببحث عن تركيب أو كلمة أو جزء من كلمة بحيث لا يكون هناك شيء زائد أما إذا هناك نقص لا مشكلة في حال كان التركيب متطابق يقوم بإرجاع القيمة <code>true</code> أما في حال كان التركيب غير متطابق يرجع <code>false</code> مثل <code>str.includes("ammmar")</code> النتيجة <code>false</code> مثل <code>str.includes("qassa")</code> النتيجة <code>true</code>
<code>startswith("رقم")</code>	تقوم بمقارنة الكلمة بأول السلسلة حصرا إذا كانت الكلمة موجودة تقوم بإرجاع <code>true</code> وإذا كانت غير موجودة تقوم بإرجاع <code>false</code> مثل <code>str.startswith("ammar")</code> النتيجة <code>true</code> ويمكن وضع رقم لتحديد مكان البدء مثل <code>str.startswith("qassab",6)</code>
<code>endswith("رقم")</code>	تقوم بمقارنة الكلمة بآخر السلسلة حصرا إذا كانت الكلمة موجودة تقوم بإرجاع <code>true</code> وإذا كانت غير موجودة تقوم بإرجاع <code>false</code> مثل <code>str.endswith("qassab")</code> النتيجة <code>true</code> ويمكن وضع رقم لتحديد مكان البدء مثل <code>str.endswith("ammar",7)</code>

string Html

الرمز	اسم الوظيفة
link("لينك")	str.link("https"); لفتح النص كلينك مثل
strike()	لعرض خط وسط السلسلة
fontsize(رقم)	
fontcolor("لون")	
sup()	
sub()	
italics()	
bold()	
big()	
small()	

Numbers

١. في الجافا سكريبت لا تحتاج لتعريف الرقم إذا كان رقم قصير أو طويل أو عشري أو صحيح على عكس اللغات
٢. يمكن أن يحتوي الرقم على 15 خانة إذا كان عدد صحيح أو 17 خانة إذا كان عدد عشري
٣. المتصفح يقرأ من اليسار إلى اليمين لذلك إذا كان هناك سلسلة على اليسار سيقوم بتحويل الأرقام إلى سلسلة

الرمز	اسم الوظيفة
الخانات	١. إذا كان العدد ليس عشري يكون مكون من 15 خانة مثل 3000000000000000 ٢. إذا كان العدد عشري يكون مكون من 17 خانة مثل 0.00000000000000003
e+ e-	في حال كان لدينا أصفار كثر نستخدم e+ أو e- مثل 123e+3 = 123000 123e-5 = 0.0123
عشري + عشري	1000000 = 1_000_000 حيث يتم تجاهل الشريطة السفلية عند جمع عددين عشريين يكون الناتج عدد عشري مع 17 خانة لذلك لحل المشكلة نضرب بعشرة ونقسم على عشرة مثل let x = 0.1 + 0.2 = 0.300000000000000004 let x = (0.1*10 + 0.2*10)/10 = 0.3
صحيح + صحيح	let x = 10 + 20 = 30
	المتصفح يقرأ من اليسار إلى اليمين لذلك في حال + إذا كان هناك سلسلة على اليسار سيقوم بتحويل الأرقام إلى سلسلة
فقط في حال +	١. let x = 10 + 20 = 30 ٢. let x = "10" + "20" = 1020 ٣. let x = 10 + "20" = 1020 ٤. let x = "10" + 20 = 1020 ٥. let x = "ammar : " + 10 + 20 = ammar : 1020 ٦. let x = 10 + 20 + " ammar" = 30 ammar
أما في حال باقي العمليات يتعامل معها كأنها عمليات حسابية مثل	١. let x = "100" - "10" = 90 ٢. let x = "100" * "10" = 1000 ٣. let x = "100" / "10" = 10

/ * -	٤. <code>let x = "100" + "10" = 10010</code>
NaN	Not a Number معناها ليس رقم أي رقم غير منطقي ١. <code>let x = "100" / "10" = 10</code> ٢. <code>let x = "100" / "ammar" = NaN</code> ٣. <code>let x = NaN + 5 = NaN</code> ٤. <code>let x = NaN + "5" = NaN5</code> ٥. <code>let x = 10 - "ammar" = NaN</code>
isNaN()	isNaN(NaN) = number
typeof NaN	number
Infinity - Infinity	زائد لانهاية = Infinity ناقص لانهاية = -Infinity ١. عندما يكون الرقم كبير تظهر الانهاية أي Infinity مثل ٢. <code>let x = 2 / 0 = Infinity</code> ٣. <code>let x = -2 / 0 = -Infinity</code>
typeof Infinity	number
toString(الأساس)	١. وظيفتها التحويل للأنظمة حسب الأساس ٢. لنفرض لدينا الرقم الصحيح وهو نظامه عشري ; <code>let x = 30</code> لتحويله للأنظمة كالتالي ٣. النظام الثنائي (Binary) مثل <code>x.toString(2); = 1110</code> أعدداه 0,1 ٤. النظام الثماني (Octal) مثل <code>x.toString(8); = 36</code> أعدداه 0,1,2,3,4,5,6,7 ٥. النظام العشري (Decimal) مثل <code>x.toString(10); = 30</code> أعدداه 0,1,2,3,4,5,6,7,8,9 ٦. النظام السادس عشر (Hexadecimal) مثل <code>x.toString(16)=1e</code> أعدداه 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
تحويل الرقم إلى كائن	<code>let x = new Number(123);</code> <code>console.log(typeof x);</code> لا تنشئ الرقم ككائن لأنه يبطن الكود ويمكن إرجاع نتائج غير متوقعة
== ===	عندما نقارن رقمين تقارن القيمة فقط عندما نقارن رقمين تقارن القيمة مع النوع

14

Number method

الرمز	اسم الوظيفة
	<code>let x = 33; let y = 3.14;</code>
toString()	تحويل الأرقام إلى نص مثل <code>x.toString()</code> ; النتيجة 33 كنص مثال آخر <code>(100 + 25).toString()</code> ; النتيجة 125 كنص ملاحظة : عندما نكتب الأساس نقوم بالتحويل إلى الأنظمة مثل <code>x.toString(2)</code>
toFixed(رقم)	يحول الرقم إلى سلسلة ويستخدم لتحديد عدد الخانات العدد العشري التي تظهر للمستخدم مثل <code>x.toFixed(1)</code> النتيجة 3.1
toPrecision(رقم)	يحول الرقم إلى سلسلة ويستخدم لتحديد طول العدد الذي يظهر للمستخدم مثل <code>x.toPrecision(2)</code> النتيجة 3.1
valueOf()	تحويل الكائن الرقمي إلى قيمته الأولية
Number()	يقوم بتحويل جميع أنواع البيانات التي يمكن تحويلها إلى أرقام أما البيانات التي لا يمكن تحويلها يرجع NaN مثل ١. <code>Number(true) = 1</code> ٢. <code>Number(false) = 0</code> ٣. <code>Number("10") = 10</code> ٤. <code>Number(" 10") = 10</code> ٥. <code>Number("10.33") = 10.33</code> ٦. <code>Number("10,33") = NaN</code> ٧. <code>Number("10 33") = NaN</code>

	٨. <code>Number("ammar") = NaN</code>
	٩. <code>Number(new Date("2017-09-30")) = 1506729600000</code>
<code>parseInt()</code>	نفس <code>Number()</code> مع اختلاف أنه يسمح بالمسافات ويرجع أول رقم صحيح مثل <code>parseInt("10.3 33") = 10</code>
<code>parseFloat()</code>	نفس <code>Number()</code> مع اختلاف أنه يسمح بالمسافات ويرجع أول رقم عشري مثل <code>parseFloat("10.3 33") = 10.3</code>
<code>isInteger(رقم)</code>	يتحقق إذا كان العدد صحيح يرجع <code>true</code> أما إذا كان غير صحيح يرجع <code>False</code>
<code>isNaN()</code>	إذا كانت الداتا <code>NaN</code> يرجع <code>true</code> أما إذا كان ليس <code>NaN</code> يرجع <code>false</code>
<code>+ unary plus</code>	١. يحول إلى أرقام موجبة ٢. <code>consol.log(+100) = 100</code> ٣. <code>consol.log(+ "100") = 100</code> ٤. <code>consol.log(+ "-100") = -100</code> ٥. <code>consol.log(+ "10.5") = 10.5</code> ٦. <code>consol.log(+ "ammar") = NaN</code> ٧. <code>consol.log(+ 0XFF) = 255</code> ٨. <code>consol.log(+ null) = 0</code> ٩. <code>consol.log(+ false) = 0</code> ١٠. <code>consol.log(+ true) = 1</code>
<code>-unary negationn</code>	١. يحول إلى أرقام سالبة ٢. <code>consol.log(-100) = -100</code> ٣. <code>consol.log(- "100") = -100</code> ٤. <code>consol.log(- "-100") = 100</code> ٥. <code>consol.log(- "10.5") = -10.5</code> ٦. <code>consol.log(- "ammar") = NaN</code> ٧. <code>consol.log(- 0XFF) = -255</code> ٨. <code>consol.log(- null) = -0</code> ٩. <code>consol.log(- false) = -0</code> ١٠. <code>consol.log(- true) = -1</code>
Math	الدوال الرياضية الجاهزة
<code>math.round()</code>	التقريب من خلال إذا كان الرقم العشري 5 وأكثر يزيد رقم أما إذا كان أقل من خمسة لا يزيد شيء مثل <code>Math.round(9.5) = 10 / Math.round(9.4) = 9</code>
<code>Math.ceil()</code>	يقرب إلى أعلى رقم دوماً مثل <code>Math.ceil(9.1) = 10</code>
<code>Math.floor()</code>	لا يقرب الأرقام أبداً مثل <code>Math.floor(9.9) = 9</code>
<code>Math.trunc()</code>	دوماً يأخذ الرقم الصحيح ويحذف الفاصلة العشرية مثل <code>Math.trunc(9.6) = 9</code>
<code>Math.sign()</code>	إذا كان الرقم موجب يرجع القيمة 1 أما إذا صفر يرجع 0 أما إذا سالب يرجع القيمة -1
<code>Math.pow()</code>	الأس مثل <code>Math.pow(8,2) = 64</code>
<code>Math.sqrt()</code>	الجذر التربيعي مثل <code>Math.sqrt(36) = 6</code>
<code>Math.cbrt()</code>	الجذر التكعيبي مثل <code>Math.cbrt(125) = 5</code>
<code>Math.abs()</code>	القيمة المطلقة مثل <code>Math.abs(-3.3) = 3.3</code>
<code>Math.PI</code>	<code>Math.PI = 3.14</code>
<code>Math.sin(d*Math.PI/180)</code>	تغيير ال d حسب الزاوية وتكون الزاوية محصورة بين <code>1 <= sin() <= -1</code>
<code>Math.cos(d*Math.PI/180)</code>	تغيير ال d حسب الزاوية وتكون الزاوية محصورة بين <code>1 <= cos() <= -1</code>
<code>Math.log()</code>	تحتسب اللوغارتم
<code>Math.log2()</code>	
<code>Math.log10()</code>	
<code>Math.max()</code>	<code>Math.max(10, 33, -33, 100) = 100</code>
<code>Math.min()</code>	<code>Math.min(10, 33, -33, 100) = -33</code>
<code>Math.random()</code>	تقوم بإظهار أرقام عشوائية ولإختيار مجال الأرقام نقوم بالتالي ١. <code>Math.floor(Math.random() * 10); = 0 => 9</code> ٢. <code>Math.floor(Math.random() * 11); = 0 => 10</code> ٣. <code>Math.floor(Math.random() * 100); = 0 => 99</code> ٤. <code>Math.floor(Math.random() * 101); = 0 => 100</code> ٥. <code>Math.floor(Math.random() * 10) + 1 ; = 1 => 10</code> ٦. <code>Math.floor(Math.random() * 10) + 3 ; = 3 => 12</code> ٧. <code>Math.floor(Math.random() * 100) + 1 ; = 1 => 100</code>

للحصول على رقم عشوائي بين رقمين
`function getRandomIntger(max, min) { return Math.floor(Math.random() * (max – min +1)) + min ; }`

Number properties

15

الرمز	اسم الوظيفة
MAX_VALUE	أكبر رقم في اللغة ولا يوجد رقم أكبر منه لذلك لا يمكن إضافة رقم عليه
MIN_VALUE	أصغر رقم في اللغة ولا يوجد رقم أصغر منه لذلك لا يمكن أنقص رقم منه
MAX_SAFE_INTEGER	أكبر رقم آمن يمكن استخدامه
POSITIVE_INFINITY	موجب لانهاية
NEGATIVE_INFINITY	سالب لانهاية
NaN	ليس رقم

arithmetic operators

16

الرمز	اسم الوظيفة
+	جمع
-	طرح
*	ضرب
/	قسمة
++	زيادة بمقدار واحد x++ مثل نزد أول ثم نطبع <code>pre increment = ++x = 1 + x</code> نطبع ثم نزيد <code>post increment = x++ = x + 1</code>
--	النقصان بمقدار واحد x-- ننقص أول ثم نطبع <code>pre decrement = --x = 1 - x</code> نطبع أول ثم ننقص <code>post decrement = x-- = x - 1</code>
**	أس هكذا في الأصدار الحديث أما في القديم يكون <code>Math.pow(x,2); = x ** 2</code>
%	باقي القسمة لمعرفة العدد الفردي 1 % 2 / إذا العدد زوجي 0 % 2 رقم
=	<code>x = y / x = y</code>
+=	<code>x += y / x = x + y</code>
-=	<code>x -= y / x = x - y</code>
*=	<code>x *= y / x = x * y</code>
/=	<code>x /= y / x = x / y</code>
**=	<code>x **= y / x = x ** y</code>
%=	<code>x %= y / x = x % y</code>
	الضرب و القسمة لها الأسبقية عن الجمع و الطرح و يمكن تغيير الأسبقية من خلال الأقواس لأن لها الأسبقية العظمة و أجراء العمليات تكون من اليسار إلى اليمين

comparison operators

الرمز	اسم الوظيفة
==	يساوي القيمة
===	يساوي القيمة و النوع
!=	لا يساوي
!==	لا يساوي لبقية و النوع
>	أكثر من
<	أقل من
>=	أكثر أو يساوي
<=	أقل أو يساوي
>>=	
<<=	
>>>=	
&=	
=	
^=	
?	عامل ثلاثي

logical operators

الرمز	اسم الوظيفة
&&	و
	أو
!	لا
&	
^	

function

الرمز	اسم الوظيفة
تعريف دالة وطريقة استدعائها	<p>function name(parameter1, parameter2, parameter3) { <i>// code to be executed</i> }</p> <p>name(parameter1, parameter2, parameter3);</p> <p>الاستدعاء</p> <p>مثل</p> <p>function myFunction(p1, p2) { return p1 * p2; }</p> <p>الريتيرن تعيد القيمة و تكتب أخر سطر في الدالة لان توقف الدالة فما بصير نكتب شي بعدها فهي مثل البريك }</p> <p>myFunction(p1, p2); let x = myFunction(4, 3);</p> <p>عندما يتم استدعائها بدون متغيرات و هي تملك متغيرات كتالي</p> <p>myFunction;</p> <p>النتيجة تكون</p> <p>{ return p1 * p2;}</p> <p>يمكن تعريفها بدون متغيرات مثل</p> <p>function name() { return 'ammar qassab'; }</p> <p>name(); ammar qassab النتيجة تكون</p> <p>يمكن تعريف الدالة في متغير مثل</p> <p>const x = function (a, b) {return a * b};</p> <p>استدعاء يكون كمتغير</p> <p>let z = x(4, 3) * 2;</p> <p>يمكن استدعاء الدالة قبل الإعلان عنها</p> <p>myFunction(5);</p> <p>function myFunction(y) { return y * y; }</p> <p>يمكن تشغيل الدالة من دون استدعائها وذلك بإضافة الأقواس () () { } function فهي تعمل عند وصول الكومبايلر لها</p> <p>(function () { document.getElementById("demo").innerHTML = "Hello! I called myself"; })();</p> <p>يمكنك طباعة الدالة من خلال تحويلها إلى نص</p> <p>function myFunction(a, b) { return a * b; }</p> <p>let text = myFunction.toString();</p> <p>ملاحظة خطيرة : يمكن استدعاء الدالة قبل أنشاءها بالحالة العادية</p>

	<p>Name(); function name(){}</p> <p>ولكن لا يمكن استدعائها قبل إنشائها في حال تم تعريفها كمتغير لذلك نعرفها أول ثم نستدعيها</p> <p>const x = function(){}; // const x = () => {}; x();</p>
<p>Arrow Functions دالة السهم</p> <p>Anonymous Function الدالة المجهولة</p>	<p>طريقة الإعلان أو التعريف</p> <p>Arrow Functions : // ES6 const x = (x, y) => x * y;</p> <p>تختلف عن الطريقة القديمة فهي لإصدار الحديث</p> <p>Anonymous Function : // ES5 var x = function(x, y) { return x * y; }</p> <p>يمكنك أن تستدعيها كذلك من خلال</p> <p>const x = (x, y) => { return x * y }; document.getElementById("demo").innerHTML = x(5, 5);</p> <p>const x = () => { return "ammar" }; document.getElementById("demo").innerHTML = x();</p> <p>تستخدم هذه الأنواع من الدوال في الأشياء التي تطلب تنفيذ لحظي أي الدالة لن نحتاج استخدامها مرة أخرى مثل</p> <p>document.getElementById("demo").innerHTML = x(); document.getElementById("demo").innerHTML = () => {}; document.getElementById("demo").innerHTML = function () {}; setTimeout(() => {},2000); setTimeout(function () {},2000);</p>
<p>Function Parameters قيم الدوال</p>	<p>function functionName(parameter1, parameter2, parameter3) { // code to be executed }</p> <p>يمكننا التعديل على القيمة أو جعلها ثابتة أي إعطاء الدالة قيم افتراضية في حال لم تعطى قيمة تعمل القيمة الافتراضية</p> <p>function myFunction(x, y = 2) { return x * y; }</p> <p>document.getElementById("demo").innerHTML = myFunction(4); >>8</p> <p>و في حال تم إعطاء قيمة لا تعمل القيمة الافتراضية</p> <p>function myFunction(x, y = 2) { return x * y; }</p> <p>document.getElementById("demo").innerHTML = myFunction(4,3); >>12</p> <p>function myFunction(x, y) { if (y === undefined) { y = 2; } // الطريقة القديمة للقيم الافتراضية y = y 2; نفس المعنى return x * y; }</p> <p>document.getElementById("demo").innerHTML = myFunction(4); >>8</p> <p>في حال لم نعرف عدد القيم التي سنمررها لدالة نستخدم ...name و هي مصفوفة حيث تخزن القيم بداخلها مثل</p> <p>function print(...num) { for (let i=0; i<num.length; i++) { console.log(num[i]); } }</p> <p>Print(11,22,34,56,78,90);</p>

	<p>دالة لجمع الأرقام</p> <pre>function sum(...num) { let cont = 0; for (let i=0; i<num.length; i++) { cont += num[i]; } return `sum = \${cont}`; } console.log(sum(11,22,34,56,78,90));</pre> <p>دالة لضرب الأرقام</p> <pre>function mult(...num) { let cont = 1; for (let i=0; i<mult.length; i++) { cont *= mult[i]; } return `mult = \${cont}`; } console.log(mult(11,22,34,56,78,90));</pre> <p>مثال الأقوى</p> <pre>function divroot(userName='un',age='un', ra=0, show='yes', ...sk) { let root = ``; let skills = ""; if (show === 'yes') { if(sk.length>0) { skills = sk.join(' '); } else { skills = 'No Skills'; } } } else { skills = 'Skills Is Hidden'; } root = ` <h2>hello \${userName}</h2> <p>age : \${age}</p> <p>Ratiip : \${ra}</p> <p>Skills : \${skills}</p> `; return root ; } document.getElementById('root').innerHTML = divroot('ammar qassab', 23, 1000, 'yes', 'Html', 'Css', 'Bootstrap', 'Javascript', 'reactjs');</pre>
The Arguments Object	<p>وظيفته الوصول إلى قيم الدالة المرسله arguments و تكون على شكل مصفوفة</p> <pre>function myFunction() { return arguments.length; } document.getElementById("demo").innerHTML = myFunction(4, 3); >>2</pre> <pre>function myFunction(a, b) { return arguments.length; } document.getElementById("demo").innerHTML = myFunction(4, 3, 5, 6); >>4</pre>

	<p>لإيجاد أكبر قيمة بين القيم الدالة</p> <pre>function findMax() { let max = -Infinity; for(let i = 0; i < arguments.length; i++) { if (arguments[i] > max) { max = arguments[i]; } } return max; } document.getElementById("demo").innerHTML = findMax(4, 5, 12, 6); >>12</pre> <p>لجمع قيم الدالة</p> <pre>function sumAll() { let sum = 0; for(let i = 0; i < arguments.length; i++) { sum += arguments[i]; } return sum; } document.getElementById("demo").innerHTML = sumAll(1, 123, 500, 115, 44, 88); >>871</pre>
typeof	<pre>function name() { return 'ammar qassab'; } typeof name; >> function</pre>
Function Invocation	<p>يتم استدعاء خواص كائن داخل دالة بواسطة this أي this مخصصة لجلب خواص كائن</p> <pre>const myObject = { firstName: "John", lastName: "Doe", fullName: function () { return this.firstName + " " + this.lastName; } } myObject.fullName(); // Will return "John Doe"</pre> <p>// This is a function constructor:</p> <pre>function myFunction(arg1, arg2) { this.firstName = arg1; this.lastName = arg2; } // This creates a new object const myObj = new myFunction("John", "Doe"); // This will return "John" myObj.firstName;</pre>
Function Call Function Apply	<p>وظيفة call() هي دمج كائنين وعندما ندمج كائنين يمكننا استخدام المفاتيح للكائنين في دالة عن طريق this كان</p> <pre>const myObject = { firstName: "John", lastName: "Doe", fullName: function () { return this.firstName + " " + this.lastName; } } // This will return "John Doe": myObject.fullName();</pre> <p>لدمج كائنين باستخدام call()</p>

```
const person = {
  fullName: function() {
    return this.firstName + " " + this.lastName;
  }
}
const person1 = {
  firstName: "John",
  lastName: "Doe"
}
const person2 = {
  firstName: "Mary",
  lastName: "Doe"
}
// This will return "John Doe":
person.fullName.call(person1);
```

مثال ٢

```
const person = {
  fullName: function(city, country) {
    return this.firstName + " " + this.lastName + ", " + city + ", " + country;
  }
}
const person1 = {
  firstName: "John",
  lastName: "Doe"
}
person.fullName.call(person1, "Oslo", "Norway");
```

وظيفة apply() مشابهة تمامًا لـ call() لكن الاختلاف بينهم هو أن call() تمرر الخواص للدالة على شكل نصوص

```
person.fullName.call(person1, "Oslo", "Norway");
```

أما apply() تمرر الخواص للدالة على شكل مصفوفة

```
person.fullName.apply(person1, ["Oslo", "Norway"]);
```

مثال

```
const person = {
  fullName: function(city, country) {
    return this.firstName + " " + this.lastName + ", " + city + ", " + country;
  }
}
const person1 = {
  firstName: "John",
  lastName: "Doe"
}
person.fullName.apply(person1, ["Oslo", "Norway"]);
```

مثال ٢

```
const person = {
  fullName: function() {
    return this.firstName + " " + this.lastName;
  }
}
const person1 = {
  firstName: "Mary",
  lastName: "Doe"
}
// This will return "Mary Doe":
person.fullName.apply(person1);
```

١. **Global Variables** : هو متغير عام تابع لكائن النافذة **the window object** وهو يكتب خارج الدالة أو الوظائف ويمكن استعماله داخل الدالة حيث الدالة تقوم بتعديل عليه وينحفظ التعديل ويمكن استخدامه مع قيمته الجديدة في وظائف أخرى ولا يمحذف إلا إذا انتقلنا لصفحة جديدة أو سكرنا الصفحة
٢. **Local Variables** : هو متغير محلي تابع لدالة أو الوظيفة المكتوب داخلها حيث يمكن استعماله داخل الدالة أو الوظيفة فقط ولا يمكن استعماله خارجها ولا تحتفظ قيمته بانتهاء الدالة أو الوظيفة بل يمحذف من الذاكرة بمجرد الإنتهاء من الدالة أو الوظيفة

```

let a = 4;
myFunction();
function myFunction() {
    document.getElementById("demo").innerHTML = a * a;
}

myFunction();
function myFunction() {
    let a = 4;
    document.getElementById("demo").innerHTML = a * a;
}

// Initiate counter
let counter = 0;
// Function to increment counter
function add() {
    counter += 1;
}
// Call add() 3 times
add();
add();
add();
// The counter should now be 3
document.getElementById("demo").innerHTML = "The counter is: " + counter; >>3

// Function to increment counter
function add() {
    let counter = 0;
    counter += 1;
}
// Call add() 3 times
add();
add();
add();
// The result is not 3 because you mix up the global and local counter
document.getElementById("demo").innerHTML = "The counter is: " + counter; >>error

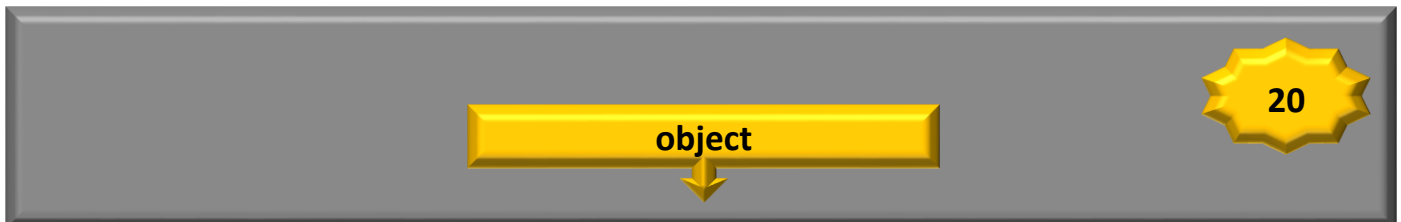
<button type="button" onclick="myFunction()">Count!</button>
<p id="demo">0</p>

<script>
const add = (function () {
    let counter = 0;
    return function () {counter += 1; return counter;};
})();
// نحفظ النتيجة النهائية في متغير عام لذلك بتزيد واحد واحد

function myFunction(){
    document.getElementById("demo").innerHTML = add();
}
</script>
كل كبسة تزيد واحد

```

	<pre> <button type="button" onclick="myFunction()">Count!</button> <p id="demo">0</p> <script> // Function to increment counter function add() { let counter = 0; counter += 1; return counter; } // Trying to increment the counter function myFunction(){ document.getElementById("demo").innerHTML = add(); } </script> </pre> <p>لا نحفظ النتيجة بمتغير عام لذلك مالح تزيد واحد غير لمرة واحدة فقط</p> <p>قد ما كبسنا على الكبسه لح ضل النتيجة واحد</p>	
Nested Functions	<pre> document.getElementById("demo").innerHTML = add(); >>1 function add() { let counter = 0; function plus() {counter += 1;} plus(); return counter; } </pre> <p>ملاحظة : الدالة الداخلية الثانية يمكنها التعديل على متغيرات الدالة الخارجية الأولى أما الخارجية لا يمكنها التعديل على متغيرات الداخلية</p>	دالة داخل دالة



الرمز	اسم الوظيفة
تعريف كائن	<p>في الجافا سكربت تقريبا كل شيء هو كائن : الجمال الشرطية و الأرقام و النصوص يمكن تحويلها لكائن <code>x = new String();</code> و التواريخ والدوال الرياضية والمصفوفات و الدوال عادية هي كلها كائنات و للكائنات نوعان : ١ . كائنات أولية مثل <code>string / number / boolean / null / undefined</code> ٢ . متغيرات مثل الأسفل</p> <pre> const name = {key:"value", key:"value"}; </pre> <p>حيث <code>key</code> تعبر عن خصائص الكائن مثل</p> <pre> const person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"}; </pre> <p>يمكن أن تحتوي الكائنات على دوال تقوم بوظائف معينة مثل</p> <pre> const person = { firstName: "John", lastName : "Doe", id : 5566, fullName : function() { return this.firstName + " " + this.lastName; } }; </pre> <p>يمكن إنشاء كائن فارغ و الإضافة عليه</p>

	<pre>const person = {}; person.firstName = "John"; person.lastName = "Doe"; person.age = 50; person.eyeColor = "blue";</pre>
استدعاء كائن من خلال خصائص الكائنات	<pre>const person = { firstName: "John", lastName : "Doe", id : 5566, fullName : function() { return this.firstName + " " + this.lastName; }, 100 : "number", "key of" : "ammar" };</pre> <p>يمكننا استدعاء الكائن من خلال يستخدم هذا النوع من الاستدعاء إذا كان المفتاح عبارة عن كلمة واحدة</p> <pre>name.key;</pre> <p>يستخدم هذا النوع من الاستدعاء إذا كان المفتاح عبارة عن كلمتين أو رقم أو أي شيء آخر غير الكلمة الواحدة أي غير محقق شروط التسمية أو إذا كان المفتاح عبارة عن متغير ديناميكي خارجي</p> <pre>name["key"];</pre> <p>مثل</p> <pre>let bra = "id"; person.firstName + " " + person.lastName; person["100"] + " " + person["key of"] + person[bra];</pre> <p>استدعاء الدوال داخل الكائن</p> <pre>person.fullName(); >> john Doe person.fullName; >> { return this.firstName + " " + this.lastName;}</pre> <p>يمكننا الوصول لكل خصائص الكائن من خلال الحلقات</p> <pre>const person = { fname:"John", lname:"Doe", age:25 };</pre> <pre>let txt = ""; for (let x in person) { txt += person[x] + " "; } document.getElementById("demo").innerHTML = txt;</pre>
Adding New Properties	<p>يمكننا إضافة خاصية جديدة للكائن من خلال ; objectName.key = value</p> <pre>const person = { firstname: "John", lastname: "Doe", age: 50, eyecolor: "blue" };</pre> <pre>person.nationality = "English"; document.getElementById("demo").innerHTML = person.firstname + " is " + person.nationality + ".";</pre>
Deleting Properties	<p>يمكننا حذف خاصية من خلال delete objectName.key; أو delete objectName["key"];</p> <pre>const person = { firstname: "John", lastname: "Doe",</pre>

	<pre> age: 50, eyecolor: "blue" }; delete person.age; document.getElementById("demo").innerHTML = person.firstname + " is " + person.age + " years old ."; // John is undefined years old . </pre>
Nested Objects	<p>كائن داخل كائن</p> <pre> myObj = { name:"John", age:30, cars: { car1:"Ford", car2:"BMW", car3:"Fiat" } } </pre> <p>طريقة الوصول لخصائص في هذه الحالة</p> <pre> document.getElementById("demo").innerHTML = myObj.cars.car2; document.getElementById("demo").innerHTML = myObj.cars["car2"]; document.getElementById("demo").innerHTML = myObj["cars"]["car2"]; </pre>
Nested Arrays and Objects	<p>طريقة بناء مصفوفة داخل كائن و طريقة الوصول إليها عن طريق الحلقات</p> <pre> let x = ""; const myObj = { name: "John", age: 30, cars: [{name:"Ford", models:["Fiesta", "Focus", "Mustang"]}, {name:"BMW", models:["320", "X3", "X5"]}, {name:"Fiat", models:["500", "Panda"]}] } for (let i in myObj.cars) { x += "<h2>" + myObj.cars[i].name + "</h2>"; for (let j in myObj.cars[i].models) { x += myObj.cars[i].models[j] + "
"; } } document.getElementById("demo").innerHTML = x; </pre>
تحويل المتغيرات إلى كائنات باستخدام new	<pre> x = new String(); // Declares x as a String object// تحويله لكائن نصي y = new Number(); // Declares y as a Number object// تحويله لكائن رقمي z = new Boolean(); // Declares z as a Boolean object// تحويله لكائن شرطي </pre>
Object Methods	<p>يمكن وضع تابع داخل الكائن و يمكن الوصول إليه من خلال this</p> <pre> // Create an object: const person = { firstName: "John", lastName: "Doe", id: 5566, fullName : function() { return this.firstName + " " + this.lastName; } }; // Display data from the object: document.getElementById("demo").innerHTML = person.fullName(); </pre>

	<p>حيث وصلنا إليه من خلال <code>person.fullName()</code> فيرجع القيمة أما <code>person.fullName()</code> فترجع <code>{ return this.firstName + " " + this.lastName; }</code> و لإضافة تابع لكائن من خارج الكائن كتالي</p> <pre>const person = { firstName: "John", lastName: "Doe", id: 5566, }; person.name = function() { return this.firstName + " " + this.lastName; }; document.getElementById("demo").innerHTML = "My father is " + person.name();</pre> <p>لاستخدام التوابيع الجاهزة داخل التابع كتالي</p> <pre>const person = { firstName: "John", lastName: "Doe", id: 5566, }; person.name = function() { return (this.firstName + " " + this.lastName).toUpperCase(); }; document.getElementById("demo").innerHTML = "My father is " + person.name();</pre>
Display Objects	<p>طرق عرض الكائن : أسم الكائن</p> <pre>const person = { name: "John", age: 30, city: "New York" }; document.getElementById("demo").innerHTML = person; // [object Object]</pre> <p>عرض خصائص معينة</p> <pre>const person = { name: "John", age: 30, city: "New York" }; document.getElementById("demo").innerHTML = person.name + ", " + person.age + ", " + person.city;</pre> <p>العرض عن طريق حلقة</p> <pre>const person = { name: "John", age: 30, city: "New York" }; let txt = ""; for (let x in person) { txt += person[x] + " "; }; document.getElementById("demo").innerHTML = txt;</pre> <p>يمكنك تحويل الكائن إلى مصفوفة ثم عرض المصفوفة عن طريق <code>Object.values()</code></p> <pre>const person = {</pre>

	<pre> name: "John", age: 30, city: "New York" }; document.getElementById("demo").innerHTML = Object.values(person); JSON.stringify() يمكن تحويل الكائن إلى نص ثم عرض النص عن طريق const person = { name: "John", age: 30, city: "New York" }; document.getElementById("demo").innerHTML = JSON.stringify(person); JSON.stringify() يمكنها تحويل التاريخ لنص var person = { name: "John", today: new Date() }; document.getElementById("demo").innerHTML = JSON.stringify(person); JSON.stringify() لا يمكنها تحويل الدوال لذلك لا تقوم بإرجاعها const person = { name: "John", age: function () {return 30;} }; document.getElementById("demo").innerHTML = JSON.stringify(person); لحل هذه المشكلة نحول الخاصية لنص أولاً ثم نقوم بعرضها const person = { name: "John", age: function () {return 30;} }; person.age = person.age.toString(); document.getElementById("demo").innerHTML = JSON.stringify(person); JSON.stringify() تقوم بتحويل المصفوفة لنص const arr = ["John", "Peter", "Sally", "Jane"]; document.getElementById("demo").innerHTML = JSON.stringify(arr); </pre>
Object Accessors	<p style="text-align: right;">Getters and Setters</p> <p>موصلات الكائنات التي تستخدم لتعديل على خصائص الكائن وهي : get lang() تستخدم للوصول إلى خصائص الكائن</p> <pre> // Create an object: const person = { firstName: "John", lastName: "Doe", language: "en", get lang() { return this.language; } }; // Display data from the object using a getter: document.getElementById("demo").innerHTML = person.lang; // Create an object: const person = { </pre> <p style="text-align: right;">set lang(value) تستخدم لتعديل على خصائص الكائن من خلال المتغيير</p>


```

firstName: "John",
lastName: "Doe",
language: "NO",
set lang(value) {
  this.language = value;
}
};

```

// Set a property using set:

```
person.lang = "en";
```

// Display data from the object:

```
document.getElementById("demo").innerHTML = person.language;
```

انتبه لطريقة الاستدعاء بدون أقواس

// Create an object:

```

const person = {
  firstName: "John",
  lastName: "Doe",
  get fullName() {
    return this.firstName + " " + this.lastName;
  }
};

```

// Display data from the object using a getter:

```
document.getElementById("demo").innerHTML = person.fullName;
```

أما في حال التابع فهي تطلب أقواس

```

const person = {
  firstName: "John",
  lastName: "Doe",
  fullName: function() {
    return this.firstName + " " + this.lastName;
  }
};

```

// Display data from the object using a method:

```
document.getElementById("demo").innerHTML = person.fullName();
```

يمكن استخدام التوابع الخاصة بالطريقتين من خلال

```

const person = {
  firstName: "John",
  lastName: "Doe",
  language: "en",
  get lang() {
    return this.language.toUpperCase();
  }
};

```

// Display data from the object using a getter:

```
document.getElementById("demo").innerHTML = person.lang;
```

//////////

// Create an object:

```

const person = {
  firstName: "John",
  lastName: "Doe",
  language: "",
  set lang(lang) {
    this.language = lang.toUpperCase();
  }
};

```

// Set a property using set:

```
person.lang = "en";
```

	<pre>// Display data from the object: document.getElementById("demo").innerHTML = person.language; Object.defineProperty() يمكن عن طريق التابع إضافة Getters and Setters للكائن بسهولة استخدام العمليات الحسابية مثل // Define an object const obj = {counter : 0}; // Define Setters and Getters Object.defineProperty(obj, "reset", {get : function () {this.counter = 0;}}); Object.defineProperty(obj, "increment", {get : function () {this.counter++;}}); Object.defineProperty(obj, "decrement", {get : function () {this.counter--;}}); Object.defineProperty(obj, "add", {set : function (value) {this.counter += value;}}); Object.defineProperty(obj, "subtract", {set : function (value) {this.counter -= value;}}); // Play with counter: obj.reset; obj.add = 5; obj.subtract = 1; obj.increment; obj.decrement; document.getElementById("demo").innerHTML = obj.counter;// return 4</pre>
Object Constructors	<p>نعلم أن يمكننا تحويل أي شئ لكائن من خلال new حيث تحول المتغير إلى كائن</p> <pre>const x1 = new String("ammar"); // A new String object const x2 = new Number(33); // A new Number object const x3 = new Boolean(true); // A new Boolean object const x4 = new Object({}); // A new Object object const x5 = new Array([]); // A new Array object const x6 = new RegExp(); // A new RegExp object const x7 = new Function(); // A new Function object const x8 = new Date(); // A new Date object</pre> <p>ملاحظة : لاتنشئ الأرقام والحروف النصوص ككائنات لانها تكون بطيئة الاستجابة</p> <p>أما الكائن المنشأ Object Constructors فيعتمد إنشائه على دالة بانية ثابتة يتم تمرير لها قيم وتحولهم إلى قيم و لمفاتيح الكائن جديد منشأ مثال ١</p> <pre>//Constructor function for Person objects function Person(first, last, age, eye) { this.firstName = first; this.lastName = last; this.age = age; this.eyeColor = eye; }</pre> <p>الدالة البانية الثابتة التي تقوم بإنشاء كائنات ذات مفاتيح ثابتة ولكن قيم هذه المفاتيح تتغير بتغيير قيم المرة { إنشاء الكائن عبر تمرير قيم مفاتيحه عبر الدالة البانية</p> <pre>const myFather = new Person("John", "Doe", 50, "blue");</pre> <p>عرض الكائن الجديد المنشأ</p> <pre>document.getElementById("demo").innerHTML= "My father is " + myFather.age + " " + typeof(myFather); >> My father is 50 object</pre> <p>مثال ٢</p> <pre>function Person(first, last, age, eye) { this.firstName = first; this.lastName = last; this.age = age; this.eyeColor = eye; }</pre> <p>يتم إنشاء كائنين مختلفين عبر دالة بانية واحدة</p> <pre>const myFather = new Person("John", "Doe", 50, "blue"); const myMother = new Person("Sally", "Rally", 48, "green");</pre>

```
document.getElementById("demo").innerHTML=
"My father is " + myFather.age + ".<br> My mother is " + myMother.age + ". ">>
// My father is 50 .
My mother is 48 .
```

مثال ٣ يمكن إضافة مفتاح جديد مع قيمة إلى الكائنات المنشأة

```
myFather.nationality = "English";
```

```
document.getElementById("demo").innerHTML=
"My father is " + myFather.nationality; >> My father is English
```

مثال ٤ يمكن إضافة مفتاح جديد مع دالة إلى الكائنات المنشأة

```
myFather.name = function() {
  return this.firstName + " " + this.lastName;
};
```

```
// Display full name
```

```
document.getElementById("demo").innerHTML =
"My father is " + myFather.name();>>My father is John Doe
```

ملاحظة : لا يمكن إضافة مفتاح إلى الدالة البانية لأن ببساطة هي دالة وليست كائن

```
function Person(first, last, age, eye) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eye;
}
Person.nationality = "English";
const myFather = new Person("John", "Doe", 50, "blue");
const myMother = new Person("Sally", "Rally", 48, "green");
```

```
document.getElementById("demo").innerHTML =
"The nationality of my father is " + myFather.nationality;
>>The nationality of my father is undefined
```

بدلاً من ذلك يتم إضافة الخاصية لدالة البانية مباشرة

```
function Person(first, last, age, eye) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eye;
  this.nationality = "English";
}
```

إضافة دالة داخل الدالة البانية

```
function Person(first, last, age, eye) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eye;
  this.name = function() {
    return this.firstName + " " + this.lastName
  };
}
const myFather = new Person("John", "Doe", 50, "blue");
document.getElementById("demo").innerHTML =
"My father is " + myFather.name(); >> My father is John Doe
```

أو يمكن استخدام prototype التي تورث الدالة والمصفوفة و الوقت الخواص

- **Date** objects inherit from **Date.prototype**
- **Array** objects inherit from **Array.prototype**
- **Person** objects inherit from **Person.prototype**

	<p>مثال ٥ : يمكن توريث الدالة البانية خواص من خلال prototype</p> <pre>function Person(first, last, age, eye) { this.firstName = first; this.lastName = last; this.age = age; this.eyeColor = eye; } Person.prototype.nationality = "English"; const myFather = new Person("John", "Doe", 50, "blue"); document.getElementById("demo").innerHTML = "The nationality of my father is " + myFather.nationality; >> The nationality of my father is English</pre> <p>يمكن توريث الدالة البانية دالة من خلال prototype</p> <pre>Person.prototype.name = function() { return this.firstName + " " + this.lastName };</pre> <p>ملاحظة : لانتشئ الأرقام و الحروف و النصوص ككائنات لانها تكون بطينة الاستجابة</p>
Object assign()	<p>تقوم بإنشاء كائن جديد من دمج عدة كائنات شكلها</p> <pre>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); // نلاحظ أن prop1 مشتركة بين الكائنين لذلك سيأخذ قيمة آخر كائن دمجو يعني prop1 : 1 let finalobject2 = object.assign(targetobject, obj1, obj2); // يمكن التعديل على القيم و إضافة خواص أخرى finalobject.prop1 = 200; finalobject.prop4 = 4; // يمكن إضافة كائن فارغ أو إضافة خواص و مفاتيح في اللحظة نفسها let finalobject3 = object.assign({}, obj1, {prop5 : 5, prop6 : 6}); console.log(finalobject3);</pre>
Object Iterables	<p>for (of) : تستخدم لطباعة أو التعديل على الكائنات القابلة لتكرار مثل Arrays, Strings, Maps, NodeLists</p> <p>مثال ١</p> <pre>const name = "W3Schools"; let text = "" for (const x of name) { text += x + "
"; } document.getElementById("demo").innerHTML = text;</pre> <p>مثال ٢</p> <pre>const letters = ["a","b","c"]; let text = ""; for (const x of letters) { text += x + "
"; } document.getElementById("demo").innerHTML = text;</pre>

Iterable objects : هو كائن قابل لتكرار حيث يتم إنشائه من خلال دالة و تعيد هذه الدالة كائن له مفتاح على شكل دالة بداخله كائن آخر له مفتاحين مع قيمتين
مثال ١

```
function myNumbers() {  
  let n = 0;  
  return {  
    next: function() {  
      n += 10;  
      return {value:n, done:false};  
    }  
  };  
}
```

الدالة تعيد كائن له مفتاح على شكل دالة و هذا المفتاح يعيد كائن آخر مع قيمتين;

```
// Create Iterable  
const n = myNumbers();  
n.next(); // 10  
n.next(); // 20  
n.next(); // 30
```

في كل مرة يتم منادات مفتاح الكائن يلي على شكل دالة يتم زيادة القيمة ب 10 حيث تتغير قيمة الكائن الداخلي
document.getElementById("demo").innerHTML = n.next().value;//40 >>40

سيتم شرح وظيفة done في المثال ٣
مثال ٢ هو عداد

```
<button onclick="app()">+</button>  
<p id="demo"></p>
```

```
<script>  
// Home Made Iterable  
function myNumbers() {  
  let n = 0;  
  return {  
    next: function() {  
      n += 10;  
      return {value:n, done:false};  
    }  
  };  
}
```

```
const n = myNumbers();
```

```
function app() {  
  document.getElementById("demo").innerHTML = n.next().value;//+10  
}
```

ملاحظة : الكائن لا يمكن تكراره عبر (of) for لذلك نقوم بتحويله لدالة التكرار Symbol.iterator التي تسمح للكائن بأن يتم تكريره و التي تعيد الكائن ذو المفتاح next() وبداخله كائن بقيمتين القيمة الأولى value هي التي تتكرر و وظيفة done إيقاف التكرار

مثال ٣

```
// Create an Object  
myNumbers = {};  
  
// Make it Iterable  
myNumbers[Symbol.iterator] = function() {  
  let n = 0;  
  done = false;  
  return {  
    next : function () {  
      n += 10;
```

	<pre> if (n == 100) {done = true} return {value:n, done:done}; } }; } let text = "" for (const num of myNumbers) { text += num + "/" } document.getElementById("demo").innerHTML = text;>> 10/20/30/40/50/60/70/80/90/ </pre>
object Map() object Set()	<p>وهي كائنات تم شرحها في الفصل 32/33 يتم إنشائها كالتالي</p> <pre> const letters = new Set(["a","b","c"]); const fruits = new Map([["apples", 500], ["bananas", 300], ["oranges", 200]]); </pre>

array

الرمز	اسم الوظيفة
الاستخدام	تستخدم المصفوفات لتخزين البيانات التي تعود لشيء واحد مثل أنواع سيارات أو أسماء أشخاص
أنشاء مصفوفة	<p>مثال</p> <pre>const array_name = [item1, item2, ...];</pre> <pre>const cars = ["mazda", "Volvo", "BMW"];</pre> <p>المسافات وفواصل الأسطر ليست مهمة. يمكن أن يمتد الإعلان على عدة أسطر:</p> <pre>const cars = ["Saab", "Volvo", "BMW"];</pre> <p>لإنشاء مصفوفة ثنائية</p> <pre>const cars = ["mazda", "Volvo", ["mazda", "Volvo", "BMW"]];</pre> <p>أو طريقة أخرى للأنشاء لكن لا نستخدمها للأنشاء من أجل السرعة</p> <pre>const cars = new Array("Saab", "Volvo", "BMW");</pre>
طريقة الإدخال	<p>أو</p> <pre>const cars = ["mazda", "Volvo", "BMW"];</pre> <pre>const cars = []; cars[0]= "Saab"; cars[1]= "Volvo"; cars[2]= "BMW";</pre> <p>في حال مصفوفة ثنائية</p> <pre>const cars = [[]]; cars[0][0]= "Saab"; cars[0][1]= "Volvo"; cars[0][2]= "BMW";</pre>
طريقة الطباعة	<p>فقط بكتابة أسم المصفوفة مثل :</p> <pre>const cars = ['ammar','qassab']; document.getElementById("demo").innerHTML = cars;</pre> <p>ملاحظة : عند طباعة مصفوفة و يكون عنصر فيها فارغ يتم طباعة الفراغ فقط</p>
الوصول إلى عناصر المصفوفة لتعديلها	<pre>const cars = ["ammar", "Volvo", "BMW"];</pre> <pre>document.getElementById("demo").innerHTML = cars[0]; >> ammar</pre> <p>حيث يكون رقم أول عنصر 0 ثم 1 و هكذا ومن أجل تعديل العنصر نصل إليه كالتالي</p> <pre>cars[0] = 'qassab'; document.getElementById("demo").innerHTML = cars[0]; >> qassab</pre> <p>للوصول للحرف الثاني داخل العنصر الأول</p> <pre>document.getElementById("demo").innerHTML = cars[0][1]; >> a</pre> <p>في حال كان لدينا مصفوفة ثنائية ونريد الوصول إليها</p> <pre>const cars = ["mazda", "Volvo", ["mazda", "Volvo", "BMW"]];</pre> <pre>document.getElementById("demo").innerHTML = cars[2][1]; >> Volvo</pre> <p>للوصول إلى كامل المصفوفة نكتب فقط أسم المصفوفة مثل</p> <pre>document.getElementById("demo").innerHTML = cars; >> ammar,Volvo,BMW</pre>
Arrays are Objects	<pre>const cars = ["mazda", "Volvo", "BMW"];</pre> <pre>cars[2];</pre> <pre>const person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};</pre> <pre>person.firstName;</pre> <p>١. تستخدم المصفوفة فهارس مرقمة</p> <p>٢. يستخدم الكائن فهارس مسماة</p>

	<p>٣. نستخدم المصفوفة عندما نريد أسماء الفهارس ارقام</p> <p>٤. نستخدم الكائنات عندما نريد أسماء الفهارس نصوص</p>
لمعرفة عدد عناصر المصفوفة	<p>نستخدم length</p> <pre>const fruits = ["Banana", "Orange", "Apple", "Mango"]; fruits.length; // Returns 4</pre>
لوصول لأول عنصر و آخر عنصر بغض النظر عن عدد العناصر	<pre>const fruits = ["Banana", "Orange", "Apple", "Mango"]; fruits[0]; // Returns "Banana" fruits[fruits.length - 1]; // Returns "Mango"</pre>
طريقة طباعة جميع العناصر	<pre>const fruits = ["Banana", "Orange", "Apple", "Mango"]; let text = ""; for (let i = 0; i < fruits.length; i++) { text += "" + fruits[i] + ""; } text += ""; document.getElementById("demo").innerHTML = text; /// const fruits = ["Banana", "Orange", "Apple", "Mango"]; let text = ""; fruits.forEach(myFunction); text += ""; document.getElementById("demo").innerHTML = text; function myFunction(value) { text += "" + value + ""; }</pre>
إضافة عنصر إلى المصفوفة	<pre>const fruits = ["Banana", "Orange", "Apple"]; fruits.push("Lemon"); // Adds a new element (Lemon) to fruits /// const fruits = ["Banana", "Orange", "Apple"]; fruits[fruits.length] = "Lemon"; // Adds "Lemon" to fruits ///</pre> <p>لا تضيف العناصر بهذه الطريقة من أجل عدم ترك فراغات في المصفوفة</p> <pre>const fruits = ["Banana", "Orange", "Apple"]; fruits[6] = "Lemon";</pre>
typeof	<pre>const fruits = ["Banana", "Orange", "Apple"]; typeof fruits; // returns object</pre>
Array.isArray()	<p>لتحقق من المصفوفة إذا كانت مصفوفة يكون الخرج true وإذا كانت ليست مصفوفة يكون الخرج false</p> <pre>const fruits = ["Banana", "Orange", "Apple"]; document.getElementById("demo").innerHTML = Array.isArray(fruits);</pre>
ملاحظات	<ol style="list-style-type: none"> لا يمكن إعادة تعيين مصفوفة تم التعريف عنها باستخدام const إلا إذا استخدمت اسم الفهرس يجب التصريح عن عناصر المصفوفة عند تعريف المصفوفة ب const أو تعرف العناصر لاحقاً باستخدام الفهرسة يرجى الانتباه إلى نطاق الكتل عند التعريف المصفوفة سواء ب const أو var var يسمح بإعادة تعديل عناصر المصفوفة بشكل يدوي على عكس const

array method

الرمز	اسم الوظيفة
toString()	تقوم بتحويل المصفوفة إلى سلاسل نصية <pre>const fruits = ["Banana", "Orange", "Apple", "Mango"]; document.getElementById("demo").innerHTML = fruits.toString();</pre> مع الملاحظة عند طباعة المصفوفة تتحول إلى سلسلة نصية
join("")	تقوم بتحويل المصفوفة إلى سلاسل نصية مع إمكانية تغيير الفاصلة بين العناصر عند الطباعة <pre>const fruits = ["Banana", "Orange", "Apple", "Mango"]; document.getElementById("demo").innerHTML = fruits.join(" / ");</pre>
pop()	وظيفتها فقط إزالة آخر عنصر بين عناصر المصفوفة <pre>fruits.pop();</pre>
shift()	وظيفتها فقط إزالة أول عنصر بين عناصر المصفوفة <pre>fruits.shift();</pre>
الحذف العناصر المحددة	<pre>const fruits = ["Banana", "Orange", "Apple", "Mango"]; document.getElementById("demo1").innerHTML = "The first fruit is: " + fruits[0]; delete fruits[0]; document.getElementById("demo2").innerHTML = "The first fruit is: " + fruits[0];</pre>
push()	وظيفتها إضافة عنصر في آخر المصفوفة <pre>const fruits = ["Banana", "Orange", "Apple", "Mango"]; fruits.push("Kiwi"); // Adds "Kiwi" to fruits</pre> أو إضافة عناصر مصفوفة الأولى في آخر مصفوفة جديدة مثال <pre>Let myNums = [1, 2, 3, 4, 5, 6]; Let newArray = []; for(let i=0;i<mynums.length;i++) { newArray.push(myNums[i] + myNums[i]); } وظيفتها جمع كل خانة في المصفوفة مع نفسها و وضعها في مصفوفة جديدة Console.log(newArray);</pre>
unshift()	وظيفتها إضافة عنصر في أول المصفوفة <pre>const fruits = ["Banana", "Orange", "Apple", "Mango"]; fruits.unshift("Lemon"); // Add "lemon"</pre>
splice()	لإضافة مجموعة عناصر إلى المصفوفة ويتم إختيار المكان حسب الرقم الأول والرقم الثاني مسؤول عن عدد العناصر التي يجب أن تحذف المكان <pre>const fruits = ["Banana", "Orange", "Apple", "Mango"]; fruits.splice(2, 0, "Lemon", "Kiwi");</pre> كما يستخدم لإزالة العناصر من المصفوفة <pre>const fruits = ["Banana", "Orange", "Apple", "Mango"]; fruits.splice(0, 1); // Removes the first element</pre>
concat()	لجمع بين مصفوفتين موجودتين <pre>const myGirls = ["Cecilie", "Lone"]; const myBoys = ["Emil", "Tobias", "Linus"]; // Concatenate (join) myGirls and myBoys const myChildren = myGirls.concat(myBoys);</pre> كما يمكنه أن يأخذ أي عدد من المصفوفات

	<p>array1.concat(array2, array3);</p> <p>كما يستخدم لإضافة العناصر إلى نهاية المصفوفة</p> <pre>const myArray = ["Emil", "Tobias", "Linus"]; const myChildren = myArray.concat("Peter") ; document.getElementById("demo").innerHTML = myChildren;</pre>
slice()	<p>لقطع أول عنصر من البداية</p> <pre>const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"]; const citrus = fruits.slice(1);</pre> <p>لقطع أول ثلاث عناصر من البداية</p> <pre>const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"]; const citrus = fruits.slice(3);</pre> <p>لقطع جزء من المصفوفة</p> <pre>const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"]; const citrus = fruits.slice(1, 3);</pre>
sort()	<p>يتم فرز العناصر حسب الابدجية</p> <pre>const fruits = ["Banana", "Orange", "Apple", "Mango"]; fruits.sort();</pre> <p>لفرز المصفوفة الأرقام تصاعديا</p> <pre>const points = [40, 100, 1, 5, 25, 10]; document.getElementById("demo").innerHTML = points ;</pre> <pre>function myFunction () { points.sort(function(a, b){return a - b}); document.getElementById("demo").innerHTML = points; }</pre> <p>لفرز المصفوفة الأرقام تنازليا</p> <pre>const points = [40, 100, 1, 5, 25, 10]; document.getElementById("demo").innerHTML = points ;</pre> <pre>function myFunction () { points.sort(function(a, b){return b - a}); document.getElementById("demo").innerHTML = points; }</pre> <p>فرز المصفوفة بترتيب عشوائي</p> <pre>const points = [40, 100, 1, 5, 25, 10]; document.getElementById("demo").innerHTML = points ;</pre> <pre>function myFunction () { points.sort(function(a, b){return 0.5 - Math.random()}); document.getElementById("demo").innerHTML = points; }</pre>
إخراج أكبر رقم في المصفوفة	<pre>const points = [40, 100, 1, 5, 25, 10]; document.getElementById("demo").innerHTML = myArrayMax(points);</pre> <pre>function myArrayMax(arr) { return Math.max.apply(null, arr); }</pre> <pre>////////// const points = [40, 100, 1, 5, 25, 10]; document.getElementById("demo").innerHTML = myArrayMax(points);</pre> <pre>function myArrayMax(arr) { let len = arr.length; let max = -Infinity; while (len--) {</pre>

	<pre> if (arr[len] > max) { max = arr[len]; } } return max; } </pre>
إخراج أصغر رقم في المصفوفة	<pre> const points = [40, 100, 1, 5, 25, 10]; document.getElementById("demo").innerHTML = myArrayMax(points); function myArrayMax(arr) { return Math.min.apply(null, arr); } //////// const points = [40, 100, 1, 5, 25, 10]; document.getElementById("demo").innerHTML = myArrayMin(points); function myArrayMin(arr) { let len = arr.length; let min = Infinity; while (len--) { if (arr[len] < min) { min = arr[len]; } } return min; } </pre>
لفرز المصفوفة الكائنات	<pre> const cars = [{type:"Volvo", year:2016}, {type:"Saab", year:2001}, {type:"BMW", year:2010}]; displayCars(); function myFunction() { cars.sort(function(a, b){return a.year - b.year}); displayCars(); } function displayCars() { document.getElementById("demo").innerHTML = cars[0].type + " " + cars[0].year + "
" + cars[1].type + " " + cars[1].year + "
" + cars[2].type + " " + cars[2].year; } //////// const cars = [{type:"Volvo", year:2016}, {type:"Saab", year:2001}, {type:"BMW", year:2010}]; displayCars(); </pre>

```
function myFunction() {
  cars.sort(function(a, b){
    let x = a.type.toLowerCase();
    let y = b.type.toLowerCase();
    if (x < y) {return -1;}
    if (x > y) {return 1;}
    return 0;
  });
  displayCars();
}

function displayCars() {
  document.getElementById("demo").innerHTML =
  cars[0].type + " " + cars[0].year + "<br>" +
  cars[1].type + " " + cars[1].year + "<br>" +
  cars[2].type + " " + cars[2].year;
}
```

array Iteration

الرمز	اسم الوظيفة
forEach()	<p>تستخدم لطباعة المصفوفة ولا تستخدم لتعديل على قيمها ولا تستخدم لإنشاء مصفوفة جديدة من قديمة هي فقط تستخدم لطباعة ولمعالجة الأحداث في ال Dom</p> <pre>const numbers = [45, 4, 9, 16, 25]; let txt = ""; numbers.forEach(myFunction); document.getElementById("demo").innerHTML = txt; function myFunction(value, index, array) { txt += value + "
"; }</pre>
map()	<p>تستخدم لإنشاء مصفوفة جديدة من خلال التعديل على قيم مصفوفة قديمة دون التأثير على المصفوفة القديمة و تأخذ داخلها (دالة و arguments) مثل map(function, arguments) و الدالة تحوي برامترات (function(value, index, array){}, arguments) وتقوم هذه الدالة بالوصول الى قيم المصفوفة القديمة والتعديل على القيم لتجعل الماب تنشيء مصفوفة جديدة</p> <p>مثال ١</p> <pre>const numbers1 = [45, 4, 9, 16, 25]; const numbers2 = numbers1.map (function(value, index, array) { return value * 2;});</pre> <p>حيث تقوم الدالة بمرور على عنصر عنصر من المصفوفة القديمة وضربهم بأثنان وتخزين في المصفوفة الجديدة حيث Value هي قيمة العنصر في المصفوفة القديمة و index هي مكان العنصر في المصفوفة القديمة array هي المصفوفة التي تتعامل معها الماب</p> <p>مثال ٢</p> <pre>document.getElementById("demo").innerHTML = numbers2;</pre> <p>مثال ٣</p> <pre>const numbers1 = [45, 4, 9, 16, 25]; const numbers2 = numbers1.map ((value) => { return value * 2;}); document.getElementById("demo").innerHTML = numbers2;</pre> <p>مثال ٤ : لتحويل الأحرف من الأحرف الكبيرة إلى صغيرة و بالعكس</p> <pre>let letter = "Ammar"; let newLetter = letter.split("").map(function (val) { return val === val.toUpperCase() ? val.toLowerCase() : val.toUpperCase() ; }).join("");</pre> <p>استخدمنا split لتحويل النص إلى مصفوفة وبعد التعديل بال map قمنا بتحويل المصفوفة المعدلة إلى نص من خلال join</p> <pre>Console.log(newLetter); >> aMMAR</pre> <p>مثال ٥ : لتحويل الأرقام السالبة إلى موجبة وبالعكس</p> <pre>let num = [-1,10,30,-40,-50];</pre>

	<pre>let invertedNumber = num.map(function(val) {return -val;}); Console.log(invertedNumber); >> [1,-10,-30,40,50]</pre> <p>مثال ٦ إزالة الأرقام من النص و الإبقاء على الأحرف</p> <pre>let ignoreNumber = "A1M2M3A4R5";</pre> <p>ملاحظة :</p> <pre>isNaN('a')= true/ isNaN(10)= false/ parseInt('a')= NaN/ parseInt(10) = 10/ isNaN(NaN)=true</pre> <pre>let ign = ignoreNumber.split("").map(function(val) { return isNaN(parseInt(val)) ? val : val=""; // isNaN(NaN)=true / isNaN(10)=false }).join(""); Console.log(ign); >> AMMAR</pre>
filter()	<p>تشبه ال map() تستخدم لإنشاء مصفوفة جديدة من خلال فلتر قيم مصفوفة قديمة من خلال شرط معين دون التأثير على المصفوفة القديمة و تأخذ داخلها (دالة و arguments) مثل filter(function, arguments)</p> <p>و الدالة تحوي برامترات (arguments, index, value, array) و تقوم هذه الدالة بالوصول الى قيم المصفوفة القديمة و فلترتها من خلال شرط يوضع بعد return (condition) في دالة الفلتر لتجعل الفلتر تنشيء مصفوفة جديدة مفلترة</p> <p>وظيفتها فلتر العناصر المصفوفة القديمة و إرجاع مصفوفة جديدة مفلترة شكلها</p> <pre>Let newArray = oldArray.filter(function(value, index, array) { return condition ? true : false; });</pre> <p>إذا نتيجة الشرط true يقوم بتمرير العنصر من المصفوفة القديمة إلى الجديدة أما إذا false لا يقوم بتمريره</p> <p>مثال ١</p> <pre>const numbers = [45, 4, 9, 16, 25]; const over18 = numbers.filter(myFunction); document.getElementById("demo").innerHTML = over18;</pre> <pre>function myFunction(value, index, array) { return value > 18; // العنصر المحقق يخزن في المصفوفة الجديدة }</pre> <p>مثال ٢ فلتر الأعداد الزوجية</p> <pre>let num = [1,2,3,4,5,6,7,8,9]; let filNum = num.filter(function(val) { return val%2 === 0 ? true : false ; }); Console.log(filNum); >> [2,4,6,8]</pre> <p>مثال ٣ فلتر الكلمات التي أحرفها أكثر من أربعة</p> <pre>let sentence = "I love foood code too playing much"; let filsentence = v.split(" ").filter(function(val) { return val.length <=4 ; }).join(" "); Console.log(filsentence); >> I love code too much</pre> <p>ملاحظة :</p> <p>الفرق بين map() و filter() :</p> <p>Map() : عند وضع داخلها شرط بدون true و false سوف ترجع نتيجة الشرط أي سوف ترجع true و false مثل</p> <pre>let ignoreNumber = "A1M2M3A4R5"; let ign = ignoreNumber.split("").map(function(val) { return isNaN(parseInt(val)) ; // فليس فقط أي المصفوفة الجديدة داخلها ترو و فليس فقط }).join(""); Console.log(ign); >> truefalse truefalse truefalse truefalse truefalse</pre> <p>filter() : عند وضع داخلها شرط بدون true و false سوف ترجع العنصر الذي حقق الشرط مثل</p> <pre>let sentence = "I love foood code too playing much"; let filsentence = v.split(" ").filter(function(val) { return val.length <=4 ; // ليس نتيجة الشرط }).join(" ");</pre>

Console.log(filsentence); >> I love code too much	
reduce()	<p>تشبه ال <code>map()</code> و <code>filter()</code> وتعمل على اختزال عناصر المصفوفة من اليسار إلى اليمين حيث تقوم بأخذ أول عنصرين من المصفوفة من خلال عملية حسابية مثل الجمع ثم نتيجة العملية الجمع بين أول عنصرين تؤخذ و تجمع مع العنصر الثالث وهكذا لها شكلان الشكل الأول :</p> <pre>oldArray.reduce(function(total, value, index, array) {return (العملية الحسابية);})</pre> <p>في هذا الشكل تكون <code>total</code> هي قيمة العنصر الأول لان لا توجد قيمة افتراضية و تكون <code>value</code> هي قيمة العنصر الثاني في المصفوفة و <code>index</code> هو اندكس العنصر الفالو دوما فيكون يساوي ١ و تكون <code>array</code> هي المصفوفة التي نأخذ منها القيم تؤخذ قيمة العنصر الثاني <code>value</code> و تجمع مع <code>total</code> فيصبح لدينا <code>total</code> جديدة تجمع مع العنصر الثالث و هكذا ليتم جمع كل العناصر في حال كانت العملية جمع أما إذا لم تكن جمع تبقى تتكرر العملية لنتهي كل المصفوفة الشكل الثاني :</p> <pre>oldArray.reduce(function(total, value, index, array) {return (العملية الحسابية);}, firsttotal)</pre> <p>في هذا الشكل تكون <code>total</code> هي قيمة الافتراضية الأولى <code>firsttotal</code> و تكون <code>value</code> هي قيمة العنصر الأول في المصفوفة و <code>index</code> هو اندكس العنصر الفالو دوما فيكون يساوي ٠ و تكون <code>array</code> هي المصفوفة التي نأخذ منها القيم تؤخذ قيمة العنصر الأول <code>value</code> و تجمع مع <code>total</code> فيصبح لدينا <code>total</code> جديدة تجمع مع العنصر الثاني و هكذا ليتم جمع كل العناصر في حال كانت العملية جمع أما إذا لم تكن جمع تبقى تتكرر العملية لنتهي كل المصفوفة مثال ١</p> <pre>const numbers = [45, 4, 9, 16, 25]; let sum = numbers.reduce(myFunction); document.getElementById("demo").innerHTML = "The sum is " + sum; >> The sum is 99 function myFunction(total, value, index, array) { return total + value; }</pre> <p>مثال ٢ يمكن إضافة عدد أولي لعملية</p> <pre>const numbers = [45, 4, 9, 16, 25]; let sum = numbers.reduce(myFunction, 100); document.getElementById("demo").innerHTML = "The sum is " + sum; >> The sum is 199 function myFunction(total, value) { return total + value; }</pre> <p>مثال ٣ اخراج أكبر كلمة في المصفوفة</p> <pre>let sentence = ["bla", "propaganda", "ammar", "abdalqader", "qassab"]; let newSentence = sentence.reduce(function(total,value) { return total.length > value.length ? total : value; }); document.getElementById("demo").innerHTML = "The " + newSentence; >> abdalqader</pre>
reduceRight()	<p>تعمل على اختزال عناصر المصفوفة من اليمين إلى اليسار من خلال عملية حسابية مثل الجمع</p> <pre>const numbers = [45, 4, 9, 16, 25]; let sum = numbers.reduceRight(myFunction); document.getElementById("demo").innerHTML = "The sum is " + sum; function myFunction(total, value, index, array) { return total + value; }</pre>
every()	<p>طريقة لتحقق من ان جميع عناصر المصفوفة أتمت الاختبار فترجع <code>true</code> or <code>false</code></p> <pre>const numbers = [45, 4, 9, 16, 25]; let allOver18 = numbers.every(myFunction);</pre>

	<pre>document.getElementById("demo").innerHTML = "All over 18 is " + allOver18; function myFunction(value, index, array) { return value > 18; }</pre>
some()	<p>طريقة لتحقق من ان بعض عناصر المصفوفة أتمت الاختبار فترجع true or false</p> <pre>const numbers = [45, 4, 9, 16, 25]; let someOver18 = numbers.some(myFunction); document.getElementById("demo").innerHTML = "Some over 18 is " + someOver18; function myFunction(value, index, array) { return value > 18; }</pre>
indexOf()	<p>تبحث عن قيمة العناصر في المصفوفة والبحث من البداية وتعيد موضعه <code>array.indexOf(item, start)</code></p> <pre>const fruits = ["Apple", "Orange", "Apple", "Mango"]; let position = fruits.indexOf("Apple") + 1; document.getElementById("demo").innerHTML = "Apple is found in position " + position;</pre>
lastIndexOf()	<p>تبحث عن قيمة العناصر في المصفوفة والبحث من النهاية وتعيد موضعه <code>array.lastIndexOf(item, start)</code></p> <pre>const fruits = ["Apple", "Orange", "Apple", "Mango"]; let position = fruits.lastIndexOf("Apple") + 1; document.getElementById("demo").innerHTML = "Apple is found in position " + position;</pre>
includes()	<p>يبحث عن قيمة العنصر في المصفوفة إذا وجد يرجع true و إذا لم يوجد يرجع false</p> <pre>const fruits = ["Banana", "Orange", "Apple", "Mango"]; document.getElementById("demo").innerHTML = fruits.includes("Mango");</pre>
find()	<p>يرجع قيمة أول عنصر في المصفوفة يجتاز الاختبار</p> <pre>const numbers = [4, 9, 16, 25, 29]; let first = numbers.find(myFunction); document.getElementById("demo").innerHTML = "First number over 18 is " + first; function myFunction(value, index, array) { return value > 18; }</pre>
findIndex()	<p>يرجع مكان العنصر في المصفوفة الذي يجتاز الاختبار</p> <pre>const numbers = [4, 9, 16, 25, 29]; document.getElementById("demo").innerHTML = "First number over 18 has index " + numbers.findIndex(myFunction); function myFunction(value, index, array) { return value > 18; }</pre>
from()	<pre>const myArr = Array.from("ABCDEFGH"); document.getElementById("demo").innerHTML = myArr;</pre>
Keys()	<p>يقوم بإرجاع الفهارس للمصفوفة</p> <pre>const fruits = ["Banana", "Orange", "Apple", "Mango"]; const keys = fruits.keys(); let text = ""; for (let x of keys) { text += x + "
"; }</pre>

document.getElementById("demo").innerHTML = text;

data

الرمز	اسم الوظيفة
new Date()	لإنشاء كائن جديد بتاريخ و التوقيت الحالي <pre>const d = new Date(); document.getElementById("demo").innerHTML = d;</pre>
new Date(year, month, ...)	يقوم بإنشاء كائن تاريخ جديد بتاريخ ووقت محددين <pre>const d = new Date(2018, 11, 24, 10, 33, 30, 0); document.getElementById("demo").innerHTML = d;</pre> <p>ملاحظة : أن الأشهر تبدأ من 0 إلى 11 و أن أي إضافة أكبر من 11 ستؤدي إلى الإضافة إلى السنة التالية</p> <pre>const d = new Date(2018, 15, 24, 10, 33, 30, 0); document.getElementById("demo").innerHTML = d;</pre> <p>(توقيت شرق أوروبا الصيفي) //Wed Apr 24 2019 10:33:30 GMT+0300 وكذلك بالنسبة للفائض بالنسبة لأيام سيؤدي لأن يضيف الفائض إلى الشهر المقبل</p> <pre>const d = new Date(2018, 05, 35, 10, 33, 30, 0); document.getElementById("demo").innerHTML = d;</pre> <p>(توقيت شرق أوروبا الصيفي) //Thu Jul 05 2018 10:33:30 GMT+0300 يمكننا إلغاء أي من العناصر ولكن ستبقى مرتبة كنالي سنه شهر يوم ساعة دقائق ثواني ميلي ثانية لكن لا يمكن ترك السنه لوحدها يجب أن يكون معها الشهر أقل شيء لان إذا تم تركها لوحدها سيتم إعتبارها ميلي ثانية</p>
new Date(dateString)	ينشئ كائن تاريخ جديداً من سلسلة نصية <pre>const d = new Date("October 13, 2014 11:13:00"); document.getElementById("demo").innerHTML = d;</pre>
new Date(milliseconds)	عند وضع خانة واحدة مثل ما قلنا سيتم إعتبارها ميلي ثانية وعند وضع قيمتها صفر سترجعنا إلى تاريخ 1970 <pre>const d = new Date(0); document.getElementById("demo").innerHTML = d;</pre> <p>(توقيت شرق أوروبا الرسمي) ///Thu Jan 01 1970 02:00:00 GMT+0200 يمكننا إضافة يوم لتاريخ من خلال إضافة الرقم 86400000 ويمكننا إنقاص يوم من خلال الرقم -86400000. وهكذا</p> <pre>const d = new Date(86400000); document.getElementById("demo").innerHTML = d;</pre> <p>(توقيت شرق أوروبا الرسمي) //Fri Jan 02 1970 02:00:00 GMT+0200</p>
toString()	وهي الافتراضية حيث تقوم بإخراج التاريخ على شكل نص <pre>const d = new Date(); document.getElementById("demo").innerHTML = d.toString();</pre>
toUTCString()	هذه ستغير طريقة عرض النص كسلسلة ويكون معيار uts <pre>const d = new Date(); document.getElementById("demo").innerHTML = d.toUTCString(); ///Thu, 09 Sep 2021 09:40:26 GMT</pre>
toDateString()	تقوم الطريقة بتحويل التاريخ إلى تنسيق أكثر قابلية للقراءة <pre>const d = new Date(); document.getElementById("demo").innerHTML = d.toDateString(); ///Thu Sep 09 2021</pre>
toISOString()	تقوم الطريقة بتحويل التاريخ إلى تنسيق القياسي <pre>const d = new Date(); document.getElementById("demo").innerHTML = d.toISOString(); /// 2021-09-09T09:43:24.420Z</pre>

data formats

الرمز	اسم الوظيفة
iso	نكتب التاريخ كسلسلة نصية
ISO Dates	<pre>const d = new Date("2015-03-25"); document.getElementById("demo").innerHTML = d;</pre>
ISO Dates (Date-Time)	<pre>const d = new Date("2015-03-25T12:00:00Z"); document.getElementById("demo").innerHTML = d;</pre> <p>حيث T تفصل بين التاريخ والوقت و تشير z أن تم تعديل الوقت إلى التوقيت العالمي أي لتوقيت المنطقة الزمنية الحالية ولكن يمكننا إزالة z وإضافة الوقت الحالي</p> <pre>document.getElementById("demo").innerHTML = new Date("2015-03-25T12:00:00-06:00");</pre> <p>عندما لا يتم تحديد المنطقة الزمنية سيتم أخذ توقيت المتصفح</p>
Short Dates	<pre>const d = new Date("03/25/2015"); document.getElementById("demo").innerHTML = d;</pre>
تحذيرات	<p>١. يمكن ان يظهر خطأ إذا تم وضع التاريخ بدون أصفار يجب أن يكون</p> <pre>const d = new Date("2015-03-25");</pre> <p>٢. لا يمكن كتابة التاريخ بالشكل هذا / <</p> <pre>const d = new Date("2015/03/25");</pre> <p>٣. لا يمكن كتابة التاريخ بالشكل هذا <</p> <pre>const d = new Date("25-03-2015");</pre>
Long Dates	<p>حيث يكون الشهر على شكل نص ويكون الترتيب على الشكل التالي يوم شهر سنة</p> <pre>const d = new Date("25 Mar 2015"); document.getElementById("demo").innerHTML = d;</pre> <p>ويمكن كتابة الشهر بشكل كامل أو إختصار الكلمة ويمكن تبديل مكان الشهر باليوم</p>
Date.parse()	<p>حيث تحول التاريخ المعطى بشكل سلسلة إلى رقم ميلي ثانية</p> <pre>const msec = Date.parse("21 March 2012"); document.getElementById("demo").innerHTML = msec;</pre> <p>يمكنك بعد ذلك استخدام عدد الملي ثانية لتحويله إلى كائن تاريخ</p> <pre>let msec = Date.parse("21 March 2012"); const d = new Date(msec); document.getElementById("demo").innerHTML = d;</pre>

Get Date Methods

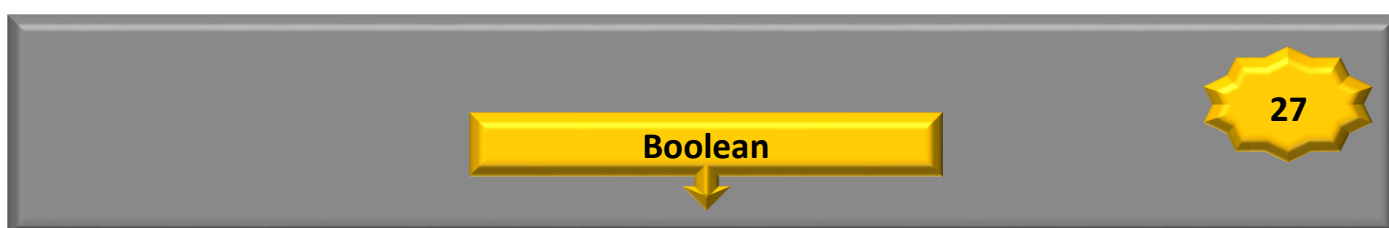
الرمز	اسم الوظيفة
	تستخدم هذه الطريقة للحصول على التاريخ و الوقت
getTime()	<p>يعطي رقم ميلي ثانية من التاريخ 1970 إلى التاريخ الحالي</p> <pre>const d = new Date(); document.getElementById("demo").innerHTML = d.getTime();</pre>
getFullYear()	<p>ترجع التاريخ الحالي كرقم مكون من أربع أرقام</p> <pre>const d = new Date(); document.getElementById("demo").innerHTML = d.getFullYear();</pre>
getMonth()	<p>ترجع الشهر الحالي كرقم من 0 إلى 11 لذلك نضفها واحد</p> <pre>const d = new Date(); document.getElementById("demo").innerHTML = d.getMonth() + 1;</pre> <p>يمكننا إرجاع الشهر كنص باستخدام هذه الطريقة</p>

	<pre>const d = new Date(); const months = ["January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November", "December"]; document.getElementById("demo").innerHTML = months[d.getMonth()];</pre>
getDate()	طريقة لإرجاع اليوم كرقم من 1 إلى 31 <pre>const d = new Date(); document.getElementById("demo").innerHTML = d.getDate();</pre>
getHours()	طريقة لإرجاع الساعات كرقم من 0 إلى 23 <pre>const d = new Date(); document.getElementById("demo").innerHTML = d.getHours();</pre>
getMinutes()	طريقة لإرجاع الدقائق كرقم من 0 إلى 59 <pre>const d = new Date(); document.getElementById("demo").innerHTML = d.getMinutes();</pre>
getSeconds()	طريقة لإرجاع الثواني كرقم من 0 إلى 59 <pre>const d = new Date(); document.getElementById("demo").innerHTML = d.getSeconds();</pre>
getMilliseconds()	طريقة لإرجاع الملي ثانية كرقم من 0 إلى 999 <pre>const d = new Date(); document.getElementById("demo").innerHTML = d.getMilliseconds();</pre>
getDay()	طريقة لإرجاع الأيام كرقم من 0 إلى 6 <pre>const d = new Date(); document.getElementById("demo").innerHTML = d.getDay() + 1;</pre> <p>ويمكن إرجاع أيام الأسبوع كنص باستخدام هذا الطريقة</p> <pre>const d = new Date(); const days=["Saturday", "Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday"]; document.getElementById("demo").innerHTML = days[d.getDay()];</pre>
UTC	وهي طريقة من أجل تحديد المنطقة الزمنية و ذلك من خلال إضافة UTC بعد get مثل getUTCFullYear()




الرمز	اسم الوظيفة
	تستخدم هذا الوظائف لتعديل جزء من التاريخ مع بقاء الجزء الآخر
setFullYear()	طريقة لتعديل جزء من التاريخ أو كله <pre>const d = new Date(); d.setFullYear(2020); document.getElementById("demo").innerHTML = d; ///////// const d = new Date(); d.setFullYear(2020, 11, 3); document.getElementById("demo").innerHTML = d;</pre>
setMonth()	طريقة لتعديل الشهر كرقم من 0 إلى 11 <pre>const d = new Date(); d.setMonth(11); document.getElementById("demo").innerHTML = d;</pre>
setDate()	طريقة لتعديل اليوم كرقم من 1 إلى 31 <pre>const d = new Date(); d.setDate(15); document.getElementById("demo").innerHTML = d;</pre> <p>كما يمكننا إضافة أيام إلى التابع</p>


	<pre>const d = new Date(); d.setDate(d.getDate() + 50); document.getElementById("demo").innerHTML = d;</pre> <p>إذا أدت إضافة أيام إلى تغيير الشهر أو السنة ، فسيتم معالجة التغييرات تلقائيًا بواسطة كائن التاريخ.</p>
setHours()	<p>طريقة لتعديل الساعات من 0 إلى 23</p> <pre>const d = new Date(); d.setHours(22); document.getElementById("demo").innerHTML = d;</pre>
setMinutes()	<p>طريقة لتعديل الدقائق من 0 إلى 59</p> <pre>const d = new Date(); d.setMinutes(30); document.getElementById("demo").innerHTML = d;</pre>
setSeconds()	<p>طريقة لتعديل الثواني من 0 إلى 59</p> <pre>const d = new Date(); d.setSeconds(30); document.getElementById("demo").innerHTML = d;</pre>
setMilliseconds()	<p>طريقة لتعديل الملي ثانية من 0 إلى 999</p> <pre>const d = new Date(); d.setMilliseconds(300); document.getElementById("demo").innerHTML = d;</pre>
مثال	<pre>let text; const today = new Date(); const someday = new Date(); someday.setFullYear(2100, 0, 14); if (someday > today) { text = "Today is before January 14, 2100."; } else { text = "Today is after January 14, 2100."; } document.getElementById("demo").innerHTML = text;</pre>



الرمز	اسم الوظيفة
Boolean()	<p>عندما نعطي شرط يتحقق من الشرط ويخرج true أو false أما لا يتم إعطاء شرط يخرج true إلا إذا كان المحتوى صفر أو فارغ يخرج false</p> <pre>document.getElementById("demo").innerHTML = Boolean(10 > 9); //// document.getElementById("demo").innerHTML = "100 is " + Boolean(100) + "
" + "3.14 is " + Boolean(3.14) + "
" + "-15 is " + Boolean(-15) + "
" + "Any (not empty) string is " + Boolean("Hello") + "
" + "Even the string 'false' is " + Boolean('false') + "
" + "Any expression (except zero) is " + Boolean(1 + 7 + 3.14); ///</pre>

	100 is true 3.14 is true -15 is true Any (not empty) string is true Even the string 'false' is true Any expression (except zero) is true ///// let x = 0; Boolean(x); // returns false ////////// let x = ""; Boolean(x); // returns false //// let x; Boolean(x); // returns false ///// let x = null; Boolean(x); // returns false ///// let x = false; Boolean(x); // returns false ///// let x = 10 / "Hallo"; Boolean(x); // returns false /////
typeof	let x = false; // x is a boolean let y = new Boolean(false); // y is an object document.getElementById("demo").innerHTML = typeof x + " " + typeof y;





الرمز	اسم الوظيفة
Comparison Operators	1. == 2. === 3. != 4. !== 5. > 6. < 7. >= 8. <=
Logical Operators	1. && 2. 3. !
Syntax	<code>variablename = (condition) ? value1:value2</code> <code>value1 = true // value2 = false</code> <input id="age" value="18" /> <button onclick="myFunction()">Try it</button> <p id="demo"></p>

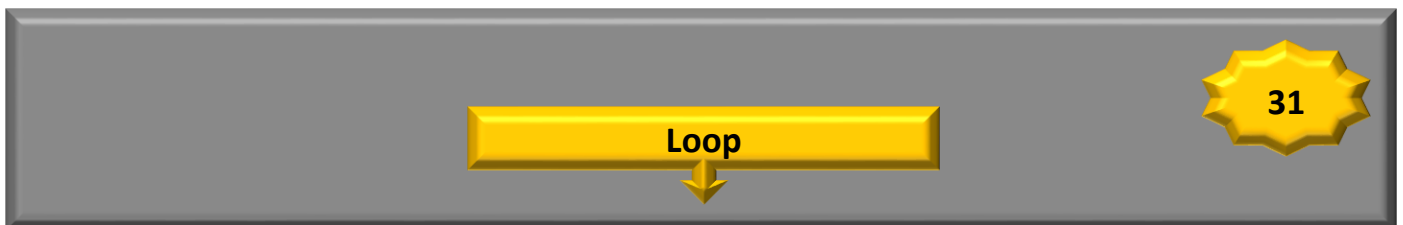
```
function myFunction() {
  let age = document.getElementById("age").value;
  let voteable = (age < 18) ? "Too young":"Old enough";
  document.getElementById("demo").innerHTML = voteable + " to vote.";
}
```

30

Conditional Statements

الرمز	اسم الوظيفة
if	<pre>if (condition) { // block of code to be executed if the condition is true }</pre>
else	<pre>if (condition) { // block of code to be executed if the condition is true } else { // block of code to be executed if the condition is false }</pre>
else if	<pre>if (condition1) { // block of code to be executed if condition1 is true } else if (condition2) { // block of code to be executed if the condition1 is false and condition2 is true } else { // block of code to be executed if the condition1 is false and condition2 is false }</pre> <p>ملاحظة : يبدأ التحقق من الشرط من الأعلى للأسفل جملة تلو الأخرى و إذا لم يتحقق أي شرط تتحقق جملة else وهي جملة شرطية بداخل جملة شرطية</p>
nested if	<pre>if (condition1) { if (condition1) { // } else { // } }</pre>
short if	<p>(condition) ? value1 : value2 >> (condition) ? true : false or else</p> <p>حيث (condition) هي if حيث true : value1 هي true حيث false أو else يعني شرط جديد يكتب هكذا</p> <p>(condition) ? true : (condition2) ? true : false ;</p> <p>ملاحظة : يمكن تخزين الشرط داخل متغير وتكون قيمة المتغير هي قيمة الشرط المحقق</p> <pre>let voteable = (10 < 18) ? "Too young":"Old enough";</pre>

	<p>لو إحتجنا أن نضيف else if نضيف ؟ ثم : التي تعبر عن else if وآخر : تعبر عن else</p> <pre>let num = (10 == "10") ? "num10" : (11>10 && 8>4) ? "num11" : "notnum";</pre> <p>لا يوجد فيها nested if أي لا يوجد if داخل if</p>
switch	<pre>switch (expression) { case x: // code block break; case y: // code block break; default: // code block }</pre> <p>ملاحظة : يتم إدخال المتغير ضمن switch ثم يتم التحقق من قيمة المتغير إذا تطابقة لأحد case يتم تنفيذ الجملة و إذا لم تكن متطابقة يتم تنفيذ default وهي لا تحتاج إلى break فوظيفة break توقيف ال switch فدوما سوف يأخذ أول case متطابقة ويمكن وضع أثنان case بمتغير مختلف لإظهار نفس النتيجة</p> <pre>switch (new Date().getDay()) { case 6: text = "Today is Saturday"; break; case 0: case 2: text = "Today is Sunday end Monday"; break; default: text = "Looking forward to the Weekend"; }</pre>



الرمز	اسم الوظيفة
for	<pre>for (statement 1; statement 2; statement 3) { // code block to be executed }</pre> <pre>for (let i = 0; i < 5; i++) { text += "The number is " + i + "
"; }</pre> <p>عند تعريف المتغير داخل for على شكل let فإن المتغير وقيمه ستبقى داخل ال for تستخدم لطباعة أو التعديل على مصفوفة أو كائن</p>
for in	<pre>for (variable in array) { code } ///// const numbers = [45, 4, 9, 16, 25]; let txt = ""; for (let x in numbers) { txt += numbers[x] + "
"; }</pre>

	<pre>document.getElementById("demo").innerHTML = txt; ////////// const person = {fname:"John", lname:"Doe", age:25}; let txt = ""; for (let x in person) { txt += person[x] + " "; } document.getElementById("demo").innerHTML = txt;</pre> <p>ملاحظة :</p> <ol style="list-style-type: none"> ١. يتم توقف الحلقة عند نفاذ عدد العناصر المصفوفة أو الكائن ٢. لا يمكن فرض قيمة لمتغير الحلقة ٣. تستخدم عندما لا يكون الترتيب مهما لعناصر المصفوفة أو الكائن
Array.forEach()	<p>و هي حلقة خاصة بالمصفوفة حيث تستدعي تابع وتقوم بتكراره حسب عدد عناصر المصفوفة مع العلم أن عند استدعاء التابع يتم الإرسال إليه ثلاث قيم وهي :</p> <ol style="list-style-type: none"> ١. قيم المصفوفة ٢. فهرس المصفوفة ٣. أسم المصفوفة <pre>const numbers = [45, 4, 9, 16, 25]; let txt = ""; numbers.forEach(myFunction); document.getElementById("demo").innerHTML = txt; function myFunction(value, index, array) { txt += value + "
"; } //////////</pre>
for Of	<p>تستخدم لطباعة أو التعديل على الكائنات القابلة لتكرار مثل Arrays, Strings, Maps, NodeLists, and more</p> <pre>const cars = ["BMW", "Volvo", "Mini"]; let text = ""; for (let x of cars) { text += x + "
"; } document.getElementById("demo").innerHTML = text; ///////// let language = "JavaScript"; let text = ""; for (let x of language) { text += x + "
"; } document.getElementById("demo").innerHTML = text; //////////////////// const fruits = new Map([["apples", 500], ["bananas", 300], ["oranges", 200]]); for (const x of fruits) { // code block to be executed }</pre>
while	<p>وهي حلقة مثل for ولكن الاختلاف أنه يتم تعريف المتغير قبل while والشرط بين قوسين while و القيمة المؤثرة على المتغير داخل while</p> <pre>while (condition) { // code block to be executed }</pre>

	<pre> let text = ""; let i = 0; while (i < 10) { text += "
The number is " + i; i++; } document.getElementById("demo").innerHTML = text; </pre> <p>ملاحظة : إذا نسيتم زيادة المتغير المستخدم في الشرط ، فلن تنتهي الحلقة أبدًا. سيؤدي هذا إلى تعطل متصفحك</p>
do while	<p>نفس مبدأ while لكن يتم تنفيذ المحتوى مرة واحدة على الأقل</p> <pre> do { // code block to be executed } while (condition); ////////// let text = ""; let i = 0; do { text += "
The number is " + i; i++; } while (i < 10); document.getElementById("demo").innerHTML = text; </pre> <p>ملاحظة : يمكن استخدام for مثل while</p> <pre> const cars = ["BMW", "Volvo", "Saab", "Ford"]; let i = 0; let text = ""; for (;cars[i];) { text += cars[i] + "
"; i++; } document.getElementById("demo").innerHTML = text; </pre> <p>حيث هنا يتم تنفيذ الحلقة لتنفيذ عناصر المصفوفة</p> <pre> const cars = ["BMW", "Volvo", "Saab", "Ford"]; let i = 0; let text = ""; while (cars[i]) { text += cars[i] + "
"; i++; } document.getElementById("demo").innerHTML = text; </pre>
break	<p>تقوم بإيقاف الحلقة نهائياً ع الأغلب يتم وضعها داخل شرط</p> <pre> let text = ""; for (let i = 0; i < 10; i++) { if (i === 3) { break; } text += "The number is " + i + "
"; } document.getElementById("demo").innerHTML = text; </pre>
continue	<p>تقوم بإيقاف الحلقة لمرة واحدة ع الأغلب يتم وضعها داخل شرط</p> <pre> let text = ""; for (let i = 0; i < 10; i++) { if (i === 3) { continue; } text += "The number is " + i + "
"; } document.getElementById("demo").innerHTML = text; </pre>

lable	<p>وهو عندما يكون لدينا حلقتين داخل بعض و نريد إيقاف وحدة دون الأخرى نستخدم lable و ذلك من خلال تسمية الفور وإيقافها عن طريق البرك أو كونتينيو</p> <pre> mainloop : for () { nestedloop for () { if () { break mainloop; } } } </pre>
-------	---

sets

الرمز	اسم الوظيفة
new Set()	لإنشاء مجموعة جديدة بداخلها مصفوفة أو كائن أو نص <pre>// Create a Set const letters = new Set(["a", "b", "c"]);</pre>
size	لمعرفة عدد العناصر داخل المصفوفة أو الكائن <pre>// Create a Set const letters = new Set(["a", "b", "c"]); // Display set.size document.getElementById("demo").innerHTML = letters.size;</pre>
add()	وظيفته إضافة القيم أو المتغيرات للمجموعة <pre>// Create a Set const letters = new Set(); // Add Values to the Set letters.add("a"); letters.add("b"); letters.add("c"); // Display set.size document.getElementById("demo").innerHTML = letters.size; ////////// // Create a new Set const letters = new Set(["a", "b", "c"]); // Add a new Element letters.add("d"); letters.add("e"); // Display set.size document.getElementById("demo").innerHTML = letters.size; ////////// // Create a Set const letters = new Set(); // Create Variables const a = "a"; const b = "b"; const c = "c"; // Add the Variables to the Set letters.add(a); letters.add(b); letters.add(c); // Display set.size document.getElementById("demo").innerHTML = letters.size; // Create a Set const letters = new Set(); // Add values to the Set letters.add("a"); letters.add("b"); letters.add("c");</pre> <p>ملاحظة : إذا أضفت عناصر متساوية ، فسيتم حفظ العنصر الأول فقط وتجاهل الجديد</p>

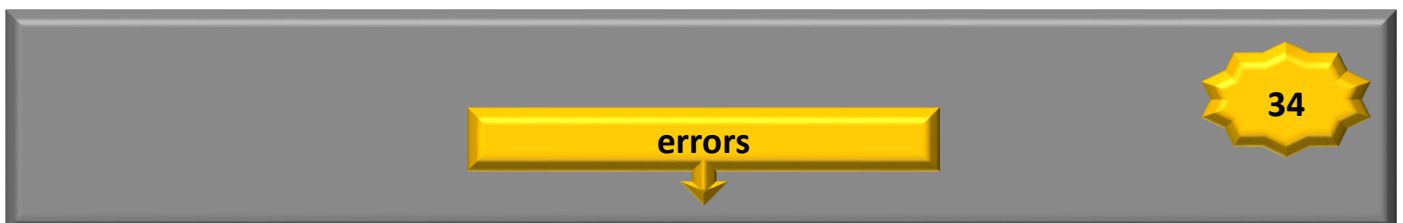
	<pre> letters.add("c"); letters.add("c"); letters.add("c"); letters.add("c"); letters.add("c"); // Display set.size document.getElementById("demo").innerHTML = letters.size; </pre> <p>سيكون عدد العناصر فقط 3</p>
forEach()	<p>تستخدم لتعديل أو طباعة المجموعة</p> <pre> // Create a Set const letters = new Set(["a", "b", "c"]); // List all Elements let text = ""; letters.forEach (function (value) { text += value + "
"; }) document.getElementById("demo").innerHTML = text; </pre>
values()	<p>يقوم التابع بإرجاع كائن مكرر جديد يحتوي على جميع القيم في مجموعة :</p> <pre> // Create a Set const letters = new Set(["a", "b", "c"]); // List all Elements let text = ""; for (const x of letters.values()) { text += x + "
"; } document.getElementById("demo").innerHTML = text; </pre>



الرمز	اسم الوظيفة
new Map()	<p>يقوم بإنشاء خريطة جديدة والخريطة عبارة عن مصفوفة بداخلها مفتاح و قيمة</p> <pre> // Create a Map const fruits = new Map([["apples", 500], ["bananas", 300], ["oranges", 200]]); </pre>
set()	<p>وظيفتها إضافة عناصر إلى الخريطة</p> <pre> // Create a Map const fruits = new Map(); // Set Map Values fruits.set("apples", 500); fruits.set("bananas", 300); fruits.set("oranges", 200); ////////// // Create a Map const fruits = new Map([</pre>

	<pre>["apples", 500], ["bananas", 300], ["oranges", 200]]); fruits.set("apples", 200);</pre>
get()	<p>وظيفته إعطاء القيمة عندما تعطيه المفتاح</p> <pre>// Create a Map const fruits = new Map([["apples", 500], ["bananas", 300], ["oranges", 200]]); document.getElementById("demo").innerHTML = fruits.get("apples");//return 500</pre>
size	<p>يقوم بإرجاع عدد العناصر في الخريطة</p> <pre>// Create a Map const fruits = new Map([["apples", 500], ["bananas", 300], ["oranges", 200]]); document.getElementById("demo").innerHTML = fruits.size;</pre>
delete()	<p>يقوم بحذف العنصر عندما تعطيه المفتاح</p> <pre>// Create a Map const fruits = new Map([["apples", 500], ["bananas", 300], ["oranges", 200]]); // Delete an Element fruits.delete("apples"); document.getElementById("demo").innerHTML = fruits.size;</pre>
has()	<p>يقوم التابع بالتأكد إذا كان العنصر موجود في الخريطة من خلا المفتاح فإذا كان موجود يرجع true و إذا كان غير موجود يرجع false</p> <pre>// Create a Map const fruits = new Map([["apples", 500], ["bananas", 300], ["oranges", 200]]); document.getElementById("demo").innerHTML = fruits.has("apples");//return true //////////////////// const fruits = new Map([["apples", 500], ["bananas", 300], ["oranges", 200]]); // Delete an Element fruits.delete("apples"); document.getElementById("demo").innerHTML = fruits.has("apples");//return false</pre>
forEach()	<p>وظيفتها طباعة أو التعديل على المفتاح أو القيمة للعناصر في الخريطة في تعطي الإمكانية لتحكم بالمفتاح أو القيمة</p>

	<pre>// Create a Map const fruits = new Map([["apples", 500], ["bananas", 300], ["oranges", 200]]); let text = ""; fruits.forEach (function(value, key) { text += key + ' = ' + value + "
" }) document.getElementById("demo").innerHTML = text;</pre>
entries()	<p>وظيفته هي عدم إرجاع العناصر التي تمتلك نفس المفتاح والعناصر التي تكرر فيها المفتاح ستكون قيمتها قيمة المفتاح الأخير المكرر</p> <pre>// Create a Map const fruits = new Map([["apples", 500], ["bananas", 300], ["oranges", 200], ["bananas", 900]]); let text = ""; for (const x of fruits.entries()) { text += x + "
"; } document.getElementById("demo").innerHTML = text;</pre>



الرمز	اسم الوظيفة
try	<p>تسمح لك العبارة بتحديد كتلة من التعليمات البرمجية ليتم اختبارها بحثاً عن الأخطاء أثناء تنفيذها .</p> <pre>try { Block of code to try }</pre>
throw	<p>يسمح لك بإنشاء نص أو رقم لخطأ مخصص داخل try يتم إظهاره لمستخدم عن طريق المتغير داخل catch</p> <pre>throw "Too big"; // throw a text throw 500; // throw a number</pre>
catch	<p>يسمح لك بتحديد كتلة من التعليمات البرمجية ليتم تنفيذها ، في حالة حدوث خطأ في كتلة try و يكون داخله متغير err يكون مربوط مع throw</p> <pre>try { Block of code to try } catch(err) { Block of code to handle errors } ////////// <p>Please input a number between 5 and 10:</p></pre>

	<pre> <input id="demo" type="text"> <button type="button" onclick="myFunction()">Test Input</button> <p id="p01"></p> <script> function myFunction() { const message = document.getElementById("p01"); message.innerHTML = ""; let x = document.getElementById("demo").value; try { if(x == "") throw "empty"; if(isNaN(x)) throw "not a number"; x = Number(x); if(x < 5) throw "too low"; if(x > 10) throw "too high"; } catch(err) { message.innerHTML = "Input is " + err; } } </script> </pre>
finally	<p>يتيح لك تنفيذ التعليمات البرمجية ، بعد إجراء الاختبار سواء كان هناك خطأ أو لا يوجد هناك خطأ .</p> <pre> try { Block of code to try } catch(err) { Block of code to handle errors } finally { Block of code to be executed regardless of the try / catch result } ////////// <p>Please input a number between 5 and 10:</p> <input id="demo" type="text"> <button type="button" onclick="myFunction()">Test Input</button> <p id="p01"></p> <script> function myFunction() { const message = document.getElementById("p01"); message.innerHTML = ""; let x = document.getElementById("demo").value; try { if(x == "") throw "is empty"; if(isNaN(x)) throw "is not a number"; x = Number(x); if(x > 10) throw "is too high"; if(x < 5) throw "is too low"; } catch(err) { message.innerHTML = "Input " + err; } finally { </pre>

	<pre>document.getElementById("demo").value = ""; } } </script></pre>
The Error Object	<p>وهي كائنات مخصصة لأخطاء محددة و توفر كائنات الخطأ خاصيتين :</p> <ol style="list-style-type: none"> ١. أسم الخطأ : يقوم بتعيين و إرجاع أسم الخطأ ٢. رسالة الخطأ : يقوم بتعيين و إرجاع رسالة الخطأ <p>ولكن لهذه الكائنات أنواع عديدة :</p> <p>eval() : يشير إلى خطأ في الوظيفة</p> <pre><p id="demo"></p> <script> try { eval("alert('Hello')"); } catch(err) { document.getElementById("demo").innerHTML = err.name; } </script> //////////</pre> <p>RangeError : يشير إلى الخطأ إذا كنت تستخدم رقم خارج النطاق</p> <pre><p id="demo"> <script> let num = 1; try { num.toPrecision(500); } catch(err) { document.getElementById("demo").innerHTML = err.name; } </script> //////////</pre> <p>ReferenceError : يشير إلى إذا كنت تستخدم متغير لم يتم التصريح عنه</p> <pre><p id="demo"></p> <script> let x = 5; try { x = y + 1; } catch(err) { document.getElementById("demo").innerHTML = err.name; } </script> //////////</pre> <p>SyntaxError : يستخدم لتقييم الجمل البرمجية إذا بها خطأ برمجي في الصياغة</p> <pre><p id="demo"></p> <script> try { eval("alert('Hello')"); } catch(err) { document.getElementById("demo").innerHTML = err.name; }</pre>

TypeError : يشير أن نوع المتغير مخالف لنوع المتوقع

```
}
</script>
//////////
```

```
<p id="demo"></p>
```

```
<script>
let num = 1;
try {
  num.toUpperCase();
}
catch(err) {
  document.getElementById("demo").innerHTML = err.name;
}
</script>
//////////
```

URIError : يستخدم لمعرفة إذا كنت تستخدم رابط فيه أحرف غير قانونية

```
<p id="demo"></p>
```

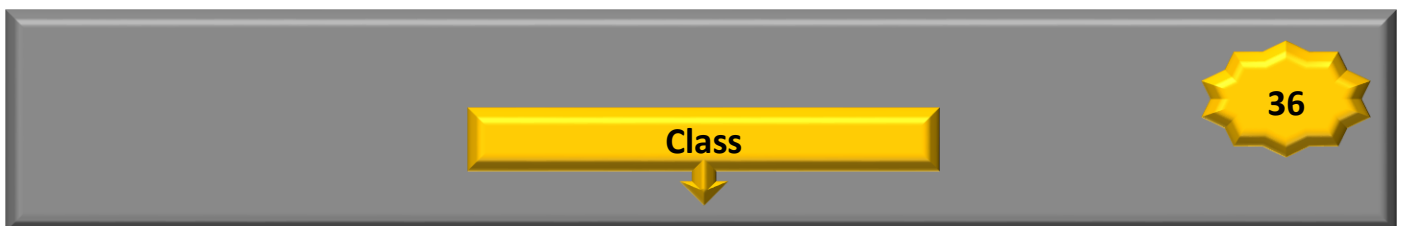
```
<script>
try {
  decodeURI("%%%");
}
catch(err) {
  document.getElementById("demo").innerHTML = err.name;
}
</script>
```

35

this Keyword

الرمز	اسم الوظيفة
this	و هي كلمة أساسية تشير إلى الكائن الذي تنتمي إليه ومن خلالها يمكننا التحكم بعناصر الكائن نفسه و لها قيم مختلفة حسب مكان استخدامها :
this in a Method	وهو تابع في الكائن this هنا تشير إلى عناصر الكائن <pre>// Create an object: const person = { firstName: "John", lastName: "Doe", id: 5566, fullName : function() { return this.firstName + " " + this.lastName; } }; // Display data from the object: document.getElementById("demo").innerHTML = person.fullName();</pre>
this Alone	عند استخدامه في ملف الجافا سكريبت الرئيسي فإنه يشير إلى كائن الويندوز [object Window] <pre>let x = this; document.getElementById("demo").innerHTML = x;</pre>
this in a Function	تكون this تشير إلى التابع نفسه

	<pre>document.getElementById("demo").innerHTML = myFunction(); function myFunction() { return this; }</pre>
this in Event Handlers	<p>في أحداث الدوم تشيير إلى العنصر نفسه</p> <pre><button onclick="this.style.display='none'">Click to Remove Me!</button></pre> <p>أو</p> <pre><button id="demo">Click to Remove Me!</button> document.getElementById("demo").onclick = function() {return this;}; // الناتج <button id="demo">Click to Remove Me!</button></pre>
Explicit Function Binding	<p>call() and apply() و هي توابع جاهزة نقوم بربط عناصر كائن بكائن آخر</p> <pre>const person1 = { fullName: function() { return this.firstName + " " + this.lastName; } } const person2 = { firstName: "John", lastName: "Doe", } let x = person1.fullName.call(person2); document.getElementById("demo").innerHTML = x;</pre>



الرمز	اسم الوظيفة
class	<p>class : هي دالة وليست كائن تستخدم الكلمة <code>class Name { }</code> لإنشائها وبداخلها دوما الدالة <code>constructor() { }</code> و دوال أخرى يتم إنشائها لتقوم بعمليات حسابية</p> <p>Syntax</p> <pre>class ClassName { constructor() { ... } }</pre> <p>ملاحظة ١ : تستخدم الكلاسات لإنشاء كائنات متعددة بقيم مختلفة من كلاس واحد وأسم الكلاس يفضل أن يبدأ بحرف كبير ملاحظة ٢ : يجب عليك تعريف الكلاس قبل استخدامه على عكس الدوال مثال ١</p> <pre>class Car { constructor(name, year) { this.name = name; // مفتاح أول مع قيمته this.year = year; // مفتاح ثاني مع قيمته } }</pre> <pre>const myCar1 = new Car("Ford", 2014); const myCar2 = new Car("Audi", 2019); document.getElementById("demo").innerHTML = myCar1.name + " " + myCar1.year + " / " + myCar2.name + " " + myCar2.year; >> Ford 2014 / Audi 2019</pre> <p>ملاحظات عن دالة المشى <code>constructor() { ... }</code> : ١ . يجب تسميتها بهذا الاسم دوما <code>constructor</code></p>

	<p>٢. يجب إنشائها دوما داخل الكلاس وإذا لم يتم إنشائها ستقوم الجافا سكربت تلقائيا بإنشاء واحدة فارغة</p> <p>٣. عند إنشاء الكائن نستدعي أسم الكلاس كدالة وهو بدوره يستدعي دالة المنشئ <code>constructor()</code></p> <p>٤. نستخدم دالة المنشئ <code>constructor()</code> لإنشاء مفاتيح الكائن و إعطائها القيم لهذه المفاتيح</p> <p>ملاحظة : يمكننا إنشاء داخل الكلاس دوال أخرى حيث تكون أسماء الدوال هي مفاتيح للكائن وترجع هذه الدوال قيم هذه المفاتيح ويتم إنشائها لنقوم بعمليات حسابية معقدة وإرجاع القيم</p> <p>مثال ٢</p> <pre> class Car { constructor(name, year) { this.name = name; // مفتاح أول مع قيمته this.year = year; // مفتاح ثاني مع قيمته } age() { // مفتاح ثالث مع قيمته let date = new Date(); return date.getFullYear() - this.year; } // أيج هو أسم المفتاح الكائن وتعيد ريتورن قيمة هذا المفتاح } let myCar = new Car("Ford", 2010); document.getElementById("demo").innerHTML = "My car is " + myCar.age() + " years old."; >> My car is 12 years old. </pre> <p>ملاحظة : يمكننا الوصول إلى قيم المفاتيح الأخرى من خلال <code>this.kay = value</code> بكتابة سيز مع اسم المفتاح نحصل على قيمة المفتاح</p> <p>مثال ٣ يمكننا تمرير القيم لدالة المنشأة</p> <pre> class Car { constructor(name, year) { this.name = name; this.year = year; } age(x) { return x - this.year; } } let date = new Date(); let year = date.getFullYear(); let myCar = new Car("Ford", 2010); document.getElementById("demo").innerHTML= "My car is " + myCar.age(year) + " years old.";>> My car is 12 years old. </pre>
Class Inheritance	<p>يمكننا توريث كلاس لكلاس أخرى باستخدام <code>extends</code> حيث ترث المفاتيح و القيم الكلاس الآخر</p> <p>ثم نستخدم <code>super(value)</code> داخل <code>constructor()</code> الكلاس الأصلية لإرسال القيمة إلى الكلاس الذي نرث منه</p> <p>مثال ١</p> <pre> class Car { constructor(brand,brand2) { this.carname = brand; // مفتاح أول مع قيمته this.carname2 = brand2; // مفتاح ثاني مع قيمته } present() { // مفتاح ثالث مع قيمته return 'I have a ' + this.carname + " / " + this.carname2; } } class Model extends Car { // نرث الكلاس كار مع مفاتيحه و قيمه constructor(brand, brand2, mod) { super(brand,brand2); // نرسل القيمتين لدالة المنشأة في كلاس كار لإنشاء المفاتيح الأول و الثاني } } </pre>

	<pre> this.model = mod; // مفتاح رابع مع قيمته } show() { // مفتاح الخامس مع قيمته return this.present() + ' , it is a ' + this.model; } } let myCar = new Model("Ford","Ford2", "Mustang"); document.getElementById("demo").innerHTML = myCar.show() + " / " + myCar.carname + " + " + myCar.carname2 ;>> I have a Ford / Ford2 , it is a Mustang / Ford + Ford2 </pre> <p>ملاحظة : <code>super(value)</code> في الكلاس الأصلي يمرر القيم إلى <code>constructor()</code> { } الكلاس المورث ويأخذ <code>super(value)</code> قيمه من <code>constructor(value) { super(value); }</code> الدالة المنشأة في الكلاس الأصلي</p>
Getters and Setters	<p><code>objectName.kay</code> : تستخدم للوصول إلى قيمة مفتاح و الاستدعاء يكون هكذا <code>objectName.kay</code> <code>objectName.kay = x</code> : تستخدم لتعديل قيمة مفتاح و الاستدعاء يكون هكذا ; مثال ١</p> <pre> class Car { constructor(brand) { this.carname = brand; } get cnam() { return this.carname; } set cnam(x) { this.carname = x; } } let myCar = new Car("Ford"); document.getElementById("demo").innerHTML = myCar.cnam;>> Ford </pre> <p>مثال ٢ نعدل قيم المفتاح</p> <pre> class Car { constructor(brand) { this.carname = brand; } get cnam() { return this.carname; } set cnam(x) { this.carname = x; } } let myCar = new Car("Ford"); myCar.cnam = "AMMAR"; // عدلنا قيمة المفتاح أولاً باستخدام سيت document.getElementById("demo").innerHTML = myCar.cnam;>> AMMAR // طبعنا القيمة عن طريق غيت </pre>
Static Methods	<p>Static Methods : وهي دالة مفتاح ثابت يتم إنشائها داخل الكلاس و لا يمكنها الوصول إلى قيم المفاتيح الأخرى داخل الكلاس وعند المحاولة الوصول سيرجع أسم الكلاس فقط مثال ١</p> <pre> class Car { constructor(name) { this.name = name; // value = Ford } static hello() { return "Hello" + " / " + this.name; } } </pre>

	<pre> } } let myCar = new Car("Ford"); document.getElementById("demo").innerHTML = Car.hello(); >> Hello / Car ملاحظة : لحل هذه المشكلة نمرر الكائن كله عند طلب المفتاح من الخارج مثال ٢ class Car { constructor(name) { this.name = name; } static hello(x) { return "Hello " + x.name; // لاحظ طريقة الكتابه لم نكتب سيز } } let myCar = new Car("Ford"); document.getElementById("demo").innerHTML = Car.hello(myCar); >> Hello Ford </pre>
--	--

عند إنشاء ال dom شاهد فيديو ٧٧

Dom

الرمز	اسم الوظيفة
intro with dom	<p>Dom : Document Object Model</p> <p>تستخدم الدوم (نموذج كائن لمستند) بشكل الأساسي للوصول إلى جميع عناصر المستند وتغييرها حيث عندما يتم تحميل صفحة الويب ينشئ المتصفح نموذج كائن للمستند Document Object Model في صفحة هذا الكائن Dom ينشئ شجرة كائنات للعناصر</p> <p>فهو يحدد معيارًا للوصول إلى المستندات</p> <p>الدوم هو المعيار الذي ينقسم إلى ٣ أجزاء مختلفة :</p> <ol style="list-style-type: none"> 1. Core DOM - standard model for all document types 2. XML DOM - standard model for XML documents 3. HTML DOM - standard model for HTML documents <p>الدوم في الجافا سكريبت تزودك بكل القوة لإنشاء صفحات html ديناميكية مثل :</p> <ol style="list-style-type: none"> ١. يمكنك تغيير جميع العناصر في الصفحة ٢. يمكنك تغيير جميع سمات العناصر في الصفحة ٣. يمكنك تغيير جميع أنماط css في الصفحة ٤. يمكنك إزالة عناصر و السمات الموجودة في الصفحة ٥. يمكنك إضافة عناصر و السمات إلى الصفحة ٦. يمكنك إنشاء أحداث للعناصر في الصفحة ٧. يمكنك التفاعل مع الأحداث العناصر في الصفحة
DOM Document	<p>Document : هو كائن يمثل صفحة الويب لذلك إذا أردت الوصول إلى أي عنصر داخل صفحة Html عليك البدء به مثل :</p> <p><code>document.getElementById(id)</code> <code>document.domain</code></p>
DOM Elements	<p>لتعديل على عناصر صفحة Html يجب أولاً الوصول إلى العناصر ويتم ذلك من خلال عدت طرق منها :</p> <ol style="list-style-type: none"> 1. Finding HTML elements by id 2. Finding HTML elements by tag name 3. Finding HTML elements by class name 4. Finding HTML elements by CSS selectors 5. Finding HTML elements by HTML object collections

١- الوصول للعنصر عن طريق id مثل

```
document.getElementById("idName")
```

٢- الوصول للعنصر عن طريق اسمه مثل

```
const element = document.getElementsByTagName("p");
```

ملاحظة : عندما يكون هناك أكثر من عنصر يتم الوصول إليهم حسب ترتيبهم في الصفحة مثل

```
element[0].innerHTML
```

ملاحظة : أو إذا أردنا الوصول لعناصر داخل عنصر مثل

```
const x = document.getElementById("main");
```

```
const y = x.getElementsByTagName("p");
```

٣- الوصول للعنصر عن طريق اسم الكلاس مثل

```
<p>Finding HTML Elements by Class Name.</p>
```

```
<p class="intro">Hello World!</p>
```

```
<p class="intro">This example demonstrates the <b>getElementsByClassName</b> method.</p>
```

```
<p id="demo"></p>
```

```
<script>
```

```
const x = document.getElementsByClassName("intro");
```

```
document.getElementById("demo").innerHTML =
```

```
'The first paragraph (index 0) with class="intro" is: ' + x[0].innerHTML;>> Hello World!
```

```
</script>
```

٤- الوصول للعنصر عن طريق اسم العنصر مع الكلاس أو اسم العنصر مع ايدي يعني مثل التسمية في css مثل

```
<h2>JavaScript HTML DOM</h2>
```

```
<p>Finding HTML Elements by Query Selector</p>
```

```
<p class="intro">Hello World!</p>
```

```
<p id="intro">This example demonstrates the <b>querySelectorAll</b> method.</p>
```

```
<p id="demo"></p>
```

```
<script>
```

```
const x = document.querySelectorAll("p.intro");
```

```
const x2 = document.querySelectorAll("p#intro");
```

```
document.getElementById("demo").innerHTML =
```

```
x[0].innerHTML + ' || ' + x2[0].innerHTML;
```

```
</script>
```

```
>>Hello World!. || This example demonstrates the querySelectorAll method.
```

٥- الوصول إلى عنصر الفورم

```
<form id="frm1" action="/action_page.php">
```

```
First name: <input type="text" name="fname" value="Ammar"><br>
```

```
Last name: <input type="text" name="lname" value="Qassab"><br><br>
```

```
<input type="submit" value="Submit">
```

```
</form>
```

```
<p>These are the values of each element in the form:</p>
```

```
<p id="demo"></p>
```

```
<script>
```

```
const x = document.forms["frm1"];
```

```
let text = "";
```

```
let text1 = 0;
```

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

```
text += x.elements[i].value + " / ";
```

```
text1+=1;
```

```
}
```


	<pre>document.getElementById("demo").innerHTML = text + text1;</pre> <p></script> >> Ammar / Qassab / Submit / 3</p> <p>ملاحظة : هناك عدة طرق أخرى للوصول لمحتويات الصفحة html :</p> <ol style="list-style-type: none"> 1. document.anchors 2. document.body 3. document.documentElement 4. document.embeds 5. document.forms 6. document.head 7. document.images 8. document.links 9. document.scripts 10. document.title
DOM Changing	<p>أسهل طريقة لتغيير محتوى عنصر Html هو استخدام innerHTML مثل</p> <pre>document.getElementById(id).innerHTML = new HTML ;</pre> <p>مثال</p> <pre><p id="p1">Hello World!</p></pre> <pre><script> document.getElementById("p1").innerHTML = "New text!"; >> New text! </script></pre> <p>لتغيير قيم صفات العناصر Html من خلال كتابة الخاصية نفسها مثل</p> <pre>document.getElementById(id).attribute = new value</pre> <p>مثال لتغيير عنوان الصورة</p> <pre></pre> <pre><script> document.getElementById("image").src = "landscape.jpg"; </script></pre> <p>تغيير المحتوى ديناميكي من خلال الدالة</p> <pre>setInterval(() => {}, 1000);</pre> <pre>function tick() { document.getElementById("demo").innerHTML = "Date : " + Date(); } tick(); setInterval(tick, 1000);</pre>
Forms	<p>التحقق من عناصر الإدخال قبل الإرسال نستخدم</p> <p>مثال</p> <pre><!DOCTYPE html> <html> <head> <script> function validateForm() { let x = document.forms["myForm"]["fname"].value; if (x == "") { alert("Name must be filled out"); return false; } } </script> </head> <body> <form name="myForm" action="/action_page.php" onsubmit="validateForm()" method="post"> Name: <input type="text" name="fname" onclick="validateForm()"> <input type="submit" value="Submit"> </form></pre>

	<pre> </body> </html> </pre> <p>أو فينا نستعمل قيم العناصر الـ HTML مثل pattern type min max required disabled</p> <p>مثال</p> <pre> <form action="/action_page.php" method="post"> <input type="text" name="fname" required> <input type="submit" value="Submit"> </form> </pre>
CSS	<p>لتغيير الستايل لعنصر ما نتبع التالي</p> <p>مثال</p> <pre> document.getElementById(id).style.property = new style </pre> <pre> <p id="p1">Hello World!</p> <p id="p2">Hello World!</p> <script> document.getElementById("p2").style.color = "blue"; document.getElementById("p2").style.fontFamily = "Arial"; document.getElementById("p2").style.fontSize = "larger"; </script> </pre> <p>يمكننا استعمال الأحداث أيضا</p> <pre> <button type="button" onclick="document.getElementById('id1').style.color = 'red'"> Click Me! </button> </pre>
Animation	<p>يمكننا فعل النيميشن عن طريق javascript من دون css</p> <p>مثال</p> <pre> <!DOCTYPE html> <html> <style> #container { width: 400px; height: 400px; position: relative; background: yellow; } #animate { width: 50px; height: 50px; position: absolute; background-color: red; } </style> <body> <p><button onclick="myMove()">Click Me</button></p> <div id="container"> <div id="animate"></div> </div> <script> function myMove() { let id = null; const elem = document.getElementById("animate"); let pos = 0; clearInterval(id); id = setInterval(frame, 5); function frame() { if (pos == 350) { clearInterval(id); </pre>

	<pre> } else { pos++; elem.style.top = pos + "px"; elem.style.left = pos + "px"; } } } </script> </body> </html> </pre>
Events	<p>عند وقوع حدث من قبل المستخدم يمكن لجافاسكريبت إجراء تغيير على العناصر من خلال :</p> <p><code>document.getElementById(id).onEvent = javascript</code></p> <p>من الأمثلة التي يمكن أن تفعلها الأحداث في الجافاسكريبت :</p> <ol style="list-style-type: none"> ١. عندما تم تحميل الصورة ٢. عندما يتحرك الماوس فوق عنصر ٣. عندما يتم تغيير حقل الإدخال ٤. عندما يتم تقديم نموذج HTML ٥. عندما يضغط المستخدم على مفتاح <p>يمكننا تفعيل الحدث بطريقتين :</p> <ol style="list-style-type: none"> ١. أما أحداث عناصر Html مع دالة تفعل عند تفعيل الحدث ٢. أو أحداث من داخل الجافاسكريبت <p>onclick : و حدث يفعل عند النقر على العنصر</p> <p>مثال ١</p> <pre> <button onclick="displayDate()">The time is?</button> <p id="demo"></p> <script> function displayDate() { document.getElementById("demo").innerHTML = Date(); } </script> </pre> <p>مثال ٢</p> <pre> <button id="myBtn">Try it</button> <p id="demo"></p> <script> document.getElementById("myBtn").onclick = displayDate; function displayDate() { document.getElementById("demo").innerHTML = Date(); } </script> </pre> <p>onload : هو حدث يفعل عند تحميل كامل العنصر مع محتوياته يوضع في body يستخدم لإيقاف شاشة التحميل</p> <p>onunload : هو حدث يفعل عند لا يتم تحميل كامل العنصر مع محتوياته</p> <p>مثال</p> <pre> <!DOCTYPE html> <html> <body onload="checkCookies()"> <h2>JavaScript HTML Events</h2> <p id="demo"></p> <script> function checkCookies() { var text = ""; if (navigator.cookieEnabled == true) { text = "Cookies are enabled."; } } </pre>

```

} else {
    text = "Cookies are not enabled.";
}
document.getElementById("demo").innerHTML = text;
}
</script>
</body>
</html>

```

onchange : غالبا ما يستخدم في حقول الإدخال حيث يفعل الحدث عندما يقوم المستخدم بإدخال المعلومات و الخروج من حقل الإدخال مثال

JavaScript HTML Events

Enter your name:

```

<script>
function upperCase() {
    const x = document.getElementById("fname");
    x.value = x.value.toUpperCase();
}
</script>

```

onmouseover : يفعل الحدث عندما تهوفر بالماوث فوق العنصر
onmouseout : يفعل الحدث عندما لا تهوفر بالماوث فوق العنصر
 مثال

```

<div onmouseover="mOver(this)" onmouseout="mOut(this)"
style="background-color:#D94A38;width:120px;height:20px;padding:40px;">
Mouse Over Me</div>

```

```

<script>
function mOver(obj) {
    obj.innerHTML = "Thank You"
}

function mOut(obj) {
    obj.innerHTML = "Mouse Over Me"
}
</script>

```

onmousedown : يفعل الحدث عند النقر على العنصر بالماوث ويبقى مفعّل مع استمرار بالنقر
onmouseup : يفعل الحدث عند النقر على العنصر بالماوث ثم إيقاف النقر
 مثال

```

<div onmousedown="mDown(this)" onmouseup="mUp(this)"
style="background-color:#D94A38;width:90px;height:20px;padding:40px;">
Click Me
</div>

```

```

<script>
function mDown(obj) {
    obj.style.backgroundColor = "#1ec5e5";
    obj.innerHTML = "Release Me";
}

function mUp(obj) {
    obj.style.backgroundColor="#D94A38";
    obj.innerHTML="Thank You";
}
</script>

```

Event Listener

قائمة بالأحداث المهمة : onevent

١. Onabort : يقع الحدث عندما يتم إلغاء تحميل الوسائط (يوضع في video)
٢. onafterprint : يقع الحدث عند بدء طباعة الصفحة أو إذا تم إغلاق مربع حوار الطباعة (يوضع في body)
٣. animationend
٤. animationiteration
٥. animationstart
٦. beforeprint
٧. beforeunload
٨. blur
٩. canplay
١٠. canplaythrough
١١. change
١٢. click
١٣. contextmenu
١٤. copy
١٥. cut
١٦. dblclick
١٧. drag
١٨. dragend
١٩. dragenter
٢٠. dragleave
٢١. dragover
٢٢. dragstart
٢٣. drop
٢٤. durationchange
٢٥. ended
٢٦. error
٢٧. focus
٢٨. focusin
٢٩. focusout
٣٠. fullscreenchange
٣١. fullscreenerror
٣٢. hashchange
٣٣. input
٣٤. invalid
٣٥. keydown
٣٦. keypress
٣٧. keyup
٣٨. load
٣٩. loadeddata
٤٠. loadedmetadata
٤١. loadstart
٤٢. message
٤٣. mousedown
٤٤. mouseenter
٤٥. mouseleave
٤٦. mousemove
٤٧. mouseover
٤٨. mouseout
٤٩. mouseup
٥٠. offline
٥١. online
٥٢. open
٥٣. pagehide
٥٤. pageshow
٥٥. paste

pause .٥٦
 play .٥٧
 playing .٥٨
 progress .٥٩
 ratechange .٦٠
 resize .٦١
 reset .٦٢
 scroll .٦٣
 search .٦٤
 seeked .٦٥
 seeking .٦٦
 select .٦٧
 show .٦٨
 stalled .٦٩
 submit .٧٠
 suspend .٧١
 timeupdate .٧٢
 toggle .٧٣
 touchcancel .٧٤
 touchend .٧٥
 touchmove .٧٦
 touchstart .٧٧
 transitionend .٧٨
 unload .٧٩
 volumechange .٨٠
 waiting .٨١
 wheel .٨٢

هون

altKey
 altKey
 animationName
 bubbles
 button
 buttons
 cancelable
 charCode
 clientX
 clientY
 code
 ctrlKey
 ctrlKey
 currentTarget
 data
 data
 defaultPrevented
 deltaX
 deltaY
 deltaZ
 deltaMode
 detail
 elapsedTime
 elapsedTime
 eventPhase
 ()getModifierState

	<inputtype </inputtype isTrusted key keyCode location metaKey metaKey newURL oldURL pageX pageY persisted ()preventDefault propertyName relatedTarget relatedTarget screenX screenY shiftKey shiftKey ()stopImmediatePropagation ()stopPropagation target targetTouches timeStamp touches type which which view
--	--

Bom

38

الرمز	اسم الوظيفة