**JAVASCRIPT**

**الوظيفة**

**1**

|  |  |
| --- | --- |
| **اسم الوظيفة** | **الرمز** |
|  |  |
|  |  |

**ملاحظة:**

**introdaction**

**1**

1. **javascript هي لغة برمجية او سكربت (scripting or programming language)**
2. **تعمل على client side (browser) and server side**
3. **تعمل على جعل الموقع تفاعلي و ديناميكي**
4. **يمكن التلاعب في css and html**
5. **يمكن تحريك محتوى الموقع**
6. **يمكن التلاعب في قواعد البيانات والتشيك على معطات المستخدم**
7. **يمكن إنشاء ألعاب وتطبيقات موبايل وديسكتوب**
8. **متطلبات هي التطبيق الجيد على css and html**

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

**ملاحظة: أضافات 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 أخر الباضي**

**ملاحظة: يتمم استخدام الكونسول الذي في كروم للإختبارات مثل الكائن window الذي له أوامر**

**مكان وضع السكربت**

**2**

1. **ملف الجافاسكربت أما خارجي أو داخلي**
2. **يتم وضع وسم <script> أما في أخر الهيد أو في أخر الباضي**
3. **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**
4. **لذلك نقوم بوضع وسم السكربت في أخر الباضي أو نقوم بتحميل الصفحة ثم تشغيل الأكواد وذلك عن طريق الكود**
5. **window.onload = function () {} نقوم بكتابته إذا كان السكربت خارجي أو في الهيد**

**comments**

**3**

|  |  |
| --- | --- |
| **اسم الوظيفة** | **الرمز** |
| كومينت سطري | // |
| كومينت متعدد الأسطر | /\* \*/ |
| وضع الملاحظات عن الأكواد أو تهميش الكود لإيقاف عمله | وظيفة الكومينت |
| ctrl + / | إختصار التهميش |

**output**

**4**

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

**طريقة كتابة الجمل و المعايير**

**5**

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

**Ecmascript**

**6**

1. **وهي منظمة عالمية تقوم بوضع معايير لمعلومات لل javascript**
2. **es6 وهو إصدار لجافاسكربت تم إصداره سنة 2015 وفيه تغيرات طفيفة وميزات كثيرة على الأكواد**
3. **es6-features وهو موقع يقوم يعرض الفرق بين الإصدارات السابقة والإصدار es6** 
   1. [**http://es6-features.org/#Constants**](http://es6-features.org/#Constants)
4. **الأصدارات الحديثة من المتصفحات هي فقط تدعم التحديثات الجديدة لجافا سكربت لذلك لحل هذا المشكلة نستخدم babel**
   1. [**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**](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&p)**=**

**Data types**

**7**

|  |  |
| --- | --- |
| **اسم الوظيفة** | **الرمز** |
| وهو فكشن لتحديد نوع البيانات حيث نكتب البيانات داخله أو بعده بفراغ  consle.log(typeof("ammar")); | 'typeof()  typeof فراغ |
| نوع نصي string | "ammar" |
| نوع رقمي number | 12345 |
| نوع رقمي number | 500.99 |
| نوع مصفوفة object = array | [11,12,13]  ["a","a","a"] |
| نوع كائن object و يأتي على شكل مفتاح وبعده قيمة | [name:"",age:24] |
| نوع شرطي boolean | true = 1  false = 0 |
| نوع قيمة undefined فارغة | undefined |
| نوع قيمة null غير موجودة | null |
| نوع تابع أو دالة function | function myFunc(){} |

**variabels**

**8**

|  |  |
| --- | --- |
| **اسم الوظيفة** | **الرمز** |
| document.getElementById("").innerHTML = "";  id.innerHTML = ""; | لإستدعاء عنصر  عن طريق ايدي |
| 1. يمكن أن يحتوي على أحرف و أرقام و شريطة سفلية \_ و الدولار $ 2. يجب أن يبدأ بحرف أو شريطة سفلية أو دولار فقط 3. الأسماء حساسة لحالة الأحرف أن كانت كبيرة أو صغيرة أي الأسم ذو الحرف الكبير يختلف عن الأسم ذو الحرف الصغير مثل user User 4. لا يمكن استخدام الكلمات المحجوزة لتسمية المتغيرات 5. لايمكن أن يبدأ برقم 6. لا يمكن أن تحتوي على سبيشيل كركتر أو مسافات 7. أذا احتوى الأسم على مقطعين يجب أن يبدأ المقطع الثاني بحرف كبير مثل userName camelCase | قواعد تسمية المتغيير |
| var namevariabels = value , namevariabels2 = value2 ;  var namevariabels , namevariabels2;   1. لغة الجافاسكربت هي لغة loosely typed أي هي لغة لا تحتاج لتعرفها نوع المتغيير 2. يجب تعريف المتغير بعدها يتم إستخدامه ويمكن استخدامه قبل تعريفه 3. يمكن تعريف المتغيير أكثر من مرة ولا يفقد قيمته ألا أذا عدلنا عليه 4. عندما يتم استخدام المتغيير قبل تعريفه يظهر undefined ولا يمكن معرفة سبب المشكلة 5. عندما يتم إنشاء متغيير بواسطة var فأنه يظهر داخل الكائن window ويمكن استخدامه 6. يمكن استخدام المتغيير في كل ملف الجافا سكربت | var |
| 1. لا يمكن تعريف المتغيير أكثر من مرة لأن يفقد قيمته فيخرج error و يمكن معرفة سبب المشكلة ومكانها 2. عندما يتم استخدام المتغيير قبل تعريفه يظهر error و يمكن معرفة سبب المشكلة ومكانها 3. عندما يتم إنشاء متغيير بواسطة let فأنه لا يظهر داخل الكائن window و لا يمكن استخدامه 4. يمكن استخدام المتغيير في الكتلة الواحدة | let |
| 1. لا يمكن تعريف المتغيير أكثر من مرة لأن يفقد قيمته فيخرج error و يمكن معرفة سبب المشكلة ومكانها 2. عندما يتم استخدام المتغيير قبل تعريفه يظهر error و يمكن معرفة سبب المشكلة ومكانها 3. عندما يتم إنشاء متغيير بواسطة const فأنه لا يظهر داخل الكائن window و لا يمكن استخدامه 4. يمكن استخدام المتغيير في الكتلة الواحدة | const |

**string**

**9**

|  |  |
| --- | --- |
| **اسم الوظيفة** | **الرمز** |
| double quotes  single quotes | "ammar"  'ammar' |
| لإظهار سنغل كوتس نكتبها داخل دبل كوتس | "ammar  'qassab' " |
| لإظهار دبل كوتس نكتبها داخل سنغل كوتس | 'ammar  "qassab" ' |
| أو نستخدم escape \ وهي تعمل لتجاهل الخطأ في العنصر يلي بعده مباشرة لذلك نستخدم لكتابة الكوتس او لتقسيم السترينغ لعددة أسطر مثل  "ammar \"qassab\" "  'ammar \'qassab\' '  "ammar \  abd alqader \  qassab"  "ammar \\ qassab" | \ |
| سطر جديد | \n |
| مسافة إفقية | \t |
| var x = new string("ammar");  console.log(typeof x);  لا تنشئ النص ككائن لان يبطئ الكود ويمكن إرجاع نتائج غير متوقعة | تحويل النص إلى كائن |
| عندما نقارن نصين تقارن القيمة فقط  عندما نقارن نصين تقارن القيمة مع النوع | ==  === |
| يتم الربط بين نصين أو متغييرين من النصوص  يتحول الرقم إلى نص | " " + " "  10 + " "/" " + 10 |
| console.log(a+" "+b+"\n"+c); = console.log(`${a} ${b}  ${c}`);  نلاحظ أن في template literals (template strings) لكتابة فراغ بين السلسلتين يكفي كتابة مسطرة أما لكتابة سطر جديد يكفي فقط البدء بسطر جديد مثال قوي  let title = "hello";  let desc = "ammar qassab";  let page = `  <div class="card">  <div class="child">  <h2>${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 رياكت بدل من ذلك | ` ${متغيير} ` |
| تستخدم || لعرض قيمة بديلة في حال كان هناك متغيير فيه القيم التالية :   1. Null 2. Undefinde 3. 0 4. "" 5. false or true   let price = Null or Undefinde or 0 or "" or false or true ;  consol.log(`the price is ${price || 333}`); | logical or || |
| تستخدم ?? لعرض قيمة بديلة في حال كان هناك متغيير فيه القيم التالية :   1. Null 2. Undefinde   let price = Null or Undefinde ;  consol.log(`the price is ${price ?? 333}`); | Nullish coalescing operator ?? |

**string method**

**10**

|  |  |
| --- | --- |
| **اسم الوظيفة** | **الرمز** |
| var str = "ammar qassab"; | str |
| تقوم بإرجاع طول السلسة وعندما تكون السلسة فارغة يرجع القيمة 0 مثل str.length(); النتيجة 12 لا تبدأ من صفر | length() |
| لعرض جزء من السلسة و نحدد جزء العرض من نقطة البداية إلى نقطة النهاية و يبدء من ال 0 مثل str.slice(6,11);  و إذا رقم سالب يبدأ العد من نهاية السلسة مثل str.slice(-6,-1);  و إذا حذفت الجزء الثاني سيطبع من البداية إلى النهاية السلسة str.slice(-6);\ str.slice(6); | slice(start,end) |
| مثل ال slice() بكن لا تقبل المعامل السالبة | substring  (start,end) |
| مثل ال slice() لكن الجزء الثاني يحدد طول السلسة وينطبق عليها السالب أو حذف الجزء الثاني  str.substr(0,5);/ str.substr(-6,6);/ str.substr(6);/str.substr(-6); | substr(start,long) |
| تقوم ياستبدال السلسلة التي في اليسار بالسلسلة التي في اليمين مع العلم أن السلسة التي في اليسار يجب أن تكون هي نفسها سواء بالطول أو في حالة الأحرف إذا كانت صغيرة أو كبيرة وتقوم بتغيير أول سلسة 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 السلسة دفعة واحدة نستخدم / /g مثل str.replace(/a/g,"h"); | replace("","") |
| يقوم بتكرار السلسلة بعدد مرات حسب الرقم مثل str.repeat(2); | repeat(رقم) |
| لتحويل الأحرف لأحرف كبيرة مثل str.toUpperCase(); | toUpperCase() |
| لتحويل الأحرف لأحرف صغيرة مثل str.LowerCase(); | toLowerCase() |
| لضم سلسلة إلى سلسة أخرى مثل str = str.concat("Hello");/str = "ammar".concat(" qassab");  و يمكن ضم أكثر من سلسله لسلسلة واحدة مثل str = str.concat("Hello",str2); | concat("") |
| يزيل المسافة الفارغة من جانبي السلسة مثل str = " ammar qassab ";/str.trim(); | trim() |
| 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 | padStart(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 | padEnd(4,0) |
| يرجع حرف واحد من السلسة حسب الرقم و في حال كانت السلسة أصغر من الرقم يرجع undeFined | str[رقم] |
| يرجع حرف واحد من السلسة حسب الرقم و في حال كانت السلسة أصغر من الرقم يرجع قيمة سلسة فارغة مثل  str.charAt(0); a | charAt(رقم) |
| لتحويل النص إلى مصفوفة مع العلم نكتب داخل التابع النص الذي بين الأحرف سواء لايوجد شيء أو , | مثل  let text = "a/b/c/d"; const myArrya = text.split("/");  لاٍستدعاء نستدعي مثل المصفوفات myArrya[1]; والرقم يستخدم لتحديد عدد المحارف المطلوبة و كتابته تكون إختيارية | split("",رقم) |

**string search**

**11**

|  |  |
| --- | --- |
| **اسم الوظيفة** | **الرمز** |
| تقوم بالبحث بالسلسة على أول تطابق لكلمة البحث في السلسة وتقوم بإرجاع موقعها وإذا لايوجد تطابق تقوم بإرجاع قيمة -1 مثل str.indexof("qas"); النتيجة 7 العد من أول السلسة  تقبل رقم لتبدأ البحث منه من بداية السلسة مثل str.indexof("a",6); | indexof("",رقم) |
| تقوم بالبحث بالسلسة على أخر تطابق لكلمة البحث في السلسة وتقوم بإرجاع موقعها وإذا لايوجد تطابق تقوم بإرجاع قيمة -1 مثل str.lastindexof("qas"); النتيجة 7 العد من أول السلسة  تقبل رقم لتبدأ البحث منه من بنهاية السلسة مثل str.indexof("a",7); | lastindex("",رقم) |
| نفس indexof لكن لاتقبل رقم لبدأ السلسة  تقوم بالبحث بالسلسة على أول تطابق لكلمة البحث في السلسة وتقوم بإرجاع موقعها وإذا لايوجد تطابق تقوم بإرجاع قيمة -1 مثل str.search("qas"); النتيجة 7 العد من أول السلسة  لا تقبل رقم لتبدأ البحث منه من بداية السلسة  وهي حساسة لحالة الأحرف وبتالي البحث قد يفشل لذلك نستخدم /string/i | search("") |
| 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 true أما في حال كان التركيب غير متطابق يرجع false  مثل str.includes("ammmar"); النتيجة false مثل str.includes("qassa"); النتيجة true | includes("") |
| تقوم بمقارنة الكلمة بأول السلسة حصرا إذا كانت الكلمة موجودة تقوم بإرجاع true واذا كانت غير موجودة تقوم بإرجاع false مثل str.startswith("ammar"); النتيجة true  ويمكن وضع رقم لتحديد مكان البدء مثل str.startswith("qassab",6); | startswith("",رقم) |
| تقوم بمقارنة الكلمة بأخر السلسة حصرا إذا كانت الكلمة موجودة تقوم بإرجاع true واذا كانت غير موجودة تقوم بإرجاع false مثل str.endswith("qassab"); النتيجة true  ويمكن وضع رقم لتحديد مكان البدء مثل str.endswith("ammar",7); | endswith("",رقم) |

**string Html**

**12**

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

**Numbers**

**13**

1. **في الجافا سكريبت لا تحتاج لتعريف الرقم إذا كان رقم قصير أو طويل أو عشري أو صحيح على عكس اللغات**
2. **يمكن أن يحتوي الرقم على 15 خانة إذا كان عدد صحيح أو 17 خانه إذا كان عدد عشري**
3. **المتصفح يقرأ من اليسار إلى اليمين لذلك إذا كان هناك سلسة على اليسار سيقوم بتحويل الأرقام إلى سلسة**

|  |  |
| --- | --- |
| **اسم الوظيفة** | **الرمز** |
| 1. إذا كان العدد ليس عشري يكون مكون من 15 خانه مثل 300000000000000 2. إذا كان العدد عشري يكون مكون من 17 خانه مثل 0.00000000000000003 | الخانات |
| في حال كان لدينا أصفار كثر نستخدم e+ أو e- مثل 123e+3 = 123000 | 123e-5 = 0.0123 | e+ | e- |
| 1000000 = 1\_000\_000 حيث يتم تجاهل الشريطة السفلية | \_ |
| عند جمع عددين عشرين يكون الناتج عدد عشري مع 17 خانه 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 | عشري + عشري |
| let x = 10 + 20 = 30 | صحيح + صحيح |
| المتصفح يقرأ من اليسار إلى اليمين لذلك في حال + إذا كان هناك سلسة على اليسار سيقوم بتحويل الأرقام إلى سلسة |  |
| 1. let x = 10 + 20 = 30 2. let x = "10" + "20" = 1020 3. let x = 10 + "20" = 1020 4. let x = "10" + 20 = 1020 5. let x = "ammar : " + 10 + 20 = ammar : 1020 6. let x = 10 + 20 + " ammar" = 30 ammar | فقط في حال + |
| 1. let x = "100" - "10" = 90 2. let x = "100" \* "10" = 1000 3. let x = "100" / "10" = 10 4. let x = "100" + "10" = 10010 | أما في حال باقي العمليات يتعامل معها كأنها عمليات حسابية مثل  / \* - |
| Not a Number معناها ليس رقم أي رقم غير منطقي   1. let x = "100" / "10" = 10 2. let x = "100" / "ammar" = NaN 3. let x = NaN + 5= NaN 4. let x = NaN + "5" = NaN5 5. let x = 10 – "ammar" = NaN | NaN |
| isNaN(NaN) = number | isNaN() |
| number | typeof NaN |
| زائد لانهاية Infinity =  ناقص لانهاية -Infinity =   1. عندما يكون الرقم كبير تظهر الانهاية أي Infinity مثل 2. let x = 2 / 0 = Infinity 3. let x = -2 / 0 = -Infinity | Infinity  - Infinity |
| number | typeof Infinity |
| 1. وظيفتها التحويل للأنظمه حسب الأساس 2. لنفرض لدينا الرقم الصحيح وهو نظامه عشري let x = 30 ; لتحويله للانظمه كالتالي 3. النظام الثنائي (Binary) مثل x.toString(2); = 1110 أعداده 0,1 4. النظام الثماني (Octal) مثل x.toString(8); = 36 أعداده 0,1,2,3,4,5,6,7 5. النظام العشري (Decimal) مثل x.toString(10); = 30 أعداده 0,1,2,3,4,5,6,7,8,9 6. النظام السادس عشر (Hexadecimal) مثل x.toString(16)=1e أعداده 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F | toString(الأساس) |
| let x = new Number(123);  console.log(typeof x);  لا تنشئ الرقم ككائن لانه يبطئ الكود ويمكن إرجاع نتائج غير متوقعة | تحويل الرقم إلى كائن |
| عندما نقارن رقمين تقارن القيمة فقط  عندما نقارن رقمين تقارن القيمة مع النوع | ==  === |

**Number method**

**14**

|  |  |
| --- | --- |
| **اسم الوظيفة** | **الرمز** |
| let x = 33; | let y = 3.14; |  |
| تحول الأرقام إلى نص مثل x.toString(); النتيجة 33 كنص مثال أخر (100 + 25).toString(); النتيجة 125 كنص  ملاحظة : عندما نكتب الأساس تقوم بالتحويل إلى الأنظمة مثل x.toString(2); | toString() |
| يحول الرقم إلى سلسة ويستخدم لتحديد عدد الخانات العدد العشري التي تظهر للمستخدم مثل x.toFixed(1)النتيجة 3.1 | toFixed(رقم) |
| يحول الرقم إلى سلسة ويستخدم لتحديد طول العدد الذي يظهر للمستخدم مثل x.toPrecision(2) النتيجة 3.1 | toPrecision(رقم) |
| تحول الكائن الرقمي إلى قيمته الأولية | valueOf() |
| يقوم بتحويل جميع أنواع البيانات التي يمكن تحويلها إلى أرقام أما البيانات التي لا يمكن تحويلها يرجع NaN مثل   1. Number(true) = 1 2. Number(False) = 0 3. Number("10") = 10 4. Number(" 10") = 10 5. Number("10.33") = 10.33 6. Number("10,33") = NaN 7. Number("10 33") = NaN 8. Number("ammar") = NaN 9. Number(new Date("2017-09-30")) = 1506729600000 | Number() |
| نفس Number() مع أختلاف أنه يسمح بالمسافات ويرجع أول رقم صحيح مثل parseInt("10.3 33") = 10 | parseInt() |
| نفس Number() مع أختلاف أنه يسمح بالمسافات ويرجع أول رقم عشري مثل parseFloat("10.3 33") = 10.3 | parseFloat() |
| يتحقق إذا كان العدد صحيح يرجع true أما إذا كان غير صحيح يرجع False | isInteger(رقم) |
| اذا كانت الداتا NaN يرجع true أما اذا كان ليس NaN يرجع false | isNaN() |
| 1. يحول إلى أرقام موجبة 2. consol.log(+100) = 100 3. consol.log(+"100") = 100 4. consol.log(+"-100") = -100 5. consol.log(+"10.5") = 10.5 6. consol.log(+"ammar") = NaN 7. consol.log(+oXFF) = 255 8. consol.log(+null) = 0 9. consol.log(+false) = 0 10. consol.log(+true) = 1 | + unary plus |
| 1. يحول إلى أرقام سالبة 2. consol.log(-100) = -100 3. consol.log(-"100") = -100 4. consol.log(-"-100") = 100 5. consol.log(-"10.5") = -10.5 6. consol.log(-"ammar") = NaN 7. consol.log(-oXFF) = -255 8. consol.log(-null) = -0 9. consol.log(-false) = -0 10. consol.log(-true) = -1 | -unary negationn |
| الدوال الرياضية الجاهزة | Math |
| التقريب من خلال إذا كان الرقم العشري 5 و أكثر يزيد رقم أما إذا كان أقل من خمسة لا يزيد شيء مثل  Math.round(9.5) = 10 / Math.round(9.4) = 9 | math.round() |
| يقرب إلى أعلى رقم دوما مثل Math.ceil(9.1) = 10 | Math.ceil() |
| لا يقرب الأرقام أبدا مثل Math.floor(9.9) = 9 | Math.floor() |
| دوما يأخذ الرقم الصحيح و يحذف الفاصلة العشرية مثل Math.trunc(9.6) = 9 | Math.trunc() |
| إذا كان الرقم موجب يرجع القيمة 1 أما إذا صفر يرجع 0 أما إذا سالب يرجع القيمة -1 | Math.sign() |
| الأس مثل Math.pow(8,2) = 64 | Math.pow() |
| الجذر التربيعي مثل Math.sqrt(36) = 6 | Math.sqrt() |
| الجذر التكعيبي مثل Math.cbrt(125) = 5 | Math.cbrt() |
| القيمة المطلقة مثل Math.abs(-3.3) = 3.3 | Matth.abs() |
| Math.PI = 3.14 | Math.PI |
| نغيير ال d حسب الزاوية وتكون الزاوية محصورة بين -1 <= sin() <= 1 | Math.sin(d\*Math.PI/180) |
| نغيير ال d حسب الزاوية وتكون الزاوية محصورة بين -1 <=cos() <= 1 | Math.cos(d\*Math.PI/180) |
| تحسب اللوغارتيم | Math.log() |
|  | Math.log2() |
|  | Math.log10() |
| Math.max(10, 33, -33, 100) = 100 | Math.max() |
| Math.min(10, 33, -33, 100) = -33 | Math.min() |
| تقوم بإظهر أرقام عشوائية ولإختيار مجال الأرقام نقوم بالتالي   1. Math.floor(Math.random() \* 10 ); = 0 => 9 2. Math.floor(Math.random() \* 11 ); = 0 => 10 3. Math.floor(Math.random() \* 100 ); = 0 => 99 4. Math.floor(Math.random() \* 101 ); = 0 => 100 5. Math.floor(Math.random() \* 10 ) + 1 ; = 1 => 10 6. Math.floor(Math.random() \* 10 ) + 3 ; = 3 => 12 7. Math.floor(Math.random() \* 100 ) + 1 ; = 1 => 100   function getRandomIntger( max, min) للحصول على رقم عشوائي بين رقمين  { return Math.floor(Math.random() \* (max – min +1) ) + min ; } | Math.random() |

**Number properties**

**15**

|  |  |
| --- | --- |
| **اسم الوظيفة** | **الرمز** |
| أكبر رقم في اللغة ولا يوجد رقم اكبر منه لذلك لا يمكن أضافة رقم عليه | MAX\_VALUE |
| أصغر رقم في اللغة ولا يوجد رقم أصغر منه لذلك لا يمكن أنقاص رقم منه | MIN\_VALUE |
| أكبر رقم أمن يمكن استخدامه | MAX\_SAFE\_INTEGER |
| موجب لانهاية | POSITIVE\_INFINITY |
| سالب لانهاية | NEGATIVE\_INFINITY |
| ليس رقم | NaN |

**arithmetic operators**

**16**

|  |  |
| --- | --- |
| **اسم الوظيفة** | **الرمز** |
| جمع | + |
| طرح | - |
| ضرب | \* |
| قسمه | / |
| ذيادة بمقدار واحد x++ مثل  نزود أول ثم نطبعpre increment = ++x = 1 + x  نطبع ثم نزود post increment = x++ = x + 1 | ++ |
| النقصان بمقدار واحد x—  ننقص أول ثم نطبعpre decrement = - -x = 1 - x  نطبع أول ثم ننقص post decrement = x- - = x - 1 | - - |
| أس هكذا في الأصدار الحديث أما في القديم يكون Math.pow(x,2); = x \*\* 2 | \*\* |
| باقي القسمة لمعرفة العدد الفردي % 2 = 1 رقم / إذا العدد زوجي % 2 = 0 رقم | % |
| x = y / x = y | = |
| x += y / x = x + y | += |
| x -= y / x = x - y | -= |
| x \*= y / x = x \* y | \*= |
| x /= y / x = x / y | /= |
| x \*\*= y / x = x \*\* y | \*\*= |
| x %= y / x = x % y | %= |
| الضرب و القسمة لها الاسبقية 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**

**17**

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

**logical operators**

**18**

|  |  |
| --- | --- |
| اسم الوظيفة | الرمز |
| و | && |
| أو | || |
| لا | ! |
|  | & |
|  | | |
|  | ^ |

**function**

**19**

|  |  |
| --- | --- |
| **اسم الوظيفة** | **الرمز** |
| function *name*(*parameter1, parameter2, parameter3*) {   // *code to be executed* }  الاستدعاء  *name*(*parameter1, parameter2, parameter3*);  مثل  function myFunction(p1, p2) {   return p1 \* p2;  } الريتيرن تعيد القيمة و تكتب أخر سطر في الدالة لان توقف الدالة فما بصيير نكتب شي بعدها فهي متل البريك  myFunction(p1, p2);  let x = myFunction(4, 3);  عندما يتم استدعائها بدون متغيرات و هي تملك متغيرات كتالي  myFunction;  النتيجة تكون  { return p1 \* p2;}  يمكن تعريفها بدون متغيرات مثل  function *name*() {   return 'ammar qassab'; }  *name*(); النتيجة تكون ammar qassab  يمكن تعريف الدالة في متغير مثل  const x = function (a, b) {return a \* b};  استدعاء يكون كمتغيير  let z = x(4, 3) \* 2;  يمكن استدعاء الدالة قبل الأعلان عنها  myFunction(5);  function myFunction(y) {   return y \* y; }  يمكن تشغيل الدالة من دون استدعائها وذلك بإضافة الأقواس ( function () {} ) (); فهي تعمل عند وصول الكومبايلر لها  (function () {  document.getElementById(“demo”).innerHTML = “Hello! I called myself”;  })();  يمكنك طباعة الدالة من خلال تحويلها إلى نص  function myFunction(a, b) {   return a \* b; }  let text = myFunction.toString();  ملاحظة خطيرة :  يمكن استدعاء الدالة قبل أنشاءها بالحالة العادية  Name();  function name(){}  ولكن لا يمكن استدعائها قبل إنشائها في حال تم تعريفها كمتغيير لذلك نعرفها أول ثم نستدعيها  const x = function(){}; // const x = () => {};  x(); | تعريف داله  وطريقة  استدعائها |
| طريقة الإعلان أو التعريف  Arrow Functions :  // ES6 const x = (x, y) => x \* y;  تختلف عن الطريقة القديمة فهي لإصدار الحديث  Anonymous Function :  // ES5 var x = function(x, y) {   return x \* y; }  يمكنك أن تستدعيها كذلك من خلال  const x = (x, y) => { return x \* y };  document.getElementById(“demo”).innerHTML = x(5, 5);  const x = () => { return “ammar” };  document.getElementById(“demo”).innerHTML = x();  تستخدم هذه الأنواع من الدوال في الأشياء التي تطلب تنفيذ لحظي أي الدالة لن نحتاج استخدامها مرة أخرى مثل  document.getElementById(“demo”).innerHTML = x();  document.getElementById(“demo”).innerHTML = () => {};  document.getElementById(“demo”).innerHTML = function () {};  setTimeout(() => {},2000);  setTimeout(function () {},2000); | Arrow Functions  دالة السهم  Anonymous Function  الدالة المجهولة |
| function *functionName*(*parameter1, parameter2, parameter3*) {   // *code to be executed* }  يمكننا التعديل على القيمة أو جعلها ثابتة أي إعطاء الدالة قيم افتراضية في حال لم تعطى قيمة تعمل القيمة الإفتراضية  function myFunction(x, y = 2) {  return x \* y;  }  document.getElementById(“demo”).innerHTML = myFunction(4); >>8  و في حال تم إعطاء قيمة لا تعمل القيمة الإفتراضية  function myFunction(x, y = 2) {  return x \* y;  }  document.getElementById("demo").innerHTML = myFunction(4,3); >>12  function myFunction(x, y) {  if (y === undefined) {  y = 2;  } الطريقة القديمة للقيم الأفتراضية  y = y || 2; نفس المعنى  return x \* y;  }  document.getElementById("demo").innerHTML = myFunction(4); >>8  في حال لم نعرف عدد القيم التي سنمررها لدالة نستخدم …name و هي مصفوفة حيث تخزن القيم بداخلها مثل  function print(…num) {  for (let i=0; i<num.length; i++) {  console.log(num[i]);  }  }  Print(11,22,34,56,78,90);  دالة لجمع الأرقام  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));  دالة لضرب الأرقام  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));  مثال الأقوى  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'); | Function Parameters  قيم الدوال |
| وظيفته الوصول إلى قيم الدالة المرسلة arguments و تكون على شكل مصفوفة  function myFunction() {  return arguments.length;  }  document.getElementById("demo").innerHTML = myFunction(4, 3); >>2  function myFunction(a, b) {  return arguments.length;  }  document.getElementById("demo").innerHTML = myFunction(4, 3, 5, 6); >>4  لإيجاد أكبر قيمة بين القيم الدالة  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  لجمع قيم الدالة  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 | The Arguments Object |
| function *name*() {   return 'ammar qassab'; }  typeof name; >> function | typeof |
| يتم استدعاء خواص كائن داخل داله بواسطة this أي this مخصصة لجلب خواص كائن  const myObject = {   firstName:"John",   lastName: "Doe",   fullName: function () {     return this.firstName + " " + this.lastName;   } } myObject.fullName();         // Will return "John Doe"  // This is a function constructor: 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; | Function Invocation |
| وظيفة call() هي دمج كائنين وعندما ندمج كائنين يمكننا استخدام المفاتيح للكائنين في دالة عن طريق this  كان  const myObject = {   firstName:"John",   lastName: "Doe",   fullName: function () {     return this.firstName + " " + this.lastName;   } } // This will return "John Doe": myObject.fullName();  لدمج كائنن باستخدام call()  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 2  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() مشابها تماما ل call() لكن الاختلاف بينهم هو  أن call() تمرر الخواص لدالة 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 2  const person = {   fullName: function() {     return this.firstName + " " + this.lastName;   } } const person1 = {   firstName: "Mary",   lastName: "Doe" } // This will return "Mary Doe": person.fullName.apply(person1); | Function Call  Function Apply |
| لعمر المتغيرات نوعين :   1. Global Variables : هو متغيير 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 قيمتو الجديدة في وظائف أخرى ولا ينحذف إلا إذا انتقلنا لصفحة جديدة أو سكرنا الصفحة 2. Local Variables : هو متغيير محلي تابع لدالة أو الوظيفة المكتوب داخلها حيث يمكن استعماله داخل الدالة أو الوظيفة فقط ولا يمكن استعماله خارجها ولا تنحفظ قيمته بانتهاء الدالة أو الوظيفه بل يحذف 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 globaland 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 متغيير عام لذلك بتزيد واحد واحد  function myFunction(){  document.getElementById("demo").innerHTML = add();  }  </script> كل كبسة تزيد واحد  <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;  } لا نحفظ النتيجة بمتغيير عام لذلك مالح تزيد واحد غير لمرة واحدة فقط  // Trying to increment the counter  function myFunction(){  document.getElementById("demo").innerHTML = add();  }  </script> قد ما كبسنا على الكبسه لح ضل النتيجة واحد | Variable Lifetime |
| دالة داخل دالة  document.getElementById("demo").innerHTML = add(); >>1  function add() {  let counter = 0;  function plus() {counter += 1;}  plus();  return counter;  }  ملاحظة : الدالة الداخلية الثانية يمكنها التعديل على متغييرات الدالة الخارجية الأولى أما الخارجية لا يمكنها التعديل على متغيرات الداخلية | Nested Functions |
| وهي عملية توزيع للبيانات  ليكن لدينا الكائن  const user = {theName: “ammar” ,theAge: 25 ,skills: {html:70 ,css: 80}};  لعرض الكائن باستخدام دالة بالطريقة العادية  show(user);  function show(obj) {  console.log(`your name : ${obj.theName}`);  console.log(`your age : ${obj.theAge}`);  console.log(`your skills : ${obj.css}`);  }  لقيام بعملية توزيع للكائن على متغييرات يمكن استخدامها داخل الدالة  show(user);  function show({theName , theAge ,{css}} = user) {  console.log(`your name : ${theName}`);  console.log(`your age : ${theAge}`);  console.log(`your skills : ${css}`);  }  يجب أن يكون أسماء المتغييرات مطابق لأسماء المفاتيح  لقيام بتغيير أسم المتغييرات  show(user);  function show({theName: n , theAge , skills: {css: c} } = user) {  console.log(`your name : ${n}`);  console.log(`your age : ${theAge}`);  console.log(`your skills : ${c}`);  } | Destructuring Functions |

**object**

**20**

|  |  |
| --- | --- |
| اسم الوظيفة | الرمز |
| في الجافا سكربت تقريبا كل شيء هو كائن :  الجمل الشرطية و الأرقام و النصوص يمكن تحويلها لكائن x = new String();  و التواريخ والدوال الرياضية والمصفوفات و الدوال عادية هي كلها كائنات و للكائنات نوعان :   1. كائنات أولية مثل string / number / boolean / null / undefined 2. متغييرات مثل الأسفل   const name = {key:"value", key:"value"};  حيث key تعبر عن خصائص الكائن مثل  const person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};  يمكن أن تحتوي الكائنات على دوال تقوم بوظائف معينة مثل  const person = {   firstName: "John",   lastName : "Doe",   id       : 5566,   fullName : function() {     return this.firstName + " " + this.lastName;   } };  يمكن إنشاء كائن فارغ و الإضافة عليه  const person = {}; person.firstName = "John"; person.lastName = "Doe"; person.age = 50; person.eyeColor = "blue"; | تعريف كائن |
| const person = {   firstName: "John",   lastName : "Doe",   id       : 5566,   fullName : function() {     return this.firstName + " " + this.lastName;   },  100 : “number”,  “key of” : “ammar” };  يمكننا استدعاء الكائن من خلال  يستخدم هذا النوع من الاستدعاء اذا كان المفتاح عبارة عن كلمة واحدة  name.key;  يستخدم هذا النوع من الاستدعاء اذا كان المفتاح عبارة عن كلمتين أو رقم أو أي شيء أخر غير الكلمة الواحدة أي غير محقق شروط التسمية أو اذا كان المفتاح عبارة عن متغيير ديناميكي خارجي  name["key"];  مثل  let bra = “id”;  person.firstName + " " + person.lastName;  person["100"] + " " + person["key of"] + person[bra];  استدعاء الدوال داخل الكائن  person.fullName(); >> john Doe  person.fullName; >> { return this.firstName + " " + this.lastName;}  بمكننا الوصول لكل خصائص الكائن من خلال الحلقات  const person = {  fname:"John",  lname:"Doe",  age:25  };  let txt = "";  for (let x in person) {  txt += person[x] + " ";  }  document.getElementById("demo").innerHTML = txt; | استدعاء كائن من خلال خصائص الكائنات |
| يمكننا إضافة خاصية جديدة للكائن من خلال objectName.key = value ;  const person = {  firstname: "John",  lastname: "Doe",  age: 50,  eyecolor: "blue"  };  person.nationality = "English";  document.getElementById("demo").innerHTML =  person.firstname + " is " + person.nationality + "."; | Adding New Properties |
| يمكننا حذف خاصية من خلال delete objectName.key; أو delete objectName["key"];  const person = {  firstname: "John",  lastname: "Doe",  age: 50,  eyecolor: "blue"  };  delete person.age;  document.getElementById("demo").innerHTML =  person.firstname + " is " + person.age + " years old ."; // John is undefined years old . | Deleting Properties |
| كائن داخل كائن  myObj = {   name:"John",   age:30,   cars: {     car1:"Ford",     car2:"BMW",     car3:"Fiat"   } }  طريقة الوصول لخصائص في هذه الحالة  document.getElementById("demo").innerHTML = myObj.cars.car2;  document.getElementById("demo").innerHTML = myObj.cars["car2"];  document.getElementById("demo").innerHTML = myObj["cars"]["car2"]; | Nested Objects |
| طريقة بناء مصفوفة داخل كائن و طريقة الوصول إليها عن طريق الحلقات  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] + "<br>";  }  }  document.getElementById("demo").innerHTML = x; | Nested Arrays and Objects |
| 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 | تحويل المتغييرات إلى كائنات باستخدام  new |
| يمكن وضع تابع داخل الكائن و يمكن الوصول إليه من خلال this  // 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();  حيث وصلنا إليه من خلال person.fullName() فيرجع القيمة  أما person.fullName فترجع function() { return this.firstName + " " + this.lastName; }  و لإضافة تابع لكائن من خارج الكائن كتالي  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();  لاستخدام التوابع الجاهزة داخل التابع كتالي  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(); | Object Methods |
| طرق عرض الكائن :  أسم الكائن  const person = {  name: "John",  age: 30,  city: "New York"  };  document.getElementById("demo").innerHTML = person;// [object Object]  عرض خصائص معينة  const person = {  name: "John",  age: 30,  city: "New York"  };  document.getElementById("demo").innerHTML = person.name + ", " + person.age +", "+ person.city;  العرض عن طريق حلقة  const person = {  name: "John",  age: 30,  city: "New York"  };  let txt = "";  for (let x in person) {  txt += person[x] + " ";  };  document.getElementById("demo").innerHTML = txt;  يمكنك تحويل الكائن إلى مصفوفة ثم عرض المصفوفة عن طريق Object.values()  const person = {  name: "John",  age: 30,  city: "New York"  };  document.getElementById("demo").innerHTML = Object.values(person);  يمكن تحويل الكائن إلى نص ثم عرض النص عن طريق JSON.stringify()  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 المشكله نحول الخاصية لنص أولا 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); | Display Objects |
| Getters and Setters  موصلات الكائنات التي تستخدم لتعديل على خصائص الكائن وهي :  get lang() تستخدم للوصول إلى خصائص الكائن  // 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;  set lang(value) تستخدم لتعديل على خصائص الكائن من خلال المتغيير  // Create an object:  const person = {  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;  أنتبه لطريقة الاستدعاء بدون أقواص  // 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 أقواص  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";  // Display data from the object:  document.getElementById("demo").innerHTML = person.language;  Object.defineProperty() يمكن عن طريق التابع إضافة Getters and Setters خارجي للكائن لسهولة 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 | Object Accessors |
| نعلم أن يمكننا تحويل أي شئ لكائن من خلال new حيث تحول المتغيير إلى كائن  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  ملاحظة : لاتنشئ الأرقام والحروف النصوص ككائنات لانها تكون بطيئة الاستجابة  أما الكائن المنشأ Object Constructors فيعتمد إنشائه على دالة بانية ثابتة يتم تمرير لها قيم وتحولهم إلى قيم و لمفاتيح الكائن جديد منشأ  مثال 1  // Constructor function for Person objects  function Person(first, last, age, eye) {  this.firstName = first;  this.lastName = last;  this.age = age;  this.eyeColor = eye;  } الدالة البانية الثابتة التي تقوم بإنشاء كائنات ذات مفاتيح ثابتة ولكن قيم هذه المفاتيح تتغير بتغيير قيم المررة  إنشاء الكائن عبر تمرير قيم مفاتيحه عبر الدالة البانية  const myFather = new Person("John", "Doe", 50, "blue");  عرض الكائن الجديد المنشأ  document.getElementById("demo").innerHTML =  "My father is " + myFather.age + " " + typeof(myFather); >> My father is 50 object  مثال 2  function Person(first, last, age, eye) {  this.firstName = first;  this.lastName = last;  this.age = age;  this.eyeColor = eye;  }  يتم إنشاء كائنين مختلفين عبر دالة بانية واحدة  const myFather = new Person("John", "Doe", 50, "blue");  const myMother = new Person("Sally", "Rally", 48, "green");  document.getElementById("demo").innerHTML =  "My father is " + myFather.age + " .<br> My mother is " + myMother.age + " ."; >>  // My father is 50 . My mother is 48 .  مثال 3 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 4 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 لان 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  بدلا 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   مثال 5 يمكن توريث الدالة البانية خواص من خلال prototype  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  يمكن توريث الدالة البانية دالة من خلال prototype  Person.prototype.name = function() {  return this.firstName + " " + this.lastName  };  ملاحظة : لاتنشئ الأرقام و الحروف و النصوص ككائنات لانها تكون بطيئة الاستجابة | Object Constructors |
| تقوم بإنشاء كائن جديد من دمج عدة كائنات  شكلها  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 أخر 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); | Object assign() |
| for ( of ) : تستخدم لطباعة أو التعديل على الكائنات القابلة لتكرار مثلArrays, Strings, Maps, NodeLists  مثال 1  const name = "W3Schools";  let text = ""  for (const x of name) {  text += x + "<br>";  }  document.getElementById("demo").innerHTML = text;  مثال 2  const letters = ["a","b","c"];  let text = "";  for (const x of letters) {  text += x + "<br>";  }  document.getElementById("demo").innerHTML = text;  Iterable objects : هو كائن قابل لتكرار حيث يتم إنشائه من خلال دالة و تعيد هذه الدالة كائن له مفتاح على شكل دالة بداخله كائن أخر له مفتاحين مع قيمتين  مثال 1  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 أخر 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 3  مثال 2 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 for ( of ) 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 3  // Create an Object  myNumbers = {};  // Make it Iterable  myNumbers[Symbol.iterator] = function() {  let n = 0;  done = false;  return {  next : function () {  n += 10;  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/ | Object Iterables |
| وهي كائنات تم شرحيها في الفصل 32/33 يتم إنشائها كتالي  const letters = new Set(["a","b","c"]);  const fruits = new Map([   ["apples", 500],   ["bananas", 300],   ["oranges", 200] ]); | object Map()  object Set() |
| وهي تبديل عناصر الكائن إلى متغيرات خارجية  مثال 1  const person = {  firstName: “John”,  lastName: “Doe”,  language: “en”  };  يتم التوزيع بنفس أسماء العناصر داخل الكائن من أجل الترتيب  const {firstName ,lastName ,language} = person;  console.log(firstName);// John  console.log(lastName);// Doe  console.log(language);// en  مثال 2  const person = {  firstName: “John”,  lastName: “Doe”,  language: “en”  };  const {firstName,language} = person;  console.log(firstName);// John  console.log(language);// en  أذا كانت المتغيرات معرفة نستخدم القوسين ()  مثال 3  const person = {  firstName: “John”,  lastName: “Doe”,  language: “en”  };  const firstName, lastName, language;  ({firstName, lastName, language} = person);  console.log(firstName);// John  console.log(lastName);// Doe  console.log(language);// en  لتغيير أسماء المتغييرات التي كانت أسماء عناصر إلى أسماء جديدة  مثال 4  const person = {  firstName: “John”,  lastName: “Doe”,  language: “en”  };  const firstName, lastName, language;  ({firstName: f , lastName: l , language} = person);  console.log(f);// John  console.log(l);// Doe  console.log(language);// en  مثال 5 لإضافة قيمة أفتراضية للعنصر أو إضافة قيمة جديدة  const person = {  firstName: “John”,  lastName: “Doe”,  language: “en”  };  const {firstName ,lastName ,language ,color = “red”} = person;  console.log(firstName);// John  console.log(lastName);// Doe  console.log(language);// en  console.log(color);// red  مثال 6  const person = {  firstName: “John”,  lastName: “Doe”,  language: “en”,  color: ”black”  };  const {firstName ,lastName: l ,language ,color = “red”} = person;  console.log(firstName);// John  console.log(l);// Doe  console.log(language);// en  console.log(color);// black | Destructuring object |
| توزيغ المتغييرات للكائنات المتداخلة  const person = {  firstName: “John”,  lastName: “Doe”,  language: “en”,  color: ”black”,  skills: {  html: 70,  css: 80  }  };  const {firstName,lastName,language: e ,color: ”black”,skills: {html,css} } = person;  console.log(firstName);// John  console.log(lastName);// Doe  console.log(e);// en  console.log(color);// black  console.log(html);// 70  يمكن الوصول إلى عناصر الكائن المتداخل فقط  const {html ,css} = person.skills;  console.log(html);// 70  console.log(css);// 80 | Destructuring Nested object |

**array**

**21**

|  |  |
| --- | --- |
| اسم الوظيفة | الرمز |
| تستخدم المصفوفات لتخزين البيانات التي تعود لشئ واحد مثل أنواع سيرات أو أسماء أشخاص | الاستخدام |
| const *array\_name* = [*item1*, *item2*, ...];  مثل  const cars = ["mazda", "Volvo", "BMW"];  المسافات وفواصل الأسطر ليست مهمة. يمكن أن يمتد الإعلان على عدة أسطر:  const cars = [   "Saab",   "Volvo",   "BMW" ];  لإنشاء مصفوفة ثنائية  const cars = ["mazda", "Volvo", ["mazda", "Volvo", "BMW"] ];  أو طريقة أخرى للأنشاء لكن لا نستخدمها للأنشاء من أجل السرعة  const cars = new Array("Saab", "Volvo", "BMW"); | أنشاء مصفوفة |
| const cars = ["mazda", "Volvo", "BMW"];  أو  const cars = []; cars[0]= "Saab"; cars[1]= "Volvo"; cars[2]= "BMW";  في حال مصفوفة ثنائية  const cars = [][]; cars[0][0]= "Saab"; cars[0][1]= "Volvo"; cars[0][2]= "BMW"; | طريقة الأدخال |
| وهي مصفوفة داخل مصفوفة  const cars = ["mazda", "Volvo", ["ammar", "qassab", [1, “abd alqader”, 2]] ]; | Nested Arrays |
| فقط بكتابة أسم المصفوفة مثل :  const cars = ['ammar','qassab'];  document.getElementById("demo").innerHTML = cars;  ملاحظة : عند طباعة مصفوفة و يكون عنصر فيها فارغ يتم طباعة الفراغ فقط | طريقة الطباعة |
| const cars = ["ammar", "Volvo", "BMW"];  document.getElementById("demo").innerHTML = cars[0]; >> ammar  حيث يكون رقم أول عنصر 0 ثم 1 و هكذا ومن أجل تعديل العنصر نصل إليه كتالي  cars[0] = 'qassab';  document.getElementById("demo").innerHTML = cars[0]; >> qassab  للوصول للحرف الثاني داخل العنصر الأول  document.getElementById("demo").innerHTML = cars[0][1]; >> a  في حال كان لدينا مصفوفة ثنائية ونريد الوصول إليها  const cars = ["mazda", "Volvo", ["mazda", "Volvo", "BMW"] ];  document.getElementById("demo").innerHTML = cars[2][1]; >> Volvo  للوصول إلى كامل المصفوفة نكتب فقط أسم المصفوفة مثل  document.getElementById("demo").innerHTML = cars; >> ammar,Volvo,BMW | الوصول إلى عناصر المصفوفة لتعديلها |
| const cars = ["mazda", "Volvo", "BMW"];  cars[2];  const person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};  person.firstName;   1. تستخدم المصفوفة فهارس مرقمة 2. يستخدم الكائن فهارس مسماة 3. نستخدم المصفوفة عندما نريد أسماء الفهارس ارقام 4. نستخدم الكائنات عندما نريد أسماء الفهارس نصوص | Arrays are Objects |
| نستخدم length  const fruits = ["Banana", "Orange", "Apple", "Mango"];  fruits.length; // Returns 4 | لمعرفة عدد عناصر المصفوفة |
| const fruits = ["Banana", "Orange", "Apple", "Mango"];  fruits[0]; // Returns "Banana"  fruits[fruits.length - 1]; // Returns "Mango" | لوصول لأول عنصر و أخر عنصر بغض النظر عن عدد العناصر |
| const fruits = ["Banana", "Orange", "Apple", "Mango"];  let text = "<ul>";  for (let i = 0; i < fruits.length; i++) {  text += "<li>" + fruits[i] + "</li>";  }  text += "</ul>";  document.getElementById("demo").innerHTML = text;  ///////////////////////////  const fruits = ["Banana", "Orange", "Apple", "Mango"];  let text = "<ul>";  fruits.forEach(myFunction);  text += "</ul>";  document.getElementById("demo").innerHTML = text;  function myFunction(value) {  text += "<li>" + value + "</li>";  } | طريقة طباعة جميع العناصر |
| 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  /////////////  لا تضيف العناصر بهذه الطريقة من أجل عدم ترك فرغات في المصفوفة  const fruits = ["Banana", "Orange", "Apple"];  fruits[6] = "Lemon"; | إضافة عنصر إلى المصفوفة |
| const fruits = ["Banana", "Orange", "Apple"]; typeof fruits;    // returns object | typeof |
| لتحقق من المصفوفة إذا كانت مصفوفة يكون الخرج true وإذا كانت ليست مصفوفة يكون الخرج false  const fruits = ["Banana", "Orange", "Apple"];  document.getElementById("demo").innerHTML = Array.isArray(fruits); | Array.isArray() |
| 1. لايمكن إعادة تعين مصفوفة تم التعريف عنها بإستخدام const إلا إذا استخدمت أسم الفهرس 2. يجب التصريح عن عناصر المصفوفة عند تعريف المصفوفة ب const أو تعرف العناصر لاحقا باستخدام الفهرسة 3. يرجى الإنتباه إلى نطاق الكتل عند التعريف المصفوفة سواء ب const أو var 4. var يسمح بإعادة تعديل عناصر المصفوفة بشكل يدوي على عكس const | ملاحظات |
| توزيع عناصر المصفوفة إلى متغييرات حيث يتم توزيع القيم على التتابع  مثال 1  const myArrya = ["Banana", "Orange", "Apple"];  const [a, b, c] = myArrya;  console.log(a);// Banana  console.log(b);// Orange  console.log(c);// Apple  مثال 2  const a = 1, b = 2, c = 3;  const myArrya = ["Banana", "Orange", "Apple"];  [a, b, c] = myArrya;  console.log(a);// Banana  console.log(b);// Orange  console.log(c);// Apple  مثال 3  const a = 1, b = 2;  const myArrya = ["Banana", "Orange", "Apple"];  [a, b] = myArrya;  console.log(a);// Banana  console.log(b);// Orange  console.log(c);// undefined  مثال 4  const a = 1, b = 2, c = 3;  const myArrya = ["Banana", "Orange"];  [a, b, c] = myArrya;  console.log(a);// Banana  console.log(b);// Orange  console.log(c);// 3  مثال 5  const a = 1, b = 2, c = 3;  const myArrya = ["Banana", "Orange"];  [a = “A”, b = “B”, c = ”C”] = myArrya;  console.log(a);// Banana  console.log(b);// Orange  console.log(c);// C  مثال 6 نستخدم الفراغ لعدم توزيع العنصر  const a = 1, b = 2, c = 3;  const myArrya = ["Banana", "Orange", "Apple"];  [a, , b] = myArrya;  console.log(a);// Banana  console.log(b);// Apple | Destructuring arrya |
| const cars = ["mazda", "Volvo", ["ammar", "qassab", [1, “abd alqader”, 2]] ];  const [ , ,[ , a, [ , b, ]] = cars;  console.log(a);// qassab  console.log(b);// abd alqader | Destructuring Nested arrya |
| لتبديل بين قيم  let book = “video”;  let video = “book”;  [book, video] = [video, book];  console.log(book);// book  console.log(video);// video | Destructuring swapping |

**array method**

**22**

|  |  |
| --- | --- |
| اسم الوظيفة | الرمز |
| تقوم بتحويل المصفوفة إلى سلاسل نصية  const fruits = ["Banana", "Orange", "Apple", "Mango"]; document.getElementById("demo").innerHTML = fruits.toString();  مع الملاحظة عند طباعة المصفوفة تتحول إلى سلسة نصية | toString() |
| تقوم بتحويل المصفوفة إلى سلاسل نصية مع إمكانية تغيير الفاصلة بين العناصر عند الطباعة  const fruits = ["Banana", "Orange", "Apple", "Mango"];  document.getElementById("demo").innerHTML = fruits.join(" / "); | join("") |
| وظيفتها فقط إزالة أخر عنصر بين عناصر المصفوفة  fruits.pop(); | pop() |
| وظيفتها فقط إزالة أول عنصر بين عناصر المصفوفة  fruits.shift(); | shift() |
| 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]; | الحذف العناصر المحددة |
| وظيفتها إضافة عنصر في أخر المصفوفة  const fruits = ["Banana", "Orange", "Apple", "Mango"]; fruits.push("Kiwi");   // Adds "Kiwi" to fruits  أو إضافة عناصر مصفوفة الأولى في أخر مصفوفة جديدة مثال  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); | push() |
| وظيفتها إضافة عنصر في أول المصفوفة  const fruits = ["Banana", "Orange", "Apple", "Mango"]; fruits.unshift("Lemon");    // Add "lemon" | unshift() |
| لإضافة مجموعة عناصر إلى المصفوفة ويتم إختيار المكان حسب الرقم الأول والرقم الثاني مسؤول عن عدد العناصر التي يجب أن تحذف المكان  const fruits = ["Banana", "Orange", "Apple", "Mango"]; fruits.splice(2, 0, "Lemon", "Kiwi");  كما يستخدم لإزالة العناصر من المصفوفة  const fruits = ["Banana", "Orange", "Apple", "Mango"]; fruits.splice(0, 1);   // Removes the first element | splice() |
| لجمع بين مصفوفتين موجودتين  const myGirls = ["Cecilie", "Lone"]; const myBoys = ["Emil", "Tobias", "Linus"]; // Concatenate (join) myGirls and myBoys const myChildren = myGirls.concat(myBoys);  كما يمكنه أن يأخذ أي عدد من المصفوفات  array1.concat(array2, array3);  كما يستخدم لإضافة العناصر إلى نهاية المصفوفة  const myArray = ["Emil", "Tobias", "Linus"];  const myChildren = myArray.concat("Peter");  document.getElementById("demo").innerHTML = myChildren; | concat() |
| لقطع أول عنصر من البداية  const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"]; const citrus = fruits.slice(1);  لقطع أول ثلاث عناصر من البداية  const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"]; const citrus = fruits.slice(3);  لقطع جزء من المصفوفة  const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"]; const citrus = fruits.slice(1, 3); | slice() |
| يتم فرز العناصر حسب الابجدية  const fruits = ["Banana", "Orange", "Apple", "Mango"]; fruits.sort();  لفرز المصفوفة الأرقام تصاعديا  const points = [40, 100, 1, 5, 25, 10];  document.getElementById("demo").innerHTML = points;  function myFunction() {  points.sort(function(a, b){return a – b});  document.getElementById("demo").innerHTML = points;  }  لفرز المصفوفة الأرقام تنازليا  const points = [40, 100, 1, 5, 25, 10];  document.getElementById("demo").innerHTML = points;  function myFunction() {  points.sort(function(a, b){return b – a});  document.getElementById("demo").innerHTML = points;  }  فرز المصفوفة بترتيب عشوائي  const points = [40, 100, 1, 5, 25, 10];  document.getElementById("demo").innerHTML = points;  function myFunction() {  points.sort(function(a, b){return 0.5 - Math.random()});  document.getElementById("demo").innerHTML = points;  } | sort() |
| const points = [40, 100, 1, 5, 25, 10];  document.getElementById("demo").innerHTML = myArrayMax(points);  function myArrayMax(arr) {  return Math.max.apply(null, arr);  }  ////////////  const points = [40, 100, 1, 5, 25, 10];  document.getElementById("demo").innerHTML = myArrayMax(points);  function myArrayMax(arr) {  let len = arr.length;  let max = -Infinity;  while (len--) {  if (arr[len] > max) {  max = arr[len];  }  }  return max;  } | لإخراج أكبر رقم في المصفوفة |
| 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;  } | لإخراج أصغر رقم في المصفوفة |
| 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 + "<br>" +  cars[1].type + " " + cars[1].year + "<br>" +  cars[2].type + " " + cars[2].year;  }  ///////  const cars = [  {type:"Volvo", year:2016},  {type:"Saab", year:2001},  {type:"BMW", year:2010}  ];  displayCars();  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;  } | لفرز المصفوفة الكائنات |

**array Iteration**

**23**

|  |  |
| --- | --- |
| اسم الوظيفة | الرمز |
| تستخدم لطباعة المصفوفة ولا تستخدم لتعدبل على قيمها ولا تستخدم لإنشاء مصفوفة جديدة من قديمة هي فقط تستخدم لطباعة ولمعالجة الأحداث في ال Dom  const numbers = [45, 4, 9, 16, 25];  let txt = "";  numbers.forEach(myFunction);  document.getElementById("demo").innerHTML = txt;  function myFunction(value, index, array) {  txt += value + "<br>";  } | forEach() |
| تستخدم لإنشاء مصفوفة جديدة من خلال التعديل على قيم مصفوفة قديمة دون التأثير على المصفوفة القديمة و تاخذ داخلها (دالة و (arguments مثل map(function, arguments)  و الدالة تحوي برامترات oldArray.map(function(value, index, array){}, arguments) وتقوم هذه الدالة بالوصول الى قيم المصفوفة القديمة والتعديل على القيم لتجعل الماب تنشيء مصفوفة جديدة  مثال 1  const numbers1 = [45, 4, 9, 16, 25];  const numbers2 = numbers1.map (function(value, index, array) { return value \* 2;});  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  document.getElementById("demo").innerHTML = numbers2;  مثال 2  const numbers1 = [45, 4, 9, 16, 25];  const numbers2 = numbers1.map ((value) => { return value \* 2;});  document.getElementById("demo").innerHTML = numbers2;  مثال 3  const numbers1 = [45, 4, 9, 16, 25];  const numbers2 = numbers1.map(myFunction);  document.getElementById("demo").innerHTML = numbers2;  function myFunction(value, index, array) {  return value \* 2;  }  مثال 4 لتحويل الأحرف من الأحرف الكبيرة إلى صغيرة و بالعكس  let letter = “Ammar”;  let newLetter = letter.split(“”).map(function (val) {  return val === val.toUpperCase() ? val.toLowerCase() : val.toUpperCase() ;  // condition ? true : false  }).join(“”);  استخدمنا split لتحويل النص إلى مصفوفة وبعد التعديل بال map قمنا بتحويل المصفوفة المعدلة إلى نص من خلال join  Console.log(newLetter); >> aMMAR  مثال 5 لتحويل الأرقام السالبة إلى موجبة وبالعكس  let num = [-1,10,30,-40,-50];  let invertedNumber = num.map(function(val) {return -val;});  Console.log(invertedNumber); >> [1,-10,-30,40,50]  مثال 6 إزالة الأرقام من النص و الإبقاء على الأحرف  let ignoreNumber = “A1M2M3A4R5”;  ملاحظة :  isNaN(‘a’)= true/ isNaN(10)= false/ parseInt(‘a’)= NaN/ parseInt(10) = 10/ isNaN(NaN)=true  let ign = ignoreNumber.split(“”).map(function(val) {  return isNaN(parseInt(val)) ? val : val=””;// isNaN(NaN)=true / isNaN(10)=false  }).join(“”);  Console.log(ign); >> AMMAR | map() |
| تشبه ال map() تستخدم لإنشاء مصفوفة جديدة من خلال فلترة قيم مصفوفة قديمة من خلال شرط معين دون التأثير على المصفوفة القديمة و تاخذ داخلها (دالة و (arguments مثل filter(function, arguments)  و الدالة تحوي برامترات oldArray.filter(function(value, index, array){}, arguments) وتقوم هذه الدالة بالوصول الى قيم المصفوفة القديمة و فلترتها من خلال شرط يوضع بعد return (condition) في دالة الفلتير لتجعل الفلتير تنشيء مصفوفة جديدة مفلترة  وظيفتها فلترة العناصر المصفوفة القديمة و إرجاع مصفوفة جديدة مفلترة  شكلها  Let newArray = oldArray.filter(function(value, index, array) {  return condition ? true : false;  اذا نتيجة الشرط true يقوم بتمريم العنصر من المصفوفة القديمة إلى الجديدة أما إذا false لا يقوم بتمريره  });  مثال 1  const numbers = [45, 4, 9, 16, 25];  const over18 = numbers.filter(myFunction);  document.getElementById("demo").innerHTML = over18;  function myFunction(value, index, array) {  return value > 18;// الشرط الذي يتحقق من كل عنصر و العنصر المحقق يخزن في المصفوفة الجديدة  }  مثال 2 فلترة الأعداد الزوجية  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]  مثال 3 فلترة الكلمات التي أحرفها أكثر من أربعة  let sentense = “I love foood code too playing much”;  let filsentense = v.split(“ “).filter(function(val) {  return val.length <=4 ;  }).join(“ “);  Console.log(filsentense); >> I love code too much  ملاحظة :  الفرق بين map() و filter() :  Map() : عند وضع داخلها شرط بدون true و false سوف ترجع نتيجة الشرط أي سوف ترجع true و false مثل  let ignoreNumber = “A1M2M3A4R5”;  let ign = ignoreNumber.split(“”).map(function(val) {  return isNaN(parseInt(val)) ; // الجواب هو ترو وفلس فقط أي المصفوفة الجديدة داخلها ترو و فلس فقط  }).join(“”);  Console.log(ign); >> truefalse truefalse truefalse truefalse truefalse  filter() : عند وضع داخلها شرط بدون true و false سوف ترجع العنصر الذي حقق الشرط مثل  let sentense = “I love foood code too playing much”;  let filsentense = v.split(“ “).filter(function(val) {  return val.length <=4 ;// ترجع قيم عناصر المصفوفة التي تحقق الشرط وليس نتيجة الشرط  }).join(“ “);  Console.log(filsentense); >> I love code too much | filter() |
| تشبه ال map() و filter() و تعمل 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 عنصريين تؤخذ و تجمع مع العنصر الثالث وهكذا  لها شكلان  الشكل الأول :  oldArray.reduce(function(total, value, index, array) {return (العملية الحسابية);})  في هذا الشكل تكون total هي قيمة العنصر الأول لان لا توجد قيمة أفتراضية  و تكون value هي قيمة العنصر الثاني في المصفوفة و index هو اندكس العنصر الفالو دوما فيكون يساوي 1  و تكون array هي المصفوفة التي نأخذ منها القيم  تؤخذ قيمة العنصر الثاني value و تجمع مع total فيصبح لدينا total جديدة تجمع مع العنصر الثالث و هكذا  ليتم جمع كل العناصر في حال كانت العملية جمع أما إذا لم تكن جمع تبقى تتكرر العملية لتنهي كل المصفوفة  الشكل الثاني :  oldArray.reduce(function(total, value, index, array) {return (العملية الحسابية);} , firsttotal)  في هذا الشكل تكون total هي قيمة الأفتراضية الأولى firsttotal  و تكون value هي قيمة العنصر الأول في المصفوفة و index هو اندكس العنصر الفالو دوما فيكون يساوي 0  و تكون array هي المصفوفة التي نأخذ منها القيم  تؤخذ قيمة العنصر الأول value و تجمع مع total فيصبح لدينا total جديدة تجمع مع العنصر الثاني و هكذا  ليتم جمع كل العناصر في حال كانت العملية جمع أما إذا لم تكن جمع تبقى تتكرر العملية لتنهي كل المصفوفة  مثال 1  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;  }  مثال 2 يمكن إضافة عدد أولي لعملية  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;  }  مثال 3 اخراج أكبر كلمة في المصفوفة  let sentense = [“bla”, “propaganda”, “ammar”, “abdalqader”, “qassab”];  let newSentense = sentense.reduce(function(total,value) {  return total.length > value.length ? total : value;  });  document.getElementById("demo").innerHTML = "The " + newSentense; >> abdalqader | reduce() |
| تعمل على اختزال عناصر المصفوفة من اليمين إلى اليسار من خلال عملية حسابية مثل الجمع  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;  } | reduceRight() |
| طريقة لتحقق من ان جميع عناصر المصفوفة أتمت الإختبار فترجع true or false  const numbers = [45, 4, 9, 16, 25];  let allOver18 = numbers.every(myFunction);  document.getElementById("demo").innerHTML = "All over 18 is " + allOver18;  function myFunction(value, index, array) {  return value > 18;  } | every() |
| طريقة لتحقق من ان بعض عناصر المصفوفة أتمت الإختبار فترجع true or false  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;  } | some() |
| تبحث عن قيمة العناصر في المصفوفة والبحث من البداية وتعيد موضعه array.indexOf(item, start)  const fruits = ["Apple", "Orange", "Apple", "Mango"];  let position = fruits.indexOf("Apple") + 1;  document.getElementById("demo").innerHTML = "Apple is found in position " + position; | indexOf() |
| تبحث عن قيمة العناصر في المصفوفة والبحث من النهاية وتعيد موضعه array.lastindexOf(item, start)  const fruits = ["Apple", "Orange", "Apple", "Mango"];  let position = fruits.lastIndexOf("Apple") + 1;  document.getElementById("demo").innerHTML = "Apple is found in position " + position; | lastIndexOf() |
| يبحث عن قيمة العنصر في المصفوفة إذا وجد يرجع true و إذا لم يوجد يرجع false  const fruits = ["Banana", "Orange", "Apple", "Mango"];  document.getElementById("demo").innerHTML = fruits.includes("Mango"); | includes() |
| يرجع قيمة أول عنصر في المصفوفة يجتاز الإختبار  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;  } | find() |
| يرجع مكان العنصر في المصفوفة الذي يجتاز الإختبار  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;  } | findIndex() |
| const myArr = Array.from("ABCDEFG");  document.getElementById("demo").innerHTML = myArr; | from() |
| تقوم بإرجاع الفهارس للمصفوفة  const fruits = ["Banana", "Orange", "Apple", "Mango"];  const keys = fruits.keys();  let text = "";  for (let x of keys) {  text += x + "<br>";  }  document.getElementById("demo").innerHTML = text; | Keys() |

**data**

**24**

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

**data formats**

**25**

|  |  |
| --- | --- |
| اسم الوظيفة | الرمز |
| نكتب التاريخ كسلسة نصية | iso |
| const d = new Date("2015-03-25");  document.getElementById("demo").innerHTML = d; | ISO Dates |
| const d = new Date("2015-03-25T12:00:00Z");  document.getElementById("demo").innerHTML = d;  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 و إضافة الوقت الحالي  document.getElementById("demo").innerHTML =new Date("2015-03-25T12:00:00-06:00");  عندما لا يتم تحديد المنطقة الزمنية سيتم أخذ توقيت المتصفح | ISO Dates (Date-Time) |
| const d = new Date("03/25/2015");  document.getElementById("demo").innerHTML = d; | Short Dates |
| 1. يمكن ان يظهر خطأ إذا تم وضع التاريخ بدون أصفار يجب أن يكون const d = new Date("2015-03-25"); 2. لا يمكن كتابة التاريخ بالشكل هذا / < const d = new Date("2015/03/25"); 3. لا يمكن كتابة التاريخ بالشكل هذا < const d = new Date("25-03-2015"); | تحزيرات |
| حيث يكون الشهر على شكل نص ويكون الترتيب على الشكل التالي يوم شهر سنه  const d = new Date("25 Mar 2015");  document.getElementById("demo").innerHTML = d;  ويمكن كتابة الشهر بشكل كامل أو إختصار الكلمة ويمكن تبديل مكان الشهر باليوم | Long Dates |
| حيث تحول التاريخ المعطى بشكل سلسلة إلى رقم ميلي ثانية  const msec = Date.parse("21 March 2012");  document.getElementById("demo").innerHTML = msec;  يمكنك بعد ذلك استخدام عدد المللي ثانية لتحويله إلى كائن تاريخ  let msec = Date.parse("21 March 2012");  const d = new Date(msec);  document.getElementById("demo").innerHTML = d; | Date.parse() |

**Get Date Methods**

**26**

|  |  |
| --- | --- |
| اسم الوظيفة | الرمز |
| تستخدم هذه الطريقة للحصول على التاريخ و الوقت |  |
| يعطي رقم ميلي ثانية من التريخ 1970 إلى التاريخ الحالي  const d = new Date();  document.getElementById("demo").innerHTML = d.getTime(); | getTime() |
| ترجع التاريخ الحالي كرقم مكون من أربع أرقام  const d = new Date();  document.getElementById("demo").innerHTML = d.getFullYear(); | getFullYear() |
| ترجع الشهر الحالي كرقم من 0 إلى 11 لذلك نضفاها واحد  const d = new Date();  document.getElementById("demo").innerHTML = d.getMonth() + 1;  يمكننا إرجاع الشهر كنص بإستخدام هذه الطريقة  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()]; | getMonth() |
| طريقة لإرجاع اليوم كرقم من 1 إلى 31  const d = new Date();  document.getElementById("demo").innerHTML = d.getDate(); | getDate() |
| طريقة لإرجاع الساعات كرقم من 0 إلى 23  const d = new Date();  document.getElementById("demo").innerHTML = d.getHours(); | getHours() |
| طريقة لإرجاع الدقائق كرقم من 0 إلى 59  const d = new Date();  document.getElementById("demo").innerHTML = d.getMinutes(); | getMinutes() |
| طريقة لإرجاع الثواني كرقم من 0 إلى 59  const d = new Date();  document.getElementById("demo").innerHTML = d.getSeconds(); | getSeconds() |
| طريقة لإرجاع الميلي ثانية كرقم من 0 إلى 999  const d = new Date();  document.getElementById("demo").innerHTML = d.getMilliseconds(); | getMilliseconds() |
| طريقة لإرجاع الأيام كرقم من 0 إلى 6  const d = new Date();  document.getElementById("demo").innerHTML = d.getDay() + 1;  ويمكن إرجاع أيام الأسبوع كنص باستخدام هذا الطريقة  const d = new Date();  constdays=["Saturday","Sunday","Monday","Tuesday","Wednesday","Thursday","Friday"];  document.getElementById("demo").innerHTML = days[d.getDay()]; | getDay() |
| وهي طريقة من أجل تحديد المنطقة الزمنية و ذلك من خلال إضافة UTC بعد get مثل getUTCFullYear() | UTC |

**Set Date Methods**

**27**

|  |  |
| --- | --- |
| اسم الوظيفة | الرمز |
| تستخدم هذا الوظائف لتعديل جزء من التاريخ مع بقاء الجزء الأخر |  |
| طريقة لتعديل جزء من التاريخ أو كله  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; | setFullYear() |
| طريقة لتعديل الشهر كرقم من 0 إلى 11  const d = new Date();  d.setMonth(11);  document.getElementById("demo").innerHTML = d; | setMonth() |
| طريقة لتعديل اليوم كرقم من 1 إلى 31  const d = new Date();  d.setDate(15);  document.getElementById("demo").innerHTML = d;  كما يمكننا إضافة أيام إلى التابع  const d = new Date();  d.setDate(d.getDate() + 50);  document.getElementById("demo").innerHTML = d;  إذا أدت إضافة أيام إلى تغيير الشهر أو السنة ، فسيتم معالجة التغييرات تلقائيًا بواسطة كائن التاريخ. | setDate() |
| طريقة لتعديل الساعات من 0 إلى 23  const d = new Date();  d.setHours(22);  document.getElementById("demo").innerHTML = d; | setHours() |
| طريقة لتعديل الدقائق من 0 إلى 59  const d = new Date();  d.setMinutes(30);  document.getElementById("demo").innerHTML = d; | setMinutes() |
| طريقة لتعديل الثواني من 0 إلى 59  const d = new Date();  d.setSeconds(30);  document.getElementById("demo").innerHTML = d; | setSeconds() |
| طريقة لتعديل الميلي ثانية من 0 إلى 999  const d = new Date();  d.setMilliSeconds(300);  document.getElementById("demo").innerHTML = d; | setMilliseconds() |
| 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; | مثال |

**Boolean**

**27**

|  |  |
| --- | --- |
| اسم الوظيفة | الرمز |
| عندما نعطي شرط يتحقق من الشرط ويخرج true أو false  أما لايتم إعطاء شرط يخرج true إلا إذا كان المحتوة صفر أو فارغ يخرج false  document.getElementById("demo").innerHTML = Boolean(10 > 9);  ////  document.getElementById("demo").innerHTML =  "100 is " + Boolean(100) + "<br>" +  "3.14 is " + Boolean(3.14) + "<br>" +  "-15 is " + Boolean(-15) + "<br>" +  "Any (not empty) string is " + Boolean("Hello") + "<br>" +  "Even the string 'false' is " + Boolean('false') + "<br>" +  "Any expression (except zero) is " + Boolean(1 + 7 + 3.14);  ///  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  ///// | Boolean() |
| let x = false; // x is a boolean  let y = new Boolean(false); // y is an object  document.getElementById("demo").innerHTML = typeof x + "<br>" + typeof y; | typeof |

**Comparison**

**29**

|  |  |
| --- | --- |
| اسم الوظيفة | الرمز |
| 1. == 2. === 3. != 4. !== 5. > 6. < 7. >= 8. <= | Comparison Operators |
| 1. && 2. || 3. ! | Logical Operators |
| *variablename*= (*condition*) ?*value1*:*value2*  *value1* = true // *value2* = false  <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.";  } | Syntax |

**Conditional Statements**

**30**

|  |  |
| --- | --- |
| اسم الوظيفة | الرمز |
| if (*condition*) {   //*block of code to be executed if the condition is true* } | if |
| 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* } | else |
| 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* }  if (time < 10) {   greeting = "Good morning"; } else if (time < 20) {   greeting = "Good day"; } else {   greeting = "Good evening"; }  ملاحظة : يبدأ التحقق من الشرط من الأعلى للأسفل جملة تلوى الأخرى و إذا لم يتحقق إي شرط تتحقق جملة else | else if |
| وهي جملة شرطية بداخل جملة شرطية  if (*condition1*) {   if (*condition1*) {  *//*  *} else {*  *//*  *}* } | nested if |
| (condition) ?  value1 : value2 >> (condition) ?  true : false or else  حيث (*condition*) ? هي if  حيث *value1* : هي true  حيث value2 : هي false أو else يعني شرط جديد يكتب هكذا  (condition) ?  true : (condition2) ? true : false ;  ملاحظة : يمكن تخزين الشرط داخل متغيير وتكون قيمة المتغيير هي قيمة الشرط المحقق  let voteable = (10 < 18) ? "Too young":"Old enough";  لو إحتجنا أن نضيف else if نضيف ? ثم : التي تعبر عن else if وأخر : تعبر عن else  let num = (10 == "10") ? "num10" : (11>10 && 8>4) ? "num11" : "notnum";  لا يوجد فيها nested if أي لا يوجد if داخل if | short if |
| switch (*expression*) {   case *x*: *// code block*     break;   case *y*: *// code block*     break;   default:     // *code block* }  ملاحظة : يتم إدخال المتغيير ضمن switch ثم يتم التحقق من قيمة المتغيير إذا متطابقة 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 بمتغيير مختلف لإظهار نفس النتيجة  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"; } | switch |

**Loop**

**31**

|  |  |
| --- | --- |
| اسم الوظيفة | الرمز |
| for (*statement 1*;*statement 2*;*statement 3*) {   // *code block to be executed* }  for (let i = 0; i < 5; i++) {   text += "The number is " + i + "<br>"; }  عند تعريف المتغيير داخل for على شكل let فإن المتغيير وقيمته ستبقى داخل ال for | for |
| تستخدم لطباعة أو التعديل على مصفوفة أو كائن  for (variable in array) {   code }  ///////  const numbers = [45, 4, 9, 16, 25];  let txt = "";  for (let x in numbers) {  txt += numbers[x] + "<br>";  }  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;  ملاحظة :   1. يتم توقف الحلقة عند نفذان عدد العناصر المصفوفة أو الكائن 2. لا يمكن فرض قيمة لمتغيير الحلقة 3. تستخدم عندما لا يكون الترتيب مهما لعناصر المصفوفة أو الكائن | for in |
| و هي حلقة خاصة بالمصفوفة حيث تستدعي تابع وتقوم بتكراره حسب عدد عناصر المصفوفة مع العلم أن عند إستدعاء التابع يتم الإرسال إليه ثلاث قيم وهي :   1. قيم المصفوفة 2. فهارس المصفوفة 3. أسم المصفوفة   const numbers = [45, 4, 9, 16, 25];  let txt = "";  numbers.forEach(myFunction);  document.getElementById("demo").innerHTML = txt;  function myFunction(value, index, array) {  txt += value + "<br>";  }  //////////// | Array.forEach() |
| تستخدم لطباعة أو التعديل على الكائنات القابلة لتكرار مثل   Arrays, Strings, Maps, NodeLists, and more  const cars = ["BMW", "Volvo", "Mini"];  let text = "";  for (let x of cars) {  text += x + "<br>";  }  document.getElementById("demo").innerHTML = text;  //////  let language = "JavaScript";  let text = "";  for (let x of language) {  text += x + "<br>";  }  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* } | for Of |
| وهي حلقة مثل for ولكن الإختلاف أنه يتم تعريف المتغيير قبل while والشرط بين قوسين while و القيمة المؤثرة على المتغيير داخل while  while (*condition*) { *// code block to be executed* }  let text = "";  let i = 0;  while (i < 10) {  text += "<br>The number is " + i;  i++;  }  document.getElementById("demo").innerHTML = text;  ملاحظة : إذا نسيت زيادة المتغير المستخدم في الشرط ، فلن تنتهي الحلقة أبدًا. سيؤدي هذا إلى تعطل متصفحك | while |
| نفس مبدأ while لكن يتم تنفيذ المحتوى مرة واحدة على الأقل  do { *// code block to be executed* } while (*condition*);  ///////////  let text = ""  let i = 0;  do {  text += "<br>The number is " + i;  i++;  }  while (i < 10);  document.getElementById("demo").innerHTML = text;  ملاحظة : يمكن استخدام for مثل while  const cars = ["BMW", "Volvo", "Saab", "Ford"];  let i = 0;  let text = "";  for (;cars[i];) {  text += cars[i] + "<br>";  i++;  }  document.getElementById("demo").innerHTML = text;  حيث هنا يتم تنفيذ الحلقة لتنفذ عناصر المصفوفة  const cars = ["BMW", "Volvo", "Saab", "Ford"];  let i = 0;  let text = "";  while (cars[i]) {  text += cars[i] + "<br>";  i++;  }  document.getElementById("demo").innerHTML = text; | do while |
| تقوم بإقاف الحلقة نهائيا ع الأغلب يتم وضعها داخل شرط  let text = "";  for (let i = 0; i < 10; i++) {  if (i === 3) { break; }  text += "The number is " + i + "<br>";  }  document.getElementById("demo").innerHTML = text; | break |
| تقوم بإقاف الحلقة لمرة واحدة ع الأغلب يتم وضعها داخل شرط  let text = "";  for (let i = 0; i < 10; i++) {  if (i === 3) { continue; }  text += "The number is " + i + "<br>";  }  document.getElementById("demo").innerHTML = text; | continue |
| وهو عندما يكون لدينا حلقتين داخل بعض و نريد أيقاف وحدة دون الأخرى نستخدم lable و ذلك من خلال تسمية الفور وإقافها عن طريق البرك أو كونتينيو  mainloop : for ()  {  nestedloop for () {  if () {  break mainloop;  }  }  } | lable |

**sets**

**31**

|  |  |
| --- | --- |
| اسم الوظيفة | الرمز |
| لإنشاء مجموعة جديدة بداخلها مصفوفة أو كائن أو نص  // Create a Set const letters = new Set(["a","b","c"]); | new Set() |
| لمعرفة عدد العناصر داخل المصفوفة أو الكائن  // Create a Set  const letters = new Set(["a","b","c"]);  // Display set.size  document.getElementById("demo").innerHTML = letters.size; | size |
| وظيفته إضافة القيم أو المتغيرات للمجموعة  // 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;  ملاحظة : إذا أضفت عناصر متساوية ، فسيتم حفظ العنصر الأول فقط وتجاهل الجديد  // Create a Set  const letters = new Set();  // Add values to the Set  letters.add("a");  letters.add("b");  letters.add("c");  letters.add("c");  letters.add("c");  letters.add("c");  letters.add("c");  letters.add("c");  // Display set.size  document.getElementById("demo").innerHTML = letters.size;  سيكون عدد العناصر فقط 3 | add() |
| تستخدم لتعديل أو طباعة المجموعة  // Create a Set  const letters = new Set(["a","b","c"]);  // List all Elements  let text = "";  letters.forEach (function (value) {  text += value + "<br>";  })  document.getElementById("demo").innerHTML = text; | forEach() |
| يقوم التابع بإرجاع كائن مكرر جديد يحتوي على جميع القيم في مجموعة :  // Create a Set  const letters = new Set(["a","b","c"]);  // List all Elements  let text = "";  for (const x of letters.values()) {  text += x + "<br>";  }  document.getElementById("demo").innerHTML = text; | values() |

**Maps**

**33**

|  |  |
| --- | --- |
| اسم الوظيفة | الرمز |
| يقوم بإنشاء خريطة جديدة والخريطة عبارة عن مصفوفة بداخلها مفتاح و قيمة  // Create a Map  const fruits = new Map([  ["apples", 500],  ["bananas", 300],  ["oranges", 200]  ]); | new Map() |
| وظيفتها إضافة عناصر إلى الخريطة  // 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([  ["apples", 500],  ["bananas", 300],  ["oranges", 200]  ]);  fruits.set("apples", 200); | set() |
| وظيفته إعطاء القيمة عندما تعطيه المفتاح  // Create a Map  const fruits = new Map([  ["apples", 500],  ["bananas", 300],  ["oranges", 200]  ]);  document.getElementById("demo").innerHTML = fruits.get("apples");//return 500 | get() |
| يقوم بإرجاع عدد العناصر في الخريطة  // Create a Map  const fruits = new Map([  ["apples", 500],  ["bananas", 300],  ["oranges", 200]  ]);  document.getElementById("demo").innerHTML = fruits.size; | size |
| يقوم بحذف العنصر عندما تعطيه المفتاح  // 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; | delete() |
| يقوم التابع بالتأكد إذا كان العنصر موجود في الخريطة من خلا المفتاح فإذا كان موجود يرجع true و إذا كان غير موجود يرجع false  // 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 | has() |
| وظيفتها طباعة أو التعديل على المفتاح أو القيمة للعناصر في الخريطة في تعطي الإمكانية لتحكم بالمفتاح أو القيمة  // Create a Map  const fruits = new Map([  ["apples", 500],  ["bananas", 300],  ["oranges", 200]  ]);  let text = "";  fruits.forEach (function(value, key) {  text += key + ' = ' + value + "<br>"  })  document.getElementById("demo").innerHTML = text; | forEach() |
| وظيفته هي عدم إرجاع العناصر التي تمتلك نفس المفتاح والعناصر التي تكرر فيها المفتاح ستكون قيمتها قيمة المفتاح الأخير المكرر  // 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 + "<br>";  }  document.getElementById("demo").innerHTML = text; | entries() |

**errors**

**34**

|  |  |
| --- | --- |
| اسم الوظيفة | الرمز |
| تسمح لك العبارة بتحديد كتلة من التعليمات البرمجية ليتم اختبارها بحثًا عن الأخطاء أثناء تنفيذها .  try {   *Block of code to try* } | try |
| يسمح لك بإنشاء نص أو رقم لخطأ مخصص داخل try يتم إظهاره لمستخدم عن طريق المتغيير داخل catch  throw "Too big";    // throw a text throw 500;          // throw a number | throw |
| يسمح لك بتحديد كتلة من التعليمات البرمجية ليتم تنفيذها ، في حالة حدوث خطأ في كتلة try  و يكون داخله متغيير err يكون مربوط مع throw  try {   *Block of code to try* } catch(*err*) {   *Block of code to handle errors* }  /////////////  <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 "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> | catch |
| يتيح لك تنفيذ التعليمات البرمجية ، بعد إجراء الاختبار سواء كان هناك خطأ أو لا يوجد هناك خطأ .  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 {  document.getElementById("demo").value = "";  }  }  </script> | finally |
| وهي كائنات مخصصة لأخطاء محددة و توفر كائنات الخطأ خاصتين :   1. أسم الخطأ : يقوم بتعيين و إرجاع أسم الخطأ 2. رسالة الخطأ : يقوم بتعيين و أرجاع رسالة الخطأ   ولكن لهذه الكائنات أنواع عديدة :  eval() : يشير إلى خطأ في الوظيفة  <p id="demo"></p>  <script>  try {  eval("alert('Hello)");  }  catch(err) {  document.getElementById("demo").innerHTML = err.name;  }  </script>  ///////////////  RangeError : يشير إلى الخطأ إذا كنت تستخدم رقم خارج النطاق  <p id="demo">  <script>  let num = 1;  try {  num.toPrecision(500);  }  catch(err) {  document.getElementById("demo").innerHTML = err.name;  }  </script>  //////////////////  ReferenceError : يشير إلى إذا كنت تستخدم متغيير لم يتم التصريح عنه  <p id="demo"></p>  <script>  let x = 5;  try {  x = y + 1;  }  catch(err) {  document.getElementById("demo").innerHTML = err.name;  }  </script>  ///////////////  SyntaxError : يستخدم لتقييم الجمل البرمجية إذا بها خطأ برمجي في الصياغة  <p id="demo"></p>  <script>  try {  eval("alert('Hello)");  }  catch(err) {  document.getElementById("demo").innerHTML = err.name;  }  </script>  ////////////////////  TypeError : يشير أن نوع المتغيير مخالف لنوع المتوقع  <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> | The Error Object |

**this Keyword**

**35**

|  |  |
| --- | --- |
| اسم الوظيفة | الرمز |
| و هي كلمة أساسية تشير إلى الكائن الذي تنتمي إليه ومن خلالها يمكننا التحكم بعناصر الكائن نفسو و لها قيم مختلفة حسب مكان استخدامها : | this |
| وهو تابع في الكائن this هنا تشيير إلى عناصر الكائن  // 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(); | this in a Method |
| عند استخدامه في ملف الجافا سكربت الرئيسي فأنه يشير إلى كائن الويندوز [object Window]  let x = this;  document.getElementById("demo").innerHTML = x; | this Alone |
| تكون this تشيير إلى التابع نفسه  document.getElementById("demo").innerHTML = myFunction();  function myFunction() {  return this;  } | this in a Function |
| في أحداث الدوم تشيير إلى العنصر نفسه  <button onclick="this.style.display='none'">Click to Remove Me!</button>  أو  <button id=”demo”>Click to Remove Me!</button>  document.getElementById("demo").onclick = function() {return this;};  // الناتج <button id=”demo”>Click to Remove Me!</button> | this in Event Handlers |
| call() and apply() و هي توابع جاهزة تقوم بربط عناصر كائن بكائن أخر  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; | Explicit Function Binding |

**Class**

**36**

|  |  |
| --- | --- |
| **اسم الوظيفة** | **الرمز** |
| class : هي دالة وليست كائن تستخدم الكلمة class Name {} لإنشائها وبداخلها دوما الدالة constructor() {}  و دوال أخرى يتم إنشائها لتقوم بعمليات حسابية  Syntax  class ClassName {   constructor() { ... } }  ملاحظة 1 :تستخدم الكلاسات لإنشاء كائنات متعددة بقيم مختلفة من كلاس واحد وأسم الكلاس يفضل أن يبدأ بحرف كبير  ملاحظة 2 : يجب عليك تعريف الكلاس قبل استخدامه على عكس الدوال  مثال 1  class Car {  constructor(name, year) {  this.name = name; // مفتاح أول مع قيمتو  this.year = year; // مفتاح ثاني مع قيمتو  }  }  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  ملاحظات عن دالة المشئ constructor() { ... } :   1. يجب تسميتها بهذا الأسم دوما constructor 2. يجب إنشائها دوما داخل الكلاس وإذا لم يتم إنشائها ستقوم الجافا سكربت تلقائيا بإنشاء واحدة فارغة 3. عند إنشاء الكائن نستدعي أسم الكلاس كدالة وهو بدوره يستدعي دالة المنشئ constructor() {} 4. نسنخدم دالة المنشئ constructor() {} لإنشاء مفاتيح الكائن و إعطائها القيم لهذه المفاتيح   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 معقده وإرجاع القيم  مثال 2  class Car {  constructor(name, year) {  this.name = name; // مفتاح أول مع قيمتو  this.year = year; // مفتاح ثاني مع قيمتو  }  age() { // مفتاح ثالث مع قيمتو  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.  ملاحظة : يمكننا الوصول إلى قيم المفاتيح الإخرى من خلال this.kay = value بكتابة سيز مع اسم المفتاح نحصل على قيمة المفتاح  مثال 3 يمكننا تمرير القيم لدوالة المنشأة  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. | class |
| يمكننا توريث كلاس لكلاس إخرى باستخدام extends حيث ترث المفاتيح و القيم الكلاس الأخر  ثم نتسخدم super(value) داخل constructor() {} الكلاس الأصلية لإرسال القيمة إلى الكلاس الذي نرث منه  مثال 1  class Car {  constructor(brand,brand2) {  this.carname = brand; // مفتاح أول مع قيمتو  this.carname2 = brand2; // مفتاح ثاني مع قيمتو  }  present() {// مفتاح ثالث مع قيمتو  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 المفتاحين الأول و الثاني  this.model = mod; // مفتاح رابع مع قيمتو  }  show() {// مفتاح الخامس مع قيمتو  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  ملاحظة : super(value) في الكلاس الأصلي يمرر القيم إلى constructor() {} الكلاس المورث  ويأخذ super(value) قيمه من constructor(value) { super(value);} الدالة المنشأة في الكلاس الأصلي | Class Inheritance |
| get kay() : تستخدم للوصول إلى قيمة مفتاح و الاستدعاء يكون هكذا objectName.kay  set kay(x) : تستخدم لتعديل قمية مفتاح و الاستدعاء يكون هكذا objectName.kay = x ;  مثال 1  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  مثال 2 نعدل قيم المفتاح  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 أولا باستخدام سيت  document.getElementById("demo").innerHTML = myCar.cnam;>> AMMAR  // طبعنا القيمة عن طريق غيت | Getters and Setters |
| 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 1  class Car {  constructor(name) {  this.name = name;// value = Ford  }  static hello() {  return "Hello" +" / " + this.name;  }  }  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 2  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 | Static Methods |

**عند إنشاء ال dom شاهد فيديو 77**

**Dom**

**37**

|  |  |
| --- | --- |
| **اسم الوظيفة** | **الرمز** |
| Dom : Document Object Model  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    فهو يحدد معيارًا للوصول إلى المستندات    الدوم هو المعيار الذي ينقسم إلى 3 أجزاء مختلفة :   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   الدوم في الجافا سكريبت تزودك بكل القوة لإنشاء صفحات html ديناميكية مثل :   1. يمكنك تغيير جميع العناصر في الصفحة 2. يمكنك تغيير جميع سمات العناصر في الصفحة 3. يمكنك تغيير جميع أنماط css في الصفحة 4. يمكنك إزالة عناصر و السمات الموجودة في الصفحة 5. يمكنك إضافة عناصر و السمات إلى الصفحة 6. يمكنك إنشاء أحداث للعناصر في الصفحة 7. يمكنك التفاعل مع الأحداث العناصر في الصفحة | intro with dom |
| Document: هو كائن يمثل صفحة الويب لذلك إذا أردت الوصول إلى أي عنصر داخل صفحة Html عليك البدء به  مثل :  document.getElementById(*id*)  document.domain | DOM Document |
| لتعديل على عناصر صفحة Html يجب أولا الوصول إلى العناصر ويتم ذلك من خلال عدت طرق منها :   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   1- الوصول للعنصر عن طريق id وهو الأسرع ولا يتكرر بالصفحة مرتين مثل  document.getElementById("idName”)  2- الوصول للعنصر عن طريق إسمه 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");  3- الوصول للعنصر عن طريق أسم 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>  4- الوصول للعنصر عن طريق تسميات css مثل أسم الكلاس لوحده أو أسم الأيدي أو أسم العنصر مع الكلاس أو أسم 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.  5- الوصول إلى عنصر الفورم وهي خاصة ب window أو بالأحرة خاصة بال BOM حيث يتم استدعاء document وبعدها النوع الذي نريد مثل form أو titel أو body  <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;  }  document.getElementById("demo").innerHTML = text + text1;  </script> >> Ammar / Qassab / Submit / 3  ملاحظة : هناك عدة طرق أخرى للوصول لمحتويات المختلفة لصفحة html من خلال BOM أو كائن window :   1. [document.anchors](https://www.w3schools.com/js/tryit.asp?filename=tryjs_doc_anchors) 2. [document.body](https://www.w3schools.com/js/tryit.asp?filename=tryjs_doc_body) 3. [document.documentElement](https://www.w3schools.com/js/tryit.asp?filename=tryjs_doc_element) 4. [document.embeds](https://www.w3schools.com/js/tryit.asp?filename=tryjs_doc_embeds) 5. [document.forms](https://www.w3schools.com/js/tryit.asp?filename=tryjs_doc_forms) 6. [document.head](https://www.w3schools.com/js/tryit.asp?filename=tryjs_doc_head) 7. [document.images](https://www.w3schools.com/js/tryit.asp?filename=tryjs_doc_images) 8. [document.links](https://www.w3schools.com/js/tryit.asp?filename=tryjs_doc_links) 9. [document.scripts](https://www.w3schools.com/js/tryit.asp?filename=tryjs_doc_scripts) 10. [document.title](https://www.w3schools.com/js/tryit.asp?filename=tryjs_doc_title) | DOM Elements |
| للوصول لسمة عناصر html نستخدم getAttribute(“NameAttribute”)  مثال  <div id=”divid” class=”cont” href=”#”>ammar</div>  <script>  let e = document.getElementById(“divid”);  let c = e.getAttribute(“class”);//cont  let l = e.getAttribute(“href”);//#  </script>  لتغيير أو إضافة سمة لعناصر html نستخدم setAttribute(“NameAttribute”,”valueAttribute”)  مثال  <div id=”divid” class=”cont” href=”#”>ammar</div>  <script>  let e = document.getElementById(“divid”);  e.setAttribute(“class”,”color”);  </script>// cont color  attributes : هي خاصة للوصول لأسماء السمات العنصر جميعها  <div id=”divid” class=”cont”>ammar</div>  <script>  let mye = document.getElementById(“divid”).attributes;  console.log(mye);// id=”divid” class=”cont”  </script>)// id=”divid” class=”cont”  hasAttribute(“NameAttribute”) : وهو يعمل تشيك للعنصر هل أسم السمة موجودة في العنصر أو لا فهو يرجع  true / false  مثال  <div id=”divid” class=”cont”>ammar</div>  <script>  let mye = document.getElementById(“divid”).attributes;  if (mye.hasAttribute(“class”)) {  console.log(“found”);  } else {  console.log(“not found”);  }  </script>//found  removAttribute(“NameAttribute”) : وهو يعمل حذف أسم قيمة من العنصر  مثال 1  <div id=”divid” class=”cont”>ammar</div>  <script>  let mye = document.getElementById(“divid”).attributes;  if (mye.hasAttribute(“class”)) {  mya.removAttribute(“class”);  console.log(mye);  } else {  console.log(“not found”);  }  </script>// id=”divid”  مثال 2  <div id=”divid” class=”cont”>ammar</div>  <script>  let mye = document.getElementById(“divid”).attributes;  if (mye.getAttribute(“class”) === “”) {  mya.removAttribute(“class”);  } else {  mye.setAttribute(“class”,”big”);  console.log(mye);  }  </script>//لو القيمة فارغة أحذفها أما أذا كانت موجودة عدل عليها | attributes |
| أسهل طريقة لتغيير محتوى عنصر Html هو استخدام innerHTML مثل  document.getElementById(*id*).innerHTML =new HTML ;  مثال  <p id="p1">Hello World!</p>  <script>  document.getElementById("p1").innerHTML = "New text!"; >> New text!  </script>  للوصول إلى المحتوى النصي الذي يقرأه المتصفح نستخدم textcontant  <p id="p1">Hello World!<span>ammar<span></p>  <script>  let a = document.getElementById("p1").textcontent;  consol.log(a);  </script>// Hello World! <span>ammar<span>  أسهل طريقة لتغيير محتوى النصي للعنصر Html هو استخدام textcontant مثل  document.getElementById(*id*). textcontant =“new HTML” ;  مثال  <p id="p1">Hello World!</p>  <script>  document.getElementById("p1").innerHTML = "New text! <span>ammar<span>";  >> New text! New text! <span>ammar<span>  يتم ارجاع عنصر html كنص وليس كعنصر وهكذا يتم طباعة الأكواد البرمجيه في الصفحة  </script>  لتغيير قيم صفات العناصر Html من خلال كتابة الخاصية نفسها مثل  document.getElementById(*id*).*attribute =* new value  مثال لتغير عنوان الصورة  <img id="image" src="smiley.gif" width="160" height="120">  <script>  document.getElementById("image").src = "landscape.jpg";  </script>  تغيير المحتوى بطريقة ديناميكية من خلال الدالة setInterval( () => {} , 1000 );  function tick() {  document.getElementById("demo").innerHTML = "Date : " + Date();  }  tick();  setInterval(tick, 1000); | DOM Changing |
| التحقق من عناصر الإدخال قبل الإرسال نستخدم onsubmit="”  مثال  <!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>  </body>  </html>  أو فينا نستعمل قيم العناصر ال Html مثل disabled | required | max | min | type | pattern  مثال  <form action="/action\_page.php" method="post">  <input type="text" name="fname" required>  <input type="submit" value="Submit">  </form>  وهناك أحداث أخرى تستخدم في form مثل onfoucs / onblur  ملاحظة : لمنع وقوع الحدث نضيف .preventDefault()  مثال  <a id=”link1” href=”<https://www.google.com>”>goolge</a>  <script>  document.getElementById(“link1”).onclick = function (e) {  e.preventDefaulte();  }  </script> مهما كبسنا على اللينك لن يفتح  تطبيق عن validate form  <form action=””>  <input type=”text” name=”username” placeholder=”max 10 chars only” />  <input type=”text” name=”age” placeholder=”Can’t Be Empty” />  <input type=”submit” value=”submit Data” />  </form>  <script>  let userInput = document.querySelector(“[name=’username’]”);  let ageInput = document.querySelector(“[name=’age’]”);  document.form[0].onsubmit = function (e) {  let userValid = false;  let ageValid = false;  if (userInput.value !== “” && userInput.value.length <= 10) {  userValid = true ;  }  if (ageInput.value !== “” ) {  ageValid = true ;  }  if (userValid === false || ageValid === false) {  e.preventDefault();  }  }  </script>  مثال عن onfocus / onblur  <form action=””>  <input type=”text” name=”username” placeholder=”max 10 chars only” />  <input type=”text” name=”age” placeholder=”Can’t Be Empty” />  <input type=”submit” value=”submit Data” />  </form>  <a id=”link1” href=”<https://www.google.com>”>goolge</a>  <script>  let userInput = document.querySelector(“[name=’username’]”);  let ageInput = document.querySelector(“[name=’age’]”);  window.onload = () => { userInput.focus();};//أول ما تحمل الصفحة يدخل إلى الحقل  ageInput.onblur() = function() {  document.getElementById(“link1”).click();  }; عندما يخرج المستخدم من الحقل الثاني يدخل على الينك  }  </script> | Forms |
| لتغيير الستايل لعنصر ما نتبع التالي  document.getElementById(id).style.property = new style;  مثال  <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>  لإضافة أكثر من ستايل في عنصر واحد  document.getElementById(id).style.cssText = “color: red; font-weight: blod;”;  يمكننا استعمال الأحداث أيضا  <button type="button" onclick="document.getElementById('id1').style.color = 'red'">  Click Me!  </button>  لحذف ستايل من عنصر موجود في html نكتب  document.getElementById(id).style.removeProperty(“color”);  لإضافة ستايل إلى عنصر موجود في html نكتب  document.getElementById(id).style.setProperty(“color”, ”red”, “important”);  لحذف ستايل من ملف css نكتب  document.stylesheets[0].rules[0].style.removeProperty(“color”);  حيث stylesheets[0] ملف css الأول في html و rules[0] التنسيق الأول في ملف css ويمكن كتابة أسمه  لإضافة ستايل إلى ملف css نكتب  document.stylesheets[0].rules[0].style.setProperty(“color”,”red”, “important”); | css |
| يمكننا فعل النيميشن عن طريق javascript من دون css  مثال  <!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);  } else {  pos++;  elem.style.top = pos + "px";  elem.style.left = pos + "px";  }  }  }  </script>  </body>  </html> | Animation |
| عند وقوع حدث من قبل المستخدم يمكن لجافاسكربت إجراء تغيير على العناصر من خلال :  document.getElementById(id).onevent = javascript  من الأمثلة التي يمكن أن تفعلها الأحداث في الجافاسكريبت :   1. عندما تم تحميل الصورة 2. عندما يتحرك الماوس فوق عنصر 3. عندما يتم تغيير حقل الإدخال 4. عندما يتم تقديم نموذج HTML 5. عندما يضغط المستخدم على مفتاح   يمكننا تفعيل الحدث بطريقتين :   1. أما أحداث عناصر Html مع دالة تفعل عند تفعيل الحدث 2. أو أحداث من داخل الجافاسكريبت   onclick : و حدث يفعل عند النقر على العنصر  مثال 1  <button onclick="displayDate()">The time is?</button>  <p id="demo"></p>  <script>  function displayDate() {  document.getElementById("demo").innerHTML = Date();  }  </script>  مثال 2  <button id="myBtn">Try it</button>  <p id="demo"></p>  <script>  document.getElementById("myBtn").onclick = displayDate;  function displayDate() {  document.getElementById("demo").innerHTML = Date();  }  </script>  onload : هو حدث يفعل عند تحميل كامل العنصر مع محتوياته يوضع في body يستخدم لإقاف شاشة التحميل  onunload : هو حدث يفعل عند لا يتم تحميل كامل العنصر مع محتوياته  مثال  <!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.";  } 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  <h2>JavaScript HTML Events</h2>  Enter your name: <input type="text" id="fname" onchange="upperCase()">  <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 مفعل 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>  ملاحظة : لمنع وقوع الحدث نضيف .preventDefault()  مثال  <a id=”link1” href=”<https://www.google.com>”>goolge</a>  <script>  document.getElementById(“link1”).onclick = function (e) {  e.preventDefaulte();  }  </script> مهما كبسنا على اللينك لن يفتح | Events |
| تستخدم لإضافة الأحداث إلى العناصر من داخل الجافاسكريبت ويمكنك إضافة عدد الأحداث التي تريد حيث يمكنك إحداث من نفس النوع لكن تنفذ دوال مختلفة ويمكنك إضافة حدث لعنصر لم يتم إنشائه بعد في الصفحة  Syntax  *element*.addEventListener(*event, function, useCapture*);  event : هي نوع الحدث أو اسم الحدث بدون on مثل click  function : وهي دالة تفعل عند وقوع الحدث وتفضل أن تكون من نوع أنينموس أو دالة خارجية  useCapture : و هي قيمة منطقية تحدد ما إذا كان سيتم استخدام الحدث وتكون أما true أو false وهي إختيارية  مثال 1  <button id="myBtn">Try it</button>  <p id="demo"></p>  <script>  document.getElementById("myBtn").addEventListener("click", displayDate);  function displayDate() {  document.getElementById("demo").innerHTML = Date();  }  </script>عند الضغط يفعل الحدث ومنه تفعل الدالة ويظهر الوقت  مثال 2  <button id="myBtn">Try it</button>  <script>  document.getElementById("myBtn").addEventListener("click", function() {  alert("Hello World!");  });  </script>  يمكننا إضافة العديد من أحداث إلى عنصر واحد ويمكن أن تكون من نفس الحدث لكن ينفذ دوال مختلفة  مثال  <button id="myBtn">Try it</button>  <p id="demo"></p>  <script>  var x = document.getElementById("myBtn");  x.addEventListener("mouseover", myFunction);  x.addEventListener("click", mySecondFunction);  x.addEventListener("click", mySecondFunction2);  x.addEventListener("mouseout", myThirdFunction);  function myFunction() {  document.getElementById("demo").innerHTML += "Moused over!<br>";  }  function mySecondFunction() {  document.getElementById("demo").innerHTML += "Clicked!<br>";  }  function mySecondFunction2() {  document.getElementById("demo").innerHTML += "Clicked2!<br>";  }  function myThirdFunction() {  document.getElementById("demo").innerHTML += "Moused out!<br>";  }  </script>حيث عند الهوفر تظهر جملة وعند عدم الهوفرة تظهر جملة وعند الضغط تظهر جملة  كما يمكننا إضافة الأحداث إلى كائن النافذة window حيث يضاف الحدث إلى الصفحة بأكملها  مثال  <p id="demo"></p>  <script>  window.addEventListener("load", function(){  document.getElementById("demo").innerHTML = "load page ok";  });  </script>حيث يفعل الحدث عند تحميل الصفحة  تمرير القيم إلى دالة خارجية و هنا يجب استخدام دالة انينموس وبداخلها نستدعي الدالة مع تمرير القيم لها  <button id="myBtn">Try it</button>  <p id="demo"></p>  <script>  let p1 = 5;  let p2 = 7;  document.getElementById("myBtn").addEventListener("click", function() {  myFunction(p1, p2);  });  function myFunction(a, b) {  document.getElementById("demo").innerHTML = a \* b;  }  </script> | Event Listener |
| تسمح بإزالة الحدث المكتوب داخل عنصر في html  مثال  <div id="myDIV">  <button onclick="removeHandler()" id="myBtn">Remove</button>  </div>  <p id="demo"></p>  <script>  document.getElementById("myDIV").addEventListener("mousemove", myFunction);  function myFunction() {  document.getElementById("demo").innerHTML = Math.random();  }  function removeHandler() {  document.getElementById("myDIV").removeEventListener("mousemove", myFunction);  }  </script>الحدث يبقى يتفعل حتى يتم الضغط على العنصر عندها يتم حذف الحدث من العنصر | remove Event Listener |
| التنقل بين القعد  نعلم أن document عبارة عن شجرة من العقد nodes حيث كل عنصر داخل html عبارة عن عقدة     1. المستند بأكمله عبارة عن عقدة مستند 2. كل عنصر html عو عقدة 3. النص داخل عنصر html عبارة عن عقدة عنصر 4. كل سمة داخل العنصر هي عقدة سمة 5. كل تعليق هو عقدة تعليق   من خلال جافا سكريبت يمكن الوصول لكل عقدة و يمكن إنشاء عقد جديدة ، ويمكن تعديل جميع العقد أو حذفها  لنأخذ المثال التالي  <html>    <head>     <title>DOM Tutorial</title>   </head>    <body>     <h1>DOM Lesson one</h1>     <p>Hello world!</p>   </body>  </html>    العقد في شجرة العقد لها علاقة هرمية مع بعضها البعض , تُستخدم مصطلحات الوالد والطفل والأخوة لوصف العلاقات   1. في شجرة العقدة ، تسمى العقدة العلوية الجذر (root) 2. كل عقدة لها والد واحد ، باستثناء الجذر (root) ليس له والد 3. يمكن أن تحتوي العقدة على عدد من الأطفال 4. الأشقاء (الأخوة أو الأخوات) هم عقد مع نفس الوالد 5. مثل الجذر root له ولدان head / body   يمكن قراءة المثال السابق :   1. <html> : هي عقدة الجذر root 2. <html> : ليس له أبوين (parent) 3. <html> : هو والد ل head / body 4. <head> : هو الطفل الأول (first child) من <html> 5. <body> : هو الطفل الأخير (last child) من <html> 6. <head> : لديه طفل واحد وهو <title> 7. <title> : لديه طفل واحد و هو (عقدة نصية) 8. <body> : لديه طفلان <h1> / <p> 9. <h1> : لديه طفل واحد و هو (عقدة نصية) 10. <p> : لديه طفل واحد و هو (عقدة نصية) 11. <h1> / <p> هم أشقاء   التنقل بين العقد Navigating Between Nodes :  يمكنك استخدام خصائص العقدة التالية للتنقل بين العقد باستخدام الجافا سكريبت   1. parentNode 2. childNodes[nodenumber] 3. firstChild 4. lastChild 5. nextSibling 6. previousSibling   للوصول إلى عقدة الجذر root نستخدم document.documentElement وهي تضمن الوثيقة كاملة  مثال  <!DOCTYPE html>  <html>  <body>  <h2>JavaScript HTMLDOM</h2>  <p>Displaying document.documentElement</p>  <p id="demo"></p>  <script>  document.getElementById("demo").innerHTML = document.documentElement.innerHTML;  </script>  </body>  </html>  >>  JavaScript HTMLDOM  Displaying document.documentElement  JavaScript HTMLDOM  Displaying document.documentElement  للوصول إلى عقدة الجسم body نستخدم document.body وهي تضمن كامل الجسم  مثال  <!DOCTYPE html>  <html>  <body>  <h2>JavaScript HTMLDOM</h2>  <p>Displaying document.body</p>  <p id="demo"></p>  <script>  document.getElementById("demo").innerHTML = document.body.innerHTML;  </script>  </body>  </html>  >>  JavaScript HTMLDOM  Displaying document.body  JavaScript HTMLDOM  Displaying document.body  الوصول للإطفال ضمن الجسم body أو أي عنصر داخل body فقط :   1. للوصول إلى أسماء العناصر الأطفال ضمن العقدة نستخدم .children 2. للوصول إلى أسم عنصر الأبن مع محتواه ضمن العقدة نستخدم .children[0] 3. للوصول إلى أسم عنصر الأبن الأول مع محتواه ضمن العقدة نستخدم .firstElementChild 4. للوصول إلى أسم عنصر الأبن الأخير مع محتواه ضمن العقدة نستخدم .lastElementChild 5. الوصول إلى الطفل الأول داخل العقدة سواء نص أو عنصر أو الكومنت أو أي شئ نستخدم .firstChild 6. الوصول إلى الطفل الأخير داخل العقدة سواء نص أو عنصر أو الكومنت أو أي شئ نستخدم .lastChild 7. للوصول إلى أي طفل عقدة داخل العقدة سواء نص أو عنصر أو الكومنت أو أي شئ نستخدم .childNodes[0] 8. للوصول إلى قيمة العقدة .nodeValue   مثال  <!DOCTYPE html>  <html>  <body>  <h1 id="id01">My First Page</h1>  <p id="id02"></p>  <p id="id03"></p>  <p id="id04"></p>  <script>  document.getElementById("id02").innerHTML = document.getElementById("id01").innerHTML;  document.getElementById("id03").innerHTML = document.getElementById("id01").firstChild.nodeValue;  document.getElementById("id04").innerHTML = document.getElementById("id01").childNodes[0].nodeValue;  </script>  </body>  </html>جميع الطرق السابقة متكافئة  nodeName : تحدد الخاصية اسم العقدة   1. nodeName : هي للقراءة فقط 2. nodeName : لعقدة عنصر هو نفس أم علامة tag 3. nodeName : لعقدة سمة هو نفس أسم السمة attribute 4. nodeName : لعقدة نص هو نفس النص text 5. nodeName : لعقدة الوثيقة document هو نفس الوثيق document   مثال  <!DOCTYPE html>  <html>  <body>  <h1 id="id01">My First Page</h1>  <p id="id02"></p>  <script>  document.getElementById("id02").innerHTML = document.getElementById("id01").nodeName;  </script>  </body>  </html>>>H1  nodeValue : تحدد الخاصية قيمة العقدة   1. nodeValue : لعقدة عنصر ترجع null 2. nodeValue : لعقدة نص text ترجع النص text 3. nodeValue : لعقدة سمة attribute ترجع قيمة السمة attribute | Navigating Between Nodes |
| إنشاء و تعديل العقد  لإنشاء عنصر html جديد نستخدم .createElement("nameTag")  خيث إذا اردنا وضعه في body نكتب document.createElement("nameTag")  إذا أردنا وضعه ضمن عنصر document.getElementById(“”).createElement("nameTag") و هكذا  حيث يتم إضافة العنصر الجديد أخر عنصر بين الأبناء  لإنشاء محتوى للعنصر نستخدم document.createTextNode("This is new.")  لوضع المحتوى داخل عنصر نستخدم element.appendChild(“”)  مثال على ما سبق  <div id="div1">  <p id="p1">This is a paragraph.</p>  <p id="p2">This is another paragraph.</p>  </div>  <script>  const para = document.createElement("p");  const node = document.createTextNode("This is new.");  para.appendChild(node);  const element = document.getElementById("div1");  element.appendChild(para);  </script>  >>  This is a paragraph.  This is another paragraph.  This is new.  أذا أردت إضافته أول عنصر استخدم element.insertBefore(elementNew , firstchild)  مثال  <!DOCTYPE html>  <html>  <body>  <div id="div1">  <p id="p1">This is a paragraph.</p>  <p id="p2">This is another paragraph.</p>  </div>  <script>  const para = document.createElement("p");  const node = document.createTextNode("This is new.");  para.appendChild(node);  const element = document.getElementById("div1");  const child = document.getElementById("p1");  element.insertBefore(para,child);  </script>  </body>  </html>  >>  This is new.  This is a paragraph.  This is another paragraph.  بحذف أي عنصر نستخدم element.remov()  مثال  <div>  <p id="p1">This is a paragraph.</p>  <p id="p2">This is another paragraph.</p>  </div>  <button onclick="myFunction()">Remove Element</button>  <script>  function myFunction() {  document.getElementById("p1").remove();  }  </script>  وهي طريقة لا تعمل على المتصفحات القديمة لذلك يمكنك أن تستخدم parent.removeChild(child) حيث يوضع الأب قبلها و الأبن داخلها  مثال  <div id="div1">  <p id="p1">This is a paragraph.</p>  <p id="p2">This is another paragraph.</p>  </div>  <script>  const parent = document.getElementById("div1");  const child = document.getElementById("p1");  parent.removeChild(child);  </script>  >> This is another paragraph.  لإستبدال عنصر بعنصر استخدم replaceChild(newelement ,oldelement)  مثال  <div id="div1">  <p id="p1">This is a paragraph.</p>  <p id="p2">This is a paragraph.</p>  </div>  <script>  const parent = document.getElementById("div1");  const child = document.getElementById("p1");  const para = document.createElement("p");  const node = document.createTextNode("This is new.");  para.appendChild(node);  parent.replaceChild(para,child);  </script>  >>  This is new.  This is a paragraph.  لإضافة سمة لعقدة العنصر بدون قيمة نستخدم e = document.createAttribute(“nameAttribute”)  ثم ندخلها إلى عنصر العقدة عن طريق element.setAttibuteNode(e)  مثال  let myElement = document.createElement(“div”);  let myAttribut = document.createAttribute(“class”);  myElement.setAttributeNode(myAttribut);  لكن الطريقة الأفضل الطريقة التي تعلمناها في السابق  let myElement = document.createAttribut(“nameAttribute”);  myElement.setAttribute(“class” ,”cont”);  لإضافة كومينت للعنصر نستخدم document.createComment(“this is comment”)  مثال  let myElement = document.createElement(“div”);  let myComment = document.createComment(“this is div”);  myElement.appendChild(myComment);  لنسخ عقدة بكامل العناصر التي داخلها نستخدم cloneNode(true)  لنسخ عقدة فقط بدون العناصر التي بداخل نستخدم cloneNode(false)  <div id=”div1” class=”cont”>  <span id=”span1”></span>  <!--comment -->  <span id=”span2”></span>  <!--comment -->  <span id=”span3”></span>  <!--comment -->  </div>  let element = document.getElementById(“div1”).cloneNode(false);  ولكن الأن أصبح لديهم نفس id لذلك نغييره من خلال  element.id = `${element.id}-clone` | Adding and Removing Nodes |
| تنقل بين عناصر الداخلية للعنصر الواحد  <div id=”div1” class=”cont”>  <span id=”span1”></span>  <!--comment -->  <span id=”span2”></span>  <!--comment -->  <span id=”span3”></span>  <!--comment -->  </div>  nextSibling : للوصول للعقدة يلي بعدوه سواء نص أو كومينت أو عنصر  let element = document.getElementById(“span2”);  element.nextSibling;// <!--comment -->  nextElementSibling : للوصول للعنصر يلي بعدوه  let element = document.getElementById(“span2”);  element.nextElementSibling;// <span id=”span3”></span>  previousSibling : للوصول للعقدة يلي قبله سواء نص أو كومينت أو عنصر  let element = document.getElementById(“span2”);  element.previousSibling;// <!--comment -->  previousElementSibling : للوصول للعنصر يلي قبله  let element = document.getElementById(“span2”);  element.previousElementSibling;// <span id=”span1”></span>  parentElement : للوصول لعنصر الأب للعنصر المحدد  let element = document.getElementById(“span2”);  element.parentElement;//<div class=”cont”></div> | Dom traversing |
| classList : وهي عبارة عن كائن يجلب جميع الكلاسات الموجودة في العنصر  contains(“one”) : وهي عبارة عن دالة تقوم بتشيك على قيم الكائن و ترجع true / false  item(”3”) : وهي عبارة عن دالة نعطيها المفتاح الكائن وتجلب قيمة المفتاح  مثال  <div id=”div1” class=”one two three”>ammar</div>  <script>  let element = document.getElementById(“div1”);  console.log(element.classList);// 0 =”one” 1=”two” 2=”three”  console.log(element.classList.contains(“ammar”));// false  console.log(element.classList.item(“2”));// two  </script>  add(“value”) : وهي دالة تضيف للكائن القيم حيث تضيف مفتاح مع قيمة في نهاية الكائن  remove(“value”) : وهي دالة تحذف القيم من الكائن مع المفتاح وذلك من خلال تحديد القيمة المفتاح في الكائن  toggle(“value”) :وهي دالة تحذف القيمة مع المفتاح من الكائن أذا كانت موجودة أو تضيف القيمة مع المفتاح إلى الكائن أذا كانت غير موجودة  مثال  <div id=”div1” class=”one two three”>ammar</div>  <script>  let element = document.getElementById(“div1”);  element.onclick = function () {  console.log(element.classList.add(“ammer”, “qassab”));// one two three ammar qassab  console.log(element.classList.remove(“ammar” ,”qassab”));// one two three  console.log(element.classList.toggle(“ammar” ,”qassab”));// one two three ammar qassab  }  </script> | Class list |
| before() : تستخدم لأضافة عنصر أو نص قبل عنصر الذي نحدده  let element = document.getElementById(“div1”);  let myElement = document.createElement(“p”);  element.before(‘hello’);  element.before(myElement);  after() : تستخدم لأضافة عنصر أو نص بعد عنصر الذي نحدده  let element = document.getElementById(“div1”);  let myElement = document.createElement(“p”);  element.after(‘hello’);  element.after(myElement);  append() : تستخدم لأضافة عنصر أو نص داخل عنصر الذي نحدده ولكن في نهاية العناصر التي في الداخل  let element = document.getElementById(“div1”);  let myElement = document.createElement(“p”);  element.append(‘hello’);  element.append(myElement);  prepend() : تستخدم لأضافة عنصر أو نص داخل عنصر الذي نحدده ولكن في بداية العناصر التي في الداخل  let element = document.getElementById(“div1”);  let myElement = document.createElement(“p”);  element.prepend(‘hello’);  element.prepend(myElement);  remove() : لحذف العنصر نهائيا من ال Dom  let element = document.getElementById(“div1”);  element.remove(); | before / after / append / prepend / remove |
| قائمة بالأحداث المهمة onevent :   1. Onabort : يقع الحدث عندما يتم إلغاء تحميل الوسائط (يوضع في video ) 2. onafterprint : يقع الحدث عند بدء طباعة الصفحة أو إذا تم إغلاق مربع حوار الطباعة (يوضع في body) 3. animationend : يقع الحدث عند اكتمال رسم متحرك الأنيمشن لـ css 4. animationiteration 5. animationstart 6. beforeprint 7. beforeunload 8. blur 9. canplay 10. canplaythrough 11. change 12. click 13. contextmenu 14. copy 15. cut 16. dblclick 17. drag 18. dragend 19. dragenter 20. dragleave 21. dragover 22. dragstart 23. drop 24. durationchange 25. ended 26. error 27. focus 28. focusin 29. focusout 30. fullscreenchange 31. fullscreenerror 32. hashchange 33. input 34. invalid 35. keydown 36. keypress 37. keyup 38. load 39. loadeddata 40. loadedmetadata 41. loadstart 42. message 43. mousedown 44. mouseenter 45. mouseleave 46. mousemove 47. mouseover 48. mouseout 49. mouseup 50. offline 51. online 52. open 53. pagehide 54. pageshow 55. paste 56. pause 57. play 58. playing 59. progress 60. ratechange 61. resize 62. reset 63. scroll 64. search 65. seeked 66. seeking 67. select 68. show 69. stalled 70. submit 71. suspend 72. timeupdate 73. toggle 74. touchcancel 75. touchend 76. touchmove 77. touchstart 78. transitionend 79. unload 80. volumechange 81. waiting 82. wheel |  |

**Bom**

**38**

|  |  |
| --- | --- |
| **اسم الوظيفة** | **الرمز** |
|  |  |
|  |  |